

Kavishnayan Nadarajah, Kevin Nikolai Mathisen,
Carl Petter Mørch-Reiersen, Martin Solevåg
Glærum

Digitalizing Environmental Research: Building and Securing a Digital Solution for Environmental Scientists at NINA

Bachelor's thesis in Digital Infrastructure and Cyber Security
Supervisor: Ernst Gunnar Gran, Erik Hjelmås
May 2024



Kavishnayan Nadarajah, Kevin Nikolai Mathisen, Carl
Petter Mørch-Reiersen, Martin Solevåg Glærum

Digitalizing Environmental Research: Building and Securing a Digital Solution for Environmental Scientists at NINA

Bachelor's thesis in Digital Infrastructure and Cyber Security
Supervisor: Ernst Gunnar Gran, Erik Hjelmås
May 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology



Digitalizing Environmental Research: Building and Securing a Digital Solution for Environmental Scientists at NINA

Kavishnayan Nadarajah
Kevin Nikolai Mathisen
Carl Petter Mørch-Reiersen
Martin Solevåg Glærum

2024

Abstract

Title: Digitalizing Environmental Research: Building and Securing a Digital Solution for Environmental Scientists at NINA
Date: 21.05.2024
Participants: Martin Solevåg Glærum, Kevin Nikolai Mathisen, Carl Petter Mørch-Reiersen, Kavishnayan Nadarajah
Supervisors: Erik Hjelmås, Ernst Gunnar Gran
Employer: Norsk institutt for naturforskning
Keywords: Webapplikasjon, Svelte, Secure Software Development
Number of pages: 122
Number of appendix: 16
Availability: Open

Abstract: Developing digital solutions for existing, inefficient systems is an important role of modern software development, with cybersecurity as a critical consideration amid an evolving digital threat landscape. This thesis presents the development of a custom digital solution for NINA, aimed at digitalizing how they interact with and visualize their environmental data of fish demographics in Norwegian rivers. The solution includes a front-end web application built with Svelte, which offers a user-friendly platform for scientist to view and manage their environmental data. Additionally, the solution consists of a back-end architecture responsible for serving the web application, serving environmental data via RESTful API endpoints and handling authentication, leveraging Nginx, PostgreSQL, PostgREST, and Django. Security was considered throughout all parts of the development process, consisting of identifying and managing risks by following relevant methodologies and practices. The result is a secure and user-friendly web application which greatly boosts the productivity of NINA's scientists in their environmental research.

Sammendrag

Tittel:	Digitalizing Environmental Research: Building and Securing a Digital Solution for Environmental Scientists at NINA
Dato:	21.05.2024
Deltakere:	Martin Solevåg Glærum, Kevin Nikolai Mathisen, Carl Petter Mørch-Reiersen, Kavishnayan Nadarajah
Veiledere:	Erik Hjelmås, Ernst Gunnar Gran
Oppdragsgiver:	Norsk institutt for naturforskning
Nøkkelord:	Web Application, Svelte, Sikker programvareutvikling
Antall sider:	122
Antall vedlegg:	16
Tilgjengelighet:	Åpen

Sammendrag: Utviklingen av digitale løsninger for eksisterende systemer er en viktig rolle i moderne systemutvikling, der cybersikkerhet blir mer og mer sentralt i et stadig utviklende digitalt trussellandskap. Denne oppgaven presenterer utviklingen av en digital løsning for NINA, med mål om å digitalisere hvordan de håndterer og visualiserer deres miljødata av fiskedemografi i Norske elver. Løsningen består av en front-end webapplikasjon bygget med Svelte, som tilbyr en brukervennlig plattform som lar forskere visualisere og organisere miljødata. Videre består løsningen av en back-end arkitektur som er ansvarlig for å levere webapplikasjonen, sende miljødata via RESTful API-enderpunkter, og håndtere autentisering, ved å benytte Nginx, PostgreSQL, PostgREST, og Django. Sikkerhet ble sett på i alle deler av utviklingsprosessen, der risiko ble identifisert og håndtert ved å følge relevante metoder og praksiser. Resultatet er en sikker og brukervennlig webapplikasjon som øker produktiviteten til NINA sine miljøforskere.

Acknowledgements

The bachelor thesis was written by Kavishnayan Nadarajah, Kevin Nikolai Mathisen and Carl Petter Mørch-Reiersen and Martin Solevag Glærum concluding our three years of Digital Infrastructure and Cyber Security at NTNU in Gjøvik. First and foremost, we would like to extend a heartfelt thank you to our supervisors at NTNU, Erik Hjelmås and Ernst Gunnar Gran. They have been extremely helpful guiding us throughout the project period. Additionally, we extend our gratitude towards NINA, especially Knut Marius Myrvold and Tobias Holter, for giving us this opportunity. They have shown great availability and willingness to help us throughout the project, something we deeply appreciate. We also want to thank Francesco Frassinelli and Nicolò Cantù from NINA's IT department for collaborating with us during the development of the application. Finally, we appreciate the teamwork and the collaborative effort among the group members throughout this bachelor project. This project has been challenging and educational, leaving us with an enhanced understanding of software development and security, and how to function in a team environment.

Contents

Abstract	I
Sammendrag	III
Acknowledgements	V
Contents	VII
List of Figures	XI
List of Tables	XV
Listings	XVII
Acronyms	XXI
1 Introduction	1
1.1 Background and Project Description	1
1.2 Delimitations	2
1.3 Target Audience	3
1.4 Project Goals	3
1.5 Group Background and Competence	4
1.6 Organization	5
1.7 Thesis Structure	6
1.8 Use of Artificial Intelligence	7
1.9 GitHub Repository	7
1.10 The Final Web Application	7
2 Theory	9
2.1 Methodologies	9
2.2 Theoretical Concepts	9
2.3 Technologies and Tools	11
3 Development Process	15
3.1 Scrum	15
3.2 Jira	16
3.3 Microsoft Security Development Lifecycle	17
3.4 DevOps	18
3.5 Risk Assessment	19
3.6 Dependency Management	23
3.7 Git	26
3.8 Quality Gates	30
3.9 Documentation	30
3.10 Commenting of Code	30

4	Requirements	33
4.1	Requirement Engineering	33
4.2	Requirement Gathering	34
4.3	Requirement Analysis	37
4.4	Requirement Specification	39
4.5	Requirement Verification	42
5	Design	43
5.1	Software Architecture	43
5.2	Design Principles	45
5.3	Site Map	48
5.4	Visual Outlines	48
5.5	Threat Modeling	50
6	Functionality Implementation	53
6.1	Code Structure	53
6.2	Retrieval of Data	56
6.3	Handling Data	60
6.4	File Handling	64
6.5	Leaflet	67
6.6	Plotly	70
6.7	Navigation	73
7	Security Implementation	75
7.1	Authentication Solution	75
7.2	Error Handling	78
7.3	Input Validation	80
7.4	Sanitization	85
7.5	Encryption	86
7.6	Content Security Policy	87
8	Testing and Quality Assurance	89
8.1	Unit Tests	89
8.2	End-to-end Tests	93
8.3	Static Code Analysis	94
8.4	GitHub Actions	95
8.5	Penetration Tests	97
8.6	Test Deployment	100
8.7	User Tests	101
9	Integration and Deployment	103
9.1	Docker	103
9.2	Nginx	105
9.3	Integration with NINA's Infrastructure Using Git Submodules	107
10	Discussion	109
10.1	Development and Collaboration	109
10.2	Requirement and Design choices	110
10.3	Coding Choices	111
10.4	Security Choices	113

10.5 Integration and Deployment Choices	114
10.6 Not Implemented	115
10.7 Evaluation of Group Work	116
10.8 Use of AI and GitHub Copilot	117
10.9 Sustainability	118
11 Conclusion	119
11.1 Results	119
11.2 Possible Improvements	121
11.3 Further Work	122
11.4 Final Remarks	122
Bibliography	123
A Web Application Screenshots	133
B Project Plan	145
C Group Rules	165
D Project Description	167
E Work Hours	171
F Sprint Reviews	173
G Jira Sprint Burnup Chart	179
H Use case diagram and Misuse case diagram	189
I User Test 1	203
J User Test 2	227
K User Test 3	239
L Threat Modeling	253
M Requirements Documentation	265
N Risk Assessment	277
O Code Implementation	287
P Meeting Minutes	333

List of Figures

1.1	Scientists at NINA collecting environmental data by using electric fishing	2
3.1	Screenshot of a Jira Scrum Board with Example Tasks	17
3.2	Dependency documented in Dependencies.md	25
3.3	Dependabot Alert for a express in Project Repository	26
3.4	Git repository structure	27
3.5	Gitflow diagram	29
4.1	Class diagram of the environmental data	34
4.2	Use case diagram	36
4.3	Misuse case diagram	37
5.1	Component interaction diagram of the svelte application	44
5.2	Network diagram of NINA's infrastructure	45
5.3	Sitemap of the application	48
5.4	Part of the Final Mockup, showing the map page with a station selected	49
5.5	Physical topology of the application	50
5.6	Logical topology of the application	51
5.7	Dataflow diagram of the application	52
6.1	Svelte and SvelteKit file structure	54
6.2	JavaScript modules file structure	54
6.3	Screenshot of the menu for selecting river or stations	60
6.4	Screenshot of the graph page with the whole sidebar visible side by side, where two stations are aggregated and two species selected	62
6.5	Screenshot of the map page with a station selected	64
6.6	Table containing the stations under a river when a river is selected on the map page	64
6.7	Screenshot of upload page with a file chosen	65
6.8	Screenshot of download page with stations, a species, and the XLSX format selected	67
6.9	Screenshot of the map page with a river selected	69
6.10	Screenshot of the graph page with a river and all its species selected	71
6.11	SvelteKit filesystem routes	74

7.1	Flow of application for managing authentication	77
7.2	Screenshot of the log in page	77
7.3	Error message when user tries to download without selecting a river	80
7.4	Error message when user tries to input an invalid character	82
8.1	Example CodeQL alert for the mock-server in the GitHub Repository	94
8.2	Input validation preventing invalid characters from being entered into the user- name field	98
8.3	Prohibited use of JavaScript syntax	98
8.4	URL injection attempt being denied	99
8.5	The network topology for the test-deployment-network running on Skyhigh . . .	102
A.1	Map Page	133
A.2	Observation summary of Gudbrandsalslågen river	134
A.3	Stations under Gudbrandsalslågen river	135
A.4	Fish data under Gudbrandsalslågen river	135
A.5	Summary of Gudbrandsalslågen 12 station	136
A.6	Fish data under Gudbrandsalslågen 12 station with satallite view selected	136
A.7	Topology view filtered after stations	137
A.8	List page	138
A.9	List filter search	138
A.10	Station list summary	139
A.11	Choosing rivers in graph	140
A.12	Bar char selected in the graph page	140
A.13	Pie-chart graph page	141
A.14	Histogram selected in the graph page	141
A.15	Box-plot selected in the graph page	142
A.16	Upload page	143
A.17	Download page	144
A.18	Login/Logout page	144
H.1	Use case diagram	190
H.2	Misuse case diagram	198
L.1	Physical Topology of the web application	254
L.2	Logical Topology of the web application	255
L.3	Data Flow Diagram for the web application	257
M.1	Use case diagram	269
M.2	Misuse case diagram	270
O.1	Svelte and SvelteKit File Structure	288
O.2	JavaScript Modules File Structure	288
O.3	Screenshot of menu for selecting river or stations	298
O.4	Example bar chart with two Stations and two species + others selected	301

O.5	Table containing the stations under a river when a river is selected on the map page	303
O.6	Screenshot of upload page with a file chosen	305
O.7	Screenshot of download page with stations, a species, and the xlsx format selected	308
O.8	Screenshot of the map page with a station selected	311
O.9	Screenshot of the graph page with a river and all its species selected	316
O.10	SvelteKit filesystem routes	319
O.11	Error message when user tries to download without selecting a river	324
O.12	How svelte uses css compared to the normal method	325
O.13	Using the html div element to style custom svelte html components	326
O.14	Error message when user tries to input an invalid character	328

List of Tables

3.1	Identified assets	20
3.2	Table showing example vulnerability	20
3.3	Table showing the session hijacking risk scenario	21
3.4	Table showing risk level given the probability and consequence	21
3.5	Table showing countermeasures	22
3.6	Table showing remaining risk after countermeasures	22
4.1	Subject-Object Matrix	35
H.1	Use case: Filter map based on data attributes	191
H.2	Use case: View map of data points	192
H.3	Use case: View data point summary	192
H.4	Use case: Filter list based on data attributes	193
H.5	Use case: View list of data points	193
H.6	Use case: View data as graphs	194
H.7	Use case: Upload data	195
H.8	Use case: Download data	196
H.9	Use case: Compare multiple data points	196
H.10	Use case: Aggregate multiple data points	197
H.11	Use case: Log in	197
H.12	Use case: Sign out	197
H.13	Misuse case 1: Compromise website through a compromised third party component	199
H.14	Misuse case 2: Denial of service	199
H.15	Misuse case 3: Inject, modify, or delete data	200
H.16	Misuse case 4: Download restricted data	200
H.17	Misuse case 5: Inject malicious script	200
H.18	Misuse case 6: Upload invalid data	201
L.1	Data Access Control Matrix	255
L.2	Description of relevant mis-actors	258
L.3	Model describing each mis-actor	258
L.4	Description of relevant non-human mis-actors	258
L.5	STRIDE definition	259

L.7	DREAD Ranking	260
L.6	Identified threats	263
M.1	Subject-Object Matrix	268

Listings

1.1	Code format example	7
3.1	NPM commands for installing and uninstalling packages	23
3.2	NPM commands for auditing packages	24
3.3	Output from NPM when running audit on the package XLSX	24
3.4	Example comments for the project in the function getObservationSpeciesCount() 31	
6.1	DateInput component	55
6.2	Request syntax for station_summary endpoint	56
6.3	Fetch request to PostgREST endpoint	57
6.4	getRiverSummary() function in datamanager module	58
6.5	Reading stores in a Svelte Component, where the rivers and stations variables will always contain the up to date content of the stores	59
6.6	Retrieving and setting the value of a store in a JavaScript module	59
6.7	The function filterRiversByDateAndSpecies() from the filterData module	61
6.8	Leaflet initialization	67
6.9	Code showing how a line is created for station markers connecting their start and end points	69
6.10	Use of map.flyTo() when selecting a station	70
6.11	Use of Leaflet component	70
6.12	Defining a plot layout in plotly	72
6.13	Defining a plot config in plotly	72
6.14	Use of Leaflet component	72
7.1	Set-Cookie header in authentication endpoint HTTP responses	76
7.2	The function authRefresh() from the auth module	78
7.3	The function fetchFromPostgrest() in the postgrest module	79
7.4	The function validateText() from the validation module	81
7.5	The JSON schema for the endpoint river-with-species	82
7.6	The function validateJson() from the validation module	83
7.7	The function validateText() from the validation module	84
7.8	The RiverOverview component	86
7.9	How the graph page updates the URL parameters for the selected rivers in the function updateUrl	86

7.10	The Content Security Policy for the application	87
8.1	Unit tests for the function getSelectableSpecies()	90
8.2	Unit tests for the component SpeciesInput	91
8.3	Mmocking using vitest	92
8.4	Test code for downloading data	93
8.5	Workflow for deploying the latest code on the developer branch to the developer test server	96
8.6	The Nginx reverse proxy configuration for sending traffic to the various services	100
8.7	Docker compose command to run the application with its test deployment	101
9.1	The Dockerfile for the main application	104
9.2	The event block in nginx.conf	105
9.3	The HTTP block in nginx.conf	106
9.4	The server block in nginx.conf	107
9.5	NINA's docker compose	108
0.1	DateInput Component	288
0.2	Use of button component with props and eventhandler	289
0.3	River_with_species SQL View	290
0.4	Request syntax for station_summary endpoint	291
0.5	Fetch request to PostgREST endpoint	291
0.6	getRiverSummary() function in datamanager module	292
0.7	updateStoreWithObjects() function in datamanager module	293
0.8	Initialization of riverStore in riverStore.js	295
0.9	Reading stores in a Svelte Component, where the rivers and stations variables will always contain the up to date content of the stores	295
0.10	Retrieving and setting the value of a store in a JavaScript module	295
0.11	The Station class with its fromJson method	296
0.12	The function filterRiversByDateAndSpecies() from the filterData module	298
0.13	The function filterObjectsBasedOnSpecies() from the filterData module	299
0.14	Bar and pie chart data structure	300
0.15	Histogram data structure, with the amount of laks between 0-4, 4-8, and 8-12 .	300
0.16	Box plot data structure, with the length of each laks observation	301
0.17	The function getIntervalsForObservations() in the plotlyData module	302
0.18	The function fishPerMinuteInStations() in the formatData module	303
0.19	The function formatstationsForSummary() in the formatData module	304
0.20	Function for browsing and selecting a file on the upload page	305
0.21	Function uploadFileToServer() in upload module	306
0.22	Function downloadFile() on download page	307
0.23	Function generateExcelFile() from fileHandler module	309
0.24	Leaflet initialization	311
0.25	createStationMarker() function	312
0.26	updateStations() function	314
0.27	Use of map.flyTo() when selecting a station	314

O.28 Use of Leaflet component	315
O.29 Creating a bar trace in plotly	316
O.30 Defining a plot layout in plotly	317
O.31 Defining a plot config in plotly	317
O.32 Drawing a plot in plotly	317
O.33 Use of Leaflet component	318
O.34 The function getUrlParams() used on the graph and download page	320
O.35 The function updateUrl() used on the graph and download page	321
O.36 The function fetchFromPostgrest() in the postgrest module	322
O.37 The function fetchFromPostgrest() in the postgrest module	323
O.38 The UserFeedbackMessage component, not including <style>	324
O.39 The function validateText() from the validation module	327
O.40 The Json schema for the endpoint river-with-species	328
O.41 The function validateJson() from the validation module	329
O.42 The function validateText() from the validation module	330

Acronyms

- AI** Artificial Intelligence. 7, 13, 92, 109, 117
- API** Application Programming Interface. 3–5, 12, 32, 40, 43, 45, 47, 56, 75, 76, 119, 120
- CSS** Cascading Style Sheets. 5, 9–11, 43, 53, 55, 67, 68, 70, 73, 85, 87, 111
- CSV** Comma-Separated Values. 40, 63, 64, 66, 79
- DOM** Document Object Model. 11, 91, 111
- HTML** HyperText Markup Language. 5, 9–11, 43, 44, 53, 55, 66, 68, 70, 72, 73, 80, 85, 86, 99, 110, 111, 113
- HTTP** Hypertext Transfer Protocol. 56, 57, 66, 76, 87, 101, 105, 106, 113, 114
- HTTPS** Hypertext Transfer Protocol Secure. 42, 44, 47, 76, 86, 106
- IP** Internet Protocol. 18, 101
- IT** Information Technology. 6, 18, 44, 119
- JSON** JavaScript Object Notation. 12, 47, 56–58, 60, 75, 78, 81–84, 100, 113
- MIT** Massachusetts Institute of Technology. 31, 40, 122
- NINA** Norsk institutt for naturforskning. 1–7, 16, 18–20, 22, 23, 25, 26, 33–35, 38–42, 44–46, 49, 65–67, 75, 76, 81, 86, 89, 97, 100–103, 107–109, 113–116, 118–120, 122, XI
- NTNU** Norwegian University of Science and Technology. 2–4, 6, 101
- REST** Representational State Transfer. 12, 45
- SQL** Structured Query Language. 5, 10, 12, 97, 99, 111
- URL** Uniform Resource Locator. 45, 57, 73, 74, 80, 86, 87, 97, 99, 108, 111, 114, XII
- XLSX** Excel Open XML Spreadsheet. 12, 40, 64–66, 79, 84, 99

Chapter 1

Introduction

1.1 Background and Project Description

Norsk institutt for naturforskning (NINA) is an organization researching the interactions between nature and society. Their overarching goal is to contribute to sustainable social development by delivering research-based and up-to-date knowledge about natural diversity, climate and society [1]. NINA has multiple offices throughout Norway, where the project of this thesis was requested by the department in Lillehammer.

Throughout their years of work, NINA has collected a considerable amount of environmental data from different rivers in Norway. The data is being collected using electric fishing, where the researchers stun the fish, allowing them to be examined and then released back into the water. This process can be seen in Figure 1.1. The data collected for each river includes the amount of fish captured, their species, size and gender. This data has so far been stored in different Microsoft Excel sheets, on several different computers. This resulted in some inefficiencies when it came to finding and using specific data, such as having to use email to distribute data and having to manually combine excel sheets. NINA therefore wanted help to create a system which could be used to visualize all of their collected data in an easy and efficient manner, which would allow them to focus more on research and less on technical complexities.

1.1.1 Problem Area

There are several methodologies, tools, and best practices for all parts of software development. It can be challenging to follow one methodology, let alone multiple, all while having to actually create a product. Moreover, in the current digital threat landscape, it is important to consider security throughout the development process, adding even more complexity to the development process.

Developing custom software for a client involves defining requirements, designing the software, writing code for its functionality and security, and verifying that it works as intended. Furthermore, software also has to be able to interact with other systems, and not only work on the developers computer, but work on various systems while handling high usage. If the software maintenance and development is intended to be transferred to another party, it is



Figure 1.1: Scientists at NINA collecting environmental data by using electric fishing

essential to clearly document and write maintainable software. Moreover, the development of software can be complex when interacting with already established data structure and systems, as the software has to be custom made to adapt and work with these. This is especially relevant when developing solutions to digitalize existing systems, such as the handling of environmental data for scientists.

This thesis presents how a custom web application was created, following the process through requirements, design, development, testing, and integration with existing systems. Furthermore, the thesis presents the various methodologies, tools, and best practices which were used during the development of software, while also managing its risk.

1.1.2 Project Description

The solution requested by NINA is a custom web application which can be used to visualize and handle their environmental data. This includes having a map to display observation points, and being able to search for observations based on name and date. It should be possible to view the data of observation points in tables, in addition to being able to visualize the data by plotting it in graphs, where you should be able to compare multiple points. Moreover, it should be possible to upload and download Excel files with environmental data in a specified format. The web application should let the user log in with their username and password from NINA's systems, and be able to run on their servers while interacting with their systems.

1.2 Delimitations

The project of this thesis were subject to delimitation from both NINA and NTNU.

1.2.1 Delimitations set by NINA

The project responsibility is only concerned with the development of the web application and its web server, not any of the services this depends on. The application has to be deployed with the use of a Dockerfile developed in the project, where NINA will handle its deployment. Furthermore, NINA will also configure the PostgreSQL database, PostgREST API, and Django. These services will handle the database and expose endpoints for data retrieval, upload, and authentication. The project will only consider the interaction with these services, and specify SQL Views for the PostgreSQL database. The web application does not have to be mobile friendly, as specified by NINA.

Because of these limitations the report does not cover the configuration of the services the web application depends on, nor does it cover security practices for detecting attacks and response management. It is agreed with NINA that this is their responsibility and therefore out of scope for this thesis.

1.2.2 Delimitations set by NTNU

The limitations set by NTNU was that the thesis needs to be finalized before May 21st, 2024. Due to the time constraint it was necessary to prioritize the central functionalities of the application rather than new features, ensuring that the product could interact with NINA's services as well as avoiding bugs in the application.

1.3 Target Audience

There are two intended target audiences for the thesis; those who are interested in the thesis itself, and those who are interested in the application developed.

1.3.1 Audience Interested in the Thesis

The thesis itself is targeted at those with a certain level of academic background within IT. They can be interested in how the theories and methodologies in the project was used, and the process of developing a custom web application.

1.3.2 Audience Interested in the Application

The application is of interest to the scientists at NINA, as they will be the ones using the final product. Moreover, as the application is open source, other developers who want to create similar custom web application to handle and display data could also be interested in the application. The code could either be used as inspiration or be built upon and modified further for displaying other types of data.

1.4 Project Goals

The goals of the project is split into the desired results, it's effects, and learning goals.

Result Goals

The project should result in the following.

1. Make a functional website and back-end fulfilling the requirements set by NINA. The website should provide an intuitive interface that simplifies the process for accessing, analysing, visualizing, and aggregating their collected data.
2. Deliver the web application packaged in a Dockerfile which NINA can use to deploy the web application without further modification or upkeep. It should seamlessly integrate with their infrastructure.
3. Deliver NINA with comprehensive documentation covering the development process for the web application, as well as the web application source code, to enable easy upkeep and patching by their internal IT-team.
4. Ensure that the web application has countermeasures in place to protect against possible threats, as well as clearly documenting any risks to NINA associated with the website.

Effect Goals

The final product should have the following effect.

1. The web application should streamline the workflow for NINA's scientists when trying to view, analyse, and visualize their collected data. It should be simple to locate a specific data point and gain an understanding of its underlying observations, which will save the scientists time and energy.
2. Make the process of testing hypotheses easier for the scientists, by easily downloading aggregated data of the specific observations they want to analyse, which should enable them to focus more on research and less on technical complexities, such as manually combining Excel documents for analyzing data.

Learning Goals

Throughout the project the team members want to learn the following.

1. Solving large projects given by an employer.
2. Experiencing working in a small group over a longer period of time to accomplish a singular task, while using the SCRUM methodology.
3. Acquire skills and learn technologies needed for creating a functional front-end service which can interact with APIs.
4. Learn how to handle and identify risk while developing software.
5. Learn to use Git and other development tools in a bigger projects.
6. Learn to use CI/CD for code testing and verification while developing software.

1.5 Group Background and Competence

The group consists of four students completing a bachelor degree in "Digital Infrastructure and Cyber Security" at NTNU Gjøvik. All group members are currently finishing their final

semester, with all members having completed all compulsory subjects leading up to the final semester. Throughout the bachelor study, the students have also completed several optional courses, which has provided each student with a unique set of skills and knowledge.

1.5.1 Academic Knowledge

All members were previously familiar with the languages HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript used for web development. Moreover, every group member was familiarized with setting up virtual machines in a cloud environment, and creating and running Docker containers. The degree has also provided experience with databases, Structured Query Language (SQL) and risk assessments for analyzing and managing risks.

Furthermore, three members had worked with secure software development methodologies and frameworks. Three had experience with penetration testing, which included knowledge about different common attack techniques used by threat actors to infect web application. All members had experience with programming, although with different levels of experience, where only some had previously worked with APIs and algorithms. Lastly, two group members had experience with statistics useful for working with data and representing it using plots.

1.6 Organization

1.6.1 Responsibilities and Roles

Each part of the development includes all group members, however the responsibilities for various aspects of development is assigned to each group member. The following is the division of responsibilities and roles in the project.

Group Leader / Scrum Master - Kavishnayan Nadarajah

This role is the main communication point between NINA, the supervisors and the group. Is responsible for keeping track of all the different exercises which needed to be completed, and making sure to fairly assign work within the group.

Head of CI/CD - Carl Petter Mørch-Reiersen

This role is responsible for setting up the build and test environments. Is also responsible for ensuring CI/CD by using GitHub Action.

Head of Design - Martin Solevåg Glærum

This role is responsible for the design for the web application. They should ensure that the web application follows standards for web design, and that it accomplishes all functionality requested.

Head of Application Logic and Integration - Kevin Nikolai Mathisen

This role is responsible for the handling and processing of data, and the interaction with the back-end services. Is also responsibly for ensuring the web application integrates with and can be ran on NINA's infrastructure.

1.6.2 Client

The client for this project is NINA, located in Lillehammer. The communication with NINA was primarily through Knut Marius Myrwold and Tobias Holter. Both are scientists working for NINA. The group members also worked closely with Francesco Frassinelli and Niccolò Cantù, which are parts of NINA's IT department. Frassinelli and Cantù were responsible for the services the web application developed depends on, and was therefore an integral part when integrating the web application with NINA's infrastructure.

1.6.3 Supervisors

The supervisors for this project are Ernst Gunnar Gran and Erik Hjelmås. Both are part of the department of Information Security and Communication Technology at NTNU. The group had weekly meetings with them throughout the project. In the meetings the supervisors answered questions and gave tips and information to guide the thesis and project in the right direction.

1.7 Thesis Structure

The thesis contains twelve chapters, with each chapter having multiple smaller sections covering a specific topic. Further details about each chapter of the thesis can be seen in the list below.

- **Chapter 1 - Introduction:** Introduces the purpose of the thesis and relevant background information
- **Chapter 2 - Theory:** Lists the methodologies, technologies, and tools useful for understanding the thesis.
- **Chapter 3 - Development Process:** Explains the methods and frameworks the group followed during the project duration.
- **Chapter 4 - Requirements:** Explains the methods and approach the group followed when defining the requirements of the web application.
- **Chapter 5 - Design:** Presents the reasoning and process behind designing the web application
- **Chapter 6 - Functionality Implementation:** Covers the coding choices and implementation of the web applications functionality
- **Chapter 7 - Security Implementation:** Covers the coding and implementation of the web applications security.
- **Chapter 8 - Testing and Quality Assurance:** Explains the reasoning behind testing choices, and how testing was performed

- **Chapter 9 - Integration and Deployment:** Covers the process and code for integrating the web application into NINA's system
- **Chapter 10 - Discussion:** Contains discussions surrounding the final product and different choices made during the project
- **Chapter 11 - Conclusion:** Covers the group's final thoughts on the project process and the finished product

1.7.1 Special Formatting

For this thesis large code examples, configuration files, and commands from the application are illustrated in the format shown in Listing 1.1. Shorter examples of code, syntax, file names, and commands are placed directly in the text using in-line formatting: `example_code`;

```
function JustAnExample (myVariable) {  
  myVariable = myVariable * 2  
  return myVariable  
}
```

Listing 1.1: Code format example

1.8 Use of Artificial Intelligence

During the project Artificial Intelligence was used in some extent to help in coding. The group utilized both GitHub copilot and ChatGPT. This allowed for faster coding, as the Artificial Intelligence (AI) tools could suggest code, help with problem solving, and assist with debugging and bug fixing. AI tools was not used when writing the thesis. The use of AI is covered more in depth in Section 10.8.

1.9 GitHub Repository

The final version of the application is stored on GitHub ¹, and is available as an attachment to the thesis. NINA made it clear that they wanted all code to be public. This makes it possible for anyone to access the code for the web application and either be inspired by it or build upon it. Readers are encouraged to clone the repository, run the code, and try to access the website, simply following the process outlined in the project `README.md`. Downloading and running the code won't include all the intended features, since the application is designed to integrate with NINA's infrastructure.

1.10 The Final Web Application

To give readers and understanding of what is being developed throughout the thesis, the final web application main features can be seen in Appendix A.

¹<https://github.com/KevinMathisen/interactive-database-for-environmental-data>

Chapter 2

Theory

This chapter contains definitions of methodologies, technologies, and tools used throughout the thesis. When these concepts are used in the thesis they link to the theory chapter. The theory chapters is therefore mainly intended as a dictionary for the concepts and theories used, which is why each concept is defined in their own isolated sections.

2.1 Methodologies

Microsoft Software Development Lifecycle (Microsoft SDL) is a process for introducing security and privacy considerations into all phases of software development[2].

DevOps is a software development methodology used for automating the work of software development (Dev) and IT operations (Ops), shortening the systems development life cycle[3].

Continuous integration / Continuous Deployment (and Delivery) is the practice of using automation in testing, building, and deploying code. This is used to increase productivity, and provide faster release cycles [4].

2.2 Theoretical Concepts

Dependency Pinning is the process of pinning specific versions of libraries or dependencies in a project. [5].

A **Web application** is software accessed using a web browser. It is stored on a server and delivered to users over the internet. This server sends the relevant data, usually HTML, CSS, and JavaScript [6].

Client-side hydration is a technique used by client-side JavaScript for converting static HTML web pages to ones with dynamic content and events in the browser. [7].

Single page application is a web application implementation which only loads a single web document. The body content of this single document is updated using JavaScript, allowing users to move between pages without having to reload the web application [8].

Separation of Concern is about breaking down complex system into smaller parts, which is a fundamental principle in software engineering. The aim is to separate organize the components in the system to only focus on single tasks in the systems [9].

JavaScript modules are JavaScript files with a function or group of similar functions serving a similar purpose. These functions can be called by other parts of the system [10].

Svelte Component in Svelte is referred to as the building blocks of the application. It is files containing code written in HTML, CSS and JavaScript, serving a single and well defined purpose, for example a button or a table. This can then be called and used in other parts of the application [11] [12].

Svelte Props is used in Svelte as a way to pass data from one component and over to its child components, allowing communication between components. This is done by declaring the variables with the export keyword [13].

Svelte stores are global data repository in Svelte applications for containing data. Components can get notified when certain store values change by subscribing to them [14].

SQL Views are virtual SQL tables, where the contents of the virtual tables are defined by SQL queries. SQL Views can then be accessed directly, instead of having to write custom SQL queries each time the same data is accessed [15].

Content Security Policy (CSP) is a security measure designed to reduce specific attack types. This includes attacks like data injection attacks and Cross-Site Scripting. Content Security Policies does this by defining which resources a web page can use, preventing resources such as JavaScript from being loaded from unsafe sources or being injected [16].

Reverse proxy is a server acting as an intermediary between a client requesting a resource and the server providing that resource. It forwards the client's requests to one or more servers instead of handling the request for the specific resource [17] [18].

Git submodules allow one to have a git repository as a subdirectory of another git repository. By doing this, one can clone another repository into the project and keep the commits separate [19].

Dockerfiles are text files containing commands used for building Docker Images [20].

Docker images are read-only templates that contain instructions used to make a docker container [21].

Docker container is a standalone executable package of software, including the software's code and dependencies, which can run on any system [22].

Docker Compose is a tool for defining and running applications utilizing multiple docker containers [23].

2.2.1 Software Testing Techniques

Unit tests is a method for separately testing smaller parts (units) of large systems, validating that each part works in isolation [24].

End-to-end (E2E) tests are a testing method for testing the data application flow between components, seeing how the sub-systems work together [25].

Static Security Testing are tests scanning code for vulnerabilities without running it [26].

Linting is a static tool that analyses the source code and reports if it identifies any syntax errors, styling inconsistencies, insecure variables or bugs [27].

Penetration tests (pen tests) is a testing method where security expert attempts to find and exploit vulnerabilities in a system, simulating a real attack by real threat actors [28].

User tests is a way to gauge the opinion of customers by letting them test your application, allowing you to watch, hear, and review their interactions with the application [29].

2.3 Technologies and Tools

Svelte is a front-end open source JavaScript framework used for web development. Svelte is component based, making it possible to create small and compact components with HTML, CSS and JavaScript. It operates as a compiler, shifting most of the work away from the browser. Commonly, the Document Object Model (DOM) is updated dynamically during runtime of the application. With the use of Svelte one can create optimized JavaScript code during the building process of the application, and this optimized code is intentionally made to more efficiently update the DOM in the browser [30] [31].

SvelteKit is a framework powered by the concepts of Svelte and Vite, providing more advanced features to the existing svelte framework and simplifying the process of developing web applications. The core of SvelteKit is its routing functionality, for example enabling the development of single page application [32] [33].

Leaflet is an open source library which allows for the use of interactive maps with JavaScript. It has multiple different map types to select from, and allows for placing markers objects on the map based on configured coordinates [34].

Plotly is an open source charting library, providing interactive visualization tools for web applications. Some chart options possible with plotly are line charts, bar charts, pie charts, histogram and box plots [35].

Exceljs is an open source library used when working with Excel spreadsheets in web applications. It allows for creating, reading and changing XLSX spreadsheets. An important feature is the Exceljs ability to translate JSON data into an XLSX spreadsheet [36] [37].

Docker is an open source tool used to deploy applications in a sandbox to run on operating systems [38]. Docker packages application (including all dependencies system tools, code, and runtime) into a standardized unit called containers. The standardization ensures the application is consistent across different environments and simplifies the process of deploying the application. Containers use operating system virtualization, similar to Virtual Machines (VMs), containers simply run on top of the existing operating system, without the need to emulate a new one [39].

Nginx is an open source software used as a web server. It can also function as a reverse proxy, load balancer, mail proxy and HTTP cache [40].

PostgreSQL An open source object-relational database system using the Structured Query Language (SQL) language [41].

PostgREST is a web server that turns a PostgreSQL database directly into a RESTful API. PostgREST creates a REST API which the web application uses to interact with a SQL database [42].

Django A free open source Python-based framework used to make it easier for developers to create web applications. It comes with features such as a login system, database connection and CRUD operations[43].

Node.js is an open source JavaScript runtime environment, enabling applications to run JavaScript code outside of browsers. Node.js makes it possible to create web servers, which are needed for web applications to execute their code on the server-side [44].

NPM is a package manager for JavaScript. It is the default package manager for any environment created with Node.js. A package manager enables installing, upgrading, configuring and removing packages [45].

Vitest is a unit testing framework powered by Vite. It includes features like mocking and snapshots [46].

Express is a web application framework for Node.js [47].

GitHub Copilot is a handy extension in visual studio code that allows the user to get AI suggested code during development. The AI can both create code based on questions or instructions [48].

GitHub Actions is a way to automate CI/CD workflows in repositories hosted on GitHub, including running scripts in the repository, testing the source code and deploying the code to the right environment [49].

Workflow are files that define what GitHub Actions should do, and when they should be ran [50].

Eslint is a tool that statically analyzes code to find, fix, and notify about linting problems [51].

CodeQL is a tool that discovers vulnerabilities statically analyzes code to find, fix, and notify about linting problems [52].

CWE stands for Common Weakness Enumeration and contains a community-developed list of common hardware and software weaknesses. A weakness is defined as a condition under certain circumstances that could contribute to introducing vulnerabilities [53].

Chapter 3

Development Process

In this chapter the methodologies and practices used throughout the whole development process are presented. This includes how the work was organized with the use of Scrum, Jira, and various communication channels. It also explores the use of DevOps and Microsoft Security Development Lifecycle during the stages of the development process. Moreover, risks were identified throughout through a risk assessments. Lastly, the chapter also includes practices followed when handling dependencies, using version control, and commenting code in the development process.

3.1 Scrum

Scrum has been used throughout the development process to organize and distribute work among team members. According to The Scrum Guide [54] one should look at scrum as a way to work in a team and complete goals, where the goals are divided into smaller tasks. Small tasks are created to accommodate for continuous feedback and improvement along the way, which are one of the key principles of scrum and agile development. This enables the development in the project to interactively build the application in small tasks, where it is easy to modify the work based on continuous feedback.

3.1.1 Sprints

According to the Scrum Guide [54], sprints are the “heartbeat in scrum”. Sprints are a fixed length period of work with short iterations to facilitate for frequent feedback and changes. New sprints start immediately after the previous sprint is concluded, a process which repeats itself until the completion of the project. A scrum sprint is comprised of a series of tasks which includes sprint planning, multiple daily scrum sessions, a sprint review and a sprint retrospective held at the end of each sprint session.

The process around sprint planning is about painting a clear picture of which tasks needs to be completed to achieve the sprint goal for a given sprint session. The planning process is based on the current stage the team is in the development process, illustrated in the attached Gantt chart, see Appendix B. The process around adding new tasks and adding story points is future clarified in the Jira section 3.2.

In the project the sprint lasts for two weeks, where every sprint is concluded with a sprint review. The sprint review is used to review the completed work, by discussing what went well, and what needs to be improved for the upcoming sprint. The uncompleted tasks are also rescheduled for the upcoming sprint session. The summary's from the sprint reviews can be read here, see Appendix F.

3.1.2 Meetings

Throughout the thesis, regular meetings were held with three separate groups; the daily team meetings, meetings with the supervisors every week, and meetings with the assigners every other week.

The daily scrum meetings are used to announce ongoing work, where each group member can address any questions or problems that have arisen. These meeting has proven essential for both maintaining a consistent workload and for fostering the teamwork.

Meetings with the supervisors are used to present and discuss the status on the web application and thesis. These meetings provided the group with valuable feedback, ideas and recommendations guiding the project in the right direction.

Meetings with the assigners at NINA are mainly used to present the progress on the web application. The purpose of these meetings is to make sure that the requested features are implemented, discuss any additional features and sometimes even performing user tests, to test the usability of the application.

These meetings have strongly contributed to improving the project and the team by fostering discussion and addressing problems and concerns. All the meetings with the supervisors and assigners are documented, see meeting minutes in Appendix P. Utilizing the meeting minutes has proven valuable in providing answers, improvements and other recommendations which have enhanced the product quality and the teamwork.

3.2 Jira

Active use of Jira makes it possible to track tasks with the scrum methodology in mind. Jira is a project management tool that comes in handy for scheduling, delegating and updating the status of the tasks added to the scrum backlog [55]. The management tool includes useful scrum features like the backlog, the scrum board and progress reports. These features make it easy for everyone to know what is being worked on, what needs to be done, and how much work is left to do for the current sprint session.

Every sprint is started by adding tasks to the backlog, as outlined in Section 3.1.1. Epics are used to categorized the tasks based on their purpose, for example documentation or development. This helps keeping the backlog and the scrum board clean and easy to read.

The scrum board helps in updating the status of the task between three main stages which are “to do”, “in progress” and “done” (See Figure 3.1). This helps the group members gain an overview of the completed tasks, what other group members are working on, and what needs to be done. Unassigned tasks are open for everyone after the first come first serve rule. Adding story points to tasks makes it easier to differentiate and give an indication on how much time

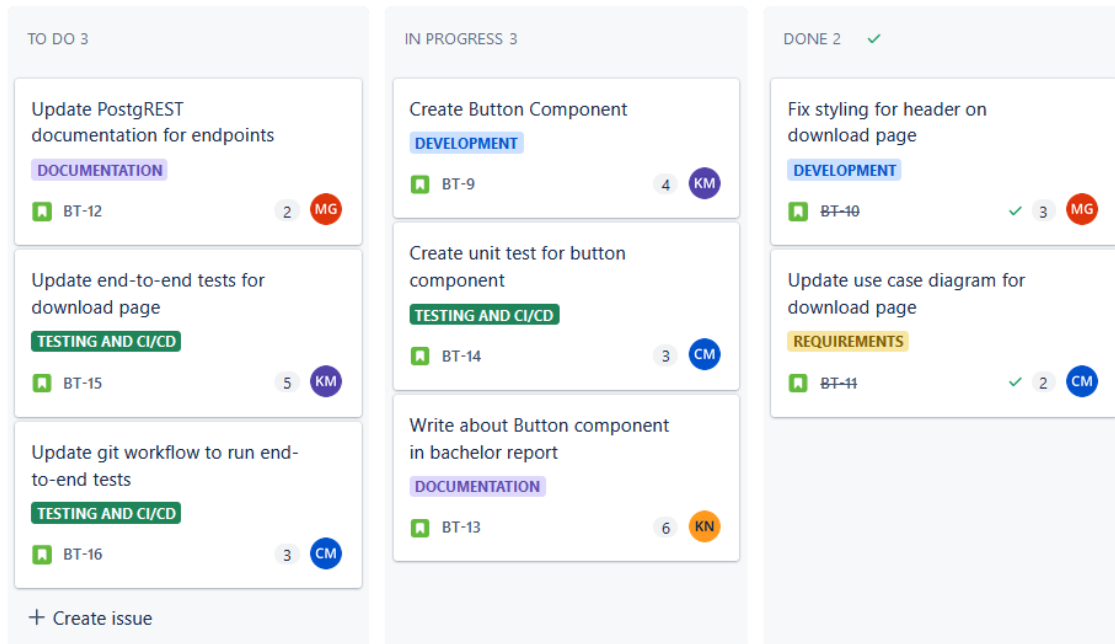


Figure 3.1: Screenshot of a Jira Scrum Board with Example Tasks

different tasks takes. The estimate for the value of one story point is set to correspond with one hour of work. The values are determined at the start of the sprint session, but are updated to correspond with the time used on the task.

Story points are also used to visualize the progress on the sprint using sprint reports. These reports contribute by providing a guideline for the pace that should be held to meet the goal and complete all the scheduled walks. The reports from all the sprint sessions can be seen in the Appendix G.

3.3 Microsoft Security Development Lifecycle

The Microsoft Security Development Lifecycle (Microsoft SDL) is a methodology for including security and privacy throughout the phases of software development [2]. The overarching phases of Microsoft SDL are followed in the thesis, however parts of each phase were skipped or not done in its entirety. The project focuses more on following the general approach and practices outlined by Microsoft SDL, not replicate every step. The following describes the phases of the Microsoft SDL, and how certain practices of the Microsoft SDL was implemented into our project.

When defining requirements for the application, Microsoft SDL states that a risk assessment should be constructed to gain an overview over the potential risks, which the project follows as seen in Section 3.5. Moreover, security requirements should be explicitly defined, as outlined in Section 4.4. This helps guide the development by considering the security of the application early on, where requirements considers the risk assessment. Quality gates, described

in Section 3.8, are also recommended to ensure a consistent level of security and quality along the development.

During the design phase, threat modeling should be conducted to visualize potential threats and vulnerabilities in the final product. The threat model is created based on the proposed design and requirements, combined with the knowledge of NINA's infrastructure. Following this, the threat model is used to update the risk assessment. This contributed to identifying and handling potential risks within the design process, reducing the changes of new risks being identified later in the development process which could require costly redesign.

Following Microsoft SDL in the implementation includes documenting dependencies and tools in detail, and only use approved tools (see Section 3.6). Furthermore, for all code created it should be statically analysed, which is done in the project by using CodeQL and linting, as seen in Section 8.3.

As part of verification dynamic analyses of the application should be conducted. Unit tests and end-to-end tests are continuously ran on the application, as seen in Section 8.1 and Section 8.2. These ensures new changes don't break existing functionality, keeping the integrity of the application. Furthermore, penetration tests are also conducted ensuring that all counter-measures works as intended, as seen in Section 8.5. Moreover, the risk assessment is updated continuously, identifying new risks and planning new safeguards (see Section 3.5).

The release of the application is NINA's responsibility. Microsoft SDL specifies that this should include creating incident response plans, detecting attacks, and deploying the application with a secure configuration. As specified in Section 1.2, this is out of scope for the thesis.

3.4 DevOps

The development process followed includes core features of DevOps. DevOps is a process for automating software development (Dev) work and IT operations (Ops). This is done to shorten and improve the software development life cycle [3]. There is no definitive definition of DevOps, however it revolves around some core characteristics. Other development philosophies, such as Waterfall, separates between development and deployment, resulting in longer development time and fewer, larger releases at a time. DevOps on the other hand focuses on constant cooperation, workflow automation and continuous and fast feedback.

DevOps encourages communication between developers, operations, and product owners. In the project the group held daily meetings discussing which features are being implemented, and weekly meetings with NINA. Meetings with NINA allows the developers to get constant feedback about the software, resulting in a continuously evolving product specification and requirements. The developed web application is deployed on a test server, allowing NINA to access it over a public IP. This gives NINA the opportunity to provide feedback outside meetings, making it easier for NINA to deliver feedback. Their suggestions can then immediately be considered, resulting in a quick implementation of new suggestions. This results in the development being flexible, focusing on constant small improvements and features instead of larger changes. Consequently, a continuous flow of development is created, making it easy to integrate new features, or remove unneeded features.

DevOps also encourages automation, and continuous integration and deployment. This is achieved in the project by utilizing GitHub Actions, where unit tests, end-to-end test, security

scans, and linting are all automated. This creates a continuous workflow where new features, functionality, and code changes are checked to make sure they can seamlessly merge into our developer/main branch (see Section 3.7.5). Furthermore, the application is continuously deployed to test servers allowing NINA to test the application and developers to view the features being developed on a live server.

3.4.1 DevSecOps

DevSecOps, or Secure DevOps, is a modernized version of DevOps, integrating security into the development process. This is related to Microsoft Security Development Lifecycle, where both encourages a continuous focus on security. In DevSecOps, each member of the development is burdened with constantly keeping possible risks and safeguards in mind. Safeguards and countermeasures should be continuously developed, making sure that all parts of the software is secure. This approach makes software more robust and secure since security is integrated as part of the system itself, instead of being added on top of the final product. Security is therefore considered when defining requirement, in the design by following design principles and doing threat modeling, and in the development by developing countermeasures. The software is also explicitly tested for security vulnerabilities by security scanning and penetration testing, while continuously updating a risk assessment.

3.5 Risk Assessment

A risk assessment for the application was conducted in accordance to the ISO27005 standard. The ISO27005 standard outlines a risk management framework for identifying, analyze, and manage security risks [56]. When following the ISO27005 standard in the project there were some deviations, where threat identification is skipped to simplify the process. The risks assessment is continuously updated while the web application is developed, ensuring that possible risks are tracked and protected against during the entire development process. The risk assessment is found in Appendix N, this section only outlines the process followed and some important findings.

3.5.1 Context Establishment

The first part of the risk assessment is context establishment. ISO27005 defines context establishment as collecting all information relevant for the risk assessment [56]. This includes defining the internal and external context, the scope of the assessment, how risk is defined in the assessment, and what constitutes acceptable risk. Defining the context then provides the groundwork for the rest of the risk assessment.

Asset Identification

The next part is to identify the assets of the application which needs protection. For the developed application the assets includes environmental data, the web server and source code, and user sessions. This can be seen in Table 3.1.

Asset	Description	Confidentiality	Integrity	Availability
Environmental Data	Data collected by NINA (River data, station data) that is used to plot the graphs.	Confidential	Critical	2 days
Web server and website source code	The web server the handles requests, and the website which is delivered to the client.	Open	Critical	2 days
User Sessions	The sessions which authenticates the users.	Strictly confidential	Expected	2 days

Table 3.1: Identified assets

Vulnerabilities

By using OWASP top ten [57] and misuse cases outlined in 4.2.4, an overview of possible vulnerabilities is created. Vulnerabilities are weaknesses that can be exploited by attackers. The vulnerabilities identified are; Insecure session handling, Vulnerable and outdated components, Data Integrity Failure, Injection, Security misconfiguration, Insecure design, Sending data to the client, and No rate-limiting for login. Each vulnerability is depicted with a detailed description in the risk assessment Appendix N. The vulnerability Insecure session handling can be seen in Table 3.2.

Id	Vulnerability	Description
1	Insecure session handling	Refers to the poor protection and management of user session data, which could allow unauthorized access or manipulation of user accounts and sensitive information. Based on <i>A07:2021 – Identification and Authentication Failures</i> from OWASP Top 10 [57]

Table 3.2: Table showing example vulnerability

3.5.2 Risk Identification

The next part of the risk assessment is risk identification. This involves creating relevant risk scenarios based on the identified assets and vulnerabilities. These scenarios consist of threats which exploit vulnerabilities to cause harm to the assets identified. The ones found are; Session Hijacking, Compromised Website, DDOS attack, Environmental data is modified or deleted, Copyright is not handled correctly, Regulations are not followed, and Environmental data is stolen. The risk Session Hijacking can be seen in Table 3.3.

Id	Risks	Description	Vulnerability	Assets
1	Session hijacking	Session hijacking by Cross-Site Scripting, brute forcing credentials, or another vulnerability, allowing an unauthorized person to steal a legitimate and authenticated user's active session, which can lead to data theft, data modification, or impersonation.	Improper session handling, Injection, Insecure design, No rate-limiting for login	User Sessions

Table 3.3: Table showing the session hijacking risk scenario

3.5.3 Risk Analysis

Risk analysis is described in ISO27005 as identifying, quantifying or qualitatively describing risks, and prioritizing them against the risk evaluation criteria. This is done by ranking the probability and consequences of risks according to the definition in the context establishment, and combining these scores to assign each risk a score. This can be seen for the risks Session Hijacking, Compromised Website, and Denial of Service in Table 3.4.

Nr	Risk Scenarios	Probability	Consequences	Risk
1	Session hijacking	High	Very serious	Very High
2	Compromised website	Medium	Very serious	High
3	Denial of Service (DDOS)	Low	Slight	Low

Table 3.4: Table showing risk level given the probability and consequence

Risk Treatment

Next, to reduce the risk scenarios identified to an acceptable level, risk treatment is done. This includes identifying countermeasures to reduce the risk. The identified countermeasures were; Proper error handling, Input validation, Output sanitization, Content Security Policy, Secure Design Principles, Handling sessions according to best practices, Pentesting, Continuous Testing, Dependency management, Secure configuration and deployment, Rate limiting requests, Logging, Backup, Explicitly reviewing copyright and licenses, Researching and following all applicable regulations. Each countermeasure gets an ID, a name, a description, and which risks they prevented. The table 3.5 contains some of the identified countermeasures.

Id	Counter-measure	Description	Risks mitigated
6	Handling sessions according to best practices	Using best practices for handling sessions, such as Samesite cookie, Secure cookie, and HttpOnly cookie, as well as using proper encryption and setting lifetime of tokens. This will make it harder to compromise and steal someones session.	1
8	Continuous Testing	By continuously doing testing during the development, in the forms of Unit tests, Security scans, and end-to-end testing. This will ensure the application functions as intended both under normal usage and unexpected scenarios.	1, 2, 3, 4, 7

Table 3.5: Table showing countermeasures

Risk Modification

After applying the countermeasures the new risk level should be calculated. The remaining risk is the estimated risk level after all countermeasures have been implemented. The table 3.6 outlines the remaining risk for the risk scenarios Session Hijacking, Compromised Website, and Denial of Service.

Risk ID	Risk	Countermeasures	Initial risk	Remaining risk
1	Session hijacking	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	High	Very Low
2	Compromised website	1, 2, 3, 5, 7, 8, 9, 10	High	Very Low
3	Denial of Service (DDOS)	1, 5, 7, 8, 10, 11, 12	Low	Very Low

Table 3.6: Table showing remaining risk after countermeasures

3.5.4 Risk Acceptance

Finally the risk acceptance by NINA is documented. It is checked if the remaining risk is under the acceptable levels NINA defined in the context establishment. If not, NINA should either accept them or decide on how to handle it. When NINA did not accept the risk, redesign of parts of the application or implementation of countermeasures were done. For example, the application originally did not have any authentication for viewing environmental data, only for uploading and downloading data. This is identified in the risk assessment as a risk of environmental data being stolen. Because NINA views it as an unacceptable risk, it is handled by redesigning the application to only allow authenticated users to view environmental data. By doing the risk assessment and ensuring that NINA accepts the risk, unwanted risk associated with the application is reduced.

3.6 Dependency Management

In software development there is often no need to write all code from scratch, as there exists millions of packages developers can leverage when developing software. As of September 2022, there are over 2.1 million packages listed in the Node Package Manager (NPM) alone [58]. By leveraging dependencies developers can save time and increase the robustness of their software. This is because the developers do not have to re-implement existing functionality, which could result in a less secure and efficient solution. However, there are potential risks associated with using dependencies, such as insecure supply chains and vulnerabilities in their code. Furthermore, using dependencies can quickly become disorganized when you have multiple dependencies, all with their own sub dependencies, all being updated regularly. To address these issues, and secure the supply chain, the following tools and practices are used during the development process.

3.6.1 Choosing Packages

When choosing packages as dependencies there are several factors to take into consideration. Wendt proposes ten rules for choosing R packages [59], which include checking how a package is shared, its supporting documentation, if it has a large and active community, and if it is well maintained. NINA also required that all dependencies were open source and preferably actively maintained, see Appendix D. Based on this the requirements in our project for choosing packages are:

1. The package has functionality we need
2. The package is available from the NPM registry
3. The package is well documented
4. The package has a large, active community
5. The package is actively maintained
6. The package is open source

These rules ensure that only packages needed are used. By only using NPM the handling and organization of all dependencies are standardized. Ensuring the packages are well documented and has an active community allows developers easy access to resources for implementing the packages into their software. Finally, by ensuring that the packages are actively maintained and open source, the risk of hidden or unpatched vulnerabilities is lowered.

3.6.2 Package manager

All of the packages in the project are managed by NPM. It consists of both a registry with packages and a command line interface. The commands in Listing 3.1 will install and uninstall packages.

```
npm install <package-name>
npm uninstall <package-name>
```

Listing 3.1: NPM commands for installing and uninstalling packages

To ensure the integrity of the dependencies managed by NPM, the commands from Listing 3.2 can be used.

```
npm audit
npm audit signatures
```

Listing 3.2: NPM commands for auditing packages

The command `npm audit` will check if there are any known vulnerabilities for the dependencies in your project [60]. The command `npm audit signatures` will verify the integrity of the packages by checking if their signatures are valid.

When finding a package for handling Excel files one of the packages considered in the project was *SheetJS* [61]. By installing it using `npm install xlsx` and running `npm audit` it can quickly be discovered that the package has a vulnerability, as seen in Listing 3.3 of the `npm audit` output. After some research it was learned that the project had migrated from NPM to its own website. [61], thereby deprecating the package in the NPM registry and allowing vulnerabilities to exist. Because the package breaks the second rule when choosing a package, as the maintained version is not available in the NPM-registry, another package named *ExcelJS* was used instead.

```
PS C:\path\to\your\directory> npm audit
# npm audit report

xlsx *
Severity: high
Prototype Pollution in sheetJS - https://github.com/advisories/GHSA-4r6h-8v6p-xvw6
SheetJS Regular Expression Denial of Service (ReDoS)
  - https://github.com/advisories/GHSA-5pgg-2g8v-p4x9
No fix available
node_modules/xlsx

1 high severity vulnerability

Some issues need review, and may require choosing
a different dependency.
```

Listing 3.3: Output from NPM when running audit on the package XLSX

3.6.3 Document Dependencies

To ensure consistency and clarity of which dependencies are used in the project the files `package.json` and `Dependencies.md` is placed in the Git repository. The file `package.json` is automatically generated by NPM for each Node project, in this case for the directories `/client/` and `/mock-server/`, and contains information about all dependencies and their versions for each project. Furthermore, `Dependencies.md` contains information about all dependencies used in the whole repository, which includes their purpose, version, how to install

them, and a link to their official documentation. Figure 3.2 shows one of the dependencies that is used, and how it's documented. By clearly documenting the dependencies it is easier for developers, especially new ones, to work with and maintain the project.

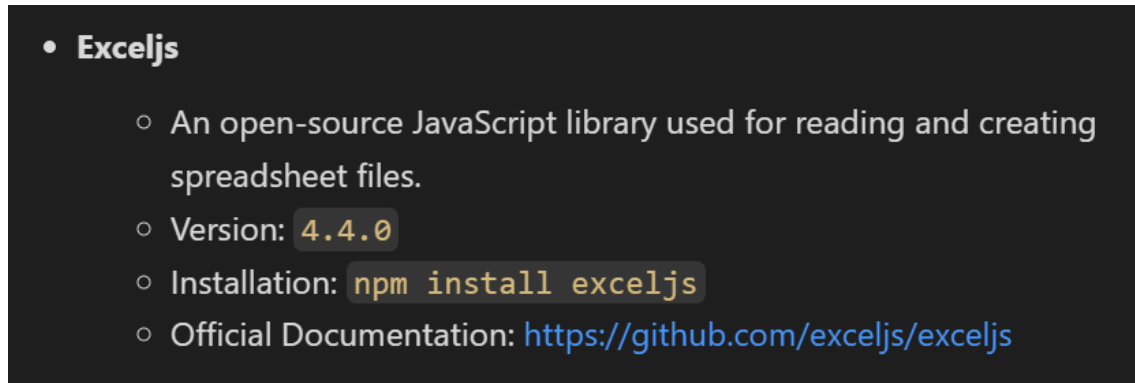


Figure 3.2: Dependency documented in Dependencies.md

3.6.4 Versioning and Dependency Pinning

To set the specific version of dependencies in the project, which is called Dependency Pinning, you simply have to specify the version in `package.json`. For example, you can specify that a package should be the exact version `1.0.1`. To gain more granular control over the version of dependencies, *semantic versioning syntax* can be used [60]. For example, to accept all minor releases and patches of a package, but not any new major releases, the `^` symbol can be used. For example, `^1.0.1` will accept any versions up to, but not including `2.0.0`. However, as NINA specifically wanted dependency pinning (see Appendix D) the project uses specific versions.

3.6.5 Tree Shaking

When using dependencies they can often have more functionality than needed. In that case, especially when it comes to web applications, it seems wasteful to include the unused code and thus increase the size of the software bundle sent to the user. Luckily, as SvelteKit is built on top of Vite [62], which uses Rollup 4 [63], it uses a concept known as *tree shaking* to remove any unused code in your software, including imported packages [64]. This prevents the use of dependencies with large amounts of functionality and code to unnecessarily increase the bundle size of your software, as only the code from dependencies directly used in the software is included when building it.

3.6.6 Dependabot

In addition to using `npm audit`, another dependency vulnerability scanner used is *Dependabot* [65]. Instead of running Dependabot in the command line, it is configured for a Git repository on GitHub. It finds all dependencies listed in the repository's package ecosystems, which is `package.json` from NPM in this project. Dependabot checks the dependencies up against

GitHub’s advisory database, which aggregate multiple vulnerability sources, including the National Vulnerability Database and NPM security advisories database [66]. The advantage of Dependabot is that it will notify the owner of a Github repository when a vulnerability is found automatically, where you do not have to manually run a vulnerability scan. This is especially useful when NINA takes over the maintenance of the project, as they will automatically get notified of any vulnerabilities in the dependencies, even if they do not regularly run security scans in the command line.

An example of how a Dependabot alert looks like is shown in Figure 3.3. This is taken from the project GitHub page, and it outlines a vulnerability in Express.js which is used in the mock server. The alert specifies the vulnerability, where it is located, its severity, and links the vulnerability to CWE. The alert also specifies how you can update your dependency to fix the issue.

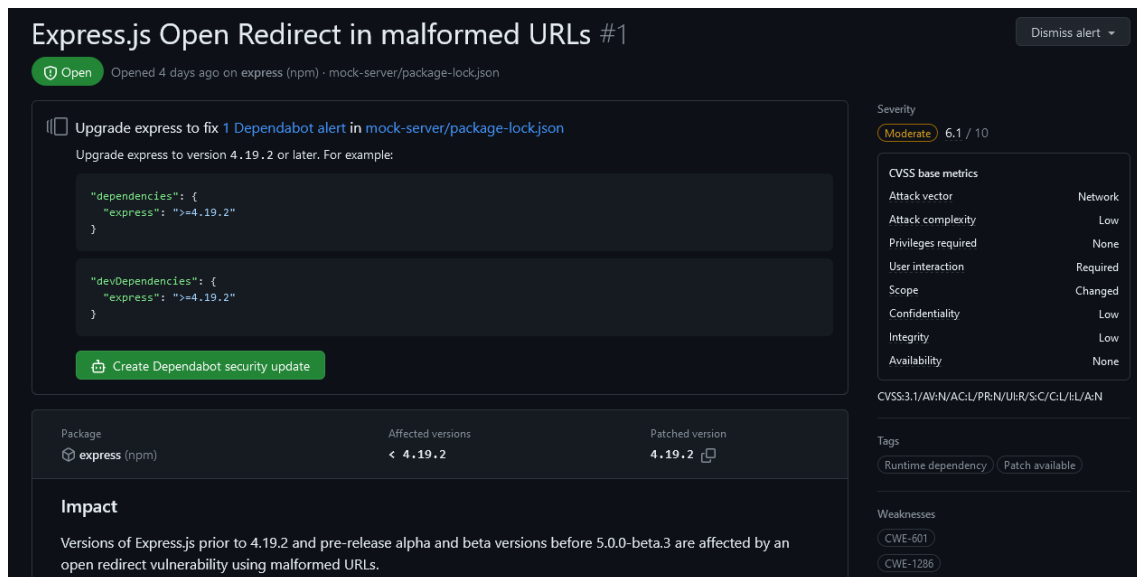


Figure 3.3: Dependabot Alert for a express in Project Repository

3.7 Git

When developing software projects, especially with multiple developers cooperating across devices, sharing your code over email is simply not sufficient. The industry standard for sharing, tracking, and organizing software project is Git, where nearly 95% of developers reporting it as their primary version control system [67]. Using Git and its features can range from simply tracking your own code changes locally, to tracking the code base of a large organization with automation and security for hundreds of developers. The Git configuration used in the project lays somewhere in the middle, where it utilizes multiple of the features Git offers to facilitate the development process.

3.7.1 Structure

There are no one size fits all solution for structuring a Git repository, however there are some common practices for all repositories, as well as some more specific to specific development environment. Common practices include to place `README.md` and `.gitignore` in the root [68], even though you can place `README.md` in the `/.github/` or `/docs/` as well [69]. The application developed in this project is built upon Svelte and SvelteKit. Some popular SvelteKit repositories include `awesome-sveltekit` [70], which has its SvelteKit project root in a `/site/` directory, and `svelte-commerce` [71], which has directories for `/docs/`, as well as having the SvelteKit project root in the root of their repository. The structuring of the Git repository for the project is inspired by these repositories and common practices, where it takes particular inspiration from placing the SvelteKit application in its own sub directory.

The Git repository structure seen in Figure 3.4 has multiple directories in the root. These are used to clearly separate the different logical parts of the project. The directory `/client/` contains all of the client side code, which is the SvelteKit project containing the whole front-end web application. `/docs/` contains documentation related to the project, both for developers and users. `/mock-server/` contains an Express server for mocking services when testing the SvelteKit application in `/client/`. Moreover, `/proxy/` contains a Nginx reverse proxy, also used when testing the SvelteKit application. There is also a `/scripts/` directory for bash scripts used for deployment. Finally there is the `/server/` directory, which stores the Nginx web server configuration, which is what serves the SvelteKit application as well as custom error pages when deploying the web application.

Common Git files are placed in the root of the project. This includes `.gitignore`, `LICENSE`, and the `README.md`. The Dockerfile of the SvelteKit application, as well as a Docker Compose for testing, are also placed in the root of the project. There are also Dockerfiles for the mock-server and reverse proxy, placed in the directories `/mock-server/` and `/proxy/` respectively. Workflow are placed under `/.github/workflows/`, as specified by GitHub[72]

```
interactive-database-for-environmental-data/
├── .github/
│   └── workflows/
├── client/
├── docs/
├── mock-server/
├── proxy/
├── scripts/
├── server/
├── .gitignore
├── Dockerfile
├── LICENSE
├── README.md
└── docker-compose.yml
```

Figure 3.4: Git repository structure

3.7.2 Gitignore

Several files in the local Git repositories should not be tracked by Git, which is accomplished by using `.gitignore`. Atlassian recommends ignoring dependency caches, build files, runtime files, as well as hidden system or IDE files [73]. The `.gitignore` in our repository therefore ignores any dependencies from Node, build directories from Svelte, logs, runtime data, and SvelteKit and Vscode hidden files. Furthermore, it also ignores `.env` files, as these can contain secrets not intended to be shared with the public.

3.7.3 Github Actions and Workflows

For automating processes on Github, such as testing and deployment, the Git repository includes workflow files which are ran by Github Actions [49]. These are used to automate the running of unit and end-to-end tests, linting tests, and security scans, when triggered by pre-defined conditions. The workflows are also used for continuously deploying the project to development test servers. These workflows are covered more in depth in Section 8.4.

3.7.4 Commits

To ensure efficient and well documented commits, several practices are followed. This includes following a modified version of conventional commits for the commit messages. Basic conventional commit messages follows the format `<type>: <description>`, for example `feat: add graph button` [74]. To clearly document which issue each commit is for, the modified structure of conventional commits include the Jira issue number, resulting in the format `<issuenumber> - <type>: <summary>`, for example `10 - feat: add graph button`. The commits in the project also follows several practices outlined by Aleksandr Hovhannisyan [75]. These suggest that each commit should be atomic, containing one logical unit of change. These practices ensure that the commits are easy to track and revert, and enables other developers to quickly comprehend the purpose and content of each commit. Furthermore, using atomic commits encourages the splitting of complex tasks into smaller, more manageable segments.

3.7.5 Branches

To separate between different working environments, both between the different developers, as well as the different features being developed, the Git repository utilizes branches. Branches are simply diverging paths from the main line of development [76], where code can be worked on in isolation until it should be moved back into the main line.

The project uses a custom approach for managing these branches, inspired by other solutions. An old, but still relevant model is *A successful Git branching model* [77]. It proposes using branches where you have a master branch for production ready code, and a developer branch with the latest development changes, both with an infinite lifetime. These should have supporting branches, with a limited lifetime, which should be used for developing new features, releases, and hotfixes. Another more modern solution is *Github Flow* [78], which proposes simply having one default branch, where you can create branches for each change you want

to make in the default branch. When a change is completed, it is reviewed in a pull request before being merged back into the default branch.

The custom approach used in the project incorporates concepts from both *A successful Git branching model* and *Github Flow*. There are two permanent branches, `main` and `developer`, along with feature branches which branch out from the `developer` branch, illustrated in Figure 3.5.

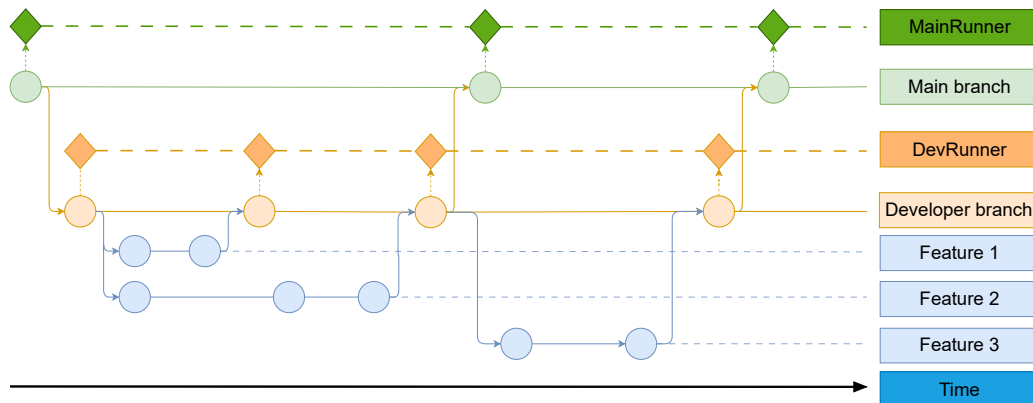


Figure 3.5: Gitflow diagram

The `main` branch contains stable code, where all the features are compatible. The `main` branch is updated after each sprint by merging it with the `developer` branch, and it is here the production ready and finished software will be. A server named *MainRunner* always runs and serves the version of the software which exists on the `main` branch. The `developer` branch contains code with the latest features, and is updated almost daily, after each new feature is merged into the branch. There is also a server running with the code from the `developer` branch called *DevRunner*. Both of these branches have infinite lifetimes and are protected by branch protections on GitHub [79]. Inspired by *GitHub flow*, any code changes to these branches requires the creation of a pull request [80], where the code being merged has to pass automatic checks, such as unit tests and linting, in addition to needing an approval and review of at least one other developer. This ensures that the code on these branches always meets a predefined code quality standard.

The feature branches are where developers can work on isolated features, without having to update the code in the `developer` branch directly. These have a limited lifetime, often one day to a week, where they are deleted after being merged into the `developer` branch. Limiting their lifetime is crucial, as the risk of complex merge conflicts rise the longer a branch is isolated without merging. Each feature branch is linked to an issue in the scrum backlog, which is also tracked on Jira (see Section 3.2), ensuring that each branch is associated with a uniquely identifiable issue number.

Naming the feature branches is accomplished by following a custom naming convention. The naming convention is inspired by the Git branch naming convention `<worktype>/<2-3 word summary>/issue-number` [81]. This is modified for the project to be as follows `<worktype>/<issue-number>-<2-3 word summary>`, for example `feature/10-navigation-buttons`. This

helps with keeping track of which issues in Jira correlates to each branch, as well as making it clear what their purpose is.

3.8 Quality Gates

Quality gates are a modern software development methodology with a set of criteria or quality standards that should be enforced throughout development process. This approach allows the team to evaluate if a code components meets the required standard before moving to the next feature. The criteria implemented in the project involve checking the code quality and its security.

These criteria are enforced each time a pull request is issued for a new feature. To add the feature to the application, it has to pass tests and a code review by another developer. These tests include unit tests, end-to-end tests, security scans, and linting. By enforcing the tests to pass and another developer to approve new features, the quality of the code in the final application is kept to a high standard, acting as a quality gate.

3.9 Documentation

To support the development of the application it has extensive documentation, which is kept up to date as the application evolves. This makes development easier by ensuring that all developers have the same common understanding of how the application is supposed to work. Furthermore, it makes it easier for NINA to overtake the maintenance of the application, as they can easier understand its inner workings and its dependencies. Good documentation also helps regular users with using the application.

The documentation for the application is outlined in various parts of the thesis. This includes diagrams and figures shown to illustrate the application, and attachments to documentation such as requirements, risks assessments, and threat modeling. In addition there is documentation found in the project repository. This includes a `README.md` containing instructions of how to deploy and set up the application, and a user manual explaining how to use it. In addition there is documentation explaining the various endpoints the application will use, including PostgREST and the SQL Views as outlined in Section 6.2.1, Authentication for Django as seen in Section 7.1, and the upload endpoint for Django as seen in section 6.4.1.

3.10 Commenting of Code

Commenting of code is an essential part of developing software. Commenting, if done correctly, improves the readability of the code and helps collaboration between developers by enabling easier communication of code purpose and functionality, and makes debugging easier. This does not only help development of functionality, but also security, as code error and potential vulnerabilities becomes less likely. To maximise the benefits of commenting, the development makes use of the following practices.

3.10.1 Commenting Principles

To ensure that the styles of comments in a project is consistent, there should be a consensus among the developers of a standardized way of writing comments. This is the first among several rules outlined by MIT ([82]). MIT mention that it's important to also put emphasis on the code itself, as the code also is important for communicating the code functionality. For example, when naming variables or functions one should use specific names with no redundant information. The variable name `const o` is less descriptive than `const mapMarker`, as the latter tells what the variable defines. If the code clearly conveys its purpose, there is no need to add comments. For example in Listing 3.4, there is no need to specify that the Javascript map containing the count of each species is initialised as empty map, as this is clear in the code. The commenting principles the group followed are outlined in list below.

1. Use consistent coding and commenting style
2. Use clear, descriptive, and unambiguous names in code
3. Write comments which give context and explains choices, rather than explaining the code itself
4. Keep comments up to date

```
/**
 * Get the count of each species in the observations
 * @param {object[]} observations - An array of observations
 * @param {string[]} allSpecies - An array of all species to include in the
  data
 * @param {boolean} includeOthers - Whether to include the 'others' category in
  the data
 * @returns {Map<string, number>} - Map of each species with their count
 */
function getObservationSpeciesCount (observations, allSpecies, includeOthers) {
  const speciesCount = new Map()

  // Find and save the amount of fish for each species
  allSpecies.forEach(species => {
    const speciesObservations = observations.filter(
      observation => observation.species === species)

    const count = amountOfFishInObservations(speciesObservations)

    speciesCount.set(species, count)
  })

  // If 'others' should be included, find and save the amount of fish for all
  other species
  if (includeOthers) {
    const otherSpecies = observations.filter(
```

```
    observation => !allSpecies.includes(observation.species))

    const count = amountOfFishInObservations(otherSpecies)

    speciesCount.set('others', count)
  }

  return speciesCount
}
```

Listing 3.4: Example comments for the project in the function `getObservationSpeciesCount()`

3.10.2 JSDoc

JSDoc is a markup language for commenting JavaScript, specifically functions, classes, and modules. Following JSDoc ensures consistency and readability by specifying how you should write comments, which includes the use of tags.

The use of JSDoc is shown in Listing 3.4, where the datatype of the function parameters and return values are specified with tags and they are given a brief description. For example, the parameter `includeOthers` is specified as a boolean, and is given a description of what it does. Because JSDoc is machine readable, several IDEs, such as Visual Studio Code [83], can display the JSDoc comments when functions are called in other files. It's also possible to use JSDoc for automatically generating API documentation [84]. Lastly, as JavaScript does not have built-in type safety, having the ability to specify data types to other developers prevents potential type errors.

Chapter 4

Requirements

When developing a software for a company it is crucial to define requirements. This includes how the application should function, what features it should have, how it should respond to attacks, and how it should be deployed. Requirements therefore ensure that both the stakeholder and the developers agree upon both the behavior and security required for the application. Throughout this project requirements were defined and continuously updated to guide the development. The continuous improvements of requirements have contributed to the quality of the web application, ensuring that it meets NINA's expectations. This chapter will cover the methods and strategies applied to the requirement process for this project.

4.1 Requirement Engineering

The requirement engineering process conducted in this project is based on theory from Official (ISC)2 Guide to the CSSLP CBK [85]. This process ensures the quality of requirements, by outlining steps and principles to follow. By clearly defining requirements, possible misunderstanding are prevented which could result in missing features, increased cost of development, and client dissatisfaction.

Requirement engineering is divided into four steps; requirement gathering, requirement analysis, requirement specification and requirement verification. The whole requirement engineering process can be found in Appendix M, where this section outlines the process followed and uses examples from the actual requirement document.

Requirement engineering should not only define the functionality of the application, but also security. According to Mano Paul, the author of the Official (ISC)2 Guide to the CSSLP CBK book, security is often considered as additional expense, providing little value, that is time consuming to implement [85, p. 94]. However, he argues that defining security requirements is as important as having a blueprint when building a skyscraper, where it adds value and is critical for ensuring the reliability, resiliency, and recoverability of software [85, p. 95]. Therefore, the requirement process followed also includes core security requirements.

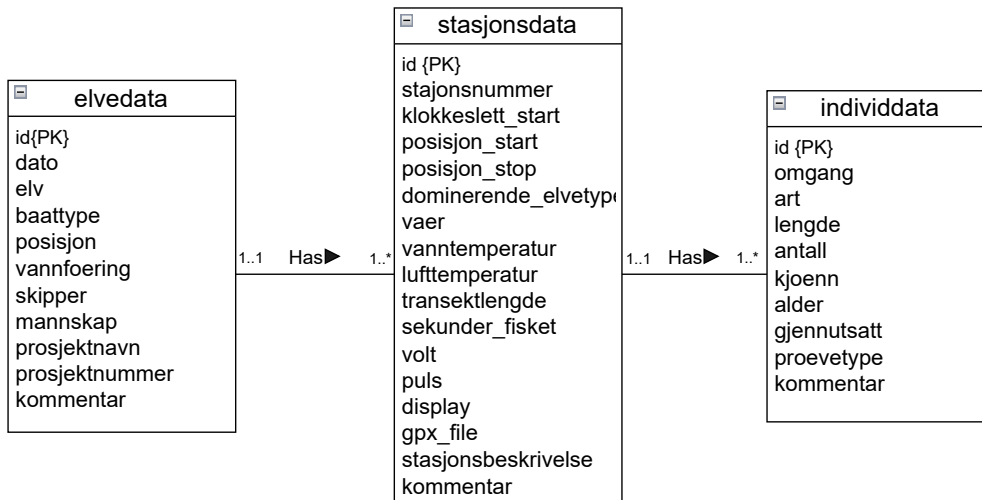


Figure 4.1: Class diagram of the environmental data

4.2 Requirement Gathering

The first phase in the requirement engineering process is requirement gathering with the overarching goal of understanding the business context, NINA's needs and requirements for the project. By engaging in discussion with NINA, we received a list of requirements outlining their requests for the application, meaning which features the website should include, and how the website should function when used. The following shows how the required documentation and information which is needed to later on define the requirements.

4.2.1 Business context and external requirements

Establishing business context and external requirements is important to do early in the software development in order to prevent breaking any laws or regulations, while at the same gain an understanding of the customer. This includes copyright laws and web accessibility guidelines, which are discussed in Appendix M.

4.2.2 Data classification

Data classification is about documenting and classifying the data the web application uses. This can help identify privacy protection requirements, but this is not relevant for the web application. This application will only process environmental data, with a structure we have created in collaboration with NINA as seen in Figure 4.1. The `elvedata` class represents an expedition to a river, where the scientist fished and collected environmental data. Each `elvedata` has multiple `stasjonsdata`, where each `stasjonsdata` is a singular trip with their boat at a specific time of day, with the start and stop positions. The `Individdata` class is the individual observations for each `stasjonsdata`, containing the species and length of the fish captured.

4.2.3 Subject-Object Matrix

A subject-object matrix is made to clearly envision allowable actions between subjects and objects. The subjects and objects are arranged into columns and rows, forming a two dimensional matrix. In this matrix the objects are the possible actions (listed across the columns) while the subjects are the different roles (listed down the rows). "Once a subject-object matrix is generated, by inverting the allowable actions captured in the subject-object matrix, one can determine threats, which in turn can be used to determine security requirements" [85, p. 140]. After discussing with NINA, it is agreed that the web application only has one privilege level when authenticated. This is illustrated in Table 4.1.

	Unauthenticated User	Authenticated User
View Data	No	Yes
Download Data	No	Yes
Upload New Data	No	Yes
Edit Data	No	Yes
Delete Data	No	Yes

Table 4.1: Subject-Object Matrix

4.2.4 Use Case and Misuse Case

In order to fully understand how the web application should function, behave, and to identify security threats, a use case and misuse case diagram is used. These diagrams contributed in the creation of specific requirements illustrated in Section 4.4.

Use cases are intended to give a systematic representation of the intended application functionality. It does this by modeling the interaction between the users and the application, and how the users can achieve their goals. Furthermore, the use case diagram visualizes the expected behavior and interaction with the application, which is useful when guiding the creation of requirements, and ensuring they are aligned with how the application should function. The use case diagram outlined for this project is illustrated in Figure 4.2. This diagram displays how an authenticated user interacts with the different sub-pages in the web application. Moreover, it illustrates how PostgREST extends data to the different sub-pages by the use of SQL views, while Django handles the authentication security mechanism. Each individual use case is described in more detail in the use case Appendix H.

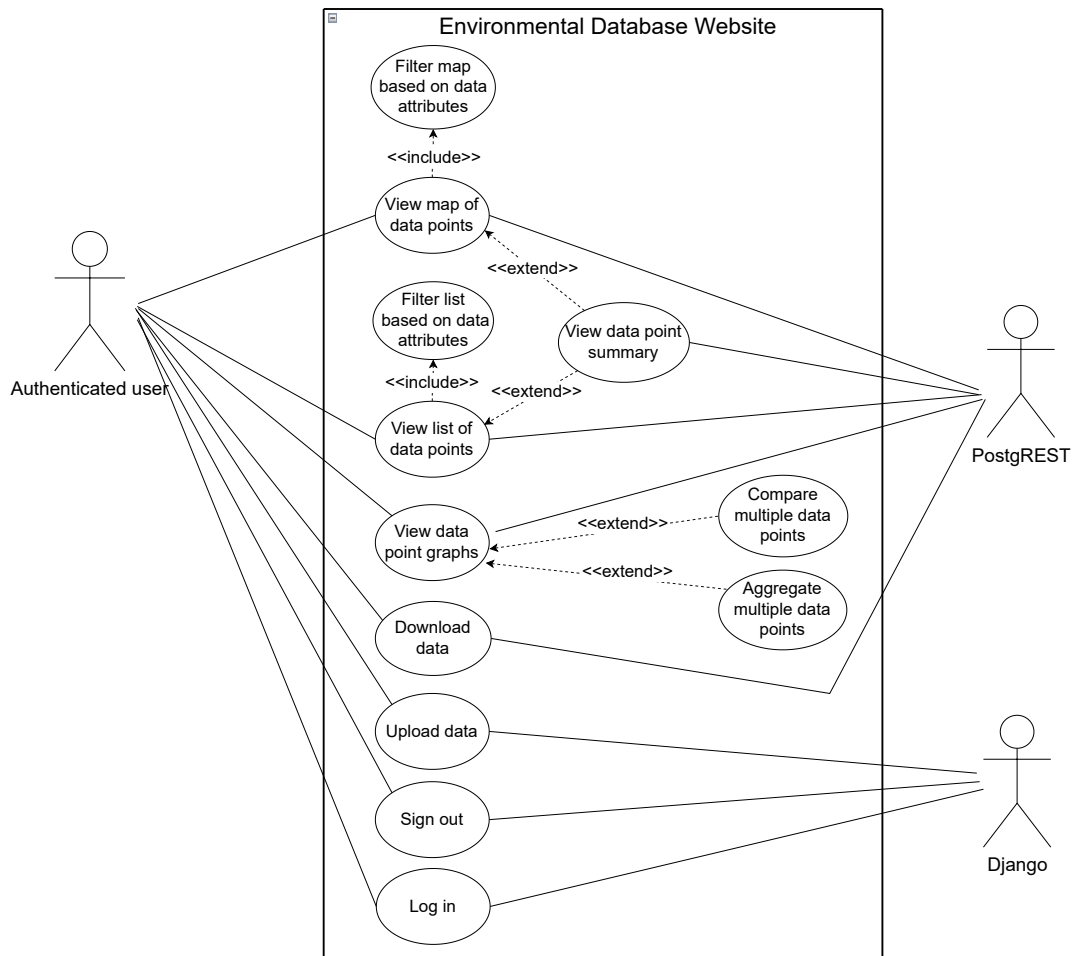


Figure 4.2: Use case diagram

Misuse cases, on the other hand, are outlined in order to provide a systematic representation of functionalities that the web application is not intended to have. The misuse cases outlined for this project models the interactions between threat actors and potential application vulnerabilities, providing potential ways of exploiting the application. Based on this the risk assessment in Section 3.5 was created including prioritized mitigation strategies and countermeasures. Having an overview over possible threats and countermeasures can contribute to guide the definition of security requirements. The misuse case diagram outlined for this project can be seen in Figure 4.3. This diagram shows potential vulnerabilities for the web application like injection vulnerabilities, compromised third party component and denial of service (DOS). Additionally, it shows potential threat actors like external attackers and internal hackers, and how they would use different vulnerabilities to threaten different sub-pages. Each individual misuse case is described in more detail in the misuse case Appendix H.

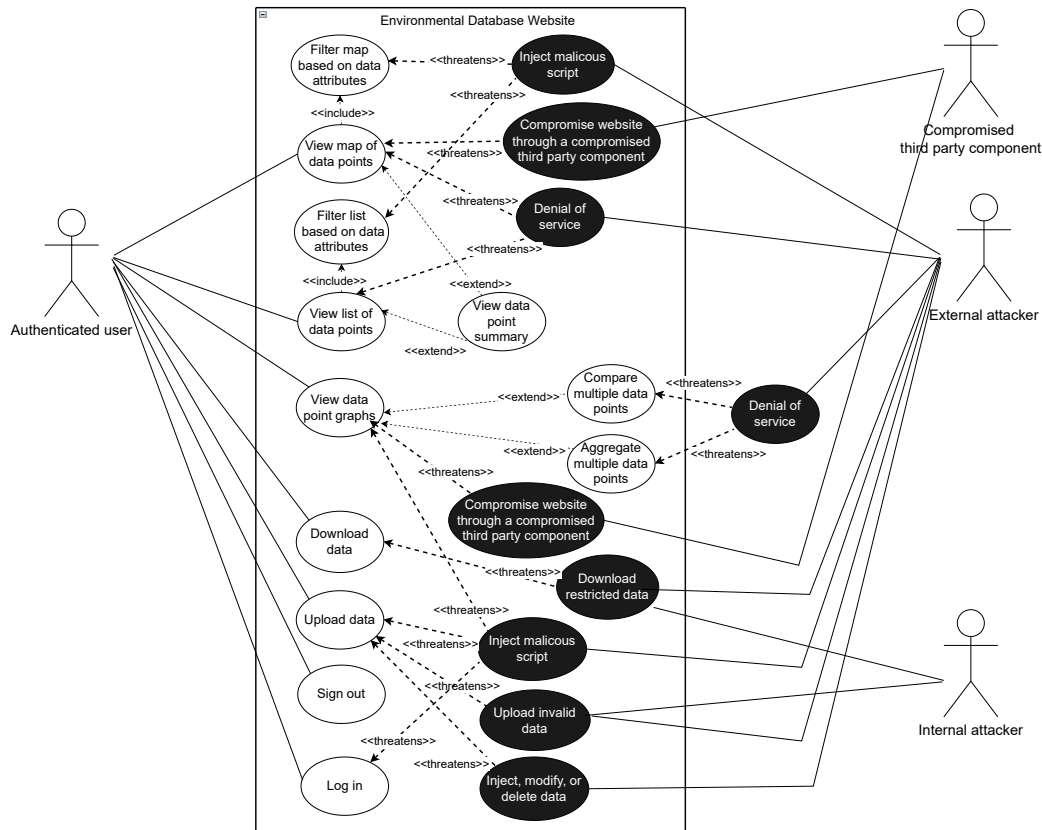


Figure 4.3: Misuse case diagram

4.3 Requirement Analysis

After gathering requirements, these needs to be analyzed, refined, and organized, in order to make them easier to understand. According to Maryland Department of Information Technology, each requirement analyzed should adhere to the principles; correct, unambiguous, complete, consistent, ranked, actionable and testable [86, p. 1]. Applying these principles to the requirement process will lead to less misunderstandings for both the developers and stakeholders. Furthermore, as a part of the requirement analysis, it is also important to identify constraints and dependencies that may have an impact on the project’s implementation, where the most important constraint of the project is the short time frame.

4.3.1 Requirement Principles

- **Correct:** Each requirement must accurately describe every necessary function and behavior intended by the stakeholders. In order to preserve correctness, every requirement stated is one that the software shall meet.

- **Unambiguous:** Each requirement must be clear with only one possible interpretation, avoiding any misunderstandings. Additionally, each characteristic of the system should only be described using a single unique term.
- **Complete:** Each requirement must be complete, meaning every significant requirement related to the web applications functionality or behavior must be represented.
- **Consistent:** Each requirement must provide every detail needed for the implementation, without the need for looking at external resources. Additionally, there should not exist any subset of individual requirements described that conflicts.
- **Ranked:** Each requirement must be prioritized by the stakeholders to insinuate their importance and urgency, guiding the development and planning process.
- **Actionable:** Each requirement must be realizable and feasible so that developers are allowed to implement it using existing tools, libraries, or techniques.
- **Testable:** Each requirement must be constructed so that it allows to be verified through testing, which ensures that the requirements meet its specified criteria.

4.3.2 How the Principles Were Applied in the Project

The principles in Section 4.3.1 are applied to the project as a part of the requirement analysis process. By engaging in discussions with NINA the group ensures that the requirements are correct, following the correct principle. Additionally, this is also verified by conducting user tests of both the mockup, and the web application, as discussed in Section 8.7. Furthermore, NINA explicitly redefined the requirements and ranked them based on what should take prioritization, ensuring that the principles are complete and consistent.

The use of the principles is illustrated in the example requirement from the project listed below. This requirement describes the expected action when clicking a river data point on the map or the list sub-page. It is clear and precise in its explanation, leaving no room for interpretation, following the principles unambiguous, complete, and consistent. Furthermore, the following list also include a measurement of the importance, where it is set as high as it is a critical feature for the application, following the ranked principle. Moreover, it is possible to check if the feature is implemented, following testable, and it is considered possible to implement, following actionable.

1. When clicking on a river data point on the map or list the user should be able to view tables containing a summary of the entire examination.
 - a. The observation should include:
 - i. General information, which includes River name, Project number, and the Date of the observation
 - ii. More specific information, which includes the amount of stations, Crew, Boat type, Water flow, and Comments
 - iii. Information of the underlying data, which includes total amount of fish found, amount of minutes fished, and amount of fish fished per minute.
 - iv. A table containing all of the underlying attached stations, and for each station the table contains information about the river-types, weather, time spent fishing, the amount of fish and how many fished fish per minute.

- v. A table containing the fish data of all underlying attached stations, and for each species the table contains information about the amount, amount per minute, average length, median length, and min and max length.
- b. Importance: High

4.4 Requirement Specification

Requirement specification is about documenting the gathered requirements, which includes the functional, non-functional and core security requirements for the software, as well as the constraints that governs its development. The specification of requirements serves as a contract between NINA and the developer team, guiding the development, validation, and testing of the application. The following lists describes some of the specification of functional, non-functional and core security requirements for this project. To view all the requirements, see the Appendix M.

4.4.1 Functional Requirements

The functional requirements describe how the application is supposed to function in order to fulfill the needs of the stakeholders. These requirements explain in detail how the system should react under specific conditions and how interactions should function.

1. The application will allow users to view and interact with a map focused on Norway, however the map will include the whole world. This map contains points marking each observation registered in the back-end database.
 - a. The user should be able to filter the observations shown by selecting the date range, species, and the type of data (river/station level) of the observations they want shown. They should also be allowed to select multiple options simultaneously.
 - b. Observations on station level should be displayed as two points connected with a line, illustrating the path the NINA took when performing the observation. The lines should only be visible after zooming in a certain amount on the station data points to avoid clutter. Observations on river level should be displayed as points at the coordinates the observation was taken. Both river level points and station level points will have it's dedicated color to distinguish them from each other. River data points on the map will be blue, station data points will be red, and selected data points (river/station level) will be orange.
 - c. Importance: High
2. The application will allow users to view a list containing all river and station level observations registered in the back-end database.
 - a. The user should be able to filter the observations shown in the list by selecting the date range, species, and the type of data (river/station level) of the observations they want shown, as well as searching for observations based on the name and date.
 - b. Importance: Medium

3. When clicking on a river data point on the map or an entry in the list, the user should be able to view tables containing a summary of its environmental data.
 - a. ...
4. When clicking on a station data point the user should be able to view tables containing a summary of that station.
 - a. ...
5. The user should be able to download data as either a XLSX or CSV file.
 - a. ...
6. The user should be able to select one or multiple river or station data points and view their data for more in depth analysis. The in depth analysis will contain graphs showing relevant data about the individual data under the rivers or stations.
 - a. ...
7. Authenticated users will be able to upload data to NINA's back-end database. The uploaded data must follow a specified XLSX format, and not contain any malicious content. The user will be informed of the result of the operation, and refreshing the page should allow the user to see the newly added changes if the operation was successful.
 - a. ...
8. The user should be able to log in if they are not authorized by inputting their username and password. They can also log out by using a log out button.
 - a. ...

4.4.2 Non-functional Requirements

The non-functional requirements are requirements not related to the functionality of the application itself, instead it is about specifying the implementation of the application. These requirements aren't ranked or organized. The reason for this, is that they together define the framework the project and application must follow, meaning all are equally important.

- The application must be capable of running in a Docker container.
- The project must implement version pinning for all libraries, frameworks frameworks, and dependencies. This is done to ensure consistent and reproducible builds.
- The application will communicate with NINA's back-end database through a PostgREST API.
- The project must use libraries and programming languages which are open source. These languages and libraries must be actively maintained and be approved by credible sources as safe to use.
- All code has to be stored in GitHub. Necessary documentation of the application and dependencies must also be stored on GitHub.
- The GitHub project must have an MIT license. This allows others to use, modify or distribute the project, as long as they give credit to the creators, including the original copyright notice.

- Any dependencies must be clearly documented. This is essential for transparency, ease of maintenance, and collaboration, since it provides a clear understanding of the software's external components, which enables efficient troubleshooting, awareness of needed updates, or replacement of dangerous or outdated dependencies.
- The project must be finished before 21.05.2024. This means the final version of the application must also be finished before this deadline, however this does not restrict NINA from updating the application after the deadline if they need to incorporate new features.

4.4.3 Core Security Requirements

The core security requirements are measures added in order to protect a system from attacks, data breaches and unauthorized access. When defining core security requirements, it's important that they ensure confidentiality, integrity and availability of the system and its corresponding data. This results in security being considered from the start of the development, which makes it easier to ensure that satisfactory security is achieved in the final product. "Properly and adequately defining and documenting security requirements, makes the measurement of security objectives or goals, once the software is ready for release or accepted for deployment, possible and easy" Paul [85, p. 98-99].

In the List 4.4.3, the core security requirements are illustrated. These are grouped in categories corresponding with the CIA triad (Confidentiality, Integrity and Availability) and is expanded to include other core requirements categories, including Authentication, Authorization and Accountability [87]. There are also security requirements not categorized, and some which falls on NINA's responsibility. These are clearly separated in their own categories.

1. Confidentiality:

- a. **Error Handling:** Implement proper error handling by handling all errors explicitly, displaying error messages in a non-verbose nature to the user. Should apply the principle fail securely, where the application always defaults to a secure state.
- b. ...

2. Integrity:

- a. **Secure Supply Chain:** Ensure that all dependencies are open source and from trusted sources to maintain a secure supply chain. Set up automatic vulnerability scans of dependencies.
- b. **Injection prevention:** Apply techniques to prevent injection, where dataformat, data types, and content are verified. Content is checked by only allowing whitelisted characters.
- c. ...

3. Availability:

- a. **Caching of Requests:** Cache data retrieved from the database on the client side to reduce the amount of data requests.
- b. ...

4. Authentication

- a. **User Authenticated for accessing environmental data:** Accessing environmental data, downloading data, or uploading data should only be allowed by authenticated users.
- 5. Authorization:
 - a. **Least Privilege Access Control:** Implement and apply the principle of least privilege for all system and user operations. Users should only be allowed to read and append environmental data, not modify or delete it.
- 6. Accountability
 - a. All requirements regarding accountability and non-repudiation will be administered by NINA.
- 7. Others
 - a. **Follow Copyright:** Copyright should be followed where relevant.
 - b. ...
- 8. NINA's responsibility
 - a. **Backup:** create an isolated backup of the environmental data.
 - b. **acrshortHTTPS:** Configure a proxy which handles HTTPS and SSL (Secure Sockets Layer) certificates for incoming requests
 - c. ...

