Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Fredrik Lønnemo
Nora Hønnåshagen Hansen

Innovative presentation of open data

Bachelor's thesis in Bachelor in Programming

Bachelor's thesis in Bachelor in Programming
Supervisor: Mariusz Nowostawski
May 2023

NTNU
Norwegian University of
Science and Technology

Fredrik Lønnemo
Nora Hønnåshagen Hansen

Innovative presentation of open data

Bachelor's thesis in Bachelor in Programming

Bachelor's thesis in Bachelor in Programming
Supervisor: Mariusz Nowostawski
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Innovative presentation of open data

Fredrik Lønnemo
Nora Hønnåshagen Hansen

Abstract

This thesis describes the process behind designing and implementing a regulation plan overlay for a 3D map. The project focuses on the front end and user experience of visualizing data in a 3D map. The project uses Three.js, and is a continuation of the wxs.three.js project by Sverre Iversen, and developed further by Gjøvik Municipality.

We have implemented user controls for height and opacity adjustments within the solution, added regulation plan overlays using custom shaders, and adjusted the map size using a Frustum. The final result is a web solution using HTML, CSS, and JavaScript. The product provides data visualization and is available through the municipality's website.

Gjøvik municipality provides services for viewing data, such as regulation plans, using interactive map interfaces. They began the development of a 3D map useful for municipal workers to see the terrain and regulation plans together, easing the construction planning process. The task was to continue developing this project and extend its initial state. The municipality plan to continue the development of this project, either internally or as a future bachelor for the next year's students.

Sammendrag

Denne oppgaven beskriver prosessen bak designet og implementeringen av reguleringsplan-lag for et 3D kart. Prosjektet fokuserer på frontenden og brukeropplevelsen av å visualisere data i et 3D kart. Prosjektet bruker Three.js, og er en fortsettelse av wxs.three.js prosjektet laget av Sverre Iversen, og videreutviklet av Gjøvik Kommune.

Vi har implementert brukerkontroller for å endre høyde og opasitet innenfor løsningen, lagt til reguleringsplan overlegg ved bruk av tilpassede teksturalgoritmer, og endret størrelsen på kartet ved bruk av et Frustum. Det endelige resultatet er en web løsning som bruker HTML, CSS og JavaScript. Produktet tilbyr visualisering av data og er tilgjengelig gjennom kommunen's nettside.

Gjøvik kommune tilbyr tjenester for å vise data, som for eksempel reguleringsplaner, ved bruk av interactive kartgrensesnitt. De begynte utviklingen av et 3D kart til hjelp for kommunens ansatte for å se terreng og regulerinsplaner sammen, som letter prosessen av å planlegge konstruksjon. Oppgaven var å fortsette utviklingen av dette prosjektet og utvide dets opprinnelige tilstand. Kommunen planlegger å fortsette utviklingen av dette prosjektet, enten internt eller som en framtidig bachelor for neste års studenter.

Preface

We would like to thank everyone involved in this project. Thanks to our supervisor, Mariusz Nowostawski, for helping us by giving advice and feedback. Thanks to our client, Municipality of Gjøvik, for allowing us to work on this exciting project. Thanks to our contact people Pål Godard and Geir Karsrud, for being actively engaged in the project's progress. And thanks to all the user testers, who took time out of their day to help give feedback on the solution.

Contents

Figures

Tables

Acronyms

NTNU Norwegian University of Science and Technology. 3, 19

Glossary

API Application Programming Interface, a type of software serving functionality to another. 3, 4, 25, 61

BBOX Bounding Box, an area defined by two longitudes and two latitudes. 35, 38, 51

CSS Cascading Style Sheets, coding language used for customising and adding style to web pages. iii, v, xix, 5, 24, 32, 33, 38, 42–45

FPS Frames Per Second, a measurement of the amount of rendered images displayed on a screen per second. 49

Frustum The portion of a pyramid that remains after its top part has been cut off by a plane parallel to its base. In 3D programming, it is used to determine what is in the camera's view. iii, v, 41, 47, 51

GeoTIFF A format extension allowing for georeference and geocoding data to be embedded into raster images in TIFF format [1]. 2, 34, 37

Github An Internet hosting service for software development and version control using Git. 37

GLSL OpenGL Shader Language, a coding language commonly used when writing shaders for graphics programming. 25, 39

GPU Graphics Processing Unit, the component in a computer or mobile device responsible for the rendering of graphics to the screen. 32

HDF Hierarchical Data Format, a data model, file format and I/O library designed for storing, exchanging, managing and archiving complex data including scientific, engineering, and remote sensing data [2]. 2

HTML HyperText Markup Language, a coding language developed for web development. iii, v, xix, 5, 24, 31–33, 35, 42–45, 52

JavaScript Scripting language used for web development alongside HTML and CSS. iii, v, 4, 5, 32–34, 42, 45, 47

JPEG Joint Photographic Experts Group, a raster image file format using lossy compression at adjustably degrees. 2, 34

Latitude A coordinate on the north-south position. 38

Longitude A coordinate on the east-west position. 38

Mesh In 3D programming, it is a 3D object made up of vertices and faces. 38, 39

MVP Minimum Viable Product, an early prototype of a product. 11

NetCDF Network Common Data Form, a set of interfaces for array-oriented data access and a freely-distributed collection of data access libraries for multiple coding languages [3]. 2

npm Node Package Manager. Package manager for the JavaScript language. 58

OpenGL Open Graphics Library, a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. 39

OpenLayers An open-source JavaScript library for displaying maps in web browsers. 4, 6, 20, 23, 34, 38, 45, 61

OpenStack Open-source cloud computing platform. 55

OrbitControls Three.js library used to control a camera. 24

PNG Portable Network Graphics, a raster image file format supporting lossless compression. 2, 34, 38, 40

Raycaster An object within 3D programming which casts a ray from the mouse pointer or the middle of the screen straight forward, and is used to find the closest object from this point. 41

Responsive Design Mode A feature in many development web browsers which allows developers to see what a website looks like on a mobile device while using a computer. 52

RESTful REpresentational State Transfer, an architectural style for an application program interface (API) that uses HTTP requests to access and use data. 61

Scene In 3D programming, a scene is a group of objects, lights, settings, etc. making up a 3D environment. 24, 38, 41, 42

SkyHiGh Cloud-native security platform. 52, 55

SRS Spatial Reference System, a framework used to measure locations as coordinates. 35, 38

Three.js JavaScript library for 3D graphics rendering using WebGL. iii, v, 4, 5, 20, 23–25, 32–34, 37, 39, 41, 43–45, 47, 58, 61, 62

TIFF Tag Image File Format, raster image file format used o store raster graphics and image information. xix, 2, 37

Topo4 Topographic coloring of maps which are color-coded; Blue for water, green for forest, white for open terrain, black for roads, etc. . 16, 34, 38–40

TrackballControls Three.js library used to control a camera. 24, 34, 52, 61

UI User Interface, the layout and frontend of a software product. xiii, 2, 6, 9, 11, 16, 19, 20, 23–26, 28, 29, 40, 42, 43, 47, 51, 57–59, 61

Unity A cross-platform game engine suitable for creating 3D graphics, capable of creating builds for WebGL and thus web browsers. 33

URL Uniform Resource Locator, an address used to locate content on the World Wide Web. 34, 35, 38, 44, 47, 51

UX User Experience, the overall experience of a software product. xiii, 2, 6, 9, 10, 12, 16, 49

Vanilla JavaScript Term commonly used to refer to plain JavaScript without frameworks. 32

WCS Web Coverage Service, standard protocol for serving data used for analyzing and rendering. 2, 4, 6, 16, 20, 34, 37, 38, 45, 62

WebGL JavaScript API for 2D and 3D graphics rendering in web browsers. xxi, 5, 15, 23, 32, 33, 39

WMS Web Map Service, standard protocol for serving map images. 2, 16, 25, 45

WMTS Web Map Tile Service, standard protocol for serving map images as sets of tiles. 2, 4, 6, 20, 34, 37, 38, 45, 62

Chapter 1

Introduction

As of May 2023, the web solution can be found online via Gjøvik Regionen here.