4.5 Requirement Verification

Verification of requirements is something that is conducted continuously throughout the requirement engineering process. This involves making sure the requirements are up to date and updating them if any new requirements are decided upon. Additionally, the step also ensures that the requirements are achievable and can be implemented within the given time frame. This continuous verification process allowed NINA to request new features and modify existing ones while the application was being developed, and after conducting user tests, see Section 8.7. It also allowed the group members to drop or deprioritize certain features not achievable within the time frame. For instance, a feature that had to be dropped was allowing the user to flag data in the application for further review.

Chapter 5

Design

Design encapsulates the phase of the project that occurs before the development of the application. It builds upon the requirements previously defined in Chapter 4, and is used to guide the development and implementation of the application. This chapter covers the process involved in deciding how the application looks and the modeling of its architecture. Moreover, it also covers secure design principles which can help design secure software. Lastly, it looks at how threat modeling can be used to find risks associated with the design, which can be used to update the risk assessment. This ensures that threats and risks are addressed explicitly in the design process, which enables mitigations and countermeasures to be incorporated into the development process.

5.1 Software Architecture

The architecture of the software gives an overview of how the developed Svelte application works, as well as the network and infrastructure it runs on. Documenting this helps communicating the indented software design, ensuring that all developers have a shared understanding of the application. This makes the development more effective, in addition to making it easier to maintain and further develop the application.

5.1.1 Svelte Application

The component interaction diagram in Figure 5.1 is made to visualise the Svelte applications interaction with external components.

For a user to retrieve the svelte application, they use their web browser to send a request to the Nginx web server. This server then sends the static Svelte application. The river and station data used by the application is retrieved through communication with the PostgREST API. The application communicates with Django to authenticate users, which sets cookies and allows for uploading of files. For the application to function properly, it also needs to retrieve the map layers from external sources, either terrain, satellite, or topology.

The application utilizes a client-side hydration architecture, combined with being a single page application. Client-side hydration means the web browser loads the application as basic HTML, CSS, and JavaScript. JavaScript is then used to dynamically fill the content with river

and station data from PostgREST. Being a single page application means the entire application is loaded as a singular HTML page. Users can then navigate between different pages without having to reload the web application.

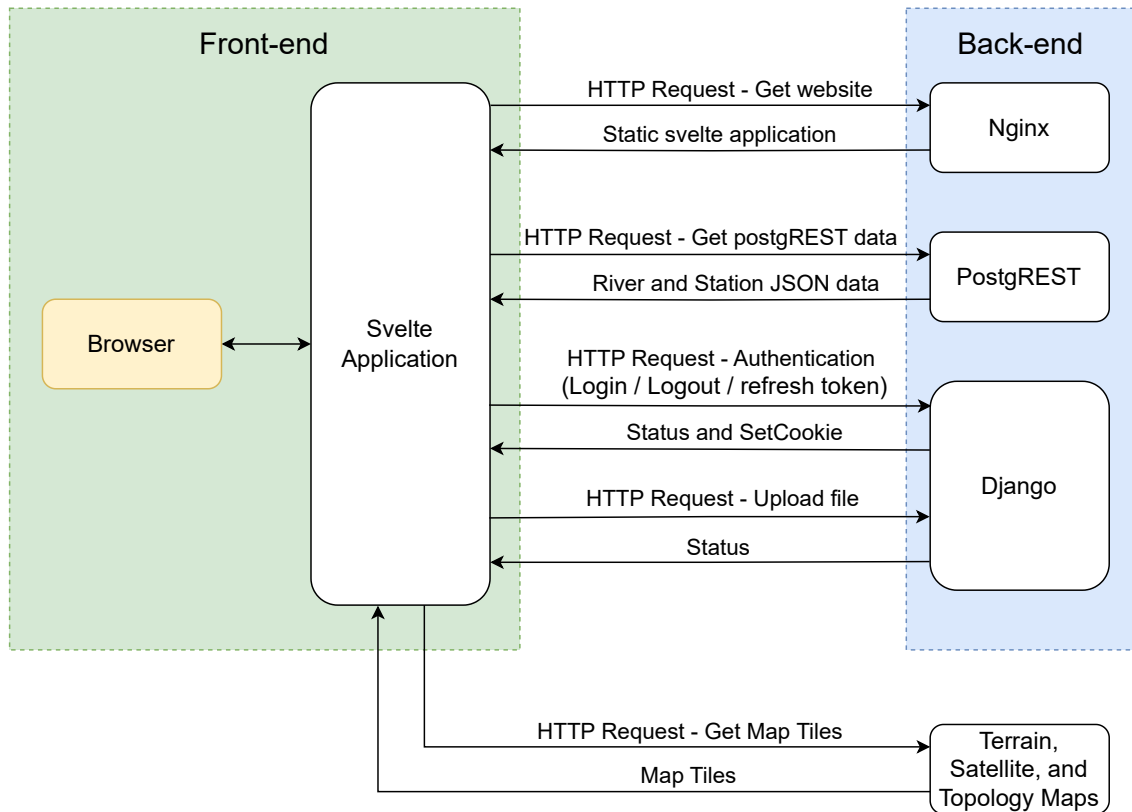


Figure 5.1: Component interaction diagram of the svelte application

5.1.2 NINA's Infrastructure

NINA's infrastructure is illustrated in Figure 5.2. The figure shows the network and infrastructure the application will be integrated with. It's important to model this to ensure that there is a consensus among developers of how the communication between the application and different parts of the infrastructure works. This is especially important when cooperating across different teams, in this project between the developers and NINA's IT department, to help the application seamlessly integrate without problems.

The web browser accessing the application will first be routed through a Nginx reverse proxy, as seen in Figure 5.2. The reverse proxy handles HTTPS and SSL certificates for all incoming traffic to NINA's servers. All traffic for our application is then routed to the Fishboat-db Docker compose stack. This is where the services the application is dependent on are running, including another Nginx reverse proxy, which in this case routes traffic to the correct services

based on their URL endpoints. There is a Nginx web server for the Svelte application, which is what we are developing. PostgreSQL stores the environmental data, where PostgREST creates RESTful API endpoints for accessing it. There is also a Django service. Django is responsible for handling authentication and setting cookies, and it communicates with NINA's internal LDAP server, which is used by NINA to authenticate users given a username and password. Furthermore, Django also exposes an endpoint for uploading files and data to the database.

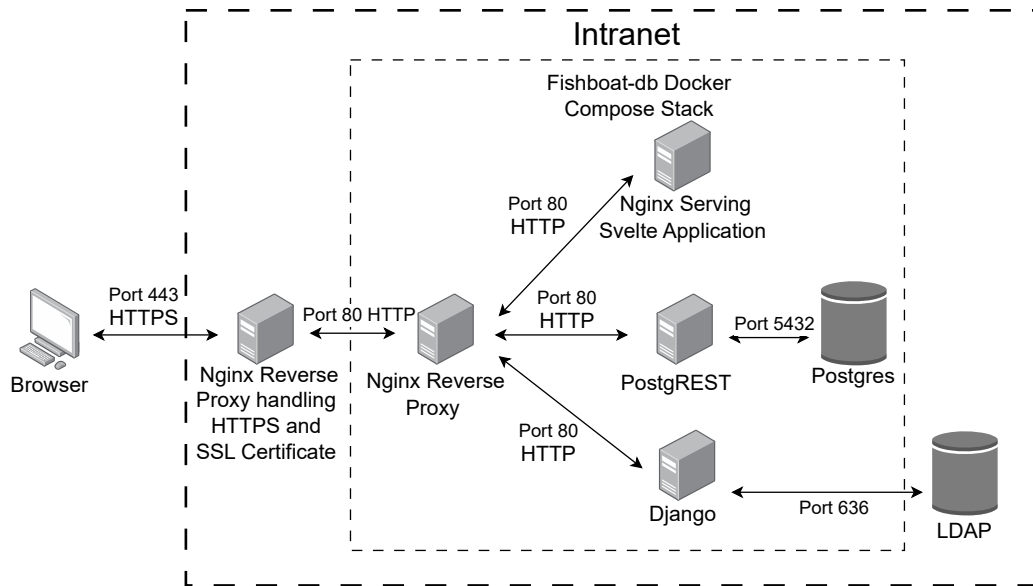


Figure 5.2: Network diagram of NINA's infrastructure

5.2 Design Principles

Design principles are concepts that help create a well-structured, maintainable, and scalable software. The principles contribute to protect the software against common vulnerabilities such as weak input validation and improper privilege management. All of these principles should be considered, but sometimes implementing one of these comes at the cost of others. The developers then have to make a trade-off between the different principles. These principles are outlined in and inspired by Official ISC2 Guide to the CSSLP CBK Second Edition [85, p. 180]. The following are the principles the application follows.

5.2.1 Least Privilege

The principle of least privilege specifies that the software should give the minimum level of access rights, for a minimum amount of time, to complete operations. In the application this is applied when fetching data from PostgREST, and when uploading files. When fetching data the user has read-only access for the time their token is valid. When uploading files the user

can append the files content to the server, if it passes validation, but they can not edit or delete existing data. In both cases the application ensures the user has the least privilege needed to complete the operations.

5.2.2 Separation of Duties

Separation of duties entail that there should be two or more conditions fulfilled before an observation can be completed. This is not implemented into the application, however it is used by the developers for the development process. Any code added to the main and developer branch needed a review from another member of the team, as specified in Section 3.7.5. This ensured that at least two people needed to approve new changes to the application.

5.2.3 Defense in Depth

Defense in depth, also referred to as layered defense, says that the breach of one vulnerability should not result in a complete compromise of the software. The software should never be entirely dependent on a singular layer of security. In the application, all input, including input from users, internal services, and third-party services, are validated. This reduces the chance of any malicious input affecting the application or the infrastructure. In addition to input validation, output is either encoded by Svelte or sanitized. This further reduces the risk associated with malicious input, as even if successfully inserted, there is still a layer of security preventing it from affecting users. The application also uses content security policy to set limitations on what is possible to run on the website, as well as storing cookies securely using HttpOnly and same-origin.

5.2.4 Fail Secure

Fail secure specifies that when software fails it should reliably function and quickly recover into a normal secure state. This maintains the resiliency of the application. It protects the application's availability by ensuring it quickly recovers from mistakes, confidentiality by ensuring no confidential information is leaked, and integrity by ensuring the application works as intended when it fails. All users of the application are denied access by default. This ensures crashing the authentication mechanism won't result in users getting automatically authenticated. All errors and exceptions are handled explicitly, where error messages are displayed in a non-verbose nature. This results in the application never leaking confidential information if any error should occur.

5.2.5 Economy of Mechanisms

Economy of mechanisms says that software should avoid implementing unnecessary features. Each added feature increases the attack surface and complexity of the application. Any features not specified in the requirements are therefore not implemented. At some point the ability to mark data used by the application was discussed. This would allow NINA to spot and mark mistakes in the data, and update the data stored in the back-end. The functionality wasn't

implemented since it could not be considered a required functionality, and it would have increased the attack surface.

5.2.6 Complete Mediation

Complete mediation entails that the software should never assume anything, and that access requests should be mediated every time. The application should always check if a user is authenticated for each request sent to the database, even if the a user has previously sent requests while being authenticated. Furthermore, the application never assumes that data from the database or users is valid, and always does validation of data.

5.2.7 Open Design

The principle of open design states that the exposure of a software's design should not compromise the protection provided by safeguards. Relying on security through obscurity is not advised, which is when the security of software is reliant on the attackers not knowing the system design and architecture. All source code for the application is stored in a public GitHub repository, available for everyone. The application uses proven, advised, and tested common security measures such as Cookies with `HttpOnly`, `SameSite=Strict`, and `Secure` for authentication and HTTPS for encryption (see Section 7). This results in the application's design and code not needing to be secret for its safeguards to be effective.

5.2.8 Psychological Acceptability

The principle of Psychological acceptability says that security mechanisms should not be too complex, but rather maximize usage and adoption by users. It implies having the best and most advanced security features won't matter if users can't be bothered to use them, because it makes the user experience noticeably worse. The application ensures this by using single sign on to let the user access all the functionality of the application if they have the required privileges. Furthermore, users only have to log in again after they have been inactive for an extended period, preventing them from constantly having to log in while using the application. Logging in regularly could have made users choose simpler passwords, posing a possible security risk. Moreover, for uploading data, the application gives users specific and detailed feedback if anything fails. This ensures that users are aware of any faults with their uploaded data, and lets them understand and fix the problems. Other error messages in the application also contains information about the problem, e.g. if some APIs were unavailable or if the user can't perform an action because they're not authenticated.

5.2.9 Leveraging Existing Components

The final principle the application explicitly follows is leveraging existing components. Leveraging existing components means utilizing already existing and approved libraries and security mechanisms. The application uses third-party libraries for validating text and JSON, for managing excel files, and for displaying maps and plots. Using existing libraries and security mechanisms saves time during development, as the code does not have to be implemented for

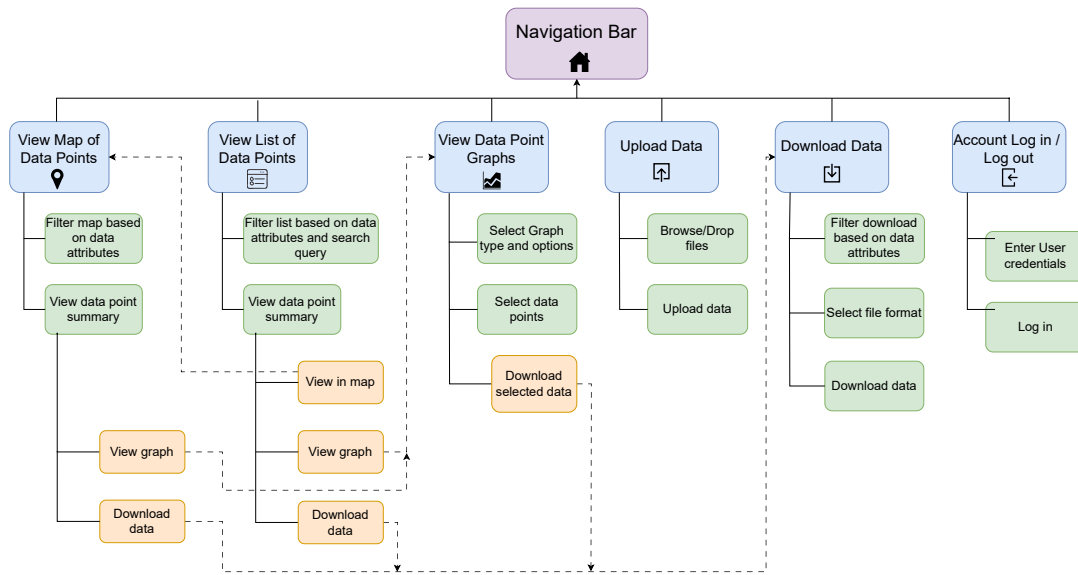


Figure 5.3: Sitemap of the application

these features by the developers. In addition, the chosen libraries are regularly updated and checked for security vulnerabilities increasing their reliability and security, which is discussed in Section 3.6.1.

5.3 Site Map

A sitemap can be used to paint a clear picture of all the different pages of an application [88]. It gives a bird’s eye view over the website’s structure, showing the key elements each page contains and how the pages are interconnected. This helps in gaining an understanding of the intended design of the application, making the process of developing the application easier.

The sitemap for the application contains lines, boxes and colours illustrating its structure, as seen in Figure 5.3. The purple box shows the navigation bar which is visible on all pages. Each page of the application are indicated with the blue boxes. User actions and input on the pages are represented by the green boxes. The orange boxes represent the buttons for cross-site navigation. The solid lines represent the structure of the website and how the features are implemented on the pages. The dashed lines represent the flow of the page when a button for cross-site navigation is clicked.

5.4 Visual Outlines

Visual outlines are a way to define how software will look before it is created. There are three stages of visual outlines; wireframes, mockups and a prototype [89]. These are an essential part of the development process for both user testing and for forming a clear picture of the

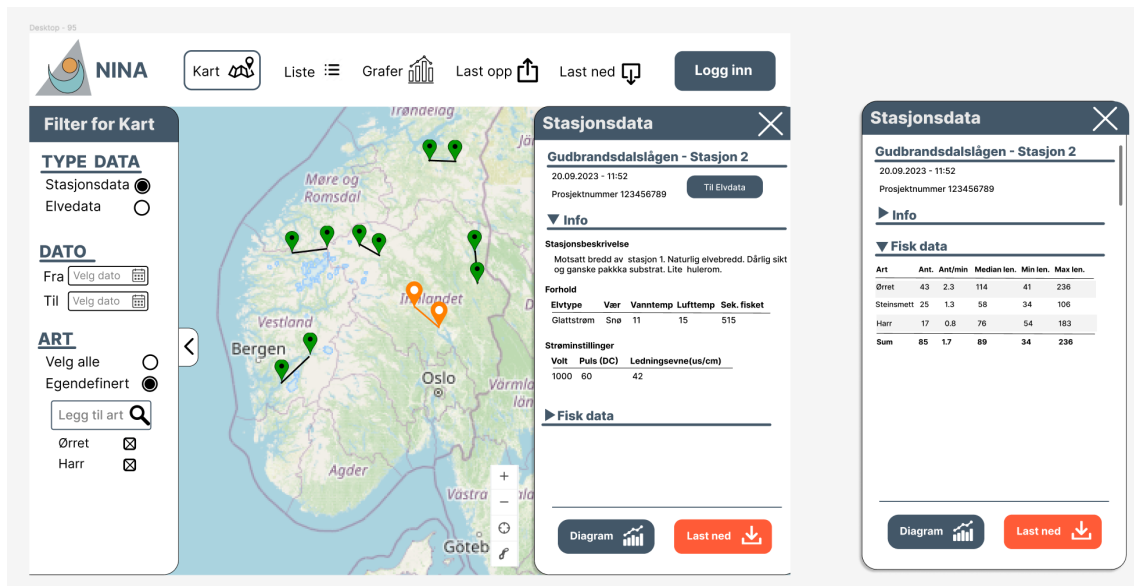


Figure 5.4: Part of the Final Mockup, showing the map page with a station selected

layout. In the project a mockup was primarily used to illustrate the intended design of the application.

5.4.1 Wireframe

The main purpose of the wireframe is to create a basic blueprint primarily referred to as the skeleton of the website. By visualising the structure, layout and the contents of the page, the developers can test different layouts and get feedback from the stakeholders. This stage was however skipped in the design phase, where a mockup was created instead.

5.4.2 Mockup

A mockup is visually closer to an actual prototype compared to a wireframe [89]. It includes all features present in a wireframe, along with styled buttons, navigation graphics and icons. Compared to a wireframe which is a blueprint of the website, a mockup is in addition a visual model that gives an accurate representation of the visual layout and design of the final product.

Creating a mockup made it possible to show NINA how we visualise the website to look based on their requests. This allowed the completion of a usability tests by NINA, where they provided valuable feedback for possible additional features and design considerations. The mockup was then modified based on the feedback. This is an advantage of using mockups, as it allows you to quickly modify the design of an application with little cost. The resulting mockup was then used as a guideline throughout the rest of the development. Figure 5.4 shows how one of the envisioned pages of the application looked in the mockup.

5.5 Threat Modeling

In the design process a threat model of the application design was conducted. The steps included in the process are; modeling application architecture, identify possible threats, identifying prioritizing and implementing controls, and finally documenting and validating the results. The result is an overview over possible threats, their severity level, and which safeguards are needed to protect against them. Threat modeling is large process, where this section only covers the outline of the process followed. A more comprehensive coverage of the threat model conducted for the application can be found in Appendix L.

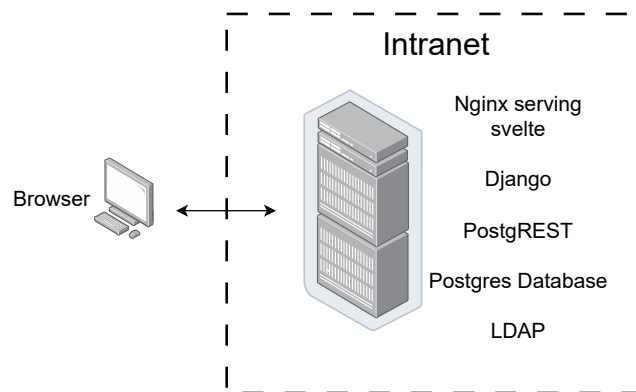


Figure 5.5: Physical topology of the application

5.5.1 Modeling Application

The modeling of the application architecture included modeling the physical and logical topology of the application, categorize human actors of the system, and define how they interact with the application. This can be seen in Figure 5.5 and Figure 5.6. The physical topology simply illustrates where things are located physically. The logical topology shows the actors and various services in the application, and how these interact with each other.

5.5.2 Identify Threats

The next step is to use these models to identify possible threats. This is accomplished by identifying possible trust boundaries, where the privilege level changes, for example between a web server and a database. The boundaries and the data-flow is then illustrated in a Data Flow Diagram (DFD), see Figure 5.7. Potential mis-actors are also identified in this stage, such as cyber criminals and state actors. Using the DFD and mis-actors, threats are created for each place data travels across trust boundaries. These are categorized by using STRIDE, which categorizes threats based on their goals. STRIDE stands for the goals Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege. These include Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of privilege. STRIDE and the categorization of threats are discussed further in Appendix L.

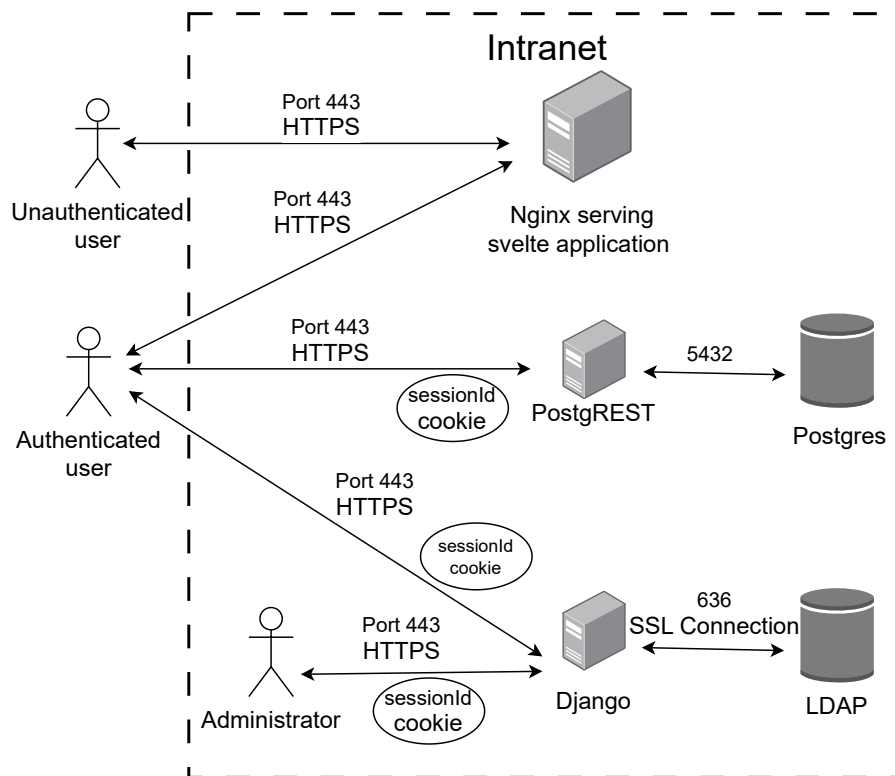


Figure 5.6: Logical topology of the application

5.5.3 Ranking Threats

After the threats are identified, they are ranked using DREAD. This includes specifying their Damage potential, Reproducibility, Exploitability, Affected Users, and Discoverability, see Appendix L. The threats found are then documented, relevant countermeasures are found, and the threats and countermeasure are used to update the risk assessment. A threat identified in the threat model is *Spoofing a file*, where an attacker could try to upload a file with an invalid extension and malicious content to the server. This is reflected in the risk assessment, where it considers risk associated with lack of input validation of files, where input validation of these files is recommended as a countermeasure among others. The threat model therefore results in the risk assessment being updated based on the design of the application.

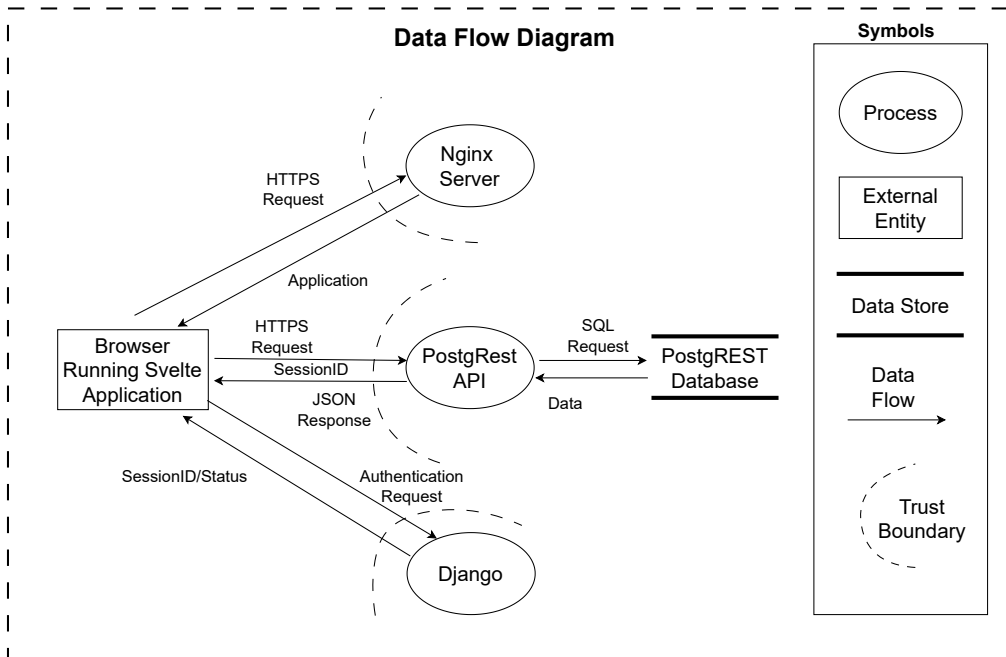


Figure 5.7: Dataflow diagram of the application

Chapter 6

Functionality Implementation

To implement the front-end web applications intended functionality, as outlined in the requirement phase (see Chapter 4), the application is mostly written in JavaScript, with some HTML and CSS. The implementation of the application also includes following the design outlined in the design Chapter 5. The application uses Svelte and SvelteKit as its framework, along with multiple external libraries for various functions, such as Leaflet for displaying a map, and Plotly for plotting graphs. This chapter covers how the web applications functionality is implemented in code using these frameworks and libraries. It looks at how the code is structured, how the environmental data is retrieved, stored, and processed, and how it is displayed using modules and libraries. The chapter will also look at how files are handled for upload and download, and how navigation is implemented with SvelteKit.

For a more detailed coverage of the code in the application see Appendix O. It covers the same functionality as this chapter, but goes more in depth when explaining code, and shows more comprehensive code examples.

6.1 Code Structure

When developing software it is important to consider the structure of the code. By setting up the file structure in a modular fashion, you can ensure separation of concern. This helps the maintainability of your code, as you can easily change one part without affecting the rest, and multiple developers can work on different modules simultaneously. Moreover, following separation of concern makes the code reusable, as code written in one module can be used multiple times across the project, independent of how it originally was implemented. The following is how the application's code is structured, which attempts to follow separation of concerns.

6.1.1 Svelte and SvelteKit Structure

As the application uses SvelteKit, it follows the project structure outlined by its official documentation [90]. The project structure (see Figure 6.1) includes a folder for source code, called `src/`, a folder for static assets called `static/`, and a folder for end-to-end tests called `tests/`. Configuration files is placed in the root of the SvelteKit folder. Furthermore, under `src/`, the

application is further split into the folder `lib/` for Svelte components and `routes/` for SvelteKit pages. This structure follows separation of concern, as different parts of the application are clearly separated.

6.1.2 JavaScript Modules

The application further splits the code into different JavaScript modules. Each of these modules has a single responsibility, where the modules are organized into different directories under `src/` (see Figure 6.2). This ensuring high cohesion, which is a principle that states that all code inside a module should together perform a single and well-defined purpose [91]. There are multiple high level categories of modules, stored in different directories. The directory `api/` is for modules interacting with external components and services, `constants/` is for hard-coded values which is used across the application, `models/` is for defining data structure and classes, `stores/` is for declaration of svelte data stores, and `utils/` is for all other miscellaneous modules.

By structuring the code this way the `plotlyData` module can call a function from `api/postgrest`, which again can retrieve a value from `constants/endpoints`, where each module does not depend on the different implementations. This follows loose coupling, where each module does not care about how the others are implemented making the modules less dependent on each other [92].

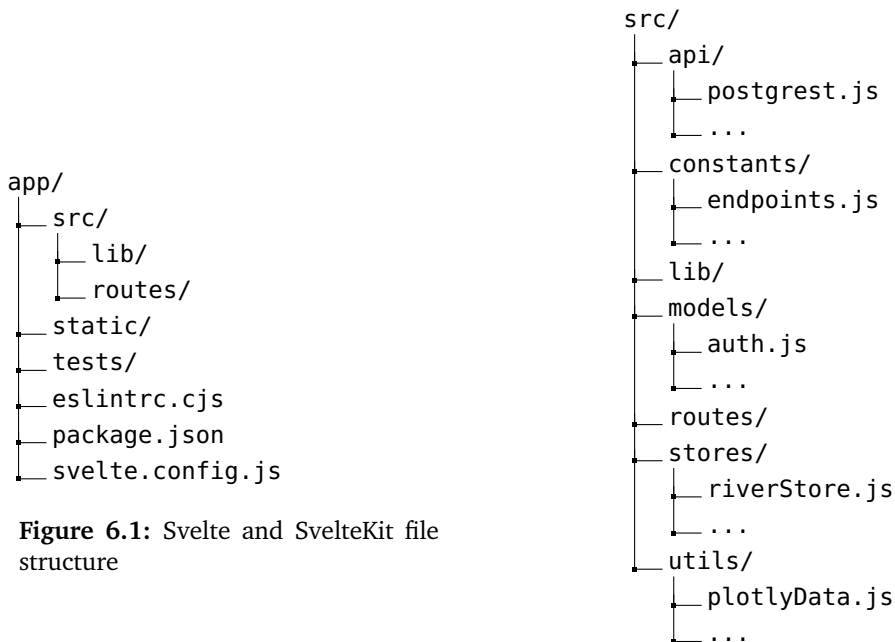


Figure 6.1: Svelte and SvelteKit file structure

Figure 6.2: JavaScript modules file structure

6.1.3 Svelte Components

The application further splits up the code into different Svelte components [11]. These are the building block of a Svelte application, where each component is its own `.svelte` file. Each component contains HTML, CSS, and JavaScript which only applies to the specific component. This follows separation of concern, with each component containing only code directly related to it, allowing it to be used in other parts of the application without having to consider its implementation.

An example component from the application is `DateInput.svelte` (see Listing 6.1). This component is only responsible for an input field where a user can select a start and end date. It has JavaScript which simply lets other components read and set the dates selected, HTML for basic date inputs, and some simple styling. Components can grow to be larger, and they can use other components themselves, however they are always structured to contain relevant code for the scope of the component.

```
<script>
  export let selectedStartDate
  export let selectedEndDate
</script>

<!-- Input for choosing start date -->
<label for='startDate'>
  Fra
  <input id='startDate' type='date' name='startDate' bind:value={
    selectedStartDate}/>
</label>
<!-- Input for choosing end date -->
<label for='endDate'>
  Til
  <input id='endDate' type='date' name='endDate' bind:value={selectedEndDate}/>
</label>

<style>
  label {
    display: block;
    padding: 0.5em;
    font-size: 1.2rem;
  }
</style>
```

Listing 6.1: DateInput component

To send values between components, Svelte uses props. This allows components to expose variables they want other component to read and set, with `export let variableName`. Other components can then update this variable by using the syntax `propName={value}`. When using

a component, you can also specify how events should be handled, which can be done with the syntax `on:eventname={handleFunction}`, where the event from a component will get sent to the function specified.

6.2 Retrieval of Data

As the application's main purpose is to display environmental data, retrieving this data in an efficient way is essential. The environmental data grows continuously as the scientists collect more, and there are already several hundred rivers, thousands of stations, and hundred of thousands of observations, which can grow ten fold, (see NINA meeting 28.02.2024 in Appendix P). Therefore the application is designed to use client-side hydration (see Section 5.1.1), where the data for the application is only fetched at the client-side. In addition, the application leverages lazy-loading, which is the process of waiting to load certain parts of the application data until they are needed, instead of loading everything at one. The following is how the application retrieves and stores this data.

6.2.1 PostgREST API

PostgREST is used as an API which exposes the environmental data the application is allowed to use. To configure the endpoints PostgREST expose one can create SQL Views in PostgreSQL. These are used to create custom endpoints which contain the exact data needed for different purposes in the application.

SQL Views

There are five SQL views used by the application. Two of them are used to fetch all rivers and all stations, called `river_with_species` and `station_with_species` respectively. As all rivers and stations are fetched on page load, these SQL views contain the minimal amount of information needed when displaying them, to minimize the data sent. The other three SQL views are only fetched when a specific river or station is selected. Two are used to get the data needed to show river and station summaries or plot their data, called `river_summary` and `station_summary`. The final one is used to fetch the data needed for a station to be downloaded, called `station_download`. How the SQL Views were configured can be seen in the code implementation Appendix O.

Endpoints

Accessing the SQL views is accomplished by sending a HTTP GET request with the correct path. Moreover, PostgREST also supports the filtering of data in endpoints [93]. For example, if you want to access `station_summary` for the stations with ID 1 and 2, the query parameter ID can be set as follows: `ID=in.(1, 2)`, as seen in Listing 6.2. These endpoints return the data as JSON.

```
GET /station_summary?id=in.<id1>, <id2>, ...)
```

Listing 6.2: Request syntax for `station_summary` endpoint

6.2.2 Fetching PostgREST with PostgREST Module

To fetch the data from PostgREST the application uses functions defined in a custom `postgrest` module. It has a function for each endpoint/SQL view, where the summary and download endpoints take the ID or ID's of the river or stations to fetch. By calling these, for example `fetchRiverSummary(id)`, a HTTP GET request is sent to the correct endpoint with the correct query parameter.

The code which creates the fetch request is illustrated in Listing 6.3. It uses `POSTGREST_URL` from an environment variable, along with the endpoint which is different based on the type of data to request. The code specifies the use of the GET method, and that all cookies for the domain should be included in the request if the URL of the website has the same origin as the request. The origin is the same if the protocol, domain, and port number is the same [94]. The response from the endpoint is handled by checking the status codes, where 401 (Not authenticated) is handled by the custom auth module, and if the status is not OK (not in the 200 range) it throws an error and displays it to the user. If the status is 204 no content, NULL is returned, and if everything worked the JSON data is parsed and then returned.

```
const response = await fetch(`${POSTGREST_URL}${endpoint}`, {
  method: 'GET',
  credentials: 'same-origin'
})
```

Listing 6.3: Fetch request to PostgREST endpoint

6.2.3 The DataManager Module, caching, and updating the stores

The data manager is called from the components and pages to ensure that the data for rivers and stations is available. It has functions for each endpoint, where the functions first check if the data already exists, then fetches the data using the `postgrest` module, validates it, and updates the storage in the application with the new data. The pages in the application can use these functions when they need specific data. For example, when the map page loads, it needs all rivers and species for displaying them in the map. To accomplish this the map page simply have to call the `getRivers()` and `getStations()` functions, allowing the map page to then find these in the global storage. Another example is when a user want to download a station. The download page calls `getStationForDownload(id)` for a specific station chosen, which ensures the page that the station data required for download exists in the global storage.

To check if the data already exists the module `checkIfDataExists` is called. It has several functions to check if each endpoint needs to be fetched, for example by checking if a specific river has the needed data to display its summary. This is a form of caching, which improved performance by only needing to fetch specific data once. However, this data is gone once the website is reloaded, which means reloading the page requires the page to fetch the data again. This does somewhat reduce load times, however, this guarantees that the user can view new data on the web page after they have uploaded it, which would not be guaranteed if the data was cached between page loads.

To validate the data, the `dataManager` module first checks if any data was received, and if so it uses a custom validation module under `utils/` to validate the JSON schema and content. Validation is further explained in Section 7.3.

The function `getRiverSummary()` from the `dataManager` module can be seen in Listing 6.4. It uses functions for checking if the rivers already exists, if not it tries to fetch their data, validate it, and update the store. It also catches any possible errors and displays an appropriate error message. The function ensures that all rivers from `river_with_species` is stored and accessible with no malicious data when the function is called.

```
/**
 * Ensures that all rivers are stored in the river store such that the
 * map and list page can display them
 * @returns {void}
 */
export async function getRivers () {
  if (doesAllRiversExistInStore()) {
    return
  }

  try {
    const fetchedRivers = await fetchRivers()

    // Validate the fetched rivers
    if (!fetchedRivers || !validateRiverWithSpecies(fetchedRivers)) {
      return
    }
    updateStoreWithObjects(riverStore, fetchedRivers, River)
  } catch (error) {
    addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.
      POSTGREST_UNAVAILABLE, FEEDBACK_MESSAGES.POSTGREST_UNAVAILABLE)
  }
}
```

Listing 6.4: `getRiverSummary()` function in `datamanager` module

Storing retrieved data is done by updating the Svelte river or station stores (Svelte stores are discussed in Section 6.2.4). The `dataManager` module has functions for updating the stores with JSON objects, called `updateStoreWithObject` and `updateStoreWithObjects`. These convert JSON objects to classes, and merge the new data with what already exists in the stores, ensuring that no data is overwritten. To view how the function `updateStoreWithObjects` is written, see the code implementation Appendix O.

6.2.4 Svelte Store and Data structure

Svelte Store

To make the data river and station data globally accessible the application uses Svelte stores. A store is a globally accessible object with a subscribe method which allows both Svelte components and JavaScript modules to read and get notified when the store is updated [95]. Furthermore, the stores can be writable, which allows the Svelte components and JavaScript modules to modify and update it.

The application has several stores, each in their own files. There is a store for rivers, stations, the authentication status, and for the feedback to display to the user. These are simply called `riverStore`, `stationStore`, `authStore`, and `userFeedbackStore` respectively. These are initialized as empty data structures, with the use of the `writable` function to make them modifiable.

To access the Svelte stores, the components uses the reactive `$store` syntax [96]. This allows components to read the content of a store, where the variable always contains the updates value (see Listing 6.5). Under the hood the reactive `$store` syntax subscribes to the store, and automatically unsubscribes when appropriate, for example when the component is destroyed. The JavaScript modules on the other hand uses the `get(store)` method, which simply subscribes to the store, reads the value, and then immediately unsubscribes. To update the store the modules either uses `store.set(content)` to overwrite the content, or `store.update(currentContent => {newContent})` to update it. Listing 6.6 shows how `get()` and `store.set()` can be used to get the content and update a store.

```
// Get rivers and stations from stores
$: rivers = $riverStore
$: stations = $stationStore
```

Listing 6.5: Reading stores in a Svelte Component, where the rivers and stations variables will always contain the up to date content of the stores

```
if (get(store).size === 0) {
  const objectMap = new Map(objects.map(object => [object.id, Class.fromJson(
    object])))
  store.set(objectMap)
  return
}
```

Listing 6.6: Retrieving and setting the value of a store in a JavaScript module

Data Structure

The data structure used in the river and station Svelte stores are simply JavaScript maps. The keys for the rivers and stations are their IDs and their value is River or Station objects. All id's are retrieved from PostgREST. This allows the application to retrieve the values of an object in the stores using `map.get(key)`, by for example retrieving a specific river as follows: `$riverStore.get(riverId)`.



Figure 6.3: Screenshot of the menu for selecting river or stations

The River and Station objects are defined in their own JavaScript files under `models/`. They both contain a JSDoc comment of their structure, documenting the data types and purposes of each of their properties. The classes contain a `fromJson()` method, which creates a River or Station object based on a JSON object. This is required as the JSON data uses `snake_case`, while the classes uses `camelCase`. Each station also contain an array of its observations (as seen in the data structure outlined in Section 4.2.2, where each observation is an object containing data such as fish length and its species. To view the station class, and more details around why rivers and stations are in separate maps, while observations are under stations, see Appendix O

6.3 Handling Data

Environment data retrieved from PostgREST is used by each of the application's pages for different purposes. The raw data is often not displayed directly, but rather filtered based on user inputs, such as by date, species, name, and datatype. The data is also formatted for different purposes, such as display in a table or for inserting into an Excel file. Furthermore, the data is processed to calculate useful data for the various pages, such as distribution of fish length, median length for a specific species in a station, minutes spent fishing, and more. This section covers how data is handled and processed by the application to achieve this.

6.3.1 Filtering of Data

Filtering of data is accomplished using a custom `filterData` module. It contains functions which takes in rivers or stations objects, and filter these based on parameters given, for example the desired date range. This allows other modules and components to call these functions when they have data to be filtered.

The pop-up menu for selecting rivers and stations, used on the graph and download page, requires filtering of data to display suggestions to the user (see Figure 6.3). Here the user can filter suggestions based on the datatype, either rivers or stations, and a date range. There are several hundred rivers and stations to choose from, therefore it is useful to be able to narrow this down. The user can then search using the river or station name and date, where suggestions will appear based on their input.

A function for filtering is `filterRiversByDateAndSpecies()` (see Listing 6.7). This function can be called by components when a user has selected the species and date-range they are interested in. The functions first calls `filterRiversBasedOnDates()`, then `filterObjectsBasedOnSpecies()`, and returns the rivers after they have been filtered by both, catching any potential errors that could occur.

```
/**
 * Filters rivers based on their species and date
 * @param {Map<number, object>} rivers - The map of rivers to filter, keyed by
   a unique id
 * @param {string[]} species - The species to filter on
 * @param {string} startDate - The start of the date interval to filter on
 * @param {string} endDate - The end of the date interval to filter on
 * @returns {Map<number, object>} - A filtered map of rivers
 */
export function filterRiversByDateAndSpecies (rivers, species, startDate,
  endDate) {
  try {
    // Filter rivers based on if they are between or have overlap with the
    // start and end date
    const filteredDateRivers = filterRiversBasedOnDates(rivers, startDate,
      endDate)

    // Filter rivers based on if they have a species that is in the species
    // list
    const filteredSpeciesAndDateRivers = filterObjectsBasedOnSpecies(
      filteredDateRivers, species)

    return filteredSpeciesAndDateRivers
  } catch (error) {
    addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.GENERIC,
      FEEDBACK_MESSAGES.GENERIC)
    return new Map()
```

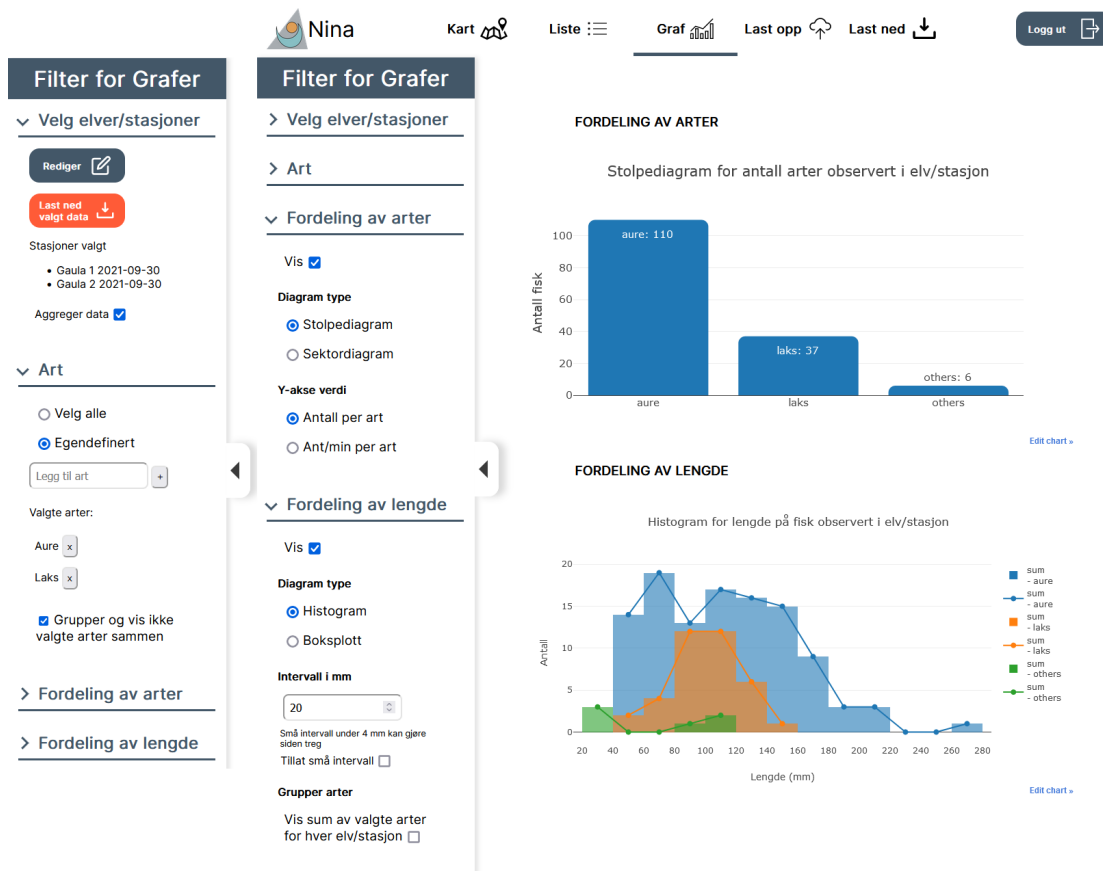


Figure 6.4: Screenshot of the graph page with the whole sidebar visible side by side, where two stations are aggregated and two species selected

```
}
}
```

Listing 6.7: The function `filterRiversByDateAndSpecies()` from the `filterData` module

Example code and an explanation of how the `filterObjectsBasedOnSpecies` function works can be seen in code implementation Appendix O.

6.3.2 Plotly data

To convert the observation data into plots, the raw observation data has to be formatted in specific formats that Plotly can understand. This is done using the custom `plotlyData` module. The formatting considers how the user wants the data to be represented, where some of the user input fields can be seen in Figure 6.4. The application uses four different Plotly plot types; bar charts, pie charts, histograms, and box plots. Each of these requires their own data structure and unique implementation, except for bar and pie charts, which use the same data structure. The data for each plot uses the observations under specific rivers or stations selected.

In addition, the user can choose between combining all selected rivers/stations, or comparing them. The user can also choose which species to view, and optionally display all species not selected in a *others* category. For bar and pie charts, the user can choose to view the absolute amount of fish per species per river/station, or to view this in relation to the amount of time spent fishing per river/station. For the histogram and box plot, the user can choose to group all species in each river/station together, to compare the sum of each river/station. In the histogram the user can choose the size of the interval used to for the length distribution. Overall, there are many factors to consider when creating the Plotly data.

Plotly data structure

Both the bar chart and the pie chart uses the same data structure. They require data on how many fish were counted for each species, grouped by each river/station. For example, there was two Laks and four Ørret in Gaula 2021. The histogram require data where each species has its fish split up into intervals based on their length, for example that there was one Laks in the interval 0-20mm, and one Laks in the interval 20-40mm. The box plot requires data which is a list of all fish lengths, for example one Laks with length 10mm, and one Laks with the length 25mm. These data structures are explained in more detail in Appendix O.

Functions

Organizing the code in the custom `plotlyData` module is accomplished by utilizing several different functions. The two main functions, which are the ones called from the components, are `dataForBarAndPieChart()` and `dataForHistogramAndBoxPlot()`. These functions take in the selected rivers or stations, the species to use, if the value should be absolute or relative, if rivers/stations should be aggregated, if other species are included, if species should be combined, and interval size. These functions return data containing the information outlined above.

The function `getIntervalsForObservations()` from the `plotlyData` module is shown and explained in Appendix O.

6.3.3 Calculate Data

In addition to wanting to view the data in plots, the user should also be able to view data about each river and station when they are clicked on the map and list page. The raw data does not contain all of the values the user should be shown. In Figure 6.5 you can see the data displayed to the user when a station is clicked. This includes information such as amount of fish for each species, amount of fish per minute, median length of fish, and more. To calculate this data the custom module `calculateData` is used.

To see how the function `fishPerMinuteInStations()` from the `calculateData` module works you can view the code implementation Appendix O.

6.3.4 Format Data

To allow users to download data and view it in tables, the module `formatData` is used to convert data into predefined formats. This includes formats for rivers, stations, and observations in CSV

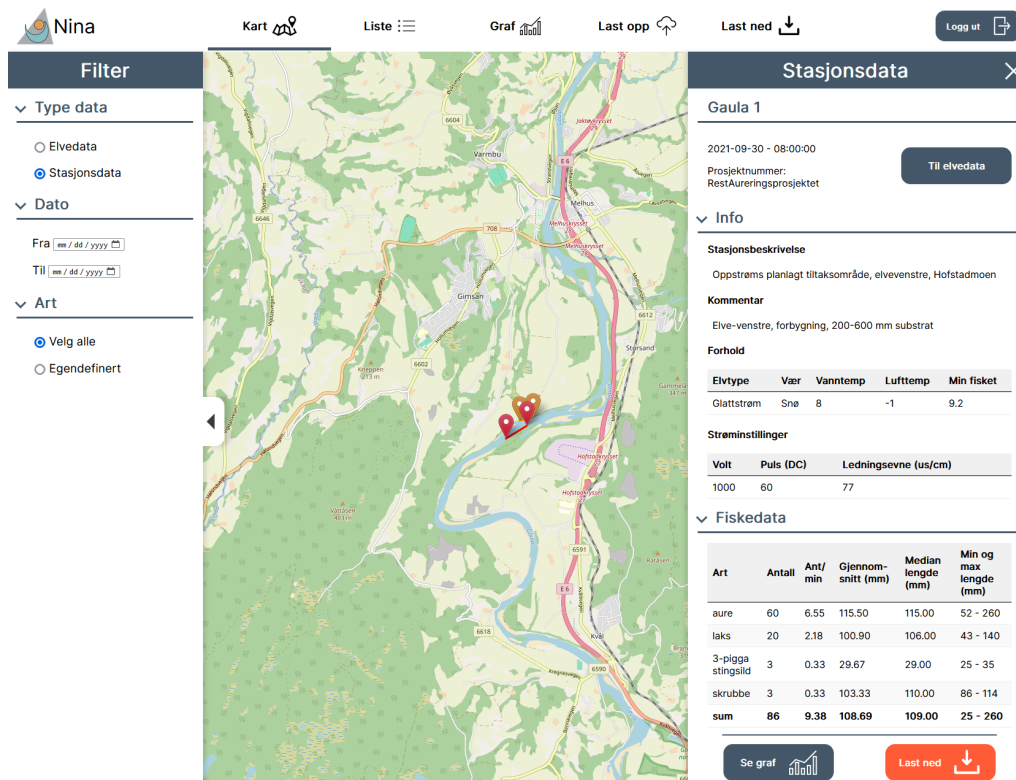


Figure 6.5: Screenshot of the map page with a station selected

Stasjon	Elvtype	Vær	Min fisket	Ant fisk	Fisk/min
1	Glattstrøm	Snø	9.2	86	9.38
2	Glattstrøm	Snø	9.5	67	7.08

Figure 6.6: Table containing the stations under a river when a river is selected on the map page

and Excel files. Moreover, it also includes formats for several types of tables which are used in the application, such as formatting multiple stations for display in a river summary (see Figure 6.6).

The implementation of the function `getIntervalsForObservations()` from the `plotlyData` module is shown and explained in Appendix O. The function `formatStationsForSummaryTable()` from this module is also shown in Appendix O, which is what creates the data inserted into the table as shown in Figure 6.6.

6.4 File Handling

The application should handle files, allowing users to both upload and download data. The data can be uploaded in the XLSX file format, and downloaded in both the CSV and the XLSX file formats. This section outlines how this was implemented as part of the application.

6.4.1 Upload of Data

To allow users to upload data, the website has implemented the possibility for users to drag and drop a file, or chose a file using the file explorer on their computer (see Figure 6.7). The file has to be an Excel file, with the `.xlsx` extension, and have the file format specified in `Upload.md` in the project repository, which is based on NINA's Excel files. Only one file can be uploaded at once, with a maximum size of 10MB.



Figure 6.7: Screenshot of upload page with a file chosen

Selecting Files

Files can be selected by clicking the browse files button, called 'Bla gjennom filer'. This button calls a function opens the file explorer windows, allowing the user to select a single XLSX file. A file can also be selected by dropping it in the outlined box, where this created an event handles by another function. This function checks if a single file was dropped and if it has the correct XLSX file extension. This functionality is explained more in the code implementation Appendix O.

Validating File

Validation happens after a user tries to upload a file. The application does validation by reading the file contents and checking the format and content of the cells in the file. This is explained further in Section 7.3.3.

Uploading File to Server

The selected file is configured to be sent to an upload endpoint managed by NINA. The endpoint expects HTTP POST requests with Content-Type: multipart/form-data, where the body will contain the file encoded as MIME parts. This is a standard way of sending files to a HTTP endpoint, where the file is sent as multiple HTTP requests to an endpoint.

Sending the file to the endpoint is done using the `uploadFileToServer()` function in the module `upload`. The function sends a HTTP request with the correct headers, including the `sessionId` for authentication (see Section 7.1), and the file encoded as MIME parts. It also checks the response and gives the user feedback based on if the file was successfully uploaded or not. The function is explained in more detail in Appendix O.

6.4.2 Download of Data

Downloading data is done by another page of the application (see Figure 6.8). On this page the user chooses which rivers or stations they want to download, the species they want to include observations for, and the type of file they want (XLSX or CSV). The downloaded file follows the same format as uploaded files. A XLSX file separates the data between different sheets, and CSV files contain all the data on the same page.

Downloading File

After the user has chosen what they want to download and clicked the download button, the `downloadFile()` function is ran. This functions first checks if the options chosen are valid. If they are, it creates the file name and the content by using the `fileHandler` module functions `generateExcelFile()` or `generateCSVFile()`. If the file was successfully created the function creates a temporary HTML element which automatically saves the file created to the users computer. The function is explained and showed more in detail in Appendix O.

The Filehandler Module

The actual functions which create the Excel and CSV files are called `generateExcelFile()` and `generateCSVFile()` which are placed in the `fileHandler` module. The function `generateExcelFile()` generates an excel file containing the selected rivers or stations and the selected species. It retrieves this data using `formatRiversForExcel()` or `formatStationsForExcel()` from the `formatData` module. These functions formats the data for by splitting it into a river, a station, and an observation sheet. Each of these sheets contain headers and rows for each river, station, and observation. The library `ExcelJS` is then used to create a workbook containing the different sheets. The workbook is finally converted to a `Blob` representing the content

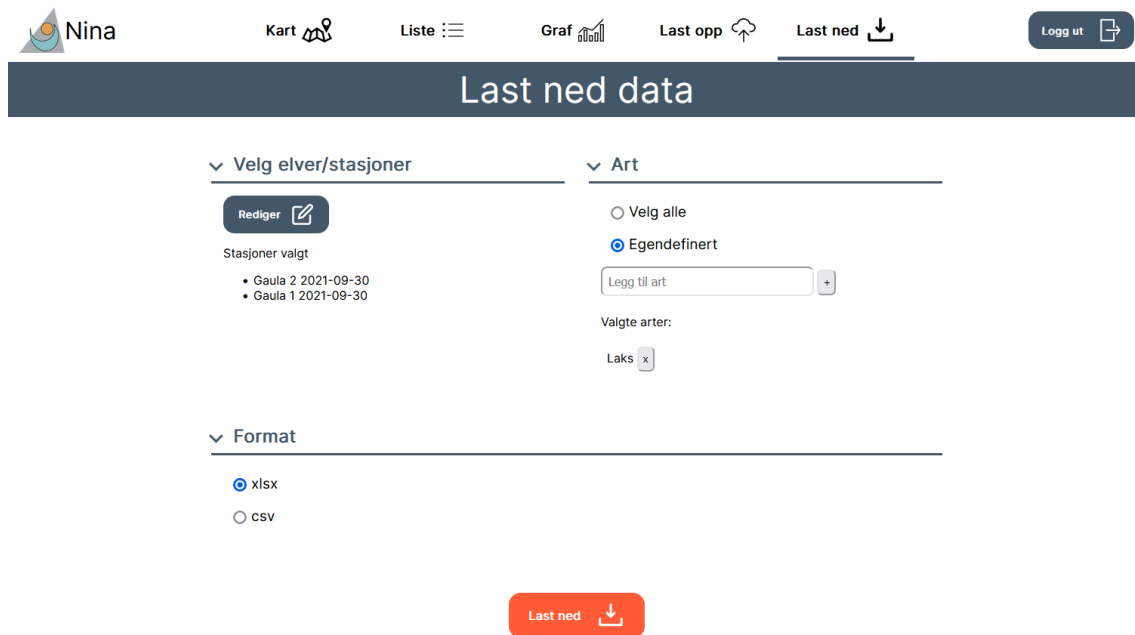


Figure 6.8: Screenshot of download page with stations, a species, and the XLSX format selected

of a spreadsheet, which is a way for JavaScript to store a file as binary data, and is then returned. It is this Blob that gets downloaded to the users computer as an Excel file. The function `generateExcelFile()` can be seen in Appendix O.

6.5 Leaflet

Leaflet is used by the application to create a interactive map, allowing the users to view and interact with all the river or station data points (see Figure 6.9). Leaflet handles rendering, placing of markers, grouping of figures, and interaction with the map. The whole Leaflet implementation is placed inside a single component. This section shows how Leaflet was implemented into the application.

6.5.1 Initialization, Controls, and Layers

The Leaflet map is only initialized after the Leaflet Svelte component is mounted (see Listing 6.8). It's crucial that the map waits for all CSS to load by using a delay, because the map might load incorrectly with wrong dimensions if initialized too early. The map is initialized with a terrain map zoomed in on Norway, since this is most of NINA's data is located. Different controls are added on top of the map using the `leaflet.control` functions, including zoom, scale, and map type controls. The river and station markers groups containing the data points are also added to the map. There are three map layers added, one for terrain, one for satellite, and one for a topology. These are defined and imported from another file.

```

onMount(async () => {
  // Wait 250ms after DOM is rendered before creating map to ensure all css is
  // loaded and applied
  setTimeout(() => {
    // defines the map and sets the view to Norway
    map = leaflet.map(mapElement, {
      layers: [mapLayers.Terrain], // Default layer
      zoomControl: false // Disable default zoom control
    }).setView([61, 12.09], 6)

    // Add zoom control in top right corner
    leaflet.control.zoom({ position: 'topright' }).addTo(map)

    // Add terrain and satellite layers option to the map
    leaflet.control.layers(mapLayers, null, {
      position: 'bottomright'
    }).addTo(map)

    // Add scale in bottom left corner
    leaflet.control.scale().addTo(map)

    // Add river and station layer groups to the map
    riverLayerGroup.addTo(map)
    stationLayerGroup.addTo(map)
  }, 250)
})

```

Listing 6.8: Leaflet initialization

To display the map it needs to be linked with a HTML element. This is accomplished by referencing the div you want to display the map in, which in this case is `mapElement`. The only CSS needed for the map div is the its height and width.

6.5.2 Markers

There are two groups of markers in the application, one for rivers and one for stations. They have similar logic for creation and handling clicks, however rivers are only represented as a single point, while stations consist of two points and a line. These represent the start and stop position of the station, while the line connects the start and stop together. This section focuses on stations, where rivers have equivalent functions.

The station markers are added with the function `createStationMarker(station)`. It uses a station object and its coordinates to create the points and the line connecting them. To allow the user to click and select a station, the function add an `onclick` event, which notifies the component with the ID of a station clicked. How the line connecting the station start and end points is shown in Listing 6.9. After creating the points and line, they are ad-

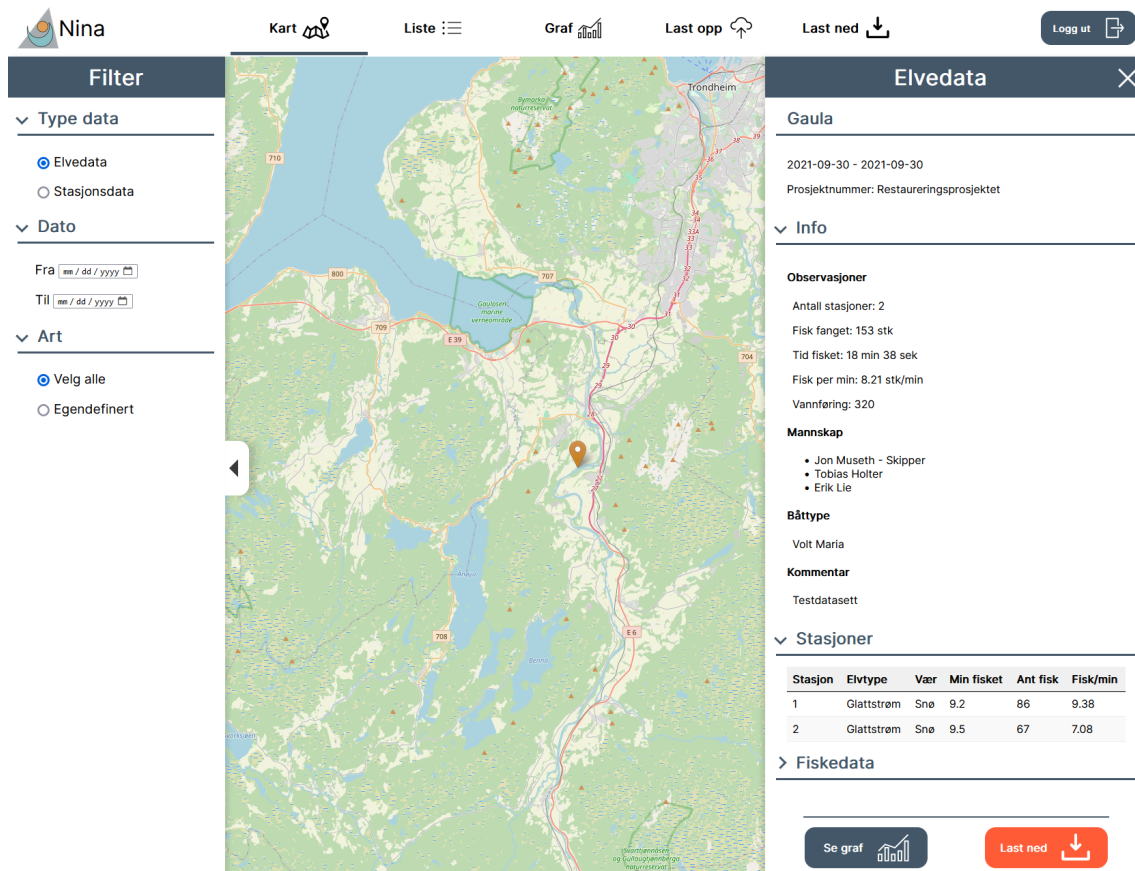


Figure 6.9: Screenshot of the map page with a river selected

ded to the `stationLayerGroup`, which displayed them on the map. The whole station marker is also created as an object and stored for later updates and removals. To view the whole `createStationMarker(station)` function, see the code implementation Appendix O.

```
// drawing the line between the start and end position of the station
const polyline = leaflet.polyline([startPos, endPos], { color: 'red' })
  .addTo(stationLayerGroup)
  .on('click', () => dispatch('stationClicked', { id: station.id })))
```

Listing 6.9: Code showing how a line is created for station markers connecting their start and end points

Users can filter stations shown by start and end dates, or by specifying the species they are interested in. The function responsible for managing and updating what stations are displayed is called `updateStations`. The function is efficient, which is important when displaying thousands of points. It never redraws station markers, instead it creates station markers once, and hides and displays them later based on the filters chosen by the user. The function `updateStations` is explained and shown in more detail in Appendix O.

The function `selectStation()` is used when a user selects a station marker. This function

first calls another function `unSelectStation()`, making sure to deselect the previous station. Then the selected station has its color of both the markers and line changed to orange. Finally the leaflet function `flyTo()` is used to zoom in and centralize the selected station (see Listing 6.10).

```
// Pan to the selected station and zoom in
map.flyTo(marker.startMarker.getLatLng(), 13, { duration: 0.5 })
```

Listing 6.10: Use of `map.flyTo()` when selecting a station

6.5.3 Svelte Integration

The Leaflet integration with Svelte is done through a singular component, which contains all Leaflet code logic. The script tag handles initialization and updating the map, the HTML is simply a single div element, and the CSS sets the width and height of the div.

To render the Leaflet component in the graph page, it is simply included in the HTML (see Listing 6.14). Using Svelte props, the Leaflet component is given the `dataType` value, the river and stations to display, and the selected river or station. The prop `dataType` tells the component if it should display the rivers or stations. On the events `stationClicked` and `riverClicked`, which are dispatched from the Leaflet component when a station or river is clicked, the appropriate functions on the graph page are called, handling the selection.

```
<!-- Map with rivers and stations -->
<LeafletMap
  {dataType}
  rivers={filteredRivers}
  stations={filteredStations}
  {selectedRiver}
  {selectedStation}
  on:stationClicked={stationClicked}
  on:riverClicked={riverClicked}/>
```

Listing 6.11: Use of Leaflet component

6.6 Plotly

Plotly is responsible using the environmental data to create figures and diagrams, including bar charts, pie charts, histograms, and box plots. Each depict and illustrate the data in a unique way, providing the user a wide range of options for visualising data. The users selects the rivers or stations they want to visualise, allowing the user to compare or aggregate data, and specify the species they want to view. The figures also include special settings, allowing user to set the value to relative or absolute, or change the interval for the histogram. This section displays how Plotly is implemented into the application.

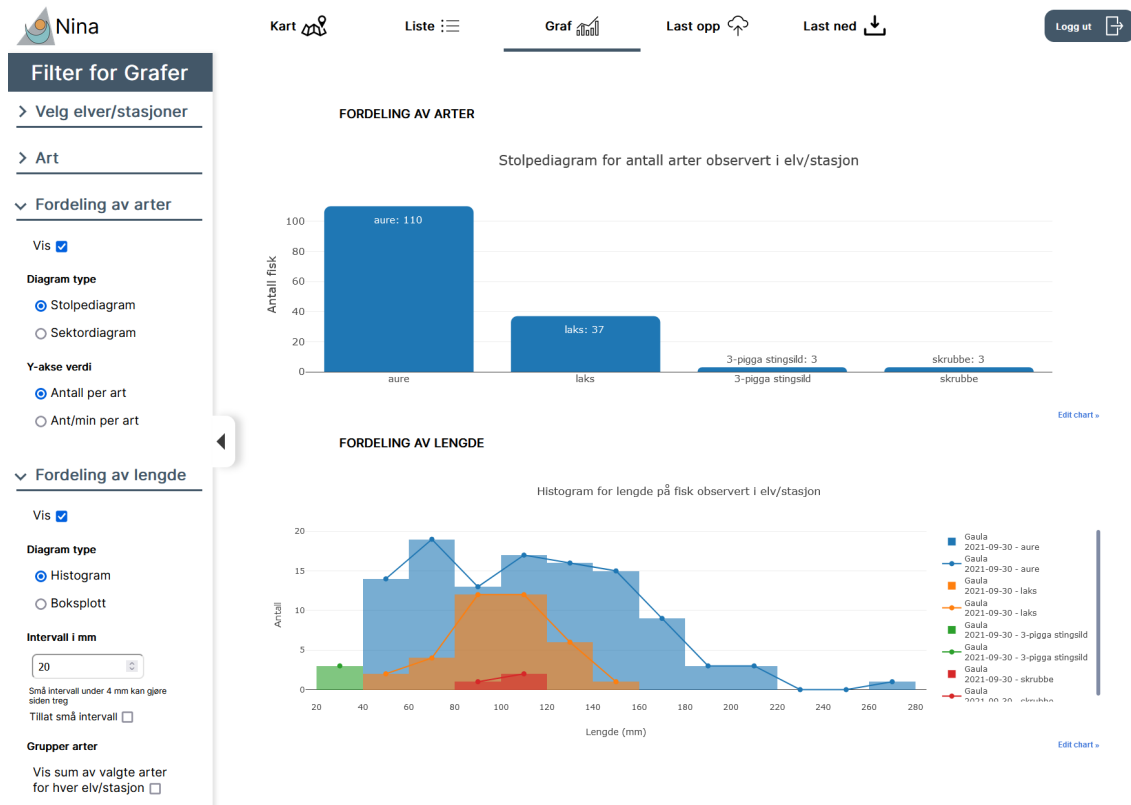


Figure 6.10: Screenshot of the graph page with a river and all its species selected

6.6.1 Drawing Plots

Drawing the environmental data is done using data from the `plotlyData` module. The data from `plotlyData` is structured differently for the different graph types, something that is discussed in more detail in section 6.3.2. To display the data in Plotly, traces are created based on the data. The traces of a plot in Plotly is the content displayed on the page. A trace can for example represent the bars in a bar-chart, a single pie chart, or a single box plot. Each of the plot types have their own component and logic for creating these traces.

The traces of the different plots are created using a function called `drawPlot(plotdata)`, which is implemented differently for each plot type component. For example, to create traces for the histogram plot each species is iterated through, where the intervals and the amount of fish in each is added to the traces as x and y values for the bars. The traces also take in values such as color, opacity, name, and type. A more in depth explanation of how the traces are drawn for the histogram plot is found in the code implementation Appendix O.

6.6.2 Styling Plots

Styling the plots is done by configuring the variables `layout` and `config`, which are used in addition to traces to draw the plots. The variable `layout` is used to define the title, axis names,

and other configuration affecting the layout of the traces (see Listing 6.12). The config on the other hand configures the behaviour of the whole Plotly element, including if it automatically resizes and which buttons to show (see Listing 6.13). When the traces, layout, and config is defined, they are all used to create the finalized plot with the function `Plotly.newPlot('fishHistogram', traces, layout, config)`.

```
// Displays the title, names of the axes and removes gaps between bars
const layout = {
  title: 'Histogram_for_lengde_på_fisk_observert_i_elv/stasjon',
  yaxis: { title: 'Antall' },
  ...
}
```

Listing 6.12: Defining a plot layout in plotly

```
// Make graph responsive, remove some buttons from the modebar, add edit link
const config = {
  responsive: true,
  modeBarButtonsToRemove: ['select2d', 'lasso2d', 'zoomIn2d', 'zoomOut2d'],
  ...
}
```

Listing 6.13: Defining a plot config in plotly

6.6.3 Svelte Integration

Each diagram type is defined in their own Svelte component, since they each have unique logic associated with drawing the plots. The script tags contains all the JavaScript needed to draw and update the plots, with the rest of the components only being a HTML div used by Plotly to draw inside.

The components are used by referencing them inside the HTML of another component. Using if statements in Svelte, the HTML can call different Plotly components based on user choices (see Listing 6.14). The components only need the formatted data from the `plotlyData` module, however bar chart also needs to know if the data is absolute or relative to display the correct unit in the y-axis.

```
<!-- Plot graph choosen by user -->
{#if type === 'barchart'}
<BarChartComponent plotData={formattedData} {absoluteValues}/>
{:else if type === 'piechart'}
<PieChartComponent plotData={formattedData}/>
{:else if type === 'boxplot'}
<BoxPlotComponent plotData={formattedData}/>
{:else}
<HistogramComponent plotData={formattedData}/>
{/if}
```

Listing 6.14: Use of Leaflet component

6.7 Navigation

The application was created with several pages, and the user needs to be able to navigate between them. Navigation includes being able to go to each page by using the navigation bar. Moreover, it also includes being able to navigate by opening selected rivers and stations on different pages, for example by clicking 'show in map'. The following is how this is achieved by the application.

6.7.1 SvelteKit Routes

Utilizing SvelteKit enables the application to be a single page application. SvelteKit allows for creating specific routes as folders in the file structure, where each file route is equivalent to an Uniform Resource Locator (URL) path [33]. Each page on the application has its own folder and route, except for the map page which is placed in the root folder (see Figure 6.11). The root folder specifies the "homepage", the default page when opening the application. Each route has a `+page.svelte`, which are components defining the specific HTML, CSS, and JavaScript for the page. For example, the upload page is placed under `src/routes/upload/`, and will be accessible on the page under `http(s)://<domain>/upload/`.

6.7.2 Configuring SvelteKit as a Single Page Application

SvelteKit is by default designed for server-side rendering, meaning when a user switches between pages, each route is rendered on the server and then sent to the client. Disabling this, and thus enabling that users can navigate between pages without reloading, is accomplished by setting the line `export const ssr = false` inside `+page.server.js` under `/routes/`. This enables client-side rendering, meaning the entire application will be rendered on the client side [97]. This results in the user getting all the pages when opening the website, preventing the pages from reloading when navigating on the site, which makes the website a single page application.

In the `+page.server.js`, SvelteKit can be configured further by setting the `prerender` option to `true` (`export const prerender = true`), telling SvelteKit to pre-render all pages in the application [98]. This results in the pages being turned into static HTML at build time, preventing the pages to be built for each request sent by a user. Furthermore, as the entire application is pre-rendered, `adapter-static` in the `svelte.config.js` can be set. This ensures that the application build outputs files for static serving, which can be used directly by Nginx.

6.7.3 Use of `<a>` Elements

SvelteKit's support of the `<a>` HTML element [99] is used to let users navigate between pages. This is done by setting the `href` attribute of the `<a>` element to the target path, for example


```

src/routes/
├── +page.svelte
├── +layout.svelte
├── +page.server.js
├── list/
│   └── +page.svelte
├── graph/
│   └── +page.svelte
├── upload/
│   └── +page.svelte
├── download/
│   └── +page.svelte
├── login/
│   └── +page.svelte

```

Figure 6.11: SvelteKit filesystem routes

``. Furthermore, the href attribute can contain URL parameters, for example ``, allowing the user to seamlessly navigate to another page in the application while the application remembers the stations or rivers selected.

6.7.4 URL Parameters

The application uses URL parameters to allow for cross-site navigation where selected rivers and stations are transferred to the new page, and to allow users to share and bookmark pages with specific rivers or stations selected. On the map and list page a URL parameters corresponds to the selected river or station. The URL parameters on the graph and download page works the same way, except that these pages allow multiple rivers or stations to be selected at the same time, resulting in multiple URL parameters. The function `getUrlParams()` reads the URL parameters on the various pages. Using `$page.url.search`, which is a Svelte variable containing all the parameters in the URL, the function retrieves the river and station ids. The function ensures the parameter strings are valid integers and then converts them to integers. The rivers and stations are then selected on the pages based on the retrieved IDs. The function `getUrlParams()` is discussed in more detail and shown in the code implementation Appendix O.

Updating the URL after a user has selected a river and stations is done by the function `updateUrl()`. This function first ensures that the windows object is defined, preventing an exception being thrown by reading an undefined URL. The function then removes the old parameters, adds each river or station id as long as they are numbers, and uses the `goto()` SvelteKit function to update the URL. The `goto()` function with `replaceState: true` replaces the current URL with the new one without reloading the page or modifying the browser history stack. By not modifying the browser stack users can go back to previous pages by clicking the back button in the browser. The function can be seen in Appendix O.

Chapter 7

Security Implementation

In addition to implementing the functionality of the application, as seen in Chapter 6, the development process also includes implementing the countermeasures outlined in the risk assessment (see Section 3.5). This is crucial to protect the application against possible threats. The implementation of these countermeasures also ensure that the security requirements outlined in Section 4.4.3 are followed. This chapter covers the implementation of countermeasures in the front-end web application, which includes proper session handling and authentication, proper error handling, input validation, and output sanitization. It also covers how countermeasures are implemented for the back-end, including encryption from NINA and the configuration of a Content Security Policy.

As in Chapter 6, there is more detailed coverage of the code in the application in Appendix O.

7.1 Authentication Solution

This section covers how the authentication mechanism for the application is implemented. Authentication is used to allow users to reading and upload the environmental data.

Authentication Endpoints

NINA has set up a Django service with an authentication API responsible for authenticating and authorizing users. It achieves this by creating endpoints for login, logout and refreshing tokens. To track if a user is authenticated Django sets and reads a cookie named `sessionid`. The cookies contains a token as its value, where each token has a certain lifetime and proves that the user is authenticated.

The login endpoint expects a POST request with a username and a password encoded as JSON data. If the credentials are valid, the endpoint will respond by setting the `sessionid` cookie with a valid session token. The logout endpoint expects a POST request with the `sessionid` cookie. The endpoint first invalidates the token, ensuring that it can no longer be used for authentication, and then responds by setting `sessionid` to an empty value. The refresh endpoint is to refresh an expired token, and also expects a POST request with the `sessionid` cookie. It allows a user to get a new valid session token in a certain time frame after the previous token

expired, without having to log in again. The time frame is defined by NINA. If the token can be refreshed, i.e. the token was previously valid and recently expired, the endpoint will respond by setting `sessionid` to a new valid token.

Setting the `sessionid` cookie is accomplished by adding a `Set-Cookie` header into HTTP responses as seen in Listing 7.1. For security reasons the `Set-Cookie` header also contains the attributes `HttpOnly`, `SameSite=Strict` and `Secure`. Enabling `HttpOnly` results in the cookie only being available for the browser, preventing the applications JavaScript from having direct access to the cookie [100]. The `SameSite=Strict` attribute prevents cookies from being sent with any cross-site request, ensuring that cookies are only included to requests to the Same-site. In this context same-site refers to requests having the same registrable domain (e.g. `nina.no`) and using the same protocol (HTTP or HTTPS). Subdomains are considered same-site [101], meaning requests to different subdomains are allowed. Lastly, `Secure` means the cookie can only be sent over HTTPS, preventing the cookie from being sent unencrypted and potentially eavesdropped.

```
Set-Cookie: sessionId=token_value; HttpOnly; SameSite=Strict; Secure;
```

Listing 7.1: Set-Cookie header in authentication endpoint HTTP responses

Authentication API Module and Authentication Store

In the application authentication is handled by a custom module under `api/` called `auth`, and a Svelte store containing the authentication status, called `authStore`. The `auth` API module handles login, logout, and refresh requests, allowing other parts of the application to use these functionalities. The `authStore` keeps track of the state of authentication, and it's responsible for showing either a login or logout button depending on the authentication status. The application itself only needs to keep track of the login status, since the tokens used for login, logout and refresh are stored in cookies. When clicking the login button, the user gets taken to a separate login page, as seen in Figure 7.2.

Figure 7.1 illustrates how the authentication state is managed. The application checks if request return `401 Unauthorized`, and if so it should try to refresh the token using the `auth` module. If this also fails, i.e. return `401 Unauthorized`, the user is told to log in. However, if requests to PostgREST or Django does not return `401 Unauthorized` nor any other errors, the application assumes the user is logged in, and sets its status accordingly using `authStore`. If requests with the user provided credentials to the login endpoint was successful the status is also updates to logged in. Logging out also updates the status accordingly if the request does not return any errors.

The function responsible for refreshing authentication is `authRefresh()`, and is shown in Listing 7.2. The function refreshes the authentication token by creating a POST request to the refresh endpoint containing the current `sessionid`. The function only needs to set the `credentials` attribute to `'same-origin'` for the `sessionid` to be included. Setting this attribute tells the browser to include any cookies from the same origin in the request, which means request to PostgREST and Django will include the cookies, as these run on the same server as the web server. It then updates the authentication status and notifies the user based on the response from the request. The `authLogin` function and `authLogout` function uses the same

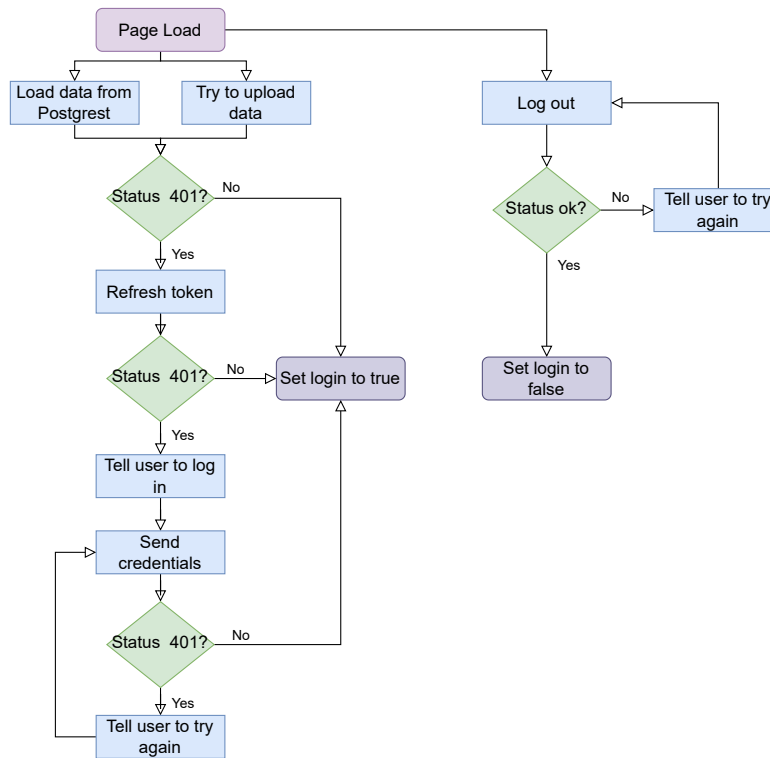


Figure 7.1: Flow of application for managing authentication

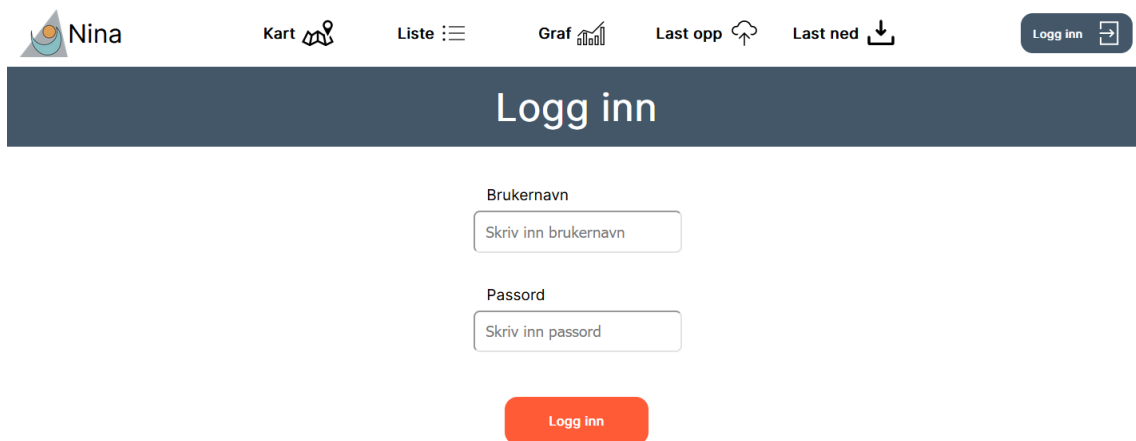


Figure 7.2: Screenshot of the log in page

logic, where they construct requests with JSON encoded credentials or cookies, then update the store and give feedback based on the response.

```
/**
 * Refresh user token
 * @returns {Promise<boolean>} - A promise which resolves to if refresh was
   successful or not
 */
export async function authRefresh () {
  try {
    // Try to refresh
    const response = await fetch(`${AUTH_URL}${REFRESH_ENDPOINT}`, {
      method: 'POST',
      credentials: 'same-origin'
    })

    // Check if the response is ok
    if (!response.ok) {
      // Tell user to log in and update authStore
      addFeedbackToStore(TYPES.ERROR, CODES.UNAUTHORIZED, MESSAGES.UNAUTHORIZED)
      authStore.set({ authenticated: false })
      return false
    }

    // If refresh was successful tell user to refresh and update authStore
    addFeedbackToStore(TYPES.SUCCESS, AUTH_SUCCESS, MESSAGES.REFRESH_SUCCESS)
    authStore.set({ authenticated: true })
    return true
  } catch (error) { // Catch any possible network or fetch errors
    addFeedbackToStore(TYPES.ERROR, CODES.AUTH_UNAVAILABLE, MESSAGES.
      AUTH_UNAVAILABLE)
    return false
  }
}
```

Listing 7.2: The function authRefresh() from the auth module

7.2 Error Handling

There are multiple potential sources for errors in the application, e.g. wrong user input, a third-party component failing, or an internal software error coming from trying to read a null value. The application handles these safely, and provides the user with useful, but non-verbose, feedback. This section looks into how this was achieved.

7.2.1 Handling Errors in Code

Handling errors in the code is done using the `try-catch` JavaScript blocks. This ensures that if any errors happens inside of the `try` block the `catch` block will explicitly handle it. This ensures the error is not exposed in a verbose nature or that the application breaks, but rather lets the application handle the error how it sees fit. An example on how this was implemented can be seen in the function `fetchFromPostgrest()`, which is discussed in the code implementation Appendix O.

Handling invalid user input is accomplished by checking the input values and informing the user of errors when appropriate. For example, the function `downloadFile()` checks if a user has selected a format (XLSX or CSV). If no format was selected, the function cancels the download and informs the user of the type of error with a relevant predefined error message (see Listing 7.3).

```
// Check if the user has chosen a file format
if (selectedFormat === '') {
  addFeedbackToStore(FEEDBACK_TYPES.ERROR,
    FEEDBACK_CODES.NOT_FOUND, FEEDBACK_MESSAGES.NO_FILE_FORMAT_SELECTED)
  return
}
```

Listing 7.3: The function `fetchFromPostgrest()` in the `postgrest` module

7.2.2 Displaying Errors and Feedback

Displaying user feedback, including errors, to the user is done utilizing a combination of stores, modules, and Svelte components. The feedback is saved in a writable Svelte store called `userFeedbackStore`. To display user feedback to the user, the function `addFeedbackToStore(type, code, message)` is called. The function creates a feedback object containing the feedback type, code, and message, and appends the message to the feedback store. Type of feedback includes; error, success, or information. The code categorises what the feedback is about, e.g. forbidden or unauthorised, while the message contains more detailed information, which is what's displayed to the user. Both the error types, codes, and messages are defined in a constants file. This means all error types, codes, and messages are pre-defined, ensuring that no internal error messages disclosing confidential information about the application are displayed to the user.

Actually displaying the user feedback in a pop-up window is accomplished with the custom component `UserFeedbackMessage`, which is included on all the pages (see Figure 7.3). This component displays all feedback messages in pop-up windows. When a user closes a pop-up menu with feedback, the feedback is deleted from the `userFeedbackStore`, as the user has seen the error. To illustrate the type of feedback, the component displays a SVG icon in the pop-up menu based on the feedback type. The component code and the logic is further outlined in Appendix O.

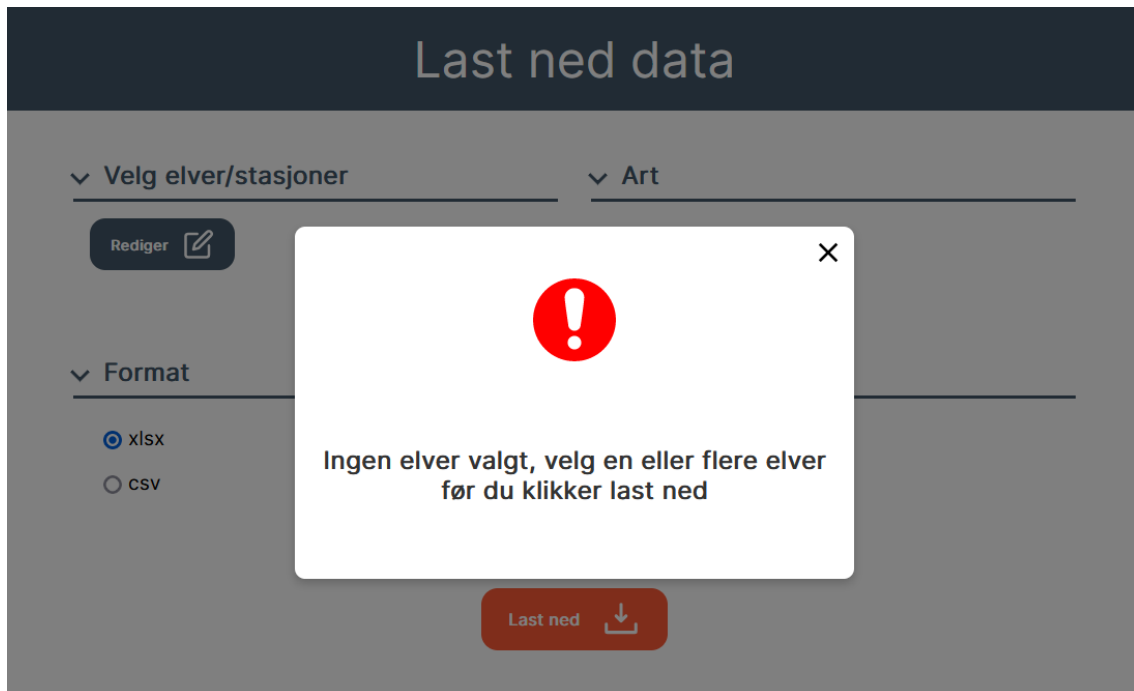


Figure 7.3: Error message when user tries to download without selecting a river

7.3 Input Validation

Injection vulnerabilities are amongst the most common web application vulnerabilities [57], and the most common countermeasure is comprehensive input validation. The application always assumes all input is unsafe, including input from users, external services, and internal services. To validate the input the application checks several recommended properties outlined by Mano Paul in his book *Official ISC2 Guide to the CSSLP CBK* [85, p. 188], including data format, datatype, and data range, making sure no input is interpreted as code.

7.3.1 User Input

The application validates all user input, including search fields, number inputs and URL parameters. To validate the input the application uses the custom module `validation` under `utils/`, which contains functions such as `validateText`, `validatePassword`, and `validateNumber`. In addition the various inputs have their own specific validation. For example, the data range of the interval on the graph page has to be a positive number, or the ID in the URL parameters have to match an existing river or station.

The function `validateNumber` checks if a string can be used as a number, ensuring the correct data type of inputs. URL parameter values are directly read as strings with no built-in validation, however the function `validateNumber` makes sure these values are validated. This function is not needed for other number inputs, as the HTML `<input>` element with type `number` has built-in validation to reject all non-numerical values [102].

The function `validateText` checks if strings contain only valid characters (see Listing 7.7). It uses regex to specify the characters that are valid, and users get a predefined feedback message if their input is invalid (see Figure 7.4). This method of validation is called whitelisting, which is the opposite of blacklisting. Blacklisting means listing everything you want to exclude, while whitelisting only specifies which characters are allowed, excluding anything else. According to OWASP [103], developers should be able to choose strong whitelists for structured input, based on the expected data structure. The text validated in the application does not follow any specific structures, but with collaboration with the regex pattern chosen for NINA whitelisting only includes the characters which is required for normal usage of the web application.

The `validateText` function is used for all text input in the application, except the password field. In addition, `validateText` is used when validating the text content of JSON and uploaded files. For validating passwords the function `validatePassword` is used. It is similar to `validateText`, except that it allows special characters frequently used in passwords [104].

```
/**
 * Validate text using regex to whitelist specific characters
 * @param {string} input - The input string to validate
 * @returns {boolean} - If the input is allowed or not
 */
export function validateText (input) {
  if (!input) {
    return true
  }

  const allowedPattern = /^[a-zA-ZæøåÆØÅ0-9 .,?!- _():+%"/*]+$/

  const isValid = allowedPattern.test(input)

  if (!isValid) {
    addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
      FEEDBACK_MESSAGES.INVALID_TEXT)
  }

  return isValid
}
```

Listing 7.4: The function `validateText()` from the validation module

7.3.2 PostgREST Input

The environmental JSON data coming from the internal PostgREST service at NINA is also assumed to be unsafe. All this data is validated on data format, data type, and on allowed characters. It does this by using the functions `validateRiverWithSpecies`, `validateStationWithSpecies`, `validateRiverSummary`, `validateStationSummary`, and `validateStationDownload` from the

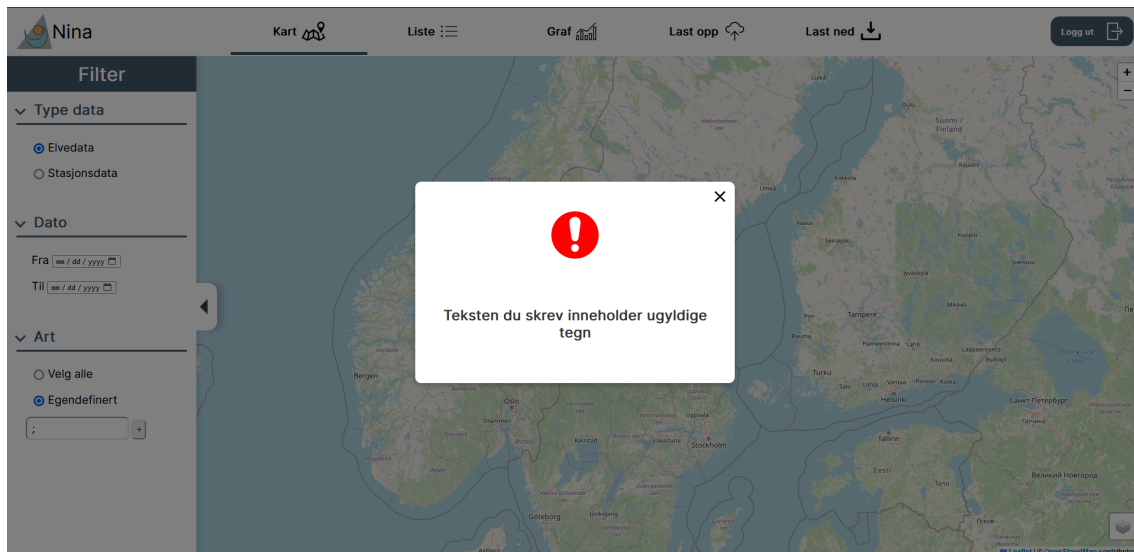


Figure 7.4: Error message when user tries to input an invalid character

custom validation module for each endpoint from PostgREST (see Section 6.2.1 for all PostgREST endpoints). All of these functions call the function `validateJson` with the JSON data and the schema the JSON should follow.

The schema for each endpoint is defined in a file called `schemas` under `src/constants/`. These are defined to be compatible with AJV, which is the library used to validate the JSON format. The schema for the `river-with-species` endpoint can be seen in Listing 7.5. Since the endpoint contains an array of objects, the schema `schemaRiverWithSpecies` expects an array with individual objects. The object schema is defined in `schemaSingleRiverWithSpecies`. It specifies what data types are allowed for each property with `properties`. Furthermore, it specifies which properties have to be included with `required`. Lastly, it specifies that the JSON can not include any other properties with `additionalProperties`. This schema results in the validated data from PostgREST always having the correct format and data types.

```
const schemaSingleRiverWithSpecies = {
  type: 'object',
  properties: {
    id: { type: 'integer' },
    name: { type: 'string' },
    pos: coordinatesSchema,
    start_date: { type: 'string' },
    end_date: { type: 'string' },
    species: { type: 'array', items: { type: 'string' } },
    project_id: { type: ['string', 'null'] }
  },
  required: ['id', 'name', 'pos', 'start_date', 'end_date', 'species'],
  additionalProperties: false
}
```

```

}
export const schemaRiverWithSpecies = {
  type: 'array',
  items: schemaSingleRiverWithSpecies
}

```

Listing 7.5: The JSON schema for the endpoint river-with-species

The function actually responsible for validating the JSON data is called `validateJson` (see Listing 7.6). This function first uses the function `validateStringsInJson`, which recursively checks all strings in the JSON data using the function `validateText`. If any string in the JSON data is invalid, all the JSON data is invalidated. After checking if the strings does not contain malicious characters, the function uses the AJV function `validate`, which was initialized as `ajv.compile(schema)`, to check if the JSON data follows the schema provided. If the schema is not followed, the user receives an error message either stating that the data from PostgreSQL is not available, or information about why the uploaded file was not allowed if a file was validated.

```

/**
 * Validate json data against a schema
 * @param {object[]} data - The data to validate
 * @param {object} schema - The schema to validate the data against
 * @param {boolean} excel - If the data is from an excel file or not
 * @returns {boolean} - If the data is valid or not
 */
export function validateJson (data, schema, excel = false) {
  // Prepare ajv and schema
  const ajv = new Ajv()
  const validate = ajv.compile(schema)

  if (!validateStringsInJson(data)) {
    return false
  }

  // Validate data against schema
  if (!validate(data)) {
    // Let user know what is wrong with their data
    const feedbackMessage = excel
      ? `${FEEDBACK_MESSAGES.INVALID_EXCEL_FORMAT} "${validate.errors[0].
        message}" at "${validate.errors[0].dataPath}"`
      : FEEDBACK_MESSAGES.POSTGRES_UNAVAILABLE

    addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
      feedbackMessage)
    return false
  }
}

```

```
    return true
  }
```

Listing 7.6: The function `validateJson()` from the validation module

7.3.3 File Input

The application also validates all data in the files the user tries to upload. The application checks for any invalid characters, if the file format is correct, and if the data type for the different columns are as expected. In addition to validating the data on the front-end, the upload endpoint implemented by NINA does validation for all files it receives. This is important since validation on the front-end alone can easily be bypassed [85, p. 389].

To validate the uploaded files the function `parseAndValidateExcel` is used. The function first checks if the file is defined, if it is of the correct type (XLSX), and if the file is under 10MB, providing the user feedback if any of these are not true. It then uses the function `readFile` from the module `fileHandler` to convert the file the user has selected into an `arrayBuffer`, which JavaScript can read. The `ExcelJS` library is used to read the `arrayBuffer` into a workbook object. Each of the three first sheets in the workbook is converted to JSON, using the function `worksheetToJson` from the module `fileHandler`. All the rows in the sheet becomes objects in a JSON array. The function `validateJson` is then called with the JSON data for each sheet and their schema, resulting in the data format, data type, and the text content being validated.

```
/**
 * Parse and validate if the content of an excel file has valid format and
 * content
 * @param {File} excelFile - The file to parse and validate
 * @returns {Promise<boolean>} - A promise which resolves to if the
 * excel file was parsed and validated successfully
 */
export async function parseAndValidateExcel (excelFile) {
  try {
    if (!excelFile) {
      addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
        FEEDBACK_MESSAGES.NO_FILE_SELCTED)

      return false
    }
    // Check if the file is an excel file
    if (!['.xlsx'].includes(excelFile.name.slice(excelFile.name.lastIndexOf('.')
      ))) {
      addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
        FEEDBACK_MESSAGES.UNSUPPORTED_CONTENT_TYPE)

      return false
    }
    // Check if the file size exceeds 10 MB
    if (excelFile.size > 10 * 1024 * 1024) {
```

```
    ...
  }

  // Read the file
  const fileContent = await readFile(excelFile)
  const workbook = new ExcelJS.Workbook()
  await workbook.xlsx.load(fileContent)

  // River, station and observation sheets
  const sheets = workbook.worksheets.slice(0, 3)

  for (const worksheet of sheets) {
    const jsonSheet = worksheetToJson(worksheet)

    // Check if the sheet name is valid
    if (!excelSchemas[worksheet.name]) {
      addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
        FEEDBACK_MESSAGES.INVALID_EXCEL_FORMAT)

      return false
    }
    // Validate the json sheet against its schema
    if (!validateJson(jsonSheet, excelSchemas[worksheet.name], true)) {
      return false
    }
  }
  return true
} catch (error) {
  ...
}
}
```

Listing 7.7: The function `validateText()` from the validation module

7.4 Sanitization

In addition to validating input, it is important to assume that data is never safe when outputting it to the application, as this is one of the most common web application vulnerabilities, called Software and Data Integrity Failures [57].

The application prevents the output of possible malicious content through Svelte, as Svelte has built-in sanitization [105]. This applies to all the places in the application where JavaScript variables are inserted into the HTML content, which includes the places where environmental data is displayed. This prevents output possibly including HTML tags, CSS styling, or JavaScript to run malicious code. This is because Svelte uses `literalization`, meaning that it never interprets special symbols or text as code [85, p. 390]. Instead text like `<div>` is treated as

plain text. An example where this is used is the component `RiverOverview` (see Listing 7.8). The component outputs the start date, end date, and project ID of a river directly into the HTML. However, as this is accomplished using Svelte's in built `{<output>}` syntax, the output is always treated as text, even if the variables contain malicious code.

```
<script>
  export let river
</script>

<div class='container'>
  <p>{river.startDate} - {river.endDate}</p>
  <p>Projektnummer: {river.projectId}</p>
</div>
```

Listing 7.8: The `RiverOverview` component

The application also sanitizes the URL parameter values when a user selects rivers and stations. This is done by stripping, which is technique that removes harmful characters from the supplied input [85, p. 390]. This is not built into Svelte, and is instead accomplished by stripping away any ID's which are not numbers from the URL parameters using the function `isNaN(input)` (see Listing 7.9).

```
// Add the selected rivers to the URL
if (dataType === DATATYPE_RIVER) {
  selectedRivers.forEach((_, id) => {
    if (!isNaN(id)) {
      url.searchParams.append(DATATYPE_RIVER, id)
    }
  })
}
```

Listing 7.9: How the graph page updates the URL parameters for the selected rivers in the function `updateUrl`

7.5 Encryption

Encryption is used for data being transferred between the client and NINA's server by utilizing HTTPS. HTTPS provides confidentiality by encrypting the traffic using the Transport Layer Security (TLS) protocol [106]. Furthermore, TLS creates Message Authentication Codes (MACs) by hashing the traffic, protecting the data in transit from modifications done by outsiders [107].

Configuring HTTPS should not be handled by the application, as it is agreed that NINA handles this part of the deployment (see NINA meeting 28.02.2024 in appendix P). The Nginx server responsible for serving the Svelte application could have been configured to use HTTPS, however NINA has their own Nginx reverse proxy handling HTTPS and SSL certificates for their domain. See Section 5.1.2 for more detail about NINA's infrastructure.

7.6 Content Security Policy

Content Security Policy (CSP) is important for limiting the resources the application can load. This reduces the risk of cross-site scripting, as only scripts, styling, and other resources from predefined sources are allowed to run. CSP is a HTTP header with one or more directives separated by `;`, where each directive defines the allowed sources for a specific category. For example, the directive `script-src` correlates to JavaScript [108].

The CSP used in the application is specified in the Nginx configuration, and can be seen in Listing 7.10. The directive `default-src` is set to `self`, ensuring that any resource is by default only allowed to be retrieved from the same origin as the application, which is when the protocol, domain, and port number is the same [94]. The directive `img-src` specifies that images can be retrieved from `self`, and that images can be loaded via data URLs [109] from specific URLs, allowing the application to retrieve the map layers. The directive `font-src` is also included to make sure fonts from Google are loaded properly.

No external JavaScript is allowed to be loaded in the website, however to support Plotly and SvelteKit with `adapter-static` (see Section 6.7.1), `unsafe-inline` and `unsafe-eval` is enabled in the `script-src` directive. These are caused by open issues from both Plotly [110], and SvelteKit with the use of `adapter-static` [111] [112]. Because of these issues, the application does not work without these directives. The source `unsafe-inline` allows the execution of inline JavaScript, such as `onclick` and `<script>`. Moreover, the source `unsafe-eval` allows JavaScript to be created from strings, for example by using `eval()`. Both of these are security risks documented in the risk assessment, which are accepted to make the application function as intended, see Appendix N.

The `style-src` directive also allows `unsafe-inline` because of Plotly [113], something that is also documented in the risk assessment. Here `unsafe-inline` allows CSS to be defined inline on the website. The `style-src` directive also allows style via data URLs from Google's CDN, enabling inline styles for defining fonts. Finally, the directive `connect-src` specifies that all fetch requests must point towards the same origin as the application.

```
default-src 'self';
img-src 'self' data: https://*.tile.openstreetmap.org
        https://server.arcgisonline.com https://*.tile.opentopomap.org;
font-src 'self' https://fonts.gstatic.com https://fonts.googleapis.com;
script-src 'self' 'unsafe-inline' 'unsafe-eval';
style-src 'self' 'unsafe-inline' data: https://fonts.googleapis.com;
connect-src 'self';
```

Listing 7.10: The Content Security Policy for the application

Chapter 8

Testing and Quality Assurance

To ensure the application behaves as expected, both under normal usage and under attack, various testing methods are used. Testing and verifying software is a central part of the development process, and is recommended by Microsoft Security Development Lifecycle and during DevOps, as mentioned in Section 3.3 and Section 3.4. Furthermore, testing is also outlined as one of the main countermeasures to manage risk, as discussed in Section 3.5.

This chapter looks at all the testing methods used in the development process, which includes unit and end-to-end testing for checking if the application functions as intended. Testing of the static code is also employed, where linting checks for simple syntax errors and CodeQL checks for known security vulnerabilities. The chapter also covers how GitHub Actions was used to automate these tests.

This chapter also covers how the application was deployed to a test server to let NINA test the application. Moreover, it covers how penetration testing was used on running application to check if the countermeasures implemented protects against common attacks. Lastly, the chapter also looks at how user tests were performed on the application during development to guide the requirements, design, and development of the application.

8.1 Unit Tests

Unit testing is a testing method based on creating small automated tests for individual units of code. The unit tests for the application tests both individual functions located in JavaScript modules, and individual Svelte components. The purpose of these tests is validating that each unit of code behaves as expected, for example checking that a function returns the expected values and that components render correctly. This allows developers to quickly discover if something breaks when adding new features or refactoring existing code. As the application contains several thousands lines of code, having unit tests are essential to check which parts of your code is working, and which parts aren't working, significantly reducing the time used for debugging.

8.1.1 Module Unit Tests

A significant part of the application code is in the JavaScript modules, which contains most of the data managing and logic of the application. Most of the export functions in these modules have unit tests, ensuring that they work as expected. An export function is a function in a module which is available for other parts of the application. If these functions behave as expected, it can be expected that the internal logic of the modules are valid. The modules that do not have unit tests are the upload, authentication, and validation modules. The main reason for this is the time constraints for the project, as the intention was to create tests for all application critical units of code".

The testing framework used is Vitest, as it is compatible with Svelte. Listing 8.1 shows how a unit test is implement for the function `getSelectableSpecies`. The `describe` block groups related tests together. Inside this block there are two unit tests, each inside a `it` block with a description and an actual test. To check the result of the tests, `vitest` uses `expect`. `expect` checks if two arguments are equal, and it uses the syntax `expect(argument1).toEqual(argument2)`. The first test simply checks if the output is empty when no input is given. In the second test a rivers map is created for input, and the test checks if the output is the expected species.

```
describe('test_getSelectableSpecies_function', () => {
  it('should_return_an_empty_array_if_the_input_map_is_empty', () => {
    expect(getSelectableSpecies(new Map())).toEqual([])
  })

  it('should_return_a_unique_list_of_species_from_multiple_rivers', () => {
    const rivers = new Map([
      [0, new River({ name: 'name', species: ['Torsk', 'Ørret'] })],
      [1, new River({ name: 'name2', species: ['Torsk', 'Laks'] })]
    ])
    expect(getSelectableSpecies(rivers)).toEqual(expect.arrayContaining(['torsk', 'ørret', 'laks']))
  })
})
```

Listing 8.1: Unit tests for the function `getSelectableSpecies()`

8.1.2 Component Unit Tests

Svelte components are responsible for the user interface for the application, in addition to also managing logic for the data, states and events. Testing components is harder than function, as the components logic are mostly visual, and may contain limited JavaScript. Most components are tested to make sure they render as the should based on their input. Here `vitest` is also used, with its plugin `@testing-library/svelte`, which is a lightweight testing library for testing Svelte components [114].

Listing 8.2 shows how the function `getSelectableSpecies` is tested. It contains one unit test checking that the component properly adds a valid species, and displays an error when an

invalid species is added. It uses the vitest function `describe`, `expect`, and `it` in the same way as the module unit tests. In addition it uses the `render` function from `@testing-library/svelte` to render components. Rendering a component means that Vitest renders it in a testing environment to DOM as it would have on a web page [115]. After rendering the component, `getByPlaceholderText` is used to retrieve the input we want to test. The function `fireEvent` from `@testing-library/svelte` is then used for simulating user interaction, first by entering a valid species and then an invalid one. By using `getText` the unit tests checks if the correct text is being displayed in the component based on the simulated input.

```
describe('SpeciesInput', () => {
  it('adds_species_to_customSpecies_and_displays_error_for_invalid_species',
    async () => {
      const { getByPlaceholderText, getByText } = render(SpeciesInput, {
        selectableSpecies: ['test_species'], customSpecies: [], chooseAll:
          false })

      const input = getByPlaceholderText('Legg_til_art')
      await fireEvent.input(input, { target: { value: 'test_species' } })
      await fireEvent.keyDown(input, { key: 'Enter', code: 'Enter' })

      expect(getByText('Test_species')).toBeTruthy()

      await fireEvent.input(input, { target: { value: 'invalid_species' } })
      await fireEvent.keyDown(input, { key: 'Enter', code: 'Enter' })

      expect(getByText('Art_finnes_ikke')).toBeTruthy()
    })
})
```

Listing 8.2: Unit tests for the component `SpeciesInput`

8.1.3 Mocking

Mocking is used in testing when components are partly or fully dependent on other components or modules to function.

Mocking involves creating a mock object or function in the test simulating the usual behavior of these other components or modules. This can include a component relying on data from an external source, needing access to a global variables, or using imported functions. By mocking these in a specific, predetermined way, it's possible to ensure that external variables always behaves as expected. This results in the unit test testing the code in isolation, since all external dependencies are simulated into the unit test through mocking. To use mocking in vitest the functions `vi.mock()` and `vi.mocked()` are used (see Listing 8.3). The function `vi.mock()` is used to declare that a certain imported function is mocked. As depicted in Listing 8.3, `get` from `svelte/store` is mocked. In the unit test the `get` function is mocked to return

a specific value by using `vi.mocked(function_to_mock).mockReturnValue(value)`. This allows the unit tests to pretend the function can access the Svelte store, where the store always contains the same expected value.

```
vi.mock('svelte/store', () => ({
  get: vi.fn()
}))

describe('test_addFeedbackToStore_function', () => {
  beforeEach(() => {
    vi.resetModules()
    vi.clearAllMocks()
  })

  it('should_not_add_feedback_message_when_same_type_exists', () => {
    const type = FEEDBACK_TYPES.ERROR
    const code = FEEDBACK_CODES.POSTGREST_UNAVAILABLE
    const message = FEEDBACK_MESSAGES.POSTGREST_UNAVAILABLE
    const currentFeedback = [new UserFeedback(type, code, message)]

    vi.mocked(get).mockReturnValue(currentFeedback)

    addFeedbackToStore(type, code, message)

    expect(userFeedbackStore.update).not.toHaveBeenCalled()
  })
  ...
})
```

Listing 8.3: Mmocking using vitest

8.1.4 Use of AI

AI, specifically GitHub Copilot, was extensively used in the creation of unit tests. It served as an assistant, explaining concepts and libraries, but mostly writing and completing unit tests. Artificial Intelligence (AI) has limitations, most notably when writing complex tests which needs mocking. This resulted in us having to manually write these tests. The main usage of AI was for repetitive tasks, which writing most unit tests were. AI wrote many of the simpler unit tests in a couple of minutes, something that would have taken hours to do manually. We always has to look over the code and possibly correct it as AI makes mistakes, however for writing large amount of repetitive and simple code it was a powerful tool, saving us both time and energy.

8.2 End-to-end Tests

End-to-end tests (E2E) is a testing method used to validate the entire system and how the various parts of the system interact. The main objective of these tests is to evaluate common user interactions in the web application. In this project, E2E tests were created for the list, map, and graph pages. The benefits of E2E tests include verifying whether isolated code pieces function as expected and ensuring that they work together coherently.

8.2.1 Use of Mocking

Applying mocking to E2E tests contribute to manage complexities and limitations associated with dependencies. It isolates the code tested, as it ensures the dependencies of the code always act the same way. Moreover, it also significantly increases the speed of the test, as the dependencies for the E2E tests include accessing external services.

In order to integrate E2E tests into this project, mocking is applied to the data retrieved from PostgREST. This approach made the tests independent from the PostgreSQL database, enabling them to run in an isolated environment and run more effectively.

8.2.2 Implementation of E2E

The library used for E2E testing is `@playwright/test`. It allows you to run isolated end-to-end tests on any browser and platform, providing methods for interacting with and reading the content of the web page [116]. Listing 8.4 displays parts of one of the E2E tests for the application, which tests if the download page behaves as expected. In this example, the `click` and `fill` methods from `@playwright/test` is used to simulate a common and expected user interaction. It checks if the page behaves as expected and that a file was downloaded successfully.

Listing 8.4: Test code for downloading data

```
...
await page.goto('/download')
...
await page.click('text=Rediger')
await page.waitForTimeout(2000)
await page.fill('input[placeholder="Legg_til_Elv"]', 'gaula')
await page.waitForTimeout(500)
await expect(page.locator('text=Gaula_2021-09-30')).toBeVisible()
...
const [download] = await Promise.all([
  page.waitForEvent('download'),
  page.click('.downloadButton_a')
])
const filename = await download.suggestedFilename()
expect(filename).toBe('elver.xlsx')
}, { timeout: 30000 })
})
```

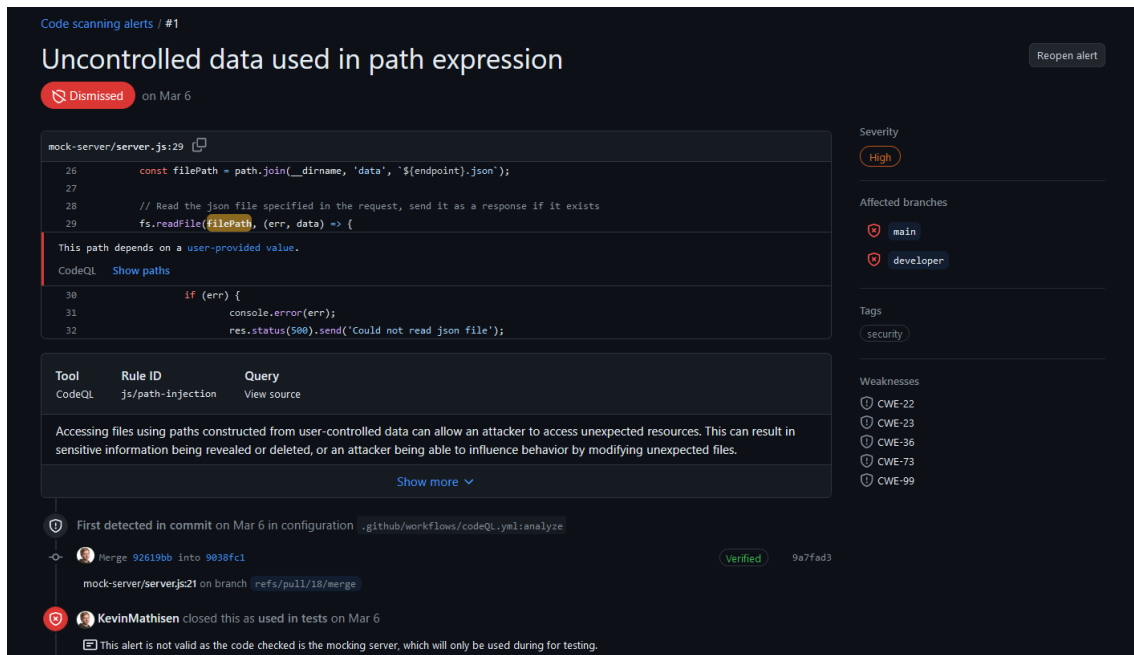


Figure 8.1: Example CodeQL alert for the mock-server in the GitHub Repository

8.3 Static Code Analysis

Static code analysis involves analysing the code without the code running. This mainly includes finding syntax and semantic errors, and detecting poor coding practices.

8.3.1 CodeQL

CodeQL is a static code analysis tool used to find security vulnerabilities in code. It's a code analysis engine provided by GitHub to automate security checks [117]. CodeQL can be configured to automatically run on JavaScript code, and it can provide alerts when potential vulnerabilities are discovered.

To configure CodeQL on GitHub it requires that the repository is set to public. GitHub repository settings has a section called code scanning, and under this section one can enable CodeQL analysis, either by using a default setup or by creating a workflow. For this project a workflow is used. The workflow is configured to run on pull requests to the main and developer branch. If it finds any medium or higher tier of errors or warnings, CodeQL will fail the check for the pull requests.

The alerts from CodeQL are detailed and includes relevant information for the discovered vulnerability. It specifies the vulnerable code, the severity, CWE weaknesses, and explains why it is a vulnerability. When a vulnerability is detected you can resolve it by fixing your code, however it is also possible to dismiss it, as can be seen in Figure 8.1. Here the vulnerability was dismissed because it was related to the mock-server.

8.3.2 Linting

Linting is a static analysis tool that scans code for common syntax mistakes, bugs and errors. Applying linting to the project can prevent common coding and syntax mistakes, while enhancing the robustness and contributing to the readability of the code. This makes the code more maintainable as all code follows the same format.

For this project, the ESLint library is utilized to run lint tests on the JavaScript and Svelte code. ESLint checks the code based on a set of rules in a configuration file. The ESLint library also allows for automated fixing of common syntax errors, saving time from manually having to correct simple styling errors such as indentation. However, more complex issues are flagged and has to be manually fixed. For instance ESLint enforces the rule `prefer-const`, which flags every `let` variable used that could have been a `const`. This has to manually fixed if flagged.

ESLint Configuration File

To configure linting for the project, an ESLint configuration file is created. It specifies that JS-Doc, discussed in Section 3.10.2, should be validated and checked. It also uses StandardJS as the JavaScript styling convention, as this is a widely used standard used by Node.js, NPM, GitHub, and more [118]. The configuration file also specifies that the linting should be compatible with Svelte, ensuring that Svelte files and their syntax is validated.

8.4 GitHub Actions

GitHub Actions is GitHub's implementation of Continuous Integration (CI) and Continuous Delivery (CD). GitHub Actions allows developers to automate the process of testing and delivering code by running workflows on GitHub [49]. Git workflows are automated processes which consist of one or more jobs, such as running tests or deploying an application. To run GitHub Actions the workflows specifies triggers, for example the creation of a pull request on a specific branch.

The project uses several workflows for CI/CD, automating both testing and deployment. There is one running CodeQL, as discussed in Section 8.3.1, where it is ran on all pull request for the main and developer branches (see Section 3.7.5 for Git Branches). There is also a workflow running tests for pull requests for the main and developer branches. It includes a job for running the unit and end-to-end tests, which are discussed in Section 8.1 and 8.2, and a job for running linting, as mentioned in Section 8.3.2. To deploy the code in the main and developer branch to their test servers, as discussed in Section 8.6, two more workflows exist, one for each branch. The result of these workflows is that each pull request automatically runs unit tests, end-to-end tests, CodeQL, linting, where the developers can view if all the checks passed before merging. The workflows also ensure that the latest code in the main and developer branches are ran on their test servers.

8.4.1 Workflow Implementation

The workflow for deploying the developer code to its test server is illustrated in Listing 8.5. The code runs on a pull request to the developer branch, or if it is manually triggered by using

workflow_dispatch. The workflow has one job called update-server running on Ubuntu.

The workflow starts by getting the repository code by using actions/checkout@v4 to clone the repository into the runner. It then uses a secret called SSH_PRIVATE_KEY which is configured on the GitHub page settings/secrets and variables and has access rights to SSH to the developer server. This prevents the SSH key from being public, as it is only available in the workflows by accessing the secret. The job also adds the developer server as a known host, preventing any errors when connecting to an unknown host. Lastly the workflow runs the deploy developer script on the developer server using SSH.

The deploy script rebuilds the Docker compose for the repository after pulling the newest changes from the developer branch. This results in the running application always being up to-date, since it automatically gets updated whenever new features are merged with the developer branch.

```
on:
  pull_request:
    branches:
      - developer
  workflow_dispatch:

jobs:
  update-server:
    runs-on: ubuntu-latest
    steps:

      - name: Checkout Repository
        uses: actions/checkout@v4

      - name: Set up SSH key
        run: |
          mkdir -p ~/.ssh
          echo "${{ secrets.SSH_PRIVATE_KEY }}" > ~/.ssh/id_rsa
          chmod 600 ~/.ssh/id_rsa
          ssh-keyscan ${{ secrets.SERVER_DEV_IP }} >> ~/.ssh/known_hosts
          chmod 644 ~/.ssh/known_hosts

      - name: Update Application
        run: |
          ssh -i ~/.ssh/id_rsa ubuntu@${{ secrets.SERVER_DEV_IP }}
              '/<path-to-repository>/scripts/deploy_developer.sh'
```

Listing 8.5: Workflow for deploying the latest code on the developer branch to the developer test server

8.5 Penetration Tests

According to Synopsys [119], penetration tests, also referred to as a pen-tests, are simulated attacks on a computer system performed with authorisation of the program owners.

The goal behind pen-tests are evaluating the security of the implementation by attacking using common attack techniques. Penetration testers use the same tools, techniques and process as potential threat actors. Once a pen-test is conducted the result is a report pointing out all the discovered security flaws. After analyzing threats in Section 5.5 and looking at risks in Section 3.5, the potential vulnerabilities that was deemed necessary to test were cross-site scripting (XSS), SQL injection and URL injection. These are among the most common vulnerabilities on websites today according to OWASP top ten [57].

The focus of the penetration testing was on the application itself, not on NINA's services. Because of this only the security mechanisms of the web application are tested.

8.5.1 SQL Injection from Web Application

SQL injection attack is a common way of bypassing authentication mechanisms. For instance a hacker could crack, guess or find a real username on the website they want to hack. Assume the following SQL statement to check if username and password is correct:

```
SELECT * FROM users WHERE username = '<valid-user-name>' AND password = '<valid-password>';
```

A potential attacker could then type the correct username followed by a quotation mark and a hashtag, which could modify the query to:

```
SELECT * FROM users WHERE username = 'admin'##' AND password = 'password';
```

If the input is not properly sanitized, this would comment out the required password field, resulting in a successful authentication for the hacker even though the hacker entered the wrong password. In order to protect the application from this vulnerability, whitelisting with a regular expression is implemented as a part of the validation. The whitelisting approach strictly prohibit any use of symbols in the username. This makes it impossible to comment out the password checks, or to use other special symbols to cause weird interactions with the authentication checks. Figure 8.2 shows the result of using invalid characters, and the code that implemented this can be found in this Section 7.3.

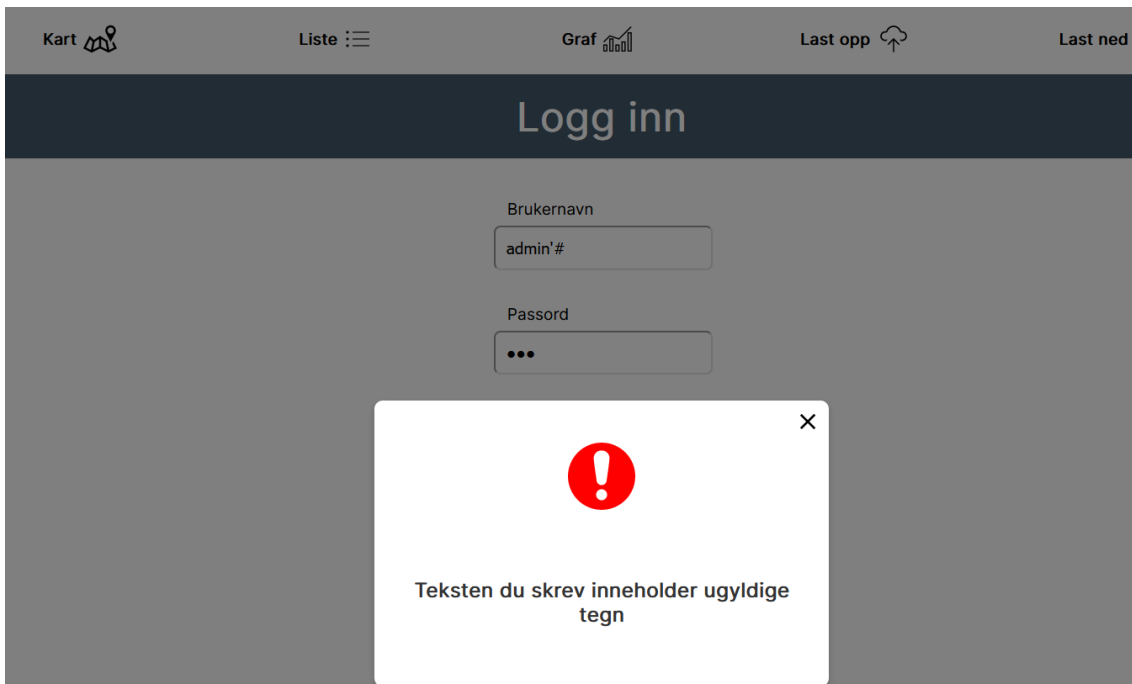


Figure 8.2: Input validation preventing invalid characters from being entered into the user-name field

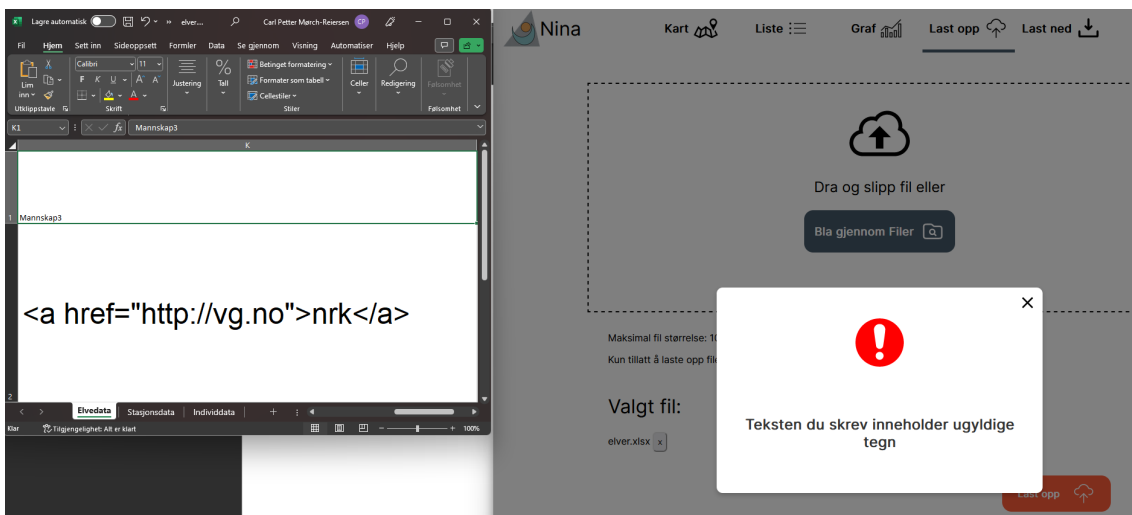


Figure 8.3: Prohibited use of JavaScript syntax

8.5.2 Cross-Site Scripting (XSS)

According to OWASP [120], Cross-Site Scripting (XSS) are types of malicious scripts which are injected into the website. It is performed by a threat actor who uses a web application to

send malicious code in the form fields of browser side scripts. By making use of a HTML code line with the purpose of redirecting the user to another site, it is possible to test if the script is accepted in the search bars. None of the input fields in the application accepts characters tried for cross-site scripting, only resulting in error messages stating that invalid characters were used.

8.5.3 SQL Injection and Cross-Site Scripting (XSS) inside XLSX file upload

Without any security measures a potential hacker could insert either SQL code or XSS tags into the Excel file when uploading a file to the PostgreSQL. This could lead to exploitations of database query mechanisms or the injection of malicious scripts into the web application running on the legitimate users machines. In order to mitigate the risk all uploaded files are validated, as discussed in Section 7.3.3. Figure 8.3 shows that the validation prohibits a file containing JavaScript code, from being uploaded, confirming that the validation of files works.

8.5.4 URL Injection

URL injection is the process of tampering with the URL parameters in order to inject malicious HTML or JavaScript code into the web application. This kind of attack is usually performed on web applications with no proper validation or output encoding present, and for applications which injects user-supplied input directly into the URL. In the application the parameters chosen in the URL specifies the selected station or river ID, which is used by the application to retrieve them from PostgREST. It is therefore important that this input is validated, which is confirmed in Figure 8.4 where malicious HTML code is attempted to be inserted into the URL variables.

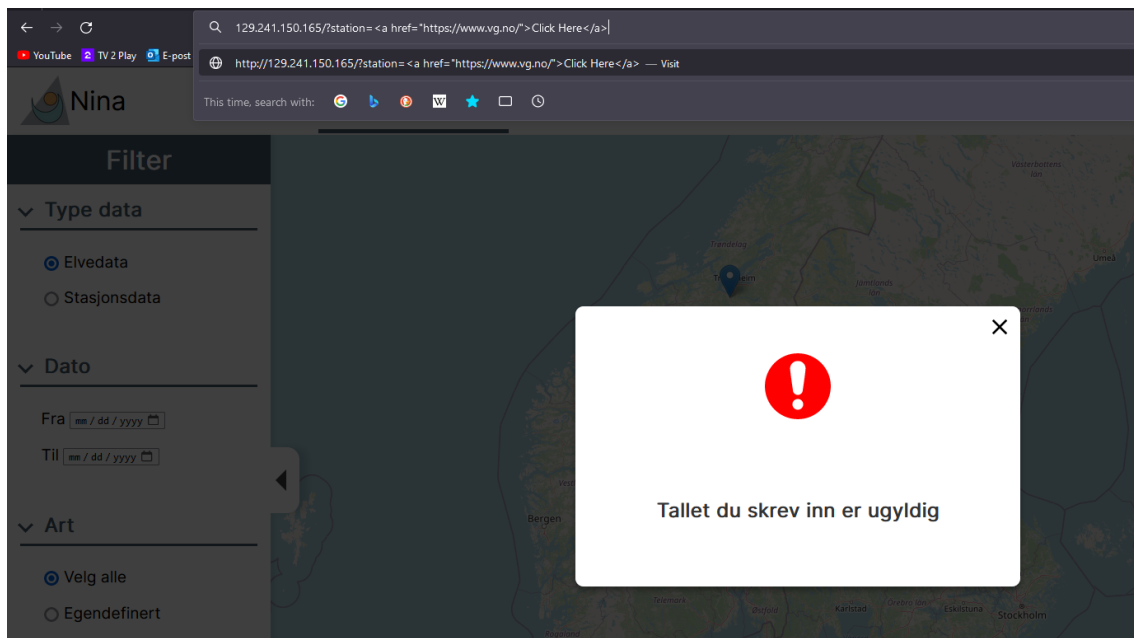


Figure 8.4: URL injection attempt being denied

8.6 Test Deployment

The GitHub repository contains mocking of the services and infrastructure the application is reliant on. This allows the application to be ran during development both locally and on a server without having to depend on NINA. Running the development locally allows developers to quickly test how their changes affect the application when it uses its dependencies. Furthermore, having a server with the application and its dependencies running allows both developers and NINA to access the application under development on a server imitating the real deployment. Lastly, as the application is in a public repository and NINA's services are in a private one, the test deployment enables people with no access to NINA's repository to still run the application using the mock services.

To mock all the services the application requires an Express server mocking the endpoints for PostgREST, upload, and authentication. There is also a Nginx reverse proxy for forwarding request to either the application or the endpoints. To manage these services there is a Docker Compose YAML file in the repository, and each of the services have their own Dockerfile.

8.6.1 Mock Server

The mock server is an Express server mocking the PostgREST, Upload, and Authentication endpoints included in the project repository. To mock the PostgREST data it uses static JSON files that is served as responses to the various endpoints, with data consisting of example rivers and stations provided to us by NINA for public use. The mock server handles authentication by setting the `sessionId` cookie as a constant value and reading this for all requests. The upload endpoint only checks if the user is authenticated, and if so it sends a success message without actually uploading the file.

8.6.2 Test Proxy

The Nginx reverse proxy forwards requests coming into port 80 based on the endpoints. Anything under `/api` is sent to `/api` on the mock server, while any request under `/postgrest` is sent to `/postgrest` on the mock server. Any other requests are sent to the Nginx serving the application (see Listing 8.6).

```
server {
    listen 80;

    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    location /api {
        proxy_pass http://mock-server:8000/api;
    }

    location /postgrest {
```

```
    proxy_pass http://mock-server:8000/postgrest ;
  }

  location / {
    proxy_pass http://frontend:80;
  }
}
```

Listing 8.6: The Nginx reverse proxy configuration for sending traffic to the various services

8.6.3 Docker Compose

To manage the test deployment there is a `docker-compose.yml` in the root of the application repository. The Docker compose file includes the three mentioned services; The Nginx serving the Svelte application, a mock server and a Nginx reverse proxy. The only port open to the outside is port 80 for the reverse proxy. This Docker compose file can be ran on any environment that has Docker and Docker compose installed by running the command in Listing 8.7 at the root of the repository.

```
docker compose up --build -d
```

Listing 8.7: Docker compose command to run the application with its test deployment

8.6.4 Deployment on SkyHigh

To allow the developers and NINA to test the application is deployed onto two test servers. These are named `MainRunner` and `DevRunner`, running the latest version of the code on the `main` and `developer` branch respectively (see Section 3.7.5 for more about the Git branches). The continuous deployment is done by GitHub Actions (see Section 8.4).

These servers are managed and ran on SkyHiGh. SkyHigh is a virtualization platform at NTNU in Gjøvik, allowing students to create virtual machines [121]. Both are running Ubuntu, and are attached to a custom network with two public floating IP address (see Figure 8.5). This allows NINA to access the deployment by using public IP addresses. The configured network has a firewall only allowing SSH and HTTP traffic.

8.7 User Tests

According to HubSpot [29], user tests are a simple way of getting a user's perspective on the product by watching, hearing and reviewing how their interactions with the system went. The user tests are essential to secure that the application meets the desired requirements, contains the requested features, and is user friendly. The user tests are used in the whole development, which includes doing user tests on multiple iterations of both the mockup and the developed application.

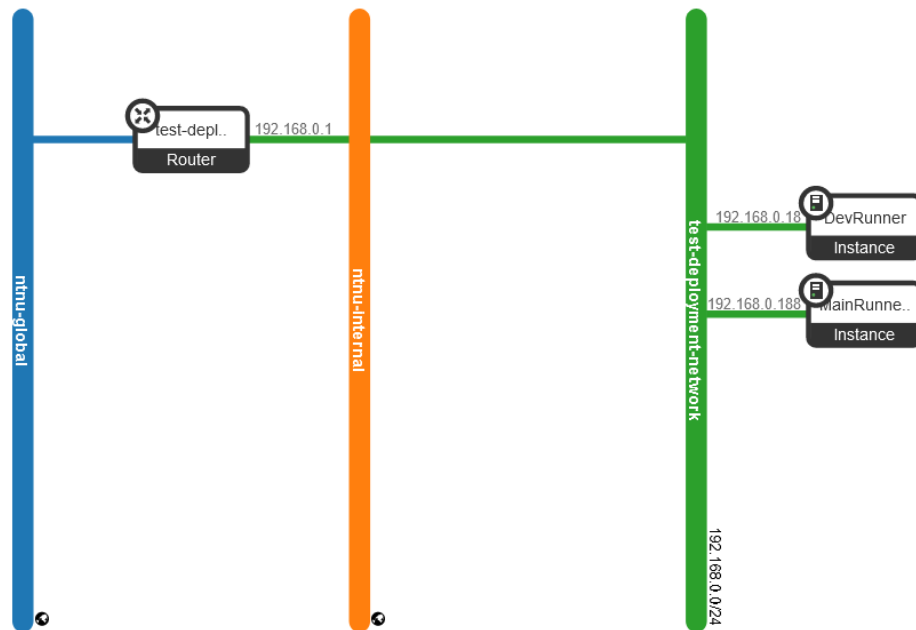


Figure 8.5: The network topology for the test-deployment-network running on Skyhigh

Every user test is documented and suggestions for improvements and changes are noted. Noted changes are ranked to ensure that important implementations and updates are prioritized. This makes it easier to see what work needs to be done and plan how to distribute the workload to implement the changes in the most effective way possible.

8.7.1 User Tests on Mockups

The first user tests are performed on the mockup, outlined in Section 5.4.2. By testing the application early on, one can make necessary updates to the design early before starting the development. This saves both time and resources, where it is easier to update the design in a mockup than potentially change the whole web application implementation. The testing of the mockup includes first testing the initial design, see user test one in Appendix I, where misconceptions of requirements and bad design choices are corrected. The design can then be updated and fixed based on the feedback, where another user test can be done verifying that the new design is satisfactory, documented in Appendix J.

8.7.2 User Tests on The Application

There was also performed tests on the developed application to ensure the web application meets the requirements and is user friendly, documented in Appendix K. Here NINA provided some final feedback, and verified that the application met their expectations and requirements. Some bugs were also found and noted.

Chapter 9

Integration and Deployment

After developing the web application it has to be served to clients from NINA's servers, and correctly interact with NINA's services. This chapter outlines how this is achieved for the application using Docker to package the application, Nginx to act as a web server, and Git Submodules to include the application in NINA's infrastructure.

9.1 Docker

The application is packed in a Dockerfile so that it can integrate into NINA's infrastructure. The Dockerfile includes both the static files for the Svelte application and a Nginx server for serving the static files. This allows NINA to run the Docker container on any platform as long as Docker is installed, making it easy to integrate the application into their infrastructure without having to consider dependencies and the deployment environment.

The Dockerfile has two stages, one for building the application, and one for running the Nginx server. This is done using multi-stage builds [122], as seen in Listing 9.1. The final Docker image built from the Dockerfile only includes the production stage, meaning that only the static build output from the first stage will be kept. The image size is therefore smaller, as all the initial source code and build tools is not included in the final image. Furthermore, having two stages also makes maintenance easier, as it is easy to separate between the build and production stages.

The build stage uses `FROM node:20` specifying that the build of the application uses Node.js version 20. This is a long-term support release that ensures that any faults or critical bugs will be fixed for 30 months [123], which is until April 2026 for version 20. This prevents NINA from having to constantly update the Node.js version, where they instead can simply update to the next long-term support release when available, which happens once a year [123].

The build stage starts by using the `/app` folder as the working directory. The Dockerfile then copies the `package.json` and `package-lock.json` into the build stage, and installs the dependencies these files specify with the Node.js command `npm install`. All the source code is then moved into the build stage using `COPY client/. ..`. Default environment arguments are specified with `ARG env_var_name = default_value`, these are used if any environment arguments are not defined when running the Docker build command. Using the source code,

environment arguments, and installed dependencies, the whole Svelte application is built using `RUN npm run build`. This command runs the command `vite build` under the hood. As explained in Section 6.7.1, the Vite build process is configured to use `adapter-static`. This results in the static build files Nginx serves being placed under `/app/build`. Lastly, in the build stage all developer dependencies are removed using `RUN npm prune --production`, retaining only the dependencies necessary for running the application [124]. This reduces the image size, in addition to reducing the attack surface by removing unused packages.

The second stage is what the final Docker image contains. The production stage uses `FROM nginx:stable-alpine`, which is a container image with a Nginx server running on Alpine Linux. This image is used as the base layer since Alpine is lightweight, both in size and resource usage [125]. The production stage starts by using `COPY --from=build-stage /app/build /usr/share/nginx/html` to include the build output from the build stage into the `/html` directory Nginx will serve files from. It also copies the `nginx.conf` and two custom error pages into the container. The Dockerfile specifies by using `EXPOSE 80` that the Nginx server will be listening on port 80. The Dockerfile specifies the commands to run in the container using `CMD`. It specifies that Nginx will run with `-g daemon off`, ensuring that Nginx runs in the foreground in the container. This is important since it means the lifetime of the container and Nginx service is directly linked [126]. By using the `CMD` command with the `exec` form, where the `CMD` takes arguments as `["executable", "param1", "param2"]`, the Dockerfile executes the command directly [127]. This prevents the command from running in a command shell, reducing the attack surface as there won't be a shell to exploit.

```
# Build stage
FROM node:20 as build-stage
WORKDIR /app
# Copy and install files into the container
COPY client/package*.json ./
RUN npm install
COPY client/. .
# Use default environment variables
ARG VITE_POSTGREST_URL="/postgrest"
ARG VITE_AUTH_URL="/api"
ARG VITE_UPLOAD_URL="/api"
# Build the app
RUN npm run build
# Prune dev dependencies
RUN npm prune --production

# Production stage
FROM nginx:stable-alpine as production-stage
# Copy the build files to the nginx server
COPY --from=build-stage /app/build /usr/share/nginx/html
# Copy nginx.conf
COPY server/nginx.conf /etc/nginx/nginx.conf
# Copy error pages
```

```
COPY server/404.html /usr/share/nginx/html
COPY server/500.html /usr/share/nginx/html
# Expose port 80 and start nginx
EXPOSE 80
CMD ["nginx", "-g", "daemon_off;"]
```

Listing 9.1: The Dockerfile for the main application

9.2 Nginx

Nginx is used as a HTTP server for serving the Svelte application to clients. It is chosen because of its high performance and low memory usage, large community support, and because it's open source [126].

To configure Nginx a `nginx.conf` file is used. It specifies settings for how the Nginx server should handle incoming request and connections, with focus on improving both performance and security. The `nginx.conf` is divided into the main blocks `events`, `http`, and `server`. The `events` block specifies the maximum amount of concurrent connections the server can accept. It can be seen in Listing 9.2. Currently it is set to 1024, as this allows for large amounts of traffic while limiting the traffic from reaching unreasonable levels and overloading the server.

```
events {
    worker_connections 1024;
}
```

Listing 9.2: The event block in `nginx.conf`

The `http` block includes settings for the HTTP server, as seen in Listing 9.3. Some notable settings are `sendfile on;`, which improves the serving of static files by allowing the system call `sendfile` to send files directly from the disk to the network [128]. `tcp_nopush on;` optimizes delivery of data by preventing the sending of partial frames [129], while `tcp_nodelay on;` allows frames to be sent regardless of package size, preventing delay [130]. The timeout for connections is set to 65 seconds with `keepalive_timeout 65;`, allowing clients to reuse connections. MIME types for the `Content-type` header is defined by including the `mime.types` file, which is the way Nginx communicates the datatype in its data, for example JSON or HTML. `gzip on;` is used to enable compression, increasing performance by significantly reducing the size of all transmitted data [131]. `server_tokens off;` is used to hide the server version, preventing attackers from finding server details. The configuration also specifies that access and error logs should be stored in log files, which is useful for monitoring and debugging the Nginx server.

The `http` block also includes settings for various HTTP headers. `add_header X-Frame-Options "SAMEORIGIN";` is set to prevent the page from being embedded on pages from other origins [132]. `add_header X-Content-Type-Options "nosniff";` is set to force the browser to use the MIME type specified in the `Content-type` HTTP header, preventing the browser from guessing the wrong MIME type [133]. The HTTP header `add_header Referrer-Policy "no-referrer-when-downgrade";` is set to prevent the source address of a request being

leaked if the protocol of a request is downgraded from HTTP to HTTPS [134]. Lastly, the HTTP header `add_header Content-Security-Policy "<esp>";` is used to specify the Content Security Policy defined in section 7.6.

```
http {
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    # enable GZIP compression
    gzip on;
    ...

    # set headers for security
    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-Content-Type-Options "nosniff";
    add_header Referrer-Policy "no-referrer-when-downgrade";
    # content security policy
    add_header Content-Security-Policy "default-src 'self';    img-src ...";

    # disable server tokens to hide server version
    server_tokens off;

    # logging files
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    server {
        ...
    }
}
```

Listing 9.3: The HTTP block in `nginx.conf`

The `server` block defines how the Nginx will serve the files and manage simple errors and can be seen in Listing 9.4. It specifies the port number it will run on, which is 80 for HTTP. `root /usr/share/nginx/html;` specifies that all the files Nginx will serve is placed under this directory, and `index index.html;` specifies that the default file to serve is `index.html`. `index.html` is served as default because it is the fallback page for the Svelte application. The fallback page is the entry point for SvelteKit applications [135], which in this project configuration represents the whole single page application, ensuring that the whole Svelte

application is served by default to clients.

The location `/` block under `server` tries for all requests to find the file located at the `$uri` first, for example the server will try to send the file `/usr/share/nginx/html/style.css` for a request to `/style.css`. If the file was not found, it checks if the file should be a directory instead and tries to serve this using `$uri/`. If no file was found, the homepage is returned with `/index.html`, and if the homepage is not found, the server returns a "404 Not Found" error. Next, the configuration defines that the error pages `/404.html` and `/500.html` will be served for the error messages 404, and the internal error 500 group range, respectively. These pages should only be accessible if an error occurs, where users cannot access these files directly since they are set to `internal`.

```
server {
    listen 80;

    # root directory
    root /usr/share/nginx/html;
    # index file
    index index.html;

    # serve svelte
    location / {
        try_files $uri $uri/ /index.html =404;
    }
    # error pages
    error_page 404 /404.html;
    location = /404.html {
        internal;
    }
    error_page 500 502 503 504 /500.html;
    location = /500.html {
        internal;
    }
}
```

Listing 9.4: The server block in `nginx.conf`

9.3 Integration with NINA's Infrastructure Using Git Submodules

To run the Dockerfile containing our application, NINA uses Docker compose together with Git submodules. This allows NINA to include the application repository, which contains the source code and the Dockerfile of the application, directly in a subdirectory in their own repository.

9.3.1 Git Submodule

A Git submodule can be added to a repository using `git submodule add <project_url> <target_directory>` [19]. This creates a directory with the submodule and a file called `.gitmodules` containing the path and URLs to the submodules in the repository. Running the command `git submodule update --remote` gets the latest changes to the submodule. After the submodule is added or updated the repository needs to track its state, which can be done by committing the updated submodule. This does not track the submodule files, but rather git recognizes it and treats it like a submodule, where it tracks which commit in the submodule the repository is using. The result is that other Git repositories can be placed inside of a Git repository, allowing it to use the up to date code from it's submodules without having to manually combine them, ensuring that each repository is responsible for their code.

9.3.2 Docker Compose

In the `docker-compose.yml` of NINA's repository they can include the Dockerfile in the Git submodule directory as the front-end service. The configuration for the front-end service is shown in Listing 9.5. This lets the Svelte application be served in its Docker container, while placing the Svelte application inside of the Docker compose containing the services it depends on. These services includes a PostgRESTservice for exposing endpoints for environmental data, PostgreSQL for storing the data, Django for authentication, an upload endpoint, and a Nginx reverse proxy for routing incoming requests to the various services.

To specify the endpoints for the services, the `docker-compose.yml` file declares their values as illustrated in Listing 9.5. This is only necessary if the default values in the Dockerfile are not correct (Default values are discussed in Section 9.1).

```
services:
  django:
    ...
  postgres:
    ...
  postgrest:
    ...
  frontend:
    build:
      context: ./frontend
    # Optional arguments for specifying endpoints
    args:
      - VITE_POSTGREST_URL=<postgrest_endpoint>
      - VITE_UPLOAD_URL=<upload_endpoint>
      - VITE_AUTH_URL=<auth_endpoint>
  proxy:
    ...
```

Listing 9.5: NINA's docker compose

Chapter 10

Discussion

During the whole development process various decisions affecting the development process and the final application were made. This chapter covers the reasoning behind the choices made and looks at possible alternatives. It also evaluates the group work, use of AI, and the sustainability of the application.

10.1 Development and Collaboration

10.1.1 Pinning of Dependencies

Dependency pinning has advantages and disadvantages. The main advantage is that it simplifies the dependency management, making it easier to use their functionality, since the dependencies don't change. This means you do not need to update your usage of functions and review the documentation when new versions are released. Dependency pinning also prevents the application from possibly breaking when dependencies update. The application breaking would be troublesome for NINA, as fixing these errors requires an understanding of how the dependencies are used in the application, something NINA lack. They do have documentation and the source code of the application, however they prefer to not use resources on having their internal IT-team learn how the application works, which they can mostly prevent by using dependency pinning.

The disadvantage of not updating dependencies is mainly the security risks involved. Dependencies might become outdated with known vulnerabilities where only the newer version are patched. By not updating, the application might contain these vulnerabilities. An example is the vulnerability CVE-2021-44228, which is a vulnerability affecting Log4j, a popular Java logging framework. It allows attackers to execute arbitrary code by controlling log messages or log message parameters and is given a severity score from NIST of 10.0 critical [136]. This vulnerability was only for certain version of Log4j, and there was a patch released a few days after the vulnerability was discovered that patched it [137], showing the benefit of having dependencies automatically updated.

The main reason for the application using dependency pinning, as outlined in Section 3.6.4 about dependency pinning, is to follow NINA's requirements. They deemed it better to

use dependency pinning, as they don't want to fix the application if updated dependencies break the system, even with the potential security risks.

10.2 Requirement and Design choices

10.2.1 Choice of Rendering Architecture

In Section 5.1.1 explaining the Svelte application architecture, it is outlined that the application utilizes client-side hydration in combination with being a single page application. An alternative to this would be to use server-side rendering. Here the server delivering the web application would also be responsible for integrating the application with the other services. The server could authenticate the user before sending the application, possibly increasing security as the user does not have to first get the web application, then be authenticated. It will also let the server handle the retrieval and processing of environmental data, reducing the load on the client browser, where the user will get the website with the data already inserted into the HTML. Furthermore, the website would load faster, as the user does not have to load the entire web application at once, but rather only the page they want to visit. By sending the client pages where the data and interacting with services are already done, it could allow the web application to not require JavaScript, supporting progressive enhancement as discussed in Section 10.3.4.

Even though there are advantages with server-side rendering, we still went for client-side hydration and single page application. The use of client side hydration allows the website to be served as static files, where the server does not need to process and render the web pages before sending them, reducing the server load and increasing scalability. Furthermore, making the web application a single page where it loads data when needed allows the user to navigate and interact with the web application without reloading the entire page. For example, when a new river is selected or the user want to see a river in the graph page the web application does not need to reload, resulting in a more responsive user experience. These factors outweighed the potential advantages of choosing server-side rendering. A more interactive web application with a lightweight server was seen as more desirable, even if this came at the cost of progressive enhancement, increased client-side processing, and the need to handle authentication on the front-end.

10.2.2 Use of Design Principles

Design principles may sometimes be contradictory when designing software since the developers have to prioritize and balance the various principles. An example taken from Section 5.2 is that when uploading data to the server, the files are rejected if it has invalid format, datatype, or content. One could argue that the Fail Securely principle states that when a validation fails, the error should expose as little information to the user as possible. At the same time the principle of Psychological acceptability may state that the user should be informed of what went wrong, so the user can easily fix the problem and upload their file. In this case we chose to let the application inform the user of the format errors, while at the same time making sure they to not expose any sensitive information.

10.3 Coding Choices

10.3.1 Choice of Front-end Framework

When starting the project we considered developing the web application in either vanilla JavaScript, React or Svelte. Vanilla JavaScript was first considered because this is what the team has previous experience using, and we didn't think the application would need a lot of JavaScript code.

We quickly realised that vanilla JavaScript would not be optimal for the project. The amount of JavaScript code needed for the application was more than initially expected. Writing it all manually instead of leveraging existing technologies provided by JavaScript frameworks would result in a lot more unnecessary work. Furthermore, using other frameworks would follow the design principle Leveraging existing components 5.2.9. The code we would have to write with vanilla JavaScript include custom Document Object Model (DOM) manipulation, state management, and reactivity, plus there would no support for reusing components.

Next we considered React and Svelte. Both of these frameworks are component based, allowing developers to reuse code effectively, while also having functionality for managing state and reactivity [138] [139]. The advantage with React is its large ecosystem with millions of developers [140], offering easy access to resources, documentation, and large amounts of libraries for various use cases. Svelte in contrast, is newer and not as popular, however in 2021 it was chosen on a public poll on stack overflow as the most loved web framework [141]. Svelte is compact, using HTML, CSS, and JavaScript similar to native JavaScript, making it easier to learn for new developers. Furthermore Svelte optimizes the JavaScript at compile time, resulting in highly optimized vanilla JavaScript. It does not utilize a virtual DOM such as traditional frameworks like React, but rather Svelte compiles code to use the actual DOM, resulting in faster rendering and less memory overhead [142].

We ended up choosing Svelte because of several reasons. As Svelte is less established than React, it enables us to write a thesis on a more unexplored topic. It is also a modern and efficient framework, making it easier to create a quick and responsive application, while being sustainable by minimizing usage of resources. Lastly, we also chose Svelte as it seemed easier to learn since it is more alike regular HTML, CSS and JavaScript compared to other frameworks like React.

10.3.2 Use of SQL Views

When planning how to retrieve the environmental data from PostgREST we were considering either using PostgREST endpoint with parameters to access the database tables directly, or access them via custom SQL Views. Using PostgREST to access tables directly would involve using the URL parameters to do many operations one can do in SQL. This includes selecting specific attributes to retrieve, filter and order data [93], and use resource embedding to combine data from multiple tables [143]. The main problem with PostgREST is that it doesn't allow you to filter out unique values.

Being able to retrieve unique values was desired, as the web application should allow users to filter all river and stations on their species. Having the data include the unique species would therefore prevent the web application from having to manually go through each observation

for rivers and stations, and calculate the unique species in them, each time it revived the data. In addition, including the unique species lets the web application not need the observation data when retrieving all river and stations. Instead, this can be fetched when a specific river or station is chosen to look at in detail.

SQL views, in contrast to accessing table directly with PostgREST, would allow the database to only expose the data the application needs. They can define the exact format the data needs to be, including which attributes to include, order of data, joining tables, and calculating unique values. All of these features were used when defining the SQL Views for the application, as mentioned in Section 6.2.1. This follows separation of concern, as the web application only has to retrieve data directly from the SQL view endpoints, instead of being responsible for doing operations and construct the data in the PostgREST query. SQL views can also increase performance significantly as they can be stored as materialized views [144], preventing data from being calculated for each request.

We decided to go for SQL views Because of its features and the increase in performance, offloading computation heavy tasks from the web application.

10.3.3 Caching

To increase the performance of the website and decrease load times, we considered using caching. Caching enables requests for the website and environmental data to be temporarily stored, reducing load times for similar requests. One problem with caching is the increased complexity with dealing with authentication, as the cache will have to know if the user is authentication before it sends the cached data to the user. Furthermore, as we optimized the Nginx serving the web application to serve static files, it functions much the same way as a cache would, were it serves files directly for request without any processing. Using SQL views without having to recalculate the data for each request also serves to reduce the processing of each request. This allows PostgREST to serve the PostgreSQL data with minimal processing, decreasing the need for a cache. Due to the increased complexity for authentication when using caching, as well as the web server and database already being optimized, we decided to not implement caching.

We also considered to use `localStorage` to store the environmental data fetched in the web application between sessions, reducing the need for fetching data for each page load. `LocalStorage` is a way for websites to store data with no expiration date in the browser [145]. This does however creates problems when the environmental data in the database is updated. For example, when a user uploads new data, the web application would have to wait for its data in `localStorage` to expire before retrieving the updated data. Instead we chose to let the web application effectively fetch only the data it needs and store it in the memory of the web application. This prevents the data from being re-fetched while the user uses the web application, while allowing the user to reload the website to retrieve the up-to-date data.

10.3.4 Deprioritization of Progressive Enhancement and Mobile-first Approach

The modern approach for web design is making websites Mobile-first, meaning websites should be designed for mobile devices first, then build the desktop version on top of this [146]. This is to ensure that websites are optimised for mobile devices first, as these have less computation

power and smaller screens. Progressive Enhancement means that websites should have a basic functional version available for simple and old devices, while adding newer and more advanced features for more modern devices. For websites this is usually done by making the website function without JavaScript, while using JavaScript to add functionality on top. There are several reasons for why users might not have access to JavaScript, including the website isn't loaded correctly, HTTP request for JavaScript failed, corporate firewalls blocking JavaScript, or web browser data savers being turned on [147].

When developing our web application we considered following both Progressive Enhancement and Mobile-first, however NINA explicitly stated that the website does not need to be Mobile-first as they will only use it through desktops, see NINA meeting 11.01.2024 in appendix P. Furthermore, as the website's whole purpose is to display the environmental data dynamically, JavaScript is required to retrieve, manage, and display the data. Without JavaScript the website would just be an empty page with no data to display. This could be fixed by using server-side rendering, such that the website loaded by users would already have data injected into the HTML, reducing the need for JavaScript. In the end we didn't choose this option because of the reasons outlined in Section 10.2.1.

10.4 Security Choices

10.4.1 JSON Tokens versus HttpOnly Cookies for Authentication

When choosing how to manage access tokens for authentication we were considering either using JSON tokens or HttpOnly cookies. If the web application were to use JSON tokens, the access tokens would be sent by the authentication endpoints in their body as JSON. This would then be managed by the JavaScript in the web application, which would allow us to gain more control over how it is stored and used. However, it would also expose the token to cross-site scripting attacks, as it is available through JavaScript. Using HttpOnly cookies on the other hand, as outlined in Section 7.1, would protect the cookies from being accessed by JavaScript. Instead the browser would manage the cookies, including storing them and sending them with requests. This follows both separation of concern, and the design principle leverage existing components explained in Section 5.2.9. In the end we ended up choosing HttpOnly Cookies because of the increased security and reduction in application complexity.

10.4.2 Too Lax Content Security Policy

When configuring the Content Security Policy for the application, as seen in Section 7.6, we ended up having to use `unsafe-inline` and `unsafe-eval` for JavaScript because of Plotly and SvelteKit, specifically when using SvelteKit with `adapter-static`. This is considered a security risk as it allows potential attackers to run injected JavaScript on the website. Unfortunately we did not learn of this problem with Plotly and SvelteKit before well into the project, and it was then decided that it was too late to change the technologies and the configuration of the application. If we had known this before choosing Plotly, we might have decided upon another plotting library. Moreover, we could have changed the application from using Nginx to serve files to use SvelteKit's built-in server, as this would allow SvelteKit to configure the Content

Security Policy itself. When configuring the Content Security Policy within SvelteKit, it can use hashes to allow inline scripts it generates to be allowed, while limiting any other scripts, removing the need to use `unsafe-inline` and `unsafe-eval` [148]. The use of SvelteKit's built-in server would involve using `adapter-node`, which is discussed in Section 10.5.1.

10.4.3 Dropping Sanitization of Downloaded File

One of the countermeasures, outlined in Section 7.4, was sanitization of output in the web application. This is done for both output on the web page and the URL fields. However, sanitization was not done for the files downloaded, which are one of the outputs of the application. This was mostly caused by a lack of time and resources, where other countermeasures and features were prioritized. In addition, the downloaded files were made up of the data from PostgREST, which is always validated before usage. Still, as outlined by the design principle defense in depth in Section 5.2, software should not rely on a single layer of security. The missing sanitization of downloaded file is a clear violation of this principle and should therefore have been implemented.

10.5 Integration and Deployment Choices

10.5.1 Choice of Website server; SvelteKit, Express, or Nginx

For serving the web application we considered multiple options. SvelteKit has an adapter called `adapter-node`, which serves the application using a node server [149]. This was considered because it is simple to set up, using SvelteKit's built-in functionality and keeping all the development within the Node.js ecosystem. It also allows to customize the server-side logic such as CSPs. However, as the web application is only static files, the adapter `adapter-static` was instead chosen [150]. This generates static files, which can be served by using a file server.

To serve static files we first considered Express. We have previous knowledge working with it, it is well supported and widely used in the Node.js community, and it would allow us to use middleware for logging, compression, HTTP headers, and more [47]. However, for the reasons mentioned in the Nginx Section 9.2, notably its high performance, we ended up using Nginx.

10.5.2 NINA's Solution Changing Throughout The Project

NINA's infrastructure and services which the application depended on changed several times throughout the project. The first version of the database structure was shown to the group February 28th, one month into the project, where NINA's authentication and upload was not yet completed and they considered using an application called Keycloak for authentication, see NINA meeting 28.02.2024 in Appendix P. April 3rd we learned in a meeting that NINA are planning to use Django as the authentication and upload solution, but they have not yet completed their configuration, see NINA meeting 03.04.2024 in Appendix P. April 10th we were shown how Django would work, and the endpoints were mostly done, see NINA meeting 10.04.2024 in Appendix P. At April 17th NINA's part of the infrastructure were finalized,

where we could finally complete our part of the integration, see NINA meeting 17.04.2024 in Appendix

The changing of NINA's solution, which the application depended on, resulted in more work during the development process. Both the code of the application handling the interacting with NINA's services and the handling of the environmental data had to be modified and updated based on NINA's evolving solution. Moreover, this also required updating documentation, diagrams, requirements, and the risk assessment. Even though this caused more work being required for the development, the environment data format remained mostly unchanged, which allowed us to build the web application based on these assumptions even with changing dependencies. Furthermore, NINA continuously updated us on their progress and status, where they were available through bi-weekly meetings and email, quickly answering our questions. This allowed us to factor their status into which parts of the development we focused on, resulting in evolving dependencies not halting the progress of the application.

10.6 Not Implemented

10.6.1 Updating Threat Model

After Threat modeling was completed in the early stages of the project, as outlined in Section 5.5, it was only partially updated to match the evolving design and risk profile of the web application. Optimally, as the project followed Scrum, we would have kept all documentation up to date with the state of the development. However, because of time constraints the threat model was deprioritized in favor of keeping the risk assessment updated, as the risk assessment was seen as more important.

10.6.2 Missing Website Features

By the end of the project there were several desirable features, although not required, which were not implemented. Ideally users should be directly redirected to the login page when not authenticated instead of always being sent to the home page. We also wanted to add a redirect to the home page after an upload or login was successful. The website also could have displayed the username of any logged in users in the header. The feedback for uploading data with invalid formats could have been formatted better and in a less verbose nature. The plots on the graph page have trouble displaying pie charts when a large amount of rivers, stations, or species are selected to plot, where Plotly could have been configured to handle this better. We also wanted to add the option to change between Norwegian and English, as well as adding dark mode and light mode functionality.

10.6.3 Performance Tests

When verifying the website we intended to perform some performance tests. Performance tests are tests which simulate real usage of an application to find its bottlenecks, trying to stress it until it breaks [151]. Performance tests are also important since they might help prevent application failure due to performance-related problems. The web application must be able to

handle all the environmental data from NINA, which includes hundreds of rivers, thousands of stations, and more than hundred of thousands of observations. This data will also continue to grow, meaning ideally the application should be able to handle five to ten times this amount without problems, see NINA meeting 28.02.2024 in Appendix P. Because of this we wanted to test how the Nginx web application server, PostgREST, PostgreSQL, and Django would handle large amounts of users, especially with large amount of environmental data. The web application itself would also have been tested with large amounts of environmental data, trying to find the limit of how much it can process without significant performance issues. Unfortunately we did not have time to test this with the time allotted for the development and testing of the application.

10.6.4 Penetration Test of Deployed Web Application

The penetration test outlined in Section 8.5 only tests the security mechanisms developed for the web application. To do a comprehensive penetration test it should have been done on the application deployed with NINA's services. This would result in any possible vulnerabilities with NINA's services, or their interaction with the web application, being uncovered. The vulnerabilities found could then be brought to NINA's attention, increasing the security of the web application. Furthermore, the penetration test which were done was only a confirmation of the countermeasures working as intended, where no new attack vectors not previously considered were tested.

The reason for the limited scope of the penetration was limited time for the thesis and project. If more time had been available a more in depth penetration test would have been done, possibly updating the risk assessment and information NINA of vulnerabilities.

10.7 Evaluation of Group Work

The collaboration between the group members have been great throughout the entire project duration. All group member actively attended the daily scrum meetings and the meetings with the supervisors and NINA. If there were any complications with attending a meeting, a message was sent early enough to alert and potentially reschedule the meeting to a more suitable time. The daily sprint meetings allowed all members to ask for help when needed, and address issues. The active use of the scrum board on Jira was an important tool to list the current tasks, to visualize the progress on our work, and distribute the work between group members, as seen in Section 3.2.

The distribution of group work was mostly fair. Everyone in the group followed the rules outlined in Appendix C, which included working 30 hours in the bachelor each week. There were some weeks where this was not followed, because of other assignments, sickness, or travel, however this was always approved by other group members. Furthermore, there was also some group members who worked more hours than others, as can be seen in Appendix E, however the extra work was done voluntarily and did not cause any issues in the group.

This thesis has been a fantastic learning period for group work. The process of choosing, planning and setting up boundaries, distributing the workload, and developing the actual application have brought rememberable and useful insight in both project management and

execution. It has developed our understanding in the importance of good teamwork, and in working together to achieve a set goal.

10.7.1 Time Spent Learning versus Working

One aspect of the group work which can be noted is the time spent learning new technologies versus actually working. Because none of the group members had previous experience with Svelte, and several group members had little previous experience programming outside of simple school projects, therefore there was a large learning barrier to overcome before starting to develop the application. This resulted in having to spend almost two weeks to learn Svelte and other technologies used before starting developing the application. This is in addition to the time spent repeating coding and web development concepts before the official project start. Allocating time for learning in the group work was necessary, where it allowed us to more effectively develop the application once we started.

10.8 Use of AI and GitHub Copilot

Artificial Intelligence, specifically ChatGPT GitHub Copilot was used throughout the project for coding assistance. All group members had the GitHub Copilot extension installed in their developer environment, which provided it direct access to our code. This allowed it to suggest code in real time, such as completing a function or writing a comment. It did not write our code, but rather suggested ways to complete what we started. This contributed to save time, in the instances when the AI correctly assumed the purpose of the code, and managed to come with good suggestions. However, it also suggested code which were not correct, which could be distracting as you would lose your train of thought by reading its suggestion. Because of the possible inaccuracy of the AI, we always read the suggestions completely before accepting it, and a lot of times we decided it would be easier to ignore it and just finish the work ourselves. Overall, it was still useful, especially for writing repetitive and simple code, for example when writing unit tests (see Section 8.1.4 for use of AI with Unit tests).

ChatGPT on the other hand was used more for questions and answers, rather than direct suggestions for code. It was used much the same way as Stack Overflow and Google, where we tried to find answers for specific coding problems and errors encountered during development. The advantage of ChatGPT when comparing it with Google, is its ability to answer your specific problem extremely quickly, while also answering any follow up questions. Also since Svelte is not amongst the most popular JavaScript frameworks, it was limited what results you could get by using a google search. In many cases, there wouldn't be an answer to a specific question we had. One of the main drawbacks, similar to GitHub Copilot, is its ability to be wrong. But, as it was used for coding related questions, it was easy to know if it was wrong as it would be reflected when you tried to run its solutions. In these cases we had to resort back to Google and websites like Stack Overflow, doing more in depth research, which were more sufficient for solving complex problems. Much like GitHub Copilot, ChatGPT was useful and efficient for answering more simple questions and their follow up, while we still had to do our own research for more complex problems.

10.9 Sustainability

This thesis contributes to sustainability in multiple ways. NINA is an independent foundation researching how human activities affect the environment [1]. The web application developed supports NINA's environmental research by enabling them to do their research more effectively and with less errors when handling their data. The web application therefore indirectly contributes to multiple of UN's sustainability goals, as specified by FN.no [152]. This includes goal 6, clean water, and its sub-goals 6.3 and 6.6, where NINA's research help protects water ecosystems, and discourage pollution and waste dumping in rivers. Furthermore, NINA publish research about the effects of global warming on the climate, supporting goal 13, stop climate change, and its sub-goal 13.3, which is about strengthening individuals knowledge and awareness of around climate change. The web application also directly supports goal 9, industry, innovation, and infrastructure, and its sub-goal 9.5, because the web application strengthens scientific research.

The web application also contributes to sustainability by being secure. It decreases the likelihood of NINA's research being manipulated, which helps them do correct research, contributing to the goals mentioned. Furthermore, the web application's security also protects them against economic loss from potential attacks, supporting economic sustainability.

In addition to supporting sustainability goals, the web application is built to be effective, reducing its environmental footprint. The application is deliberately built using lightweight and efficient technologies, such as Svelte and Nginx. It also only requests data it needs from the database, reducing the bandwidth used to serve the application. This helps the web application consume fewer resources when running on the server, when it sends data over the internet, and when the application runs in the users web browser.

The application requires running multiple servers, consuming resources that weren't previously used, as NINA previously only had to use local Excel files for the same functionality. Furthermore, the application will require network traffic and bandwidth to send the application and data, and require computing power on the users computers. This goes against goal 12, which is about making sure to use as little resources as possible, resulting sustainable consumption of resources. However, it could be argued that the benefits of streamlining NINA's research outweigh the downsides.

Chapter 11

Conclusion

Building upon the previous chapters, this chapter concludes with summarizing what was achieved in the project and how this measures against the original goals. Moreover, the conclusion also covers possible improvements of both the thesis and project, and discuss potential further work.

11.1 Results

To evaluate the results of the thesis and its project, the results are compared against the goals outlined in Section 1.4. This includes looking at the result of the project, it's effect on NINA, and what was learned by the group.

11.1.1 Result Goals

The development process resulted in a secure and user friendly web application which fulfilled all the requirements outlined by NINA. It has an interactive map and a search page for viewing observation points, and allows the user to select these to view their data in more detail. The user can also select multiple river or stations to view their data plotted as bar charts, pie charts, histograms, and box plots. Furthermore, the web application allows users to both upload and download data with a user friendly interface. This satisfies result goal 1.

A Dockerfile was created containing a Nginx web server serving the web application. It can be deployed in NINA's infrastructure with no modification or upkeep. The Dockerfile integrates with NINA's services, using agreed upon API specifications, where NINA only has to insert the predefined SQL views into PostgreSQL to configure the PostgREST endpoints. They can further specify the service endpoints using the Dockerfile's environment variables. However, the endpoint values are by default compatible with NINA's services. To run the Dockerfile NINA includes the web application project repository as a sub-folder in their own repository, using Git Submodules, where the Dockerfile is used as a service in NINA's Docker compose. The result is that NINA can easily deploy the web application without configuring it, where it seamlessly integrates with their infrastructure, satisfying result goal 2.

To allow NINA's internal IT team to easily take over the upkeep of the web application, comprehensive documentation was created. This includes commenting of code, diagrams ex-

plaining the application logic, documents describing the APIs, overview over dependencies, and a user manual. This satisfies the result goal 3.

Security was considered throughout the whole development process. The methodologies Microsoft Security Development Lifecycle and DevSecOps was followed, which outlines several processes and practices to follow in each stage of development. This resulted in the development process including continuous risk assessments, requirement engineering, threat modeling, and both static and dynamic testing of the application. These tests included linting, security scans, unit tests, end-to-end tests, penetration tests, and user tests. Considering security throughout the development process resulted in the final web application following secure design principles and having several countermeasures. These countermeasures include using proper error handling, implementing input validation, implementing output sanitizing, configuration of a content security policy, and handling session tokens securely. However, there was some countermeasures not implemented completely. The content security policy allows JavaScript to be ran from unsafe-inline and there is no sanitization of downloaded files. Still, the risks of the application are clearly documented in a risk assessment, and the web application has countermeasures in place protecting against possible threats, so we consider result goal 4 mostly satisfied.

11.1.2 Effect Goals

Feedback given by NINA in user test two J, confirms that they are satisfied with the result and functionality of the application, highlighting its ease of use. They are confident that the website increases their productivity when working with their environmental data. Viewing and visualizing their data is made easier by being able to view the data points in an interactive map, or search for them. Furthermore, the application lets them view and compare the rivers and stations, and their underlying observations, in tables and graphs. This fulfills effect goal 1.

The application also allows users to download files, where the user can select multiple river or stations, and which species to focus on. This helps the scientists test hypotheses easier as they can easily aggregate the data they are interested in for further analysis, fulfilling effect goal 2.

11.1.3 Learning Goals

We gained extensive knowledge and experience working with the development of the application. The whole project involved regular meetings with NINA, where we had to ensure our work aligned with their goals. This helped us learn how to interact with an employer in a professional setting, while working on a large project, satisfying learning goal 1.

The project let us learn how to use SCRUM over a longer time in a small group, all while working towards a specified task, satisfying learning goal 2.

To develop the web application the group members had to learn how to use several technologies. These include using Svelte and SvelteKit, and utilize libraries such as Leaflet and Plotly. It also includes learning how to interact with external sources for terrain and satellite maps, and interact with PostgREST and Django from NINA. This satisfies learning goal 3.

The group members had previously done risk assessments, but not while developing software, least of all with continuously evolving requirements and design. The project therefore

gave us experience in doing risk assessments while developing software, where we learned how to identify risks in the various stages of development, how misuse cases and threat modeling can assist in identifying risks, and the importance of identifying risks early on. Furthermore, we also learned how to handle risks by following secure design principles and implementing countermeasures. This experience satisfies learning goal 4.

During the project Git and Jira was used to support the development. We learned how to use Git and GitHub in a large repository, including the use of branches, GitHub Actions, and pull requests. We also learned how Jira could be used to track work and issues, and organize these in sprints. This gave us experience in working with development tools in large projects, satisfying learning goal 5.

Lastly, the project used continuous integration and continuous deployment (CI/CD) for testing and deploying the software. This gave us experience in working with GitHub Actions and scripts to configure CI/CD. We also learned how to configure unit and end-to-end tests using Vitest, ESLint for linting, and CodeQL for security scans. This satisfies learning goal 6.

11.2 Possible Improvements

There are some aspects of the project which could have been improved. This includes everything not implemented (see Section 10.6), where the threat model was not continuously updated, some website features were not implemented, performance tests were never done, and the penetration tests had a limited scope. Moreover, the Content Security Policy should have been more strict, where the application should have used technologies not requiring a weak CSP to work correctly. In addition, the downloaded files should have been sanitized, ensuring defense in depth.

Another possible improvement is related to the usage of methodologies. As we followed several methodologies simultaneously, it was difficult to completely follow one. The result was that we partially followed several methodologies, for example how we used several of the practices outlined in the Microsoft Security Development Lifecycle, but never followed it completely. This can be seen as both an advantage and disadvantage. Not being restricted to follow all methodologies allowed us to use the parts we deemed relevant, exploring how one can utilize multiple methodologies when developing software. On the other side, it prevented us from being able to correctly follow methodologies how they were intended, possibly limiting their effect on the development process and the knowledge gained in the thesis.

The coverage of several methodologies and the whole development process resulted in a large thesis covering many factors of developing secure applications. This can be seen as a weakness, as the thesis covers many topics, where some are only briefly covered. For example, the thesis could have focused more on the development of the application, instead of all the processes around it. On the other hand, it could have focused more on the methodologies around the development, rather than the code and configuration which were performed. Any of these approaches may have resulted in a more cohesive thesis, with the possibility of discussing topics in more detail.

11.3 Further Work

In the development of secure web application for digitalizing existing systems, there are several further areas of interest. In our thesis we have not covered the release and response phases of the Microsoft Security Development Lifecycle. This includes how you can use a secure configuration when deploying an application, how you can monitor and discover attacks, and how you can effectively respond to security incidents. This could be especially relevant to research for smaller organizations, where they may not have resources for a dedicated response team.

There is also possibility for more in depth research of how the methodologies followed can affect the development of software. Most of these were only briefly covered in the thesis, and includes Microsoft SDL, DevSecOps, risk assessments, and threat modeling.

According to the scientists at NINA there is also a need for more digitalisation of old systems, especially for handling data for research, see NINA meeting 24.04.2024 in Appendix P. Because of this they are planning on presenting our solution in scientific conferences, where they hope to inspire other scientists to leverage custom solution to modernise their data handling and visualization. As the application source code is published to GitHub with an MIT license, other scientist can leverage and build upon our code for their custom solutions.

11.4 Final Remarks

To conclude, we managed to develop a secure and user friendly custom web application satisfying all of NINA's requirements, which should improve their productivity and efficiency when working with their environmental data. Furthermore, we managed to utilize several relevant methodologies and practices during the development process, assisting in managing risk and the quality of the application. All the goals we set are met, however there is still possibilities for further work, both by continuing research in secure development, and by building more digital solutions to support scientific research.

Bibliography

- [1] NINA. 'Norsk institutt for naturforskning (nina).' (), [Online]. Available: <https://www.nina.no/0m-NINA/0m-oss>. (accessed: 15.04.2024).
- [2] M. S. Engineering. 'Overview.' (), [Online]. Available: <https://www.microsoft.com/en-us/securityengineering/sdl>. (accessed: 02.05.2024).
- [3] wikipedia.org. 'Devops.' (), [Online]. Available: <https://en.wikipedia.org/w/index.php?title=DevOps&oldid=1223472296>. (accessed: 20.05.2024).
- [4] wikipedia.org. 'Ci/cd.' (), [Online]. Available: <https://en.wikipedia.org/w/index.php?title=CI/CD&oldid=1180549991>. (accessed: 11.05.2024).
- [5] easypost. 'Dependency pinning guide.' (), [Online]. Available: <https://www.easypost.com/dependency-pinning-guide>. (accessed: 10.05.2024).
- [6] wikipedia.org. 'Web application.' (), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Web_application&oldid=1217467157. (accessed: 28.04.2024).
- [7] wikipedia.org. 'Hydration (web development).' (), [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Hydration_\(web_development\)&oldid=1188036964](https://en.wikipedia.org/w/index.php?title=Hydration_(web_development)&oldid=1188036964). (accessed: 11.05.2024).
- [8] developer.mozilla.org. 'Spa (single-page application).' (), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Glossary/SPA>. (accessed: 11.05.2024).
- [9] geeksforgeeks.org. 'Separation of concerns (soc).' (), [Online]. Available: <https://www.geeksforgeeks.org/separation-of-concerns-soc/>. (accessed: 11.05.2024).
- [10] K. Ubah. 'Javascript modules – explained with examples.' (), [Online]. Available: <https://www.freecodecamp.org/news/javascript-modules-explained-with-examples/>. (accessed: 11.05.2024).
- [11] svelte.dev. 'Svelte components.' (), [Online]. Available: <https://svelte.dev/docs/svelte-components>. (accessed: 05.05.2024).
- [12] developer.mozilla.org. 'Componentizing our svelte app.' (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_components. (accessed: 12.05.2024).
- [13] learn.svelte.dev. 'Declaring props.' (), [Online]. Available: <https://learn.svelte.dev/tutorial/declaring-props>. (accessed: 11.05.2024).

- [14] developer.mozilla.org. 'Working with svelte stores.' (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client_side_JavaScript_frameworks/Svelte_stores. (accessed: 11.05.2024).
- [15] learn.microsoft.com. 'Views.' (), [Online]. Available: <https://learn.microsoft.com/en-us/sql/relational-databases/views/views?view=sql-server-ver16>. (accessed: 11.05.2024).
- [16] developer.mozilla.org. 'Content security policy (csp).' (), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>. (accessed: 11.05.2024).
- [17] wikipedia.org. 'Reverse proxy.' (), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Reverse_proxy&oldid=1222683361. (accessed: 10.05.2024).
- [18] wikipedia.org. 'Proxy server.' (), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Proxy_server&oldid=1224552640. (accessed: 10.05.2024).
- [19] git. '7.11 git tools - submodules.' (), [Online]. Available: <https://git-scm.com/book/en/v2/Git-Tools-Submodules>. (accessed: 05.05.2024).
- [20] docker.docs. 'Dockerfile reference.' (), [Online]. Available: <https://docs.docker.com/reference/dockerfile/>. (accessed: 10.05.2024).
- [21] docker.docs. 'Docker overview.' (), [Online]. Available: <https://docs.docker.com/get-started/overview/>. (accessed: 10.05.2024).
- [22] docker.com. 'Use containers to build, share and run your applications.' (), [Online]. Available: <https://www.docker.com/resources/what-container/>. (accessed: 10.05.2024).
- [23] docker.docs. 'Docker compose overview.' (), [Online]. Available: <https://docs.docker.com/compose/>. (accessed: 10.05.2024).
- [24] wikipedia.org. 'Unit testing.' (), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Unit_testing&oldid=1222172687. (accessed: 14.05.2024).
- [25] E. F. Playbook. 'E2e testing.' (), [Online]. Available: <https://microsoft.github.io/code-with-engineering-playbook/automated-testing/e2e-testing/>. (accessed: 14.05.2024).
- [26] wikipedia.org. 'Static application security testing.' (), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Static_application_security_testing&oldid=1213839405. (accessed: 14.05.2024).
- [27] wikipedia.org. 'Lint (software).' (), [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Lint_\(software\)&oldid=1222271128](https://en.wikipedia.org/w/index.php?title=Lint_(software)&oldid=1222271128). (accessed: 14.05.2024).
- [28] cloudflare.com. 'What is penetration testing? | what is pen testing?' (), [Online]. Available: <https://www.cloudflare.com/learning/security/glossary/what-is-penetration-testing/>. (accessed: 09.05.2024).
- [29] HubSpot. 'User testing: The ultimate guide.' (), [Online]. Available: <https://blog.hubspot.com/service/user-testing>. (accessed: 05.05.2024).
- [30] svelte.dev. 'Introduction.' (), [Online]. Available: <https://svelte.dev/docs/introduction>. (accessed: 11.05.2024).

- [31] sanity. 'Svelte overview.' (), [Online]. Available: <https://www.sanity.io/glossary/svelte>. (accessed: 11.05.2024).
- [32] kit.svelte.dev. 'Introduction.' (), [Online]. Available: <https://kit.svelte.dev/docs/introduction>. (accessed: 11.05.2024).
- [33] kit.svelte.dev. 'Routing.' (), [Online]. Available: <https://kit.svelte.dev/docs/routing>. (accessed: 06.05.2024).
- [34] leafletjs. 'An open-source javascript library for mobile-friendly interactive maps.' (), [Online]. Available: <https://leafletjs.com/>. (accessed: 11.05.2024).
- [35] plotly. 'Plotly javascript open source graphing library.' (), [Online]. Available: <https://plotly.com/javascript/>. (accessed: 11.05.2024).
- [36] IBM. 'Exceljs.' (), [Online]. Available: <https://github.com/exceljs/exceljs>. (accessed: 11.05.2024).
- [37] V. Sattanatha. 'Custom excel export or import in client side using exceljs library.' (), [Online]. Available: <https://www.servicenow.com/community/developer-articles/custom-excel-export-or-import-in-client-side-using-exceljs/ta-p/2519012>. (accessed: 11.05.2024).
- [38] docker-curriculum.com. 'Learn to build and deploy your distributed applications easily to the cloud with docker.' (), [Online]. Available: <https://docker-curriculum.com/>. (accessed: 12.05.2024).
- [39] aws.amazon.com. 'What is docker?' (), [Online]. Available: <https://aws.amazon.com/docker/>. (accessed: 12.05.2024).
- [40] wikipedia.org. 'Nginx.' (), [Online]. Available: <https://en.wikipedia.org/w/index.php?title=Nginx&oldid=1221441786>. (accessed: 12.05.2024).
- [41] postgresql.org. 'About.' (), [Online]. Available: <https://www.postgresql.org/about/>. (accessed: 12.05.2024).
- [42] postgres.org. 'Postgres documentation.' (), [Online]. Available: <https://postgres.org/en/v12/>. (accessed: 12.05.2024).
- [43] w3schools.com. 'Django introduction.' (), [Online]. Available: https://www.w3schools.com/django/django_intro.php. (accessed: 12.05.2024).
- [44] B. Semah. 'What exactly is node.js? explained for beginners.' (), [Online]. Available: <https://www.freecodecamp.org/news/what-is-node-js/>. (accessed: 12.05.2024).
- [45] wikipedia.org. 'Package manager.' (), [Online]. Available: https://en.wikipedia.org/wiki/Package_manager. (accessed: 12.05.2024).
- [46] Vitest. 'Getting started.' (), [Online]. Available: <https://vitest.dev/guide/>. (accessed: 12.05.2024).
- [47] kit.svelte.dev. 'Static site generation.' (), [Online]. Available: <https://expressjs.com/>. (accessed: 05.05.2024).
- [48] G. Docs. 'About github copilot.' (), [Online]. Available: <https://docs.github.com/en/copilot/about-github-copilot>. (accessed: 12.05.2024).

- [49] G. Docs. 'Understanding github actions.' (), [Online]. Available: <https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>. (accessed: 09.05.2024).
- [50] G. Docs. 'Workflow syntax for github actions.' (), [Online]. Available: <https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions>. (accessed: 12.05.2024).
- [51] ESLint. 'Find and fix problems in your javascript code.' (), [Online]. Available: <https://eslint.org/>. (accessed: 12.05.2024).
- [52] G. Docs. 'About codeql.' (), [Online]. Available: <https://codeql.github.com/docs/codeql-overview/about-codeql/>. (accessed: 12.05.2024).
- [53] cwe.mitre.org. 'About cwe.' (), [Online]. Available: <https://cwe.mitre.org/about/index.html>. (accessed: 12.05.2024).
- [54] K. Schwaber and J. Sutherland, 'The definitive guide to scrum: The rules of the game,' *The Scrum Guide*, pp. 1–14, 2022. DOI: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>.
- [55] atlassian. 'Welcome to jira.' (), [Online]. Available: <https://www.atlassian.com/software/jira/guides/getting-started/introduction#dig-into-specific-features>. (accessed: 20.04.2024).
- [56] 'Information security, cybersecurity and privacy protection — Guidance on managing information security risks,' *International Standard*, 2022. DOI: <https://cdn.standards.iteh.ai/samples/80585/7bca93ac16fd426a9bc717cad9284d9/ISO-IEC-27005-2022.pdf>.
- [57] OWASP. 'Owasp top ten.' (), [Online]. Available: <https://owasp.org/www-project-top-ten/>. (accessed: 03.05.2024).
- [58] node. 'An introduction to the npm package manager.' (), [Online]. Available: <https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager#an-introduction-to-the-npm-package-manager>. (accessed: 05.05.2024).
- [59] C. J. Wendt and G. B. Anderson. 'Ten simple rules for finding and selecting r packages.' (), [Online]. Available: <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1009884#sec008%20Ten%20simple%20rules%20for%20finding%20and%20selecting%20R%20packages>. (accessed: 06.05.2024).
- [60] Emmanuel Ferdman and David LJ and Brian DeHamer and Luke Karrys and Gar. 'Npm-audit.' (), [Online]. Available: <https://docs.npmjs.com/cli/v10/commands/npm-audit>. (accessed: 06.05.2024).
- [61] 'Why the move away from npm registry? 2667.' (), [Online]. Available: <https://github.com/SheetJS/sheetjs/issues/2667>. (accessed: 06.05.2024).
- [62] kit.svelte.dev. 'Integrations.' (), [Online]. Available: <https://kit.svelte.dev/docs/integrations#vite-plugins>. (accessed: 07.05.2024).
- [63] Vite. 'Migration from v4.' (), [Online]. Available: <https://vitejs.dev/guide/migration.html#rollup-4>. (accessed: 07.05.2024).

- [64] Rollup. 'Tree-shaking.' (), [Online]. Available: <https://rollupjs.org/introduction/#tree-shaking>. (accessed: 07.05.2024).
- [65] G. security. 'About dependabot alerts.' (), [Online]. Available: <https://docs.github.com/en/code-security/dependabot/dependabot-alerts/about-dependabot-alerts>. (accessed: 07.05.2024).
- [66] G. security. 'About the github advisory database.' (), [Online]. Available: <https://docs.github.com/en/code-security/security-advisories/working-with-global-security-advisories-from-the-github-advisory-database/about-the-github-advisory-database#malware-advisories>. (accessed: 07.05.2024).
- [67] 'Version control.' (), [Online]. Available: <https://survey.stackoverflow.co/2022/#section-version-control-version-control-systems>. (accessed: 08.05.2024).
- [68] G. Docs. 'Ignoring files.' (), [Online]. Available: <https://docs.github.com/en/get-started/getting-started-with-git/ignoring-files>. (accessed: 09.05.2024).
- [69] G. Docs. 'About readmes.' (), [Online]. Available: <https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-readmes>. (accessed: 09.05.2024).
- [70] 'Awesome sveltekit.' (), [Online]. Available: <https://github.com/janosh/awesome-sveltekit/tree/main>. (accessed: 09.05.2024).
- [71] 'Svelte-commerce.' (), [Online]. Available: <https://github.com/itswadesh/svelte-commerce>. (accessed: 09.05.2024).
- [72] G. Docs. 'About workflows.' (), [Online]. Available: <https://docs.github.com/en/actions/using-workflows/about-workflows>. (accessed: 09.05.2024).
- [73] atlassian. 'Git ignore.' (), [Online]. Available: <https://www.atlassian.com/git/tutorials/saving-changes/gitignore#git-ignore-patterns>. (accessed: 09.05.2024).
- [74] conventionalcommits.org. 'Conventional commits 1.0.0.' (), [Online]. Available: <https://www.conventionalcommits.org/en/v1.0.0/>. (accessed: 09.05.2024).
- [75] A. Hovhannisyanyan. 'Make atomic git commits.' (), [Online]. Available: <https://www.aleksandrhovhannisyanyan.com/blog/atomic-git-commits/#3-atomic-commits-make-it-easier-to-complete-larger-tasks>. (accessed: 09.05.2024).
- [76] git. '3.1 git branching - branches in a nutshell.' (), [Online]. Available: <https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell>. (accessed: 09.05.2024).
- [77] V. Driessen. 'A successful git branching model.' (), [Online]. Available: <https://nvie.com/posts/a-successful-git-branching-model/>. (accessed: 09.05.2024).
- [78] G. Docs. 'Github flow.' (), [Online]. Available: <https://docs.github.com/en/get-started/using-github/github-flow>. (accessed: 09.05.2024).
- [79] G. Docs. 'About protected branches.' (), [Online]. Available: <https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/about-protected-branches>. (accessed: 09.05.2024).

- [80] G. Docs. 'About pull requests.' (), [Online]. Available: <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests>. (accessed: 09.05.2024).
- [81] F. Saraf. 'Git naming convention > branch naming.' (), [Online]. Available: <https://namingconvention.org/git/branch-naming.html>. (accessed: 09.05.2024).
- [82] D. Chien and P. Clemons. 'Coding and comment style.' (), [Online]. Available: <https://mitcommlab.mit.edu/broad/commkit/coding-and-comment-style/>. (accessed: 09.05.2024).
- [83] V. S. Code. 'Jsdoc support.' (), [Online]. Available: https://code.visualstudio.com/docs/languages/javascript#_jsdoc-support. (accessed: 09.05.2024).
- [84] 'Jsdoc.' (), [Online]. Available: <https://github.com/jsdoc/jsdoc>. (accessed: 09.05.2024).
- [85] M. Paul, *Official (ISC)² Guide to the CSSLP CBK*, 2nd ed. CRC Press, 2013, ISBN: 978-1-4665-7127-3.
- [86] M. D. of Information Technology. 'Sdlc phase 04 requirements analysis phase single cots.' (2004), [Online]. Available: <https://doit.maryland.gov/SDLC/Documents/SDLC%20Phase%2004%20Requirements%20Analysis%20Phase%20Single%20COTS.pdf>. (accessed: 14.05.2024).
- [87] H. F. Tipton and M. Krause, *Information Security Management Handbook*, 5th. Boca Raton, FL: Auerbach, 2005.
- [88] Lucidchart. 'Benefits of creating a visual sitemap.' (), [Online]. Available: <https://www.lucidchart.com/blog/7-unexpected-ways-to-use-sitemaps>. (accessed: 28.04.2024).
- [89] Lucidchart. 'Wireframes vs mockups: Determining the right level of fidelity for your project.' (), [Online]. Available: <https://www.lucidchart.com/blog/wireframes-vs-mockups>. (accessed: 29.04.2024).
- [90] kit.svelte.dev. 'Project structure.' (), [Online]. Available: <https://kit.svelte.dev/docs/project-structure>. (accessed: 05.05.2024).
- [91] geeksforgeeks.org. 'Coupling and cohesion – software engineering.' (), [Online]. Available: <https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/>. (accessed: 09.05.2024).
- [92] P. Kirvan. 'Loose coupling.' (), [Online]. Available: <https://www.techtarget.com/searchnetworking/definition/loose-coupling#:~:text=Loose%20coupling%20is%20an%20approach,to%20the%20least%20extent%20practicable>. (accessed: 09.05.2024).
- [93] postgres.org. 'Tables and views.' (), [Online]. Available: https://postgres.org/en/v12/references/api/tables_views.html. (accessed: 05.05.2024).
- [94] developer.mozilla.org. 'Origin.' (), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin>. (accessed: 05.05.2024).
- [95] learn.svelte.dev. 'Writable stores.' (), [Online]. Available: <https://learn.svelte.dev/tutorial/writable-stores>. (accessed: 06.05.2024).