1.1 Background

Our bachelor thesis is a task provided by the Municipality of Gjøvik. Our primary contact person was Pål Godard, as well as Geir Karsrud.

The municipality is responsible for providing services for its inhabitants within multiple fields, including education, health, culture, and traffic [4]. Additionally, another field they have responsibilities in is construction work. With its big range of responsibilities, the municipality has a wide range of data collected in different fields, including; pedestrian and cyclist counters, recycling percentages, water temperatures, air quality, etc. The initial project proposal was very open, revolving around using any of this data and finding a way to present them to the municipality's inhabitants.

However, a new proposal emerged after the initial project proposal was sent to the university, and we were given it. Geir Karsrud, an employee within the municipality, had begun work on a 3D map of the region based on the open-source `wxs.three.js` project [5]. The municipality currently has a 2D map available where one may view additional information, such as regulation plans. Viewing such plans is an integral part of construction work, as they provide construction workers with important info regarding what an area may be used for and what may be built there. The 2D solution they currently have worked well. However, it does not represent an entirely accurate picture of real life, as levels in the terrain, like hills and valleys, may bring challenges or change how the work should be planned out and accomplished.

The 3D map solution in development was thought to be a possible aid to help combat these challenges by using data to construct a map capable of accurately representing different levels in terrain, giving construction workers the ability to

see this information and negating the need to travel to the areas in question to make decisions. The map in progress had the terrains and topography texture set; however, due to time constraints, Geiur Karsrud could not finish the project himself and shared the in-progress code for anyone wanting to continue development. Our contact person Pål Godard saw this as an excellent opportunity and presented the project for our consideration. We found the proposal interesting, and after considering the different possible scopes, we settled on working with the map as we found the scope interesting.

1.1.1 Domain

This section will provide some needed details to help understand the concepts used in this thesis.

WMS Web Mapping Service and WMTS Wep Mapping Tile Service

WMTS is a protocol used to serve map data in the form of pre-rendered images in formats such as PNG, JPEG, TIFF etc. It is similar to WMS, with the difference between the two being WMS serving the data as one image rendered on the server side and WMTS serving the data as sets of image tiles cached on the client side. The benefit of WMTS over WMS is the reduced waiting time, as with WMS the entire image must be rendered as a whole before being served, whereas WMTS serves the individual tiles as they have been loaded, with the process being faster as a result of their smaller size.

WCS Web Coverage Service

WCS is a standard issued by the Open Geospatial Consortium or OGC. A WCS service, in contrast to the static images provided by WMTS, provides data that can be used for analysis and modeling, referred to as coverages [6]. Data provided through these services include; GeoTIFF, NetCDF, HDF, etc. This data can be used for multiple purposes, for example, by using values from an endpoint to create 3D objects.

User Experience

User experience (UX) is the experience a product gives its users and how it affects their views on the company or developers behind the product. This topic is explained in greater detail in chapter 2.

User Interface

User Interface (UI) refers to the layout and appearance of a product and how it can affect the overall UX. This is talked about in greater detail in chapter 2.

1.1.2 Delimitations

Our project focuses on the visualization of open data, and it is not within our scope to ensure the data is correct as it is fetched from government-distributed APIs. While the data fetched covers the entire country of Norway, only the area containing the municipality will be focused on, and the quality of other regions' representation is not a responsibility.

1.1.3 Task description

The initial work on the 3D map done by the municipality was incomplete. The task, therefore, was to further develop the 3D map of Gjøvik Municipality to be presentable and to add a visualization of area plans which can be brought up by clicking a building with a mouse or touch controls on mobile devices. This is the minimum scope, where additional tasks such as visualizing other open data from the municipality are possible additions.