- [96] svelte.dev. 'Prefix stores with *toaccesstheirvalues*.' (), [Online]. Available: [https://svelte.dev/docs/svelte-components#script-4-prefix-stores-with-\\$-to-access-their-values](https://svelte.dev/docs/svelte-components#script-4-prefix-stores-with-$-to-access-their-values). (accessed: 06.05.2024).
- [97] kit.svelte.dev. 'Page options – ssr.' (), [Online]. Available: <https://kit.svelte.dev/docs/page-options#ssr>. (accessed: 06.05.2024).
- [98] kit.svelte.dev. 'Page options – prerender.' (), [Online]. Available: <https://kit.svelte.dev/docs/page-options#prerender>. (accessed: 06.05.2024).
- [99] kit.svelte.dev. 'Link options.' (), [Online]. Available: <https://kit.svelte.dev/docs/link-options>. (accessed: 06.05.2024).
- [100] developer.mozilla.org. 'Using http cookies - security.' (), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#security>. (accessed: 08.05.2024).
- [101] developer.mozilla.org. 'Using http cookies.' (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#samesite_attribute. (accessed: 08.05.2024).
- [102] developer.mozilla.org. '<input type='number'>.' (), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/number>. (accessed: 05.05.2024).
- [103] OWASP. 'Input validation cheat sheet - allowlist vs denylist.' (), [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html#allowlist-vs-denylist. (accessed: 05.05.2024).
- [104] P. Krawczyk. 'Password special characters.' (), [Online]. Available: <https://owasp.org/www-community/password-special-characters>. (accessed: 05.05.2024).
- [105] svelte.dev. 'Html tags.' (), [Online]. Available: <https://svelte.dev/tutorial/html-tags>. (accessed: 05.05.2024).
- [106] datatracker.ietf.org. 'Http semantics.' (), [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc9110>. (accessed: 05.05.2024).
- [107] T. Dierks and E. Rescorla, 'The transport layer security (tls) protocol,' no. 1.2, 2008. DOI: <https://www.ietf.org/rfc/rfc5246.txt>.
- [108] C. S. P (Q. R. Guide. 'Content security policy reference.' (), [Online]. Available: <https://content-security-policy.com/>. (accessed: 05.05.2024).
- [109] developer.mozilla.org. 'Data urls.' (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Data_URLs. (accessed: 05.05.2024).
- [110] D. Lukanov and A. F. Kiær. '[bug] forced to use 'unsafe-eval' and 'unsafe-inline' in csp 1794.' (), [Online]. Available: <https://github.com/plotly/dash/issues/1794>. (accessed: 05.05.2024).
- [111] M. Valim. 'Svelte prerendered site not working with csp script-src 'self' 9496.' (), [Online]. Available: <https://github.com/sveltejs/kit/issues/9496>. (accessed: 05.05.2024).

- [112] chirudeepnamini. 'Prerendered pages with static adapter not working unless 'unsafe-inline' is used in content security policy when deployed to iis 11009.' (), [Online]. Available: <https://github.com/sveltejs/kit/issues/11009>. (accessed: 05.05.2024).
- [113] 'Plotly uses inline css.' (), [Online]. Available: <https://github.com/plotly/plotly.js/issues/2355>. (accessed: 05.05.2024).
- [114] M. Cousins. 'Intro.' (), [Online]. Available: <https://testing-library.com/docs/svelte-testing-library/intro/>. (accessed: 05.05.2024).
- [115] M. Cousins. 'Api.' (), [Online]. Available: <https://testing-library.com/docs/svelte-testing-library/api/#render>. (accessed: 05.05.2024).
- [116] Playwright. 'Playwright enables reliable end-to-end testing for modern web apps.' (), [Online]. Available: <https://playwright.dev/>. (accessed: 05.05.2024).
- [117] G. Docs. 'About code scanning with codeql.' (), [Online]. Available: <https://docs.github.com/en/code-security/code-scanning/introduction-to-code-scanning/about-code-scanning-with-codeql>. (accessed: 05.05.2024).
- [118] StandardJS. 'Javascript standard style - why should i use javascript standard style?' (), [Online]. Available: <https://standardjs.com/#why-should-i-use-javascript-standard-style>. (accessed: 05.05.2024).
- [119] Synopsys. 'Penetration testing.' (), [Online]. Available: <https://www.synopsys.com/glossary/what-is-penetration-testing.html>. (accessed: 05.05.2024).
- [120] OWASP. 'Cross site scripting (xss).' (), [Online]. Available: [https://owasp.org/www-community/attacks/xss/#:~:text=Cross%2DSite%20Scripting%20\(XSS\),to%20a%20different%20end%20user..](https://owasp.org/www-community/attacks/xss/#:~:text=Cross%2DSite%20Scripting%20(XSS),to%20a%20different%20end%20user..) (accessed: 05.05.2024).
- [121] NTNU. 'Openstack at ntnu.' (), [Online]. Available: <https://www.ntnu.no/wiki/display/skyhigh>. (accessed: 10.05.2024).
- [122] docker.docs. 'Multi-stage builds.' (), [Online]. Available: <https://docs.docker.com/build/building/multi-stage/>. (accessed: 05.05.2024).
- [123] node. 'Node.js releases.' (), [Online]. Available: <https://nodejs.org/en/about/previous-releases>. (accessed: 05.05.2024).
- [124] node. 'Npm-prune.' (), [Online]. Available: <https://docs.npmjs.com/cli/v8/commands/npm-prune>. (accessed: 05.05.2024).
- [125] 'Alpine-nginx.' (), [Online]. Available: <https://github.com/yobasystems/alpine-nginx>. (accessed: 05.05.2024).
- [126] 'What is nginx?' (), [Online]. Available: https://hub.docker.com/_/nginx. (accessed: 05.05.2024).
- [127] docker.docs. 'Dockerfile reference - shell and exec form.' (), [Online]. Available: <https://docs.docker.com/reference/dockerfile/#shell-and-exec-form>. (accessed: 05.05.2024).
- [128] N. Docs. 'Serving static content.' (), [Online]. Available: <https://docs.nginx.com/nginx/admin-guide/web-server/serving-static-content/>. (accessed: 05.05.2024).

- [129] N. Docs. 'Serving static content - enabling `tcp_nopush`.' (), [Online]. Available: https://docs.nginx.com/nginx/admin-guide/web-server/serving-static-content/#enabling-tcp_nopush. (accessed: 05.05.2024).
- [130] N. Docs. 'Serving static content - enabling `tcp_nodelay`.' (), [Online]. Available: https://docs.nginx.com/nginx/admin-guide/web-server/serving-static-content/#enabling-tcp_nodelay. (accessed: 05.05.2024).
- [131] N. Docs. 'Compression and decompression.' (), [Online]. Available: <https://docs.nginx.com/nginx/admin-guide/web-server/compression/>. (accessed: 05.05.2024).
- [132] developer.mozilla.org. 'X-frame-options.' (), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>. (accessed: 05.05.2024).
- [133] developer.mozilla.org. 'X-content-type-options.' (), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options>. (accessed: 05.05.2024).
- [134] developer.mozilla.org. 'Referrer-policy.' (), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy>. (accessed: 05.05.2024).
- [135] kit.svelte.dev. 'Single-page apps - usage.' (), [Online]. Available: <https://kit.svelte.dev/docs/single-page-apps#usage>. (accessed: 05.05.2024).
- [136] N. V. DATABASE. 'Cve-2021-44228 detail.' (), [Online]. Available: <https://nvd.nist.gov/vuln/detail/CVE-2021-44228>. (accessed: 05.05.2024).
- [137] A. C. D. Agency. 'Mitigating log4shell and other log4j-related vulnerabilities.' (), [Online]. Available: <https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-356a>. (accessed: 05.05.2024).
- [138] React. 'React the library for web and native user interfaces.' (), [Online]. Available: <https://react.dev/>. (accessed: 08.05.2024).
- [139] svelte.dev. 'Svelte cybernetically enhanced web apps.' (), [Online]. Available: <https://svelte.dev/>. (accessed: 08.05.2024).
- [140] React. 'React community.' (), [Online]. Available: <https://react.dev/community>. (accessed: 08.05.2024).
- [141] '2021 developer survey - web frameworks.' (), [Online]. Available: <https://insights.stackoverflow.com/survey/2021/#section-most-loved-dreaded-and-wanted-web-frameworks>. (accessed: 08.05.2024).
- [142] svelte.dev. 'Virtual dom is pure overhead.' (), [Online]. Available: <https://svelte.dev/blog/virtual-dom-is-pure-overhead>. (accessed: 08.05.2024).
- [143] postgres.org. 'Resource embedding.' (), [Online]. Available: https://postgres.org/en/v12/references/api/resource_embedding.html. (accessed: 08.05.2024).
- [144] postgresql.org. '41.3. materialized views.' (), [Online]. Available: <https://www.postgresql.org/docs/current/rules-materializedviews.html>. (accessed: 08.05.2024).

- [145] developer.mozilla.org. 'Window: Localstorage property.' (), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>. (accessed: 14.05.2024).
- [146] L. Wroblewski, *Mobile First. A BOOK APART*, 2011, ISBN: 978-1937557027.
- [147] kryogenix.org. 'Everyone has javascript, right?' (), [Online]. Available: <https://www.kryogenix.org/code/browser/everyonehasjs.html>. (accessed: 08.05.2024).
- [148] kit.svelte.dev. 'Configuration - csp.' (), [Online]. Available: <https://kit.svelte.dev/docs/configuration#csp>. (accessed: 05.05.2024).
- [149] kit.svelte.dev. 'Node servers.' (), [Online]. Available: <https://kit.svelte.dev/docs/adapter-node>. (accessed: 05.05.2024).
- [150] kit.svelte.dev. 'Static site generation.' (), [Online]. Available: <https://kit.svelte.dev/docs/adapter-static>. (accessed: 05.05.2024).
- [151] H. Sarojadevi, 'Performance testing: Methodologies and tools,' vol. 1, no. 5, 2011. DOI: <https://core.ac.uk/download/pdf/234676929.pdf>.
- [152] FN. 'Fns bærekraftsmål.' (), [Online]. Available: <https://fn.no/om-fn/fns-baerekraftsmaal>. (accessed: 10.05.2024).
- [153] lovdata.no. 'Lov om opphavsrett til åndsverk mv. (åndsverkloven).' (), [Online]. Available: <https://lovdata.no/dokument/NL/lov/2018-06-15-40?q=%C3%85ndsverkloven>. (accessed: 14.05.2024).
- [154] Regjeringen.no. 'Stortinget vedtok nye krav om universell utforming av nettsteder og mobilapplikasjoner.' (), [Online]. Available: <https://www.regjeringen.no/no/dokumentarkiv/regjeringen-solberg/aktuelt-regjeringen-solberg/kud/nyheter/2021/stortinget-vedtok-nye-krav-om-universell-utforming-av-nettsteder-og-mobilapplikasjoner/id2854752/>. (accessed: 14.05.2024).
- [155] w3.org. 'Web content accessibility guidelines (wcag) 2.1.' (), [Online]. Available: <https://www.w3.org/TR/WCAG21/>. (accessed: 14.05.2024).
- [156] wcag.com. 'Wcag 101: Understanding the web content accessibility guidelines.' (), [Online]. Available: <https://wcag.com/resource/what-is-wcag/>. (accessed: 14.05.2024).
- [157] lovdata.no. 'Lov om elektronisk kommunikasjon (ekomloven).' (), [Online]. Available: https://lovdata.no/dokument/NL/lov/2003-07-04-83/KAPITTEL_2#%C2%A72-7b. (accessed: 14.05.2024).
- [158] intersoft consulting. 'General data protection regulation gdpr.' (), [Online]. Available: <https://gdpr-info.eu/>. (accessed: 14.05.2024).
- [159] norid.no. 'Domain name policy for .no.' (), [Online]. Available: <https://www.norid.no/en/om-domenenavn/regelverk-for-no/>. (accessed: 14.05.2024).
- [160] H. Tutorials. 'Drawing entity relationship diagrams with uml notation using lucidchart.' (), [Online]. Available: <https://holowczak.com/drawing-entity-relationship-diagrams-with-uml-notation-using-lucidchart/>. (accessed: 14.05.2024).

Appendix A

Web Application Screenshots

This document shows the web application developed in the thesis project. It shows how the various pages can be used to view and manage environmental data.

The Map Page

The first page that is presented to users when opening the web application is the map page. The map page displays all rivers the scientists at NINA has collected data from. These data points can be filtered from the menu on the left, where the user can choose the date range the observations should be in, and which species it should contain. The map page can be seen in Figure A.1.

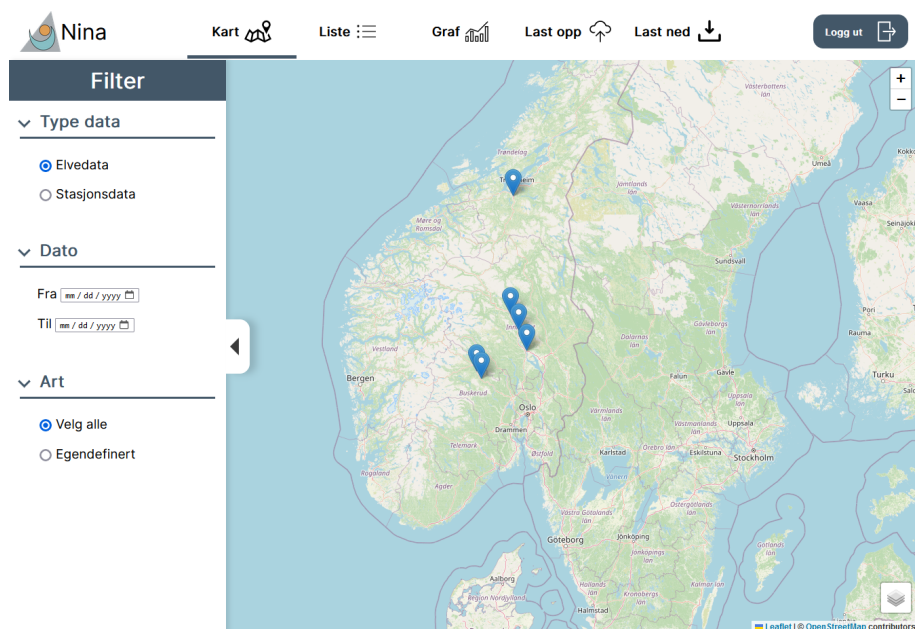


Figure A.1: Map Page

If the user clicks on a river data point, a summary of the observations are listed on the right hand side of the page. This can be seen in Figure A.2. From here the river can also be opened in the graph or download page.

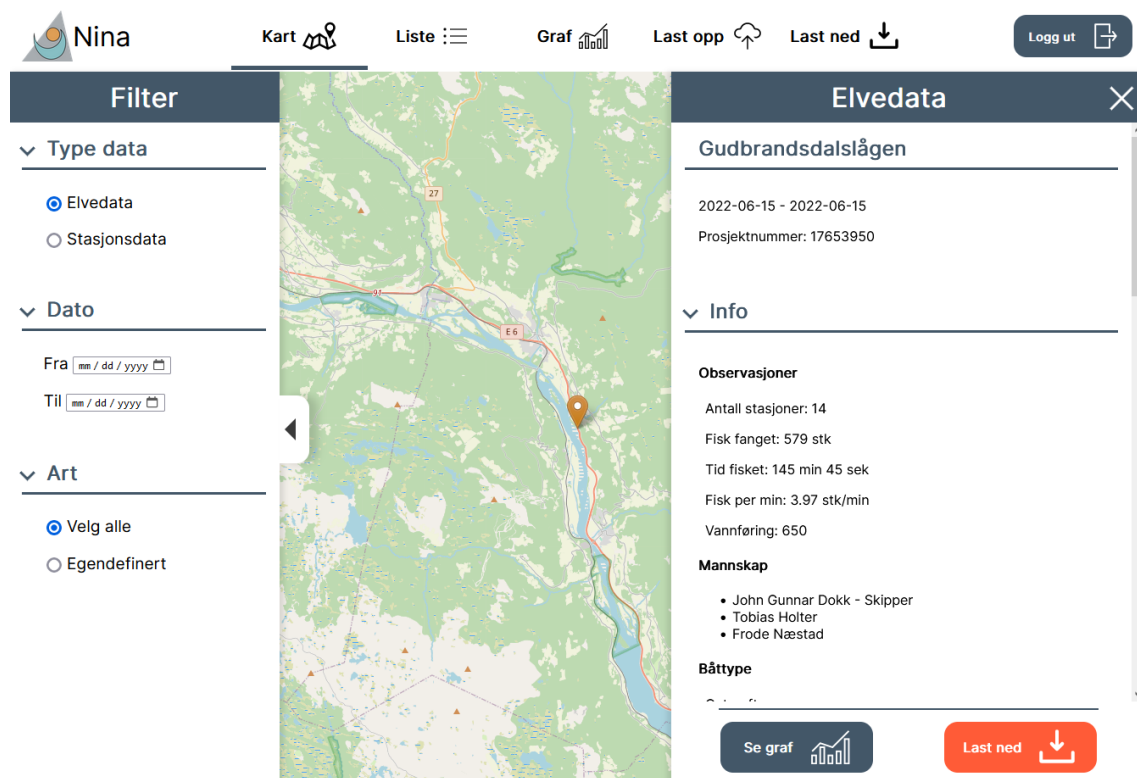
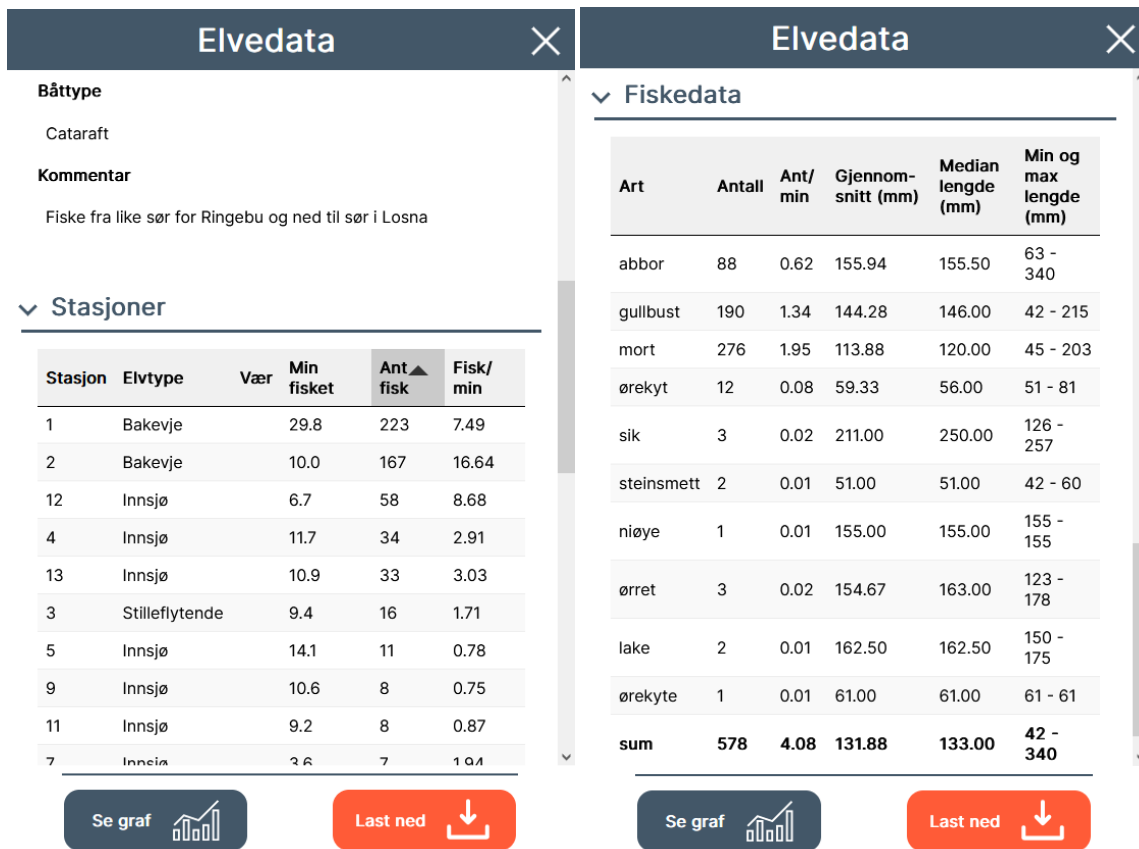


Figure A.2: Observation summary of Gudbrandsdalslågen river

When the user scrolls down the summary page, they can also see information about the stations which are under the specific river in a table, and their information, as shown in Figure A.3. These can be clicked to select the stations. Moreover, the user can also see fish data, from each observation in that specific river. This is displayed in Figure A.4.



When a station is selected, either by choosing it from a river, or by selecting the type of data and stations in the filter, stations are displayed in the map. Each station has a line between where the trip started and where it ended. When a station is selected it also changes color to orange, indicating the selection. This can be seen in Figure A.5. Here the stations data is also displayed, such as comments, weather, and power settings. It is also possible to open the station on the graph and download page from here.

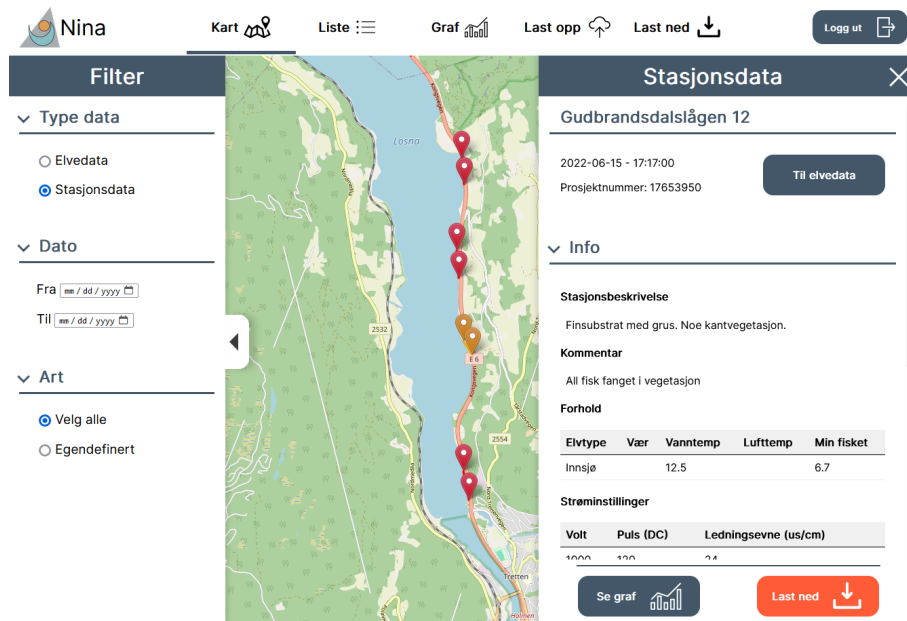


Figure A.5: Summary of Gudbrandsdalslågen 12 station

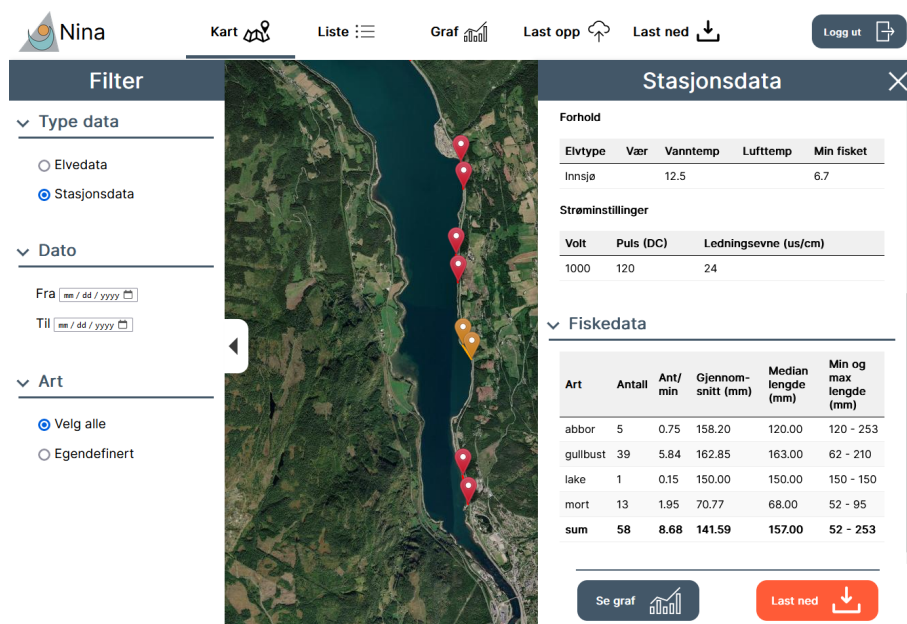


Figure A.6: Fish data under Gudbrandsdalslågen 12 station with satellite view selected

By scrolling down it is also possible to see the fish data for the specific station selected, shown in Figure A.6. Here the map is set to satellite view.

In Figure A.7 the map page has been zoomed in on station points, where the ones shown are in the date range 05.01.2020 - 05.17.2024, and has the species Ørret. Here the map type

topology has been selected.

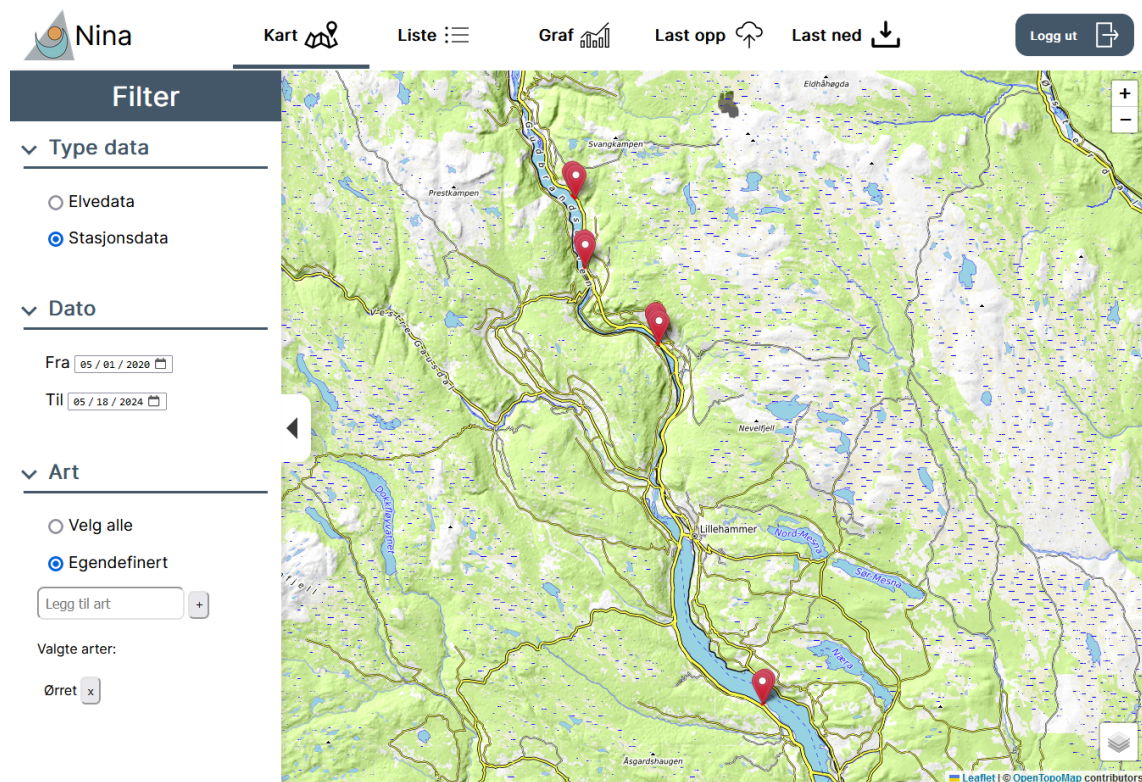


Figure A.7: Topology view filtered after stations

The List Page

The next page in this web application is the list page. On this page the same filtering menu is used as for the map page. The list page can be seen in Figure A.8.

The screenshot shows the 'Liste' view of the Nina application. On the left, there is a 'Filter' sidebar with three sections: 'Type data' (Elvedata selected, Stasjonsdata unselected), 'Dato' (date range from 'Fra' to 'Til'), and 'Art' (Velg alle selected, Egendefinert unselected). At the top, there is a search bar with the text 'Søk etter Elv navn' and a note 'Bruk filter for å filtrere resultat'. Below the search bar is a table with three columns: 'Navn', 'Dato', and 'Prosjektnummer'.

Navn	Dato	Prosjektnummer
Hallingdalselva	2014-09-16	None
Gaula	2021-09-30	Restaureringsprosjektet
Hallingdalselva	2021-09-21	None
Gudbrandsdalslågen	2023-09-20	1234567689
Gudbrandsdalslågen	2022-06-15	17653950
Mjøsa	2021-06-16	None

Figure A.8: List page

In the list page, the user can search after specific rivers and station by their name or date, in combination with the filter, as shown in Figure A.9

This screenshot shows the same 'Liste' view as Figure A.8, but with the 'Type data' filter set to 'Stasjonsdata'. The search bar contains the text 'lågen 1'. The table below shows a filtered list of stations for Gudbrandsdalslågen, with columns for 'Navn', 'Dato', and 'Kl.'.

Navn	Dato	Kl.
Gudbrandsdalslågen 1	2022-06-15	11:37:00
Gudbrandsdalslågen 10	2022-06-15	16:35:00
Gudbrandsdalslågen 11	2022-06-15	16:54:00
Gudbrandsdalslågen 12	2022-06-15	17:17:00
Gudbrandsdalslågen 13	2022-06-15	17:41:00

Figure A.9: List filter search

Once the desired station or river is identified, it can be clicked. This opens up a summary page with similar information as in the map page. This can be seen in Figure A.10 where a

station has been clicked for further information. From here it is possible to open river and stations in the map page, graph or download page.



Figure A.10: Station list summary

The Graph Page

The next page is the graph page, where data for rivers and stations is plotted in graphs. In this page, the user can select one or multiple rivers or stations that they want to display and research further. This is done by opening the menu depicted in Figure A.11.

This page displays the data using graphs and diagrams, like a bar chart which can be seen in the accompanying Figure A.12. Here the species Gjedde, Ørret, and Ørekyt is selected to be shown in bar charts for the two rivers selected, comparing the amount of these species each river contained.

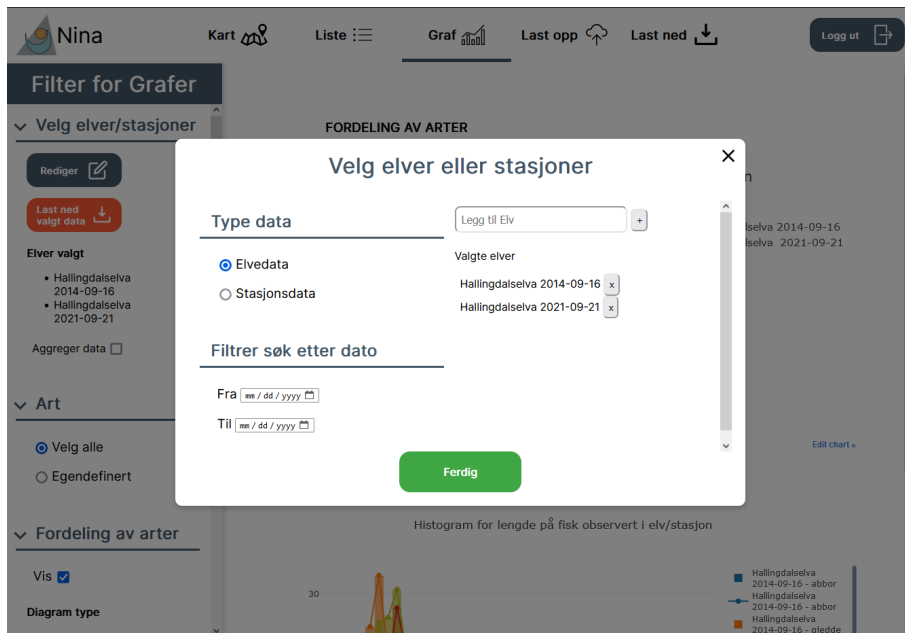


Figure A.11: Choosing rivers in graph

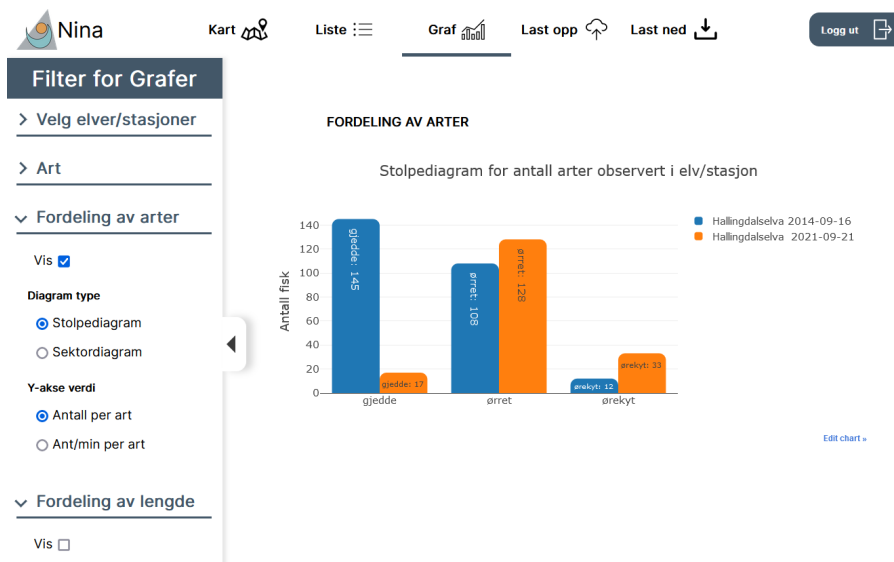


Figure A.12: Bar char selected in the graph page

Additionally, the user can choose to get the data displayed as a pie chart (see in Figure A.13), a histogram (see Figure A.14) or as a box plot (see Figure A.15).

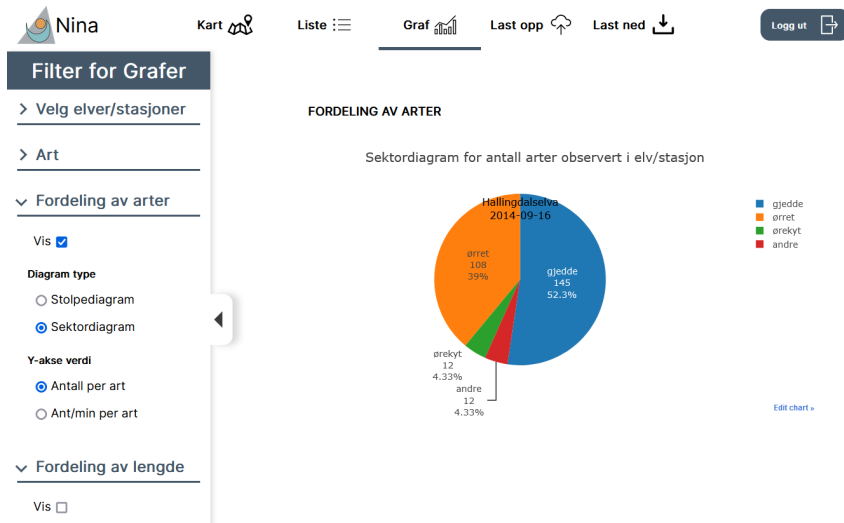


Figure A.13: Pie-chart graph page

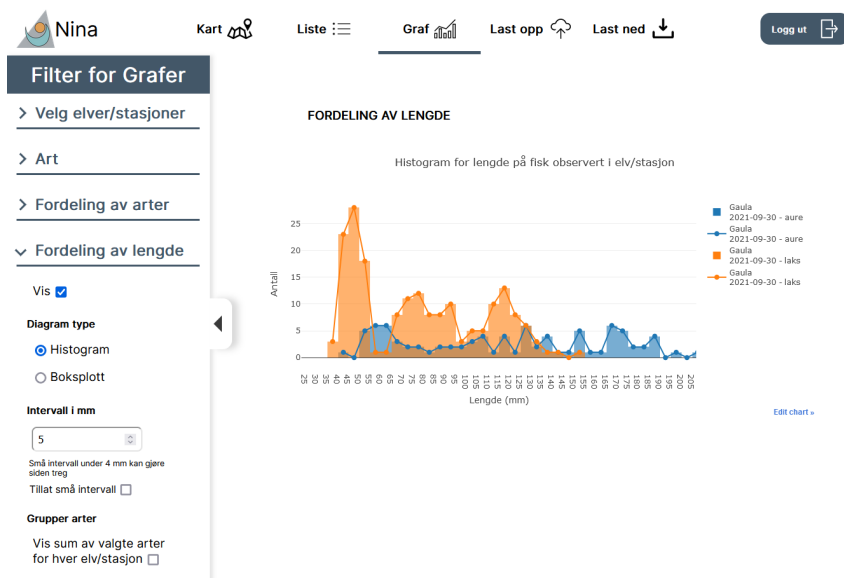


Figure A.14: Histogram selected in the graph page

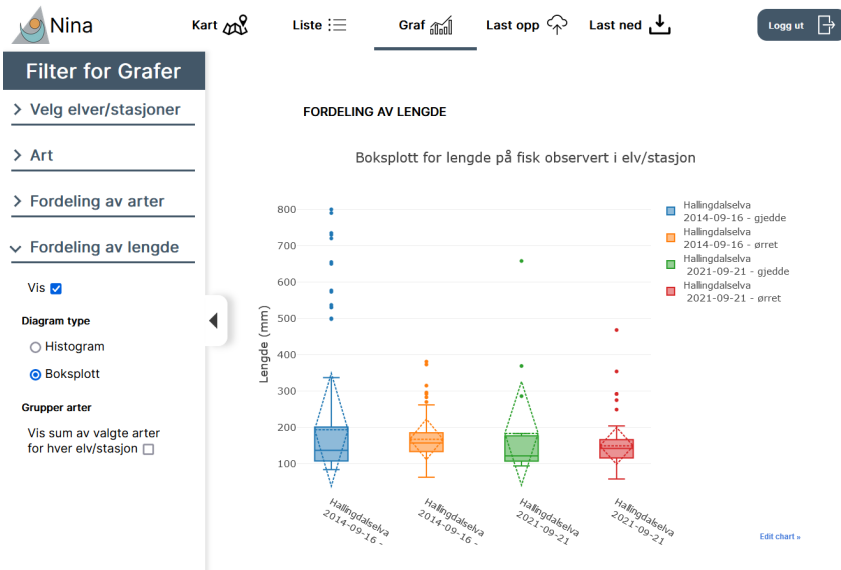


Figure A.15: Box-plot selected in the graph page

The Upload Page

The next page is used for uploading files to the PostgreSQL database, that then updates the river/stations data used in the web application. The page features drag and drop functionality, or browsing of the file explorer (compatible with all operating systems) for selecting files. The upload page is displayed in Figure A.16.

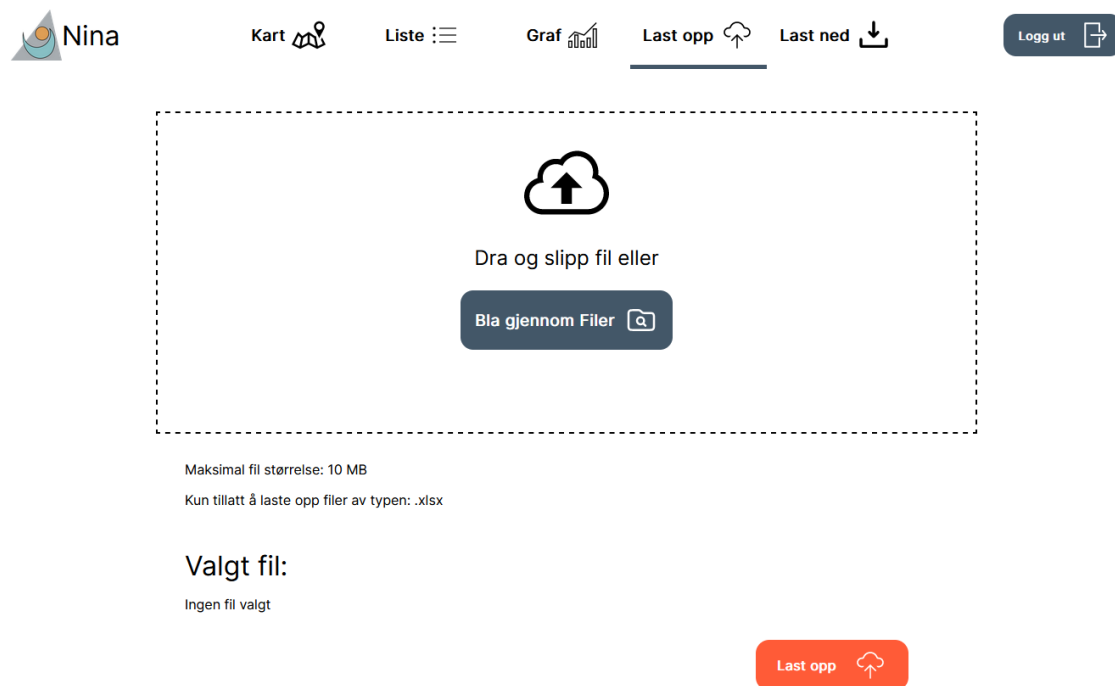


Figure A.16: Upload page

The Download Page

The web application also has a download page, which allows the user to download either river or station data in CSV or XLSX files. This can be seen in the accompanying Figure A.17. Here the user can choose which rivers or stations to download, which species to include, and the desired file format.

Nina Kart Liste Graf Last opp Last ned Logg ut

Last ned data

Velg elver/stasjoner

Rediger

Elver valgt

- Hallingdalselva 2014-09-16
- Hallingdalselva 2021-09-21

Art

Velg alle

Egendefinert

Legg til art

Valgte arter:

Ørret x

Gjedde x

Format

xlsx

csv

Last ned

Figure A.17: Download page

The Login/logout Page

Lastly, the web application also has a login page. This page allows user to enter their username and password, and if these are valid, be logged in. The page is shown in Figure A.18.

Nina Kart Liste Graf Last opp Last ned Logg inn

Logg inn

Brukernavn

Skriv inn brukernavn

Passord

Skriv inn passord

Logg inn

Figure A.18: Login/Logout page

Appendix B

Project Plan

DCSG2900
BACHELOROPPGAVE BACHELOR I DIGITAL INFRASTRUKTUR
OG CYBERSIKKERHET

Interactive database for Environmental data
Project Plan



Kavishnayan Nadarajah, Kevin Nikolai Mathisen, Carl Petter
Mørch-Reiersen, og Martin Solevåg Glærum

Spring 2024

Contents

1	Goals and restrictions	2
1.1	Background	2
1.2	Project Goals	2
1.3	Project Framework	3
2	Scope	4
2.1	Project description	4
2.2	Project Delimitation	5
3	Project organization	6
3.1	Responsibilities and roles	6
3.2	Group Rules and Routines	7
4	Planning, Follow-up and Reporting	8
4.1	Choice of project management framework	8
4.2	How we are going to use scrum	8
4.3	Plan for status meetings	9
4.4	Project structure	9
4.5	Milestones for the project	10
4.6	Sprints	10
5	Organization of quality assurance	11
5.1	Tools	12
5.2	Risk analysis	13
6	Implementation plan	16
	Bibliography	17

1 Goals and restrictions

1.1 Background

Norsk institutt for naturforskning (NINA) is an organization researching the interactions between nature and society. Their overarching goal is to contribute to sustainable social development by delivering research-based and up-to-date knowledge about natural diversity, climate and society [5]. NINA has multiple offices throughout Norway, among which this project was requested by the department in Lillehammer.

Throughout their years of work, NINA has collected a considerable amount of environmental data from different rivers in Innlandet. The data is being collected using electric fishing, where the researchers stun the fish, allowing them to examine the fish, and then release it back in the water. The data collected for each river includes how many fish have been found, and their species, size and gender. This data has so far been stored in different Microsoft excel sheets, on several different computers. This resulted in some complications when it came to finding and using specific data which had been collected. NINA wanted help to create a system which could be used to visualize all of their collected data in an easy and efficient way, which would allow them to focus more on research and less on technical complexities.

1.2 Project Goals

The goals for the project have been divided into effect goals, result goals and learning goals. Effect goals describe the desired effect and benefits expected for NINA which they will get from the product. Result goals define what the final product we create should accomplish. Learning goals are what we hope to learn during this project. This includes skills and experience which should be used for future projects not related to NTNU.

Effect Goals

The final product should have the following effect:

1. The website should streamline the workflow for NINA's scientists when trying to view, analyze, and visualize their collected data. It should be simple to locate a specific data point and gain an understanding of its underlying observations, which will save the scientists time and energy.
2. Make the process of testing hypotheses easier for the scientists, by easily downloading aggregated data of the specific observations they want to analyze, which should enable them to focus more on research and less on technical complexities, such as manually combining excel documents for analyzing data.

Result Goals

When we are done with the bachelor we want to have achieved the following.

1. Make a functional website and backend fulfilling the requirements set by NINA. The website should provide an intuitive interface that simplifies the process for accessing, analyzing, visualizing, and aggregating their collected data. This should make the process of testing hypotheses easier for the scientists, which should enable them to focus more on research and less on technical complexities.
2. Deliver the website packaged in a dockerfile which NINA can use to deploy the web application without further modification or upkeep. It should seamlessly integrate with their

infrastructure.

3. Deliver NINA with comprehensive documentation covering the development process for the website, as well as the website source code, to enable easy upkeep and patching by their internal IT-team.
4. Ensure that the website has countermeasures in place to protect against possible threats, as well as clearly documenting any risks to NINA associated with the website.

Learning Goals

We want to learn as much as possible from working in a team and building a website. And we have some specific goals in mind like:

1. Solving projects given by an employer.
2. Experiencing working in a small group over a longer period of time to accomplish a singular task, while using the SCRUM methodology.
3. Acquire skills and learn technologies needed for creating a functional front-end service which can interact with an API.
4. Learn how to handle and identify risk while developing software
5. Learn to use Git and other development tools in a bigger projects.
6. Learn to use CI/CD for code testing and verification while developing software.

1.3 Project Framework

Framework describes restrictions on the project set by NINA

- The server will run in a docker container.
- The project must utilize version pinning.
- The webserver has to consume postgREST API.
- The project must use libraries and programming languages which are open source, and these should be actively maintained.
- All code has to be stored in GitHub with a MIT license.
- The project must be finished before 21.05.2024.
- Any dependencies should be clearly documented.

2 Scope

2.1 Project description

We will design a web application which is capable of communicating with NINA's back-end server. The web application will be able to efficiently load the data from the back-end, and visualize this data in a way which satisfies NINA's needs. It will also be possible to upload and download data to and from the back-end database through the web application. As uploading data through a web application includes risks, there will be a focus on finding and mitigating any threats for the web application.

The web application should have the following functionality:

1. The user should be able to view and interact with a map of Norway and Sweden, which contains points marking where each observation has taken place
 - (a) The user should be able to filter the observations shown by selecting the dates, species, and the type (river/station level) of the observations they want shown.
 - (b) Observations on station level should be displayed as two points connected with a line, illustrating the path the observation took. Observations on river level should be displayed as points.
2. When clicking on a river data point the user should be able to view tables containing a summary of the data from its underlying station observations, as well as general information about the river data point.
 - (a) The user should be able to download the aggregated data of all the underlying station observations as a csv file.
3. When clicking on a station data point the user should be able to view tables containing a summary of the data from its observations, such as the amount of fish per species, as well as general information about the station data point, such as name and project number.
 - (a) The user should be able to download the data of the station observations as a csv file.
4. The user should be able to select one or multiple station data points, and open these in a new window for more in depth analysis. The data of the observations will be shown together in figures, allowing for easy comparison.
 - (a) The data will be visualized using figures, graphs, and tables.
 - (b) The user should be able to select what they want to be plotted, which can include the distribution of species (optionally up against the time spent fishing), the distribution of size for selected species (with adjustable intervals for size grouping, standard 5mm), and more.
 - (c) If multiple stations are selected, the user should be able to either aggregate the data to be shown together, or visualize and compare the differences between the stations.
5. The user should be able to upload data in a specified csv format when authenticated, which should provide feedback if the task was successful or not.

6. The user should have the capability to flag data for further review when authenticated, for example if they observe a station with invalid data. An administrator can then find the flagged data and modify or delete it if necessary.

The web application should have the following security functionality:

1. When downloading or uploading data the user should be authenticated.
2. Handle sessions according to best practices.
3. A user should not be able to upload data to the web application if it contains invalid format or malicious data.
4. The system should log who uploads the data to ensure non-repudiation.
5. Handle exceptions and errors according to best practices.
6. All dependencies should be open source from secure sources, ensuring a secure supply chain.

The web application should have the following performance functionality:

1. Under normal usage the web application should be quick and responsive.
2. Requests should be cached to minimize database queries and improve performance.

2.2 Project Delimitation

Our project is mainly about creating and designing the web application of the system NINA is trying to create. We will also create a dockerfile which can be used to launch the web application. While we will test the functionality of the dockerfile by deploying it to a server, it is not our responsibility to manage the deployment of the web application for NINA, as they will add our dockerfile to their already existing docker cluster. Furthermore we will not design, modify or deploy the Postgres database, as NINA has already created a sql schema and deployed it. We will only interact with the database using the PostgREST API they have configured.

While making the web application user friendly for mobile users isn't off the table, NINA made it clear that that this shouldn't be our focus as they will mainly be using the web application through their computers.

NINA has established organization-wide authentication through LDAP and web solutions for login and session management. Therefore the plan is to integrate their solution to our web application, rather than building authentication from the ground up.

3 Project organization

3.1 Responsibilities and roles

Even though there are roles for different aspects of the website, all parts of the development will include all group members. Division of responsibilities and roles:

Group leader / Scrum master - Kavi

The group leader is the main communication point between NINA / supervisors and the group. He establishes meetings with them and is the one leading the discussions between the group and NINA / supervisors. The group leader is responsible for keeping track of all the different exercises which need to be completed, and making sure to fairly designate the exercises between members of the group. He also has the main responsibility of creating tasks when needed, to make sure all members are continuously working with something on the project.

Being scrum-master is also part of the group leader's responsibilities and is therefore responsible for helping the product owner in developing a scrum plan and the product backlog so that it is understandable for the scrum team [4].

Secretary - Kavi

Secretary is the one responsible for taking notes during all meetings with NINA and with the supervisors. These notes are later used by the group to recap what was said and learned during meetings. All group members can help add information to the notes to increase the level of detail of the notes.

Quality assurance - Everybody

After consideration, we decided it would be best if everybody had this role. All work that is done should be of good quality, so it's everybody's responsibility to ensure this is the case. Each group member is thus responsible for reading and understanding work done by other group members, and checking that the work is of adequate quality. This role also incorporates the responsibilities of continuously testing code, and reviewing code written by others.

Responsibility for DevOps (CI/CD, GIT, deployment) - Carl Petter

The DevOps role has to ensure continuous integration and continuous delivery by using GitHub Action, implement automated code testing, and manage Git repository branches and other aspects. It also includes establishing and maintaining a Linux server in SkyHigh for testing software, which automatically updates and runs the newest version of code from the repository.

Head of Design - Martin

Responsible for leading the design for the website which will satisfy NINA. Should also ensure that the website follows standards for web design and other design best practices. Should ensure that the website accomplishes all functionality requested.

Head of Back-end - Kevin

Is responsible for leading the creation and logic of the backend, and make sure it follows current best practices, such as proper error handling and status messages. Should ensure that the website is able to interact with the postgREST API, aggregate data, and handle download and upload of data.

3.2 Group Rules and Routines

You can find the Group rules and routines in the attached document [Grouprules.pdf].

4 Planning, Follow-up and Reporting

4.1 Choice of project management framework

After evaluating the different possible methodologies, we have decided to adopt Scrum as the methodology for this project. There are multiple reasons for this choice, which includes scrums flexibility which comes from it's iterative nature, as well as it's fixed routines and structure. We therefore think that an agile methodology like Scrum would be the best to satisfy our needs.

Scrum would allow the customer to see the progress and come with comments on changes and improvements, and would therefore give us flexibility when it comes to adjustments. This would enable us to scale up and add features if the allotted time allows it, as well as enabling us to modify our product to ensure we finish a product the customer will be satisfied with [8].

Scrum would also let the development team work more transparently, by implementing Daily Scrum. These daily meetings would be held to inspect the progress towards the Scrum goal. Daily meetings would ensure communication between the participants in the development team and resolve any uncertainties on what the developers tasks are. Any necessary adjustments to the Scrum backlog would be made under these daily meetings, which would last 15 minutes and be held digitally or in person. [7].

This model would let us split the work between the developers, where they can work independently and create their own solutions to problems we may face along the way. The team can then discuss the solution under the daily scrum meetings to verify the solution. This way of working would enable the team to work on multiple features at the same time.

4.2 How we are going to use scrum

Each sprint will be held for two weeks, where every sprint is going to end with a meeting with NINA and the supervisors. We think that setting the sprints to 2 weeks is a reasonable time span. It isn't too long, as it enables us to regularly have meetings with the product owner allowing them to give feedback, as well as fitting with the limited time of our bachelor. It is also not too short, so that the developers actually get the time to understand and execute the task they have been given in a timely manner until the next sprint, as well as reducing overhead by only having status meetings every other week.

Under the sprint review we will look at what has been accomplished, what is unfinished, what went well, and what needs improvement. The next sprint would also be planned with new work added to the Sprint Backlog from the Product backlog. The product backlog contains information of the work that needs to be done to finish the program. The tasks in the product backlog are added in a descending order from the most important to the least important work. It would then be separated where each separation would contain the task for one sprint, where the tasks for the current sprint will be added to the sprint backlog. [6].

The sprint backlog would only contain the work that needs to be executed in the sprint that the team is currently working on. It also shows the team the current status of what has been started on, what is finished and what needs to be done before the sprint ends. We will make use of a Burndown chart which could give us a good indication for the progress we have made and keep us on the track by showing us how much we have left. We can then use this information

to predict the likelihood for the team to complete tasks before the deadline. [1].

The scrum team would consist of one product owner, one scrum master and four developers. The product owner is the one accountable for maximizing the value of the product, by developing the product goal and the product backlog while clearly communicating these to the rest of the team. The scrum master is the one responsible for clearly communicating the scrum theory over to the scrum team, and would also actively help the product owner in developing the product goal and the product backlog and make them more understandable for the scrum team.

The developers are the ones accountable for developing the product and also responsible for doing the planned work under each sprint. They would be accountable for planning the sprint backlog and adapting the plan to meet the sprint goal. [4]. To get a visual representation of the work we put in, we will use the program “Jira” to get a “Burndown Chart”, which would give us an indication on the progress we have made on the project.

4.3 Plan for status meetings

- Internal Status Meeting will be done after each sprint, where the group will gain an overview of the project and its progress.
- We will also have weekly meetings with our supervisors each Thursday at 11.30, or less frequently if the group does not deem the meetings necessary.
- Meeting with NINA will be after each sprint at Wednesdays 14:00, where we will show our progress and let NINA give us feedback.

4.4 Project structure

Pre-Project

- Write the project plan
- Setup communication and meetings with supervisors and NINA.
- Setup tools, git, server

Design

- One sprint where we develop the design of the application using wireframes and user stories. We will also do a user test to evaluate the design.

Main Development

- Development of the website and its features, which will consist of four sprints.
- Also includes writing automated unit tests, as well as integration tests while the website is being developed.
- The security will also be evaluated.
- We will do a user test towards the end with the almost completed product.

Bug fixing and last adjustments

- Fixing bugs and testing the security. Will be done in one sprint.

Finishing the report

- We will continuously write on the report throughout the whole bachelor, but we will also have two sprints focusing on writing and finishing the report when we are done with the website.

Presentation

- Will be preparation for the presentation after the report is delivered.

4.5 Milestones for the project

These are important milestones in our project, which we will use to plan our work and monitor our progress. By setting our own deadlines we ensure that we have sufficient time for all our tasks.

- 31.01.24 - Project plan deadline, decided on technologies and tools to use, git, server and programming environment ready.
- 01.02.24 - Start of project.
- 15.02.24 - Starting to code, should have completed user test of design.
- 11.04.24 - Deadline for main features, second user test completed.
- 25.04.24 - Deadline for fixing bugs and security-related issues.
- 16.05.24 - Deadline for report, only quality assurance left.
- 21.05.24 - Deadline for submitting Project Report.
- 05.06.24 - Presentation of the Project.

4.6 Sprints

These are the sprints we will complete during the semester, where each are two weeks long, except for sprint 8 which is one week.

- Sprint 1 - Design: 01.02.24-14.02.24
- Sprint 2 - Coding: 15.02.24-28.02.24
- Sprint 3 - Coding: 29.02.24-13.03.24
- Sprint 4 - Coding: 14.03.24-27.03.24
- Sprint 5 - Coding: 28.03.24-10.04.24
- Sprint 6 - Bug fixing: 11.04.24-24.04.24
- Sprint 7 - Report writing: 25.04.24-08.05.24
- Sprint 8 - Finish report: 09.05.24-20.05.24

5 Organization of quality assurance

Documentation

This is how we will ensure quality assurance and availability of documentation.

- Documentation will be stored primarily on Google Docs, with backups locally on our machines. This documentation includes Meeting minutes, worklogs, and other documentation.
- Final report and deliveries will be written in Latex using Overleaf. This will be stored on Overleaf, with backups on both Google Docs and locally on our machines.
- We will document resources we use for learning, documentation of libraries used, and other choices related to programming underway in the project.

Routines when using Scrum

We will use scrum for quality assurance, by ensuring proper communication, documentation, and resolving misunderstandings. To achieve this we will use scrum as follows:

- All tasks are defined and documented, then added to the backlog in scrum.
- Daily stand-up meetings, where misunderstandings can be resolved, and problems are fixed.
- Use sprint reviews to look for problems in our methodology, and how to fix them. Furthermore the product owner can look at our progress and correct any mistakes, acting as quality control.
- Before marking a backlog item as done, it is tested and documented.

Routines when using git

When using git we will follow best practices, which includes proper use of commits and branches. Our methodology follows best practices, and is inspired by the quality assurance from [3]. Our use of branches are inspired by Vincent Driessen's Git branching model [2].

- There will be 3 layers of branching.
 - The primary branch will be the master, which contains stable versions of our application.
 - The second layer will be the developer branch, which will contain new features being integrated. At the end of each sprint the developer branch is merged with the master branch.
 - The third layer is issue branches, which are features or bugs under development on their own branches. On completion of the issue, it is merged with the developer branch.
- Each commit should be atomic, with a simple logical unit of change in code. Commits should use conventional commit messages for standard format and meaning.
- Commit messages should include an issue ID.
- Each commit should contain inline comments following the programming language's comment convention describing the function of the code.

Programming and testing

For testing and coding of our application we will follow the following guidelines to ensure quality code and software.

- All of our programming will follow the guidelines and conventions for the language, such as naming convention, formatting, comments, error handling, and other best practices.
- We will conduct two user tests, one on the design, and one on the application at the end of the development.
- We will use automatic testing tools locally and on git, using GitHub actions, which will consist of unit tests, integration tests, and HTTP tests.
- There will also be done end-to-end tests to simulate and test real user scenarios.
- Performance and security tests will be done towards the end of the development.
- After each sprint we will have meetings with the product owner, where they can test and review the product. This feedback will be used to steer the development in the right direction

5.1 Tools

The following tools will be used to support our work in the bachelor program, as seen in table 5.1.

Name	Type	Used for
Github	Hosting and version control for source code	Version control
Github copilot	Coding assistant	Development/Coding
Github actions	Automation of parts of ci/cd, such as automated testing and deployment	Testing/deployment
Visual studio code	IDE	Development/coding
Draw.io	Diagram tool	Creation of diagrams, such as UML and Flowcharts
Google drive	Hosting of files	Documentation
Overleaf	Latex file editor	Report writing/Documentation
SkyHigh	IaaS	Hosting of test application
Docker	Containerization platform	Deployment
TeamGantt	Gantt editor	Creation of gantt diagram
Jira	Scrum project management and issue tracking tool	Project Management
Pointing poker	Online tool for planning poker	Scrum planning
Chatgpt	Assistant for coding and spelling	Quality assurance
Postman	API Testing and debugging	Testing
Jest	Framework for unit testing	Testing
Lint	Analyze source code, report errors and bugs	Testing
Prettify	Improve code readability	Testing
CodeQL	Identify security vulnerabilities	Testing

5.2 Risk analysis

In table 2, various risks associated with our project are analyzed. These risks and their countermeasures are inspired by the risk analysis in [3]. In table 2, the risk is calculated based on the probability of the event occurring and the potential consequences it may have. The probability and consequences are further divided into tiers, as illustrated in the accompanying risk level table 1. The probability are divided into four tiers: unlikely, low, medium and high. Similarly, the consequences are also divided into four tiers: negligible, slight, serious and very serious. The classification provides a structured framework that increases the understanding of potential threats and their potential magnitude. This is useful for creating an efficient risk management and mitigation plan. The table also has color codes that identifies the risk. These color codes represents five tiers: Blue - Very Low risk, Green - Low risk, Yellow - Medium risk, Orange - High risk, Red - Very high risk. However, its important to note that putting risk events into this system will only be an estimation of what we expect the risk to be.

Risk Level		Consequences			
		Negligible	Slight	Serious	Very Serious
Likelihood	Unlikely	1	2	3	4
	Low	2	4	6	8
	Medium	3	6	9	12
	High	4	8	12	16

Table 1: Table showing risk tiers

Nr	Risk Scenarios	Probability	Consequences	Risk
1	The project is not completed by the deadline of May 21. Delays in the work can occur in various ways, such as poor planning and time management, technical challenges, illness, or personal reasons.	Unlikely	Very serious	Low
2	A team member gets ill and is unable to participate in a meeting or collaborate with the team on that particular day or week.	High	Slight	Medium
3	A group member chooses to quit the project. This may be caused by a poor working environment, low motivation, or personal reasons.	Unlikely	Very Serious	Low
4	Loss of data, code, or information. Loss of data/code can occur if we get merging conflicts without recovery possibilities, accidental deletion, or intentional deletion.	Medium	Slight	Medium
5	Code errors, bugs, and design flaws. Errors can either occur syntactically or logically, if we directly copy from an untrusted source, avoid using debugger and automated tests.	High	Slight	Medium
6	NINA may modify the mandatory specifications for the website, which can occur if they suddenly require a feature that was not initially included in the bachelor description.	Low	Serious	Medium

Table 2: Table showing risk scenarios

Risk Management Plan

Based on the risk analysis with six risk events, measures are developed that can be implemented to reduce the likelihood of a risk event occurring and, if the event were to occur, mitigate its consequences. These countermeasures are organized into a risk management plan, detailed in table 3. This table corresponds to each previously identified risk event, outlining specific mitigation strategies and countermeasures with the overarching goal of reducing the total risk and make the risk acceptable for the team. It also shows the estimated remaining risk for each risk event after the countermeasures from the risk management plan has been implemented.

Nr	Countermeasure	Remaining Risk
1	If we realize that we are unable to meet the deadline of the project, we must inform the product owner and discuss what tasks can be dropped. To reduce the risk, we should also ensure at all times that we are either ahead of schedule or, if not, on schedule. If it becomes obvious that this is unattainable, it should be communicated as early as possible in the project phase.	Very Low
2	While it's challenging to completely avoid getting ill, during the semester, we can take proactive measures to prepare for the situation. Since we have four group members we are more robust and resilient towards individuals facing illness. However, this requires the rest of the group to work even harder during this period.	Low
3	Efficient work routines can contribute to enhancing the collective working environment and thereby increasing the motivation of the group members. In addition, having social activities beyond bachelor meetings can contribute to creating a better collective environment as well.	Very Low
4	Git has version control, allowing you to view previous versions. You can also create multiple branches where you can work on separate code, minimizing merging conflicts. Additionally, with regular meetings and communication, the likelihood of accidental deletion because of misunderstandings decreases. Intentional deletion is unlikely, and with local backups for documentation it should be possible to restore any deleted documentation or code.	Low
5	In order to avoid errors and bugs, we can implement Github actions. With GitHub action we utilize CI/CD for automated testing like unit testing, code formatting, security scans, and versioning and release management.	Low
6	With clear communication and regularly weekly meetings, the likelihood of NINA modifying the mandatory specifications is greatly reduced. Staying proactive with our work allows us to better tolerate changes or additions to features as they arise.	Very low

Table 3: Table showing risk management plan

6 Implementation plan

The implementation plan for this project has been visualized using a Gantt Chart, which you can see in figure 1. The plan specifies when we intend to work on different parts of the project, and has divided our work into distinct parts; Pre-project, Webpage, Testing, Report, and Presentation. It also includes five major deadlines for our project.

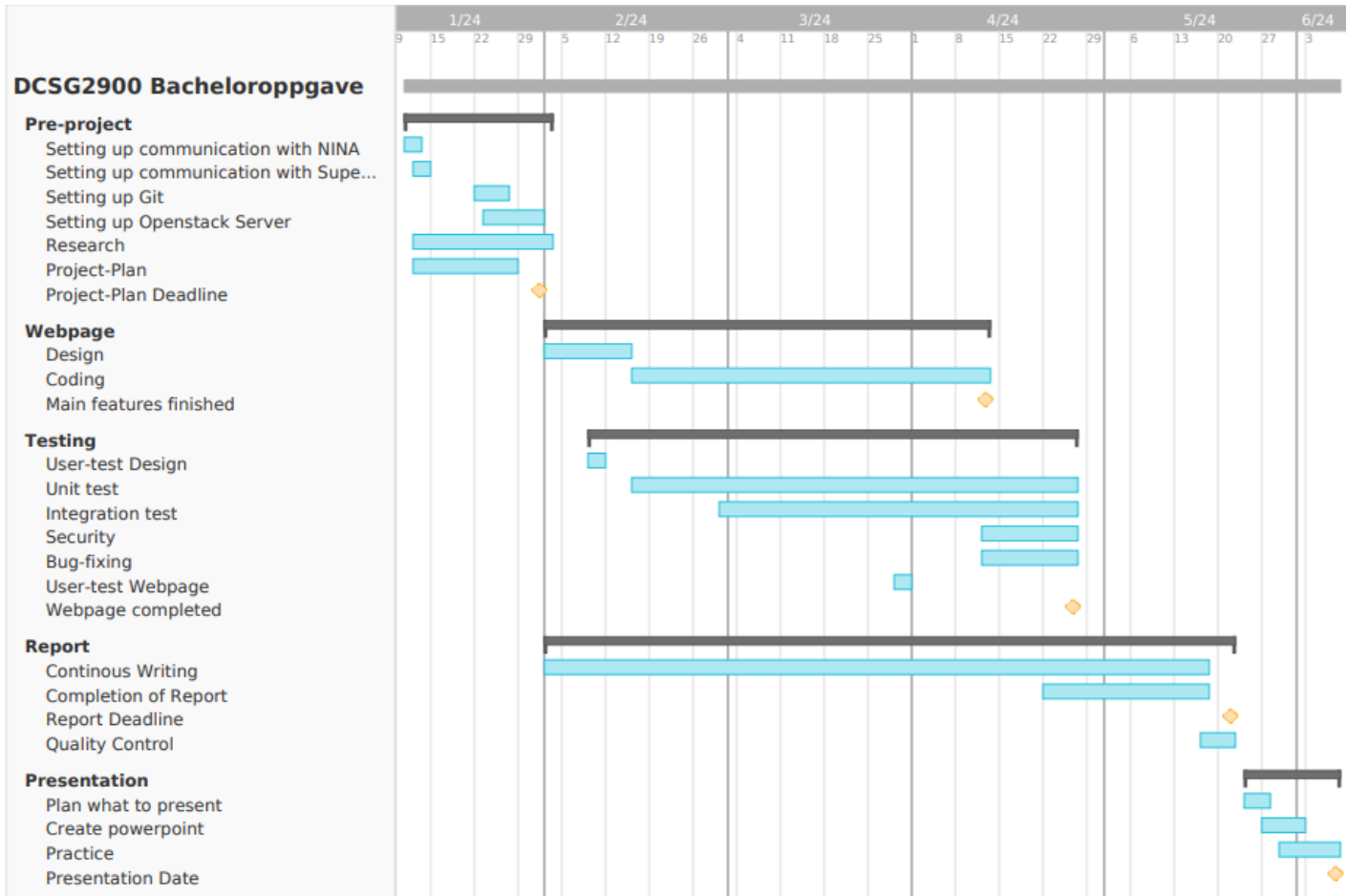


Figure 1: Gantt Chart

Bibliography

- [1] ATlassian. *Learn how to use burndown charts in Jira Software*. Accessed on 14.01.2024. 2023. URL: <https://www.atlassian.com/agile/tutorials/burndown-charts>.
- [2] Vincent Driessen. *A successful git branching model*. Accessed on 14.01.2024. 2010. URL: <https://nvie.com/posts/a-successful-git-branching-model/>.
- [3] Steffen Granberg, Steinar Opphus, and Ole André Slettum. *Viten i senter*. Bachelor's thesis accessed 2024-01-11. 2017. URL: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2461854>.
- [4] SCRUM GUIDES. *The 2020 Scrum Guide™*. Accessed on 14.01.2024. 2023. URL: <https://scrumguides.org/scrum-guide.html>.
- [5] NINA. *Norsk institutt for naturforskning (NINA)*. Accessed on 12.01.2024. URL: <https://www.nina.no/Om-NINA/Om-oss>.
- [6] Scrum.org. *Learn About the Scrum Artifact: Product Backlog*. Accessed on 14.01.2024. 2023. URL: <https://www.scrum.org/resources/what-is-a-product-backlog>.
- [7] Scrum.org. *What is a Daily Scrum?* Accessed on 14.01.2024. 2023. URL: <https://www.scrum.org/resources/what-is-a-daily-scrum>.
- [8] knowledgehut upGrad. *Why Do We Use Scrum ?* Accessed on 14.01.2024. URL: <https://www.knowledgehut.com/tutorials/scrum-tutorial/why-do-we-use-scrum>.

Appendix C

Group Rules

1 Group rules

1. All expenses will be split evenly between group members.
2. Always let the group know if you are ill or can't attend a meeting for some other important reason.
3. Work at least 30 hours a week, and it is expected to work more if required.
4. Create and update hours spend on the project in the worklog. This can be further discussed in project meetings.
5. Mandatory to attend all meetings, which includes project and status meetings, and supervisor and employer/customer meetings.
6. Always write meeting minutes.

Failure to comply with the rules, will result in extra work being given to this member to catch up to the others. If this work is not done, the group member could be expelled from the group or punished by paying for a meeting at McDonald's.

2 Routines

- Expected to work at least 4 hours each day from Monday-Friday, however if this is not enough to finish your designated tasks it's expected that you work more
- Every member should be available from 10:00-16:00 each workday, unless they have given a notice.
- Group members should be available for communication on weekends, and will be required to work on weekends if needed.
- The group will have weekly meetings with supervisors every Thursday 11:30
- The group will have meetings with NINA every other week at Wednesday 14.00 or more often when needed.
- Each sprint is 2 weeks, and concludes on Wednesday where we will have a meeting with NINA and a sprint review. The sprints starts on Thursday, where we will have sprint planning meetings.

Appendix D

Project Description

Oppgavetittel: Interaktiv database for miljødata

Bedrift: NINA Norsk institutt for naturforskning
Kontaktperson: Knut Marius Myrvold
E-post: knut.myrvold@nina.no
Telefon: 92064963
Lokasjon: Lillehammer

Beskrivelse av oppgaven

Tittel: Interaktiv database for miljødata

Naturovervåkingen i Norge tar mange ulike former, og genererer årlig store datamengder. For mange av måleparameterne finnes gode og etablerte lagringssystemer, gjerne finansiert av offentlig forvaltning. Annen type naturovervåking, som er mindre vanlig eller som ikke ennå er av et omfang som muliggjør offentlig finansiering, mangler ofte sikre og effektive lagringsløsninger.

Norsk Institutt for Naturforskning (NINA) jobber med utvikling av nye overvåkingsmetoder både på land og i vann, og bruker overvåkingsdata inn i sine forskningsoppgaver. En relativt ny metode i Norge er elektrisk fiske ved hjelp av båt. Her fiskes transekter i elver hvor hensikten gjerne er å undersøke artssammensetning, størrelsesfordeling og mengde fisk på ulike elvestrekninger. Datasettene fra hvert transekt inneholder typisk informasjon om art, lengde og vekt for hvert individ som ble fanget (nivå 1), samt informasjon om tid, sted (koordinater) og miljøforhold som for eksempel vanntemperatur (nivå 2). Vi kjører ofte mange transekter på forskjellige strekninger i samme vassdrag (nivå 3) og gjentar ofte arbeidet flere år på rad (nivå 4). Dette gjentas for alle vassdragene (nivå 5) vi overvåker – et antall som stadig øker.

Vi jobber for tiden med å etablere en databasestruktur, og ønsker også en egnet frontend som gjør det mulig å 1) enkelt importere data til databasen, 2) få en visuell oversikt over hvilke data som ligger i databasen og 3) interagere med databasen (eks. til nedlasting av data). Dette trenger vi hjelp til!

For eksempel ønsker vi ofte å kunne velge ut data fra en elv i en viss tidsperiode og visualisere hvor mange arter som var tilstede - og lengdefordelingen av individene for hver art - eller å sammenligne lengdefordelingen av fisk mellom to vassdrag. Slikt arbeid kan ofte være svært tidkrevende når dataene ligger spredt i enkelte excelark, samtidig som at potensialet for feil øker for hvert steg av den manuelle kombineringsprosessen.

Interesse for eller bakgrunn innen databaser og statistikk er ønskelig, samt en pragmatisk tilnærming til praktiske løsninger innen datahåndtering og hierarkiske datastrukturer. Utviklingsmiljø og konkrete oppgaver bestemmes i dialog med oppdragsgiver og vår IT-avdeling. Gruppen vil få tilgang til databasestrukturen med eksempelmateriale og vi legger opp til jevnlig kontakt gjennom semesteret. Her er noen foreløpige detaljer:

- Dere kan velge det programmeringsspråket, rammeverket og bibliotekene dere foretrekker, så lenge de har åpen kildekode
 - o Bedre hvis den brukes/vedlikeholdes aktivt
- Frontend skal konsumere REST API fra PostgREST på toppen av Postgres
 - o Enkle spørringer kan gjøres direkte på eksisterende tabeller
 - o Komplekse spørringer (f.eks. etter aggregerte data) kan gjøres i egendefinerte SQL views som kan defineres av brukeren
- Koden bør lagres i et Git repository med GPLv3 eller MIT lisens
 - o Kode formatter/linter/kontroller anbefalt
- Programvarekrav og avhengigheter bør være tydelig angitt
 - o Versjon pinning anbefalt (Conda anbefales ikke)
 - o Dockerfile/Containerfile anbefalt

Oppgaven passer en gruppe på 2-4 personer.

Appendix E

Work Hours

Work Hours

	Kavishnayan Nadarajah	Martin Solevåg Glærum	Carl Petter Mørch- Reiersen	Kevin Nikolai Mathisen
Before Project Start	102t	86t	83t	102t
Sprint 1 (01.02.24-14.02.24)	63t	60t	64t	67t
Sprint 2 (15.02.24-28.02.24)	67t	61t	64t	54t
Sprint 3 (29.02.24-13.03.24)	60t	56t	58t	100t
Sprint 4 (14.03.24-27.03.24)	56t	57t	50t	107t
Sprint 5 (28.03.24-10.04.24)	36t	36t	34t	61t
Sprint 6 (11.04.24-24.04.24)	65t	57t	56t	100t
Sprint 7 (25.04.24-08.05.24)	59t	56t	63t	73t
Sprint 8 (09.05.24-20.05.24)	61t	73t	68t	82t
Total	569t	539t	540t	746t

Appendix F

Sprint Reviews

Sprint review 1

Hva som gikk bra

Vi fikk gjort mye, jobbet effektivt. Fikk gjort:

- Risk assessment
- Requirement engineering
 - use case, misuse case
 - subject object matrix
 - data classification
 - quality gates
 - overview of external and internal requirements
 - security and functional requirements
- Design
 - Laget 2 iterasjoner av wireframe med bra tilbakemeldinger
 - Site map
 - 2 user tests
 - Data flow diagram
 - Threat modeling (identification, evaluation, analysis, controls)
 - Design principles

Vi hadde bra planlegging og struktur, noe scrum hjalp med. Alle deltok også aktivt på møter, jobbet aktivt og gjort oppgavene sine. Vi tok også møtetidspunkt mer seriøst enn tidligere i prosjektet.

Hva som kunne gått bedre

Vi kunne satt av tid til å lære seg tema ordentlig før man prøver der det er naturlig. Det var noen ganger vi begynte å gjøre ting uten at vi hadde research om temaet, som gjorde at vi senere måtte endre på arbeidet.

Vi kunne brukt flere rammeverk og kilder rundt prosesser, i stedet for å basere det på det vi har lært og kan. Dette vil sikre at vi bruker best practice, og dermed har referanser som vi kan bruke for å begrunne valgene våre.

Ved fordeling av arbeidsoppgaver ble det litt skjev fordeling når det kom til typer oppgaver. Vi kunne heller ha fordelt arbeidsoppgaver med mer hensyn på variasjon i arbeid, forhindre for mye repetitivt arbeid.

Ved flytting av møter var det flere ganger gitt for sen beskjed, som førte til at noen ikke fant ut av flytting før de var i møtet. Folk burde derfor gi beskjed rimelig tid i forveien. Hvis det er andre omstendigheter som påvirker arbeidssatsen til folk burde dette også være klart for de andre gruppemedlemmene.

Backlog

- Gjøre ferdig threat modeling
- Oppdatere risk assessment basert på ny requirements
- Skrive om design i rapport
- Sequence diagram
- Finpusse på diagram
- Potensiell restrukturering av tabeller og dokumentasjon
- Forprosjektplan retting

Sprint review 2

Hva som gikk bra

Har avklart hvordan vi skal løse oppgaven teknisk, som inkluderer hvilket framework vi skal bruke (Svelte), server (nginx), og også hvordan vi skal integrere løsningen vår med infrastrukturen til NINA.

Har også satt opp struktur for applikasjon og html og css.

Vi hadde bra planlegging og struktur, noe scrum hjalp med. Alle deltok også aktivt på møter, jobbet aktivt og gjort oppgavene sine.

Fikk også satt oss inn i Svelte og Sveltekit, og plotly.

Hva som kunne gått bedre

Vær mer spesifikk i hva man har gjort i arbeidsloggen.

Mer formelt forberede til møter. Skrive ned spørsmål i felles dokument, evt. navn til hvem som skal stille hvert spm. evt. sende spm til NINA i forkant.

Backlog

- Skrive om risk assessment i rapporten
- Flytte over CI/CD til github
- Sette opp mocking av PostgREST (Ved å sette opp database med testdata)

Sprint Review 3

Hva som gikk bra

- Vi har fått gjort mye, kommet langt på nettsiden
- Har blitt ish ferdig med MVP
- Har avklart struktur på rapporten
- Nina var fornøyd med produktet så langt

Hva som kunne gått bedre

- Kunne sett mer på security og testing, ikke bare programmering
- Kunne hatt mer balanse mellom teknisk og skriving på rapport
- Viktig å tenkte på å gi beskjeder til gruppen med god varsel
- Burde bruke Jira og arbeidslogg for å sikre at man ikke jobber på samme oppgaver parallelt.
- Kunne lagt opp til mer teknisk arbeid når man er hjemme i Oslo ved f.eks. ta med/låne ekstra skjerm, mus, tastatur

Backlog

- Download funksjonalitet
- Upload funksjonalitet
- Input validation
 - Validation of data fetching
- Button component
- Marker and map component
- Research implementations against cross site scripting
- Integrere key cloak/autentisering
- Skrive integration tests
- Se på end-to-end tests, http tests,
- Skrive om testing og CI/CD i rapport
- Skrive om collaboration in development process
- Oppdatere diagram basert på infrastruktur fra NINA
- Skrive om design i rapporten
- Skrive om risk assessment i rapporten
- Skrive om requirement engineering i rapporten (kravspec)

Sprint review 4

Hva som gikk bra

- Nina er fornøyd med produktet
- Ble nesten ferdig med nettsiden
- Deilig med påskeferie

Hva som kunne gått bedre

- Nina har ikke gjort ferdig excel endpoint eller autentisering
- Kunne jobbet ihvertfall 30 timer hver uke, noen ganger det ble litt under

Backlog

- Skrive mer på rapporten, begynne å fylle ut deler
- Excel upload funksjonalitet
- Se elver og stasjoner på liste siden
- Navigasjon på siden
- Styling på siden
- Cookies og autentisering på siden
- Plotly graf design/styling
- Generell cleanup i github
- Ekstra funksjonalitet:
 - Velge 'og' ved valg av arter
 - Vise kommentar ved stasjoner
 - Aggregere data på graf side
- Få postgres til å gjøre på dev servere

Sprint review 4

Hva som gikk bra

- Nådde nina sine krav
- Ble ferdig med funksjonalitet innen fristen vi satte i gantt skjema
- Ferdig med e2e tests
- Fikk løst autentisering
- Fikk løst upload

Hva som kunne gått bedre

- Mindre tid på tester, evt. nedprioritert når det ble for mye arbeid
- Skrevet mer på rapporten
- Kunne skrevet inn i arbeidsloggen mer regelmessig

Backlog

- Rydde opp i litt unit tests
- Oppdatere user manual
- Små fiks på koden
- Fullføre integrering med Nina
- Oppdatere risikorapport, og annen dokumentasjon/diagram
- Fullføre dokumentasjon av user test, få underskrift fra deltakere
- Jobbe videre på rapport

Sprint review 5

Hva som gikk bra

- Skrev mer på rapporten, over halvveis ferdig
- Konkrete oppgaver i Jira, lett å se hva folk jobbet med

Hva som kunne gått bedre

- Ikke noe spesielt

Backlog

- Oppdatere risk assessment
- Oppdatere requirements
- Fullføre pentesting
- Fullføre 90% av rapporten, inkludert discussion, conclusion, og abstract

Appendix G

Jira Sprint Burnup Chart

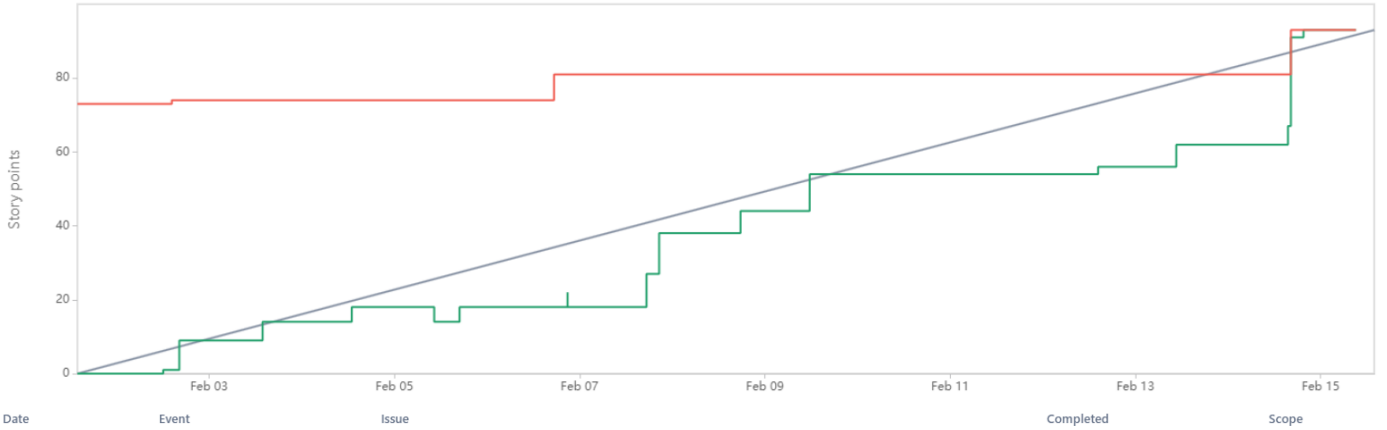
Sprint 1

Sprint: Estimation field:

Date - February 1st, 2024 - February 15th, 2024

Sprint goal - Finish design and requirements.

— **Completed work** (Number of story points completed this sprint)
— **Guideline** (Ideal burn rate)
— **Work Scope** (Number of story points to be completed this sprint)

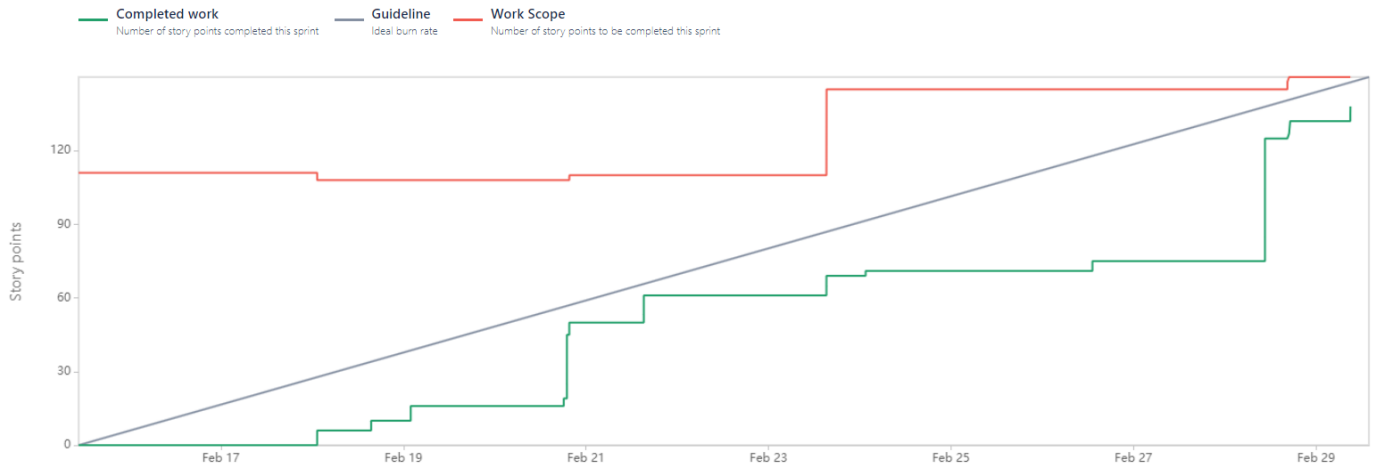


Date	Event	Issue	Completed	Scope
Thu, Feb 01 2024, 1:53pm	Sprint started	<ul style="list-style-type: none"> SCRUM-4 Asset management SCRUM-5 Identifying risk SCRUM-8 Abuse cases SCRUM-7 Use cases SCRUM-9 Initial wireframe SCRUM-6 Security requirements SCRUM-11 Dataflow diagram SCRUM-10 Design and security principles SCRUM-12 Threat modeling SCRUM-20 Wireframe iteration SCRUM-22 Context establishment SCRUM-21 Quality gates SCRUM-13 Site map SCRUM-24 Mitigating risk SCRUM-23 Analyzing risk SCRUM-15 User test 1 SCRUM-14 User test 1 planning SCRUM-25 Risk acceptance SCRUM-17 User test 2 planning SCRUM-16 User test 1 review SCRUM-19 User test 2 review SCRUM-18 User test 2 	0	73

Sprint 2

Sprint: SCRUM Sprint 2 Estimation field: Story points

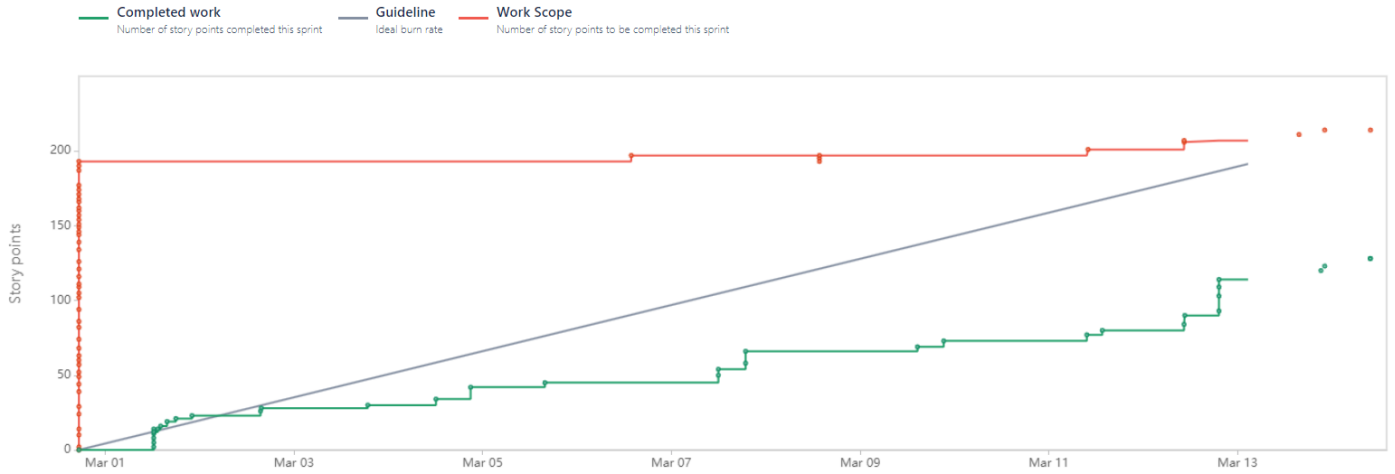
Date - February 15th, 2024 - February 29th, 2024



Date	Event	Issue	Completed	Scope
Thu, Feb 15 2024, 10:35am	Sprint started	<ul style="list-style-type: none"> SCRUM-28 Update risk assessment and requirements based on new requirements SCRUM-29 Complete Threat modeling SCRUM-31 Write about User tests in report SCRUM-30 Write about Wireframes in report SCRUM-33 Update DFD diagram SCRUM-32 Write about site map in report SCRUM-35 Correct Preproject plan SCRUM-34 Write about risk assessment in report under development process SCRUM-36 Prepare Git Repo SCRUM-40 Plan Interfaces/modules/architecture SCRUM-42 Research PostgREST and Fetching SCRUM-41 Research plotly.js SCRUM-44 Set up basic nginx server, development server SCRUM-43 Research Svelte, Javascript rendering with JSON SCRUM-46 Initial list page html/css SCRUM-45 Initial map page html/css SCRUM-48 Initial download page html/css SCRUM-47 Initial graph page html/css SCRUM-49 Initial upload page html/css SCRUM-50 Initial javascript structure/modules SCRUM-37 Set up skyhigh SCRUM-39 Set up and Research Docker SCRUM-38 Set up test PostgREST mocking 	0	111
Sun, Feb 18 2024, 7:00pm	Added to sprint	SCRUM-53 Add SSL certificate	10	108
Wed, Feb 28 2024, 4:54pm	Added to sprint	SCRUM-57 Create Logical and Physical Topology	125	148
Wed, Feb 28 2024, 4:28pm	Added to sprint	SCRUM-56 Update Report Structure	125	145

Sprint 3

Date - February 29th, 2024 - March 14th, 2024

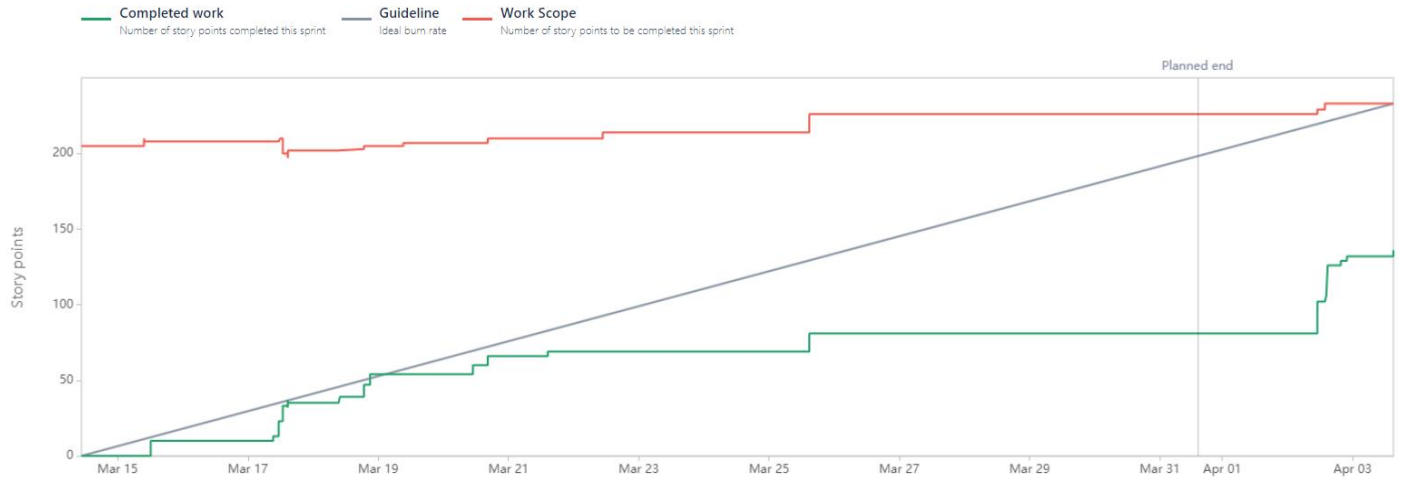


Date	Event	Issue	Completed	Scope
Thu, Feb 29 2024, 5:20pm	Sprint started		0	0
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-98 Configure dependency management, secure supply chain using dependabot	0	0 → 2
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-34 Write about risk assessment in report under development process	0	2 → 10
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-38 Set up test PostgREST mocking	0	10 → 14
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-60 Start writing Requirement gathering	0	14 → 24
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-62 Create project structure	0	24 → 29
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-61 Write about design of software architecture	0	29 → 39
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-64 Start writing about collaboration in Development process	0	39 → 44
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-63 Start writing Data classification	0	44 → 49
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-51 Set up CI/CD and workflows on main github	0	49 → 52
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-58 Start writing background	0	52 → 57
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-80 Convert Download Page to Svelte	0	57 → 60
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-82 Split up filter in new, smaller components	0	60 → 63
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-84 Set up Javascript for basic interactivity on the list page	0	68 → 74
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-83 Set up Javascript for basic interactivity on the map page	0	74 → 82
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-86 Set up Javascript for basic interactivity on the upload page	0	82 → 86
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-85 Set up Javascript for basic interactivity on the graph page	0	86 → 94
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-88 Set up Error handling structure in Sveltekit	0	94 → 102
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-66 Update diagrams based on new structure and documentation from NINA	0	102 → 105
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-87 Set up Javascript for basic interactivity on the download page	0	105 → 109
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-65 Update wireframe	0	109 → 111
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-68 Research and document integration tests and end to end tests	0	111 → 116
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-89 Research and document how we will integrate authentication	0	116 → 121
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-67 Write e2e tests	0	121 → 126

Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-91 Create javascript module for fetching of data, integrate with postgREST	0	126 → 134
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-90 Research and document how we will generate files for download	0	134 → 139
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-93 Research input validation	0	139 → 144
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-71 Update Dockerfile	0	144 → 146
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-92 Research CSP and how to configure it with Sveltekit	0	146 → 149
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-70 Set up automated testing and deployment on skyhigh servers	0	149 → 151
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-95 Start writing preface	0	151 → 154
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-73 Implement upload of excel file to remote API	0	154 → 157
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-94 Research website specific implementations against cross site scripting, look at risk assessment	0	157 → 160
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-72 Create Nginx config	0	160 → 162
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-97 Implement validation on website	0	162 → 166
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-75 Update HTML and CSS to reflect new Wireframe	0	166 → 168
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-96 Set up basic modal components	0	168 → 171
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-74 Set up SQL views	0	171 → 174
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-77 Convert List Page to Svelte	0	174 → 177
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-76 Plan and document Svelte components	0	177 → 187
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-79 Convert Upload Page to Svelte	0	187 → 190
Thu, Feb 29 2024, 5:20pm	Added to sprint	SCRUM-78 Convert Graph Page to Svelte	0	190 → 193
Wed, Mar 06 2024, 1:48pm	Added to sprint	SCRUM-99 Set up initial unit tests	45	193 → 197
Fri, Mar 08 2024, 1:39pm	Added to sprint	SCRUM-101 Create javascript module for filtering of data	66	193 → 197
Mon, Mar 11 2024, 9:58am	Added to sprint	SCRUM-102 Basic leaflet functionality	77	197
Tue, Mar 12 2024, 10:23am	Added to sprint	SCRUM-103 Implement dynamic markers based on data on page	84	201
Wed, Mar 13 2024, 3:37pm	Added to sprint	SCRUM-104 Create button component	114	207 → 211
Wed, Mar 13 2024, 10:09pm	Added to sprint	SCRUM-105 Create UI svelte components with javascript on download page	120	211

Sprint 4

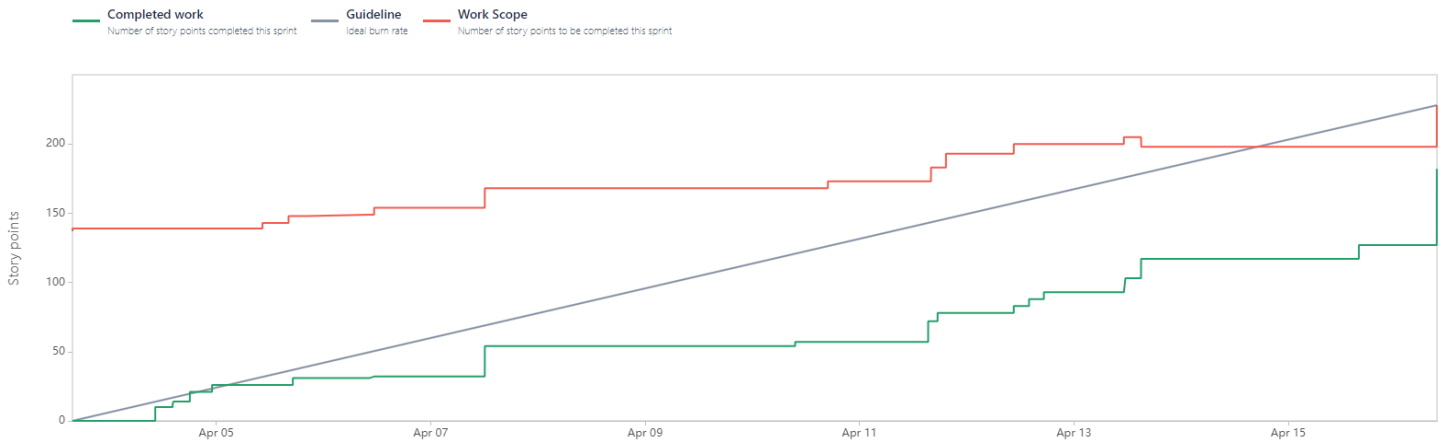
Date - March 14th, 2024 - March 31st, 2024



Date	Event	Issue	Completed	Scope
Thu, Mar 14 2024, 10:39am	Sprint started	SCRUM-60 Start writing Requirement gathering SCRUM-62 Create project structure SCRUM-61 Write about design of software architecture SCRUM-64 Start writing about collaboration in Development process SCRUM-63 Start writing Data classification SCRUM-66 Update diagrams based on new structure and documentation from NINA SCRUM-68 Research and document integration tests and end to end tests SCRUM-89 Research and document how we will integrate authentication SCRUM-67 Write e2e tests SCRUM-93 Research input validation SCRUM-73 Implement upload of excel file to remote API SCRUM-94 Research website specific implementations against cross site scripting, look at risk assessment SCRUM-97 Implement validation on website SCRUM-34 Write about risk assessment in report under development process SCRUM-104 Create button component SCRUM-103 Implement dynamic markers based on data on page SCRUM-100 Fixing linting errors SCRUM-106 Create git flow diagram SCRUM-107 Start writing about Theory SCRUM-108 Write README.md on github SCRUM-109 Document dependencies used SCRUM-110 Update risk assessment SCRUM-111 Update dockerfile SCRUM-112 Fix unit test errors on github SCRUM-113 Write unit tests for svelte components created so far SCRUM-114 Write unit tests for javascript modules created so far SCRUM-115 Create summary page for rivers SCRUM-116 Create summary page for stations SCRUM-117 Implement setting and retrieving variables from url SCRUM-118 Implementing navigation on page using buttons SCRUM-119 Create module for formatting of plotly data SCRUM-120 Extracting hardcoded text values from svelte components into constants. Extract javascript functions from svelte components into separate javascript modules. clean up code not following standardjs SCRUM-121 Implement accessibility in html and css SCRUM-122 Implement website responsiveness using flexbox, relative values, and media queries SCRUM-123 Create code for handling creation of excel and csv files SCRUM-124 Refactor javascript functions for functional programming SCRUM-125 Create styling for download page SCRUM-126 Create custom scroll bar	0	205
Fri, Mar 15 2024, 9:43am	Added to sprint	SCRUM-127 Create new dockerfile and docker compose for svelte and mock server	0	205
Sun, Mar 17 2024, 2:42pm	Added to sprint	SCRUM-129 Create module for creating download data	35	197
Tue, Mar 19 2024, 9:16am	Added to sprint	SCRUM-130 Organize components and modules in subfolders, update paths to use relative paths in svelte	54	205
Wed, Mar 20 2024, 4:22pm	Added to sprint	SCRUM-131 Create unit tests for calculateData module	66	207
Mon, Mar 25 2024, 2:52pm	Added to sprint	SCRUM-133 Set up postgres and postgres with test data for local testing	81	221
Tue, Apr 02 2024, 1:49pm	Added to sprint	SCRUM-134 Implement postgres with real data into application	102	229

Sprint 5

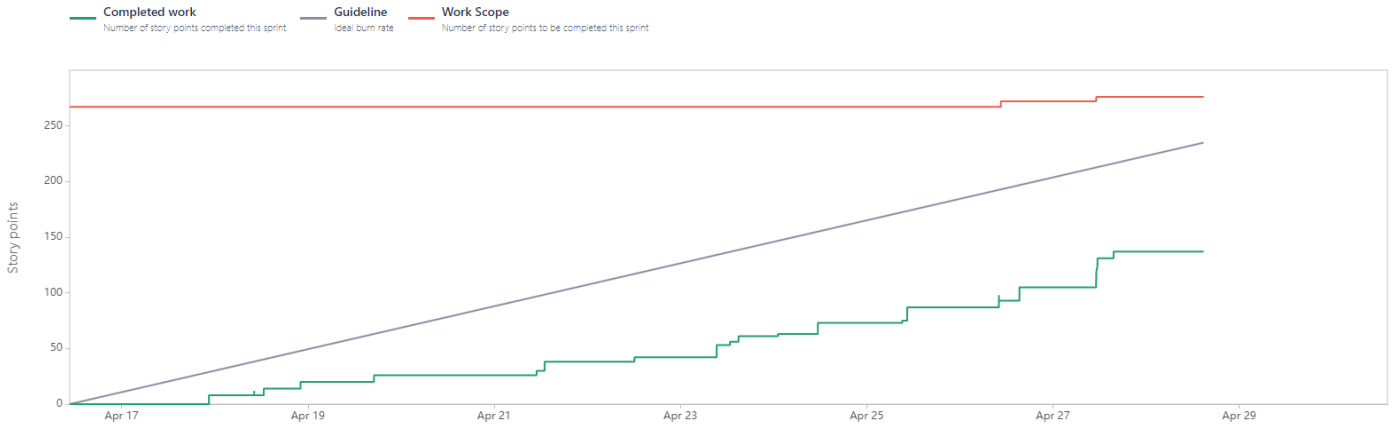
Date - April 3rd, 2024 - April 15th, 2024



Date	Event	Issue	Completed	Scope
Wed, Apr 03 2024, 3:48pm	Sprint started	SCRUM-122 Implement website responsiveness using flexbox, relative values, and media queries SCRUM-121 Implement accessibility in html and css SCRUM-120 Extracting hardcoded text values from svelte components into constants, Extract javascript functions from svelte components into separate javascript modules, clean up code not following standardjs SCRUM-60 Start writing Requirement gathering SCRUM-61 Write about design of software architecture SCRUM-63 Start writing Data classification SCRUM-66 Update diagrams based on new structure and documentation from NINA SCRUM-89 Research and document how we will integrate authentication SCRUM-67 Write e2e tests SCRUM-118 Implementing navigation on page using buttons SCRUM-117 Implement setting and retrieving variables from url SCRUM-110 Update risk assessment SCRUM-73 Implement upload of excel file to remote API SCRUM-97 Implement validation on website SCRUM-34 Write about risk assessment in report under development process SCRUM-126 Create custom scroll bar SCRUM-107 Start writing about Theory SCRUM-130 Organize components and modules in subfolders, update paths to use relative paths in svelte SCRUM-125 Create styling for download page SCRUM-124 Refactor javascript functions for functional programming SCRUM-135 Clean up component unit tests SCRUM-136 Set up postgres on development servers SCRUM-137 Integrate svelte application with nina docker infrastructure SCRUM-138 Update styling for plotly components SCRUM-139 Add satellite view for leaflet SCRUM-140 Fix bugs, add history SCRUM-141 Fix selection of markers on map page and error messages	0	137
Fri, Apr 05 2024, 10:19am	Added to sprint	SCRUM-142 Research penetration testing	26	139
Fri, Apr 05 2024, 4:10pm	Added to sprint	SCRUM-143 River and station summary page on list page	26	143
Fri, Apr 05 2024, 8:21pm	Added to sprint	SCRUM-144 Update buttons	31	148
Sat, Apr 06 2024, 11:19am	Added to sprint	SCRUM-145 User manual	32	149
Thu, Apr 11 2024, 4:00pm	Added to sprint	SCRUM-146 Login in page and functionality	72	173
Fri, Apr 12 2024, 10:32am	Added to sprint	SCRUM-147 Fix bugs and add species selection on download page	83	195
Sat, Apr 13 2024, 11:20am	Added to sprint	SCRUM-148 Update user manual images and check spelling mistakes	93	205

Sprint 6

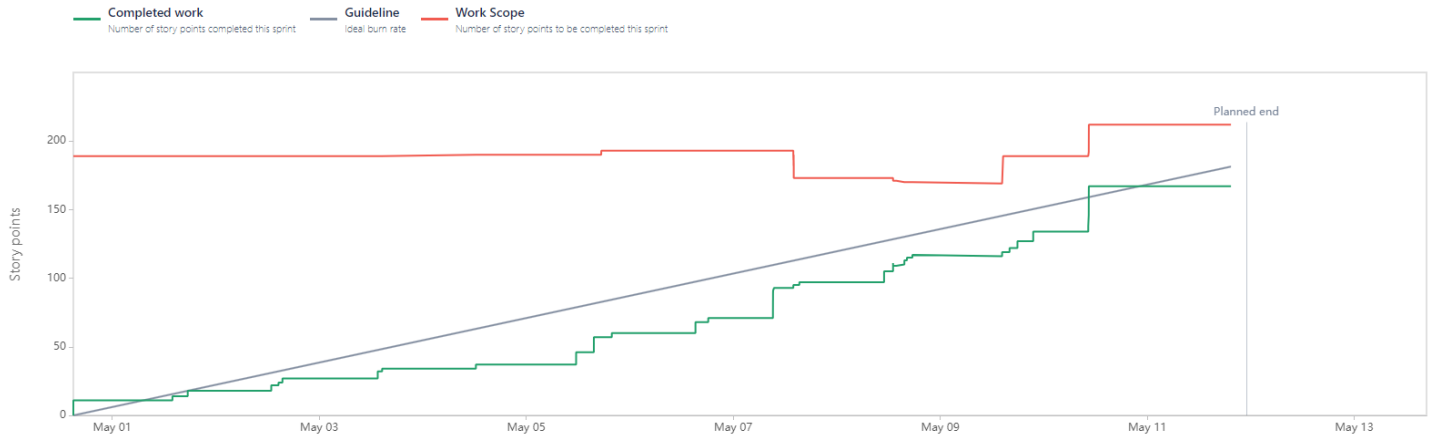
Date - April 16th, 2024 - April 30th, 2024



Date	Event	Issue	Completed	Scope
Tue, Apr 16 2024, 10:36am	Sprint started	<ul style="list-style-type: none"> SCRUM-137 Integrate svelte application with nina docker infrastructure SCRUM-135 Clean up component unit tests SCRUM-60 Start writing Requirement gathering SCRUM-61 Write about design of software architecture SCRUM-63 Start writing Data classification SCRUM-66 Update diagrams based on new structure and documentation from NINA SCRUM-110 Update risk assessment SCRUM-34 Write about risk assessment in report under development process SCRUM-148 Update user manual images and check spelling mistakes SCRUM-149 Update use case diagram SCRUM-150 Update sitemap SCRUM-151 Update data flow diagram SCRUM-152 Finalize website SCRUM-153 Complete user test documentation, get signatures SCRUM-154 Write about Subject object matrix SCRUM-155 Write about use case and misuse case diagram SCRUM-156 Write theory SCRUM-157 Write about requirement engineering SCRUM-158 Write about quality gates SCRUM-160 Write about site map SCRUM-161 Write about wireframe SCRUM-162 Write about threat modeling SCRUM-163 Write about implementation page structure SCRUM-165 Write about implementation code structure SCRUM-167 Write about design principles followed SCRUM-168 Write about implementation Retrieval and storage of data SCRUM-169 Write about implementation handling data SCRUM-171 Write about implementation File handling SCRUM-172 Write about implementation map/leaflet SCRUM-173 Write about implementation plotly SCRUM-174 Write about implementation navigation SCRUM-175 Write about implementation error handling SCRUM-176 Write about implementation style SCRUM-177 Write about implementation accessibility SCRUM-178 Write about security authentication solution SCRUM-179 Write about security encryption SCRUM-180 Write about security CSP SCRUM-181 Write about security data validation SCRUM-182 Write about security output encoding SCRUM-183 Write about testing qithub actions SCRUM-184 Write about testing unit tests SCRUM-185 Write about testing end to end tests SCRUM-186 Write about testing security scans SCRUM-187 Write about testing linting SCRUM-190 Write about testing accessibility tests SCRUM-191 Write about testing mock server SCRUM-192 Write about testing deployment on test server SCRUM-193 Write about testing user tests SCRUM-194 Write about integration docker SCRUM-195 Write about integration nginx SCRUM-196 Write about integration NINA infrastructure SCRUM-197 Write about integration environment variables SCRUM-199 Write about development / collaboration scrum and agile SCRUM-200 Write about development / collaboration jira SCRUM-201 Write about development / collaboration Communication channels SCRUM-202 Write about development / collaboration git repo SCRUM-203 Write about development jsdoc and commenting of code SCRUM-204 Write about development / collaboration dependency management SCRUM-205 Write about development / collaboration microsoft software development lifecycle 	0	267
Fri, Apr 26 2024, 10:34am	Added to sprint	SCRUM-223 Improve introduction	93	267

Sprint 7

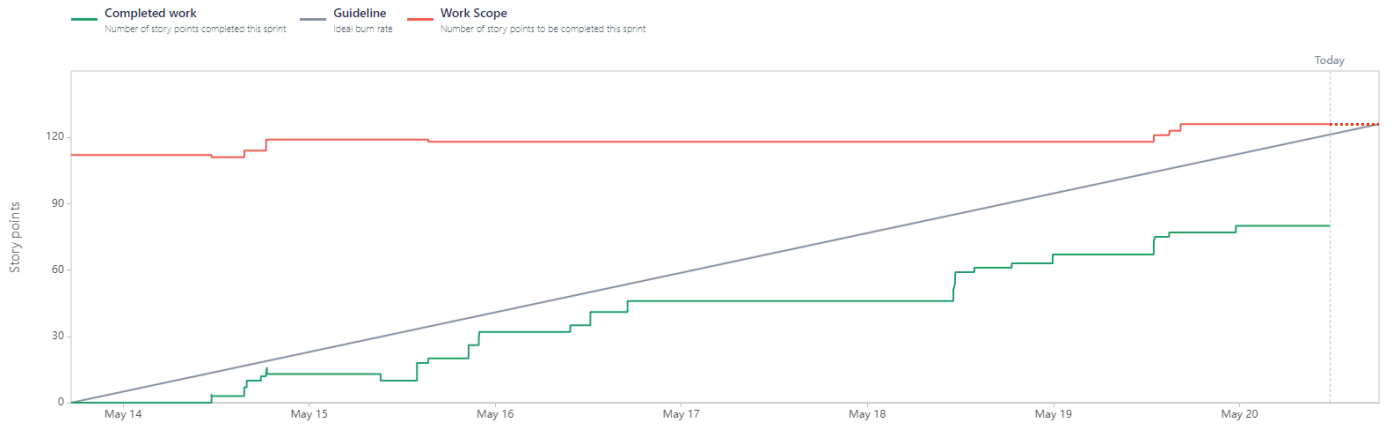
Date - April 30th, 2024 - May 11th, 2024



Date	Event	Issue	Completed	Scope
Tue, Apr 30 2024, 2:59pm	Sprint started	<ul style="list-style-type: none"> SCRUM-184 Write about testing unit tests SCRUM-183 Write about testing github actions SCRUM-182 Write about security output encoding SCRUM-186 Write about testing security scans SCRUM-181 Write about security data validation SCRUM-180 Write about security CSP SCRUM-179 Write about security encryption SCRUM-174 Write about implementation navigation SCRUM-178 Write about security authentication solution SCRUM-177 Write about implementation accessibility SCRUM-175 Write about implementation error handling SCRUM-205 Write about development / collaboration microsoft software development lifecycle SCRUM-158 Write about quality gates SCRUM-193 Write about testing user tests SCRUM-110 Update risk assessment SCRUM-191 Write about testing mock server SCRUM-190 Write about testing accessibility tests SCRUM-34 Write about risk assessment in report under development process SCRUM-196 Write about integration NINA infrastructure SCRUM-195 Write about integration nginx SCRUM-194 Write about integration docker SCRUM-197 Write about integration environment variables SCRUM-192 Write about testing deployment on test server SCRUM-222 Write abstract SCRUM-189 Write about testing pentests SCRUM-221 Write about conclusion SCRUM-220 Write about discussion sustainability SCRUM-215 Write about discussion integration SCRUM-213 Write about discussion development SCRUM-212 Write about discussion progressive enhancement SCRUM-219 Write about discussion use of AI SCRUM-218 Write about discussion evaluation of group work SCRUM-217 Write about discussion security SCRUM-216 Write about discussion improvements SCRUM-211 Write about discussion dependency pinning development SCRUM-210 Write about learning goal results SCRUM-206 Write about functionality results SCRUM-224 Final review introduction SCRUM-225 Write theory SCRUM-226 Continue and complete writing devops SCRUM-227 Update requirements SCRUM-229 Requirement analysis SCRUM-230 Requirement specification SCRUM-231 Requirement verification 	0	189
Sun, May 05 2024, 5:23pm	Added to sprint	SCRUM-232 update documentation in repo	57	190
Sun, May 05 2024, 5:24pm	Added to sprint	SCRUM-233 refactor and comment code	57	190
Tue, May 07 2024, 1:59pm	Added to sprint	SCRUM-235 write about discussion nina architecture changing and documentation	95	186
Tue, May 07 2024, 1:59pm	Added to sprint	SCRUM-236 write about discussion design choice of architecture	95	188
Wed, May 08 2024, 11:02am	Added to sprint	SCRUM-238 write about discussion performance tests	105	172
Thu, May 09 2024, 2:35pm	Added to sprint	SCRUM-240 review whole report	119	169
Fri, May 10 2024, 10:28am	Added to sprint	SCRUM-241 read trough and correct report whole group	163	192

Sprint 8

Date - May 13th, 2024 - May 20th, 2024



Date	Event	Issue	Completed	Scope
Mon, May 13 2024, 5:18pm	Sprint started	SCRUM-235 write about discussion nina archite ture changing and documentation SCRUM-242 Fix comments in introduction SCRUM-243 Create screenshots showing final product SCRUM-244 Add sources (thesis.bib) and \cite in report SCRUM-245 Add all acronyms to acronyms (add page numbers) SCRUM-246 Write introductions to chapters and sections, ensuring better flow. Remove headings where appropriate SCRUM-247 Fix comments in theory, remove redundant text SCRUM-248 Add references to theory chapter SCRUM-249 Finish acknowledgements SCRUM-250 Fix comments in development/chapter 3, reference more to figures in text. SCRUM-251 Remove meetings/ add some text about meetings in scrum/sprint SCRUM-252 Write about SBOM in dependencies SCRUM-253 Fix comments in chapter 4, make use case figure bigger. SCRUM-254 Add paragraph explaining more about use case and misuse case in report SCRUM-255 Convert figure to pdf/svg SCRUM-256 Fix comments in chapter 5 SCRUM-257 Fix comments in chapter 6, cut down redundant information, extract to separate document SCRUM-258 Get source for regex pattern SCRUM-259 Get academic/book source for scrum SCRUM-260 Fix comments chapter 7 SCRUM-261 Fix comments chapter 8 SCRUM-262 Fix comments chapter 9 SCRUM-263 Fix comments chapter 10 SCRUM-264 Fix comments chapter 11, reference and discuss each result goal separately. SCRUM-265 Final formatting (ensure code on same page, text breaks correct, etc.) SCRUM-266 Get/generate frontpage from ntnu SCRUM-267 Create inspera upload	0	112
Tue, May 14 2024, 3:38pm	Added to sprint	SCRUM-268 Update requirements	7	111
Tue, May 14 2024, 3:38pm	Added to sprint	SCRUM-269 Clean up threat model	7	111
Tue, May 14 2024, 6:27pm	Added to sprint	SCRUM-270 Fix attachments by creating separate pdf files for each with their own short frontpage/introduction.	13	114
Sun, May 19 2024, 12:57pm	Added to sprint	SCRUM-272 create attachment showing the final product	73	118
Sun, May 19 2024, 2:56pm	Added to sprint	SCRUM-273 Fix final comments for the entire bachelor	77	121
Sun, May 19 2024, 2:56pm	Added to sprint	SCRUM-274 fix sources in attachments	77	121
Sun, May 19 2024, 2:59pm	Added to sprint	SCRUM-275 create work hours and jira screenshot attachment	77	123
Sun, May 19 2024, 3:03pm	Added to sprint	SCRUM-276 create example screenshot of issue board Jira	77	123
Sun, May 19 2024, 3:15pm	Added to sprint	SCRUM-277 Create KI-skjema	77	123

Appendix H

Use case diagram and Misuse case diagram

Use case

The following are the use case diagram developed for the application. The interaction between the use cases are visualised in the figure, and interactions are described in detail in the tables following after the use case diagram.

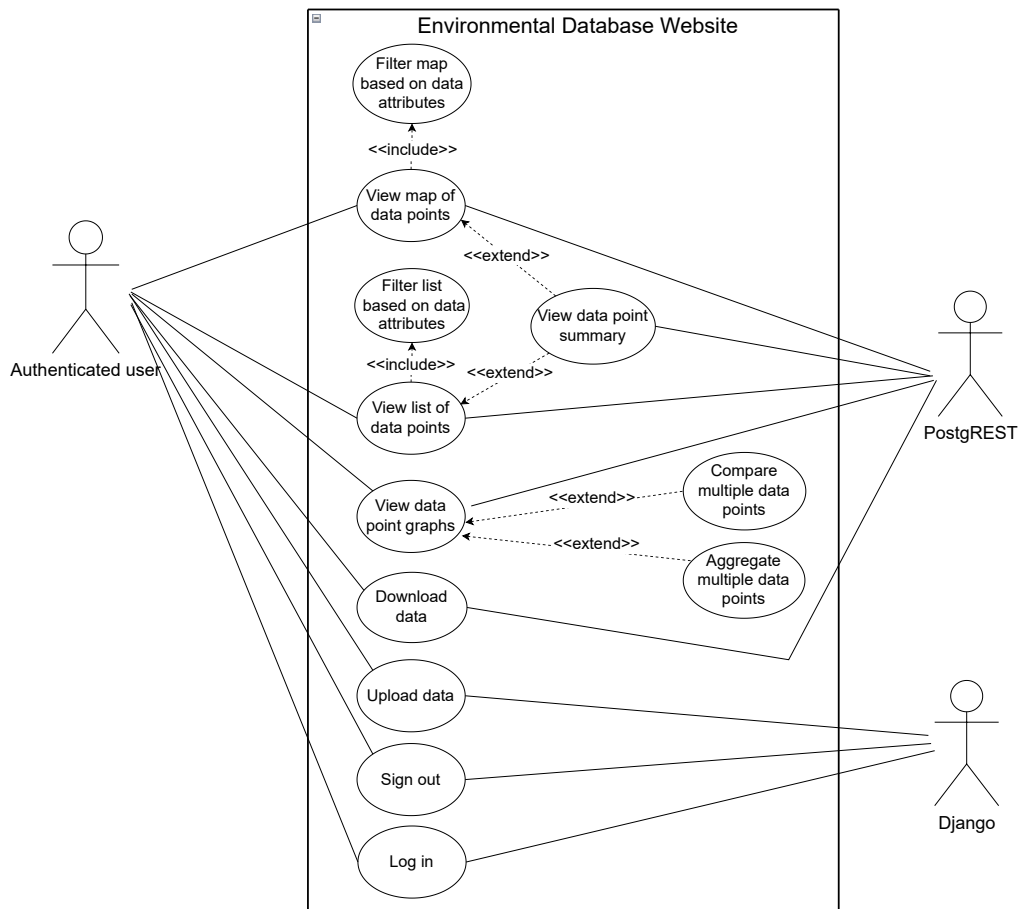


Figure H.1: Use case diagram

UC Name	Filter map based on data attributes
Actor	User
Trigger	Press button to open filters
Pre-condition	Be on the map the page
Post-condition	The data points shown on the map are filtered based on the data attributes.
Main Flow	<ol style="list-style-type: none"> 1. User choose type of data point, either “Stasjonsdata” or “Elvedata” 2. User choose years to filter on 3. User choose fish type to filter on
Exception	<p>The user selects combination of attributes for which no data points exist:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that no data points exist for the selected year 2. The user clicks ok <p>There is a problem getting data from the server:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that there was a problem retrieving data 2. The user clicks ok

Table H.1: Use case: Filter map based on data attributes

UC Name	View map of data points
Actor	User, PostgREST
Trigger	Open frontpage of website. Click map icon in the header
Pre-condition	
Post-condition	User views data points on map
Main Flow	<ol style="list-style-type: none"> 1. User filters what is shown using the use case “Filter map based on data attributes” 2. The data is loaded from PostgREST 3. User zooms in and pans on the map to find data points they want to look at 4. User clicks on a data point to view their data, opening the use case “View data point summary”
Exception	<p>There is a problem getting data from the server:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that there was a problem retrieving data 2. The user clicks ok <p>There is a problem getting the map:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that there was a problem retrieving the map 2. The user clicks ok

Table H.2: Use case: View map of data points

UC Name	View data point summary
Actor	User, PostgREST
Trigger	Click on a data point in the map or the list
Pre-condition	Be on the map page or list page
Post-condition	The user can view data for the selected point
Main Flow	<ol style="list-style-type: none"> 1. The data is loaded from PostgREST 2. The type of data shown is based on the type of data point, either “Stasjonsdata” or “Elvedata” 3. The user can click on the view graphs button to bring them to the use case “View data point graphs” 4. The user can click on the download button to bring them to the use case “Download data”
Exception	<p>There is a problem getting data from the server:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that there was a problem retrieving data 2. The user clicks ok

Table H.3: Use case: View data point summary

UC Name	Filter list based on data attributes
Actor	User
Trigger	Press button to open filters
Pre-condition	Be on the map page
Post-condition	The data points shown on the map are filtered based on the data attributes
Main Flow	<ol style="list-style-type: none"> 1. User chooses type of Data point, either “Stasjonsdata” or “Elvedata” 2. User chooses years to filter on 3. User chooses fish type to filter on 4. User search for data by typing in river names or project IDs in the search field
Exception	<p>The user selects combination of attributes for which no data points exist:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that no data points exist for the selected year 2. The user clicks ok <p>There is a problem getting data from the server:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that there was a problem retrieving data 2. The user clicks ok

Table H.4: Use case: Filter list based on data attributes

UC Name	View list of data points
Actor	User
Trigger	Click list icon in the header
Pre-condition	
Post-condition	User can see data points in a list
Main Flow	<ol style="list-style-type: none"> 1. The data is loaded from PostgREST 2. User filter what is shown using the use case “Filter list based on data attributes” 3. User can scroll to find any data points they want to look more at 4. User clicks on a data point to view their data, opening the use case “View data point summary”
Exception	<p>There is a problem getting data from the server:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that there was a problem retrieving data 2. The user clicks ok

Table H.5: Use case: View list of data points

UC Name	View data as graphs
Actor	User, PostgREST
Trigger	Click on the view graphs button on a data point summary page, or click graph icon in header
Pre-condition	
Post-condition	The user can graphs data for the selected point
Main Flow	<ol style="list-style-type: none"> 1. The data is loaded from PostgREST 2. If the data point selected is “Elvedata”, all of the stations under “Elvedata” will be selected for comparison, which will start the “Compare multiple data points” use case 3. The user can choose which species they want to plot 4. The user chooses to plot the distribution of fish length for the species selected, or choose to show the distribution of the fish species selected 5. If multiple stations are selected the user can choose to compare or aggregate the data. This will start the “Compare multiple data points” or “Aggregate multiple data points” use cases 6. The user can choose to download the data for the selected data points and species, which will start the “Download data” use case
Exception	<p>There is a problem getting data from the server:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that there was a problem retrieving data 2. The user clicks ok <p>There was a problem processing the data:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that there was a problem processing the data 2. The user clicks ok

Table H.6: Use case: View data as graphs

UC Name	Upload data
Actor	Authenticated user, Unauthenticated user, PostgREST, Authentication server
Trigger	Click on the upload icon in the header
Pre-condition	
Post-condition	The data selected from the user is uploaded to PostgREST
Main Flow	<ol style="list-style-type: none"> 1. The user selects a xlsx file to upload 2. The file is checked for format and content 3. The user clicks upload file 4. The website checks if the user is authenticated using the authentication server 5. The file is uploaded to PostgREST 6. The file is checked for format and content 7. The user gets notified that the upload was successful
Exception	<p>The user is not authenticated:</p> <ol style="list-style-type: none"> 1. The user gets a popup informing them they have to be logged in 2. The user clicks ok <p>The file has invalid content of invalid format:</p> <ol style="list-style-type: none"> 1. The user gets a popup informing them the data selected is invalid. 2. The user clicks ok <p>The upload failed:</p> <ol style="list-style-type: none"> 1. The user gets a popup informing them that the data failed to upload 2. The user clicks ok

Table H.7: Use case: Upload data

UC Name	Download data
Actor	Authenticated user, Unauthenticated user, PostgREST, Authentication server
Trigger	Click on the download button on a data point summary page, or click download icon in header
Pre-condition	
Post-condition	User gets a csv or xlsx file containing the data selected for download
Main Flow	<ol style="list-style-type: none"> 1. The user selects what data they want to download 2. The user clicks download 3. The website checks if the user is authenticated using the authentication server 4. The data is retrieved from PostgREST 5. The user chooses where to download the file
Exception	<p>The user is not authenticated:</p> <ol style="list-style-type: none"> 1. The user gets a popup informing them they have to be logged in 2. The user clicks ok

Table H.8: Use case: Download data

UC Name	Compare multiple data points
Actor	User
Trigger	Choosing multiple data points and to compare them on the graph page
Pre-condition	The user is on the graph page
Post-condition	The user can compare the data from two or more data points
Main Flow	<ol style="list-style-type: none"> 1. The user can choose which data points to compare 2. The user can then choose to compare the distribution of fish length for the species selected, or to compare the distribution of fish species between the stations
Exception	<p>If the user chooses more than 10 stations:</p> <ol style="list-style-type: none"> 1. The user will not be able to select more data points 2. The user clicks ok <p>If there is an error when comparing the data:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that there was a problem when comparing the data 2. The user clicks ok

Table H.9: Use case: Compare multiple data points

UC Name	Aggregate multiple data points
Actor	User
Trigger	Choosing multiple data points and to aggregate them on the graph page
Pre-condition	The user is on the graph page
Post-condition	The user can aggregate the data from two or more data points
Main Flow	<ol style="list-style-type: none"> 1. The user can choose which data points to compare 2. The user can then choose to compare the distribution of fish length for the species selected, or to compare the distribution of fish species between the stations
Exception	<p>If there is an error when comparing the data:</p> <ol style="list-style-type: none"> 1. The user gets a popup explaining that there was a problem when comparing the data 2. The user clicks ok

Table H.10: Use case: Aggregate multiple data points

UC Name	Log in
Actor	User, Authentication server
Trigger	Click on log in button in the header
Pre-condition	The user is not authenticated
Post-condition	The user is authenticated, their username and a sign out button is displayed instead of the log in button.
Main Flow	<ol style="list-style-type: none"> 1. The user is sent to the log in page 2. The user gets a Web token for authentication.
Exception	If the user is not authenticated nothing happens on the website

Table H.11: Use case: Log in

UC Name	Sign out
Actor	User
Trigger	Click on sign out button in the header
Pre-condition	The user is authenticated
Post-condition	The user is not authenticated, the log in button is shown
Main Flow	1. The user is signed out
Exception	

Table H.12: Use case: Sign out

Misuse case

The following are the misuse case diagram developed for the application. The interaction between the use cases and misuse cases are visualised in the figure, and interactions are described in detail in the tables following after the misuse case diagram.

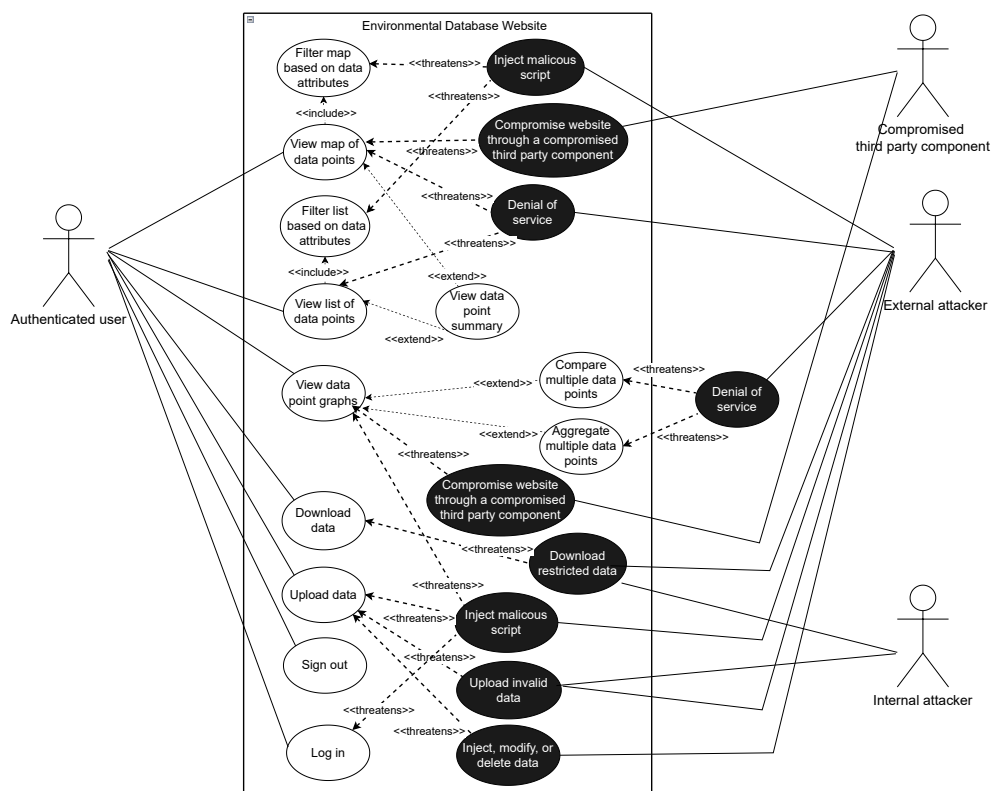


Figure H.2: Misuse case diagram

Name	Compromise website through a compromised third party component
Actor	Compromised third party component
Pre-condition	Component has write access to the website content
Threatens	Viea map of data point and View data points graphs, as these use third party components
Steps	A used component gets compromised without the knowledge of the product owners. The attacker exploits the website access, and insert scripts or content to the website
Post-condition	Website's integrity is compromised, which could lead to the website breaking, clients credentials or other information being stolen, or infection of the clients web browser with malware

Table H.13: Misuse case 1: Compromise website through a compromised third party component

Name	Denial of service
Actor	External Attacker
Pre-condition	User has access to the internal NINA network.
Threatens	View map data points, View list of data points, Compare multiple data points, and Aggregate multiple data points. All of these require large amount of data being requested
Steps	Attacker finds requests which return large amounts of data compared to the request, which have no rate limit Attacker floods the website with these requests, overwhelming the server
Post-condition	Legitimate users are not able to access the website, disrupting the work done by NINA

Table H.14: Misuse case 2: Denial of service

Name	Inject, modify, or delete data
Actor	External attacker
Pre-condition	User has access to the internal NINA network, website has write access to database.
Threatens	Upload data, as it writes to the database
Steps	Attacker find injection points to send commands which will either inject, modify or delete data on the database The attacker then injects, modifies or deletes data using the injection points
Post-condition	NINA may do research with faulty environmental data, Potential loss of environmental data

Table H.15: Misuse case 3: Inject, modify, or delete data

Name	Download restricted data
Actor	External attacker, Internal attacker
Pre-condition	User has access to the internal NINA network.
Threatens	Download data
Steps	Attacker bypasses authentication or exploits a vulnerability to gain access to large amounts of data, or an insider uses their authentication to access the data Attacker downloads the data.
Post-condition	Massive amount of data publicly available, something NINA wants to keep internal for their own research. Competitors can use the data for their own research.

Table H.16: Misuse case 4: Download restricted data

Name	Inject malicious script
Actor	External attacker
Pre-condition	User has access to the internal NINA network.
Threatens	Filter map baes on data attributes, Filter list based on data attributes, View data point graphs, Upload data, as all of these have inputs which can be used for injection
Steps	Attacker identifies a cross site scripting, injection vulnerability, or weak file upload validation in the website Attacker injects a malicious script that is executed in users browser
Post-condition	Can lead to stolen cookies, session hijacking, and spreading of malware.

Table H.17: Misuse case 5: Inject malicious script

Name	Upload invalid data
Actor	External attacker, Internal attacker
Pre-condition	User has access to the internal NINA network.
Threatens	Upload data
Steps	Attacker either exploits vulnerability with authentication of file upload, or is an internal attacker The attacker uploads invalid data to affect NINA's research
Post-condition	Can lead to NINA using research faulty data

Table H.18: Misuse case 6: Upload invalid data

Appendix I

User Test 1

User Test 08.02.24

Planning

Goals with User test

- Understand how different users interact with our initial wireframe. Evaluate if the wireframe is intuitive, easy to use, and efficient when it comes to functionality, design (e.g. use of colors), as well as layout.
- Use this to validate and find weaknesses in our current wireframe, by identifying both user satisfaction and usability issues.
- Iterate on our design further based on the findings.

Methodology

The user test will use the testing method "Usability test", where the user will be asked to complete tasks on the wireframe, and then asked open ended questions related to the task. It will be conducted remotely and in-person in two separate sessions, both times in Norwegian.

Participants

Session 1

Name	Knut Marius Myrvold
Age group	Adult
Gender	Male
Location	Lillehammer
Technology use	Medium
Relation to product	Product owner

Name	Tobias Holter
Age group	Adult
Gender	Male
Location	Lillehammer

Technology use	Medium
Relation to product	Product owner

Session 2

Name	Thomas Selvig
Age group	Young adult
Gender	Male
Location	Gjøvik
Technology use	Frequent
Relation to product	None

Name	Storm Hansen
Age group	Young adult
Gender	Male
Location	Gjøvik
Technology use	Frequent
Relation to product	None

User Test Structure

Descriptions of the website

Dette programmet går ut på å la forskere hos NINA visualisere og jobbe med data de har samlet fra elver i Norge. Dataen beskriver fisk NINA har fanget, og egenskaper ved fiskene, som lengde og kjønn. Det er et kart og en liste som lar deg se de forskjellige punktene med data. Det er to forskjellige typer data punkt: Elvedata, og stasjonsdata. Elvedata er observasjoner ved en elv på en dag, der stasjoner er alle de mindre strekningene med observasjoner som ble gjort i den elven. Man kan også se data for hver stasjon med grafer, samt laste opp og ned data.

Questions

1. Kart siden

- a. Filtre art etter "Ørret" og liste opp alle stasjonsdataene mellom år 2020 og 2024.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Lukk og åpne filter menyen på venstre side.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- c. Velg ett elvedata punkt og legg merke til at det er flere stasjoner som blir listet opp. Åpne stasjon 2 under elvedatapunktet.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- d. Prøv å lese data fra stasjonspunktet som er valgt, finn stasjonsbeskrivelsen.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- e. Lukk stasjonsdata elementet og kom tilbake til startsidene med bare kart.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

2. List siden

- a. Finn frem til liste siden.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Finn samme stasjon som i sted (Gudbrandsdalslågen - stasjon 2).
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- c. Finn elvedata til Gudbrandsdalslågen, registrert 20.09.2023.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

3. Graf siden

- a. Finn frem til graf siden.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Søk etter "alna elven stasjon 1" og velg art-ene Ørret, Harr og Gjedde.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- c. Vis fordeling av arter, for antall fanget av de ulike artene per minutt. Bytt til sektordiagram. Lukk sektordiagrammet slik at den ikke vises lenger.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- d. Vis forelingen av fisk lengden til ørret. Prøv å bytte intervallet til 20 mm. Bytt til å vise fordelingen med boksplokk. Lukk elementet.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- e. Gå tilbake til kartet i startsidene og velg en elvedata som du ønsker å se på og åpne grafen til denne elvedataen.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- f. Velg stasjonene "Alna elven, stasjon 1" og "Gjøvikselven, stasjon 2" og slå sammen dataene for de to stasjonene og finn lengdefordelingen av Ørret og Harr med boksplokk. Sammenlikn dataene fra begge stasjonene som er valgt.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

4. Laste opp filer

- a. Prøv å trykke på “last opp” knappen uten at det er noen filer lagt til. Prøv å legge til filer.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Prøv å legge til tre riktige filer.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

5. Log in

- a. Prøv å logge inn i systemet.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Observations and feedback - Session 1

Marius

Kart:

1a.

Filtrere art etter “Ørret” og liste opp alle stasjonsdataene mellom år 2020 og 2024. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Ser bra ut, fint å få med hele Norden, mest punkter i Norge, noen i Sverige, fint å få med Nord-Norge.

Når Knut Marius fikk i oppgave om å finne stasjonsdata punktet mellom 2020 og 2024, identifiserte han umiddelbart filtrerings boksene for “Art” og “År” som han benyttet til å finne riktig stasjon. Han syntes at denne prosessen virket intuitiv og effektiv. I tillegg nevnte han at utseende så elegant og brukervennlig ut. Det ble diskutert mellom Knut Marius og Tobias om vi skal ha dato framfor år i venstre marg. I tillegg var de også usikre på om de ville ha en nedlastingsknapp nede i venstre marg hvor man allerede her kan velge å laste ned dataen. Disse to punktene skulle de komme tilbake til ved neste møte. Til slutt diskuterte vi også om vi ville ha en scrollable liste på venstre marg, enten for hele filteret eller kun for art. Vi ble også enige om å type data, år og så art til slutt, siden art var lenger.

1b.

Lukk og åpne filter menyen på venstre side.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Knut Marius hadde ingen problemer med å identifisere hvordan man skulle skyve vekk filteret og åpne det igjen. Han mente ikon bruket vårt var kjent og lett å forstå. Han spesifiserte at de ønsket at valgene i filteret i venstre marg skulle bli lagret om man skjulte og deretter åpnet margen igjen. I tillegg ville han innenfor "Art" filteret ha en mulighet til å velge alle arter og ikke bare en og en. Vi ble også enige om å sortere artene etter alfabetisk rekkefølge, men at "Alle" skulle ligge øverst som en egen mulighet.

1c.

Velg ett elvedata punkt og legg merke til at det er flere stasjoner som blir listet opp.

Åpne stasjon 2 under elvedatapunktet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Knut Marius syntes det var intuitivt å se alle tilhørende stasjonsdata fra et elvedata punkt. Han likte også ideen om å inndele stasjonsdata med en farge, og elevdata med en annen. I tillegg ønsket han også at den markerte elvedata eller stasjonspunktet ble markert med en tredje farge og gjerne med store kontraster (se CWAG 2.0). Her nevnte han Orasnje, Svart og Blå som eksempler. Han var også enig om at man ikke burde se strekene elve datapunktet før man zoomet inn.

1d.

Prøv å lese data fra stasjonspunktet som er valgt, finn stasjonsbeskrivelsen.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Var intuitivt og lett å finne fram til de ulike stasjonene og deres beskrivelse. Han sa også at det så ryddig og oversiktlig ut for et raskt innblikk. Derimot diskuterte vi i etterkant om vi ville ha noe mer her, dette var de litt usikre på, fordi da blir det fort også uryddig. Her vurderte vi scrolling funksjonalitet, og alternativene var å ha hele boksen med scrolling funksjon eller kunne stasjonene. Her ble det også nevnt muligheten om å fjerne og skjule beskrivelsen med en pil feks. Han synes det også var helt kurant at stasjonsdata blir huket av når man trykker på en stasjon i elvedata.

1e.

Lukk stasjonsdata elementet og kom tilbake til startsidene med bare kart.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Det var intuitivt å lukke stasjonspunktet og komme tilbake til kartet, ikonene var som sagt kjente og det gjorde det veldig brukervennlig.

List siden:

2a.

Finn frem til liste siden.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å finne fram og bruke funksjoner. Han ønsket at filtrene skulle kunne endre på lista på siden. Ellers synes han det var fint å ha med en søkefunksjon på id eller elvenavn.

2b.

Finn samme stasjon som i sted (Gudbrandsdalslågen - stasjon 2).

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Dette var intuitivt likt, særlig at teksten ble hevet ut når man trykket på stasjonen. Oversikten over dataen blir det samme som i stad, tydelig og oversiktlig.

2c.

Finn elvedata til Gudbrandsdalslågen, registrert 20.09.2023.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Hele prosessen var intuitiv og han likte godt at det var flere måter og gjøre det på, enten søke på Elven, eller gå tilbake til startsidene og filtrere på år og elvedata. Det å ha flere veier til rom, kan være fint når man får nye ansatte eller brukere.

0.

Hva synes du om list-siden, er det noe du føler at den mangler eller noe du ønsker å legge til for å få brukeren til å føle seg mer velkommen.

Her kom han med lite ettersom han syntes det så veldig bra ut, men det ble diskutert om at listen burde ha en header, og også om det kunne søkes etter disse dataene i etterkant.

Graf siden:

3a.

Finn frem til graf siden.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Intuitivt inntrykk vanskelig å si ved første user test, men fint at det er interaktiv prosess. Han påpekte at det var fint å ha med 5mm som standard lengde i filteret.

3b.

Søk etter "alna elven stasjon 1" og velg art-ene Ørret, Harr og Gjedde.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Intuitivt å navigere seg frem og tilbake. Derimot var usikker på om han skulle krysse av for ingen eller alle, derfor foreslo han "Alle" knappen nevnt tidligere.

3c.

Vis fordeling av arter, for antall fanget av de ulike artene per minutt. Bytt til sektordiagram. Lukk sektordiagrammet slik at den ikke vises lenger.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Når det gjaldt diagrammene syntes han brukervennligheten var god og oversiktlig. Derimot hadde han en innvending vedrørende diagrammene. Selvom linjediagram viser det samme som søylediagram (frekvens fordeling), vil det ofte bli enklere og mer oversiktlig med et søylediagram i deres foretning så det ble vi enige om å legge til, men han ville også beholde linjediagrammet fordi det var fint når vi fint å se når vi sammenliknet dataen. Her var han også påpasselig med å minne oss på fargebruk.

3d.

Vis forelingen av fisk lengden til ørret. Prøv å bytte intervallet til 20 mm. Bytt til å vise fordelingen med boksplokk. Lukk elementet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Han nevnte at boksplokk så bra ut og at vi hadde tatt med streken inne i boksen som representerer medianverdien.

3e.

Gå tilbake til kartet i startsiden og velg en elvedata som du ønsker å se på og åpne grafen til denne elvedataen.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Her syntes han det var lett å legge til en stasjon til, det blir det samme som tidligere. Han nevnte at han likte at vi skilte mellom aggregere og sammenlign øverst i venstre marg.

Vedrørende kakediagram, legg til en opsjon som lar deg velge om man vil inkludere en kategori som heter "annet" når man har valgt spesifikke fisker. Denne kategorien 'annet' vil da representere alle arter som ikke er valgt.

3f.

Velg stasjonene "Alna elven, stasjon 1" og "Gjøvikselven, stasjon 2" og slå sammen dataene for de to stasjonene og finn lengdefordelingen av Ørret og Harr med boksplokk. Sammenlikn dataene fra begge stasjonene som er valgt.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Intuitivt å bytte fra aggregere data til å sammenlikne data. Ingen innvendinger her.

0.

Hva synes du om grafsiden(e), er det noe du føler at den mangler eller noe du ønsker å legge til for å få brukeren til å føle seg mer velkommen.

Mulighet til å laste ned data mens man ser på punktene, mulighet til å laste ned sammenligninger og aggregerte data.

Last opp filer:

4a.

Prøv å trykke på “last opp” knappen uten at det er noen filer lagt til. Prøv å legge til filer.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Intuitivt og logge inn og trykke på knappen for å laste opp filer, og han så de tillatte formatene med en gang, .csv og .xls. Sett ned grensen for opplasting siden filene ikke er så store, (10mb per fil). Fint å kunne laste opp flere ganger, intuitivt og kjent grensesnitt. Legg til .xlsx også som et filformat siden det er nye formatet til excel.

4b.

Prøv å legge til tre riktige filer.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Ikoner og kontraster er veldig tydelig og gode.

0.

Hva synes du om fil opplastingssiden, er det noe du føler at den mangler eller noe du ønsker å legge til for å få brukeren til å føle seg mer velkommen.

Opplastingssiden var tydelig og god, og han likte oversikten man fikk når man la til filer i bufferen, spesielt det med at man kunne fjerne uønskede filer og klare feilmeldinger hvis man prøvde å laste opp ingenting.

Log in:

5a.

Prøv å logge inn i systemet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Intuitivt og lett å forstå, veldig kjent og standard.

0.

Hva synes du om innloggingssiden, er det noe du føler at den mangler eller noe du ønsker å legge til for å få brukeren til å føle seg mer velkommen.

Innlogging funksjonalitet virket veldig kjent og han likte at man fikk instruktive feilmeldinger når man tastet inn feil brukernavn eller passord.

Score:

Ellers, er det noe annet du ønsker å si om programmet, noen forbedringer som kan bli gjort til funksjonene som kan forbedre brukeropplevelsen? Hva var bra?

Soleklart 10 😊

Tobias

Kart:

1a.

Filtrere art etter “Ørret” og liste opp alle stasjonsdataene mellom år 2020 og 2024. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å bruke filteret. Når det gjelder filteret på startsidene nevnte han at NINA ser på ca. 20 arter. Dette betyr at en checkliste for de ulike artene kan bli veldig lang, og ta opp mye av skjermplassen. Kanskje valg av arter være et annet format, eller så burde det flyttes under de andre filtrene.

1b.

Lukk og åpne filter menyen på venstre side.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Enkelt og lett

1c.

Velg ett elvedata punkt og legg merke til at det er flere stasjoner som blir listet opp. Åpne stasjon 2 under elvedatapunktet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å gjøre. Når man velger elvedata, så kan det være greit at antall stasjoner står også. Denne informasjonen vil stå sammen med vannføring og kommentar . Ble foreslått at all

informasjonen kan komprimeres som default, der det bare står en linje med tekst: "Info om elv". Trykker man på denne vil man ende opp med dataen slik som vist i wireframen.

Elvedata kan inneholde mange stasjoner (opptil 100, eller kanskje enda flere). Diskuterte derfor ulike måter å scrolle listen på, f.eks. scrolle hele venstre-siden eller bare den lille listen nederst. De foretrakk å scrolle hele venstre-siden. Denne diskusjonen er en alternativ løsning i forhold til muligheten til å komprimere informasjons-feltene.

1d.

Prøv å lese data fra stasjonspunktet som er valgt, finn stasjonsbeskrivelsen.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Når man ser på "fisk data" for en stasjon, så vil de gjerne også se antall fisk funnet i forhold til tid (f.eks. fisk / minutt). Vil også ha en linje under dataen om de ulike fiskene, som er total. Den vil si summen på antall for alle fiskene og lignende for de andre kolonnene med data (min lengde vil være den minste lengden oppdaget av alle de ulike fiskeartene).

Ville vært gunstig når man velger "last ned" når man er på stasjonsdata, at man også får mulighet til velge å laste ned andre stasjoner i samme elv. Lignende funksjonalitet vil de gjerne ha når man trykker "last ned" på elvedata, altså at man da får opp en liste med alle stasjonene og så huker man av de man vil laste ned data om. Hvis det var mulig, ville de også ha mulighet til å laste ned stasjoner fra ulike elver samtidig, f.eks. stasjon 1 fra to ulike elver. De vil også ha muligheten til å velge å laste ned stasjonsdata for den samme stasjonen over flere år. Altså spesifisere hvilke år man laster ned data om.

Når man får opp stasjonsdata siden, ville de gjerne ha mulighet til å bytte år på den siden. Dette gjør det lett å bytte fra en stasjon i ulike år. Slik det er nå, må man gå tilbake til kartet og velge punktet som representerer stasjonen for et annet år. Å legge til dette vil gjøre nettsiden mer fleksibel. Folk foretrekker ulike ting, så å ha flere måter å oppnå det samme gjør brukeropplevelsen bedre.

1e.

Lukk stasjonsdata elementet og kom tilbake til startsidene med bare kart.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Enkelt og intuitivt

Likte at det var linjer som separerte de ulike delene for elvedata og stasjonsdata. Det gjorde at det var lettere å lese, men også at det var lettere å finne informasjonen man lette etter.

De vil gjerne ha mulighet for ulike kart, f.eks. satellitt, topologisk,

List siden:

2a.

Finn frem til liste siden.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å finne fram, likte symbolet for liste. Nevnte at å filtrere på år kanskje ikke alltid vil være spesifikt nok. Vil derfor heller ha mulighet til å filtrere på datoer. Hvis man allikavel ender opp med å bruke år, vil de ha en drop down menu som inneholde alle relevante år, istedenfor å manuelt skrive inn årene.

2b.

Finn samme stasjon som i sted (Gudbrandsdalslågen - stasjon 2).

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å finne. Likte at den ble markert når man valgte den.

2c.

Finn elvedata til Gudbrandsdalslågen, registrert 20.09.2023.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å finne og bruke filter.

0.

Hva synes du om list-siden, er det noe du føler at den mangler eller noe du ønsker å legge til for å få brukeren til å føle seg mer velkommen.

Det ble nevnt at de kanskje var interessert i flere filtreringsmuligheter, f.eks. klokkeslett. Det gjelder spesielt for når ting er vist som liste, da start og stopp klokkeslett er kolonner som vises. Ønsker at søkefunksjonalitet i liste ikke starter på starten av ordet, men ser etter om søkemønsteret finnes i ordet. F.eks. søker man på "omma", så vil glomma vises siden glomma inneholder søkemønsteret. Andre metoden for søking går ut på at man f.eks. skriver "g", og så vises bare elvene som starter med bokstaven g. Altså hvis en elv inneholde g, men ikke starter på denne bokstaven, så vil elven ikke vises.

For listevising av data, vil de gjerne ha en header over dataen. Denne inneholder kolonnene titler som da sier hva kolonnene inneholder. Denne headeren vil også ha funksjonalitet som gjør at hvis man trykker på et av elementene, så sorteres listen utifra dette. For eksempel trykker man på "elvenavn" så sorteres listen alfabetisk etter navnene, men trykker man på "prosjekt-id", så sorteres numerisk etter id-en.

Graf siden:

3a.

Finn frem til graf siden.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å finne fram til graf siden, bra bruk av symbol her også.

3b.

Søk etter “alna elven stasjon 1” og velg art-ene Ørret, Harr og Gjedde.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Greit å bruke filteret, men har man samme problemstilling som med filteret på startsiden, altså at den kan bli mange arter. De vil at vi bruker litt tid på å tenke på dette, og finne beste løsning for menyen, men også framstilling av data når mange arter er involvert.

3c.

Vis fordeling av arter, for antall fanget av de ulike artene per minutt. Bytt til sektordiagram. Lukk sektordiagrammet slik at den ikke vises lenger.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Fint å bruke meny til å få fram diagram, men Wireframen mangler tydelige aksetitler på grafene, noe de påpekte som veldig viktig at var med for det ferdige produktet.

3d.

Vis forelingen av fisk lengden til ørret. Prøv å bytte intervallet til 20 mm. Bytt til å vise fordelingen med boksploott. Lukk elementet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Greit å få opp diagramet og bytte til 20mm. Synes linjediagram ble litt uoversiktlig bare for en art. Da ville de heller bruke søylediagram med intervaller, f.eks. 10-15 cm, 16-20cm. Kan hende at intervallene er ulike for ulike arter, men dette er ikke helt avklart. Ble heller ikke helt avklart om de foretrakk cm eller mm, men de sa at vi burde bruke mm inntil videre da dette er det de bruker for dataene sine. Linjediagram var veldig nyttig når flere arter skulle vises samtidig.

De var ikke veldig godt kjent med alle typer grafer, men spesielt box-plot syntes de var litt forvirrende. De likte ideen bak box-plot når det ble forklart, da den gå informasjonen på en god og unik måte. Det var ønskelig om grafene hadde et slags spørsmålsteget i et av hjørnene. Trykker man på dette, vil man få informasjon om typen graf som er brukt. Her har man også kanskje mulighet til å endre på spesifikasjonene for grafene, f.eks. ranges for søylediagram eller fordeling på box-plot.

3e.

Gå tilbake til kartet i startsiden og velg en elvedata som du ønsker å se på og åpne grafen til denne elvedataen.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å finne fram, var veldig klart hva som ble valgt man åpnet en stasjon i et nytt vindu. Nå finnes muligheten til å trykke vis graf når man ser på elvedata. De vil gjerne da ha at det dukker opp en slags liste over stasjonene, og så velger man de man vil skal være med over når man ser på grafene.

3f.

Velg stasjonene “Alna elven, stasjon 1” og “Gjøvikselven, stasjon 2” og slå sammen dataene for de to stasjonene og finn lengdefordelingen av Ørret og Harr med boksplott. Sammenlikn dataene fra begge stasjonene som er valgt.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

For graf-delen ville vi gjerne sette en maks grense for antall stasjoner, da visning av f.eks. 1000 stasjoner gjør ting uoversiktlig og tregt. Vi diskuterte oss fram til en midlertidig grense på 50 stasjoner, som betyr at man kan slå sammen informasjon fra maks 50 ulike stasjoner for grafene.

Last opp filer:

4a.

Prøv å trykke på “last opp” knappen uten at det er noen filer lagt til. Prøv å legge til filer.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Bra tilbakemelding, enkelt å forstå.

4b.

Prøv å legge til tre riktige filer.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å gjøre, intuitiv løsning. Likte symbolene. Ble nevnt at opplasting av filer må støtte den nyeste versjonen av microsoft xl.

Log in:

5a.

Prøv å logge inn i systemet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Veldig lett, ingenting å forberede.

Andre notater etter utført user test:

For listevising av data, så vil det trengs en kolonnene til. Denne kolonnen er enda ikke en del av NINA sin backend, som er grunnen til at den ikke ble implementert som en del av user-testen. Den nye kolonnen inneholder stop-data, som er der fordi noen ganger fisker de over flere dager.

Kanskje kommer noe som heter elvenummer-id, men dette er ikke avklart enda.

(kan bli flere linjer for samme elv og stasjon, hvis de fisker på flere dager. Er noe de ikke har avklart enda)

Results

The user test gave valuable insight into what was done well and what could be improved. There were also several features mentioned which will be implemented for the next iteration of the wireframe.

What was good:

- Interacting with the map page and filtering points was intuitive.
- Use of symbols was good, helped with understanding functionality of buttons.
- Navigation between pages and site structure was easy to understand and intuitive.
 - Multiple paths between pages.
- Easy to use search/list page, good to be able to search on project ID or name.
- Good feedback when selecting items on the different pages, such as highlighting selected elements.
- Good to have default values in input fields.
- Easy to interact with and read diagrams.
- Good and informative feedback when making mistakes or when errors happen.

Areas for improvement:

- Use contrast and color blind friendly options for marking points and highlighting important elements.
- Filter points by date instead of year.
- Use Bar diagram instead of line diagram.
- Change upload page file size limit
- Important that diagrams have descriptions for the axis and data.
- Implement a max limit for amount of stations you can select in the graph page.

Recommended New features:

- Make species a scrollable list, with search function instead of box select, and a option for select all.
- Scrolling in the summary page, to allow for more information. Possibly having collapsible elements .

- Add an option for adding a category called “others” when you have selected specific species. The “other” category will then represent all the species not selected.
- Should be able to download data from the graph page.
- Show summary of station information for rivers, such as amount of stations, sum of fish collected, fish per minute.
- Change what information is shown when selecting a station to include fish per minute
- Add a header to the stations listed in the list page, which you can click on to change the sorting of stations.
- Have multiple options for the map, such as topology as satellite
- When downloading data a pop up window or new window should be opened, where the user can see what data will be downloaded, including stations and attributes selected, and modify this.

Observations and feedback - Session 2

Thomas

1. Kart siden

- a. **Filtrere art etter “Ørret” og liste opp alle stasjonsdataene mellom år 2020 og 2024.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
 Forrivrende at kartsiden er hjemsiden. Er det logo deres?
- b. **Lukk og åpne filter menyen på venstre side.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
 Intuitivt
- c. **Velg ett elvedata punkt og legg merke til at det er flere stasjoner som blir listet opp. Åpne stasjon 2 under elvedatapunktet.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
 Gir mening. Er usikker på om summary overlayer kartet.
- d. **Prøv å lese data fra stasjonspunktet som er valgt, finn stasjonsbeskrivelsen.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
 Mulig å ha data og filter på samme side?
- e. **Lukk stasjonsdata elementet og kom tilbake til startside med bare kart.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
 Intuitivt

2. List siden

- a. **Finn frem til liste siden.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
 Ryddig header.
- b. **Finn samme stasjon som i sted (Gudbrandsdalslågen - stasjon 2).**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Forrivrrende at art enda er et valg. Burde være velg alle som er valgt som default. Burde være mulig å søke på gul 1 for å finne den.

Liker ikke overlay. Burde kanskje vise stasjonsdata i stedet for resultater.

Alle ikon, som søke symbolet, burde kanskje være mulig å trykke på.

c. Finn elvedata til Gudbrandsdalslågen, registrert 20.09.2023.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Ga mening. Mye informasjon på en gang, filter + resultat + stasjonsdata.

3. Graf siden

a. Finn frem til graf siden.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Søkebar øvest i stedet for i siden. Eller ser bra ut. Lett å komme seg fram.

b. Søk etter “alna elven stasjon 1” og velg art-ene Ørret, Harr og Gjedde.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Gir mening.

c. Vis fordeling av arter, for antall fanget av de ulike artene per minutt. Bytt til sektordiagram. Lukk sektordiagrammet slik at den ikke vises lenger.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Forrivrrende at “vis” button. Kunne kanskje hatt mulighet til å adde

“diagramer”, og så velge instillinger for den.

Clean data view. Liker hvordan man velger opsjoner, kan velge mellom to

valg.

d. Vis forelingen av fisk lengden til ørret. Prøv å bytte intervallet til 20 mm. Bytt til å vise fordelingen med boksploott. Lukk elementet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Greit å bruke instillinger. Kunne vært ide å ha tabs å ha flere grafer samtidig

e. Gå tilbake til kartet i startsidene og velg en elvedata som du ønsker å se på og åpne grafen til denne elvedataen.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

God måte å vise stasjoner valgt på. Lett å finne fram.

f. Velg stasjonene “Alna elven, stasjon 1” og “Gjøvikselven, stasjon 2” og slå sammen dataene for de to stasjonene og finn lengdefordelingen av Ørret og Harr med boksploott. Sammenlikn dataene fra begge stasjonene som er valgt.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Gir mening å aggregere og sammenligne data. Liker valg av meny. Dataen ser bra ut, den er lett å lese og representerer det han er interessert i.

4. Laste opp filer

a. Prøv å trykke på “last opp” knappen uten at det er noen filer lagt til.

Prøv å legge til filer.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Forrivrrende at man kan se siden uten å logge inn. Bra ikon og knapper.

b. Prøv å legge til tre riktige filer.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Bra med feedback avhengig av resultat last opp. Dumt at man ikke kan se hva man tidligere har lastet opp. Utenom dette er det intuitivt.

Gi teknisk tilbakemelding når noe ikke går galt.

5. Log in

a. Prøv å logge inn i systemet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å logge inn, så bra ut.

Ekstra kommentarer

Ryddig struktur og navigasjon.

Føles ut som at man må "kunne" systemet før man bruker det, på grunn av mye informasjon om ganger som gjør at man må filtrere ut det som faktisk er viktig.

Burde ha færre bokser, mindre borders og heller bruke avstand.

Storm

1. Kart siden

a. Filtrere art etter "Ørret" og liste opp alle stasjonsdataene mellom år 2020 og 2024.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Intuitivt.

b. Lukk og åpne filter menyen på venstre side.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Vurdere å flytte knapp for å åpne filtrering, utenom det var det intuitivt.

c. Velg ett elvedata punkt og legg merke til at det er flere stasjoner som blir listet opp. Åpne stasjon 2 under elvedatapunktet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Greit å finne et punkt. Kunne klarer skilt stasjonene, og kanskje hatt header fordi var litt blandet inn i den andre teksten.

d. Prøv å lese data fra stasjonspunktet som er valgt, finn stasjonsbeskrivelsen.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Tydlig at man kan komme seg til stasjondata fra elv. Burde kanskje kunne trykke på hele stasjonen for å komme til den fra elven. Indikere at man kan trykke på en stasjon ved for eksempel endre fargen rundt til blå.

Siden med stasjondata ser bra ut, er mye data men er oversiktelig.

ha mulighet til å slide stasjonsdata siden for å endre størrelse.

Knappene er lette å forstå hva gjør.

e. Lukk stasjonsdata elementet og kom tilbake til startside med bare kart.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å gjøre, fint symbol.

2. List siden

a. Finn frem til liste siden.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å gjøre, liste symbolet og navnet gjorde det klart, navigasjon ga mening.

b. Finn samme stasjon som i sted (Gudbrandsdalslågen - stasjon 2).

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Fint å åpne stasjonen, lett å søke.

Veldig mye data kom opp. Redd for å trykke feil sted og miste dataen.

Kunne hatt drop down meny, eller at det åpnet i en egen link. Da burde man ha også mulighet til å åpne link fra kartet.

c. Finn elvedata til Gudbrandsdalslågen, registrert 20.09.2023.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Greit å forstå. Bra at kan enda kan flytte til stasjonen.

3. Graf siden

a. Finn frem til graf siden.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å finne fram. Veldig boksete design, kunne kombinert flere av menyene.

Bakgrunnsfargen skaper for mye rom mellom elementer. For stor kontrast.

b. Søk etter "alna elven stasjon 1" og velg art-ene Ørret, Harr og Gjedde.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å addere stasjoner. Bra at man får vite at man ikke har lagt til stasjon.

Burde ha pluss knapp når man skal legge til stasjon, sånn at man ikke blir redd for at stasjonen man har valgt blir borte.

c. Vis fordeling av arter, for antall fanget av de ulike artene per minutt. Bytt

til sektordiagram. Lukk sektordiagrammet slik at den ikke vises lenger.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Knottete å først måtte trykke på arter, så vis. Vis er synlig nok til å forstå at man måtte bruke den med en gang. Også burde være på med default.

d. Vis forelingen av fisk lengden til ørret. Prøv å bytte intervallet til 20 mm.

Bytt til å vise fordelingen med boksploott. Lukk elementet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Sier igjen at vis boksene burde være default på, sånn at de endrer seg når man endrer data man søker med. Greit å bytte intervall.

e. Gå tilbake til kartet i startsidene og velg en elvedata som du ønsker å se på og åpne grafen til denne elvedataen.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Naturlig å aggregere data fra en elv. Intuitivt når man flytter seg fra kart til graf.

- f. **Velg stasjonene “Alna elven, stasjon 1” og “Gjøvikselven, stasjon 2” og slå sammen dataene for de to stasjonene og finn lengdefordelingen av Ørret og Harr med boksplott. Sammenlikn dataene fra begge stasjonene som er valgt.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Greit å sammenligne stasjoner. Diagrammene gir mening.

4. Laste opp filer

- a. **Prøv å trykke på “last opp” knappen uten at det er noen filer lagt til.**

Prøv å legge til filer.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Burde bli redirecta til logg inn når man trykker på last opp data.

Burde ikke være bokser, lik bagrunn over alt. Bruke heller avstand mellom boksene.

Utenom det var det bra bruk av symboler, og det er lett å se hva man skal gjøre.

- b. **Prøv å legge til tre riktige filer.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Burde ha rename funksjonalitet. Utenom det ser det bra ut, lett å bruke.

Redd for å miste dataen man lastet opp når man må logge inn etter dataen allerede r lastet opp.

User feedback er nice.

5. Log in

- a. **Prøv å logge inn i systemet.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å bruke.

Results

What was good:

- Interacting with the map, points and the filter menu was easy and intuitive.
- Use of symbols was good, helped with understanding functionality of buttons.

- Navigation between pages and site structure was easy to understand and intuitive. Liked the header design.
- Good feedback when selecting items on the different pages, such as highlighting selected elements.
- Good to have default values in input fields.
- Good and informative feedback when making mistakes or when errors happen.
- Good use of radio buttons

Areas for improvement:

- Don't overwhelm the user with information. For example, open a new window or dropdown with information about a station on the list page instead of overlaying the search results and filters.
- Could have headers to more clearly section different information on the page
- Could have made the button to move from a river to station point bigger and easier to click.
- Could have the select station element on the graph page stand out more.
- The user interface when selecting diagrams to show. The "show" button is not highlighted enough.
- Use less borders and more space to signify different parts of the page
- Could change how you add stations to signify that you are adding more station instead of replacing the one selected

Recommended New features:

- Should have "select all species" button
- Possibly rework the way you display diagrams on the graph page. Instead of having a "show" button, could have a "add new diagram", where you can then choose the settings.
- Should not let the user enter the upload page without being logged in.
- Would be good to see upload history for a user.
- Could have a slider which lets you resize the summary section on the map page.
- The user should be able to rename a file when uploading it

Summary and prioritized improvements

The navigation, interaction with different elements, and the site structure got positive feedback. The use of symbols was also good and made the product easier to use. All users also liked the feedback you got from popup windows and text which helped navigate and explain problems and states to users.

There were several areas for improvement, as well as features which would be nice to have. Here is a list of improvements and new features categorized by importance:

High importance

- Change color use and design on some parts of website

- Use contrast and color blind friendly options for marking points and highlighting important elements.
- Don't overwhelm the user with information. For example, open a new window or dropdown with information about a station on the list page instead of overlaying the search results and filters.
- Could have headers to more clearly section different information on the page
- The "show" button is not highlighted enough on the graph page
- Restructure of summary page. Scrolling in the summary page, to allow for more information. Possibly having collapsible elements.
- Use less borders and more space to signify different parts of the page
- Should not let the user enter the upload page without being logged in.
- Improve diagrams
 - Important that diagrams have descriptions for the axis and data. Use Bar diagram instead of line diagram.
- Improve input fields
 - Make species a scrollable list, with search function instead of box select, and an option for select all.
 - Add a header to the stations listed in the list page, which you can click on to change the sorting of stations.
- Create a separate download page
 - When downloading data a pop up window or new window should be opened, where the user can see what data will be downloaded, including stations and attributes selected, and modify this.
- Add new information to the pages
 - Show summary of station information for rivers, such as amount of stations, sum of fish collected, fish per minute.
 - Change what information is shown when selecting a station to include fish per minute

Medium importance

- Filter points by date instead of year.
- Implement a max limit for amount of stations you can select in the graph page.
- Change upload page file size limit
- Should be able to download data from the graph page.
- Could make the button to move from a river to station point bigger and easier to click.
- Could have the select station element on the graph page stand out more.
- Could change how you add stations to signify that you are adding more station instead of replacing the one selected

Low importance

- Add an option for adding a category called "others" when you have selected specific species. The "other" category will then represent all the species not selected.
- Have multiple options for the map, such as topology as satellite
- Would be good to see upload history for a user.

- Could have a slider which lets you resize the summary section on the map page.
- The user should be able to rename a file when uploading it

Signature:

Tobias Holter



Lillehammer 08.02.2024

Knut Marius Myrvold



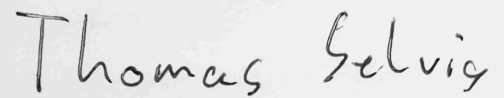
Lillehammer 08.02.2024

Storm Hansæl



Gjøvik 08.02.2024

Thomas Selvig



Gjøvik 08.02.2024

Appendix J

User Test 2

User Test 14.02.24

Planning

Goals with User test

- Understand how different users interact with our wireframe. Evaluate if the wireframe is intuitive, easy to use, and efficient when it comes to functionality, design (e.g. use of colors), as well as layout.
- Use this to validate and find weaknesses in our current wireframe, by identifying both user satisfaction and usability issues.
- Iterate on our design further based on the findings.

Methodology

The user test will use the testing method “Usability test”, where the user will be asked to complete tasks on the wireframe, and then asked open ended questions related to the task. It will be conducted remotely and in-person in two separate sessions, both times in Norwegian.

Participants

Session 1

Name	Knut Marius Myrvold
Age group	Adult
Gender	Male
Location	Lillehammer
Technology use	Medium
Relation to product	Product owner

Name	Tobias Holter
Age group	Adult
Gender	Male
Location	Lillehammer

Technology use	Medium
Relation to product	Product owner

User Test Structure

Descriptions of the website

Dette programmet går ut på å la forskere hos NINA visualisere og jobbe med data de har samlet fra elver i Norge. Dataen beskriver fisk NINA har fanget, og egenskaper ved fiskene, som lengde og kjønn. Det er et kart og en liste som lar deg se de forskjellige punktene med data. Det er to forskjellige typer data punkt: Elvedata, og stasjonsdata. Elvedata er observasjoner ved en elv på en dag, der stasjoner er alle de mindre strekningene med observasjoner som ble gjort i den elven. Man kan også se data for hver stasjon med grafer, samt laste opp og ned data.

Questions

1. Kart siden

- a. Filtrere art etter "Ørret" og "Harr" og liste opp alle elvedataene mellom to selvvalgte datoer.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Filtrer etter alle arter. Velg ett elvedata punkt.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- c. Åpne stasjon 2 under elvedatapunktet. Prøv å lese data fra stasjonspunktet som er valgt, finn sum av all fiskedata.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- d. Lukk stasjonsdata elementet og kom tilbake til startsidene med bare kart.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

2. List siden

- a. Finn frem til liste siden. Se på siden. Prøv å sortere resultat etter stasjonsnavn.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Lat som at det er veldig mange stasjoner vist. Bruk søk og filter til å finne samme stasjon som i sted (Gudbrandsdalslågen - stasjon 2). Se på dataen til stasjonen.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- c. Gå tilbake til listesiden. Finn elvedata til Gudbrandsdalslågen, registrert 20.09.2023. Se på dataen til elven.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

3. Graf siden

- a. Finn frem til graf siden. Se på siden.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Du skal se på en stasjon som heter "alna elven stasjon 1". Du er interessert i artene Ørret og Harr, så velg disse. Du vil også se alle andre arter som en egen kategori. Se på diagrammene som kommer opp.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- c. Bytt til sektordiagram for fordeling av arter. Lukk sektordiagrammet slik at den ikke vises lenger. Åpne boksplokk for fordeling av lengde, og bytt intervall til 20mm.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- d. Prøv å åpne elven alnaelven uten å forlate siden. Prøv å sammenligne den med elven "Gjøvikselven".
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- e. Gå tilbake til kartet i startsidene og velg en elvedata (alna elven) som du ønsker å se på og åpne grafen til denne elvedataen. Du har lyst til å se på stasjonsdataen som ligger under denne elven, og sammenligne disse stasjonene.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

4. Laste opp filer

- a. Prøv å trykk på "last opp" knappen i headeren uten at du er logget inn.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Logg inn og se på siden. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

- c. Trykk på “last opp” knappen uten at det det har blitt lagt til noen filer. Prøv så å legge til tre riktige filer, og så laste dem opp. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

5. Laste ned filer

- a. Velg datatypen “elvedata”, legg til elvene du vil laste ned dataen til. Last ned data for alle artene i elvene som er valgt. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Velg datatypen “stasjonsdata”, legg til stasjonene du vil laste ned dataen til. Velg hvilke arter som skal være med i disse datasettene. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Observations and feedback - Session 1

Marius

3. Kart siden

- a. Filtrere art etter “Ørret” og “Harr” og liste opp alle elvedataene mellom to selvvalgte datoer. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Veldig lett, greit å bruke meny. Bra visuell feedback.

- b. Filtrer etter alle arter. Velg ett elvedata punkt. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Veldig lett.

- c. Åpne stasjon 2 under elvedatapunktet. Prøv å lese data fra stasjonspunktet som er valgt, finn sum av all fiskedata. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Data er fint strukturert og lett å lese. Kunne ha lagt til gjennomsnitt og kommentar til fiske.

- d. Lukk stasjonsdata elementet og kom tilbake til startsidene med bare kart. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Wireframe manglet kryss for å komme seg ut av stasjonsdata meny.

4. List siden

- d. Finn frem til liste siden. Se på siden. Prøv å sortere resultat etter stasjonsnavn.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å navigere. Siden er oversiktelig.

- e. Lat som at det er veldig mange stasjoner vist. Bruk søk og filter til å finne samme stasjon som i sted (Gudbrandsdalslågen - stasjon 2). Se på dataen til stasjonen.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å bruke filter. Gir mening at man kan bruke filter og søk samtidig. Liker at søk matcher med hva som helst som ligner på det du skriver undervies.

- f. Gå tilbake til listesiden. Finn elvedata til Gudbrandsdalslågen, registrert 20.09.2023. Se på dataen til elven.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å bytte og se elvedata.

3. Graf siden

- f. Finn frem til graf siden. Se på siden.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å navigere. Mye informasjon på siden, men er bra strukturert og det er lett å forstå hva de forskjellige elementene gjør.

- g. Du skal se på en stasjon som heter "alna elven stasjon 1". Du er interessert i artene Ørret og Harr, så velg disse. Du vil også se alle andre arter som en egen kategori. Se på diagrammene som kommer opp.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å bruke, menyen er intuitiv. Kunne kanskje vurdert å bytte navn til "se andre arter" knappen.

- h. Bytt til sektordiagram for fordeling av arter. Lukk sektordiagrammet slik at den ikke vises lenger. Åpne boksplokk for fordeling av lengde, og bytt intervall til 20mm.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å bytte samt å velge intervall.

- i. Prøv å åpne elven alnaelven uten å forlate siden. Prøv å sammenligne den med elven "Gjøvikselven".
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Intuitivt å bytte mellom elv og stasjon. Lett å sammenligne elver.

- j. Gå tilbake til kartet i startsidene og velg en elvedata (alna elven) som du ønsker å se på og åpne grafen til denne elvedataen. Du har lyst til å se på stasjonsdataen som ligger under denne elven, og sammenligne disse stasjonene.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å gjøre, bra bruk av symboler på knapper.

4. Last opp filer

- d. Prøv å trykk på "last opp" knappen i headeren uten at du er logget inn. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett og greit.

- e. Logg inn og se på siden. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Siden ser enkel og intuitiv ut. Bruk av kjente knapper og symboler.

- f. Trykk på "last opp" knappen uten at det har blitt lagt til noen filer. Prøv så å legge til tre riktige filer, og så laste dem opp. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Bra tilbakemelding ved feil og suksess. Lett å forstå funksjon av knapper

5. Last ned filer

- c. Velg datatypen "elvedata", legg til elvene du vil laste ned dataen til. Last ned data for alle artene i elvene som er valgt. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å bruke menyen, men kunne vært forbedring for å lettere velge stasjoner og elver ved mange valg. Mulig løsning er å kunne filtrere stasjoner/elver på dato. Videre burde man både vise elve/stasjons-navn og dato ved seleksjon.

- d. Velg datatypen "stasjonsdata", legg til stasjonene du vil laste ned dataen til. Velg hvilke arter som skal være med i disse datasettene. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Samme kommentar som over. Ellers bra.

Tobias

5. Kart siden

- a. Filtrere art etter "Ørret" og "Harr" og liste opp alle elvedataene mellom to selvvalgte datoer.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
Enkelt og intuitivt å bruke, ser bra ut.

- b. Filtrer etter alle arter. Velg ett elvedata punkt.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Enkelt og intuitivt

Prosjektdata vil bli ha start + slutt, ikke bare en data

- c. Åpne stasjon 2 under elvedatapunktet. Prøv å lese data fra stasjonspunktet som er valgt, finn sum av all fiskedata. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Fint strukturert data. Liker at man kan lukke seksjoner.

For stasjoner listen etter å ha trukket på et elvedata punkt, så er "vær" kolonnen mist viktig hvis det blir lite plass

- d. Lukk stasjonsdata elementet og kom tilbake til startside med bare kart. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
Enkelt og intuitivt

6. List siden

- g. Finn frem til liste siden. Se på siden. Prøv å sortere resultat etter stasjonsnavn.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
Enkelt og intuitivt. Likte at den viste pil.

Kan tenke på muligheten å søke på dato (på liste over alle stasjoner)

- h. Lat som at det er veldig mange stasjoner vist. Bruk søk og filter til å finne samme stasjon som i sted (Gudbrandsdalslågen - stasjon 2). Se på dataen til stasjonen.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å finne fram og bruke filter/søk.

Etter å ha valgt en stasjon i listen, så vil han ha muligheten til å trykke på en knapp for å vise nøyaktig hvor dette punktet er på kartet (også for elver)

- i. Gå tilbake til listesiden. Finn elvedata til Gudbrandsdalslågen, registrert 20.09.2023. Se på dataen til elven. Hvordan var det å finne frem? Er noe som

kan forbedres? Hva likte du?

Lett å navigere, lett å finne elv.

Header på elvedatalisten må endres, kan hete "lokalitet" eller bare elv

3. Graf siden

- k. Finn frem til graf siden. Se på siden.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Kan være klarer fordelt seksjoner. Ellers ser det bra ut.

- l. Du skal se på en stasjon som heter "alna elven stasjon 1". Du er interessert i artene Ørret og Harr, så velg disse. Du vil også se alle andre arter som en egen kategori. Se på diagrammene som kommer opp.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å bruke filter, diagram ser bra ut. Kunne kanskje ha mulighet til å hovre over filter som vil vise skrift som forklarer hva det gjør.

mulighet for å laste ned en figur (en knapp) (kanskje en meny dukker opp som sier hva du er i ferd med å laste ned)

- m. Bytt til sektordiagram for fordeling av arter. Lukk sektordiagrammet slik at den ikke vises lenger. Åpne boksplokk for fordeling av lengde, og bytt intervall til 20mm.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å bruke meny.

- n. Prøv å åpne elven alnaelven uten å forlate siden. Prøv å sammenligne den med elven "Gjøvikselven".

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Gir mening at man beholder samme elv.

- o. Gå tilbake til kartet i startsidene og velg en elvedata (alna elven) som du ønsker å se på og åpne grafen til denne elvedataen. Du har lyst til å se på stasjonsdataen som ligger under denne elven, og sammenligne disse stasjonene.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å finne fram. Intuitivt å bruke.

4. Laste opp filer

- g. Prøv å trykk på "last opp" knappen i headeren uten at du er logget inn.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Det gir mening at man ikke får tilgang. Lett å forstå menyen.

- h. Logg inn og se på siden. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Ser bra ut, intuitive symbol og knapper.

- i. Trykk på “last opp” knappen uten at det det har blitt lagt til noen filer. Prøv så å legge til tre riktige filer, og så laste dem opp. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Bra tilbakemelding ved feil og suksess. Lett å bruke.

Når filopplastning var en suksess, så vil en kanskje ha muligheten til å ha en knapp som er “vis i kart” som viser nylig opplastet data

5. Laste ned filer

- e. Velg datatypen “elvedata”, legg til elvene du vil laste ned dataen til. Last ned data for alle artene i elvene som er valgt. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å finne frem, valgene er lette å forstå.

Viktig å ha muligheten til å filtrere mer, for eksempel basert på dato.. Også vil ha startdato på elvene, siden de besøker samme elvene i ulike år Burde kunne laste ned dataen i ulike formater (csv, xlsx) (også pdf, så man kan lese en rapport om dataen)

Se på muligheten når man laster ned, at det man laster ned har flere sider (csv)

- f. Velg datatypen “stasjonsdata”, legg til stasjonene du vil laste ned dataen til. Velg hvilke arter som skal være med i disse datasettene. Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Lett å bruke, samme kommentarer som over.

Summary

The user test gave valuable insight into what was done well and what could be improved. There were also several features mentioned which will be implemented for the next iteration of the wireframe.

What was good:

- Interacting with the map page and filtering points was intuitive.
- Use of symbols was good, helped with understanding functionality of buttons.

- Navigation between pages and site structure was easy to understand and intuitive.
 - Multiple paths between pages.
- Easy to use search/list page, good to be able to search on project ID or name.
- Good feedback when selecting items on the different pages, such as highlighting selected elements.
- Good to have default values in input fields.
- Easy to interact with and read diagrams.
- Good and informative feedback when making mistakes or when errors happen.

Areas for improvement:

- Could more clearly section different parts of the graph page
- Could add help text for filters when a user hovers over them

Recommended New features:

- Should display start and stop date for river observations
- Should display average fish length for each species as well as comments on the station page
- Should be able to view a river or station selected in the map when searching on the list page.
- To be able to download a figure from the graph page.
- Should be able to view a station or river in the map when it is uploaded
- Should be able to download data in multiple formats
- Should be able to filter station and river by dates when selecting them for download

Signature:

Knut Marius Myrvold



Lillehammer 14.02.2024

Tobias Holter



Lillehammer 14.02.2024

Appendix K

User Test 3

User Test 10.04.2024

Planning

Goals with User test

- Understand how different users interact with our web application.
- Evaluate the intuitiveness, efficiency, performance, design (e.g. use of colors) as well as the layout.
- Evaluate if the application satisfies NINA's expectations regarding the functionality and the layout of the web application.
- Use this to validate and find weaknesses, such as bugs, in our current iteration of the web application, by identifying both user satisfaction and usability issues.
- Iterate on our design further based on the findings.

Methodology

This will be the final user test where we will conduct a "usability test" and look for potential bugs. During this test, the test subjects will perform certain tasks on the web application. The usability test will be conducted remotely, where the test subjects will be given a URL link which will connect them to the current iteration of the website. While performing the tasks, they will get to answer some open-ended questions regarding the intuitiveness and performance of the website. Both tests will be conducted in Norwegian. For the duration of the test the participants will also be asked to share their screens, so that we can see their progress and assess how they go forward with solving the tasks.

Participants

Session 1

Name	Francesco Frassinelli
Age group	Adult
Gender	Male
Location	Trondheim
Technology use	High
Relation to product	Backend programmer NINA

Name	Tobias Holter
Age group	Adult
Gender	Male
Location	Lillehammer
Technology use	Medium
Relation to product	Product owner

User Test Structure

Descriptions of the website

Dette programmet går ut på å la forskere hos NINA visualisere og jobbe med data de har samlet fra elver i Norge. Dataen beskriver fisk NINA har fanget, og egenskaper ved fiskene, som lengde og kjønn. Det er et kart og en liste som lar deg se de forskjellige punktene med data. Det er to forskjellige typer data punkt: Elvedata, og stasjonsdata. Elvedata er observasjoner ved en elv på en dag, der stasjoner er alle de mindre strekningene med observasjoner som ble gjort i den elven. Man kan også se data for hver stasjon med grafer, samt å laste opp og ned data.

Questions

1. Kart siden

- a. **Filtrere art etter "Ørret" og liste opp alle stasjonsdataene mellom år 2020 og 2024.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. **Lukk og åpne filter menyen på venstre side.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- c. **Velg ett elvedata punkt og legg merke til at det er flere stasjoner som blir listet opp. Prøv å lese data for alle observasjonene under elven. Åpne så en stasjon under elvedatapunktet.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- d. **Prøv å lese data fra stasjonspunktet som er valgt, finn stasjonsbeskrivelsen.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

- e. Prøv å gå tilbake til elven som stasjonen ligger under. Lukk elvedata elementet og kom tilbake til startsidene med bare kart.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- f. Endre til satellitt og så topologisk kart, og se rundt.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

2. List siden

- a. Finn frem til liste siden.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Prøv å finne stasjon Gudbrandsdalslågen - stasjon 2. Prøv å bruke filter og søk.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- c. Finn elvedata til Gudbrandsdalslågen, registrert 20.09.2023.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

3. Graf siden

- a. Finn frem til graf siden.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Velg elven Gudbrandsdalslågen og velg art-ene Ørret, Harr og Ørekyte.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- c. Prøv de forskjellige innstillingene for fordeling av arter.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- d. Prøv de forskjellige innstillingene for fordeling av lengde.
Prøv å bytte intervallet til 40 mm. Bytt til å vise fordelingen med boksplokk. Lukk elementet.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- e. Gå tilbake til kartet i startsidene og velg en elvedata som du ønsker å se på og åpne grafen til denne elvedataen.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- f. Velg en elv til å sammenligne med og velg to arter du vil fokusere på. Du vil også se de andre artene, men du vil gruppere alle de andre artene sammen.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

- g. På histogrammet vil du bare se de to elvene sammenlignet, uten å bry deg om art. Prøv å gjøre det.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

4. Laste opp filer

- a. Prøv å trykke på “last opp” knappen uten at det er noen filer lagt til.
Prøv å legge til filer (Vurderr å legge til dette).
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- b. Prøv å legge til tre riktige filer.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

5. Log ut

- a. Prøv å logge ut av systemet.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

6. Last ned

- a. Gå til liste siden og prøv å komme deg til last ned siden.
Hvordan var denne opplevelsen, var det enekt å finne frem?
- b. Prøv å laste ned dokumentet i excel format og se på dokumentet som ble lastet ned.
Hvordan var opplevelsen ser dokumentet ut som det sal være. Er du fornøyd med formatet?
- c. Prøv å laste ned en ny fil som inneholder stasjonsdata. Hvordan var dette, kom du fram?

Observations and feedback - Session 3

Francesco

1. Kart siden

- a. Filtrere art etter “Ørret” og liste opp alle stasjonsdataene mellom år 2020 og 2024.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Det var intuitivt og klart å filtrere etter stasjoner, data og art. Derimot kunne det vært fint med en drop-down meny når man legger til egendefinerte arter. Kunne også tydeliggjort hva som menes med “valgte arter”, burde komme

tydeligere frem at punktene som vises på kartet viser alle stasjoner/ elver som har minst et av artene som er valgt.

b. Lukk og åpne filter menyen på venstre side.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Liker filter menyen, glitcher ikke. På min laptop med en litt smalere skjerm fikk jeg en scrollbar til høyre, på grunn av litt for mye padding.

c. Velg ett elvedata punkt og legg merke til at det er flere stasjoner som blir listet opp. Prøv å lese data for alle observasjonene under elven.

Åpne så en stasjon under elvedatapunktet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Det var lett å velge et datapunkt på kartet og de ble tydelig inndelt med farge og zoomet inn på valgte elementer. Når jeg valgte elvedatapunktet og gikk ned til tabellen var det fint å sortere etter kolonnene, men ikke intuitivt at hele tabell raden var klikkbar. En løsning vil være å kun gjøre ID nummerene klikkbare. En annen observasjon var at navnet på elven ikke hadde en klistret plassering, noe som gjorde det litt uoversiktlig.

d. Prøv å lese data fra stasjonspunktet som er valgt, finn stasjonsbeskrivelsen.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Det så veldig bra ut og muligheten til å sortere på kolonnene var bra, men kunne også ha lagt til stasjonsnavn og nummer klistret (sticky) på menyen, samme for elvedata.

e. Prøv å gå tilbake til elven som stasjonen ligger under. Lukk elvedata elementet og kom tilbake til startsidene med bare kart.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Ja dette var intuitivt og gikk lett for seg, ga mening i forhold til hvordan det ble gjort i de tidligere stegene. Merket at når filteret og informasjonssiden var synlig, burde elementene som zoom inn og ut, satellitt, topologi og kart mulighetene være synlige og ikke gjemt bak informasjonssiden.

f. Endre til satellitt og så topologisk kart, og se rundt.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Det var fint med et satellitt view og intuitivt å finne frem så fort man lukket informasjonssiden.

2. List siden

a. Finn frem til liste siden.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Ja, det var intuitivt å finne fram i nav baren. Derimot ble jeg ikke redirected tilbake til kartsiden når jeg kikket på tilbakeknappen i nettleseren.

b. Prøv å finne stasjon Gudbrandsdalslågen - stasjon 2. Prøv å bruke filter og søk.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Det var lett å filtrere etter stasjoner for å søke opp riktig stasjon. Syntes det var fint at dere bare tillot små bokstaver i søkefeltet. Derimot når jeg klikker på “vis i kart” som fører meg til stasjonens plassering på kartet, ville det vært intuitivt å ha en “returner tilbake til liste” knapp. Kunne istedenfor hatt et lite kartvindu på liste-siden, men det ville tatt mer tid.

c. Finn elvedata til Gudbrandsdalslågen, registrert 20.09.2023.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Det var igjen lett å finne fram og det fulgte den samme løsningen som ved kart siden.

3. Graf siden

a. Finn frem til graf siden.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Var intuitivt å finne fram til kart siden. Derimot syntes jeg det burde vært et mer generisk navn en “rediger elver”, ettersom at vi også behandler stasjoner i filter menyen. Videre når vi trykte på “velg elver”, valgte stasjonsdata og søkte etter en stasjon kom det alt for mange forslag opp. Kan sette en maksgrense til ti forslag. Videre kunne man også kanskje byttet navn på plotly sine innebygde skjermbilde funksjon til å navngi bildene etter hva dataene representerte.

b. Velg elven Gudbrandsdalslågen og velg art-ene Ørret, Harr og Ørekyte.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Grafene så fine ut, kunne kanskje skjult valgmulighetene til grafen om “vis” ikke var haket av. Når jeg toggler “vis” knappen vil jeg at siden skal “ankre” (skroller automatisk for å fokusere på grafen som ble valgt til å vises), slik at brukeropplevelsen blir bedre for de med mindre skjermer

c. Prøv de forskjellige innstillingene for fordeling av arter.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Innstillingene for fordeling av arter så veldig fine ut, og ser ingen umiddelbare forbedrings punkter.

d. Prøv de forskjellige innstillingene for fordeling av lengde.

Prøv å bytte intervallet til 40 mm. Bytt til å vise fordelingen med boksploott. Lukk elementet.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Innstillingene for fordeling av lengde så veldig fine ut og tabellen så også fin ut med 40 mm intervall. Ser ingen umiddelbare forbedringer her heller.

e. Gå tilbake til kartet i startsidene og velg en elvedata som du ønsker å se på og åpne grafen til denne elvedataen.

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Det var veldig intuitivt å finne grafen til en elv, her brukes samme logikk som tidligere.

- f. **Velg en elv til å sammenligne med og velg to arter du vil fokusere på. Du vil også se de andre artene, men du vil gruppere alle de andre artene sammen.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
Her brukte jeg samme funksjon som tidligere hvor jeg la til to elver, og trykket deretter på grupper elementer, noe som var veldig lett forståelig.

- g. **På histogrammet vil du bare se de to elvene sammenlignet, uten å bry deg om art. Prøv å gjøre det.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
Sammenlikningen så veldig fin ut, men når jeg velger arter og så fjerner alle valgte arter vil jeg returnere status til "velg alle" (ettersom at ingen egendefinerte arter er valgt), da korresponderer filter siden og graf siden mye bedre.

4. Laste opp filer

- a. **Prøv å trykke på "last opp" knappen uten at det er noen filer lagt til. Prøv å legge til filer (Vurderr å legge til dette).**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
Dette så veldig fint ut, meningen er å kun laste ned en fil om gangen i utgangspunktet, og filer større enn 100 mb excel filer er blokkert fra vår kode. Ville lagt til mer "button-margin" for "last opp" knappen.

- b. **Prøv å legge til tre riktige filer.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
Ser bra ut dette, men når jeg trykker laste opp, burde alle tidligere listede-elementer fjernes, slik at om jeg skal laste opp noe filer, starter jeg med en blank side.

5. Log ut

- a. **Prøv å logge ut av systemet.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
Ser veldig intuitivt ut, kan være det er bedre å bytte ut logoen av en mann med et "logg ut" ikon. .

6. Last ned

- a. **Gå til liste siden og prøv å komme deg til last ned siden.**

Hvordan var denne opplevelsen, var det enekt å finne frem?

Dette gikk fint. Burde fjerne csv og ha xlsx valgt som standard. Her opplevde jeg samme problemet med "rediger elver", og "elver valgt" når det kom til å kunne velge stasjoner, burde derfor ha et mer generisk navn på disse knappene.

- b. **Prøv å laste ned dokumentet i excel format og se på dokumentet som ble lastet ned.**
Hvordan var opplevelsen ser dokumentet ut som det sal være. Er du fornøyd med formatet?
Var enkelt å laste ned og det så veldig bra ut, dokumentet fordelte seg fint i tre excel sheets.
- c. **Prøv å laste ned en ny fil som inneholder stasjonsdata. Hvordan var dette, kom du fram?**
Helt som forventet, fikk to excel sheets som forventet.

Tobias

1. Kart siden

- a. **Filtrere art etter "Ørret" og liste opp alle stasjonsdataene mellom år 2020 og 2024.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- Lurer på om det er mulig å lage custom meny for når man velger dato. Nevner at aure og ørret er det samme, de bare bytter på å skrive av og til. De skal fikse det selv.
Vil kanskje også ha mulighet for "og", altså når han velger ørret og gjedde vil man kunne se bare de med gjedde og ørret, og ikke de med gjedde eller ørret.
- b. **Lukk og åpne filter menyen på venstre side.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- Ser veldig bra ut
- c. **Velg ett elvedata punkt og legg merke til at det er flere stasjoner som blir listet opp. Prøv å lese data for alle observasjonene under elven. Åpne så en stasjon under elvedatapunktet.**
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?
- Nevner at "min fisket" burde være sekunder fisket.
Veldig fornøyd med visning av fiskedata. Synes det var veldig oversiktlig, lett og forståelig.
Fisker der man ikke har målt lengde, påvirker diagram-siden negativt. De vises ikke på summen av fisker. Den må inkludere alle fiskene, men bare ikke telle med lengden på gjennomsnitt og andre ting. Altså antall fisk er ikke riktig på graf siden.

Likte veldig det at man kan ta egendefinerte arter på graf siden. Også det at man kunne sette intervallet, var veldig nyttig. Bruker intervallet til å se på fordelingen av ulike generasjoner fisk.

Synes ideen om at man kan trykke på stasjoner under elvedata var genialt. Mener kanskje knappen "til elvedata" som finnes på stasjonsdata kan ha en tilbakepil, og sier "tilbake til elvedata". (ikke så viktig, kan nedprioriteres)
Liker at kartet zoomer inn og ut automatisk

Vannføring burde ha enhet, altså slik som tid har "sek" og "min", mens antall har "stk".

Usikker på om kommentaren vises ordentlig. Mener vi burde ta en dobbeltsjekk for å passe på at det er riktig.

- d. Prøv å lese data fra stasjonspunktet som er valgt, finn stasjonsbeskrivelsen.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?**

Funket veldig bra.

- e. Prøv å gå tilbake til elven som stasjonen ligger under. Lukk elvedata elementet og kom tilbake til startsidene med bare kart.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?**

Var enkelt og greit

- f. Endre til satellitt og så topologisk kart, og se rundt.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?**

Liker at man kan ha ulike karttyper. Vet ikke om det er nyttig, men liker muligheten. Så at et av karttypene hadde retning på elven, som var litt interessant. Sier også at man ikke alltid kan stole på satellittbilder, kan være misvisende siden bildene ikke alltid er helt nye.

2. List siden

- a. Finn frem til liste siden.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?**

Lett å gå til liste, siden. Lett å vite at man er på listesiden, siden det er markert med strek under.

- b. Prøv å finne stasjon Gudbrandsdalslågen - stasjon 2. Prøv å bruke filter og søk.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?**

Liker søkesiden, synes den var enkel å bruke. Likte at han kunne sortere på arter. Likte at den var responsiv.

- c. Finn elvedata til Gudbrandsdalslågen, registrert 20.09.2023.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?**

Elvedata siden ser veldig bra ut. Synes fremstillingen var veldig fin. Påpekte at vi har litt ledig plass, kan kanskje legge noe der. Kanskje man kan legge inn bilde av stasjonen på kartet, men det er ikke noe vi burde prioritere.

Enkelt å navigere fra liste til kart-siden.

Vil ha mulighet å gå tilbake fra kart til liste (etter at han har gått fra liste til kart)

Kanskje hvis man trykker på en stasjon og så trykke liste, så vises den i listen

3. Graf siden

- a. Finn frem til graf siden.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?**

Lett å navigere til graf siden

- b. Velg elven Gudbrandsdalslågen og velg art-ene Ørret, Harr og Ørekyte.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?**

Synes det var greit å finne fram til elven. Menyene var greie å bruke.

Mener at "rediger elver" knappen kanskje burde vært "stylet" annerledes, så den er lik som andre knapper.

Kjempeintuitivt å velge arter.

Trenger ikke fjerne verktøylinjen øverst på hver graf, selv om vi eller han var usikker på hva den gjorde

- c. Prøv de forskjellige instillingene for fordeling av arter.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?**

Synes innstillingene var greie, og lette å bruke. Synes grafene var kjempefine å bruke. Ser kjempebra ut.

- d. Prøv de forskjellige instillingene for fordeling av lengde.
Prøv å bytte intervallet til 40 mm. Bytt til å vise fordelingen med boksplokk. Lukk elementet.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?**

Alt fungerte slik han forventet, ser bra ut.

- e. Gå tilbake til kartet i startsidene og velg en elvedata som du ønsker å se på og åpne grafen til denne elvedataen.
Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?**

Det var bra at man kunne gå fra kart til graf. Sier de kommer til å bruke den en god del. Synes at det vil være greit at man går fra kart->graf->kart, så vises stasjonen på kartet fortsatt. Hvis flere er brukt på graf, så burde bare ingen være selected.

- f. **Velg en elv til å sammenligne med og velg to arter du vil fokusere på. Du vil også se de andre artene, men du vil gruppere alle de andre artene sammen.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Veldig "fit" at man kan sammenligne elver. Synes det så veldig nyttig ut.

- g. **På histogrammet vil du bare se de to elvene sammenlignet, uten å bry deg om art. Prøv å gjøre det.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Synes det er gøy å trykke på de ulike mulighetene på graf siden. At det var så mange muligheter gjør at det å "leke" med nettsiden for å se dataene på ulike måter blir gøy.

4. Laste opp filer

- a. **Prøv å trykke på "last opp" knappen uten at det er noen filer lagt til. Prøv å legge til filer (Vurderr å legge til dette).**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Synes siden var enkel å grei. Det er et problem at man ikke kan fjerne filer man har valgt. Usikker på om det var meningen at man kan velge både csv filer og xlsx filer samtidig.

- b. **Prøv å legge til tre riktige filer.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Fungerte bra, var enkelt og greit.

5. Log ut

- a. **Prøv å logge ut av systemet.**

Hvordan var det å finne frem? Er noe som kan forbedres? Hva likte du?

Fant logg ut knappen. Funktet jo ikke nå, men så veldig greit ut.

6. Last ned

- a. **Gå til liste siden og prøv å komme deg til last ned siden.**

Hvordan var denne opplevelsen, var det enkelt å finne frem?

Var enkelt og greit.

b. Prøv å laste ned dokumentet i excel format og se på dokumentet som ble lastet ned.

Hvordan var opplevelsen ser dokumentet ut som det skal være. Er du fornøyd med formatet?

Synes det var gøy å teste ut all funksjonalitet på nettsiden. Bryr seg ikke om id til fisken. TRUE skal være ja (på data i excel filen).

Ønsker at stasjon burde være lik som i elvedata. Altså nummeret i excel/csv filen.

c. Prøv å laste ned en ny fil som inneholder stasjonsdata. Hvordan var dette, kom du fram?

Virket bra, veldig fornøyd.

Summary

~~The user test gave valuable insight into what was done well and what could be improved. There were also several features mentioned which will be implemented for the next iteration of the wireframe:~~

The user test gave valuable insight into what was good and what could be improved. There were some features that were mentioned that could be improved and which will be implemented for the next iteration of the web application.

The user test gave us valuable insight regarding our current iteration, and what could be improved. There were some suggestions for improvements and we are going to implement most of them, but because of the little timespan, we will not be able to complete every improvement that was suggested. We therefore have to rank the suggestions based on how vital they are, to make sure that all the important improvements are made.

What was good:

- The site was fast and responsive
- The line under the page name made it easy to know which pager you were in.
- The automatic zoom function for the map made it easy to use.
- Intuitive way to choose different species.
- Nice option to sort the tables after the different species.
- Useful to compare the data from multiple rivers.
- Loved all the different functionalities, especially on the graph site.

- The graphs were fun to use and were fun to play around with the different settings
- The site was easy to use, no real problems
- Didn't encounter any serious bugs.
- Easy to know

Areas for improvement:

- Make sure that all buttons and text are accurate and clear, and follow the same format.
- Make dropdown choices for all the search bars and make a maximum of how many names get listed on the dropdown choices.
- Have the site remember what the user does, so it's easier to transition between different parts of the site.
- Lessen the padding size in the map page to get rid of the scrollbar on smaller screens and add more bottom margin to the button on the upload page.
- Change the log out icon from a human shadow to a log out icon.

Recommended New features:

- Add the possibility to remove files you want to upload after they were selected.
- On the pop-up menus, there should be a confirm button to signify to the user that what they have selected is saved.
- Add a small window in the river data and station data page, showing where the station or the river is.
- Hide choices if the user chooses to hide one or both the graphs.

Tips til presentasjon:

Ta orkla, sorter på laks med lengdefordeling. Da kan man se de ulike årsgruppene, som viser befolkning av hver årsgruppe.

Også halligsdalelva. Sammenlign to ulike år. Har lagt inn tiltak for å få færre gjedder, og det kan man se på grafen.

Spør NINA om tips til hva vi kan presentere

Signature

Tobias Holter



Lillehammer 10.04.2024

Appendix L

Threat Modeling

Threat modeling is a structured approach used to identify and mitigate potential security threats. The threat modeling process is a core part of software development. The overarching goal with threat modeling is to get a clear view over the most critical risks and ways to reduce the risks using security controls.

The process of threat modeling can be separated into four phases. These phases are modeling the application architecture, identifying threats, finding and analyzing security controls, and lastly documenting and validating the results.

Model Application Architecture

The application architecture modeling phase is about gaining a comprehensive understanding of the systems structure and interactions, aiding in the identification of potential security vulnerabilities. By visualizing the system's components and their relationships, it's easier to identify potential threats and attack options, which in turn makes it possible to implement relevant security controls.

The Physical Topology:

The physical topology of the application provides valuable insight into where and how the application will be deployed. This interface is however mostly controlled by NINA, since they will handle the back-end surrounding the application. This implies the express server, postgREST, postgREST database and LDAP. Based on the context established, the application will be running on an internal network, with access to a postgREST database server and an express server as depicted in *Figure L.1*.

Logical Topology:

The logical topology of the application illustrates the logical tiers; presentation, business, service and data. These tiers are further represented in the following *Figure L.2*, by components, services, protocols and ports that are configured and implemented as part of the application. It also covers the identities that are used to determine how the authentication will be designed

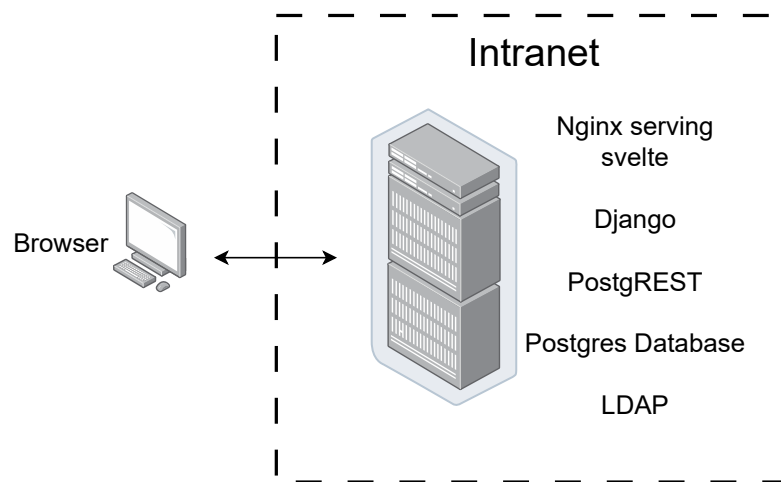


Figure L.1: Physical Topology of the web application

in the application.

Human Actors of the System:

In collaboration with NINA, we have defined three types of human actors of the system:

- Regular user - Normal end-user with granted access to the web application. This user has the minimal amount of privileges possible.
- Privileged user - Users granted extra privileges, like uploading and/or downloading data.
- Administrator - Responsible for managing and overseeing the operation of the application.

NINA has mentioned that the different user's IT knowledge can be seen as decent, however this varies between different users.

Data Access Control Matrix:

To get an understanding of the data used by the application, a data access control matrix was designed (see Table L.1). This matrix lists the types of data used by the application, and shows what privileges different users have in relation to each type of data. The privileges are determined by CRUD, which stands for:

- **Create** - privilege of being able to add new data to the database
- **Read** - privilege of being able to read existing data from the database
- **Update** - privilege of being able to modify existing data in the database
- **Delete** - privilege of being able to delete data from the database

Technologies used in building the application

The web application will be developed using SvelteKit. The files for the web application will be distributed to the end users using a server created with express.

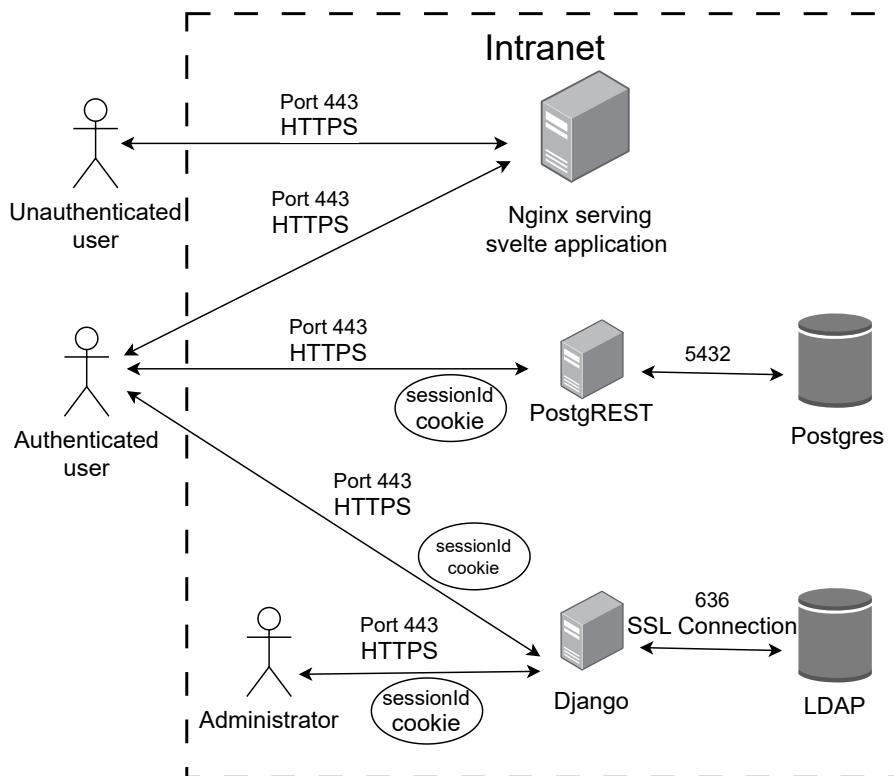


Figure L.2: Logical Topology of the web application

Type of data	Regular user	Privileged user	Administrator
Research data	R	C R	C R U D

Table L.1: Data Access Control Matrix

Identify Threats

In this section of threat modeling the goal is to uncover threats and vulnerabilities. The first part of this process will be to map trust boundaries, entry points, exit points, data flows and privileged functionality. Additionally possible mis-actors will be introduced, which is used to determine potential and applicable threats.

Identifying trust boundaries

Trust boundaries helps identify what actions and behavior are appropriate and inappropriate. A trust boundary is a point where the privilege level changes. It's therefore crucial to identify trust boundaries and adequately add a level of protection within each boundary. The trust boundaries are illustrated in Figure L.3.

For the web application we are developing, the trust boundaries are located between the

web browser and the application since the user gets elevated privileges when entering NINA's intranet, and between the front-end and authentication server since the authentication server needs higher privileges. Moreover, there are trust boundaries between the front-end and postgREST, since it holds all data and information, and between front-end and express since the express server manages requests, responses and request deletion. Lastly, there is a trust boundary between postgREST and the authentication server, since the authentication server needs higher privileges.

The security around the authentication method will be managed by NINA, however the application will need to implement NINA's authentication method. This means the application will have to manage a cookie for authentication of users, but the development and maintenance of this authentication solution is done by NINA.

Identifying entry points

Identifying entry points is important, since entry points define areas of the system where data can enter. Entry points can be described as places in the system which takes user input. These are prime targets for threat actors, as poor handling of entry points allows for entering dangerous input. The application has three entry points:

- URL
 - Setting url parameters can be used to select rivers and stations on the webpage. It takes in their ID, and sends this ID to the postgREST API to retrieve their information. This means that users can insert content into the URL to access PostgREST.
- Search field
 - The web application utilizes a search field to filter station/river data shown, or to locate specific data. There are also other text input fields on the application.
- Import data feature
 - An import feature was added to allow authorized user to upload data directly to the back-end database.

Identifying exit points

Exit points are parts of the web application that displays data from inside the system or parts that transfers data out of the system. It can be the source of information disclosure, and therefore vital to protect. One exit point is the entire site, since it is displaying data from NINA's back-end. This means data stored in the back-end is gathered and visualized by the browser for all users that can access the website. Another exit point is the export feature the application contains. This allows privileged users to export parts of the data stored in NINA's back-end. This feature moves data that is stored in the back-end entirely out of the system, to the users local storage.

Data flow diagram

The Data Flow Diagram (DFD) (see Figure L.3) illustrates how the application will accept, process and handle data as it moves across trust boundaries. It also distinguishes between external and internal processes and data stores. The DFD represents the flow of data graphically. This includes the back-end data, storage elements and their relationships between data sources and

destinations. An explanation for the different parts of the DFD is illustrated on the right hand side in the figure.

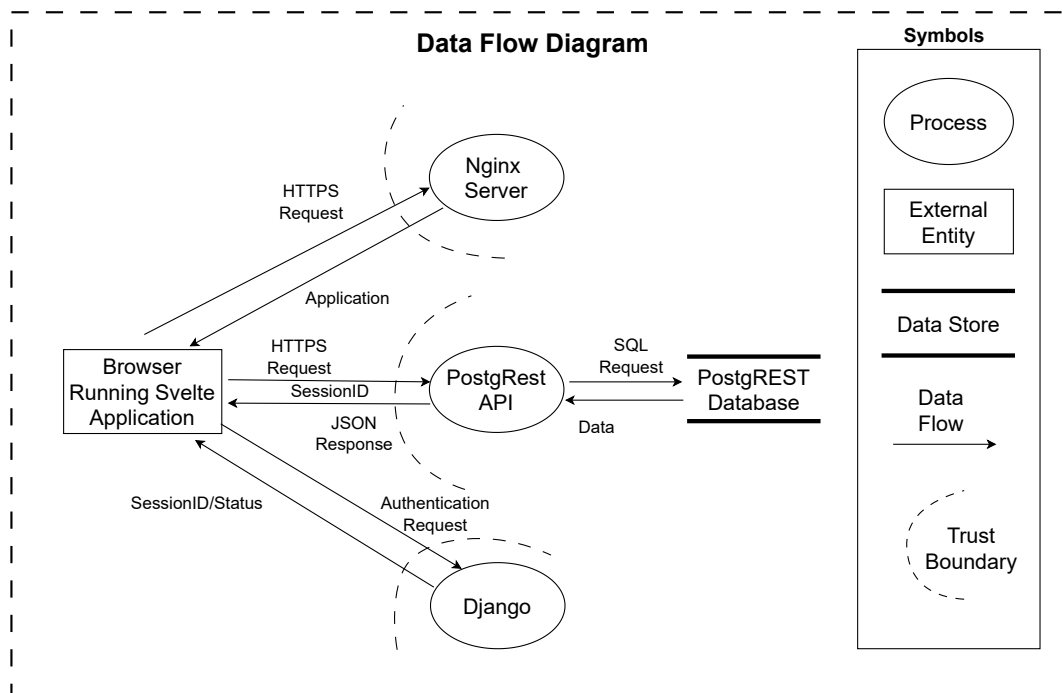


Figure L.3: Data Flow Diagram for the web application

Identifying mis-actors

Before identifying the threats, it's important to establish relevant mis-actors. These are users who intentionally try harming the systems. Mis-actors can be actors such as ignorant users, external hackers, rogue administrators, hacktivist groups or fraudulent sales administrators. In the following table there is an overview over relevant mis-actors, and a description of why they pose a threat to the application. We will first take an in depth look at the human mis-actors, and later mention non-human mis-actors.

Mis-actor	Description
Cyber Criminals	Refers to any criminal using the cyberspace to attack the application
Competitors	Competitors gathering confidential information for the purpose of gaining a competitive advantage. Could also be threats wanting to gather confidential information to sell to interested parties.
Insiders	Typically individuals who have privileged access or knowledge within an organization, which they can use to achieve activities such as leaking confidential information.
Ignorant User	An employee who causes damage to NINA without having bad intentions.
Script Kiddies	Individuals who utilizes easily available tools to conduct attacks or take advantage of known vulnerabilities in the system.

Table L.2: Description of relevant mis-actors

Mis-actor	Goals	Technical ability	Risk tolerance	Work-factor
Cyber Criminals	Monetary	Medium	Low/Medium	Low/Medium
Competitors	Information	High	Low	High
Insiders	Monetary	Medium/High	High	Medium
Ignorant User	None	-	-	-
Script Kiddies	Recognition/Monetary	Low	High	Low

Table L.3: Model describing each mis-actor

Table L.4 describes some noteworthy non-human mis-actors. Non-human mis-actors refer to entities/elements that can cause malicious activities in the system without human intervention.

Mis-actor	Description
Loss of essential services	Services and libraries that the application depends on can stop working, or be manipulated into no longer being safe
Compromise of information	Manipulation or deletion of information
Technical failure	Technical failure refers to an unexpected problems in a system, equipment, or technology that disrupts its normal working condition.

Table L.4: Description of relevant non-human mis-actors

Identifying threats

When identifying threats, we opted for the STRIDE method. The STRIDE method is a goal-

based approach, since it categorizes the threats based on their goals. Each letter of STRIDE relates to one type of goal, which is described further in *Table L.5*

	Goal	Description
S	Spoofing	Can an attacker impersonate another user or identity?
T	Tampering	Can the data be tampered with while it is in transit or in storage or archives?
R	Repudiation	Can the attacker (user or process) deny the attack?
I	Information Disclosure	Can information be disclosed to unauthorized users?
D	Denial of Service	Is denial of service a possibility?
E	Elevation of Privilege	Can the attacker bypass least privilege implementation and execute the software at elevated or administrative privileges?

Table L.5: STRIDE definition

Identify, Prioritize and Implement Controls

To quantitatively and qualitatively represent the risk ranking of the identified threats, we will follow the DREAD (Damage Potential, Reproducibility, Exploitability, Affected Users, and Discoverability) framework. The framework uses the average ranking system consisting of numeric values, from one to three, for the risk ranking categories probability, impact and business impact.

1. **(D - Damage Potential)** Ranks the damage that will be caused when a threat is materialized or vulnerability exploited.
 - a. 1: Nothing
 - b. 2: Individual user data is compromised or affected
 - c. 3: Complete system or data destruction
2. **(R - Reproducibility)** Ranks the ease of being able to recreate the threat and the frequency of the threat exploiting the underlying vulnerability successfully.
 - a. 1: Very hard or impossible, even for administrators of the application
 - b. 2: One or two steps required; may need to be an authorized user
 - c. 3: Just the address bar in a web browser is sufficient, without authentication
3. **(E - Exploitability)** Ranks the effort that is necessary for the threat to be manifested and the preconditions, if any, that are needed to materialize the threat.
 - a. 1: Advanced programming and networking knowledge, with custom or advanced attack tools
 - b. 2: Malware exists on the Internet, or an exploit is easily performed using available attack tools
 - c. 3: Just a web browser

4. **(A - Affected Users)** Ranks the number of users or installed instances of the software that will be impacted if the threat materializes.
 - a. 1: None
 - b. 2: Some users or systems, but not all
 - c. 3: All users
5. **(D - Discoverability)** Ranks how easy it is for external researchers and attackers to discover the threat, if left unaddressed.
 - a. 1: Very hard-to-impossible; requires source code or administrative access
 - b. 2: Can figure it out by guessing or by monitoring network traces
 - c. 3: Information is visible in the web browser address bar or in a form

In the following table, the threats identified in table L.6 are ranked according to DREAD. The risk calculation is done by multiplying Probability of Occurrence by Business Impact. This is using the values calculated with Dread, where Probability is equal to the sum of Reproducibility, Exploitability and Discoverability, while Impact is the sum of the values for Damage potential and Potential users.

Threat	D	R	E	A	D	Risk (P x I)
Spoofing a file	3	3	2	3	2	48
Spoofing a machine / user	2	1	2	2	1	16
Tampering with a file	3	1	1	3	3	30
Repudiation of actions	2	2	3	2	2	28
Error message manipulation	2	3	3	2	2	32
Information disclosure	2	3	3	1	2	24
Denial of service	1	3	3	2	3	27
Elevation of privilege	2	1	1	2	3	20

Table L.7: DREAD Ranking

Document and Validate

The final part of the threat modeling process is called document and validate, and involves documenting the threats in a structured manner to find applicable safeguards. Safeguards are supposed to be implemented to reduce the risks involved with each threat. Threats with higher scores are prioritized when it comes to finding applicable safeguards and implementing these safeguards. Best practice involves utilizing proven safeguards for the threats, as these are better documented in terms of effectiveness and what threats they mitigate. All identified threats will be listed with some information too further help specify the threat. These details include the threats target, attack techniques one can use to accomplish the threat, the security impact the threat will have if it succeeds, the risk associated with each threat and possible safeguards to implement to reduce the risk.

1. **Spoofing a file:**

- Threat Identifier: #01
- Targets: Back-end database
- Attack techniques: File extension spoofing, mimicking file names or icons, Embedding malicious code in documents using the upload feature of the application
- Security Impact: Malware Infection, Malware Propagation, System or Network compromise
- Risk: Very High
- Safeguard to implement: File type restrictions, File Size Limitations, Input validation, Logging of attempts to upload files

2. Spoofing a user or a machine:

- Threat Identifier: #02
- Targets: Authentication methods, user credentials and network services.
- Attack techniques: IP spoofing, ARP spoofing, DNS spoofing, Session hijacking.
- Security Impact: Data breach, Network intrusion, Identity theft, Service disruption
- Risk: Low
- Safeguard to implement: Strong authentication mechanisms, strong encryption, Network segmentation and monitoring, Secure DNS practices, ARP spoofing detection tools and Educating users

3. Tampering with a file:

- Threat Identifier: #03
- Targets: External libraries, System files
- Attack techniques: Gaining control over external libraries to include hidden malicious code, Hiding malicious code inside system files
- Security Impact: Data Breach, Malware Infection, System compromise
- Risk: High
- Safeguard to implement: Documenting of dependencies, Code signing, Integrity checks on the system files, System monitoring, Updating dependencies to newest versions

4. Repudiation of actions:

- Threat Identifier: #04
- Targets: login page, importing and exporting data fields
- Attack techniques: Utilizing gained credibility, Persuasiveness
- Security Impact: Loss of trust or Fraud
- Risk: Medium
- Safeguard to implement: Logging and monitoring, Notification when users are authorised from a new devices.

5. Error message manipulation:

- Threat Identifier: #05
- Targets: Website features and functionality
- Attack techniques: Fuzzing, Brute force, Timeout-based attacks
- Security Impact: Information disclosure, denial of Service

- Risk: Medium
- Safeguard to implement: Input validation, proper error handling, proper coding practices, error logging

6. Information disclosure against data stores:

- Threat Identifier: #06
- Targets: Authentication server, transit data.
- Attack techniques: SQL injection, Cross-Site Scripting, Brute force
- Security Impact: Confidential information disclosure, Loss of data integrity.
- Risk: Low-Medium
- Safeguard to implement: Strict access control with sanitation, HttpOnly cookie, Same-site cookie, JSON Web token, Implementing separation of duties and security in depth

7. Denial of service:

- Threat Identifier: #07
- Targets: Website, Login page
- Attack techniques: Reflection, Amplification, Botnet
- Security Impact: System disruption
- Risk: High
- Safeguard to implement: Firewalls to filter traffic, Analyzing normal rate of traffic to easily identify denial of service attacks

8. Elevation of privilege:

- Threat Identifier: #08
- Targets: Trust boundaries, Admin users
- Attack techniques: Phishing, Brute forcing, SQL injection, Cross-Site Scripting.
- Security Impact: Confidential information disclosure, Loss of data integrity.
- Risk: Low
- Safeguard controls to implement: Strict access control with sanitation, HttpOnly cookie, SameSite cookie, JSON Web token, Implementing separation of duties and security in depth

Id	Type	Mis-actor	Description
1	S	Cyber criminals, Insiders, Script kiddies	Spoofing a file Uploading a malicious file by disguising it as file which will be approved by the application
2	S	Cyber criminals, Insiders, Script kiddies, Competitors	Spoofing a machine / user Utilize spoofing of ip-addresses or other network based identification to position yourself inside NINA's network
3	T	Cyber criminals, Insiders, Script kiddies	Tampering with a file Includes the manipulation of external libraries used by the application and manipulating the files the system uses to create the web application
4	R	Insiders, Ignorant users	Repudiation of actions Users claiming to have not done an action or users using the account of someone else
5	I	Cyber criminals, Insiders, Competitors, Script kiddies, Ignorant users	Error message manipulation Utilizing invalid input to force errors, hoping to extract information about the system because of improper error handling by the application
6	I	Cyber criminals, Insiders, Competitors, Script kiddies, Ignorant users	Information disclosure against data stores Taking advantage of weak or missing access control to gain access to data which you should not have access to or relying on poor system management to read files as they move around the network
7	D	Cyber criminals, Competitors, Script kiddies	Denial of service Utilizing denial of service attacks to request the webpage multiple times, overloading the API with work or overloading the login authorization mechanism making users unable to authorize themselves
8	E	Cyber criminals, Script kiddies, Ignorant users	Elevation of privilege Taking advantage of authorization failures, buggy authorization checks, missing authorization checks or tampering data related to authorization checks to gain privileges you're not supposed to have

Table L.6: Identified threats

Appendix M

Requirements Documentation

Business context and External Requirements

The Norwegian Institute for Nature Research (NINA) is an independent foundation dedicated to studying the natural world and its intersection with society. NINA engages in a wide range of activities related to this case, environmental monitoring, encompassing research, consultancy, and evaluation. Utilizing their expertise in species, ecosystems, and societal interactions with nature, NINA provides insights into the many environmental factors that exist, aiding in informed decision making and sustainable practices. [1]

NINA's department in Lillehammer regularly carry through research trips in different rivers throughout Norway and Sweden, collecting necessary data for NINA's operations and research. NINA's department in Lillehammer are in a need of a better way of utilizing their stored data. They are currently storing their collected data in separate excel sheets for each research trip. This is inefficient and impractical when trying to use the data for research, making the scientists spend unnecessary amounts of time organizing, aggregating, and analyzing their own data. We will develop a web application which is capable of visualizing the data NINA has collected in a simple, yet detailed manner. This allows NINA to focus less on technical overhead and more on research, which lets them put more effort into their business goals. NINA's overarching business goal is to contribute to sustainability and community development by delivering research based and relevant knowledge about natural diversity, climate and society.

External requirements defines external factors beyond the project's boundaries. For this project, the external factors are laws and regulations that needs to be complied with for our website to be seen as legal. The intention behind this website is for it to be used only on NINA's internal network, meaning most of these laws and regulations don't apply. However, to strengthen the quality and longevity of the application, we intend to follow the regulations as if our application is supposed to be utilized on the internet.

Copyright Laws are laws one is required to follow no matter the significance of the project. These state that any images, text or other content used in the application which were not originally created by us, must contain proper licensing. The intention behind this law is to grant rights to creators, performers and investors in intellectual property and related achievements,

while balancing these rights with the interests of users and the general public, in order to ensure fair use and safeguard freedom of information and expression, as well as facilitate the easy conclusion of agreements on the use of works [153].

In 2021 Norway approved a law requiring websites used in the public sector to follow the accessibility guidelines for web defined by WCAG 2.1 [154]. "Web Content Accessibility Guidelines (WCAG) 2.1 covers a wide range of recommendations for making web content more accessible. Following these guidelines will make content more accessible to a wider range of people with disabilities, including accommodations for blindness and low vision, deafness and hearing loss, limited movement, speech disabilities, ... " [155]. These guidelines are the globally adopted set of standards for web accessibility [156], meaning that it's seen as best practice to follow them, even if your not legally required to.

There are strict external regulations regarding the use of cookies and processing personal data. The Norwegian law [157] states that "storing information in the user's communication equipment, or gaining access to such, is not permitted without the user being informed about which information is being processed, the purpose of the processing, who is processing the information, and that the user has consented to this." This website intends to avoid storing personal data to make it more user friendly, and because we see no need to store and process personal data. The GDPR, which the Norwegian personal data processing laws are based on, also state that personal data should only be collected or processed if its necessary for the applications functionality, and its encouraged to achieve similar functionality without the collection and processing of personal data [158]. There will be an authorization system used by the website which will process personal data, however this authorization is done through NINA's systems.

As previously stated, this website is supposed to be used on NINA's internal network. This means the website won't need a domain, however we decided to include the domain name regulations for Norwegian domains anyway. This is to ensure that if NINA's intent to use the website on the internet, they will be aware of these regulations and the risks involved will be calculated. The purpose of these regulations are "to ensure that administration and assignment of domain names under .no serves the public interest and is in accordance with the guidelines provided by Norwegian authorities" [159].

Data classification

This application will only process environmental data. The environmental data is read from NINA's backend database, and used to create all markers about stations and rivers. The data also lays the foundation for the graphs and diagrams related to each marker on the website. Since the data is stored in Nina's backend database, it's their responsibility to implement a proper back-up system.

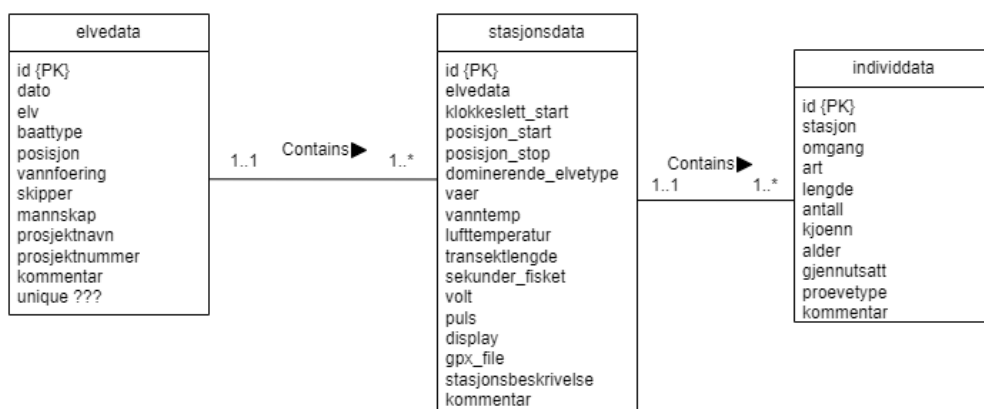
The environmental data is structured, confirming to the structured database language SQL.

NINA is using three SQL tables for storing all the environmental data. One is called "elvedata", and contains all data relevant to each research trip. The second table of data is "stasjonsdata", and it contains data connected to each station NINA's has visited. Multiple visits to the same station counts as separate rows in the "stasjonsdata" table. Lastly, the SQL database contains a table called "individdata", which contains data for each individual entity NINA has collected. These entities are fish, which are what NINA research during their trips down rivers.

All the environmental data is used to plot the different markers, graphs and diagrams on the application we're creating. Authorization is not needed to look at these visual elements, meaning anyone capable of opening the website will gain visual access to the environmental data. This sets the confidentiality level of the data as open. Even if the confidentiality level is low, the application will only run on NINA's internal network, meaning only users on this network will have access to the application. Regular users can also only see the data visually on the website, as downloading the data, or uploading new rows of data will require authorization. The integrity of the data is critical, as its critical that the data is always correct. This data is used as foundation for NINA's research, and also for planning new research trips. The entire application would be worthless, if one couldn't trust that the data visualized is correct. NINA's operations is not hugely dependent on always having the data available for them, setting the availability requirements of the data to 2 days. The 2 days represents a weekend, referencing that if NINA lost connection to the data on a Friday it wouldn't be a huge problem. They could simply wait until Monday to fix the issues.

As the all the data is necessary for the functionality of the website, current intentions from NINA are that the data stored in the database, is stored indefinitely. There is no indications of when data becomes outdated or no longer relevant for their research, meaning there is no system in place for deleting data after it's been stored for longer periods.

To illustrate how the tables of data are structured and how the tables interact with each other, we designed an entity-relationship model using the UML notation [160].



Subject-Object Matrix

A subject-object matrix is made to clearly envision allowable actions between subjects and objects. The subjects and objects are arranged into columns and rows, forming a matrix. The matrix then becomes a two-dimensional representation of different roles and components.

	Unauthenticated user	Authenticated user with read access	Authenticated user with upload access
view data	no	yes	yes
download data	no	yes	yes
upload new data	no	no	yes
edit data	no	no	no
delete data	no	no	no

Table M.1: Subject-Object Matrix

Use Case and Misuse Case

To identify how the application should function, as well as possible security threats, we created a use case diagram and a misuse case diagram. By using these we can more clearly define functional and security requirements, as these give insight into how the application should work. The use case can be seen in Figure M.1, and the misuse case in Figure M.2. The individual case descriptions for both can be found in the Appendix H use case document.

Use cases are intended to give a systematic representation of the intended application functionality. It does this by modeling the interaction between the users and the application, and how these actors can achieve their goals. By doing this you can more easily identify expected behavior and interaction with the application, which is useful for guiding the creation of requirements, and ensuring they are aligned with how the application should function.

Misuse cases are on the other hand intended to give a systematic representation of the application functionality which is not intended. It does this by modeling the interaction between threat actors and potential application vulnerabilities, which lets you depict how the application can be exploited. Based on these you can find and prioritize mitigation strategies and countermeasures. Having an overview over possible threats and countermeasures can then help guide the implementation of comprehensive security requirements.

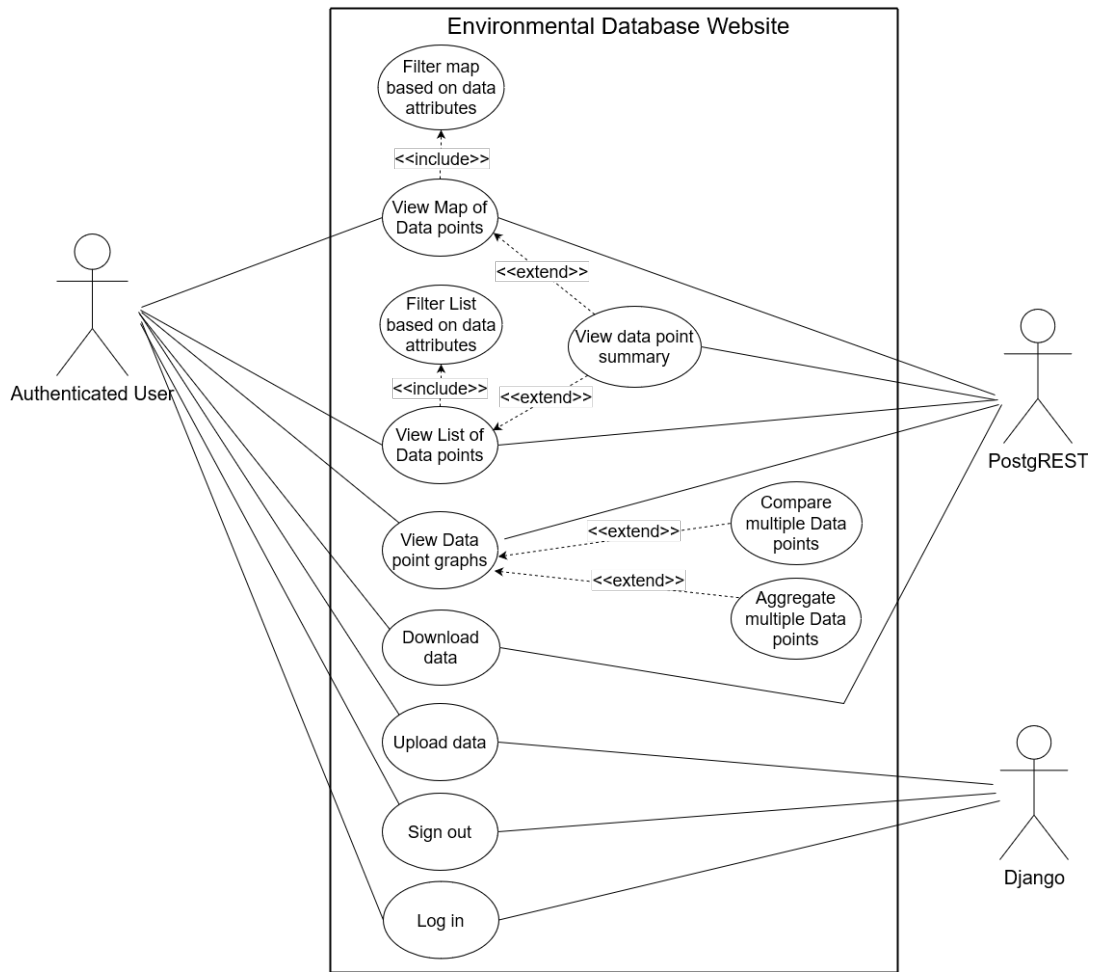


Figure M.1: Use case diagram

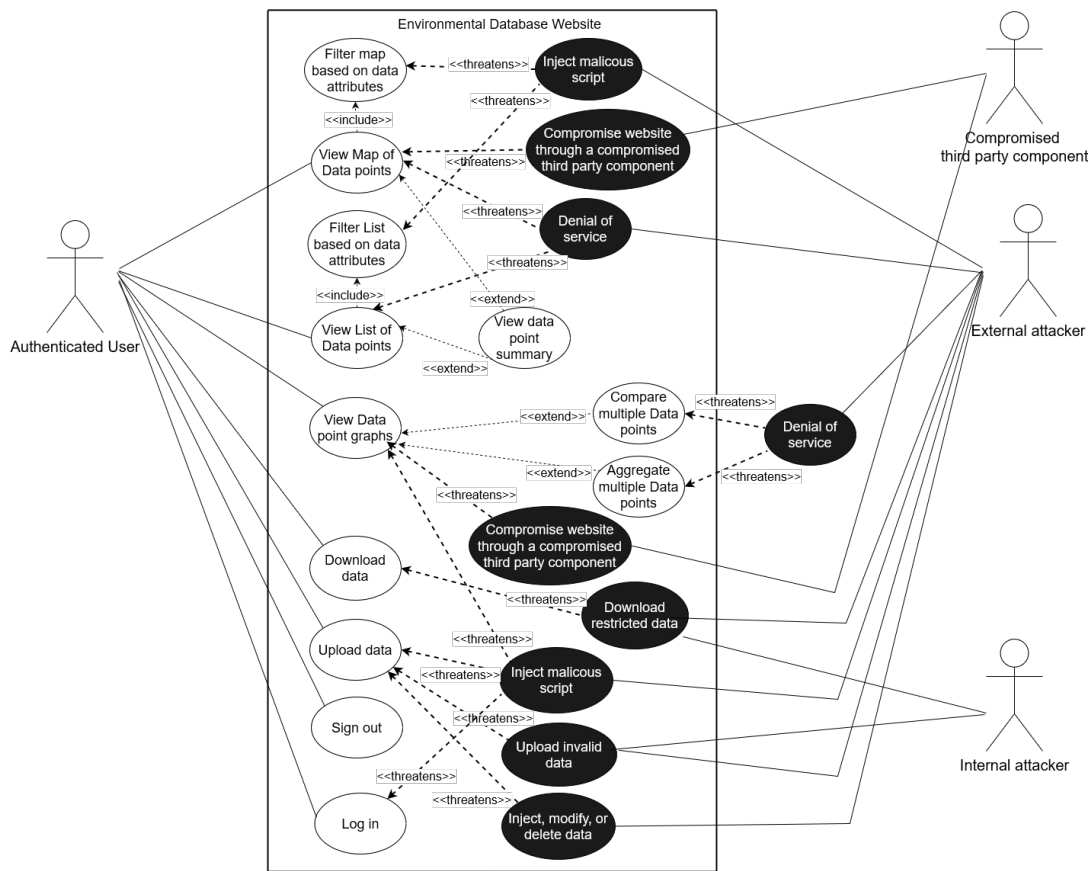


Figure M.2: Misuse case diagram

Requirements

The specification of requirements follow the process of requirement engineering.

The first step in the requirement phase is requirement gathering, with the overarching goal of understanding the business context of NINA and the end-users needs and requirements, for the project. By engaging in discussion with NINA, we clarified what the application should provide, which features the website should include, and how the website should function on a daily basis. This took place during the early phases of the project, where we asked NINA to provide us a prioritized list of their requirements, ranked based on their importance in alignment with their needs. After discussing and evaluating their preferences up against what we deemed possible to implement, we applied the updated requirements. *In the requirements gathering phase of the software development life cycle (SDLC), we are only required to identify which requirements are applicable to the business context and the software functionality serving that context. Details on how these requirements will be implemented are to be decided when the software is designed and developed* Paul [85, p. 100].

After gathering requirements in the previous step, we achieved an overview and understanding of the expected functionality. In the next step, requirement analysis, the requirements are further studied and analyzed. This implies refining and organizing the gathered requirements, making them easier to grasp and follow. Additionally, the requirements are clarified and prioritized. Prioritizing the requirements is about making sure to understand the importance of each requirement. Understanding the importance of the requirements makes it easy to focus on the most central features, and makes sure these features are given the highest priority and that they are implemented first. This means that we must delegate enough time for each requirement, based on their priority as early in the project as possible. Refining and organizing the requirements also allows us to easier communicate them with NINA, which in turn increases the probability that the application satisfies their needs, and that they are satisfied with the final product.

Functional Requirements

The functional requirements describes how the application is supposed to function in order to fulfill the needs of the stakeholders. These requirements explain in detail how the system should react under specific conditions and how interactions should function.

1. The application will allow users to view and interact with a map focused on Norway, however the map will include the whole world. This map contains points marking each observation registered in the back-end database.
 - a. The user should be able to filter the observations shown by selecting the date range, species, and the type of data (river/station level) of the observations they want shown. They should also be allowed to select multiple options simultaneously.
 - b. Observations on station level should be displayed as two points connected with a line, illustrating the path the NINA took when performing the observation. The lines should only be visible after zooming in a certain amount on the station data points to avoid clutter. Observations on river level should be displayed as points at the coordinates the observation was taken. Both river level points and station level points will have it's dedicated color to distinguish them from each other. River data points on the map will be blue, station data points will be red, and selected data points (river/station level) will be orange.
 - c. Importance: High
2. The application will allow users to view a list containing all river and station level observations registered in the back-end database.
 - a. The user should be able to filter the observations shown in the list by selecting the date range, species, and the type of data (river/station level) of the observations they want shown, as well as searching for observations based on the name and date.
 - b. Importance: Medium
3. When clicking on a river data point on the map or an entry in the list, the user should

be able to view tables containing a summary of its environmental data.

- a. The observation should include:
 - i. General information, which includes River name, Project number, and the Date of the observation
 - ii. More specific information, which includes the amount of stations, Crew, Boat type, Water flow, and Comments
 - iii. Information of the underlying data, which includes total amount of fish found, amount of minutes fished, and amount of fish fished per minute.
 - iv. A table containing all of the underlying attached stations, and for each station the table contains information about the river-types, weather, time spent fishing, the amount of fish and how many fished fish per minute.
 - v. A table containing the fish data of all underlying attached stations, and for each species the table contains information about the amount, amount per minute, average length, median length, and min and max length.
 - b. Importance: High
4. When clicking on a station data point the user should be able to view tables containing a summary of that station.
- a. The observation should include:
 - i. General information, which includes Station name, Project number, and the Date and time of the observation
 - ii. More specific information, which includes Crew, Boat type, River type, Water flow, Weather, Water temperature, Air temperature, Conductivity, Time spent fishing, Electricity settings, and Comments
 - iii. A table showing the amount of fish per species, minimum and maximum length, and amount per minute spent collecting.
 - b. Importance: High
5. The user should be able to download data as either a xlsx or csv file.
- a. The user should be able to chose the rivers or stations to download, the species to include, and if the data should be contained in a xlsx or csv file.
 - b. If rivers is chosen for download, the file should contain the aggregated data for the specific river points, meaning all the underlying station observations and their individual fish data as either a xlsx or csv file.
 - c. If stations is chosen for download, the file should contain the aggregated data for the specific station points, meaning all the station observations and their individual fish data, and its river as either a xlsx or csv file.
 - d. When a user has opened a river observation summary, they should be able to open the download page with the river selected.
 - e. When a user has opened a station observation summary, the user should be able to open the download page with the station selected.
 - f. When a user has selected rivers or stations on the graph page, the user should be able to open the download page with the same rivers or stations selected.

- g. Importance: High
- 6. The user should be able to select one or multiple river or station data points and view their data for more in depth analysis. The in depth analysis will contain graphs showing relevant data about the individual data under the rivers or stations.
 - a. The data will be visualized using graphs and diagrams.
 - b. The user should be able to select what they want to be plotted, includes which species to include in the graphs. The user can then choose to plot the distribution of species (optionally up against the time spent fishing), or the distribution of size for selected species (with adjustable intervals for size grouping, standard 5mm).
 - c. If multiple stations are selected, the user should be able to either aggregate the data to be shown together, or visualize and compare the differences between the stations.
 - d. Importance: Medium
- 7. Authenticated users will be able to upload data to NINA's backend database. The uploaded data must follow a specified xlsx format, and not contain any malicious content. The user will be informed of the result of the operation, and refreshing the page should allow the user to see the newly added changes if the operation was successful.
 - a. Importance: High
- 8. The user should be able to log in if they are not authorized by inputting their username and password. They can also log out by using a log out button.
 - a. Importance: High

Non-functional Requirements

The non-functional requirements are requirements not related to the functionality of the application itself, instead it is about specifications that describes how the system should behave. These requirements aren't ranked or organized. The reason for this, is that they together define the framework the project and application must follow, meaning all are equally important.

- The application must be capable of running in a docker container.
- The project must implement version pinning for all libraries, frameworks, and dependencies. This is done to ensure consistent and reproducible builds.
- The application will communicate with NINA's back-end database through a PostgREST API.
- The project must use libraries and programming languages which are open source. These languages and libraries must be actively maintained and be approved by credible sources as safe to use.
- All code has to be stored in GitHub. Necessary documentation of the application and dependencies must also be stored on GitHub.
- The GitHub project must have an MIT license. This allows others to use, modify or distribute the project, as long as they give credit to the creators, including the original copyright notice.

- Any dependencies must be clearly documented. This is essential for transparency, ease of maintenance, and collaboration, since it provides a clear understanding of the software's external components, which enables efficient troubleshooting, awareness of needed updates, or replacement of dangerous or outdated dependencies.
- The project must be finished before 21.05.2024. This means the final version of the application must also be finished before this deadline, however this does not restrict NINA from updating the application after the deadline if they need to incorporate new features.

Core Security Requirements

The core security requirements are measures added in order to protect a system from cyber-attacks, data breaches and unauthorized access. When defining core security requirements it's important that they ensure confidentiality, integrity and availability of the system and its corresponding data. "Properly and adequately defining and documenting security requirements, makes the measurement of security objectives or goals, once the software is ready for release or accepted for deployment, possible and easy" Paul [85, p. 98-99]. To simplify they define the expected level of security before applying appropriate security controls and mechanisms.

In the following List M, the core security requirements are illustrated. These are grouped in categories corresponding with the CIA triad (Confidentiality, Integrity and Availability) and is expanded to include other core requirements categories, including Authentication, Authorization and Accountability [87]. There are also security requirements not categorized, and some which falls on NINA's responsibility. These are clearly separated in their own categories.

1. Confidentiality:

- Encryption Standards:** Utilize HTTPS for internet connections in order to protect data confidentiality. To achieve this the application should be compatible with a NINA proxy handling HTTPS and SSL certificates.
- Session Token Security:** Handle session tokens securely by using cookies with the secure, HttpOnly, and samesite=strict attributes.
- Error Handling:** Implement proper error handling by handling all errors explicitly, displaying error messages in a non-verbose nature to the user. Should apply the principle fail securely, where the application always defaults to a secure state.
- Penetration testing:** Conducting penetration testing to identify vulnerabilities in a system, including the web application and servers.

2. Integrity:

- Data Sanitation and Validation:** Ensure that all data imported into the PostgreSQL is sanitized and validated against the valid data format, using a whitelisting approach maintaining data integrity by omitting data that fails to meet this criteria.
- Automated Updates:** Implement automatic updates for components to avoid legacy versions and preventing security faults and vulnerabilities.
- Secure Supply Chain:** Ensure that all dependencies are open source and from

trusted sources to maintain a secure supply chain. Set up automatic vulnerability scans of dependencies.

- d. **Injection prevention:** Apply techniques to prevent injection, where data format, data types, and content are verified. Content is checked by only allowing whitelisted characters.
 - e. **CodeQL workflows:** Apply CodeQL workflow to detect security weaknesses in code. Using CodeQL continuously can proactively analyze the code for security flaws, thereby maintaining the integrity and reducing the risk of security breaches.
 - f. **Continuous Testing:** Apply continuous integration, continuous deployment and testing into GitHub action workflow files, to ensure that the codebase development process includes mechanisms that maintains the integrity of the system.
 - g. **Explicitly reviewing copyright and licenses:** Documenting any third-party components and reviewing if we are obliged to use the resources can reduce the probability of misuse, ensuring the accuracy, reliability and trustworthiness of the system and its data. This will contribute to enhancing the integrity of the system.
3. Availability:
 - a. **Performance on client side:** Write code using efficient functions where possible to increase the responsiveness on the website, enhancing its availability.
 - b. **Caching of Requests:** Cache data retrieved from the database on the client side to reduce the amount of data requests.
 4. Authentication
 - a. **User Authenticated for accessing environmental data:** Accessing environmental data, downloading data, or uploading data should only be allowed by authenticated users.
 5. Authorization:
 - a. **Least Privilege Access Control:** Implement and apply the principle of least privilege for all system and user operations. Users should only be allowed to read and append environmental data, not modify or delete it.
 6. Accountability
 - a. All requirements regarding accountability and non-repudiation will be administered by NINA.
 7. Others
 - a. **Oblige Regulations:** Researching relevant regulations for our web application, and then applying them, would contribute to maintain security and its also closely related to ensuring integrity and availability.
 - b. **Complete Mediation:** Apply the principle of complete mediation. The principle ensures that every request to access is subject to explicit authorization and validation. This leaves no opportunities for abuse or unauthorized access.
 8. NINA's responsibility
 - a. **Backup:** create an isolated backup of the entire environment to ensure its availab-

ility even when the data is modified or deleted.

- b. **HTTPS:** Configure a proxy which handles HTTPS and SSL certificates for incoming requests.
- c. **Logging and Monitoring:** Log attempts to export and import data to ensure accountability, supporting the principle of non-repudiation.
- d. **Uptime Requirement:** The maximum tolerated downtime (MTD) and Recovery Time Objective (RTO) must be decided in the service level agreement (SLA) to ensure service availability.
- e. **Performance and Reliability:** Ensure the application can load quickly by allocating sufficient server resources for the web server, and if required apply load balancing like Haproxy to enhance the availability .
- f. **Notification on login:** When a user signs in from a new computer, the legitimate user should be notified on the mail address connected to their account to prevent malicious users being undiscovered. Implementing this will enhance the confidentiality.

Appendix N

Risk Assessment

1 Risk assessment

The process we followed when conducting a risk assessment of the software was the ISO27008 standard. The risk assessment was conducted as part of the Microsoft Security Development Lifecycle, described in detail in the methodology (Section 2.5 of the thesis). The process establishes the external and internal context, assesses the risks and treats the risks using a risk treatment plan to implement the recommendations and decisions. Risk management analyses what can happen and what the possible consequences can be, before deciding what should be done and when, to reduce the risk to an acceptable level.

1.1 Context establishment

1.1.1 Internal context

This risk assessment will consider the development of the web application "Interactive Database for Environmental Data" for NINA. The application uses environmental data which NINA also use to do their research. This data has to be correct, as the data will impact their research. It is also important that sessions are handled correctly as the website will share authentication mechanisms with other systems at NINA.

1.1.2 Scope

The risk assessment will mainly focus on the development of the web application. The deployment, monitoring, and response management of the application is partly out of scope, as NINA will be responsible for this aspect of the application. Still, relevant countermeasures will be included, which NINA may choose to implement.

1.1.3 Risk Evaluation Criteria

The table 1 illustrates the different classifications of assets for NINA which are relevant to the application. These are used to categorize the prioritization of the assets.

Level	Confidentiality	Integrity	Availability
1	Public	Not required	Not important
2	Internal	Expected	2 Days
3	Confidential	Necessary	4 Hours
4	Strictly Confidential	Critical	Immediately

Table 1: Table showing the risk evaluation criteria

1.1.4 Impact criteria

The following are the different levels of consequences and probabilities for risks identified, which will be used to rank and prioritize them.

The probability are divided into four tiers: unlikely, low, medium and high. Similarly, the consequences are also divided into four tiers: negligible, slight, serious and very serious. The classification provides a structured framework that increases the understanding of potential threats and their potential magnitude. This is useful for creating an efficient risk management and mitigation plan. The table also has color codes that identifies the risk. These color codes represents five tiers: Blue - Very Low risk, Green - Low risk, Yellow - Medium risk, Orange - High risk, Red - Very high risk. However, its important to note that this is only a rough estimation of the expected risk.

Risk Level		Consequences			
		Negligible	Slight	Serious	Very Serious
4*Likelihood	Unlikely	1	2	3	4
	Low	2	4	6	8
	Medium	3	6	9	12
	High	4	8	12	16

Table 2: Table showing risk tiers

1.1.5 Risk acceptance

In general the risks will be acceptable when they are at a low risk level. However exceptions may happen if decided by NINA, which has the final say. Factors such as time and cost to implement the countermeasures are important to consider.

1.1.6 External context

Copyrights laws must be accounted for, where it is important to properly make sure the original creator is credited where relevant. The application should conform to the web accessibility guidelines from WCAG 2.1. These help improve the experience of web content for people with disabilities. Laws regarding cookies and processing of personal data are very strict, but does not need to be considered as the application will not collect or process personal data. There are also laws regarding registering a domain, however this is not relevant for the scope of the risk assessment.

1.2 Asset identification

The second step when conducting a risk assessment is to identify the assets for the web application. The assets discovered can then quickly be protected against possible threats. By identifying these assets early in the project phase, we can also conduct vulnerability scans, that can help mitigate some potential risk. In the table below, there are five identified assets for the web application. These assets have been given a score based on the CIA-triad, confidentiality, integrity and availability. The confidentiality is divided into four tiers; public, internal, confidential and strictly confidential. Similarly the integrity and availability is also divided into four tiers each. For integrity the tiers are as follows; no demand, expected, dependent and critical. Lastly, for the availability we have the tiers; no demand, two days, four hours and immediate.

Asset	Description	Confidentiality	Integrity	Availability
Environmental Data	Data collected by NINA (River-data, station-data) that is used to plot the graphs.	Confidential	Critical	2 days
Web server and website source code	The web server the handles requests, and the website which is delivered to the client.	Open	Critical	2 days
User Sessions	The sessions which authenticates the users.	Strictly confidential	Expected	2 days

Table 3: Identified Assets

1.3 Vulnerabilities

This section is an overview over possible vulnerabilities for our application. Several of these are common vulnerabilities found in many applications, based on OWASP Top ten. These are not precise descriptions, but rather they describe high level vulnerabilities which cover multiple implementation bugs and design flaws. There are also more specific vulnerabilities identified for the specific application. The vulnerabilities we found to be most relevant is shown in Table 4. The vulnerabilities are used to construct possible risks for the application.

Id	Vulnerability	Description
1	Insecure session handling	Refers to the poor protection and management of user session data, which could allow unauthorized access or manipulation of user accounts and sensitive information. Based on <i>A07:2021 – Identification and Authentication Failures</i> from OWASP Top 10
2	Vulnerable and outdated components	Represent software components such as libraries, frameworks, or plugins that have security flaws or lack updates. This poses a risk to the overall security of a website as these components are integrated into the website. This components could also be changed by malicious owners to perform attacks like session hijacking or key-logging. This vulnerability is more likely if dependency pinning is used, as security patches for outdated components will not be automatically applied. Based on <i>A06:2021-Vulnerable and Outdated Components</i> from OWASP Top 10
3	Data Integrity Failure	Refers to situations where the website’s environmental data is compromised, leading to potential errors, data corruption or unauthorized modifications, for example if the environmental data contains malicious code. Based on <i>A08:2021 – Software and Data Integrity Failures</i> from OWASP Top 10
4	Injection	Injection attacks represents exploiting vulnerabilities by injecting malicious commands or code into input fields to manipulate or compromise the website’s database or to execute unauthorized actions. Injection could be used to manipulate the postgRest queries, hijack session tokes or inject malicous code/scripts into NINA’s database. Based on <i>A03:2021 – Injection</i> from OWASP Top 10
5	Security misconfiguration	Involves improper implementation or setup of security measures, such as having exposed ports, not removing default accounts, not configuring security headers for the server, and insecure database configuration. Based on <i>A05:2021 – Security Misconfiguration</i> from OWASP Top 10
6	Insecure design	Refers websites with inherent architectural flaws or poor security choices, leaving the website with multiple vulnerabilities which can be exploited by threat actors. Based on <i>A04:2021 – Insecure Design</i> from OWASP Top 10
7	Sending data to the client	This is a potential vulnerability as the application originally allowed unauthenticated user to view the data being sent to their machine, even though they should not able to download it directly from the website.
8	No rate-limiting for login	As there is no rate limit or captacha for login a threat actor can brute force credentials. Based on <i>A07:2021 – Identification and Authentication Failures</i> from OWASP Top 10

Table 4: Table showing relevant vulnerabilities

1.4 Risk identification

This part uses the identified assets and vulnerabilities to construct relevant risk scenarios. These will consist of threats which exploit vulnerabilities to cause harm to the assets identified.

Id	Risks	Description	Vulnerability	Assets
1	Session hijacking	Session hijacking by cross site scripting, brute forcing credentials, or another vulnerability, allowing an unauthorized person to steal a legitimate and authenticated user's active session, which can lead to data theft, data modification, or impersonation.	Improper session handling, Injection, Insecure design, No rate-limiting for login	User Sessions
2	Compromised website	Attackers can compromise website by vulnerable third party components, or using cross site scripting, which can be enabled by vulnerable code and lack of integrity checks. This would result in the website being compromised, which could break the website, steal the clients credentials or other information, or infecting the clients web browser with malware.	Vulnerable and outdated components, Data Integrity Failure, Injection, Insecure design	Web server and website
3	Disruptive Denial of Service (DDOS)	Attackers can take down the website by overloading it, which can be enabled by insecure design by not using load balancing and having inefficient use of resources, as well as misconfiguration. This can lead to loss of data and website availability.	Security Misconfiguration, Insecure design	Web server and website, Environmental Data
4	Environmental data is deleted or modified	An attacker tries to modify data, inject false data, or destroy data in the web application by utilizing injection or other vulnerabilities in the code.	Injection, Vulnerable and outdated components, Insecure session handling, Security misconfiguration	Environmental Data
5	Copyright is not handled correctly	That the copyright or license of a third party component is not followed, which can result in legal action and economic consequences.	Lack of research and implementation of requirements surrounding components used in the application.	Economic
6	Regulations are not followed	That regulations are not followed, which can result in legal action and economic consequences.	Lack of research and implementation of regulations which apply to the application.	Economic
7	Environmental data is stolen	That the environmental data is stolen and made public, which can be done by being an insider with authentication, stealing a session, or simply by reading the data being sent to the frontend in the console and network monitor.	Improper session handling, Insecure design, Sending data to client	Environmental Data

Table 5: Table showing risk scenario

1.5 Risk Analysis

In Table 6, various risks associated with the project are analyzed. The risk is calculated based on the probability of the event occurring and the potential consequences it may entail, before any security measures have been applied. This is to give an indication of how important it is that the various risks are considered under development. (The probability and consequences are further divided into tiers, as illustrated in the accompanying risk level Table 2).

Session hijacking has a high probability, as it can be of interest for potential attackers by allowing them access to any resources the user can access, and it is quite easy to do when no security measures are in place. Nina's assets are not that valuable, as it is mostly research data, as well as the website being hosted in Nina's internal network, but we still set the probability to high. The consequence is very serious because of all the resources and access you get from a user. This can be used for financial gain, espionage, or to sabotage NINA because of ideological reasons.

Compromised website has a medium probability, because even though the assets themselves hold little value beyond NINA, a compromised website can be used to sabotage NINA for ideological reasons or financial gain by infecting clients with ransomware. In addition it would be easy to do cross site scripting without any measures in place. The website can also get compromised by insecure or compromised third party components, which

also increase the probability. The consequence is very serious, as clients can get credentials stolen, lose access to the website, and get invalid data.

Denial of service has a low probability, as it is easy to do but not much is gained from it other than delaying research. The consequence is slight, as it will only mildly inconvenience NINA which would have to delay their research.

Environmental data being deleted or modified has a medium probability, as it is possible with no measures in place. The motivations can financial gain by using their data as ransom, or ideological by preventing their environmental research. It could result in invalid research, and loss of over a decade of environmental data which is why the consequence is very serious.

The probability of copyright and licenses not being handled correctly is low, as this is simple task which is easy to do. Several libraries even adds copyright watermarks automatically, giving credit to their work. The consequence however is serious, as it could cost money and result in the website temporarily being taken down.

The probability of regulations not being followed is medium, as this is easy to not do correctly if it is not properly taken into consideration early on in the development. The consequence is medium, as it could result in fines and potential legal action against NINA.

Environmental data is stolen has a high probability, as it is easy to do, either by an attacker by simply reading the data being sent before it is shown in figures, or by an insider. The consequence is serious, as the data is the result of decades of manual collection, which NINA as a private foundation considers valuable. Losing the data to competitors would therefore be undesirable.

Nr	Risk Scenarios	Probability	Consequences	Risk
1	Session hijacking	High	Very serious	Very High
2	Compromised website	Medium	Very serious	High
3	Denial of Service (DDOS)	Low	Slight	Low
4	Environmental data is deleted or modified	Medium	Very Serious	High
5	Copyright is not handled correctly	Low	Serious	Medium
6	Regulations are not followed	Medium	Serious	High
7	Environmental data is stolen	High	Serious	High

Table 6: Table showing risk scenarios

Risk Management Plan

The following are security measures we will use under development to reduce the likelihood and consequences of the risks identified. These countermeasures are organized into a risk management plan, detailed in the Tables 7 and 8. Each mitigation refers to previously identified risks, outlining specific mitigation strategies and countermeasures with the overarching goal of reducing the total risk and make the risk acceptable for NINA. Table 1.5 shows the estimated remaining risk for each risk event after the countermeasures from the risk management plan has been implemented.

Id	Counter-measure	Description	Risks mitigated
1	Proper error handling	Handling errors with best practice. This implies utilizing try-catch blocks, logging error records, and implementing custom error classes and centralizing error handling middle-ware. Additionally, limiting error information shown to end users. These practices collectively improves the application's stability, as well as limiting what users can learn from the application through errors.	1, 2, 3, 4, 7
2	Input validation	Using Input validation on user input, files uploaded, and on environmental data from the server. This will make it harder to inject anything malicious into the application.	1, 2, 4, 7
3	Output sanitization	Sanitizing all output in the application, ensuring that it can not be interpreted as anything executable. This will reduce the consequences if any malicious data gets into the application.	1, 2
4	Content Security Policy	Configuring the Content Security Policy to only allow specific required resources to be loaded to the application, reducing the chances that malicious code or resources can be loaded in the application	1, 2, 7
5	Secure Design Principles	By following Secure design principles we can ensure that the development considers security best practices already in the design phase. These principles include Least privilege, Separation of duties, Defense in depth, Fail Securely, Complete mediation, Open design, among others.	1, 2, 3, 4, 7
6	Handling sessions according to best practices	Using best practices for handling sessions, such as Samesite cookie, Secure cookie, and HTTPOnly cookie, as well as using proper encryption and setting lifetime of tokens. This will make it harder to compromise and steal someones session.	1
7	Pentesting	Pentesting the web application and the web server using tools like Hydra, Burp Suite, nmap any other online resources is essential. Conducting pentesting on the website can enhance it's resilience and robustness against incoming attacks. Techniques that are common when testing are as follows: brute forcing, SQL injection, session cookie manipulation, XSS hacking, URL manipulation and HTML manipulation.	1, 2, 3, 4, 7
8	Continuous Testing	By continuously doing testing during the development, both with automation in the forms of Unit, API, Security and integration testing, as well as more manual tests such as end-to-end testing, the testing will ensure the application functions as intended.	1, 2, 3, 4, 7

Table 7: Table showing risk management plan for countermeasures 1-8

Id	Counter-measure	Description	Risks mitigated
9	Dependency management	Prevent vulnerabilities in third-party dependencies. This can be done by actively updating dependencies to ensure the latest security patch, and reduce the risk of using outdated packages. This process can in some cases be automated, where the developers can be notified of any possible vulnerabilities in the dependencies. The automation also make sure to test the code for vulnerabilities before patching, notifying developers if there are any security problems with the dependencies, which helps ensure a secure supply chain. Even though NINA want to use dependency pinning, they should still regularly check for security vulnerabilities in the dependencies.	1, 2, 4, 7
10	Secure configuration and deployment	Configuring security headers and disabling server tokens in Nginx. Configuring HTTPS, firewalls, and other deployment configuration securely.	1, 2, 3, 4, 7
11	Rate limiting requests	Rate limiting requests to the server, especially for authentication. Prevents brute forcing of credentials, and DDOS attacks.	1, 3
12	Logging	Logging can help with troubleshooting and debugging by providing valuable insight into the application behavior. Additionally, it can notify you when issues arise. It can also ensure non-repudiation when it comes to downloading and uploading of data	3, 4, 7
13	Backup	Having an isolated backup of the environmental data will ensure its availability even if the data is modified or deleted.	4
14	Explicitly reviewing copyright and licenses	By documenting any third party component, finding out how we are allowed to use it, we can reduce the probability of misuse.	5
15	Researching and following all applicable regulations.	By researching which regulations apply to our website, and then ensure we follow them when developing the application, we can mitigate the risk of any fines or legal action.	6

Table 8: Table showing risk management plan for countermeasures 9-15

Remaining Risk

In Table 9 every risk identified in the risk identification (Section 1.4) is associated with fitting countermeasures discussed in the Risk management (Section 1.5). We estimate how much impact the countermeasure contribute with reducing the initial risk we found in the risk analysis (Section 1.5).

Based on all the countermeasures we plan to implement, we are estimating that the remaining risk for 'Session hijacking', 'Compromised website', 'Denial of Service', 'Environmental data is deleted or modified', and 'Environmental data is stolen' to be very low. All of these risks will be mitigated by implementing best practices in the development process by doing testing and ensuring a secure supply chain. Implementing secure design, developing countermeasures, such as input validation and output sanitization, in addition to using a secure configuration will all further reduce the risk, both by making it harder to exploit vulnerabilities, as well as reducing the consequences if they are exploited.

Taking backups and logging access to data, secure configuration, and rate limiting, are all NINA's responsibility. The remaining risk is based on if they choose to implement these countermeasures, where the risk will be higher if they do not implement them.

Handling copyright and regulations are set to Very Low and Low respectively, as they are easy to deal with when explicitly handled. The reason we are setting handling regulations to low instead of very low is because of the nuances in regulation and law, which may be harder to follow exactly to specification than licenses and copyright.

Risk ID	Risk	Countermeasures	Initial risk	Remaining risk
1	Session hijacking	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	High	Very Low
2	Compromised website	1, 2, 3, 5, 7, 8, 9, 10	High	Very Low
3	Denial of Service (DDOS)	1, 5, 7, 8, 10, 11, 12	Low	Very Low
4	Environmental data is deleted or modified	1, 2, 5, 7, 8, 9, 10, 12, 13	High	Very Low
5	Copyright is not handled correctly	10	Medium	Very Low
6	Regulations are not followed	11	High	Low
7	Environmental data is stolen	1, 2, 4, 5, 7, 8, 9, 10, 12	High	Very Low

Table 9: Table showing remaining risk

1.6 Acceptable risk threshold

ISO27005 describes this phase as the decision being made to accept the risks, and mentioning the responsibilities of the decision makers. To bring the risk threshold to an acceptable level we are going to implement all of the identified countermeasures mentioned in Section 1.5, when developing the application. By doing this all of the residual risk identified as the remaining risk (Section 1.5) are under the acceptable levels NINA has given. It is worth noting that this is a continuously evolving risk assessment, where vulnerabilities, risks, and countermeasures are added and modified as the risk evolves throughout the application development.

Appendix O

Code Implementation

Code Structure

When developing software it is important to consider the structure of the code. By setting up the file structure in a modular fashion, you can ensure Separation of Concern. This helps the maintainability of your code, as you can easily change one part without affecting the rest, and multiple developers can work on different modules simultaneously. Moreover, following Separation of Concern makes the code reusable, as code written in one module can be used multiple times across the project, independent of how it originally was implemented. The following is how the application's code is structured, which tries to follow separation of concerns.

Svelte and SvelteKit Structure

As the application uses SvelteKit, the application follows the project structure outlined by the official SvelteKit documentation ¹. The project structure (see Figure O.1) includes a folder for source code, called `src/`, a folder for static assets, called `static/`, and a folder for end to end tests, called `tests/`. Configuration files is placed in the root of the SvelteKit folder. Furthermore, under `src/`, the application is further split into the folder `lib/`, for Svelte components, and `routes/`, for SvelteKit pages, with `app.html` being the page template. This structure follows separation of concern, where different parts of the application are clearly separated.

JavaScript Modules

The application further splits the code into different JavaScript modules. Each of these modules has a single responsibility, ensuring high cohesion, where the modules are organized into different directories under `src/` (see Figure O.2). There are multiple high level categories of modules. `api/` are for modules interacting with external components and services, `constants/` are for hard-coded values which are used across the application, `models/` are for defining data structure and classes, `stores/` are for declaration of svelte data stores, and `utils/` are for all other miscellaneous modules.

¹<https://kit.svelte.dev/docs/project-structure>

By structuring the code this way the `plotlyData` module can call a function from `api/postgres`, which again can retrieve a value from `constants/endpoints`, where each module does not depend on the different implementations. This follows loose coupling), where each module does not care about how the others are implemented.

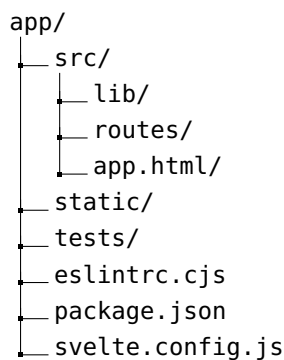


Figure O.1: Svelte and SvelteKit File Structure

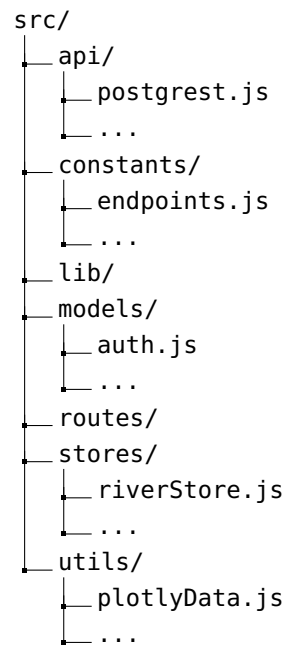


Figure O.2: JavaScript Modules File Structure

Svelte Components

The application further splits up the code into different Svelte components². These are the building block of a Svelte application, where each component is its own `.svelte` file. Each component contains HTML, CSS, and JavaScript which only applies to the specific component. This follows separation of concern, where each component only has code directly related to it, where it can be used other places without having to consider its implementation.

An example component from the application is `DateInput.svelte` (see Listing O.1). This component is only responsible for an input field where a user can select a start and end date. It has JavaScript which simply lets other components read and set the dates selected, HTML for basic date inputs, and some simple styling. Components can grow to be larger, and they can use other components themselves, however they always only contain relevant code to the scope of the component.

```
<script>
  export let selectedStartDate
  export let selectedEndDate
```

²<https://svelte.dev/docs/svelte-components>

```

</script>

<!-- Input for choosing start date -->
<label for='startDate'>
  Fra
  <input id='startDate' type='date' name='startDate' bind:value={
    selectedStartDate}/>
</label>
<!-- Input for choosing end date -->
<label for='endDate'>
  Til
  <input id='endDate' type='date' name='endDate' bind:value={selectedEndDate}/>
</label>

<style>
  label {
    display: block;
    padding: 0.5em;
    font-size: 1.2rem;
  }
</style>

```

Listing O.1: DateInput Component

To set the values of variables in a component, and getting the updates values if the component modifies them, Svelte uses props. A component can specify a prop by using the syntax `export let variableName`, which the component can both read from and update. When another component uses the component, they can either set the value of the prop with the syntax `propName={value}`, or if the value is a variable with the same name as the prop, `{value}`. The parent component or page can also listen to any updates of the prop, and update the variable they sent, by using the syntax `bind:propName={variable}`.

Svelte also supports the use of event dispatchers, which allows components to create events, which can then be handled by the parent component with the syntax `on:eventname={handleFunction}`. An example of a component being used with props and event handlers can be seen in Listing O.2.

```

<Button on:buttonClick={handleSelectRiverStation} type='blue' size='small'>
  Rediger
</Button>

```

Listing O.2: Use of button component with props and eventhandler

Retrieval of data

As the application's main purpose is to display environmental data, retrieving this data in an efficient way is essential. The environmental data grows continuously as the scientists collect

more, and there are already several hundred rivers, thousands of stations, and hundred of thousands of observations, which can grow ten fold. Because of this the application is designed to use client-side hydration, where the data for the application is only fetched at the client-side. In addition the application leverages lazy-loading, where only the data needed is fetched and loaded at any given time. The following is how the application retrieves and stores this data.

PostgREST API

PostgREST is used as an API which exposes the environmental data the application is allowed to use. To configure the endpoints PostgREST expose you can create SQL Views in PostgreSQL. These are used to create custom endpoints which contain the exact data needed for different purposes in the application.

SQL Views

There are five SQL views used by the application. Two of them are used to fetch all rivers and all stations, called `river_with_species` and `station_with_species` respectively. As all rivers and stations are fetched, they contain the minimal amount of information needed when displaying all of them. This includes their names, ids, coordinates, date, project id, and the species which were observed in them. Including species and dates allows the user to filter based on these attributes. Coordinates are used to plot each station and river on the map.

The other three SQL Views are only fetched when a specific river or station is selected. One retrieves all information needed to show the summary of a river, called `river_summary`. Another contains the information needed to show the summary of a station, called `station_summary`. The final one is used to fetch the data needed for a station to be downloaded, called `station_download`. The station summary and download SQL Views both contain the observations under the stations, where each observation includes information such as its length, species, and count.

Figure O.3 shows the configuration for SQL View `river_with_species`. All the attributes needed are defined in the select statement, where the PostgreSQL functions `jsonb_agg` and **DISTINCT** are used to get all distinct species for a station and aggregate these into a JSON array. The `COALESCE` function with `FILTER` and `[]` ensures that if there are no species found, an empty JSON array is returned instead of **NULL**. To find the species of the observations under a river, the river table has to join with the `stasjonsdata` and `individdata` tables, where the stations and observations are grouped by and linked to their river.

```
CREATE VIEW "river_with_species" AS
SELECT
  elvedata.id,
  elvedata.elv AS name,
  elvedata.posisjon AS pos,
  lower(elvedata.dato) :: date AS start_date,
  (upper(elvedata.dato) - '1_day'::interval) :: date AS end_date,
  COALESCE(
    jsonb_agg(DISTINCT individdata.art)
```

```

    FILTER (WHERE individdata.art IS NOT NULL), '[]') AS species,
    elvedata.prosjektnummer AS project_id
FROM elvedata
JOIN stasjonsdata ON elvedata.id = stasjonsdata.elvedata
LEFT JOIN individdata ON stasjonsdata.id = individdata.stasjon
GROUP BY elvedata.id;

```

Listing O.3: River_with_species SQL View

Endpoints

Accessing the SQL views is done by sending a HTTP GET request with the correct path. Moreover, PostgREST also supports the filtering of data in endpoints ³. For example, if you want to access `station_summary` for a stations with ID 1 and 2, the query parameter `id` can be set as follows: `id=in.(1, 2)` (see Listing O.4). These endpoints return the data as JSON.

```
GET /station_summary?id=in.<id1>, <id2>, ...)
```

Listing O.4: Request syntax for `station_summary` endpoint

Fetching PostgREST with PostgREST module

To fetch the data from PostgREST the application uses functions defined in the `postgrest` module. It has a function for each endpoint/SQL View, where the summary and download endpoints take the `id` or `ids` of the river or stations to fetch. By calling these, for example `fetchRiverSummary(id)`, a HTTP GET request is sent to the correct endpoint with the correct query parameter.

The code which creates the fetch request is illustrated in Listing O.5. It uses `POSTGREST_URL` from an environment variable, along with the endpoint which is different based on the type of data to request. The code specifies the use of the GET method, and that all cookies for the domain should be included in the request if the URL of the website has the same origin as the request. The origin is the same if the protocol, domain, and port number is the same ⁴. The response from the endpoint is handled by checking the status codes, where 401 (Not authenticated) is handled by the `auth` module, and if the status is not ok (not in the 200 range) it throws an error and displays it to the user. If the status is 204 no content, `NULL` is returned, and if everything worked the JSON data is parsed and then returned.

```

const response = await fetch(`${POSTGREST_URL}${endpoint}`, {
  method: 'GET',
  credentials: 'same-origin'
})

```

Listing O.5: Fetch request to PostgREST endpoint

³https://postgrest.org/en/v12/references/api/tables_views.html

⁴<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin>

The DataManager module, caching, and updating the stores

The data manager is called from the components and pages to ensure that the data for rivers and stations is available. It has functions for each endpoint, where the functions first check if the data already exists, then fetches the data using the postgres module, validates it, and updates the storage in the application with the new data. The pages in the application can use these functions when they need specific data. For example, when the map page loads, it needs all rivers and species for displaying them in the map. To accomplish this the map page simply have to call the `getRivers()` and `getStations()` functions, allowing the map page to then find these in the global storage. Another example is when a user want to download a station. The download page calls `getStationForDownload(id)` for a specific station chosen, which ensures the page that the station data required for download exists in the global storage.

To check if the data already exists the module `checkIfDataExists` is called. It has several functions to check if each endpoint needs to be fetched, for example by checking if a specific river has the needed data to display its summary. This is a form of caching, which improved performance by only needing to fetch specific data once. However, this data is gone once the website is reloaded, which means reloading the page requires the page to fetch the data again. This does somewhat reduce load times, however, this guarantees that the user can view new data on the web page after they have uploaded it, which would not be guaranteed if the data was cached between page loads.

To validate the data, the `dataManager` module first checks if any data was received, and if so it uses the `validation` module to validate the JSON schema and content. Validation is further explained in section about data validation.

The function `getRiverSummary()` from the `dataManager` module can be seen in Listing O.6. It uses functions for checking if the rivers already exists, if not it tries to fetch their data, validate it, and update the store. It also catches any possible errors and displays an appropriate error message. The function ensures that all rivers from `river_with_species` is stored and accessible with no malicious data when the function is called.

```
/**
 * Ensures that all rivers are stored in the river store such that the
 * map and list page can display them
 * @returns {void}
 */
export async function getRivers () {
  if (doesAllRiversExistInStore()) {
    return
  }

  try {
    const fetchedRivers = await fetchRivers()

    // Validate the fetched rivers
    if (!fetchedRivers || !validateRiverWithSpecies(fetchedRivers)) {
      return
    }
  }
}
```

```

    }

    updateStoreWithObjects(riverStore, fetchedRivers, River)

  } catch (error) {
    addFeedbackToStore(FEEDBACK_TYPES.ERROR,
      FEEDBACK_CODES.POSTGREST_UNAVAILABLE, FEEDBACK_MESSAGES.
        POSTGREST_UNAVAILABLE)
  }
}

```

Listing O.6: `getRiverSummary()` function in `datamanager` module

Storing retrieved data is done by updating the Svelte river or station stores. The `dataManager` module has functions for updating one or multiple objects, called `updateStoreWithObject` and `updateStoreWithObjects` respectively. The `updateStoreWithObjects` function takes in the store to update, the objects to update with, and the type of the objects (see Listing O.7). If the store is empty it sets it to the retrieved objects. However, if not, the function iterates through each object, simply setting their value if they do not exist, and if they already exists it merges the old and new object. It does this by using *object spread syntax*⁵, where the old and new object is merged to a new one with both properties, where the new object overwrites any properties if they are both have them. This results in the function updating the store with objects, where there is no problem if some of the objects already exists in the store with partial of its information already fetched.

```

/**
 * Updates a store with given objects converted to a given class
 * Ensures that if there is any overlap between new and existing objects,
 * that the properties of the new object overwrite the old one, while the
 * old properties are kept
 * @param {import('svelte/store').Writable} store - The store to update
 * @param {Array<object>} objects - The objects to update the store with
 * @param {Function} Class - The class to convert the objects to
 * @returns {void}
 */
function updateStoreWithObjects (store, objects, Class) {
  // If the store is empty, simply set the store with the objects converted to
  the class
  if (get(store).size === 0) {
    const objectMap = new Map(objects.map(object => [object.id, Class.fromJson(
      object)]))
    store.set(objectMap)
    return
  }
}

```

⁵https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax

```

// If the store is not empty, update the store with the objects
store.update(currentMap => {
  objects.forEach(newObject => {
    // If the object does not exist in the store, add it
    if (!currentMap.has(newObject.id)) {
      currentMap.set(newObject.id, Class.fromJson(newObject))
      return
    }

    const existingObject = currentMap.get(newObject.id)
    // Get the new object from json and remove any null values to avoid
    // overwriting existing values
    const newClassObject = Class.fromJson(newObject)
    const newObjectFiltered = Object.fromEntries(
      Object.entries(newClassObject).filter(([_ , value]) => value !== null))

    // Merge the new object with the existing object
    const updatedObject = new Class({
      ...existingObject,
      ...newObjectFiltered
    })

    // Update the map with the updated object
    currentMap.set(newObject.id, updatedObject)
  })

  return currentMap
})
}

```

Listing O.7: updateStoreWithObjects() function in datamanager module

Handling data

Environment data retrieved from PostgREST is used by each of the application's pages for different purposes. The raw data is not displayed directly, but rather filtered based on user inputs, such as by date, species, name, and datatype, and formatted for different purposes, such as display in a table or for inserting into an Excel file. Furthermore, the data is processed from raw observations into useful data for the various pages in the application, such as distribution of fish length, median length for a specific species in a station, minutes spent fishing, and more. The section covers how data is handled and processed by the application to achieve this.

Svelte Store and Data structure

Svelte Store

To make the data river and station data globally accessible the application uses Svelte stores. A store is a globally accessible object with a subscribe method which allows both Svelte components and JavaScript modules to read and get notified when the store is updated ⁶. Furthermore, the Stores can be writable, which allows the Svelte components and JavaScript modules to modify and update it.

The application has several stores, each in their own files. There is one for rivers, one for stations, one for the authentication status, and one for the feedback to display to the user. These are simply called `riverStore`, `stationStore`, `authStore`, and `userFeedbackStore` respectively. These are initialized as empty data structures, with the use of the `writable` function to make them modifiable (see Listing O.8) .

```
import \{ writable \} from 'svelte/store'  
  
export const riverStore = writable(new Map())
```

Listing O.8: Initialization of `riverStore` in `riverStore.js`

To access the Svelte stores, the components uses the reactive `$store` syntax ⁷. This allows components to read the content of a store, where the variable it is assigned to is updated each time the store is (see Listing O.9). Under the hood the reactive `$store` syntax subscribes to the store, and automatically unsubscribes when appropriate, for example when the component is destroyed.

```
// Get rivers and stations from stores  
\$: rivers = \ $riverStore  
\$: stations = \ $stationStore
```

Listing O.9: Reading stores in a Svelte Component, where the rivers and stations variables will always contain the up to date content of the stores

To read the stores the JavaScript modules have to use the `get(store)` method, which simply subscribes to the store, reads the value, and then immediately unsubscribes. To update the store the modules either uses `store.set(content)` to overwrite the content, or `store.update(currentContent => {newContent})` to update it. The update method takes a function which creates the new content based on the current content. The Listing O.10 shows how `get()` and `store.set()` can be used to get the content and update a store.

```
// If the store is empty, simply set the store with the objects converted to  
the class  
if (get(store).size === 0) \{  
  const objectMap = new Map(objects.map(object => [object.id, Class.fromJson(  
    object])))
```

⁶<https://learn.svelte.dev/tutorial/writable-stores>

⁷[https://svelte.dev/docs/svelte-components#script-4-prefix-stores-with-\\$-to-access-their-values](https://svelte.dev/docs/svelte-components#script-4-prefix-stores-with-$-to-access-their-values)

```
store.set(objectMap)
return
\}
```

Listing O.10: Retrieving and setting the value of a store in a JavaScript module

Data Structure

The data structure used in the river and station Svelte stores are simply JavaScript maps. The keys for the rivers and stations are their IDs and their value is River or Station objects. All id's are retrieved from PostgREST. This allows the application to retrieve the values of an object in the stores using `map.get(key)`, by for example retrieving a specific river as follows: `$riverStore.get(riverId)`.

The River and Station objects are defined in their own JavaScript files under `models/`. They both contain a JSDoc comment of their structure, documenting the data types and purposes of each of their properties. Both use a constructor to initialize the objects created from the classes, and any property not explicitly defined is set to null. The classes also contain a `fromJson()` method, which creates a River or Station object based on a JSON object. This is required as the JSON data uses `snake_case`, while the classes use `camelCase`. The Station class can be seen in Listing O.11, where it has a JSDoc comment, a constructor, and a `fromJson()` method.

```
/**
 * Represents a station
 * @class
 * @property {number} id - Unique id of station
 * @property {string} name - Name of station
 * @property {string} date - Date of station
 * @property {{coordinates: [number, number]}} startPos - Start position of
   station observation
 * @property {{coordinates: [number, number]}} endPos - End position of station
   observation
 * @property {string} time - Time of day
 * @property {number} riverId - Id of river the station is under
 * @property {string} description - Description
 * @property {string} comment - Comment
 * @property {string} riverType - Type of river
 * @property {string} weather - Weather
 * @property {number} waterTemp - Water temperature
 * @property {number} airTemp - Air temperature
 * @property {number} secFished - Seconds fished
 * @property {number} voltage - Voltage of boat
 * @property {number} pulse - Pulse of boat
 * @property {number} conductivity - Conductivity of water
 * @property {string[]} species - Unique species in station
```

```

* @property {Observation[]} observations - Individual observations
* @property {number} transectLength - Length of transect
* @property {boolean} display - Display
* @property {string} gpxFile - Gpx file
*/
export class Station {
  constructor ({
    id = null,
    ...
    projectId = null
  } = {}) {
    this.id = id
    ...
    this.species
      = species ? species.map(s => s.toLowerCase()) : null // Ensure species
        is lowercase
    this.observations
      = observations ? observations.map(observation => new Observation(
        observation)) : null
    ...
    this.projectId = projectId
  }

  // Diabile eslint camelcase rule because of the PostgreSQL naming convention
  /* eslint-disable camelcase */
  static fromJson (object) {
    return new Station({
      id: object.id,
      ...
      projectId: object.project_id
    })
  }
  /* eslint-enable camelcase */
}

```

Listing O.11: The Station class with its fromJson method

According to the data structure outlined in the requirement section about data classification, observations exist under stations, and stations exist under rivers. Because of this each stations observation is placed under its Station in an array. Each observation is its own object, containing data such as fish length and count. Whereas Rivers and Stations are in separate maps, Observations are still placed directly under their Station. This is because in contrast to Rivers and Stations, where Stations can exist and be shown without referencing the River, Observations are only relevant in relation to the Station and have the same lifetime. It does not make sense to store an Observation when its Station does not exist. The Observation objects

Figure O.3: Screenshot of menu for selecting river or stations

also have a JSDoc comment and a constructor, however they not need a `fromJson()` method because their JSON data contains the same property names as the class.

Filtering of data

Filtering of data is done using the `filterData` module. It contains functions which takes in rivers or stations objects, and filter these based on parameters given, for example the desired date range. This allows other modules and components to call these functions when they have data to be filtered.

The pop-up menu for selecting rivers and stations, used on the graph and download page, requires filtering of data to display suggestions to the user (see Figure O.3). Here the user can filter suggestions based on the datatype, either rivers or stations, and a date range. There are several hundred rivers and stations to choose from, it is useful to be able to narrow this down. The user can then search using the river or station name and date, where suggestions will appear based on their input.

A function for filtering is `filterRiversByDateAndSpecies()` (see Listing O.12). This function can be called by components when a user has selected the species and date-range they are interested in. The functions first calls `filterRiversBasedOnDates()`, then `filterObjectsBasedOnSpecies()`, and returns the rivers after they have been filtered by both, catching any potential errors that could occur.

```
/**
 * Filters rivers based on their species and date
 * @param {Map<number, object>} rivers - The map of rivers to filter, keyed by
   a unique id
 * @param {string[]} species - The species to filter on
 * @param {string} startDate - The start of the date interval to filter on
 * @param {string} endDate - The end of the date interval to filter on
 * @returns {Map<number, object>} - A filtered map of rivers
 */
export function filterRiversByDateAndSpecies (rivers, species, startDate,
  endDate) {
  try {
    // Filter rivers based on if they are between or have overlap with the
    start and end date
    const filteredDateRivers = filterRiversBasedOnDates(rivers, startDate,
      endDate)

    // Filter rivers based on if they have a species that is in the species
    list
    const filteredSpeciesAndDateRivers = filterObjectsBasedOnSpecies(
      filteredDateRivers, species)
```

```

    return filteredSpeciesAndDateRivers
  } catch (error) {
    addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.GENERIC,
      FEEDBACK_MESSAGES.GENERIC)
    return new Map()
  }
}

```

Listing O.12: The function `filterRiversByDateAndSpecies()` from the `filterData` module

The function `filterObjectsBasedOnSpecies` filters objects by iterating through them and only keeping the ones with the correct species (see Listing O.13). First it checks if there are any species to filter on, and if there are none it returns all rivers or stations. Then it creates a set with the selected species. Sets are hash table under the hood (source1), ensuring a significant performance boost compared to arrays when checking if values exist in them.

After creating the set, `filterObjectsBasedOnSpecies` checks if each object has any of the species specified in their species attribute. It does this by creating an array from the objects, using `[...objects]`, and then using `.filter(()=> {})` to filter out each object not satisfying the condition provided. The condition checks if the object has any species defined, and if so, it checks if any of the species are in the selected species to filter on. By using `.some()`, the function will only check the object species until one of them are selected, avoiding iterating through all of them. Finally the function returns the objects which has the selected species.

```

/**
 * Returns objects which have a species that is in the speciesToFilterOn
 * @param {Map<number, object>} objects - The map of objects to filter
 * @param {string[]} speciesToFilterOn - The species to filter on
 * @returns {Map<number, object>} - A filtered map of objects
 */
function filterObjectsBasedOnSpecies (objects, speciesToFilterOn) {
  // If no species are given, don't filter on species
  if (!speciesToFilterOn || speciesToFilterOn.length === 0) return objects

  // Use set instead of array for faster lookups
  const speciesToFilterOnSet = new Set(speciesToFilterOn)

  // Return objects which have a species that is in the speciesToFilterOn
  return new Map([...objects].filter(([_, object]) =>
    object[attributesToFilterOn.SPECIES] &&
    object[attributesToFilterOn.SPECIES].some(
      objectSpecies => speciesToFilterOnSet.has(objectSpecies))
  ))
}

```

Listing O.13: The function `filterObjectsBasedOnSpecies()` from the `filterData` module

Plotly data

To convert the observation data into plots, the raw observation data has to be formatted in specific formats that Plotly can understand. This is done using the `plotlyData` module. The formatting considers how the user wants the data to be represented. The application uses four different Plotly plot types; bar charts, pie charts, histograms, and box plots. Each of these requires their own data structure and unique implementation, except for bar and pie charts, which use the same. The data for each plot uses the observations under specific rivers or stations selected. In addition, the user can choose between combining all selected rivers/stations, or comparing them. They can also choose which species to view, and optionally display all species not selected in a *others* category. For bar and pie charts, the user can choose to view the absolute amount of fish per species per river/station, or to view this in relation to the amount spent fishing per river/station. For the histogram and box plot, the user can choose to group all species in each river/station together, to compare the sum of each river/station. In the histogram the user can choose the size of the interval used to for the length distribution. All in all there are a lot of factors to consider when creating the Plotly data.

Plotly data structure

Both the bar chart and the pie chart uses the same data structure. They require data where each river/station contains the amount of fish for each species (see Listing O.14). Plotly groups the river/station together and displays the amount of each species (see Figure O.4).

```
{
  river1: {
    laks: 10,
    ørret: 1
  },
  river2: {
    laks: 6,
    ørret: 0
  }
}
```

Listing O.14: Bar and pie chart data structure

The data structure for histogram and box plots are quite similar, however they are not identical. Both group the data by each unique river/station and species pair, for example 'river1 - laks'. The difference between them is that for histogram, each "river/station - species" pair contains three values; count, intervals, and interval. Count is the amount of fish in each interval, intervals is the center of each intervals the observations are grouped in, and interval is simply the size of each interval (see Listing O.15).

```
{
  'river1_-_laks': {
    count: [1, 0, 2],
    intervals: [2, 6, 10],
```

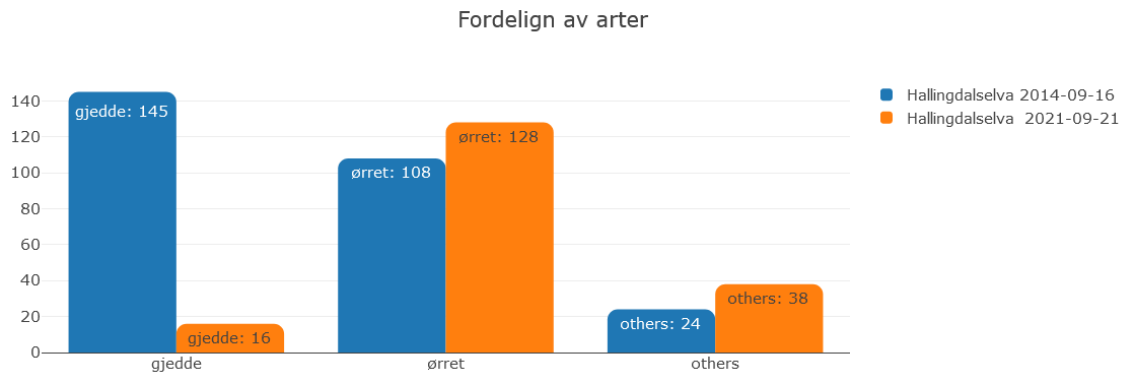


Figure O.4: Example bar chart with two Stations and two species + others selected

```

    interval: 4
  }
}

```

Listing O.15: Histogram data structure, with the amount of laks between 0-4, 4-8, and 8-12

The box plot on the other hand contains an array of each observation's length per "river/station - species" pair (see Listing O.16).

```

{
  "river1_-_laks": {
    length: [3, 10, 11]
  }
}

```

Listing O.16: Box plot data structure, with the length of each laks observation

Functions

Organizing the code in the `plotlyData` module is done utilizing several different functions. The two main functions, which are the ones called from the components, are `dataForBarAndPieChart()` and `dataForHistogramAndBoxPlot()`. These functions take in the selected rivers or stations, the species to use, if the value should be absolute or relative, if rivers/stations should be aggregated, if other species are included, if species should be combined, and interval size.

There are several other functions in the `plotlyData` module, one notable one being `getIntervalsForObservations()` (see Listing O.17). This function creates the count, intervals, and interval value for each "river/station - species" pair. It first checks if there are any observations, and if so it find the minimum and maximum values of the observations. It uses this to calculate where to start and end each interval. Using the start and end of the intervals in combination with the interval length given by the user, the function iterates through all the

intervals and counts the amount of fish in each. After counting the amount, it centers each interval position so Plotly knows where to place each bar and point in the histogram, and finally the function returns the counts, intervals, and the interval size.

```
/**
 * Counts the observations in intervals, and returns the count with the
   intervals
 * @param {Observation[]} observations - The observations to group by interval
 * @param {number} intervalSize - The interval in cm to group the data by
 * @returns {{
 * count: number[], intervals: number[], interval: number
 * }} - The count of observations in each interval
 */
function getIntervalsForObservations (observations, intervalSize) {
  // If there are no observations, return empty data
  if (observations.length === 0) {
    return { count: [], intervals: [], interval: intervalSize }
  }

  // Find minimum and maximum length of observations, rounded down to the
  nearest interval
  const lengths = observations.map(observation => observation.length)
  const min = Math.floor(Math.min(...lengths) / intervalSize) * intervalSize
  const max = Math.floor(Math.max(...lengths) / intervalSize) * intervalSize

  // Create the intervals
  const intervals = []
  for (let i = min; i <= max; i += intervalSize) {
    intervals.push(i)
  }

  // Count the observations in each interval
  const count = intervals.map(interval => {
    const intervalObservations = observations.filter(observation =>
      observation.length >= interval && observation.length < interval +
      intervalSize)
    return amountOfFishInObservations(intervalObservations)
  })

  // Shift each interval to the middle of the interval for placing the bars in
  a histogram
  intervals.forEach((_, index) => {
    intervals[index] = intervals[index] + intervalSize / 2
  })
}
```

Stasjon	Elvtype	Vær	Min fisket	Ant fisk	Fisk/min
1	Glattstrøm	Snø	9.2	86	9.38
2	Glattstrøm	Snø	9.5	67	7.08

Figure O.5: Table containing the stations under a river when a river is selected on the map page

```
return { count, intervals, interval: intervalSize }
}
```

Listing O.17: The function `getIntervalsForObservations()` in the `plotlyData` module

Calculate data

In addition to wanting to view the data in plots, the user should also be able to view data about each river and station when they are clicked on the map and list page. The raw data does not contain all of the values the user should be shown. This includes information such as amount of fish in a river, amount of fish per minute in a station, median length of fish in a station, and more. To calculate this data the module `calculateData` is used.

A function from this module is `fishPerMinuteInStations()`, as can be seen in Listing O.18. This function takes in a map of stations, where it first uses the function `secondsSpentFishingInStations()` to find the amount of time spent fishing in seconds. To then find the amount of fish the function `amountOfFishInStations()` is called, and the function then returns this divided by minutes spent fishing, while rounding the number to two decimals.

```
/**
 * Calculates the amount of fish observed per minute in multiple stations
 * @param {Map<number, Station>} stations - The stations to calculate based on
 * @returns {number} - The amount of fish observed per minute
 */
export function fishPerMinuteInStations (stations) {
  const secSpentFishing = secondsSpentFishingInStations(stations)
  return (amountOfFishInStations(stations) / (secSpentFishing / 60)).toFixed(2)
}
```

Listing O.18: The function `fishPerMinuteInStations()` in the `formatData` module

Format data

To allow users to download data and view it in tables, the module `formatData` is used to convert data into predefined formats. This includes formats for rivers, stations, and observations in CSV and Excel files. Moreover, it also includes formats for several types of tables which are used in the application, such as formatting multiple stations for display in a river summary (see Figure O.5).

A function from this module is `formatStationsForSummaryTable()` (see Listing O.19). This function creates an object containing headers and rows for a table based a map of stations. The headers are imported from another file, while the rows are created by iterating through the stations, adding each station's data to their row. This includes both raw data, such as their station number and river-type, and calculated values, such as the amount of fish in station, which is calculated using the module `calculateData`.

```
/**
 * Formats the station data for the station summary table
 * @param {Map<number, Station>} stations - The stations to be formatted
 * @returns {{headers: string[], rows: string[][]}} - Headers and rows for the
   table
 */
export function formatStationsForSummaryTable (stations) {
  const headers = headersConstants.STATION_SUMMARY_HEADERS_TABLE

  const rows = []
  // Create a row for each station
  stations.forEach(station => {
    rows.push([
      station.id,
      station.name.split('_').pop(), // Get the station number from the name
      station.riverType || '',
      station.weather || '',
      (station.secFished / 60).toFixed(1),
      amountOfFishInStation(station),
      fishPerMinuteInStation(station)
    ])
  })

  return { headers, rows }
}
```

Listing O.19: The function `formatstationsForSummary()` in the `formatData` module

File handling

The application should handle files, allowing users to both upload and download data. The data can be uploaded in the `xlsx` file format, and downloaded in both the `csv` and the `xlsx` file formats. This section shows how this was implemented as part of the application.

Upload of data

To allow users to upload data, the website has implemented the possibility for users to drag and drop a file, or chose a file using the file explorer on their computer (see Figure O.6). The

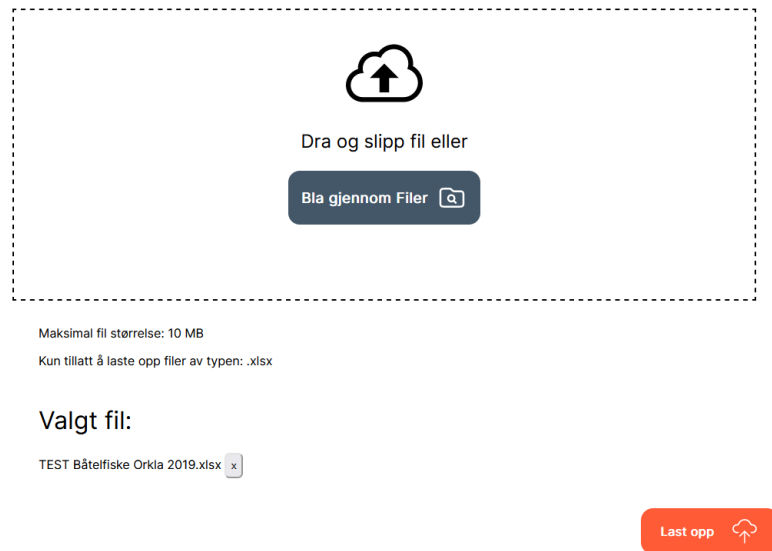


Figure O.6: Screenshot of upload page with a file chosen

file has to be an Excel file, with the `.xlsx` extension, and have the format specified in the `Upload.md` documentation file in the project repository. Only one file can be uploaded at once, with a maximum size of 10MB.

Selecting files

The upload page uses a variable named `selectedFile`, which contains the file the user has selected. As long as a file hasn't been selected, this variable is set to `null`. When a user clicks the browse files button, called 'Bla gjennom filer', a function called `selectFile()` runs (see Listing O.20). This function creates an input element accepting one file Excel file. The variable `selectedFile` is then set to the file chosen in the input by the user.

```
function selectFile () {  
  const fileInput = document.createElement('input')  
  fileInput.type = 'file'  
  fileInput.multiple = false  
  fileInput.accept = '.xlsx'  
  fileInput  
    .addEventListener('change', (e) => { selectedFile = e.target.files[0] })  
    .click()  
}
```

Listing O.20: Function for browsing and selecting a file on the upload page

Using the property `on:drop|preventDefault={handleDrop}` on the box element allows users to drop files into the box. This property ensures that the default behaviour of dropped

files is avoided, which is to immediately open them. Furthermore the `handleDrop` function is called, which checks the amount of files and type, and if these are correct it sets the file dropped as the `selectedFile`.

Parsing file

Validation happens after a user tries to upload a file. The application does validation by reading the file contents and checking the format and content of the cells in the file. This is explained further in the validation section under the security implementation chapter.

Uploading file to server

The selected file is configured to be sent to an upload endpoint managed by NINA. The endpoint expects HTTP POST requests with `Content-Type: multipart/form-data`, where the body will contain the file encoded as MIME parts. This is a standard way of sending files to a HTTP endpoint, where the file is sent as multiple HTTP requests to an endpoint.

Sending the file to the endpoint is done using the `uploadFileToServer()` function in the module `upload` (see Listing O.21). This function creates a `formData` object containing the file, and then sends a request to the upload endpoint. It uses the POST method, sends all cookies in the domain if the origin is the same, and specifies that the body should contain the `formData`. Because the `formData` object is used in the body, the browser will automatically split the request into separate HTTP requests, each with `Content-Type: multipart/form-data` and parts of the file encoded as MIME⁸. The function then checks the status and gives the user feedback based on if the file was successfully uploaded or not.

```
/**
 * Upload file to server
 * @param {File} file - The file to upload
 * @returns {Promise<boolean>} - A promise which resolves to if upload was
   successful or not
 */
export async function uploadFileToServer (file) {
  // Create a new FormData object and add file to it
  const formData = new FormData()
  formData.append('file', file)

  try {
    // Try to upload the file
    const response = await fetch(`${UPLOAD_URL}${UPLOAD_ENDPOINT}`, {
      method: 'POST',
      body: formData,
      credentials: 'same-origin'
    })
  }
```

⁸<https://openjavascript.info/2022/06/08/how-to-upload-a-file-using-the-fetch-api/>

```
    if (response.status === 401) {
      ...
    }

    if (!response.ok) {
      ...
    }

    const result = await response.json()

    if (result.success) {
      addFeedbackToStore(FEEDBACK_TYPES.SUCCESS,
        FEEDBACK_CODES.UPLOAD_SUCCESS, FEEDBACK_MESSAGES.UPLOAD_SUCCESS)
      return true
    }
    ...
  } catch (error) { // Catch any possible network or fetch errors
    ...
  }
}
```

Listing O.21: Function `uploadFileToServer()` in `upload` module

Download of data

Downloading data is done by another page of the application (see Figure O.7). On this page the user chooses which rivers or stations they want to download, the species they want to include observations for, and the type of file they want (`xlsx` or `csv`). The downloaded file follows the same format as uploaded files. A `xlsx` file separates the data between different sheets, however since `csv` files only have one sheet, they have all the data on the same page.

Downloading file

After the user has chosen what they want to download and clicked the download button, the `downloadFile()` function is ran (see Listing O.22). This functions first checks if the options chosen are valid. If they are, it creates the file name and the content by using the `fileHandler` module functions `generateExcelFile()` or `generateCSVFile()`. If the file was successfully created the function creates a temporary HTML element which automatically saves the file created to the users computer.

```
/**
 * Downloads a file with the content specified by the user
 * @returns {void}
 */
async function downloadFile () {
```

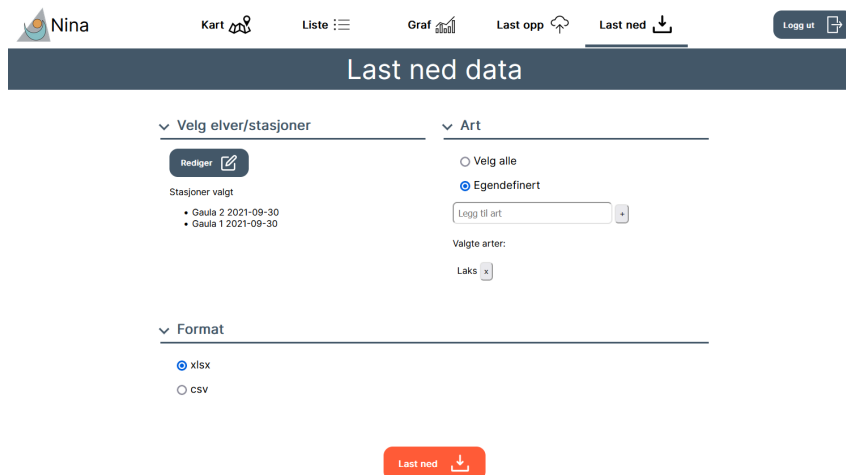


Figure O.7: Screenshot of download page with stations, a species, and the xlsx format selected

```
// Check if the user has chosen a file format
if (selectedFormat === '') {
  ...
}

// Check if the user has chosen rivers but not selected any
if (dataType === DATATYPE_RIVER && selectedRivers.size === 0) {
  ...
}

// Check if the user has chosen stations but not selected any
if (dataType === DATATYPE_STATION && selectedStations.size === 0) {
  ...
}

// Create a file name and file extension
const fileExtension = selectedFormat === 'xlsx' ? '.xlsx' : '.csv'
const fileName = dataType === DATATYPE_RIVER ? 'elver' : 'stasjoner' +
  fileExtension

// Create a blob with the data
const blob = selectedFormat === 'xlsx'
  ? await generateExcelFile(selectedRivers, selectedStations, dataType,
    selectedSpecies)
  : await generateCSVFile(selectedRivers, selectedStations, dataType,
    selectedSpecies)
```

```

// If no file was created, return
if (blob.size === 0) {
  return
}

// Create a URL for the blob
const blobUrl = URL.createObjectURL(blob)

// Create a temporary anchor element, set the href and download attributes to
  the URL and file name
const a = document.createElement('a')
a.href = blobUrl
a.download = fileName
a.style.display = 'none'

// Append the anchor element to the DOM and click it
document.body.appendChild(a).click()

// Remove the anchor element from the DOM
document.body.removeChild(a)
}

```

Listing O.22: Function `downloadFile()` on download page

The filehandler module

The actual functions which create the Excel and CSV files are called `generateExcelFile()` and `generateCSVFile()` which are placed in the `fileHandler` module. `generateExcelFile()` generates an excel file containing the selected rivers or stations and the selected species (see Listing O.23). It retrieves this data using `formatRiversForExcel()` or `formatStationsForExcel()` from the `formatData` module. These functions format the data for by splitting it into a river, a station, and an observation sheet. Each of these sheets contains headers and rows for each river, station, and observation. The library `ExcelJS` is then used to create a workbook containing the different sheets. The workbook is finally converted to a `Blob` representing the content of a spreadsheet, which is a way for JavaScript to store a file as binary data, and is then returned.

```

/**
 * Generates an Excel file from the given data
 * @param {Map<number, River>} rivers - The rivers to generate the Excel file
  from
 * @param {Map<number, Station>} stations - The stations to generate the Excel
  file from
 * @param {string} type - - The type of data ('river' or 'station')
 * @param {string[]} selectedSpecies - The species to include in the Excel file

```

```

* @returns {Promise<Blob>} A promise that resolves with a Blob representing
  the Excel file.
*/
export async function generateExcelFile (rivers, stations, type,
  selectedSpecies) {
  try {
    // Format the data for Excel
    const data = type === 'river' ? formatRiversForExcel(rivers,
      selectedSpecies)
      : formatStationsForExcel(stations, selectedSpecies)

    const workbook = new ExcelJS.Workbook()

    // Create a river worksheet, and add each river to it
    const riverSheet = workbook.addWorksheet('Elvedata')
    riverSheet.addRow(data.riverHeader)
    data.riverRows.forEach(row => {
      riverSheet.addRow(row)
    })

    // Create a station worksheet, and add each station to it
    ...

    // Create an observation worksheet, and add each observation to it
    ...

    // Write the workbook to a buffer
    const buffer = await workbook.xlsx.writeBuffer()

    // Convert the buffer to a blob
    const blob = new Blob([buffer],
      { type: 'application/vnd.openxmlformats-officedocument.spreadsheetml.
        sheet' })

    return blob
  } catch (error) {
    addFeedbackToStore(FEEDBACK_TYPES.ERROR,
      FEEDBACK_CODES.NOT_FOUND, FEEDBACK_MESSAGES.ERROR_GENERATING_FILE)
    return new Blob()
  }
}

```

Listing O.23: Function generateExcelFile() from fileHandler module

Leaflet

Leaflet is used by the application to create an interactive map, allowing the users to view and interact with all the river or station data points (see Figure O.8). Leaflet handles rendering, placing of markers, grouping of figures, and interaction with the map. This following shows how Leaflet was implemented into the application.

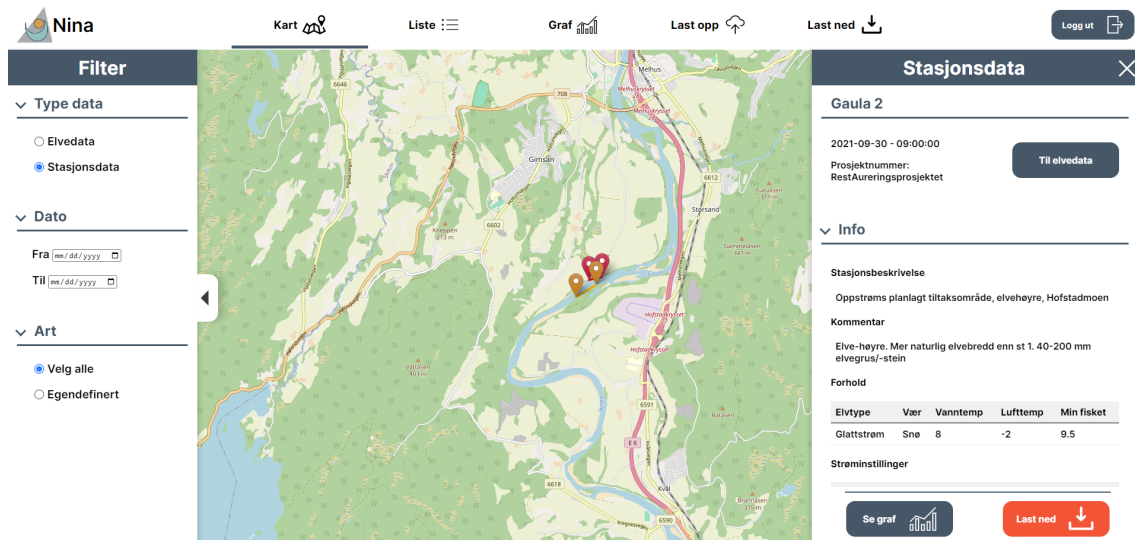


Figure O.8: Screenshot of the map page with a station selected

Initialization, controls, and layers

The Leaflet map is only initialized after the Leaflet Svelte component is mounted (see Listing O.24). It's crucial that the map waits for all CSS to load by using a delay, because the map might load incorrectly with wrong dimensions if initialized too early. The map is initialized with a terrain map zoomed in on Norway, since this is most of NINA's data is located. Different controls are added on top of the map using the `leaflet.control` functions, including zoom, scale, and map type controls. The river and station markers groups containing the data points are also added to the map. There are three map layers added, one for terrain, one for satellite, and one for a topology. These are defined and imported from another file.

```
onMount(async () => {
  // Wait 250ms after DOM is rendered before creating map to ensure all css is
  // loaded and applied
  setTimeout(() => {
    // defines the map and sets the view to Norway
    map = leaflet.map(mapElement, {
      layers: [mapLayers.Terrain], // Default layer
      zoomControl: false // Disable default zoom control
    }).setView([61, 12.09], 6)
  })
})
```

```

// Add zoom control in top right corner
leaflet.control.zoom({ position: 'topright' }).addTo(map)

// Add terrain and satellite layers option to the map
leaflet.control.layers(mapLayers, null, {
  position: 'bottomright'
}).addTo(map)

// Add scale in bottom left corner
leaflet.control.scale().addTo(map)

// Add river and station layer groups to the map
riverLayerGroup.addTo(map)
stationLayerGroup.addTo(map)
}, 250)
})

```

Listing O.24: Leaflet initialization

To display the map it needs to be linked with a HTML element. This is done by referencing the div you want to display the map in, which in this case is `mapElement`. The only CSS needed for the map div is the its height and width.

Markers

There are two groups of markers in the application, one for rivers and one for stations. They have similar logic for creation and handling clicks, however rivers are only a single marker, while stations consist of two markers and a line. These represent the start and stop position of the station, while the line connects the start and stop together. To illustrate how markers are handled in the application we will focus on station markers.

The station markers are added with the function `createStationMarker(station)` (see Listing O.25). Given a station object, the coordinates are extracted and used for creating a start and an end marker, and a line connecting them. All three have on-click functions dispatching an event specifying that a station was clicked, and the id of the station. The event will be sent to the parent component, described further in section about leaflet Svelte integration. Each of the markers and the line is added to the `stationLayerGroup`, which displays them to the user when `stationLayerGroup` is attached to the map. In addition, a `stationMarker` object is created and returned to handle later updates and potential removal of markers.

```

/**
 * Creates a marker for a station
 * @param {object} station - The station data
 * @returns {object} - The station start, end, and line markers
 */
function createStationMarker (station) {

```

```
// Get start and end position of the station
const startPos = [station.startPos.coordinates[1], station.startPos.
  coordinates[0]]
const endPos = [station.endPos.coordinates[1], station.endPos.coordinates[0]]

// creates a marker for the start position of the station
const startMarker = leaflet.marker(startPos, { icon: redIcon })
  .addTo(stationLayerGroup)
  .on('click', () => dispatch('stationClicked', { id: station.id }))

// creates a marker for the end position of the station
const endMarker = leaflet.marker(endPos, { icon: redIcon })
  .addTo(stationLayerGroup)
  .on('click', () => dispatch('stationClicked', { id: station.id }))

// drawing the line between the start and end position of the station
const polyline = leaflet.polyline([startPos, endPos], { color: 'red' })
  .addTo(stationLayerGroup)
  .on('click', () => dispatch('stationClicked', { id: station.id }))

// Return the markers and line in a object
const stationMarker = {
  startMarker,
  endMarker,
  line: polyline
}

return stationMarker
}
```

Listing O.25: createStationMarker() function

Users can filter stations by start and end dates, or by specifying the species they are interested in. The function responsible for managing and updating what stations are displayed is called `updateStations` (see Listing O.26). The function doesn't waste resources and focuses on being efficient, which is important when displaying thousands of points. It never redraws station markers, it only adds or removes them from the map. That means all station markers are only created once. To display stations in the map, the function iterates through all stations which should be displayed and adds the ones not already displayed. To add these the previously mentioned function `createStationMarker()` is utilized. All `stationMarker` objects are then saved in a JavaScript map using the unique ID for each station, enabling efficient later retrieval and updates of the stations drawn. Removal of stations are done by iterating through all `stationMarkers`, and removing the markers and lines when they are not part of the stations which should be displayed. Stations are both removed from the `stationLayerGroup`, which removes it from the map, and removed from `stationMarkers`, which tells the application that

it no longer is displayed.

```
/**
 * Updates the stations displayed in the map
 */
function updateStations () {
  // Add each station not already added to the map
  stations.forEach(station => {
    // Checks if the station already has a marker
    if (stationMarkers.has(station.id)) {
      return
    }
    // Create the station start, end, and line markers
    const stationMarker = createStationMarker(station)

    // Store the markers and line in a map using the station id as key
    stationMarkers.set(station.id, stationMarker)
  })

  // Remove station markers that are not in the data
  stationMarkers.forEach((stationMarker, id) => {
    if (!stations.has(id)) {
      stationMarker.startMarker.removeFrom(stationLayerGroup)
      stationMarker.endMarker.removeFrom(stationLayerGroup)
      stationMarker.line.removeFrom(stationLayerGroup)
      stationMarkers.delete(id)
    }
  })
}
```

Listing O.26: updateStations() function

The function `selectStation()` is used when a user selects a station marker. This function first calls another function `unSelectStation()`, making sure to deselect the previous station. Then the selected station has its color of both the markers and line changed to orange. Finally the leaflet function `flyTo()` is used to zoom in and centralize the selected station (see Listing O.27).

```
// Pan to the selected station and zoom in
map.flyTo(marker.startMarker.getLatLng(), 13, { duration: 0.5 })
```

Listing O.27: Use of `map.flyTo()` when selecting a station

Svelte integration

The Leaflet integration with Svelte is done through a singular component, which contains all Leaflet code logic. The script tag handles initialization and updating the map, the HTML is

simply a single div element, and the CSS sets the width and height of the div.

To render the Leaflet component in the graph page, it is simply included in the HTML (see Listing O.33). Using Svelte props, the Leaflet component is given the `dataType` value, the river and stations to display, and the selected river or station. The prop `dataType` tells the component if it should display the rivers or stations. On the events `stationClicked` and `riverClicked`, which are dispatched from the Leaflet component when a station or river is clicked, the appropriate functions on the graph page are called, handling the selection.

```
<!-- Map with rivers and stations -->
<LeafletMap
  {dataType}
  rivers={filteredRivers}
  stations={filteredStations}
  {selectedRiver}
  {selectedStation}
  on:stationClicked={stationClicked}
  on:riverClicked={riverClicked}/>
```

Listing O.28: Use of Leaflet component

Plotly

Plotly is responsible using the environmental data to create figures and diagrams, including bar charts, pie charts, histograms, and box plots. Each depict and illustrate the data in their in a unique way, giving the user a wide range of options for visualising data. The users selects the rivers or stations they want to visualise, allowing the user to compare or aggregate data, and specify the species they want to view. The figures also include special settings, allowing user to set the value to relative or absolute, or change the interval for the histogram. This section shows how Plotly is implemented into the application.

Drawing plots

Drawing the environmental data is done using data from the `plotlyData` module converted to traces. The data from `plotlyData` is structured differently for the different graph types, something that is discussed in more detail in section Plotly data creation under data handling. This section mainly focuses on how histograms are drawn. The data from `plotlyData` for the histogram contains the amount of fish in given intervals based on their length, grouped by species and the river or station of the observation. For example, the data could contain that there are twelve Laks in Gaula 2021 with a length between 80-100mm, and four Laks in Gaula 2021 with a length between 60-80mm. This data is then used to draw the traces.

The traces of a plot in Plotly is the content displayed on the page. A trace can for example be the bars showing the species count for a specific river, a single pie chart, or a single box plot. In the case of histograms, the traces represent each river/station - species par, for example the distribution of Laks in Gaula 2021. The traces are created using the custom function `drawPlot` (`plotData`), which iterates trough all the groups of data given and creates a trace for each.

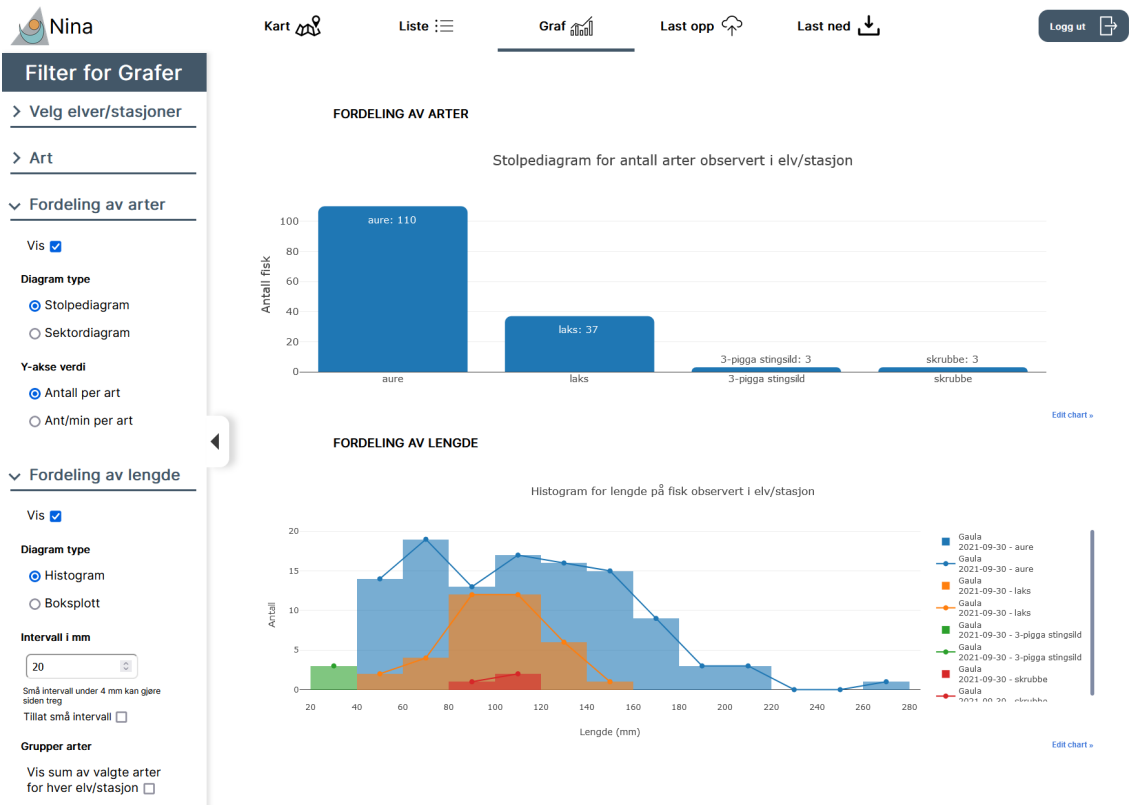


Figure O.9: Screenshot of the graph page with a river and all its species selected

The traces are different values based on the graph type, however they all take in data from the `plotlyData` module in addition to some other configuration such as type.

For the histogram, both bars and lines are displayed to clearly illustrate the distribution of fish length. When drawing the histogram the function `drawPlot(plotData)` iterates through each group of species and data-point, and for each group it creates one trace with bars and one with lines. These are then appended to an array called `traces` (see Figure O.9). Each bar trace takes in the group intervals and the count of fish for each interval, which are used as values for the x-axis and y-axis (see Listing O.29). Furthermore, configurations such as the color, opacity, name, hovertext, type (which is set to `bar`), and width of each bar are also set for the trace. This results in the boxes seen in Figure O.9. The lines are created using the same values and data as the bars, the difference being it uses different styling and has the type set as `line`.

```
// Create bars for histogram
traces.push({
  x: observationPoint.intervals,
  y: observationPoint.count,
  type: 'bar',
  name: nameWithEnter,
  width: observationPoint.interval,
```

```

marker: { color, opacity: 0.6 },
hovertext: hoverText,
hoverinfo: 'text'
})

```

Listing O.29: Creating a bar trace in plotly

Styling plots

Styling the plots is further done configuring the variables `layout` and `config`, which are used in addition to traces to draw the plots. The `layout` is used to define the title, axis names, and other configuration affecting the layout of the traces (see Listing O.30). The `config` on the other hand configures the behaviour of the whole Plotly element, including if it automatically resizes and which buttons to show (see Listing O.31). When the traces, layout, and config is defined, they are all used to create the finalized plot as seen in Listing O.32.

```

// Displays the title, names of the axes and removes gaps between bars
const layout = {
  title: 'Histogram_for_lengde_på_fisk_observert_i_elv/stasjon',
  xaxis: { title: 'Lengde(mm)', dtick: plotData.entries().next().value[1].
    interval }, // Set the interval between text on the x-axis the same as
    the interval between bars
  yaxis: { title: 'Antall' },
  barmode: 'overlay',
  hovermode: 'closest',
  bargap: 0
}

```

Listing O.30: Defining a plot layout in plotly

```

// Make graph responsive, remove some buttons from the modebar, add edit link
const config = {
  responsive: true,
  modeBarButtonsToRemove: ['select2d', 'lasso2d', 'zoomIn2d', 'zoomOut2d'],
  showLink: true,
  plotlyServerURL: 'https://chart-studio.plotly.com'
}

```

Listing O.31: Defining a plot config in plotly

```
Plotly.newPlot('fishHistogram', traces, layout, config)
```

Listing O.32: Drawing a plot in plotly

Svelte Integration

Each diagram type is defined in their own Svelte component, since they each have unique logic associated with drawing the plots. The script tags contains all the JavaScript needed to draw and update the plots, with the rest of the components only being a HTML div used by Plotly to draw inside.

The components are used by referencing them inside the HTML of another component. Using `if` statements in svelte (source1), the HTML can call different Plotly components based on user choices (see Listing 6.14). The components only need the formatted data from the `plotlyData` module, however bar chart also needs to know if the data is absolute or relative to display the correct unit.

```
<!-- Plot graph choosen by user -->
{#if type === 'barchart'}
<BarChartComponent plotData={formattedData} {absoluteValues}/>
{:else if type === 'piechart'}
<PieChartComponent plotData={formattedData}/>
{:else if type === 'boxplot'}
<BoxPlotComponent plotData={formattedData}/>
{:else}
<HistogramComponent plotData={formattedData}/>
{/if}
```

Listing O.33: Use of Leaflet component

Navigation

The application was created with several pages, and the user needs to be able to navigate between them. Navigation includes being able to go the each page by using the navigation bar. Moreover, it also includes being able to navigating by opening selected rivers and stations on different pages, for example by clicking 'show in map'. The following is how this is achieved by the application.

SvelteKit routes

Utilizing SvelteKit enables the application to be a single page application. SvelteKit allows for creating specific routes as folders in the file structure, where each file route is equivalent to an URL path⁹. Each page on the application has it's own folder and route, except for the map page which is placed in the root folder. The root folder specifies the "homepage", the default page when opening the application (see Figure O.10). Each route has a `+page.svelte`, which are components defining the specific HTML, CSS, and JavaScript for the page. For example the upload page is placed under `src/routes/upload/`, and will be accessible on the page under `<domain>/upload/`.

⁹<https://kit.svelte.dev/docs/routing>

SvelteKit is by default designed for server-side rendering, meaning when a user switches between pages, each route is rendered on the server and then sent to the client. Disabling this, and thus enabling that users can navigate between pages without reloading, is done by setting the line `export const ssr = false` inside `+page.server.js` under `/routes/`. This is called client-side rendering, meaning the entire application will be rendered on the client side (source1). This results in the user getting all the pages when opening the web site.

In the `+page.server.js`, SvelteKit can be configured further by setting the `prerender` option to `true` (`export const prerender = true`), telling SvelteKit to pre-render all pages in the application (source1). This results in the pages being turned into static HTML at build time, preventing the pages to be built for each request sent by a user. Furthermore, as the entire application is pre-rendered, `adapter-static` in the `svelte.config.js` can be set. This ensures that the application build outputs files for static serving, which can be used directly by Nginx.

```

src/routes/
├── +page.svelte
├── +layout.svelte
├── +page.server.js
├── list/
│   └── +page.svelte
├── graph/
│   └── +page.svelte
├── upload/
│   └── +page.svelte
├── download/
│   └── +page.svelte
├── login/
│   └── +page.svelte

```

Figure O.10: SvelteKit filesystem routes

Use of `<a>` elements

SvelteKit's support of the `<a>` HTML element (source1) is used to let users navigate between pages. This is done by setting the `href` attribute of the `<a>` element to the target path, for example ``. Furthermore, the `href` attribute can contain URL parameters, for example ``, allowing the user to seamlessly navigate to another page in the application while the application remembers the stations or rivers selected.

URL parameters

The application uses URL parameters to allow for cross site navigation where selected rivers and stations are transferred to the new page, and to allow users to share and bookmark pages with specific rivers or stations selected. On the map and list page a URL parameters corresponds

to the selected river or station. The URL parameters on the graph and download page works the same way, except that these pages allow multiple rivers or stations to be selected at the same time, resulting in multiple URL parameters. The function `getUrlParams()` reads the URL parameters on the graph and download page (see Listing O.34). Using `$page.url.search`, which is a Svelte variable containing all the parameters in the URL, the function retrieves the river and station ids. The function ensures the parameter strings are valid integers and then converts them to integers. The correct datatype is then set, and the function adds all id's corresponding to existing river or stations into a `selectedRiver` or `selectedStation` array.

```
/**
 * Gets the rivers or stations based on the URL parameters
 */
function getUrlParams () {
  // Get the river and station ids
  const searchParams = new URLSearchParams($page.url.search)
  const riverIdsIn = searchParams.getAll(DATATYPE_RIVER)
  const stationIdsIn = searchParams.getAll(DATATYPE_STATION)

  // Check if each river and station id is a valid integer
  if (riverIdsIn.some(id => !validateInteger(id))
    || stationIdsIn.some(id => !validateInteger(id))) {
    urlParamsLoaded = true
    return
  }

  // Convert the ids to numbers
  const riverIds = riverIdsIn.map(Number)
  const stationIds = stationIdsIn.map(Number)

  const selectedRiversUrl = new Map()
  const selectedStationsUrl = new Map()

  // Select the rivers or stations and datatype based on the ids
  if (riverIds.length > 0) {
    dataType = DATATYPE_RIVER
    riverIds.forEach(id => {
      // Check if river is selectable, if not ignore
      const river = rivers.get(id)
      if (river) {
        selectedRiversUrl.set(id, river)
      }
    })
    selectedRivers = selectedRiversUrl
  } else if (stationIds.length > 0) {
    dataType = DATATYPE_STATION
```

```

stationIds.forEach(id => {
  // Check if station is selectable, if not ignore
  const station = stations.get(id)
  if (station) {
    selectedStationsUrl.set(id, station)
  }
})
selectedStations = selectedStationsUrl
}

urlParamsLoaded = true
}

```

Listing O.34: The function `getUrlParams()` used on the graph and download page

Updating the URL after a user has selected a river and stations is done by the function `updateUrl()` (see Listing O.35). This function first ensures that the `window` object is defined. The `window` object can for example be undefined if code runs before a page has completed loading, resulting in an exception being thrown when trying to read the undefined URL. After ensuring it is defined, the function removes the old parameters, adds each river or station id as long as they are numbers, and uses the `goto()` SvelteKit function to update the URL. The `goto()` function with `replaceState: true` replaces the current URL with the new one without reloading the page or modifying the browser history stack. By not modifying the browser stack users can go back to previous pages by clicking the back button in the browser.

```

/**
 * Updates the URL to reflect the selected rivers or stations
 * @param {Map} selectedRivers - The selected rivers
 * @param {Map} selectedStations - The selected stations
 */
function updateUrl (selectedRivers, selectedStations) {
  // Check if the component is running in the browser
  if (typeof window === 'undefined') return

  // Get the current URL and remove any old river and station parameters
  const url = new URL(window.location.href)
  url.searchParams.delete(DATATYPE_RIVER)
  url.searchParams.delete(DATATYPE_STATION)

  // Add the selected rivers to the URL
  if (dataType === DATATYPE_RIVER) {
    selectedRivers.forEach((_, id) => {
      if (!isNaN(id)) {
        url.searchParams.append(DATATYPE_RIVER, id)
      }
    })
  }
}

```



```

} else if (dataType === DATATYPE_STATION) {
  // Add the selected stations to the URL
  selectedStations.forEach((_, id) => {
    if (!isNaN(id)) {
      url.searchParams.append(DATATYPE_STATION, id)
    }
  })
}
// Update the URL
goto(url.toString(), { replaceState: true })
}

```

Listing O.35: The function `updateUrl()` used on the graph and download page

Error Handling

There are multiple potential sources for errors in the application, e.g. wrong user input, a third-party component failing, or an internal software error coming from trying to read a null value. The application handles these safely, and provides the user with useful, but non-verbose, feedback. This section looks into how this was achieved.

Handling errors in code

Handling errors in the code is done using the `try-catch` JavaScript blocks. These ensures that if any errors happens inside of the `try` block the `catch` block will handle it. In the function `fetchFromPostgrest()`, a `try` block has been wrapped around the construction of a `fetch` request and the function which handles the response (see Listing O.37). If any errors happens in this function when constructing the request, sending it, or receiving it, the error will be explicitly handled by the `catch` block. The user is then given information about the error, in this case that the application was not able to retrieve the data from PostgREST.

```

/**
 * Fetches data from the PostgREST API on the endpoint specified
 * @param {string} endpoint - The endpoint with optional parameters to fetch
 *   data from
 * @returns {Promise<object>} - A promise which resolves to json data
 * @throws {Error} - Thrown if the fetch fails
 * @async
 */
async function fetchFromPostgrest (endpoint) {
  // Tries to fetch data, catches any errors
  try {
    // Fetch data from given url and endpoint
    const response = await fetch(`${POSTGREST_URL}${endpoint}`, {

```

```

    method: 'GET',
    credentials: 'same-origin'
  })
  // Return the response
  return handleResponse(response)
} catch (error) {
  addFeedbackToStore(FEEDBACK_TYPES.ERROR,
    FEEDBACK_CODES.POSTGREST_UNAVAILABLE, FEEDBACK_MESSAGES.
      POSTGREST_UNAVAILABLE)
}
}

```

Listing O.36: The function `fetchFromPostgrest()` in the `postgrest` module

Handling invalid user input is done by checking the inputted data and informing the user when appropriate. For example, the function `downloadFile()` checks if a user has selected a format. If no format was selected, the function cancels the download and informs the user of the type of error with a relevant predefined error message (see Listing O.37).

```

// Check if the user has chosen a file format
if (selectedFormat === '') {
  addFeedbackToStore(FEEDBACK_TYPES.ERROR,
    FEEDBACK_CODES.NOT_FOUND, FEEDBACK_MESSAGES.NO_FILE_FORMAT_SELECTED)
  return
}

```

Listing O.37: The function `fetchFromPostgrest()` in the `postgrest` module

Displaying errors and feedback

Displaying user feedback, including errors, to the user is done utilizing a combination of stores, modules, and Svelte components. The feedback is saved in a writable Svelte store called `userFeedbackStore`. To display user feedback to the user, the function `addFeedbackToStore(type, code, message)` is called. The function creates a feedback object containing the feedback type, code, and message, and appends the message to the feedback store. Type of feedback are error, success, or information. The code categorises what the feedback is about, e.g. forbidden or unauthorised, while the message is more detailed information, which is what's displayed to the user. Both the error types, codes, and messages are defined in a constants file. This means all error types, codes, and messages are pre-defined, ensuring that no internal error message are disclosed to the user.

Actually displaying the user feedback in a pop-up window is done with the component `UserFeedbackMessage`, which is included on all the pages (see Listing O.38 for component code, screenshot O.11 for example on website). This component displays all feedback messages in pop-up windows using the `Modal` component, which is a component that creates pop-up menus with given content. When a user clicks close on a displayed feedback, it is deleted

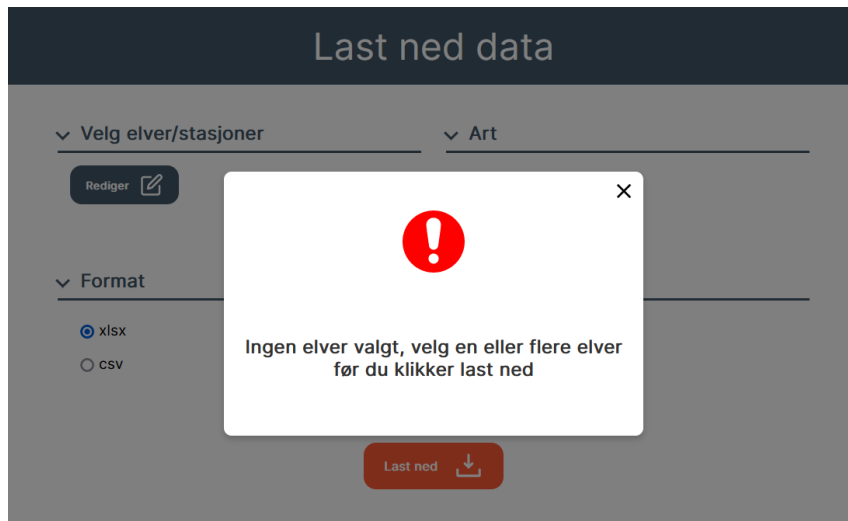


Figure O.11: Error message when user tries to download without selecting a river

from the feedback store, allowing the user to continue using the page or revealing another feedback message underneath. To indicate the type of feedback, the pop-up menu uses a SVG figure representing the feedback type, and displays the feedback message as the text.

```

<script>
  import { userFeedbackStore } from '../stores/userFeedbackStore.js'
  import Modal from './Modal.svelte'
  import { SVG_PATHS } from '../constants/svgPaths'

  $: userFeedback = $userFeedbackStore

  /**
   * Handles the close event of the modal
   * @returns {void}
   */
  function handleClose () {
    userFeedbackStore.update((feedback) => {
      feedback.pop()
      return feedback
    })
  }
</script>

<!-- Shows each user feedback messages as modals, overlaying each other -->
{#each userFeedback as feedback}
  {#if feedback.message}
    <Modal on:close={handleClose}>

```

```

<img
  src={feedback.type ? SVG_PATHS[feedback.type] : SVG_PATHS.CLOSE}
  alt={feedback.type}
  class='icon'
/>
<h3>{feedback.message}</h3>
</Modal>
{/if}
{/each}

```

Listing O.38: The UserFeedbackMessage component, not including `<style>`

Style

CSS

CSS is easy to manage with Svelte, mainly because CSS is scope-limited to the component in which the style is coded. The styles specified in each component only apply for that specific component. Figure O.12 shows how Svelte applies specific classes to make sure each component is unique. The left part of the figure shows a `<p>` (paragraph) styled in CSS using Svelte, while the right shows a `<p>` (paragraph) styled normally.

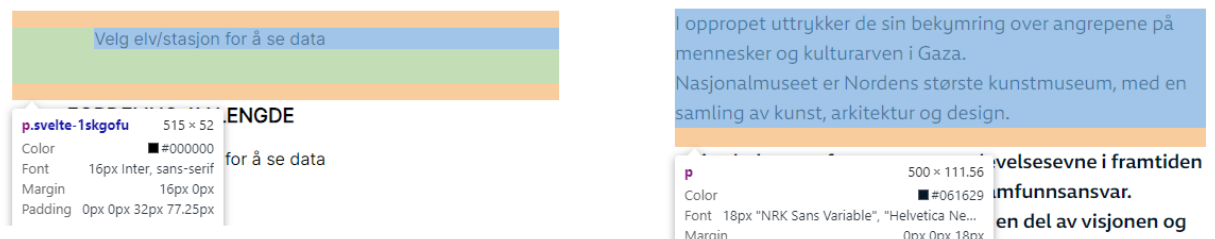


Figure O.12: How svelte uses css compared to the normal method

The application is separated into many components, each component containing basic HTML only relevant for the component. This causes there to be less CSS in each file, making the code less complex and more maintainable. Changing the style for one component won't interfere with other files, which means it won't change the design for the entire site, only the parts which correlate to the component that is changed. This makes it so when multiple people are working on CSS for different components at the same time, it will never cause any problems.

CSS is applied using the class attribute, an attribute which can be applied to any HTML element. CSS classes are usually named something that relates to the HTML element itself. The class `downloadHeader` indicates that the style is being applied to the header on the download page.

The usual practice of applying classes is using HTML div elements, which are created around the real HTML content. One reason for this is that it's currently not possible to apply styles directly to custom elements, meaning the simplest solution is surrounding the custom elements with something that can be styled. The component-based nature of svelte results in a lot of the HTML code being custom components. Figure O.13 shows a div element used for styling a custom elements. There are multiple elements one can use to surround custom HTML elements, but div's are considered the best because they contain no semantic values by default. Div elements never interferes with other styling choices since they have none themselves, making them perfect for applying styles.

```
<!-- River/station page, invisible unless a river or station is selected -->
{#if selectedRiver.id}
<div class='riverStationPage'>
  <Sidebar title={riverStationPageTitle} typeClose='cross' on:close={toggleSummaryPage}>
    <RiverSummary river={selectedRiver} wide={true} on:goToStationData={stationClicked}/>
  </Sidebar>
</div>
{:else if selectedStation.id}
<div class='riverStationPage'>
  <Sidebar title={riverStationPageTitle} typeClose='cross' on:close={toggleSummaryPage}>
    <StationSummary station={selectedStation} wide={true} on:goToRiverData={riverClicked}/>
  </Sidebar>
</div>
{/if}
```

Figure O.13: Using the html div element to style custom svelte html components

Input validation

Injection vulnerabilities are amongst the most common web application vulnerabilities (source1), and the most common countermeasure is comprehensive input validation. The application always assume all input is unsafe, including input from users, external services, and internal services. To validate the input the application checks several recommended properties outlined by Paul in his book *Official ISC2 Guide to the CSSLP CBK* (source2), including data format, datatype, and data range, making sure no input is interpreted as code.

User input

The application validates all user input, including search fields, number inputs and URL parameters. To validate the input the application uses the custom module validation under `utils /`, which contains functions such as `validateText`, `validatePassword`, and `validateNumber`. In addition the various inputs have their own specific validation. For example, the data range of the interval on the graph page has to be a positive number, or the id in the URL parameters have to match an existing river or station.

The function `validateNumber` checks if a string can be used as a number, ensuring the correct data type of inputs. URL parameter values are directly read as strings with no built-in validation, however the function `validateNumber` makes sure these values are validated. This function is not needed for other number inputs, as the HTML `<input>` element with type `number` has built-in validation to reject all non-numerical values (source2).

The function `validateText` checks if strings contain only valid characters (see listing O.42). It uses `regex` to specify the characters that are valid, and users get a predefined feedback message if their input is invalid (see screenshot O.14). This method of validation is called `whitelisting`, which is the opposite of `blacklisting`. `Blacklisting` means listing everything you want to exclude, while `whitelisting` only specified which are allowed, excluding anything else. According to OWASP (source), developers should be able to choose strong whitelists for structured input, based on the expected data structure. The text validated in the application does not follow any specific structures, but with collaboration with NINA the `regex` pattern chosen for `whitelisting` only includes the characters which is required for normal usage of the web application.

The `validateText` function is used for all text input in the application, except the password field. In addition `validateText` is used when validating the text content of JSON and uploaded files. For validating passwords the function `validatePassword` is used. It is similar to `validateText`, except that it allows special characters frequently used in passwords (source1).

```
/**
 * Validate text using regex to whitelist specific characters
 * @param {string} input - The input string to validate
 * @returns {boolean} - If the input is allowed or not
 */
export function validateText (input) {
  if (!input) {
    return true
  }

  const allowedPattern = /^[a-zA-ZæøåÆØÅ0-9 .,?!\\-_():+%"/*]+$/

  const isTextValid = allowedPattern.test(input)

  if (!isTextValid) {
    addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
      FEEDBACK_MESSAGES.INVALID_TEXT)
  }

  return isTextValid
}
```

Listing O.39: The function `validateText()` from the validation module

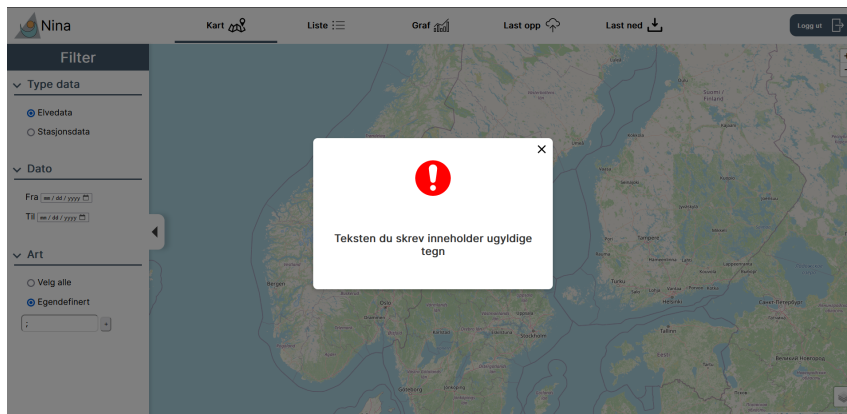


Figure O.14: Error message when user tries to input an invalid character

PostgreSQL input

The environmental JSON data coming from the internal PostgreSQL service at NINA is also assumed to be unsafe. All this data is validated on data format, data type, and on allowed characters. It does this by using the functions `validateRiverWithSpecies`, `validateStationWithSpecies`, `validateRiverSummary`, `validateStationSummary`, and `validateStationDownload` from the custom validation module for each endpoint from PostgreSQL (see Section O for all PostgreSQL endpoints). All of these functions call the function `validateJson` with the JSON data and the schema the JSON should follow.

The schema for each endpoint is defined in a file called `schemas` under `src/constants/`. These are defined to be compatible with AJV, which is the library used to validate the JSON format. The schema for the `river-with-species` endpoint can be seen in listing O.40. Since the endpoint contains an array of objects, the schema `schemaRiverWithSpecies` expects an array with individual objects. The object schema is defined in `schemaSingleRiverWithSpecies`. It specifies what data types are allowed for each property with `properties`. Furthermore, it specifies which properties have to be included with `required`. Lastly, it specifies that the JSON can not include any other properties with `additionalProperties`. This schema results in the validated data from PostgreSQL to always have the correct format and data types.

```
const schemaSingleRiverWithSpecies = {
  type: 'object',
  properties: {
    id: { type: 'integer' },
    name: { type: 'string' },
    pos: coordinatesSchema,
    start_date: { type: 'string' },
    end_date: { type: 'string' },
    species: { type: 'array', items: { type: 'string' } },
    project_id: { type: ['string', 'null'] }
  },
  required: ['id', 'name', 'pos', 'start_date', 'end_date', 'species'],
}
```

```

    additionalProperties: false
  }
  export const schemaRiverWithSpecies = {
    type: 'array',
    items: schemaSingleRiverWithSpecies
  }

```

Listing O.40: The Json schema for the endpoint river-with-species

The function actually responsible for validating the JSON data is called `validateJson` (see listing O.41). This function first uses the function `validateStringsInJson`, which recursively checks all strings in the JSON data using the function `validateText`. If any string in the JSON data is invalid, all the JSON data is invalidated. After checking if the strings does not contain malicious characters, the function uses the AJV function `validate`, which was initialized as `ajv.compile(schema)`, to check if the JSON data follows the schema provided. If the schema is not followed, the user gets an error message either stating that the data from PostgREST is not available, or information about why the uploaded file was not allowed if a file was validated.

```

/**
 * Validate json data against a schema
 * @param {object[]} data - The data to validate
 * @param {object} schema - The schema to validate the data against
 * @param {boolean} excel - If the data is from an excel file or not
 * @returns {boolean} - If the data is valid or not
 */
export function validateJson (data, schema, excel = false) {
  // Prepare ajv and schema
  const ajv = new Ajv()
  const validate = ajv.compile(schema)

  if (!validateStringsInJson(data)) {
    return false
  }

  // Validate data against schema
  if (!validate(data)) {
    // Let user know what is wrong with their data
    const feedbackMessage = excel
      ? `${FEEDBACK_MESSAGES.INVALID_EXCEL_FORMAT} "${validate.errors[0].
        message}" at "${validate.errors[0].dataPath}"`
      : FEEDBACK_MESSAGES.POSTGREST_UNAVAILABLE

    addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
      feedbackMessage)
    return false
  }
}

```



```
    return true
  }
```

Listing O.41: The function `validateJson()` from the validation module

File input

The application also validates all data in the files the user tries to upload. The application checks for any invalid characters, if the file format is correct, and if the data type for the different columns are as expected. In addition to validating the data on the front-end, the upload endpoint implemented by NINA does validation for all files it receives. This is important since validation on the front-end alone can easily be bypassed (source1).

To validate the uploaded files the function `parseAndValidateExcel` is used. The function first checks if the file is defined, if it is the correct type (`.xlsx`), and if the file is under 10MB, giving the user feedback if any of these are not true. It then uses the function `readFile` from the module `fileHandler` to convert the file the user has selected into an `arrayBuffer`, which JavaScript can read. The `ExcelJS` library is used to read the `arrayBuffer` into a workbook object. Each of the three first sheets in the workbook is converted to JSON, using the function `worksheetToJson` from the module `fileHandler`. All the rows in the sheet becomes objects in a JSON array. The function `validateJson` is then called with the JSON data for each sheet and their schema, resulting in the data format, data type, and the characters in strings being validated.

```
/**
 * Parse and validate if the content of an excel file has valid format and
 * content
 * @param {File} excelFile - The file to parse and validate
 * @returns {Promise<boolean>} - A promise which resolves to if the excel file
 * was parsed and validated successfully or not
 */
export async function parseAndValidateExcel (excelFile) {
  try {
    // Check if the file is defined
    if (!excelFile) {
      addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
        FEEDBACK_MESSAGES.NO_FILE_SELCTED)
      return false
    }

    // Check if the file is an excel file
    if (!['.xlsx'].includes(excelFile.name.slice(excelFile.name.lastIndexOf('.')
      ))) {
      addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
        FEEDBACK_MESSAGES.UNSUPPORTED_CONTENT_TYPE)
      return false
    }
  }
}
```

```
}

// Check if the file size exceeds 10 MB
if (excelFile.size > 10 * 1024 * 1024) {
  addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
    FEEDBACK_MESSAGES.CONTENT_TO_LARGE)
  return false
}

// Read the file
const fileContent = await readFile(excelFile)
const workbook = new ExcelJS.Workbook()
await workbook.xlsx.load(fileContent)

// River, station and observation sheets
const sheets = workbook.worksheets.slice(0, 3)

// Validate each sheet
for (const worksheet of sheets) {
  // Convert the sheet to json
  const jsonSheet = worksheetToJson(worksheet)

  // Check if the sheet name is valid
  if (!excelSchemas[worksheet.name]) {
    addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
      FEEDBACK_MESSAGES.INVALID_EXCEL_FORMAT)
    return false
  }

  // Validate the json sheet against its schema
  if (!validateJson(jsonSheet, excelSchemas[worksheet.name], true)) {
    return false
  }
}

return true
} catch (error) {
  addFeedbackToStore(FEEDBACK_TYPES.ERROR, FEEDBACK_CODES.FORBIDDEN,
    FEEDBACK_MESSAGES.GENERIC)
  return false
}
}
```

Listing O.42: The function `validateText()` from the validation module

Appendix P

Meeting Minutes

Notater møte 10.01.2024

Gå gjennom det vi lærte i lynkurset i felleskap og diskutere noen problemstillinger som roller og språk

Lagde dokumenter som skal brukes i løpet av prosjektet, det vil si rapport, prosjektplan, to-do liste og møtereferat dokument

Finne på spørsmål til møtene med veileder og NINA i morgen

Brukte tiden på å se igjennom presentasjonen som nettop ble presentert i seminaret.

Startet fra side 5

Antall timer som er forventet å bli lagt inn i ua er ca. 35 timer.

Skrevet ned spørsmål som skal spørres under møtet med prosjekteier og veilederene under de to møtene som er fastsatt.

Laget to Overleaf prosjekter som alle medlemmene er med i.

Relevant formatering og noen dokumenter ble laget.

Laget Google Docks mappe som er delt med alle medlemmene.

Fleire relevante dokumenter ble laget.

Satser på "A", fornøyd med b.

Tok en gjennomgang av Prosjektplan malen, men ser nærmere på dette i morgen.

Skal også se på utviklingsmodell vi skal gå for.

Skal også se nærmere på

Bærekraft

14 - Livet i havet

Nettside som kan sjekke hvor miljøvennlig nettsiden er.

Spørre om å få tilgang til SkyHigh ressurser.,

Arbeidsoppgaver til neste møte:

Lese en prosjektplan.

Større arbeidsoppgave:

Lese på en bachelor oppgave.

Notater møte NINA 11.01.2024

signering av prosjektavtale

funksjoner som vi skal ha, funksjoner vi bør ha, funksjoner som vi kan ha

Hvordan skal data visualiseres?

funksjonelle funksjoner og sikkerhet funksjoner

Skal vi ha innlogging/autentisering?

inkludere sikkerhet, skalerbarhet, robusthet i oppgaven (og dermed kanskje litt mer backend, som deployment)

Skal nettsiden være mobilvennlig (hvis det bare skal brukes akademisk)?

rammer rundt hvilke teknologier vi kan bruke

avgrensinger på hvor mye av backend vi skal gjøre

Skal vi definere SQL views, evt hva annet (kan dette gjøres via postgres)?

hvor ofte møtes skal vi ha møter (annenhver uke)?

Ble også spurt om de trenger access til skyhigh

Spør NINA om hvilket språk som burde brukes (vi foretrekker engelsk), de hadde ikke noen ytterlige formeninger.

Avtalte faste møtetidspunkter kl 14:00 på onsdager.

Noe konfidensialitet?

Bli kjent med alle.

Frist for innlevering av rapport 21 mai

Sluttproduktet:

- Database og back-end fikser Nina

- Front-end er det de trenger hjelp med

 - Båtel-fiske data.

 - Dataene er registrert på individnivå.

 - Strekning, start og slutt.

 - Kart over hvor de har vært, hvor har de god dekning, hvor er det hull.

 - Tidsoppløsninger for hvor de har vært i ulike tider av året, for eksempel.

 - Ønsker å trykke på et punkt på en strekning for å få artene og størrelsesfordelingen til fiskene som har blitt fanget opp i denne strekningen.

 - Samle alt av data på en plass/ intern database, og standardisere hvordan data legges inn og lagres, dette er noe som Nina fikser internt.

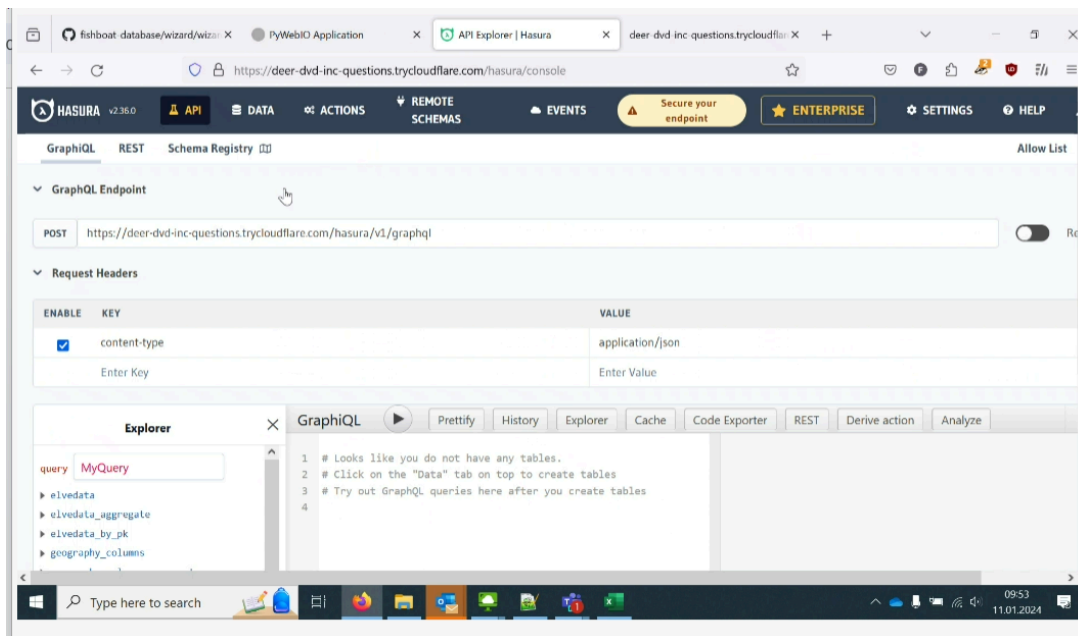
De har laget et testdatasett, standardisert mal som vi kanskje kan komme med endringer til.

Database strukturen:

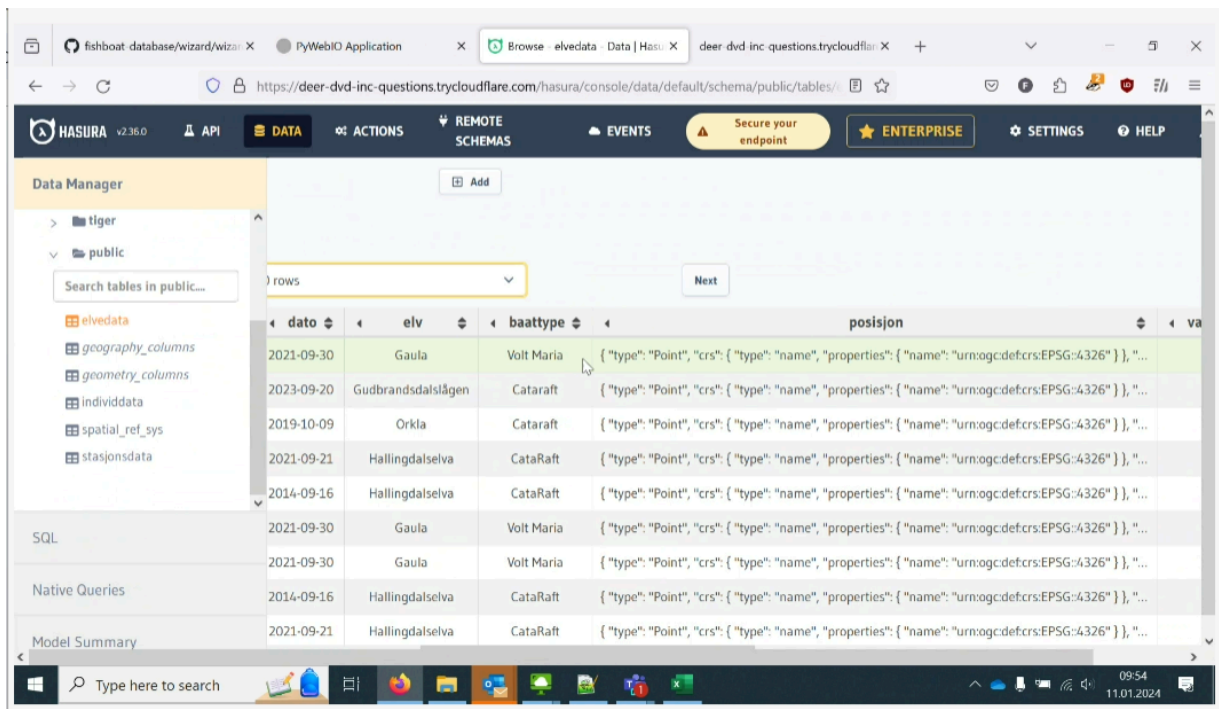
```
Code Blame 49 lines (49 loc) · 1.88 KB Raw
1 version: "3"
2 services:
3 postgres:
4 image: postgres/postgis:16.3.4
5 #command: "-c log_statement=all"
6 environment:
7 POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
8 volumes:
9 - pgdata:/var/lib/postgresql/data
10 dbmate:
11 build: db
12 environment:
13 DATABASE_URL: postgres://postgres:${POSTGRES_PASSWORD}@postgres:5432/postgres?sslmode=disable
14 graphql-engine:
15 image: hasura/graphql-engine:v2.36.0
16 restart: always
17 environment:
18 ## postgres database to store Hasura metadata
19 #command: "-c log_statement=all"
20 #command: "-c log_statement=all"
21 #command: "-c log_statement=all"
22 #command: "-c log_statement=all"
23 #command: "-c log_statement=all"
24 #command: "-c log_statement=all"
25 #command: "-c log_statement=all"
26 #command: "-c log_statement=all"
27 #command: "-c log_statement=all"
28 #command: "-c log_statement=all"
29 #command: "-c log_statement=all"
30 #command: "-c log_statement=all"
31 #command: "-c log_statement=all"
32 #command: "-c log_statement=all"
33 #command: "-c log_statement=all"
34 #command: "-c log_statement=all"
35 #command: "-c log_statement=all"
36 #command: "-c log_statement=all"
37 #command: "-c log_statement=all"
38 #command: "-c log_statement=all"
39 #command: "-c log_statement=all"
40 #command: "-c log_statement=all"
41 #command: "-c log_statement=all"
42 #command: "-c log_statement=all"
43 #command: "-c log_statement=all"
44 #command: "-c log_statement=all"
45 #command: "-c log_statement=all"
46 #command: "-c log_statement=all"
47 #command: "-c log_statement=all"
48 #command: "-c log_statement=all"
49 #command: "-c log_statement=all"
```

Global identifiers blir byttet i databasen?

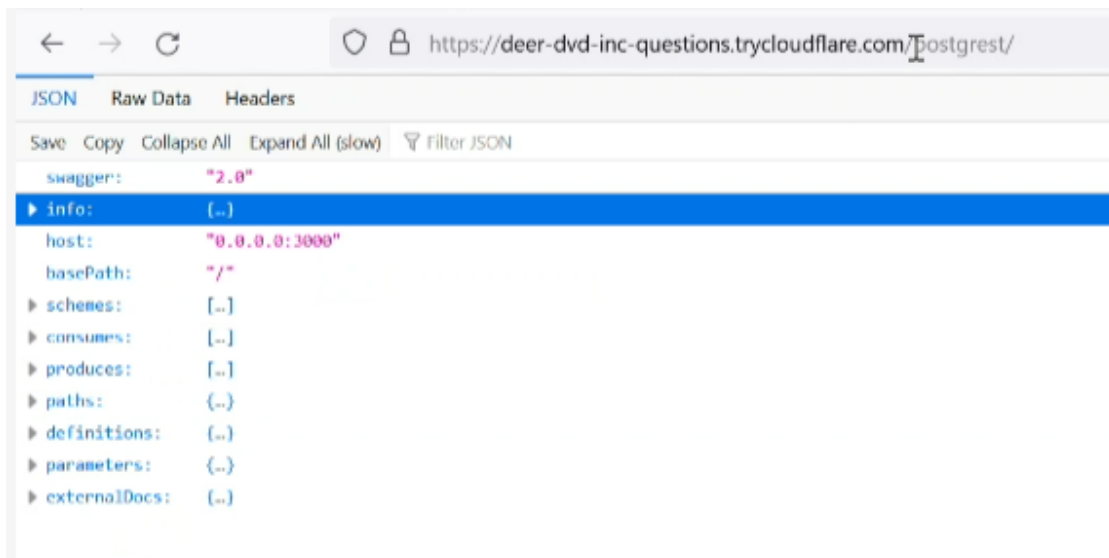
Litt usikker.



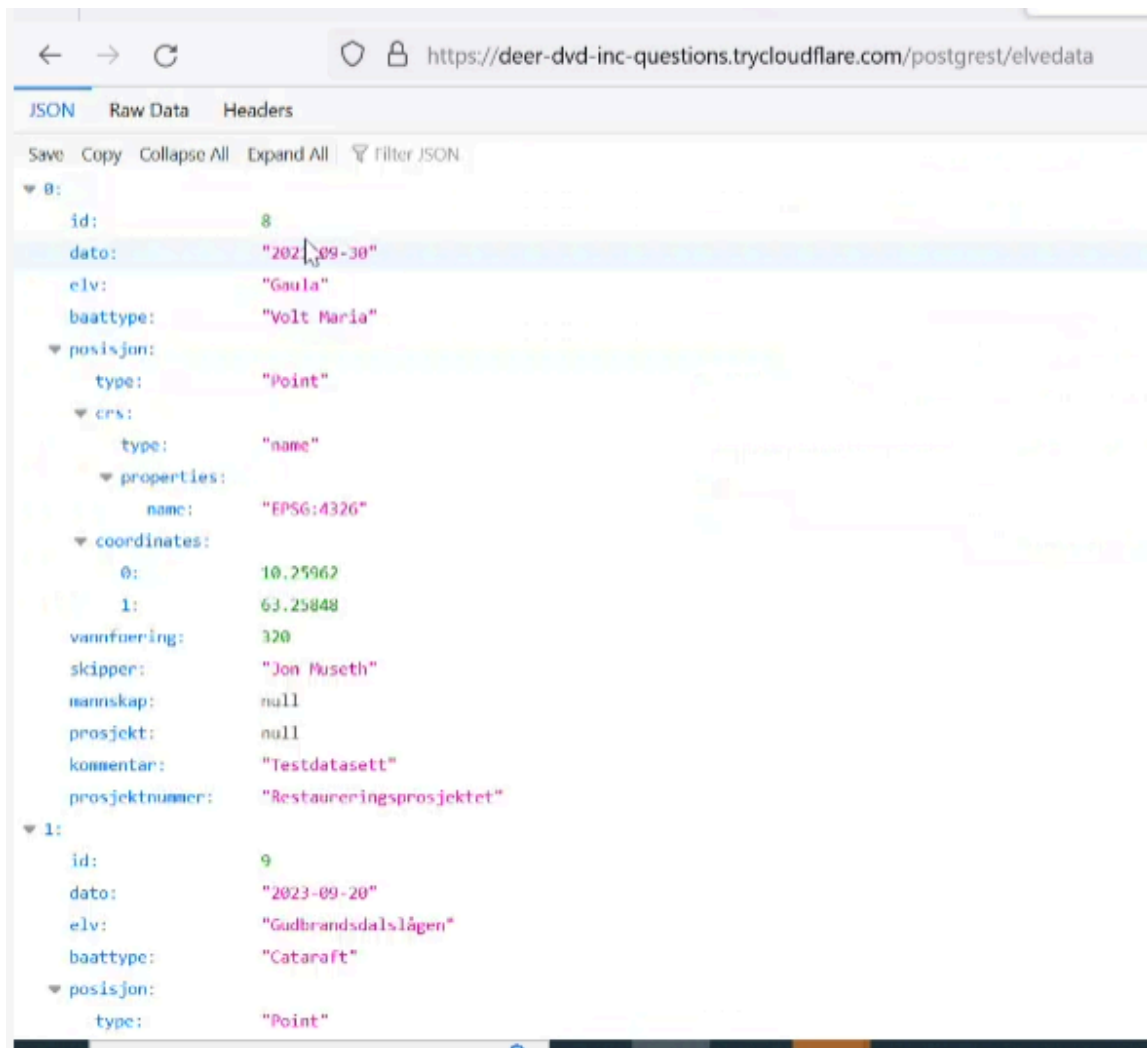
Plasseres på toppen av databasen med REST API



Samme informasjon som de har i databasen.



Database med REST API - ganske fleksibel.



Alle resultater kommer opp som JSON data.

Kan i ettertid filtrere dette.

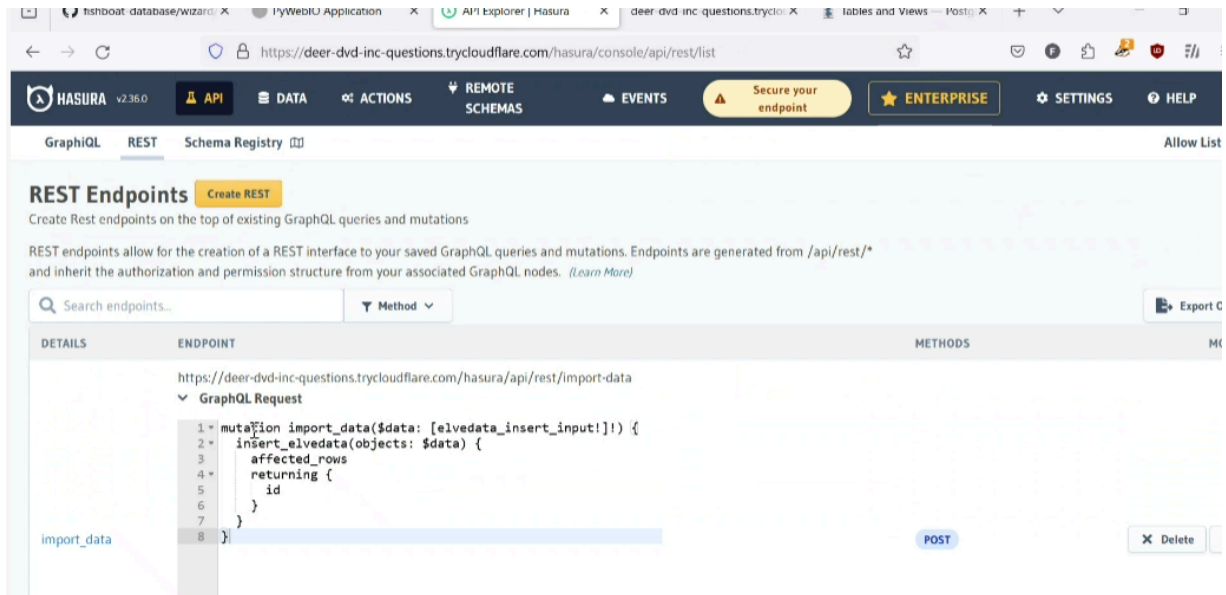
Det ser ut til at vi for det meste bare skal fokusere på front-end.

SQL view

Transparent view

Bruker PostgREST for å lese data.

Han har laget en REST wrapper, hva er dette? (Les mer på det)



For å endre data?

GrafQL for å laste opp data

Møter.

- De fikser felles teams kanal.

- Møte annen hver uke.

- Får til møte Onsdag 14:00.

- Vi kan inkludere - Innlogging og autentisering (passordbeskyttelse)

- Alle som skal ha tilgang til dataene får en bruker som er knyttet opp mot NINAs allerede eksisterende kontoer.

 - Dette er noe vi kan bygge videre på om vi ønsker det.

 - Slik det er nå er det ikke god sikkerhet på tjenesten.

Det ser ut til at de fikser back-end-en slik som loadbalancing, så de kommer kun til å ta i bruk front-end delen av containeren vi sender dem. (Usikker)

Ikke en prioritet at programmet skal være mobilvenlig.

- Kan gjøre dette om vi får tid til det.

Hvordan dataen skal visualiseres.

- Ønsker å sammenlikne fiskesamling

- Lengdefordeling på fisken som er funnet i elven.

 - Frekvenstabeller over ulike lengder på fisken.

- Kommer med et forslag på dette innen de kommende ukene, og kommer derfor tilbake til dette.

 - Ønsker å raskt visualisere forskjellene mellom flere stasjoner.

Ønsker å kunne laste ned dataene fra en til flere plasser på samme tid og få det ned i en .csv fil for eksempel.

Ønsker å vite lengden som er kjørt, visualisering med en strek for hvor man har vært. En rett strek fra start til slutt.



Ønsker en slik representasjon.

Kunden ønsker å få tilgang til det vi jobber med på SkyHigh.

Vi bestemmer språk som skal skrives på. (Engelsk)

Det ser ikke ut til å være noe konfidensielt i form av dataene som er samlet inn.

De er med på å se over hva det er de ønsker å være med i programmet og hva som er "nice to have" i programmet.

Kunden ønsker å få tilgang til GitHub-en.

Kontrakten blir sendt via e-post eller over Teams kanalen.

Notater møte Veiledere 11.01.2024

Litt repetisjon av det vi snakket om i går.

Mye frontend er det vi skal jobbe med.

Vi skal vise NINA hva vi har jobbet med.

Kildebruk - Husker ikke helt hva den heter, men se på tidligere bacheloroppgaver. (Ikke fotnoter)

Bruk overleaf, kan endre kildetype senere.

Kildehenvisning til artikler og websider, ikke så mye til youtube videoer. Refererer heller til guider fremfor videoer.

Sjekk ut OWASP.

Bruk CI/CD - koden testes automatisk.

De kan ha gammel autentisering, sjekk dette opp og spør NINA, og sjekk om dette kan eksponeres for internett.

Ikke glem at vi ikke er utviklere med at vi også skal tenke på sikkerhet

Hyggelig at den er

WCAG

Sjekk nettsider for kontrast og slikt.

Rapporten er det vi blir vurdert på.

Problemstilling - når det er snakk om nettsider.

Sjekk litt på tidligere oppgaver på

Supply chain.

Hvor kommer programmer vi bruker fra?

Hvor komemr biblioteket vi har hentet ting fra?

Sjekk at basefilen ikke er "compromised" ved bruk av docker?

Sjekk ut compendiet til Erik

Modifiser modellen "Scrum" etter eget bruk.

Opprett felles kanal

Møter fast fra 11:30 - 12:00 (T540)

Neste uke 18.01.2024 - T405

<https://www.ieee.org/> mulig å finne vitenskaplige artikkler her.

Google scolar.

BRUK NTNU VPN!

Skriv gjerne på engelsk

Møtoreferater og timeliste skal legges til som vedlegg.

Dette kan være på norsk, har ikke så mye å si.

22,5/60 studiepoeng

Ca. 32 - 35 timer i uka.

Om vi får tid kan vi bruke tiden på å forbedre eller kanskje lage brukertester av prototypen for eksempl.

Gjøre den mobilvennlig.

KI bruk - si ifra om vi bruker det.

Ser mer på dette underveis.

Bruk Copilot (GitHub) for alt det er vært.

Språkvask med KI er ikke juks.

Engelsk språket vektlegges ikke 100%

Hvilken kildehenvisning apa 7?

Hva syntes dere om oppgaven/tenker de at vi burde gå mer i dybden på sikkerhet/deployment

Hvor viktig er design kontra koding/funksjonalitet

Hvor mye av karakteren er nettsiden i forhold til rapporten

Hva burde vi prioritere framover (nevne ideene/planen vår)

Avtale faste møtetidspunkter

Hvilke prosessrammeverk anbefaler de (iterativ eller sekvensiell)

Er begge veiledere for oss, eller bytter det på.

Hvem burde vi kontakte for spørsmål (begge?)

Kan de forklare de ulike avtalene som skal signeres

Skal vi dokumentere hver gang vi bruker ki

Skal vi dokumentere yt-videoer vi ser på

burde vi prioritere kurs/bøker/nettsider/stackoverflow/ytvideo/ki som kilder

hvilket språk vi skal bruke

hva mentes med "bruk biblioteket sine databaser og kurs"

Emnebeskrivelse sier at vi skal utarbeide og formulere problemstilling som del av vurdering, hva vil dette være i vår kontekst

(er det bare å utvikle nettside for databasen?)

Notater møte 11.01.2024

Gjennomgår referatene fra møtene vi har hatt i dag med NINA og med veilederene

MÅ se nærmere på "REST wrapper"

Spørsmål til neste møte med prosjekteierene:

Hva er "Transparent viwe"

Hva er "REST wrapper"

Arbeidsoppgaver:

Kavishnayan - Sender mail til NINA for å få dokumentet signert.

Kavishnayan - Send møteinnkalling 17.01.2024 14:00 (Onsdag)

Kavishnayan - Send møtereferatet til Erik og Ernst.

Alle - Fikser GitHub premium

Notater møte 15.01.2024

Gjennogå hva vi har jobbet med over helgen og hvor langt vi har kommet.

Gjennomgått Gantt sjemaet og samarbeidet for å ferdigstille datoene og arbeidsoppgavene som skal være listet opp i skjemaet.

Planlagt faste gruppemøter (mandag, tirsdag og fredag) = 10:00

Onsdag og torsdag kan øtetider variere.

Fordeling av nye arbeidsoppgaver

Martin - Jobbe litt med Overleaf

Kevin - skrive ferdig "Plan for statusmøter og beslutningspunkter i perioden"

Kavishnayan - skrive ferdig "Scrum delen" og lese og forstå programmet "Jira".

Carl Petter - skrive ferdig ...

Senere på dagen:

Blir ikke veileder møte denne uken, men send melding om det er spørsmål.

Notater møte 17.01.2024

Presentasjon de viste oss.

Økende mengder data...

- Innsatsen genererer mye data og vi jobber (endelig) med en databasestruktur for å lagre data på en sikker måte
- Hierarkisk struktur: sted, tid, miljøforhold og enkeltfisk
- Eksempel: undersøkelser på forskjellige områder i Glomma i flere år

The image shows two overlapping Excel spreadsheets. The top spreadsheet, titled 'Stasjonsdata', has columns for station data including date, clock start, lat/long start/stop, river name, and environmental variables like water level, temperature, and flow. The bottom spreadsheet, titled 'Individdata', has columns for individual fish data including ID, station, passage, type, length, count, sex, age, and whether it was sampled.

Stasjon	Dato	Klokkeslett start	Lat start	Long start	Lat stopp	Long stopp	Dimensjonerte elvertyp	Vær	Vannnivå (Celsius)	Lufttemperatur (Celsius)	Transekthengde (m)	Sekunder fisket (s)
1	20.09.2023	11:12	61.27026	10.40249	61.27026	10.40247	Stryk	Sol	10	15	250	517
2	20.09.2023	11:52	61.26837	10.39834	61.26781	10.40004	Glattstrom	Opphi	11	15	200	515
3	20.09.2023	12:12	61.26550	10.40598	61.26427	10.40794	Innsjø	Sne	12	15	400	370

ID	Stasjon	Omgang	Art	Lengde	Antall	Kjønn	Alder	Gjenutsatt (ja/nei)	Prøvetatt (ja/nei)	Kommentar
1	1	1	1 Ørret	208	1			ja	Ja	Skjell
2	2	1	1 Ørret	168	1			ja	Ja	Skjell
3	3	1	1 Ørret	137	1			ja	Ja	Skjell
4	4	1	1 Ørret	165	1			ja	Ja	Skjell
5	5	1	1 Ørret	184	1			ja	Ja	Skjell

Hierarki

- Elv > Stasjon > Individ (enkeltfisk)

ID	Stasjon	Omgang	Art	Lengde	Antall	Kjønn	Alder	Gjenutsatt (ja/nei)	Prøvetatt (ja/nei)	Kommentar
1	1	1	1 ørret	208	1			ja	ja	Skjell
2	2	1	1 ørret	168	1			ja	ja	Skjell
3	3	1	1 ørret	137	1			ja	ja	Skjell
4	4	1	1 ørret	165	1			ja	ja	Skjell
5	5	1	1 ørret	184	1			ja	ja	Skjell

En tur med ennskepaten registreres som en rad i fanen Elvedata, her oppgis koordinatene for det øverste punktet.

- Det kjøres flere transekter på en tur. Hvert enkelt transekt registreres som en rad i fanen Stasjonsdata
- På hver stasjon fanges og registreres enkeltfisk. Hvert individ registreres med en egen rad i fanen Individdata.
- (vi bruker Elvedata og Undersøkelse om hverandre. Det samme gjelder Stasjon og Transekt, og Individ og enkeltfisk.)

Transekt og stasjon brukes om hverandre.

Måler lengde DNA og slikt fg registrerer hvert individ for hver rad.

Forskjell ppå tur og stasjon:

En tur er en dag, men man kjører igjennom en eller flere stasjoner.

Hvert transekt er på et eget geografisk sted.

Hierarki og struktur

- Startpunkt og antall transekter kan variere fra tur til tur, selv om vi undersøker samme elv.
- Vannføring, tidspunkt, oppdrag...
- Derfor registrerer vi en ny undersøkelse i samme elv (for eksempel Glomma) med en ny rad for hvert nye oppdrag (for eks. Glomma 2020 og Glomma 2021 osv.)
- Følgelig blir stasjons- og individdata knyttet til dette øvre nivået: Stasjon nr. 1 i Glomma i 2020 er ikke den samme som Stasjon nr.1 i Glomma i 2021 og enkeltfisk knyttes videre til disse.



Noen ganger ser vi på hoveddelven, men andre ganger ser man på mindre elver i denne elven.

10 - 20 transekter på en dag.

10 - 20 km kjøres med båten, men måler kanskje bare 3 km. Dette er transektet som skal tegnes opp på nettsiden ikke båt kjøre lengden.

Litt forskjellig spor hver gang,

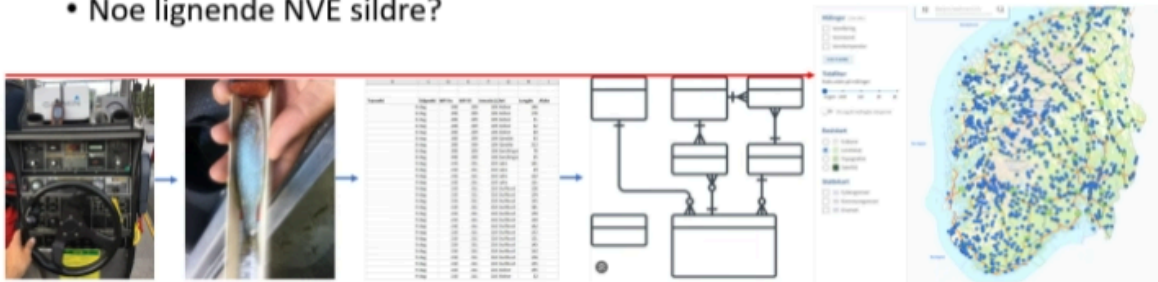
Farge kodede transketer.

...og vi trenger litt hjelp på frontend

- Frontend som gjør det mulig å

- 1) enkelt importere data til databasen
- 2) få en visuell oversikt over hvilke data som ligger i databasen
- 3) interagere med databasen (eks. til nedlasting av data, aggregering av data etc.)

- Noe lignende NVE sildre?

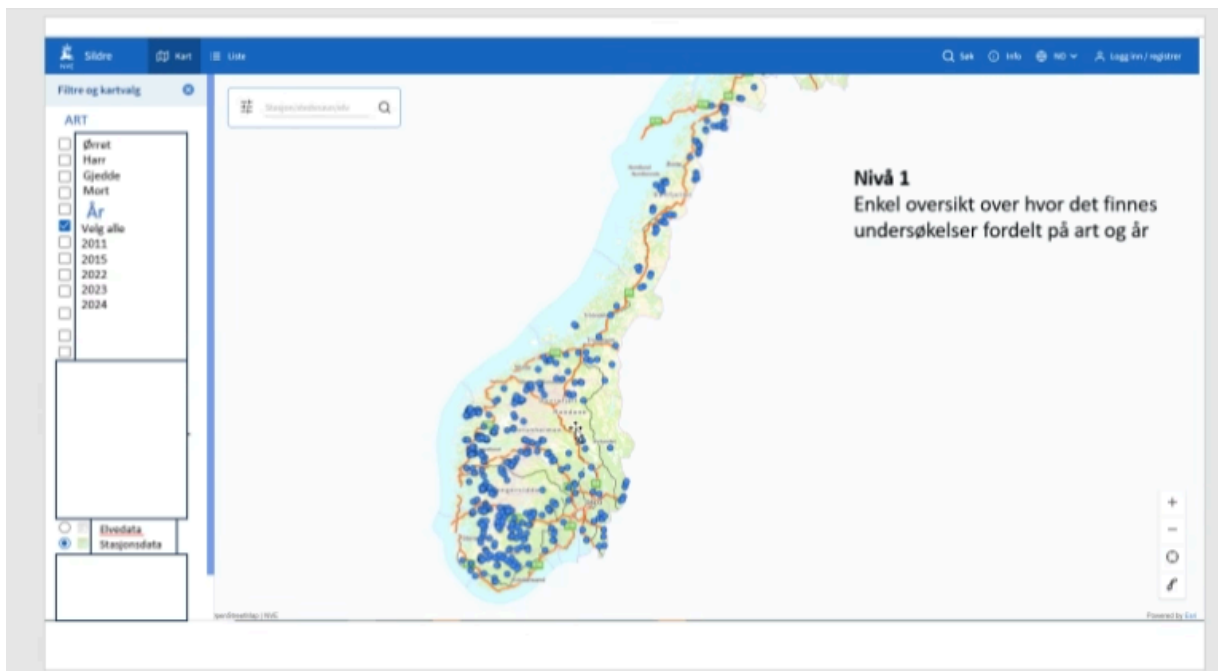


Det de ser for seg at skal bli laget (primær funksjonalitet)

Prioriteringsliste

- 1. Kart som viser alle gjennomførte undersøkelser (elv og stasjon) som punkter i kartet (Nivå 1)
 - Mulighet til å vise punkter på stasjonsnivå og elvenivå. Elvenivå har ett punkt per undersøkelse og dermed færre punkter enn stasjonsnivået og blir derfor litt kanskje ryddigere.
 - Filtrere vekk data ved å huke av på «År» og «Art». Mulighet til å velge flere.
- 2A. Sammendrag av hele undersøkelsen ved markering av et «Elvedata»-punkt (Nivå 2A)
 - Ved markering av en elvedatapunkt: Få opp et sammendrag fra hele undersøkelsen (i en elv i et år), dvs. alle stasjoner og enkeltfisk knyttet til undersøkelsen.
 - Mulighet til å laste ned alle tilhørende data (elv, stasjon og individ) fra det aktuelle elfisket.
- 2B. Ved markering av en stasjon (Nivå 2B)
 - Ved markering av en stasjon: Få opp stasjonsinfo hentet fra rådata på høyre side + en tabell som viser antall fisk fordelt på art.
 - Lenke til «Stasjonsdata» – åpner ny side.
- 3. Ved klikk på stasjonsdata → pop-up side(?) med mer utfyllende data og valgmuligheter.
 - Mulighet til å kunne velge ut hva man ønsker å plote: stasjon, art, innsats (antall sekunder fisket)
 - Må kunne aggregere ved valg av flere stasjoner
 - Frekvensfordeling av antall fisk for valgte stasjoner (evt. per minutt).
 - Lengdefordeling for valgte arter
 - Justerbare intervaller, 5mm bins som standard.
 - Mulighet for å sammenligne (ikke aggregere) flere stasjoner(?)

Prioriteringsliste over tingene de ønsker å ha med i nettsiden som skal lages.



Tatt utgangspunkt i den "Sildre" siden.

Elvedata vil vi ha færre av enn stasjonsdata.

Også klassifisering etter art og år lik at de kan se på "ørret" i en eller flere elver fra ett eller flere år.

Prioriteringsliste

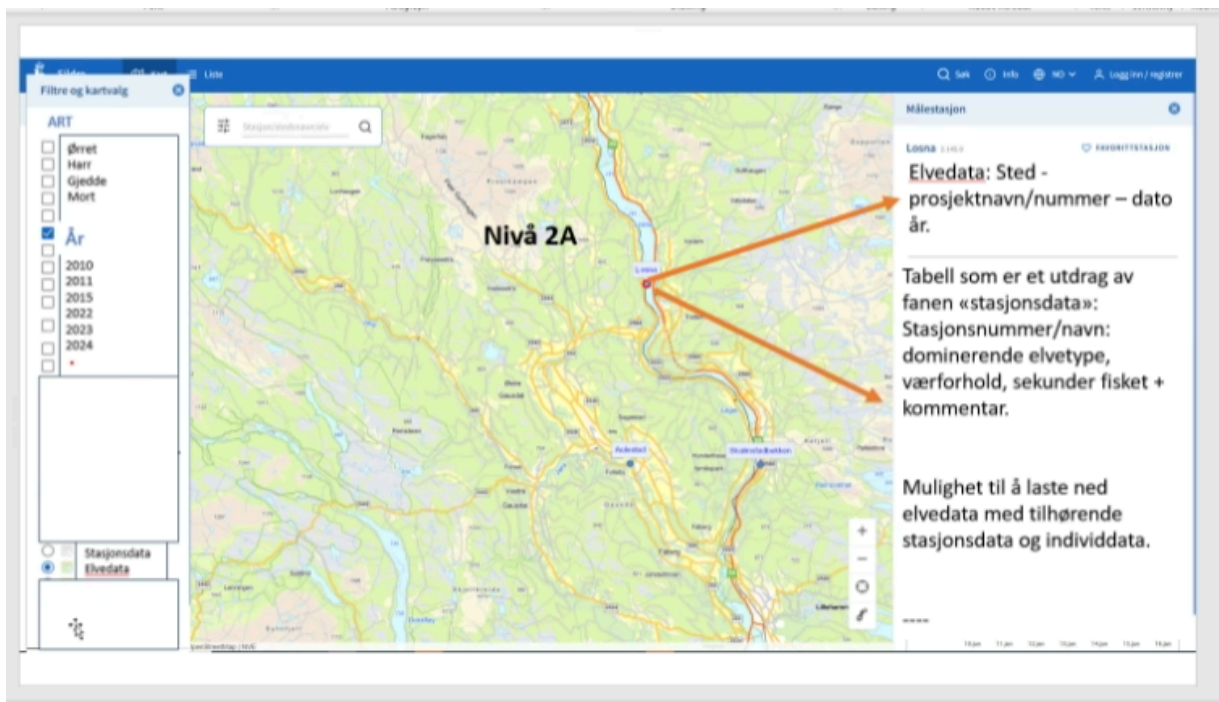
- 1. Kart som viser alle gjennomførte undersøkelser (elv og stasjon) som punkter i kartet (Nivå 1)
 - Mulighet til å vise punkter på stasjonsnivå og elvenivå. Elvenivå har ett punkt per undersøkelse og dermed færre punkter enn stasjonsnivået og blir derfor litt kanskje ryddigere.
 - Filtrere vekk data ved å huke av på «År» og «Art». Mulighet til å velge flere.
- 2A. Sammendrag av hele undersøkelsen ved markering av et «Elvedata»-punkt (Nivå 2A)
 - Ved markering av en elvedatapunkt: Få opp et sammendrag fra hele undersøkelsen (i en elv i et år), dvs. alle stasjoner og enkeltfisk knyttet til undersøkelsen.
 - Mulighet til å laste ned alle tilhørende data (elv, stasjon og individ) fra det aktuelle elfisket.
- 2B. Ved markering av en stasjon (Nivå 2B)
 - Ved markering av en stasjon: Få opp stasjonsinfo hentet fra rådata på høyre side + en tabell som viser antall fisk fordelt på art.
 - Lenke til «Stasjonsdata» – åpner ny side.
- 3. Ved klikk på stasjonsdata → pop-up side(?) med mer utfyllende data og valgmuligheter.
 - Mulighet til å kunne velge ut hva man ønsker å plote: stasjon, art, innsats (antall sekunder fisket)
 - Må kunne aggregere ved valg av flere stasjoner
 - Frekvensfordeling av antall fisk for valgte stasjoner (evt. per minutt).
 - Lengdefordeling for valgte arter
 - Justerbare intervaller, 5mm lins som standard.
 - Mulighet for å sammenligne (ikke aggregere) flere stasjoner(?)

Nivå 2.

Agrigerte data knyttet til denne/ disse undersøkelsene.

Basert på stasjon eller individ.

Elfiske i glomma 2023.



Dette er det de ønsker å se.

Få ut informasjonen fra to stasjoner/ de stasjonene i orsådet?

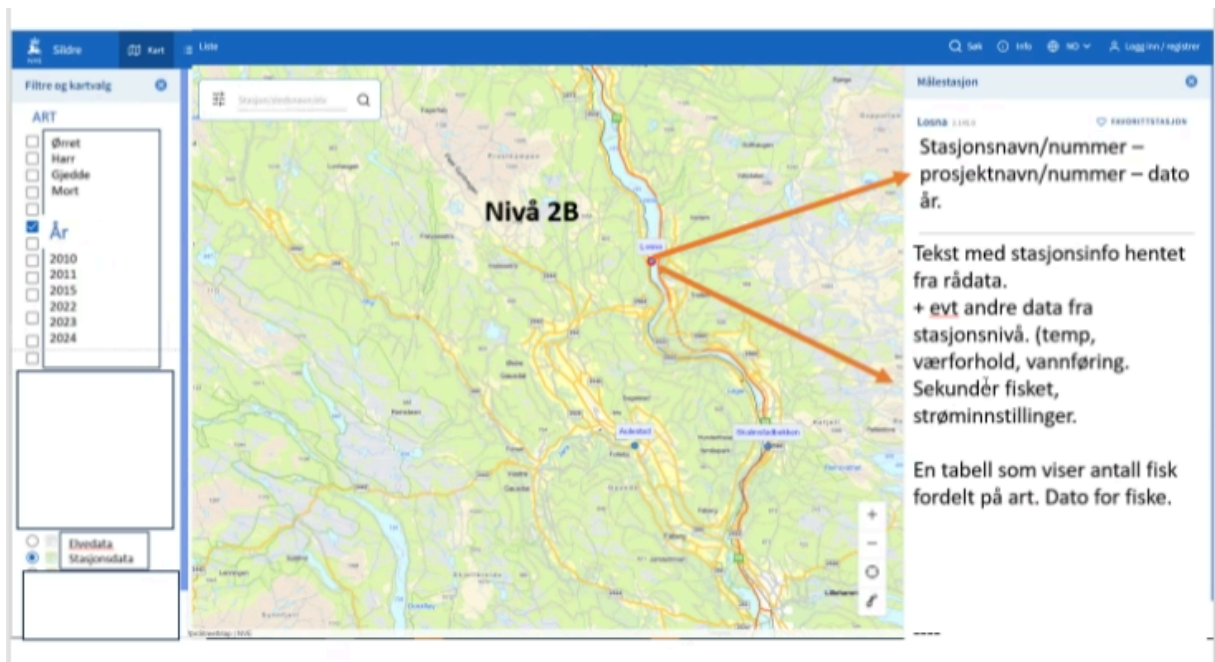
Dataene skal ligge som kolumnen i import malen så ikke tenk på dataene.

Prioriteringsliste

- 1. Kart som viser alle gjennomførte undersøkelser (elv og stasjon) som punkter i kartet (Nivå 1)
 - Mulighet til å vise punkter på stasjonsnivå og elvenivå. Elvenivå har ett punkt per undersøkelse og dermed færre punkter enn stasjonsnivået og blir derfor litt kanskje ryddigere.
 - Filtrere vekk data ved å huke av på «År» og «Art». Mulighet til å velge flere.
- 2A. Sammendrag av hele undersøkelsen ved markering av et «Elvedata»-punkt (Nivå 2A)
 - Ved markering av en elvedatapunkt: Få opp et sammendrag fra hele undersøkelsen (i en elv i et år), dvs. alle stasjoner og enkeltfisk knyttet til undersøkelsen.
 - Mulighet til å laste ned alle tilhørende data (elv, stasjon og individ) fra det aktuelle «fisket».
- 2B. Ved markering av en stasjon (Nivå 2B)
 - Ved markering av en stasjon: Få opp stasjonsinfo hentet fra rådata på høyre side + en tabell som viser antall fisk fordelt på art.
 - Lenke til «Stasjonsdata» – åpner ny side.
- 3. Ved klikk på stasjonsdata → pop-up side(?) med mer utfyllende data og valgmuligheter.
 - Mulighet til å kunne velge ut hva man ønsker å plote: stasjon, art, innsats (antall sekunder fisket)
 - Må kunne aggregere ved valg av flere stasjoner
 - Frekvensfordeling av antall fisk for valgte stasjoner (evt. per minutt).
 - Lengdefordeling for valgte arter
 - Justerbare intervaller, 5mm bins som standard.
 - Mulighet for å sammenligne (ikke aggregere) flere stasjoner(?)

Hvor mange ørret, hvor mange (andre arter) ...

SKal åpne ny side herifra?



Slik skal det se ut.

Ønskeren tabell ørret 26

Rask oversikt om hva som ble gjort på denne stasjonen i 2022.

Skal kunne velge hva som skal plotts

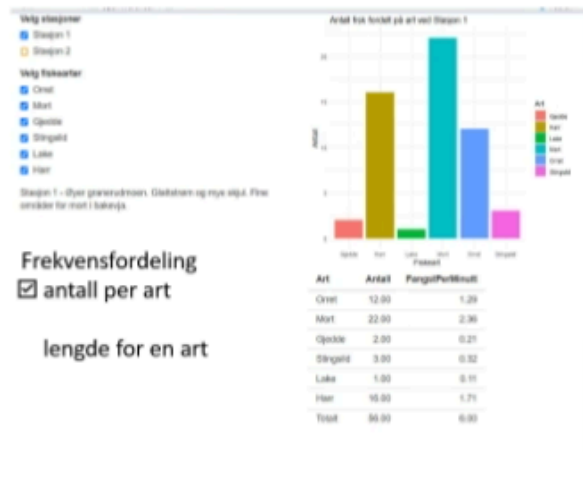
Prioriteringsliste

- 1. Kart som viser alle gjennomførte undersøkelser (elv og stasjon) som punkter i kartet (Nivå 1)
 - Mulighet til å vise punkter på stasjonsnivå og elvenivå. Elvenivå har ett punkt per undersøkelse og dermed færre punkter enn stasjonsnivået og blir derfor litt kanskje ryddigere.
 - Filtrere vekk data ved å huke av på «År» og «Art». Mulighet til å velge flere.
- 2A. Sammen drag av hele undersøkelsen ved markering av et «Elvedata»-punkt (Nivå 2A)
 - Ved markering av en elvedatapunkt: Få opp et sammen drag fra hele undersøkelsen (i en elv i et år), dvs. alle stasjoner og enkeltfisk knyttet til undersøkelsen.
 - Mulighet til å laste ned alle tilhørende data (elv, stasjon og individ) fra det aktuelle elvfisket.
- 2B. Ved markering av en stasjon (Nivå 2B)
 - Ved markering av en stasjon: Få opp stasjonsinfo hentet fra rådata på høyre side + en tabell som viser antall fisk fordelt på art.
 - Lenke til «Stasjonsdata» – åpner ny side.
- 3. Ved klikk på stasjonsdata → pop-up side(?) med mer utfyllende data og valgmuligheter.
 - Mulighet til å kunne velge ut hva man ønsker å plote: stasjon, art, innsats (antall sekunder fisket)
 - Må kunne aggregere ved valg av flere stasjoner
 - Frekvensfordeling av antall fisk for valgte stasjoner (evt. per minutt).
 - Lengdefordeling for valgte arter
 - Justerbare intervaller, 5mm bins som standard.
 - Mulighet for å sammenligne (ikke aggregere) flere stasjoner(?)

Nivå 3

Nivå 3 – åpne egen side for stasjon(er)

- Et eksempel på hvordan en «stasjonsside» kan vise ulike valgmuligheter.
- Ønske:
 - Frekvensfordeling av arter
 - Lengdefordeling av en bestemt art
 - Mulig å velge flere stasjoner enn kun den valgte stasjonen.
 - Enkel nedlasting av valgte variabler



SKal kunne huke av for ulike ting slik som art og lengde, får opp et kollonne diagram for en visuell framstilling.

Ønksr å se på lengdefordelingen for en art, frekvensfordeling i 5mm binds.

En demografi for fisk.

Det viktigste i dette nivået.:

Fordeling av arter.

Frekvensfordelingen av enkelstarter.

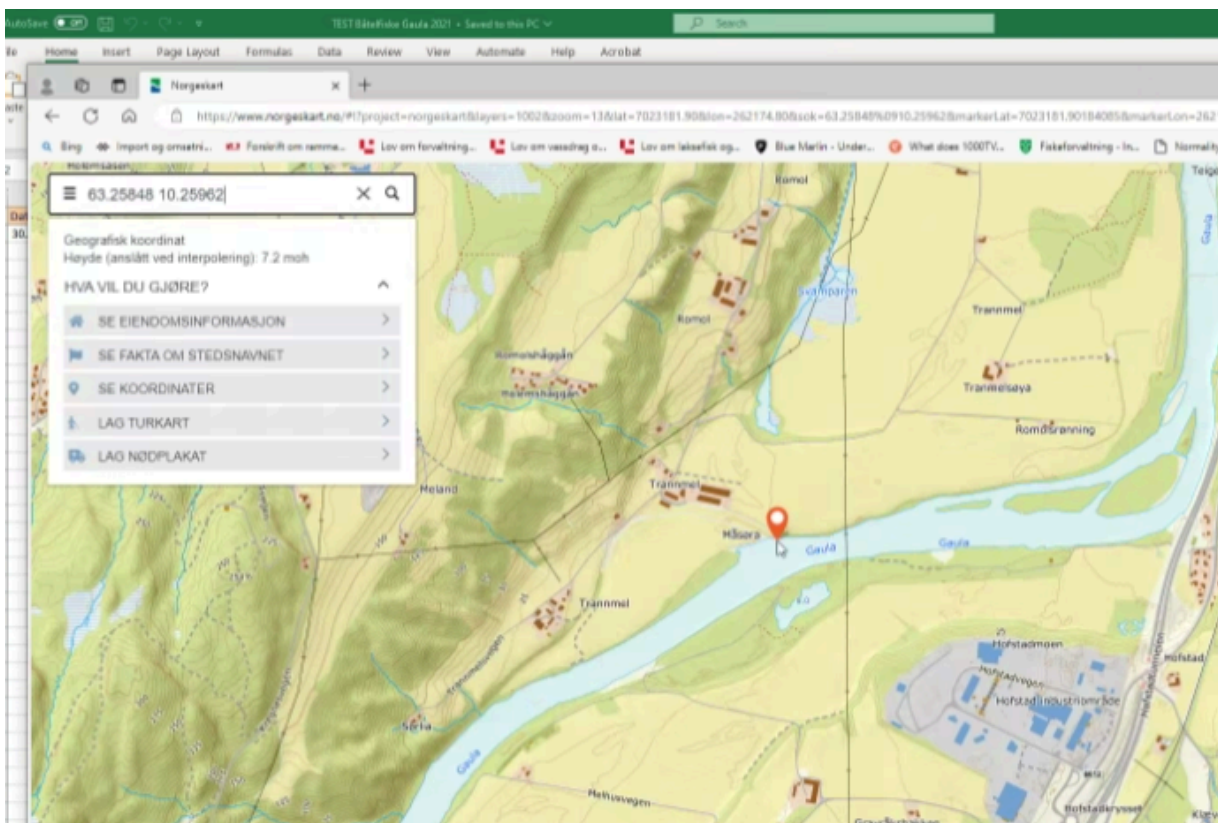
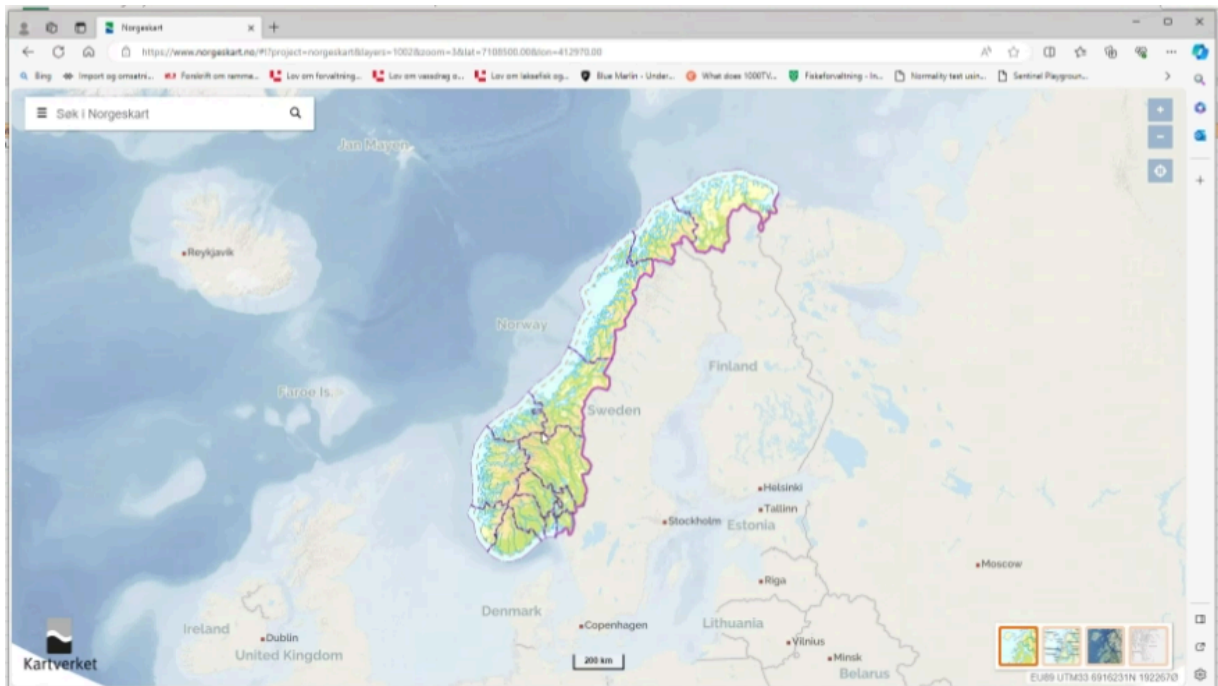
Ønsker å kunne velge flere stasjoner samtidig i det samme vinduet.

Alle astasjonene innen dinne enven.

Om vi får tid kan vi også legge til funksjon for å sjekke stasjonenes data i den gotte elven over flere år.

	A	B	C	D	E	F	G	H	I	J	K	L	
1	Date	Elv	Båttype	Lat	Long	Vannføring (slidre.no)	Skipper	Mannskap1	Mannskap2	Mannskap3	Prosjekt	Prosjektnummer	Kommentar
2	30.09.2021	Gaula	Volt Maria	61.25848	10.25962		320 Jon Museth	Tobias Holter	Erik Lie				Restaureringsprosjekt Testdatasett

De sender oss eksempel data som vi kan se på.



hvor det startet.

<https://www.norgeskart.no/#!?project=norgeskart&layers=1002&zoom=7&lat=6801817.45&lon=335149.67>

A	B	C	D	E	F	G	H	I	J	K	L	
Dato	Elv	Båttype	Lat	Long	Vannføring (l/s)	Skipper	Mannskap1	Mannskap2	Mannskap3	Prosjekt	Prosjektnummer	Kommentar
30.09.2021	Gaula	Volt Maria	63.25848	10.25962		320 Jon Museeth	Tobias Holter	Erik Lie				Restaurering av Testdatasett

En base vil si en prikk i frontend for denne elven.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
Stasjon	Båttype	Dato	Klokkeslett start	Lat start	Long start	Lat stopp	Long stopp	Dominerende elvtype	Vær	Vanntemp (Celsius)	Lufttemperatur (Celsius)	Ladningsvev (µs/cm)	Transekthøgde (m)	Sekunder fisket (s)	Volt	Puls (DC)	Display	Gpx fil?	Stasjonsbeskrivelse	Kommentar
1	Volt Maria	30.09.2021	08:00	63.25848	10.25962	63.25877	10.26344	Glattstrøm	Sne	8	-1	77	550	1000	60				Oppstrøms planlagt Elve-v	
2	Volt Maria	30.09.2021	09:00	63.25826	10.25984	63.25789	10.26170	Glattstrøm	Sne	4	-2	77	568	1000	60				Oppstrøms planlagt Elve-h	
3	Volt Maria	30.09.2021	10:00	63.25856	10.26799	63.25894	10.27621	Styrk	Regn	10	2	77	367	1000	60				Tittaksområde, styrk/Elve-h	
4	Volt Maria	30.09.2021	11:00	63.25922	10.26851	63.25911	10.27604	Styrk	Regn	9	4	77	243	1000	60				Tittaksområde, elvev/Naturf	
5	Volt Maria	30.09.2021	12:00	63.25896	10.27703	63.25936	10.28131	Bakveie	Regn	9	4	77	201	1000	60				Nedstrøms tittaksom/Mest F	

kjører en km ca.

Logger hvor mange sekunder de sender ut strøm. (Relevant informasjon)

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
Stasjon	Båttype	Dato	Klokkeslett start	Lat start	Long start	Lat stopp	Long stopp	Dominerende elvtype	Vær	Vanntemp (Celsius)	Lufttemperatur (Celsius)	Ladningsvev (µs/cm)	Transekthøgde (m)	Sekunder fisket (s)	Volt	Puls (DC)	Display	Gpx fil?	Stasjonsbeskrivelse	Kommentar
1	Volt Maria	30.09.2021	08:00	63.25848	10.25962	63.25877	10.26344	Glattstrøm	Sne	8	-1	77	550	1000	60				Oppstrøms planlagt Elve-v	
2	Volt Maria	30.09.2021	09:00	63.25826	10.25984	63.25789	10.26170	Glattstrøm	Sne	4	-2	77	568	1000	60				Oppstrøms planlagt Elve-h	
3	Volt Maria	30.09.2021	10:00	63.25856	10.26799	63.25894	10.27621	Styrk	Regn	10	2	77	367	1000	60				Tittaksområde, styrk/Elve-h	
4	Volt Maria	30.09.2021	11:00	63.25922	10.26851	63.25911	10.27604	Styrk	Regn	9	4	77	243	1000	60				Tittaksområde, elvev/Naturf	
5	Volt Maria	30.09.2021	12:00	63.25896	10.27703	63.25936	10.28131	Bakveie	Regn	9	4	77	201	1000	60				Nedstrøms tittaksom/Mest F	

“Kver stasjon har individ data over alle individene som ble fanget og hvilken stasjon de ble fanget på.

A	B	C	D	E	F	G	H	I	J	K
ID	Stasjon	Ompgang	Art	Lengde	Antall	Kjønn	Allder	Gjennutsatt (ja/nei)	Prøvetatt (ja/nei)	Kommentar
1										
2	1	1	Ørret	176				ja		Testdata
3	2	1	Ørret	128				ja		Testdata
4	3	1	Ørret	166				ja		Testdata
5	4	1	Ørret	171				ja		Testdata
6	5	1	Ørret	158				ja		Testdata
7	6	1	Ørret	92				ja		Testdata
8	7	1	Ørret	260				ja		Testdata
9	8	1	Ørret	128				ja		Testdata
10	9	1	Ørret	149				ja		Testdata
11	10	1	Ørret	115				ja		Testdata
12	11	1	Ørret	150				ja		Testdata
13	12	1	Ørret	154				ja		Testdata
14	13	1	Ørret	169				ja		Testdata
15	14	1	Ørret	150				ja		Testdata
16	15	1	Ørret	165				ja		Testdata
17	16	1	Ørret	115				ja		Testdata
18	17	1	Ørret	80				ja		Testdata
19	18	1	Ørret	152				ja		Testdata
20	19	1	Ørret	102				ja		Testdata
21	20	1	Ørret	99				ja		Testdata
22	21	1	Ørret	60				ja		Testdata
23	22	1	Ørret	58				ja		Testdata
24	23	1	Ørret	58				ja		Testdata
25	24	1	Ørret	180				ja		Testdata
26	25	1	Ørret	78				ja		Testdata
27	26	1	Ørret	126				ja		Testdata
28	27	1	Ørret	151				ja		Testdata
29	28	1	Ørret	55				ja		Testdata
30	29	1	Ørret	88				ja		Testdata
31	30	1	Ørret	99				ja		Testdata
32	31	1	Ørret	72				ja		Testdata
33	32	1	Ørret	78				ja		Testdata
34	33	1	Ørret	136				ja		Testdata
35	34	1	Ørret	196				ja		Testdata
36	35	1	Ørret	125				ja		Testdata
37	36	1	Ørret	60				ja		Testdata
38	37	1	Ørret	138				ja		Testdata
39	38	1	Laks	118				ja		Testdata
40	39	1	Laks	111				ja		Testdata
41	40	1	Laks	89				ja		Testdata
42	41	1	Laks	113				ja		Testdata
43	42	1	Laks	95				ja		Testdata
44	43	1	Laks	121				ja		Testdata

Har samlet dette i en "pivåtabel" tideligere.

The screenshot shows an empty PivotTable on the left with 'Row Labels' and a 'Grand Total' row. On the right is the 'Choose fields to add to report' task pane. The 'Art' field is selected in the list. The 'Filters' area is empty, and the 'Columns' area contains a blue circle. The 'Rows' area shows 'Stasjon' and the 'Values' area is empty.

Som de filtrere

videre på og får ut det de ønsker.

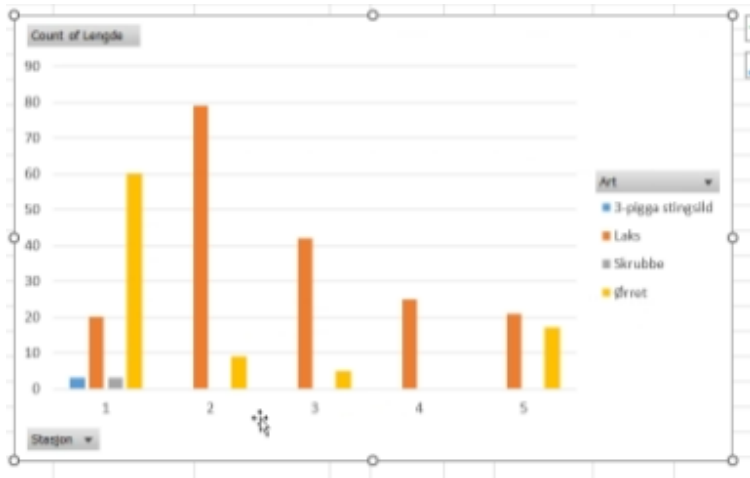
The screenshot shows a PivotTable with data on the left. The task pane on the right shows 'Art' selected in the list, 'Art' in the 'Columns' area, 'Stasjon' in the 'Rows' area, and 'Count of Lengde' in the 'Values' area.

Row Labels	3-pligg stingslid	Laks	Skrubbe	Øvret	Grand Total
1	3	20	3	60	86
2		79		9	88
3		42		5	47
4		25			25
5		21		17	38
Grand Total	3	187	3	91	284

Også laget en graf utifra dette.

Men det er dette de spør oss om å gjøre i nettsiden slik at alt er klart etter få klikk.

Ønsker en slik visuell framstilling.



En frekvensfordeling basert på art.

Ønsker en frekvensfordeling på lengde.

Liste for å velge ulike typer figurer

Velger for en art

SKal fremstille en art om gangen.

Når det kommer til tabellene som skal lages. Så kan det være et valg

Ønsker brukere på to nivåer.

Lav nivå, kan ikke laste den kun se data?

Ikke å ten nettside, skal være i eget intranett, inernt nettverk.

Kan sette opp autentisering slik vi ønsker.

Kong - kobles med azura og postgres

Ikke sett opp egen LDAP server.

Francesco fikser et eksempe, som vi kanskje kan videre utvikle på.

Kan utvikle reste ved behov.

JDVA AS?

Versjonpinning og slikt.

Greit med vedenskart, men kan fikse en løsning hvor startpunktet starter i norge og sverige.

Ønsker å se hvilket år dataene kommer fra.

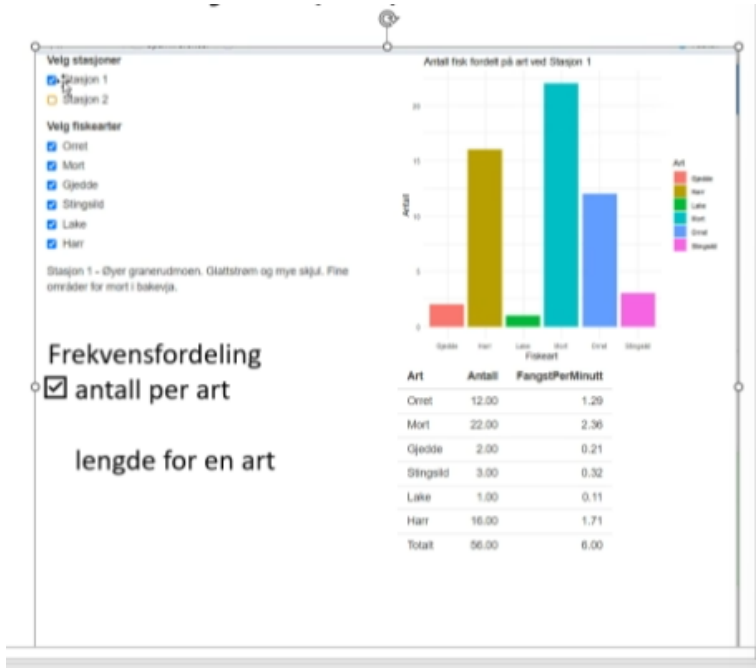
Ønsker å trukke på en elv, få opp stasjonene, velge stasjonen og sammenlikne dataene fra disse stasjonene.

Få opp hvilket år disse dataene fra hvert av stasjonene som sammenlines/ skrives ut blir leistet opp.

Farger på nettsiden:

NINA maler på bannere og slikt.

De senter



Prioriteringsliste

1. Kart som viser alle gjennomførte undersøkelser (elv og stasjon) som punkter i kartet (Nivå 1)
 - Mulighet til å vise punkter på stasjonsnivå og elvenivå. Elvenivå har ett punkt per undersøkelse og dermed færre punkter enn stasjonsnivået og blir derfor litt kanskje ryddigere.
 - Filtrere vekk data ved å huke av på «År» og «Art». Mulighet til å velge flere.
- 2A. Sammendrag av hele undersøkelsen ved markering av et «Elvedata»-punkt (Nivå 2A)
 - Ved markering av en elvedatapunkt: Få opp et sammendrag fra hele undersøkelsen (i en elv i et år), dvs. alle stasjoner og enkeltfisk knyttet til undersøkelsen.
 - Mulighet til å laste ned alle tilhørende data (elv, stasjon og individ) fra det aktuelle elfisket.
- 2B. Ved markering av en stasjon (Nivå 2B)
 - Ved markering av en stasjon: Få opp stasjonsinfo hentet fra rådata på høyre side + en tabell som viser antall fisk fordelt på art.
 - Lenke til «Stasjonsdata» – åpner ny side.
3. Ved klikk på stasjonsdata → pop-up side(?) med mer utfyllende data og valgmuligheter.
 - Mulighet til å kunne velge ut hva man ønsker å plote: stasjon, art, innsats (antall sekunder fisket)
 - Må kunne aggregere ved valg av flere stasjoner
 - Frekvensfordeling av antall fisk for valgte stasjoner (evt. per minutt).
 - Lengdefordeling for valgte arter
 - Justerbare intervaller, 5mm bins som standard.
 - Mulighet for å sammenligne (ikke aggregere) flere stasjoner(?)

Skal nettsiden være på engelsk eller norsk

(kan legge inn valg for språk, og ha begge hvis det er best)

Fordel å ha den på norsk.

Ettersom at alle dataene er på norsk.

Brukermanualen

Ganske åpne på språket.

Kan ha det på engelsk og norsk

Gå gjennom funksjonalitet og design vi har definert, og hva de har definert

Hvordan vi kan implementere autentisering (Hvis de vil ha)

Hvordan skal designet se ut (Like farger som hjemmesiden deres? Lignende Sildre?)

Notater møte 25.01.2024 - NINA

Fikset kommunikasjonskanal over teams som vi kan bruke, NINA kommer til å bruke sine ntnu kontoer.

Ønsker dere at det skal være mulig å fjerne datapunkter, kanskje om dataene er gamle?

Er det noe problem at vi bruker to express servere?

Vise fram det vi har utviklet

Kan gjøre det vi selv ønsker, men rest api-en kommer til å være igjennom postgres. Men vi kan lage et lag på toppen av rest api-ene.

Definer SQL view.

Vi kan fint bruke to docker filer om vi helst ønsker det, men frafra ville bare at vi skulle fokusere mer på frontend og slikt.

Se Api tables and view i git-en hans.

Download funtionalitet.

Har ikke grouping funktionalitet.

Må sikkert skrive queries selv.

Spør han om det er noe som er usikkert når det kommer til dokumentasjonen som er tilgjengelig i git-en.

Ser på prosjektplanen vår.

Stasjon 1 kommer til å være stasjonen som er øverst i vassedraget, ikke forvirr dette.

Vise stasjonsdata samtidig som elvedata.

Etter kundens ønske selv om det kan bli litt uryddig.

Ønnsker kart over sverige også.

De bruker ikke varnish som caching system. Han tror ikke at det kommer til å være flaskehalsen i arkitekturen, blir heller databasen.

Kan kanskje sette opp varnish over hele systemet, men dette er ikke fikset enda.

Ønsker dere at det skal være mulig å fjerne datapunkter, kanskje om dataene er gamle?
Er ikke ønskelig å slette data.

Kanskje flagge data som man kanskje er usikre på.

Sette opp en kolonne til for å flagge data i databasen.

Prioriteter, noen kan kun laste aopp data andre kan kun lese data.

Databasen er oppe å går.

Kommer mer testdata, de skal aktivt legge inn data i databasen de kommende dagene/ukene.

Trenger en ekstra uke for å ferdigstille docker compose.

Det vi skal fikse.

Skal fiske design de kommende ukene som kunden kan teste.

Skal programmere i midten av februar.

Skal gi sem tilgang til skyhigh.

Onsdager kl.14.00 passer fint de fleste dager. (14. februar)

Får testdata fortløpende (etterpå?)

En docker konteiner som skal skjøre programmet.

Varnish kommer til å kjøre på serverene til NINA, vi skal ikke tenke på.

Skriv om caching løsninger i rapporten.

Notater møte 25.01.2024 - Veiledere

Viser prosjektplanen.

kildehenvisning

Bilder - Sjekk om vi kan bruke bildet og kjeldehenvis til hvor det er hentet.

Alle ressurser i referanselista skal være listet aktivt i teksten også.

Skal ha nummererte lister (1.2)

Modeller og tabeller skal forklares med tilstrekkelig tekst (en setning til et avsnitt) når de blir referert til. Alle modeller og tabeller skal bli referert til i teksten.

Resultatmål og grupperegler skal også være nummererte lister.

Skal gjøre det enklere å se tilbake til og referere til.

Rutiner og slikt skal ut av regel delen av grupperegelene.

Snnsynlig å referere til i senere tid, lag nummerert liste.

Trenger vil kilder for bilder til forprosjektplan?

JA

Express og moduler

Hvordan sikre supplychain ettersom at det er så mye å være avhengig av.

Finn det beste verktøyet dere kan.

Bruk github sin supply chain

Sbom fil - for hvor ting er hentet fra (Retningslinjer) for å forbedre sikkerhet. Brukes ikke mye i prosjekter men vi kan se på dette.

Package manager?

Pakkehåndtering sikkerhet, ganske variabelt hos utviklere, men vi må sjekke sikkerheten ettersom at det er hva vi jobber med.

Npm verden er sikker?

Les på github og hva de ser etter når det kommer til supplychain.

Har studentpakke med mer funksjonalitet.

Kan ikke stole på noen, vær skeptisk til det som brukes av pakker og tjenester.

1 - 2 punkter om rutiner

Skal nevnes mer spesifikt at 20 timer jobbing er minimum aktiv tid.

Bilder

Forklar figuren og laget ut fra ... referanse (til bildet/ figuren)

Skal vite hva personen skal få ut av figuren og når vi skal se på figuren.

Mest for tolkningens del.

Ikke referanse til tabellen, men heller til hvor teorien for denne tabellen er hentet ifra. Risikoanalyse. (Se risiko compendiet til gaute).

Utnytt overleaf

Legg inn tabeller selv.

Vektor grafikk skal lages i rapporten.

Screenshots kan være bilder.

Trenger ikke å grunngi hvor programmer og verktøy er hentet ifra.

Å lenge du føler at dette er ditt trenger du ikke å referere til det, hør på magesfølelsen.

Github, script på vm-en som kjøres fra github (workflow). Eller skrive det inn i vm-en?

Mer forsinkelser med ssh forbindelser til vm-en?

Raskere å kjøre scriptet, slik vi har det nå. (Argumenter hvorfor denne måten er valgt).

Forklar om bruken av workflow og parallellisering av oppgaver.

Karakter

Mengden arbeid (ekstra arbeid)

Hvor mye, hvor vanskelig, hvor vitenskaplig er rapporten. Subjektivt opp til sensoren som retter oppgaven.

Ganske presis karakter.

Vi får noen som kan noe om programvare utvikling.

Vurderer fra kriterier som er satt for oss, ingen tulle sensorer.

Lever.

Last ned gitrepoet og lever dette inn.

Conventional commits.

Versions Nummerering

1.0 - 1.1 ...

Viser modellen

Bruker developer og merger med main på enden av hver sprint.

Finn blogger (anerkjent utvikler) og dokumentasjon som kilde

Finn kilde på bruk av

semver.org - Se denne.

Wireframes skal lages og slikt som NINA kan test ut.

Kan vise eksempel nettside også.

Kan man skrive ting i punktform?

- ja, eksemplene vi fikk av gode prosjektplaner hadde det

Burde vi nevne at vi ble inspirert i med tanke på bruk av verktøy til gantt

- ja, Erik sa dette i mailen, men kan kanskje spørre om hvor nøyaktig det er

Hva er karakterkravene, er det nok å fullføre kravene til NINA samtidig som man har en god rapport for å få høy karakter

Notater møte 01.02.2024 - Veiledere

Vise projektplan og planen videre.

Opprette backlog.

Først tenkter vi å ta

- Risk assesment

 - Grunnlag for trethmodeling og usecases

 - Mer konkrete sikkerhetskrav

- Usecases

- Misuse cases

 - Dataflowdiagrams

 - Trethmodeling

- citemap

- Designpresipper og sikkerhetsprinsipper.

- Wireframes

 - Usertest

 - Utvikle videre

 - Figma

 - SKal gjøre det mer detalijert.

 - Nyttig og gratis

 - Mer vanlig å bruke.

- Qualitygates

 - Krav til deg selv for hvor mye sikkerhet som må være implementert for å kunne gå videre til neste steg.

 - Bygger inn sikkerhet hele tiden.

 - Bruker online ressurser for å danne en oversikt over vanlie trussler som er til stedet.

- Data sikkring

 - Minske sjangsen for å laste opp eller laste ned data.

 - Etter autorisasjon.

 - Alt går over en database.

 - Vet ikke hva nettsiden skal brukes til i fremtiden.

- Skal være sikker, men det er jo offentlig data.

- SIKKERHET ER SENTRALT!!!

Nå

- Leser bashelor

- Reperere det vi har lært tidligere som webtek

- Repetere sikkerhetsfag.

Satt opp automatisk testing

Moduler som testes blir fikset også.

Beskriv hvordan vi gjør dette til bachelor oppgaven

Kommit melding

Variabel som viser hvilke filer som er endret på.

Fikser "linting"

Prittyfy

Etter standard js.

Endrer etter beste standard (var til let)

CodeQL, ser etter sikkerhet prinsipper.

De ønsker drt i et git repo

RAPPORTEN ER VIKTIG SER DR HVA SENSOR SER FØRST!!!

SKal inneholde hva dsom kan bli bedre og hva som kunne ha blitt gjort annerledes.

Bruk hele

Februar, kap 1 og 2.

Polert og ferdig til modten av mars.

Spørre om tilbakemeldinger.

Referansebruk og slikt.

Ikke sett av skriving til bar skriving, gjør det underveis.

HUSK DETTE!!!

Vær flink til å skrive, bedre å kaste noe inn i overleaf og notater slik at det blir lett å skrive senere.

Forklar valgene man tar.

Har skrevet underveis, fortsett med det.

Ny standard for git medlinger og branchmeldinger.

Modifisert "conventional commits".

Etter hvilke kommit meldinger som kommer fra hvilke feature branches

Kilder

Kilder skal brukes mer akryvt, henhvis tidlig fremfor sent i avsnittet.

Vis til kilden første gang den blir brukt. Bedre enn å referere på slutten av avsnittet.

Fiks dette til bachelor oppgaven.

Notater møte 08.02.2024 - Veiledere

Forprosjekt rapport, se kommentarer fra Erik, sendt til Martin.

Gokjent, men endre på disse forslagene før den legges inn som vedlegg til bachelor oppgaven.

Hvordan vil nina som organisasjon endre seg ved at de får produktet av oss.

Hva skjer med NINA, den langsiktige endringen.

Hva er målet

Hvorfor er det vi gjør viktig for NINA.

Langsiktige eventuelle endringen av NINA som organisasjon.

Hv endring hos oppdragsgiver.

I all hovedsak er det produktet vi skal levere ikke rapporten.

Drift og sikkerhet.

Sier noe om at vi skal gjøre en leveranse som er lagt opp til fremtidig utvikling og patching.

Stryk "code", alt er "open source".

Vi har skrevet en krav spesifikasjon.

Se på sikkerhet

For utviklere, definer eksakt hva som skal skje.

Kvalitet.

Sikkerhet, skal ikke gjøre mer enn det som er

Uoppdaget funksjonalitet som kan bli utnyttet (Dette er en sårbarhet som vi må unngå) (Missuse case fikser dette)

Access kontroll er bare en liten del av sikkerhet, vær litt mer spesifikk.

Performance er knyttet til availability.

Dette er slikt i rapporten.

Ytelse kan også gå under usability.

Availability er også noe vi må tenke innen sikkerhet.

Nettsiden skal takle så så mange, dette er noe NINA ordner. De ordner deploying av programmet som vi lager.

Det er i integreringen at de fleste fil skjer, omformuler denne setningen?

Best practis er ikke statisk.

Dette er noe som endrer seg over tiden.

Vær skeptisk.

Argumenter for hvordan vi skal bruke scrum.

1 - 12 skal skrives med bokstaver.

Nummere på grupperegler, dette fikser vi.

Hvordan ser følgende ut:

- risk assessment

-

- use case

- misuse case
- site map
- wireframe
- requirements
- vise github?

Vet dere om noen eksterne lover som vi burde være klar over (noe vi ikke har tenkt på)
Anbefalinger, alle i offentlig sektor skal ha sccessibility.
Vi kan mye om det.

Hvor spesifikk bør security requirements være?

Bruk sterk kryptering eller https, burde ikke spesifisere requirements?

Sjekk ut OWASP.

Hvordan bruker vi det, skriv mer om dette.

Notater møte 08.02.2024 - NINA

Hva er det som er akseptabl risiko?

De skal tenke litt på om det er akseptabel risiko at alle skal ha leserettigheter uten å logge inn. Snakker om at man må logge inn for å ha innsyn til dataene, men skal tenke på det til neste gang.

Forklar bedre hva som menes med version pinning? Skla vi bare definere versjoner for prosjektet, eller skal vi definere ulike versjoner av alle libraries og lignende osm vi bruker for å forikre reproducability (e.g. ha et dokument med dependencies der alt står, + hvilke versjon av dependencien som ble brukt)

Skal data som blir uploaded til NINA's backend være i et csv format?

Er NINA en del av offentlig sektor

Privat stiftelse.

Boxplott

25 og 75% persentiler.

Type data, år også art

Ønsker at det skal være en scrollfunksjon på hele "Elvedata" elementet.

WCAG 2.0, kontrast og slike retningslinjer.
Ser totalt antall fisk på elvenivå også.
Skilt/ sammenlagt av alle dataene som er samlet inn.
Ser mer på skjerm størrelser til neste møte.

Last opp filer, max grense skal være lavere enn 50 MB.
Ingen låsing om feil passord.

Notater møte 14.02.2024 - NINA

Hele siden bak innlogging
Flere nivå med rettigheter på siden:
 Leserettigheter
 Leserettigheter, opplasting og nedlasting (Admin)

Notater møte 14.02.2024 - Veiledere

Skrev usertest spørsmålene og svarene på norsk
 Ikke et problem.
Sprint review på norsk
 Går helt fint - forklar innledningshvis på engelsk.
Authenticate - authorized
 Knyttet mot hverandre men hvilken er best å bruke?
 Autentisert - identiteten din - loggin
 Autorisere - gir tilgang til ressurser.

Rapport struktur
 Er det noe mal?
 Introduction
 Background
 Hvordan det ble gjort
 Resultat
 Ikke noe som er spikret, men etter preferanser.
 Fremstill en naturlig struktur for leseren.
 Bygg teksten opp etter hvordan vi løste utfordringer.

Slik vi har tenkt å strukturere rapporten:

- Introduksjon
- Background
- Methodology design devops
- Resultater
- Diskusjon
- Konklusjon bærekraft i bunnen diskusjon.

Skriv i presens, mest mulig nå tid.

Vente med personlige ytringer til diskusjonsdelen.

Minst mulig bruk av personlig pronomen.

Kilder

Innrykk - sitater over 40 tegn

Lærebøker er sammenstilling fra flere kilder, finn kobling til original kilden.

Vis til kilder i løpende tekst.

Ikke kommenter det som er gjort av feil i lærebøker, si at du bare brukte strukturen.

Tydligjør at sitatet ligger der for å gjøre det lettere for brukeren å forstå det som står i testen vår.

User test:

Mye nyttig tilbakemelding fra usertestene, endringsforslag ble lagt inn og blir implementert i en ferdigstilt wireframe.

Ny funksjonallitet kommer til å bli lagt inn om vi får tid

Lager rapport ut ifra tabellene og grafene som vises, slik at de kan bli lastet ned.

Risk assessment

Skal oppdateres.

Threat modeling - ikke helt ferdig.

Dread

Countermeasures.

Ikke skriv "In the following table", skriv heller tabell navnet/ nummeret, Brukervennlighet er viktig.

Skal med i rapporten.

Kravspec

Utføring

...

Notater møte 22.02.2024 - Veiledere

Viste nettsiden.

Svelte - fremfor React - se litt nærmere på dette.

De som valgte svelte gikk fra "Angular".

Frem til nå - ukentlige møter med NINA

Grunnet user test og oppgvedefinering

Fra nå av - Hver andre uke.

Ny risk assesment grunnet endring av krav fra NINA

Skal vi legge den vi allerede har laget som vedlegg?

Burde være med, dokumenter det dere har gjort.

Skriv om erfaring med prosjektet nederst i rapporten, og nevnt denne

endringen i denne delen av rapporten og reerer til den "gamle" for å vise til hva som ble endret.

Ser mye på react akkurat nå.

Hvordan får vi til HTTPS sertifikat?

De har de ikke gjort før

Snakk med IT avdelingen

Forklarer hvordan Git-en vår fungerer.

Commit historikk

Har dette noe å si?

JAI!!

Viktige å ha omfattende commitlog, alle må være synelige i commitloggen, men ikke viktig at alle har like mange commits.

Kan ha en stor commit i starten og ha mindre kommit underveis.

Karakter

Gruppe karakter

Om man klager, kan karakteren endres for denne spesifikke personen.

Om Webtek faget - den er privat

Si ifra om dette til tilitsvalgte ved NTNU.

Blackboard

Mapper som mangler fra noen fag.

Using HTTPS in Development

Sjekk IT avdelingen - Lars Erik og Egil.

De kjenner bedre til dette med HTTPS sertifikater.

Notater møte 28.02.2024 - NINA

Har de backup? Ja det har de.

Skal ikke være overlapping på constraint.

Bruker Wizard - unngår overlapping?

De er fornøyde med databasestrukturen som de har satt opp nå.

De har lastet opp dataene i dag.

Har dataene id?

Elvedata har ID, som er randomisert.

Constraint på ID- og timestraint.

Wizard kobler sammen stasjonsdata med elvedata, for å sjekke elv med time range. (Bruker en unik identifier).

Systemet lager en unik id hver gang.

Azura - genererer alle id-er og får dem til å koble sammen.

Ganske enkel struktur.

Han fikk et eksempel vi kan se på, slik at vi ikke bare ser på Python strukturen, men kan bruke Python scriptet for å se på hvordan ting er strukturert.

Bruker python kode for å konvertere JSON data og kobler dette til Azura?

Var det dette han mente?

Vi må finne en måte for å konvertere filene til JSON format.

Vi skal lage SQL views for å vise dataen.

Nå har de dato, underscore, create table, etc.

Kan lage views som lager en tom fil med dette navnet.

Scimm immigration.

Bruker PostgREST.

Bruker websocket for å snakke med Python Wizard.

Brukes fordi de kan få progress bar og for at de store filene blir mer stabile. - foreslår at vi bruker Wizard.

Kan laste opp flere filer i samme fil, en større fil.

Progress bar, gir feedback til brukeren, nyttig.

Det er bare noen endringer han skal gjøre med wizard, jeg skal pushe siste versjon innen idag eller i morgen.

Tobias skal tagge alle.

Testdata skal slettes senere, men det fikser vi.

```
-- migrate:up

create extension if not exists postgis;

create table elvedata(
  id integer primary key generated always as identity,
  dato date not null,
  elv text not null,
  baattype text not null,
  posisjon geometry(point, 4326) not null,
  vannfoering integer,
  skipper text not null,
  mannskap text[],
  prosjektnavn text,
  prosjektnummer text,
  kommentar text,
  unique (dato, elv)
);
```

Byttet dato med dato range.

Lagde en ekstra constraint på dette, så dette ser bra ut.

Har de kommet noe lenger på autentisering.

- Er knyttet til andre prosjekter også for autentisering

 - Bruker Keycloak.

 - Bruker reklamebasert, Lda, jgpt?

 - Kan brukes til å koble til en statisk webserver.

 - Kan vise profil øverst til høyre.

 - Les på keyclock, som de skal koble til LDAP.

 - Vi kan lage tester som vi kan teste på egen maskin.

 - De skal integrere keycloak litt mer før de bruker dette.

 - Han sender siste versjon snart.

- Databasen er viktigere og hvordan man snakker med databasen er høyere prioritet, men fikser 90% i starten av mars.

Kikklogg

Man kan vise hvem som er logget inn, etter innloggingssiden.

Login på fronten.

Kan legge inn mer.

Sette inn brukernavn i headeren.

Azura implementerer ikke identifisering.

En enkleste å lage er en jbt?

Kan sende med en token for å se hvem som er brukeren, men kan vi se rettigheter.

Kikklogg,

Ganske god sikkerhet.

Browsere kan bruke samme token for å identifisere brukeren, men dette skal de se nærmere på og jobbe mer med.

Skal ha en bruker som er logget inn

Skal enten se data, eller se data og laste ned data,

Token kan vi se autorisering med

Kikklogg - kan se gruppe her også i tillegg til autentisering.

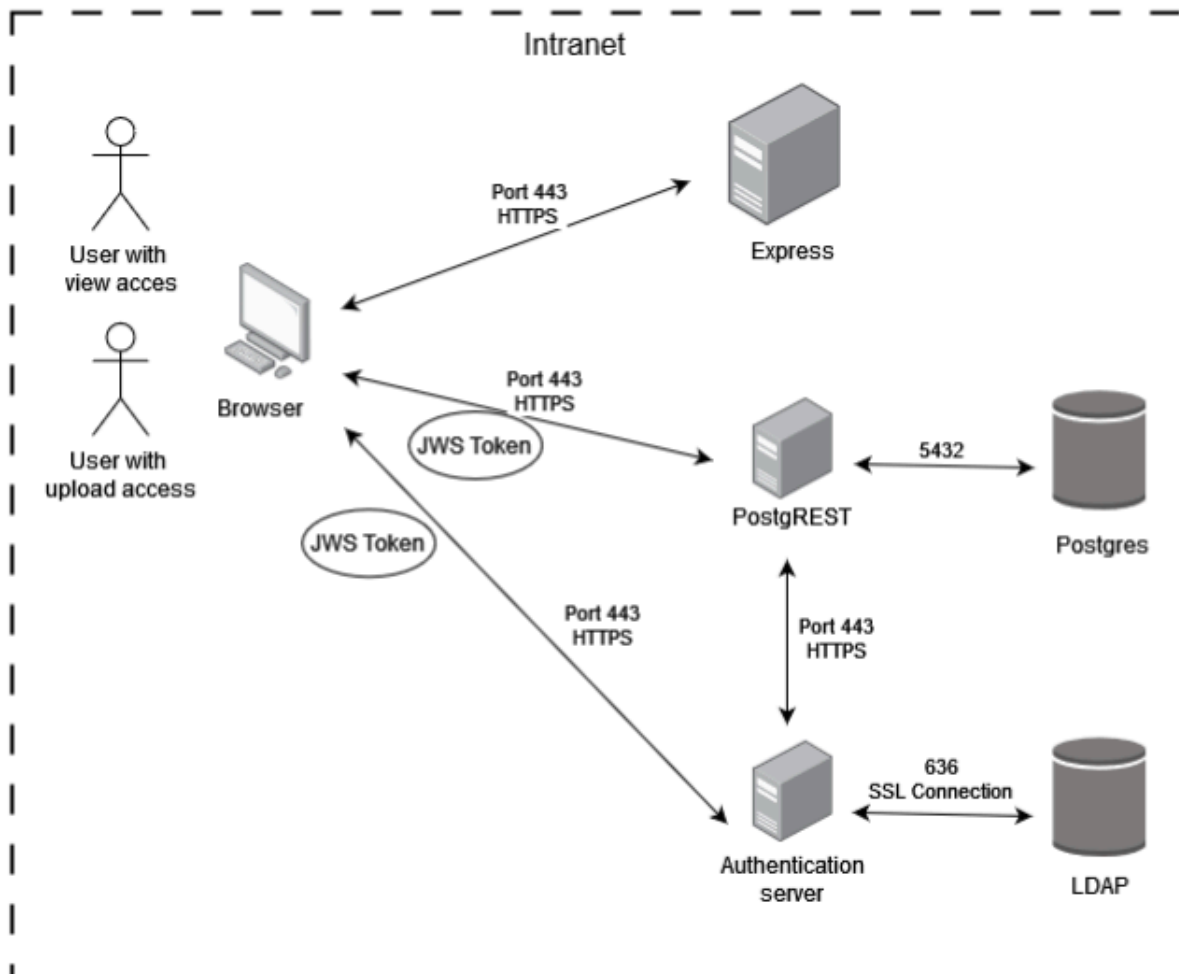
Hvis det er noen feil eller noe, kommer databasen til å si nei, en sikkerhets ting de skal sjekke opp og implementere.

Bruker LDAP på en side, andre side jgpt token.

Kan lage diagrammer på et nettverk, men dette har vi gjort.

Bruker Nginx.

Bruker blir autentisert, reaktiv kikklogg.



Har to ulike databaser

Postgrest

jpt .

Synkronisert med lista

Briwser kobler til database.

Express bak enginix.

Fjerner header som ikke skal være dær for høyere sikkerhets nivå.

Har azora for å laste opp data.

Bruker ssl offloading for data.

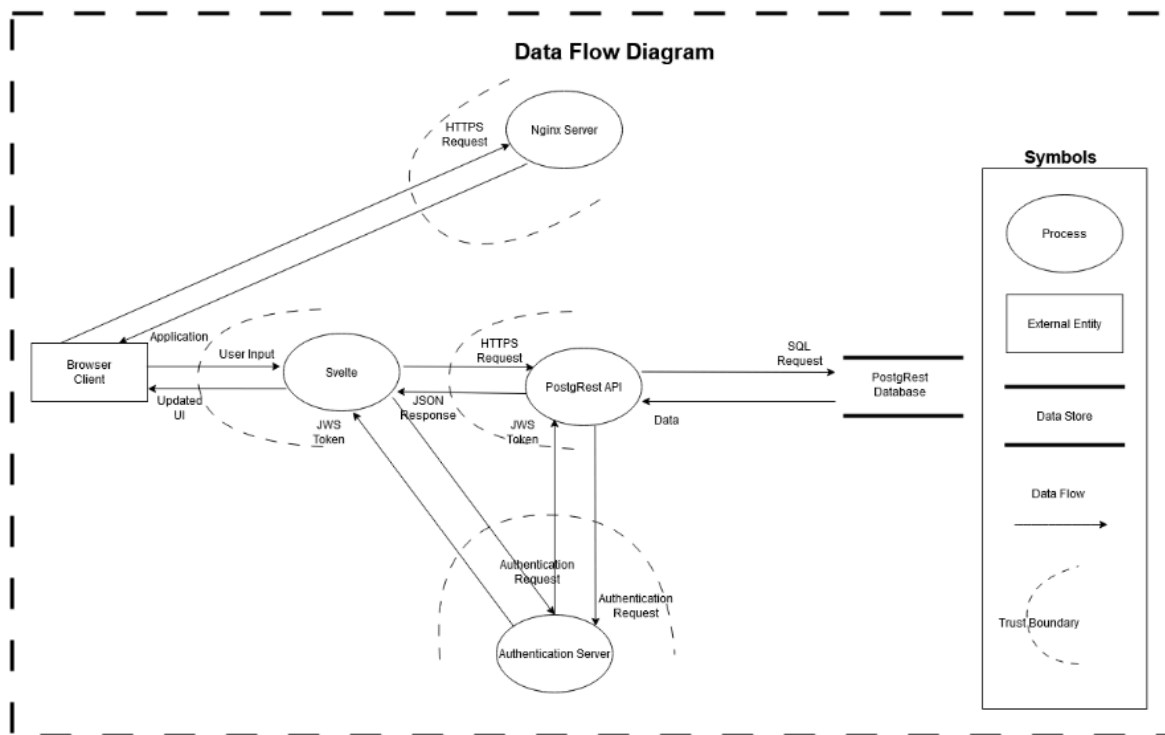
Kobler til enginex war???

Bruker websocket, azorea ...

Enginex snakker autentification server, mer sikkerhet fremfor at hver del sjekker hver del hver for seg. Enginex sjekker alt.

Spør han om enginex, han kan hjelpe oss.

Viser dataflow diagrammet.



Svelte er en god løsning.

FraFra veldig glad.

Code er unmounted på code.

Omvisningsretning

Static file på nginx.

Vi skal levere static file.

Ikke noe problem for dem.

Docker compose blir for å ha tjenester på production server.

Skal se mer på dette.

Lag et diagram for dette?

NINA. - lager en for autentisering med noen linjer av dokumentasjon.

Murmade.js - kodeblokk.

SSL certificate

Har to nginx server på proxy-en.

En for it drift med offloading.

Nginx server s

Vi trenger ikke å tenke på ssl certificate.

SSL sertifikat på sen docker key. Ikke sikker?

Reverse proxy server osv kan lage en felles server.

Trefick på toppen.

Renovation i stedet for proxy på dette. De har to webservere.

Hva gjør workflow filen?

Github action.

Leser fra github registry

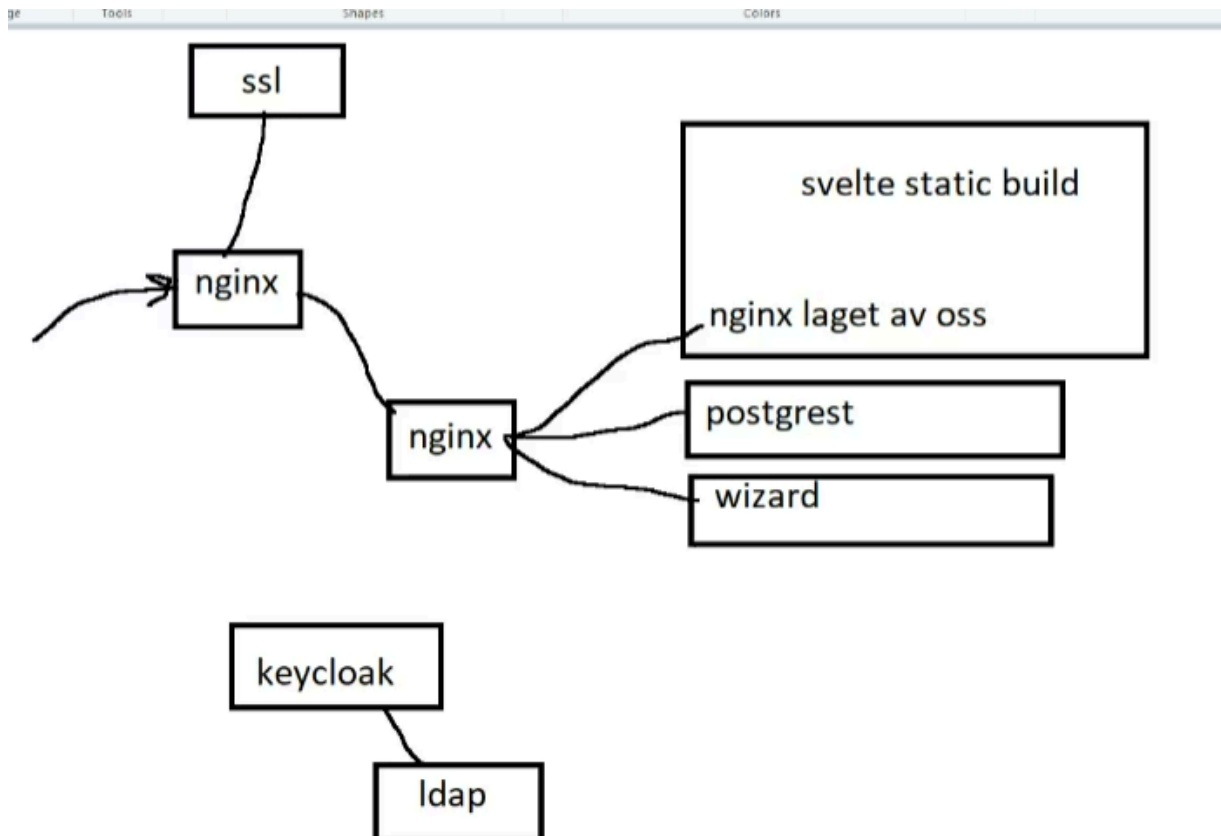
Kjører alle docker konteinerene.

Liknende løsning som vi har.

Docker ompose app.

Kan tilpasse dette med egen bash script.

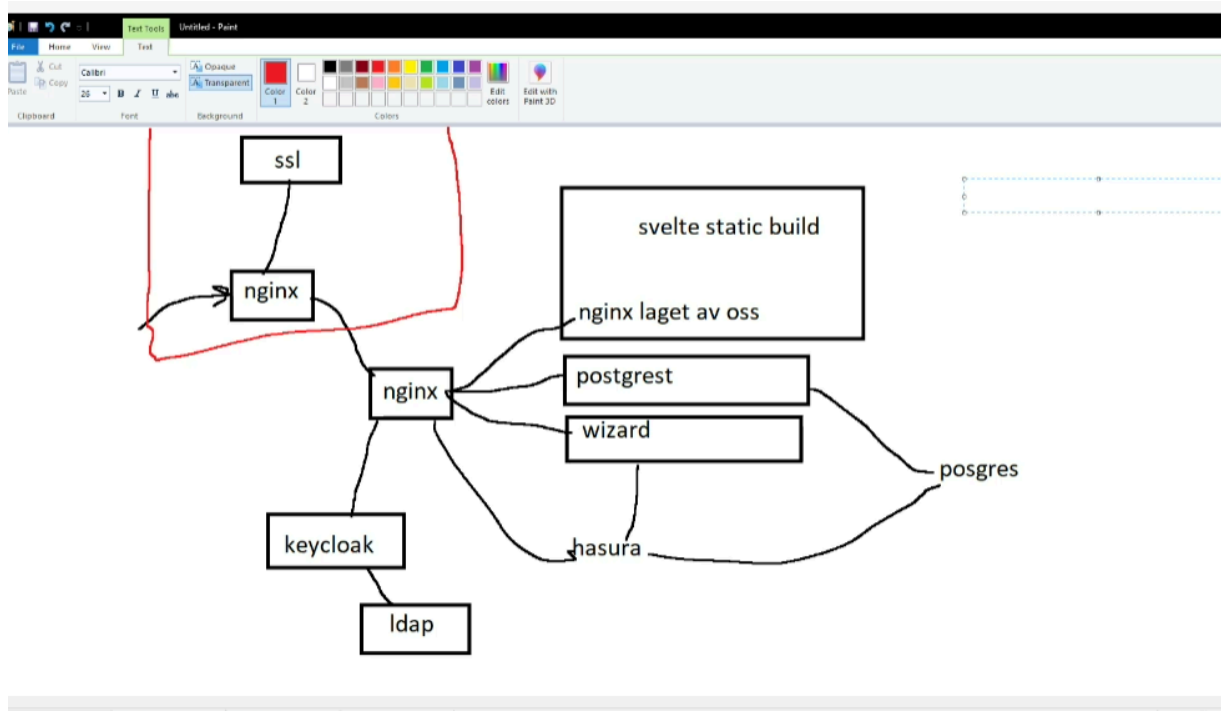
Vi sender det vi har gjort med github workflows og actions slik at han kan se på dette.



De har bare to ikke tre.

Jklfjklajfklasfj

V



Skal ha docker file, må ha noe som nginx etter oppgaven vår.

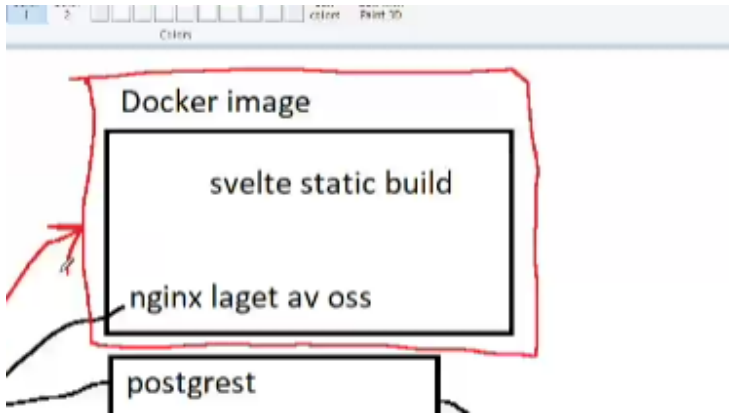
Kan ha bildet og path av det, hva da????

Har en mappe i git sm heter Svelte.

Har docker compose også?

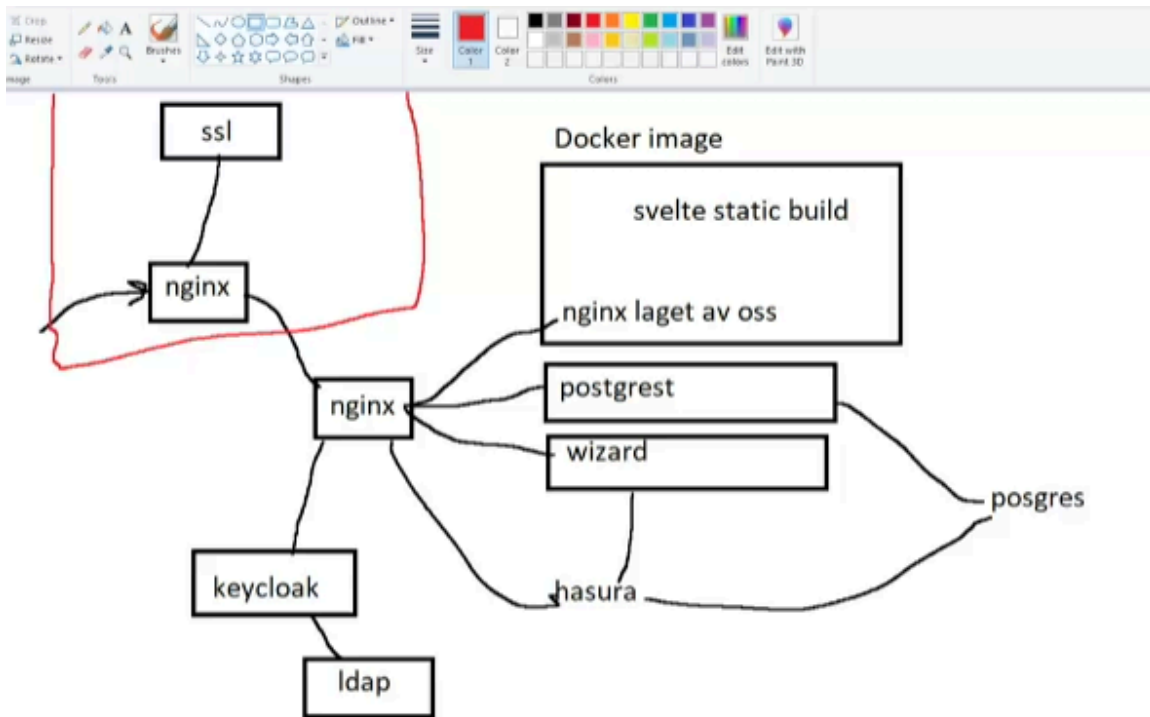
Dette er løsningen hans, men vi kan forbedre dette og jobbe med frafra.

Gjør hva vi føler er best, så kan vi endre på dette senere, kan sjekker om det er nyttig å endre på dette senere kan gå or tre nginx.

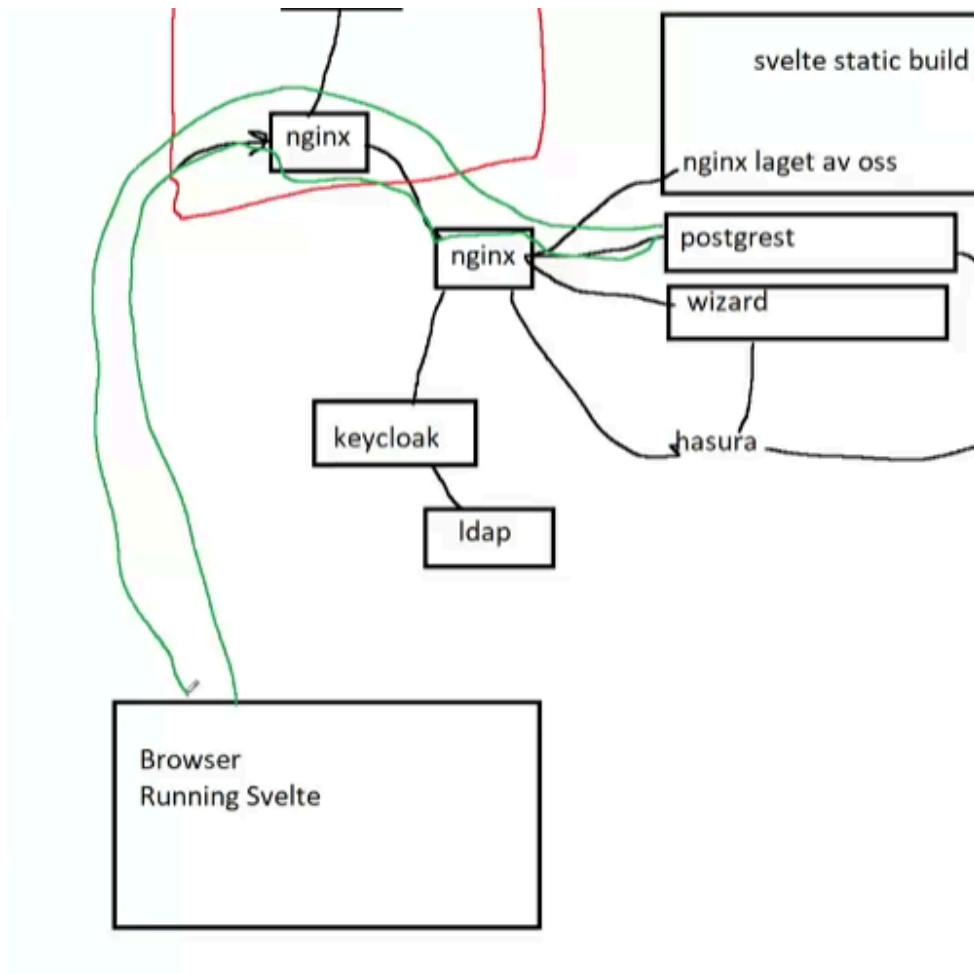


Får docker filen fra oss det i rø boks på høyre side som de kan kjøre på sin ønskede port. Bruker port 80 som standard.

Vi er på samme side her.



Ksla ha clientside rentering som kjører på en browser.



Spør om data fortløpende.

Autentisert med nginx. Noen har lov til å gå til postgrest og noen har tilgang til wizard. Kiklogger.

SKal endre på config filen

Til client side rendering

FraFra skal dokumentere og visualisere dette bedre.

Trenger noen ekstra dager for å fikse kikklogg og slikt.

Autentisering og hvordan ting snakker med hverandre er det viktigste.

Dokumentering og slikt bør stå i docker compose.

Må vise til at kryptering og https er håndteret.

Noen av valgene våre er basert på bærekraft.

Hvor mange elvedata er det totalt, og hvor mange kommer det til å være totalt?

Topper 112 - elvedata

1000 stasjonsdata.
Nærmere 50 000 individ data.

Om nettsiden klarer å takle nærmere 1000 elvedata punkter, 10 000 stasjonspunkt, halv million individdata eller mer

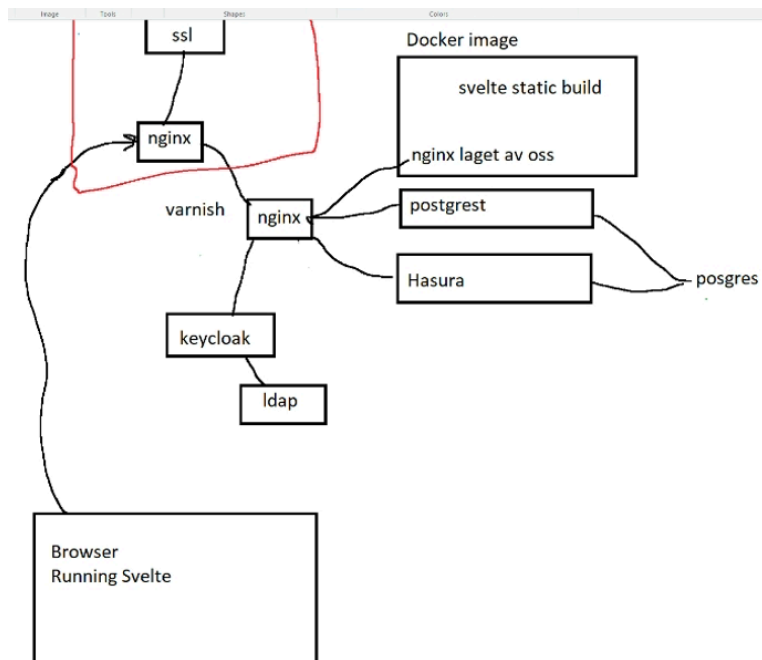
Bruker bare primary keys - ikke så bra.
Hvordan de indekserer data.
Bruker Varnish som caching.
Det fikser de, trenger ikke å tenke på dette.

Skal lagre dataene i "local browseren".
PostgREST kan man gjøre mye med.

Blir to uker til neste møt.
Se på autentiserings metoder.

Vi må lage en ekte database med mocking av PostgREST for testing, som vi fikser til neste møte.
Setter inn testdata som vi får fra NINA.
Post innlegg med oppgaver vi tenker å gjøre til neste møte, send dette i Teams gruppa.
Legger dette ut i morgen - vis til det som ligger i Jira.

Det vi har gått igjennom



Statiske filer - Filer som er fra svelte som er likt for alle som får tilgang til alle.
 Snakker om hvordan ting er koblet sammen, har to Nginx servere som er koblet til Hasura som igjen er koblet til postgres serveren.
 Keycloak er koblet til med nginx og skal autentisere brukere ved gyldig brukernavn og passord.

Notater møte 29.02.2024 - Veileder

Møte - 08.03.2024 - Fredag - 11.30 - 12.00

Begynner ny sprint i dag

Se bachelor rapport struktur

Spør om vi kan bruke logoen til NINA,
 Seksjoner for metodmetodikk og requirements, design, utvikling og collaboration.

Design

Design av software arkitektur.

Sitemap

Wireframe

Threat Modeling

User test

...

Det er en forskjell mellom hvordan man skal gjøre noe og hvordan man faktisk gjør det.
Har eget design dokument som vedlegg.

I rapporten beskriver vi hvordan vi gjorde det og hva vi har fulgt og litt om hvorfor vi gjorde det vi gjorde.

Requirement engineering ligger som dokumentasjon.

Quality gate

Data qualification.

Skriver hvordan og hvorfor med litt mer detaljert beskrivelse i vedlegget.

Setter design dokument som vedlegg, etter det som er satt i tidligere oppgaver.

Forklar design aktivt i selve oppgaven (logisk oppbygning).

Skal lese rapporten uten å se på vedleggene og få mye av det hele bildet.

Forklar hvordan man kommer frem til god design i oppgaven.

Lag et designdokument med full liste for design elementer som vedlegg.

Se helheten og flytt på ting som mangler.

Leser igjennom rapporten

kan bli unaturlig hopp om det ikke står noe om design.

Må ha en mellom ting

Design og implementasjons kapittelet mangler, men vi skal inkludere noe i resultatdelen som skal forsikre leseren om at man kanskje ikke trenger å se på vedleggene for å kunne forstå oppgaven.

Minimer sjansene for at leser oppdager at de går glipp av noe.

Be noen andre om å lese rapporten for å se om de legger merke til noen unaturlige hopp, lever rapport til veleder når vi begynner å bli ferdig med rapporten.

Diskusjon.

Om alt ble implementert som det skulle...

Hva gikk bra/ dårlig.

Implementering - administrativt.

Hvordan resultatene rundt oppgaven kommer til syne.

Integration.

Drift og livssyklus.

Innebærer - hvordan programvaren skal vedligeholdes.

Ha noen tanker om hvordan NINA skal drifte nettsiden, snakk litt med dem om denne.

Discussion.

Alle har dette med- standard

Alternative valg vi kunne ha gjort under oppgave gjennomføringen

Ideer og disposisjoner

Viser til valg og begrunner dette etter hvorfor vi har gjort det vi har gjort.

Ingen fasit, bygger på etter hvordan ting er strukturert og lagt opp.

Bruk overleaf, gjør det lettere å flytte ting rundt, bruk kommandoene aktivt.

Bruker paragraph fremfor subsubsub...

Bruk Section * (For at nummerering ikke skal synes)

Konklusjon.

Bærekraft,

Vedlegg.

Noen dokumenter skal over til vedlegg, det fikser vi.

Kommenter hele prosessen, det vi har jobbet med.

Henvis til problemstillingen og drøft denne, hvor da.

Ikke bestemt enda men gjør dette i konklusjonsdelen.

Dokumenterer det vi har gjort,

Lite rapportskrivning grunnet jobbing med kode.

Skal sette opp strukturen for infrastrukturen vår.

Brukt tid på å lære oss Svelte.

NINA syntes at svelte var bra.

Svelte - mer bærekraftig.

Svelte er lettere å implementere

Viser bilde av strukturen.

Kiklog for autentisering

Https - det har NINA tenkt på.

Forklar alle grensesnitt som vi har forholdt oss til, men trenger ikke å snakke så mye om hva NINA har ansvar for å implementere.

Viser Dataflow diagram- skal oppdateres.

Ha diagram og highlight hva vi har gjort og vår interaksjon slik at man kan se hva vi har jobbet med/ hatt ansvar for.

Gjør boksene våre store og tydelige

Notater møte 08.03.2024 - Veileder

Skal svare mail i løpet av dagen.

Time med Frode var oppklarende. (Retningslinje)

Har startet godt med utviklingen.

Kapitler.

Metodevalg

Vi har valgt forskjellige metoder i ulike steg av utviklingen, og brukte noen metoder rundt dette.

Skal vi forklare metodevalg i requirements kapitlet?

I slutten av kappittel 1, forklar strukturen i rapporten.

Ikke en metodekapittel, men nevnt dette i andre kapitler.

Utviklingen vår skal komme frem i rapporten.

Resultater som en underdel av avslutningen.

Fint å ha som et eget kapittel før siste kapittelet.

Avslutning skal være kort, oppsummerer alt med forslag til videre arbeid.

Trekke frem om vi har oppnådd de ulike målene som vi har satt oss i innledningen.

Hva det er man har laget

Det vi har fått ut av prosessen, hva ville vi ha gjort annerledes i prosjektet om vi skulle ha gjort det på nytt.

Læringsmål - knyttet til oss.

Drøfter selve oppgaven også helt til slutt skal man komme med en subjektiv del om samarbeid og prosess

Litt mer personlig om våre meninger.

Hva vi fikk til som en gruppe.

Avslutning

Drøfting, bærekraftighet og resultat.

Drøfting før resultat delen, eller motsatt?

Ta den sekvensen som vi føler at vil være naturlig

Utvikling og implementering, hvordan vi kodet

Utvikling - hvordan koden utviklet seg under prosessen

Implementering - eksempl fra koden og hva koden gjør.

Word: se under «Dokumenter» i Blackboard. Er innebygd i LaTeX-malen.

Ekstra forside?

Forord: Takk til, kort presentasjon av oppdragsgiver

Innholdsfortegnelse

+ Egen tabell- og figuroversikt(?)

Innledning:

- (Bakgrunn,) Problemområde, avgrensning og oppgavedefinisjon
- Målgruppe (for rapporten og oppgaven) og målene (resultat-, effekt- og læringsmål)
- Egen bakgrunn og kompetanse. Hva må læres?
- Rammer (tidsmessig (jfr.fremdriftsplan), fysisk, praktisk, arbeidsmetoder, teknisk, prosjektorganisering)
- Øvrige roller (oppdragsgiver, veileder)
- Selve rapporten (organisering, terminologi (bruk norsk), praktisk (angående layout, style, fonter o.l.))

Teori:

- Fagfeltet, bakgrunn, mer i dybden om problemstillingen/oppgaven, perspektiv ift. annet faglig/stoff
- Formål/hvorfor dette emnet
- Teori om emnet

Hovedkapitler: (for mange: *oppsummering* av Scrum-prosessen)

- Kravspesifikasjon (Use Cases, User Stories, Product Backlog)
- Design (GUI og teknisk) | Gjerne diskusjon om alternativer underveis.

Innledning, teori og avslutning er de samme, men hovedkapitlene kan variere.

Hovedkapitler: (for mange: oppsummering av Scrum-prosessen)

- Kravspesifikasjon (Use Cases, User Stories, Product Backlog)
- Design (GUI og teknisk) | Gjærne diskusjon om alternativer underveis
- Utviklingsprosessen
- Implementering / koding / produksjon (evt. vise *klart* til gjenbruk)
- Testing, kvalitetssikring (om sårbarhet ved evt. gjenbruk)
- Installasjon / realisering

Når lite programutvikling (jfr. IMRaD: <https://sokogskriv.no/skriving/imrad-modellen.html>):

Metodevalg (Presentasjon av eksisterende metoder, Begrunnelse/beskrivelse eget valg)

Det vi skal ha (markert).

Huskeliste over hva det er vi kan si noe om.

Kan styre dette litt selv.

Vise til det vi har jobbet med.

Dokumentasjonsstandard.

Bruker bare standard javascript dokumentasjon.

Doxygen.

Følger standard innenfor javascript. (JSDoc).

Hvordan bruker vi git branches.

kortlevende

Henvise til litteratur/ git best practices.

Git branches documentation.

Unit test.

Intigration test

Pentesting - for å sikre sikkerhet i applikasjonen vi har laget.

Modulær med noen konstanter.

Lettere for NINA å opprettholde.

Egen mappe for modeller.

Bruker Linting.

Bruker camelcase.

Disablet camelcase linting coden - unntak for camelcase linting (Kun i det dokumentet som ble vist i møtet)

Notater møte 13.03.2024 - NINA

Viser det vi har jobbet med, (Nettsiden).

Viser forsiden.

Butte mellom elver og stasjoner.

Viser at funksjonene fungerer som de skal.

Går igjennom alle sidene på nettdisen og viser funksjonalitetet, tester og viser at filtering og slikt fungerer som det skal.

Ønsker å beholde prikkene (rå dataene ved siden av boxplot-ene).

Synes at det ser veldig livende ut.

Synes at kartet og alt ser bra ut.

Notater møte 21.03.2024 - NINA

Viser til github-en.

Notater møte 03.04.2024 - NINA

Noen spørsmål.

Kan ta bruker test neste uke 10.04.2024

Viser nettsiden.

Dette er testdataen som vi har lagt inn og testet med på nettsiden.

Litt forskjellig navn, krasjet, må fikset dette før vi kan legge inn de test dataene.

Noe format og mal må endres

Kan endre på fargene på histogrammet senere. Samkjøre fargene på linjene og histogrammene.

Kan bli litt mye ting på sidene ettersom at det er doblet opp på siden, dette er noe vi burde endre på.

Biksene skal være mer gjennomsiktede.

Synes t dette var informativt og veldig enkelt å sere på intervallene som er veldig positivt.

Endre fra "SANN" til "JA" på filene som lastes ned.

Tobias skal gå over alle filene og sjekke om de er riktig, for å se om det er noe som må endres på når det kommer til "terminologi".

Ble gjort noen småendringer når de endret på malen.

Noen stasjoner som det ikke ble funnet noen fisker, men ønsker at skasjonen vises med stasjons info-en, men da uten fiskedata.

Stasjonsbeskrivelse (Kommentar på stasjonsdata også), Fiskedata var tom og at det vle null fangst.

De syntes at dette ser kjempefint ut.

To uker med programmering.

Autentisering - implementering.

HAr oppnått og har nesten allt vi trenger til nå, skal rekke å bli ferdig med alt, men kommer til å starte med rapporten om to uker.

SKal implementere flere kart, med sattelitt kart om vi får tid, som en bonus.

Under neste brukertest skal vi ta en gjennomgang av nettsiden

Jango på pstgrest koblet til postgrest,.

Fikset slik at vi kan bruke og test lokalt.

Lage en graft pr, skal lage pull request slik at francesco ca se hva slags endringer som er gjort på repoet.

Stasjonsnummeret = det vi la til, og skal ikke være lik null, denne er ikke id-en???

Id generert av postgrest

Status på excel upload endpointen = Har inter kode som fungerer, må bare sette det opp på jango og laste det opp på fredag.

Trenger ikke å bruke Wisard.

Autentiserings systemet - satt på yango rst frameworken. Bace token. Skal dele det med oss, se link:(<https://www.django-rest-framework.org/api-guide/authentication/#api-reference>)

FraFra kan hjelpe oss med dette neste uke.

Koden blir ferdig på fredag, også kan vi jobbe sammen med integrasjonen neste uke.

Dagen som passer best - Torsdag 12:30

Lager mteinkalling - Frafra tilkaller de andre som kan være med på møtet.

Vi har lages SQL views, de må selv modifisere dette.

River summary

De gir beskjed når de er ferdig med excel endpointet.

Tobias skal gjøre de nødvendige endringene på excel filene og sende ut de opdaterte versjone.

Forklar bedre hva som menes med version pinning? Skal vi bare definere versjoner for prosjektet, eller skal vi definere ulike versjoner av alle libraries og lignende osm vi bruker for å forikre reproducability (e.g. ha et dokument med dependencies der alt står, + hvilke versjon av dependencien som ble brukt)

Vise nettsiden så langt

Vil de vise stasjoner som har ingen observasjoner?

Hvordan går det med postgrest

Hvordan går det med keycloak

Om de har laget diagram for infrastrukturen sin

Nytt

Problemer og endringer vi har gjort med fishboat-database repo

Autentisering

Endpoint for excel filer

User test neste uke

Notater møte 10.04.2024 - NINA

FraFra har kommet langt med det han skal gjøre

han viser Django til Kevin og Carl Petter, forklarer hvordan det fungerer, viser hvordan infrastrukturen kommer til å bli på NINA sin side. Forklarer hvordan applikasjonen skal bruke det.

Tar brukertester på FraFra og på Tobias.

Stort sett fornøyde med nettsiden, noen forslag til hva som kan gjøres bedre.

Notater møte 11.04.2024 - Veiledere

Viser nettsiden.

Så og si ferdig med nettsiden.

Se på Box plott, burde kanskje ikke bli så stor at den tar opp store deler av skjermen når den står alene. Se på dette om det er mulig.

Kevin og Carl Petter, møte med NINA - lage endpoint for å laste opp filer.

Bare to stykker, se på hvordan vi skal laste opp excel filer opp til nettsiden.

Innloggings siden skal være på samme domene.

Cookie. tilgjengelig for nettsiden. Den vil automatisk lese cookien på din der,s,

Sikkerhet rundt cookie. cookie skal ikke være tilgjengelig for javascripten, kun nettleseren.

Kun nettleseren og ikke nettsiden som skal ha tilgang til cookien. Basert på sikkerhets best practice fra OWASP.

IT eksperten kom med noen bugs som burde fikses, dette var bare noen små ting ellers var de veldig fornøyde med nettsiden.

Nesten alle skal skrive på rapporten fra mandag.

Bra at vi har tatt kontakt med Egil før vi gjennomfører pentesting på nettsiden slik at han kan si ifra til NTNU digital sikkerhet for å si at de kan ignorere varslingsene som kommer inn fra den og den ip-adressen.

Intrigrasjons tenting kommer vi kanskje ikke til å bli 100% ferdig med ettersom at vi ikke har nok tid.

Mocking til alle dataene og utføre testene og sjekker m testene er utført riktig.

Ingen som blir 100% ferdig med slike tester.

Vanskelig med testing når det kommer til Svelte og "components". Vanskelig med autogenerated kode når det kommer til svelte kode, ettersom at den er ganske ny.

Ikke 100% dekning på testing av programmet. Fokuser på de delene av kodene som kommer til å endre seg mest, der hvor folk kommer til å gjøre endringer. Og de dere tenker at det er viktig å teste.

Dropper kanskje ikke drag and dropp funksjonaliteten, skal se på det fortløpende.

Koden relatert til opplastnings siden er den vi har ventet mye på, men tror at den skal være ferdigstilt til å implementeres til NINA møtet 12.30.

Kan få veiledning på bachelor oppgaven.

Mulig å sende inn halvveis ferdige oppgaver.

Kommer til å lese hele rapporten en gang, vi kan selv vurdere hvordan vi vil sende dem filene, men det er ideelt om vi sender med filene etter kronologisk rekkefølge etter hvordan de vil være plassert i rapporten.

Kronologisk og følger en rød tråd igjennom hele rapporten.

Dokumentere dependencies.

Sikker supply chain.

Version pinning

En rekkefølge på hvordan man burde skrive kode tester, hvilke begreper vi burde bruke se teams for å se denne malen.

Skriv om alt dere har gjort av sikkerhets testing og programmering, forklar hvorfor vi har tatt de valgene vi har tatt og reflekter rundt disse valgene.

For eksempel når det kommer til pen testing.

Vis til pen testing rammeverk. henvis til dokumentasjon som følges for å utføre testene.

Forankre det vi skriver inn i artikler og nyttige læring ressurser.

Vis til refleksjon.

Cross-site scripting

Prøver å følge alt av best practice når det kommer til å implementere sikker kode for å danne en sikker nettside.

Notater møte 17.04.2024 - NINA IT

Viser GitHub fishboat-database fra NINA

forklarer hvordan cookies blir satt med django, hvordan alle ferdige django endpointsene funker

forklarer hvordan man kan laste opp til django endpoints
viser noen endringer i databasestruktur, de vurderer å sette project id til integer i stedet for tekst
de viser hvordan man kan lese error meldinger fra upload endpoint og vise til bruker
diskutere fordel og ulemper med GitHub submodules, blir enig om at dette er beste løsning

Notater møte 24.04.2024 - NINA

Veldig fornøyd med nettsiden.

Synes at vi har vært veldig raske med implementering med alt av endringer de har kommet med.

Brukernavn og passord er "admin" for å få opp dataen på nettsiden.

Kun ett punkt som er synlig - ikke koblet opp til deres systemer enda.

Andre kan bruke og se på nettsiden uavhengig av NINA sin implementasjon, men det er satt opp slik at det er lett å implementere det inn i NINA sin infrastruktur.

Nettsiden er testet og fungerer på NINA sin side med deres data.

Snakker med FraFra og Nikolo.

Alt av funksjonalitet på nettsiden fungerer som det skal.

Legge til flere brukere og slikt er uavhengig av oss.

LDAP- I følge FraFra er det dette som brukes internt hos NINA.

Små endringer er ikke et stort problem å endre på, bare å si oss om det er noen små endringer dere vil at vi skal gjøre.

Føler oss 99% ferdig med nettsiden.

Hvis vi har tid (nice to)

Label for startpunkt, stasjonspunkt og dato.

Dette er noe vi har sett på, men det er ikke lett å implementere.

Noe vi kan se på senere

Veldig fornøyd med hvordan nettsiden ser ut.

Noen fikk livsgnist i å legge inn dataene sine inn på nettsiden.

Gjør det artigere å jobbe med dataene og få det fram og få en oversikt over dataene og slik. Gleder meg til å laste opp filene dine fra over ti år gamle Excel- filer over til databasen og bruke nettsiden.

Gjør jobben lettere.

Rutinemessig dokumentasjon og slik blir lettere, noe som er veldig nyttig for dem.

Det er nå mulig å laste ned data direkte fra graf siden.

En ny funksjon som er lagt til.

Nedlastning av filer problematisk (ikke sikker side) vil fikse seg når nettsiden er lastet opp på deres interne infrastruktur der nettsiden vil være under "https" og ikke "http" noe den er nå.

Nytten de får av nettsiden.

Visualisering av grafer.

Skrik ut om vi trenger hjelp med formuleringer og slikt når det kommer til formålet med prosjektet.

Har ikke hatt kompetansen for å hjelpe teknisk. men kan hjelpe nå.

NINA lever av formidling, ønsker å dele dette med andre.

Konferanser og slikt kommer til å presentere arbeidet.

Kommer til å vise oss frem som medforfattere.

Vise dette til andre som sitter i samme situasjon, bruke ideer eller kode fra vårt arbeid. Mener at flere andre forskere har samme problem og kunne trenge lignende løsninger

Ønsker at vi skal holde kontakten etter dette også.

Dele mailen vår med dem.

Vi møtes under presentasjonen 5 eller 6 juni.

Kompetansen vår med å lage en slik nettside kommer til å være veldig nyttig. Veldig gunstig nå fremover ifølge NINA.

Færreste har kompetansen til å gjøre dette skikkelig.

Usikker på hvor viktig det er med møte på onsdag 08.05.2024, men har de nå inntil videre.

Kan spørre spørsmål over teams om det skulle være relevant.

Notater møte 25.04.2024 - Veiledere

Viser nettsiden.

Ferdig med nettsiden.

Tenker å gjøre pentesting og oppdatere risikorapporten.

Utenom dette så er det bare rapport skriving

Så å si ferdig tekst på ca. Over 30 sider.

Tester, unit testing

Definisjoner i teori delen.

Manipuler denne definisjonen etter hva slags tester som er utført og snakkes om.

Teksten skal være "ren", unngå repetisjon, uheldig med dette.

Teoridelen

Naturlig med seksjon om testing. Kort forklaring av relevant test metode.

Metode - valg av rammeverk

Definerer teknologien i teoridelen, også refererer vi til denne teorien når vi forklarer hvordan vi har brukt det.

Testing, flere mulige måter å utføre testing på.

Funksjonell og ikke funksjonell testing.

Innenfor testing er det ulike metoder.

Kan høre hjemme begge steder (metode delen og i teoridelen).

Skriv, så sier veilederne hva som er riktig og hva som ikke er riktig.

Forklar hva liting er i teori og ikke metode, CI/CD er mer metode.

CI/CD etter hver push og det vi har benyttet testingen er mer på metodevalg som kommer der.

Noen kapitler

Har introdusert flere konsepter, etter best practices om hvordan vi har strukturert koden vår.

Er dette teori, eller skal konseptene komme i teori og hvordan vi bruker det andre steder.

Alt skal ikke være i teori delen.

Kommer til å merke dette når vi og veilederne leser igjennom.

Om det er noe andre ikke skjønner, ta dette med i teoridelen. (store ting i teoridelen. Kan legge inn noen nye ting i andre deler av teksten også, om det skulle være nødvendig).

Alle skal skjønne starten av rapporten.

Første side

Gjør det lett å lese.

Bakgrunn og beskrivelse, burde alle skjønne, også kan de bli færre og færre (blir mer snevert) senere underveis i rapporten. Færre og færre vil skjønne det.

Nå tid i alle seksjoner, hvordan er det med dette?

Generelle ting skrives i presens, allmenngyldig.

Føles det naturlig å lese det?

Alle mulige former av verbet.

Veilederne vil kommenterer på det. (Skriv som at leserne kan føle at det er naturlig å lese rapporten)

Development, hvordan den er. Mer nåtid.

CodeBlocks for å liste koden i teksten. Kode listings

Liker at vi har fått til denne fonten.

Følger Frode sin github mal.

Github - cop ccc

Kan vi endre på tekst bredden,

La det være slik den er,

Vurdere to versjoner hahaha.

Text ttt, reagerer om vi har et spesialtegn.

Inline (\verb+) pluss skal da ikke bli brukt i kode teksten.

Latec reagerer på understrek.

Stygg tekst med manglende formatering, sjekk formateringen.

Verbatim - bruk dette.

Ikke farger, over på listings.

Morsomt med (\pmb)

Fin måte å jukse til seg fet skrift.

Ikke bruk

Hvor ferdige for å få dere til å lese på teksten?

Avhengig av hvilke kommentarer vi ønsker å få

Hull i teksten – Kommer de til å danne mange antakelser.

Håper at de snakker om det eller ikke snakker om det.

De leser alt vi har en gang.

Fin struktur, to første kapitlene må være - polert og ferdig.

Avslutningskapittelet skal være om hva vi har kommet frem til.

Innlegning, bakgrunns teori

Forbedring av teksten «inne i rapporten» er det vi kan jobbe videre på.

Sammendrag kan vi skrive nå,

Skal stå hva vi har oppnådd ikke hvordan vi har gjort det.

Nå tid, fokus på hva vi har gjort.

Sammendraget kommer til å ligge ute, fange leseren.

Etter 5 til 10 mai blir de tettpakket.

Mandag 13 mail,

(14.30 – 15.30) for å se på bachelor oppgavene.

Ser an på denne datoen og tiden.

Vanlige møter frem til det.

Sender melding på teams frem til 13 mai. Send inn 11 lørdag kveld.

Referering til presentasjoner (Prosjektledelse)

Finn et tilsvarende sted som sier det samme, som er mer åpent for andre.

Presentasjon

Vis hva vi har gjort, vis nettsiden.

Video - backup

Live - mer wow faktor.

Send mail til Lars Erik for å forsikre at SkyHigh serverne ikke resettes før presentasjonen.



 **NTNU**

Norwegian University of
Science and Technology