1.2 Target audience

1.2.1 Thesis

This thesis' target audience consists of reviewers tasked with grading the thesis, students interested in reading about our work, and the client. The product is open for further development, and this thesis will serve as documentation aiding that process.

1.2.2 Product

The primary target audience is internal employees working for Gjøvik Municipality, primarily construction workers, thus focusing on optimizing the product for desktop computers. However, as the product will be available for anyone to use via Gjøvik Municipality's website, a secondary target audience is the general public of the municipality and possibly the rest of the country. To satisfy the wider demographic, the project will be given a responsive design making it just as easy to use on mobile devices such as phones and tablets.

1.3 Group background

1.3.1 Academic background

Both group members on the team are students in the Bachelor in Programming programme at the Norwegian University of Science and Technology in Gjøvik. Through this programme, we acquired knowledge in many areas of programming, including web development and general software development. Of the many courses we have taken, the most relevant to this project are shown in 1.1 on page 4.

IDG1362 Introduction to User-Centered Design	Creating user tests, design of user interface based on usability
PROG2004 Software Development	The process of developing a software product
PROG2053 Web Technologies	Development for web
PROG2002 Graphics Programming	Working with 3D computer graphics
PROG2005 Cloud Technologies	Fetching and using data from APIs

Table 1.1: Relevant courses

Although we have been through web development in our study program, we do not have much experience working with map services and creating 3D maps from map information. The initial source code we were given used libraries such as Three.js, a JavaScript library for creating and displaying 3D graphics for web, and OpenLayers, a library for creating and maps, and map protocols such as WCS and WMTS.

1.3.2 Motivations

After reviewing the possible project proposals, we landed on this one for the most part due to the openness of the scope. This gives us more creative freedom to choose a scope we are more interested in, and we can work with the client to find an exciting scope we both want to work on.

1.4 Constraints

The following constraints will affect our project.

1.4.1 Time constraints

Time is a constraint, as we have been given one semester to complete the project. The flexibility of the scope is an advantage here, as we have our base goal to which we can add more features. Should we find less time available than first anticipated, we can focus on completing the minimum goal.

1.4.2 Software, hardware constraints

Our end-users should use modern web browsers. Software and hardware are constantly moving forward, so the type of devices and web browsers are of a wide

variety. If a user uses an older web browser version than expected, there is a risk the product will not be able to run on it as support for WebGL is required to display the 3D map. Listed below are the minimum required browser versions.

- Google Chrome 9+
- Safari 5.1+
- Microsoft Edge 12+
- Firefox 4+
- Internet Explorer 11+
- Samsung Internet 4+
- Opera 15+

The project relies heavily on 3D graphics, which may result in poor performance on low-end hardware. Although focused on computer performance, the project is also expected to run on mobile devices and tablets. Approximate computer hardware specifications are found in section 3.3.

1.5 Project goals

Detailed in this section are the goals we wish the finished project should fulfill now and in the future.

1.5.1 Result goals

We want to achieve a fully functional 3D map of the municipality of Gjøvik, with the possibility of seeing regulation plans. The layout design should be responsive, meaning it changes to fit any device type. It should be compatible with the most popular web browsers to ensure that many people can access it and work well on different device types.

1.5.2 Effect goals

The solution should positively affect construction work by making regulation plans readily available and easy to find. It should help shorten the time required to plan construction by showing regulation plans and different terrain levels all in one program, negating the need to travel to the area in question just to be able to consider these aspects.

1.5.3 Learning goals

This project is an excellent opportunity to enhance our knowledge in web development, where we will primarily be using HTML, CSS, and JavaScript. We will also get the opportunity to learn new things, namely both graphics programming for the web using Three.js and WebGL, and working with mapping services such

as WMTS and WCS, and the library for creating maps in web development, OpenLayers. As most of our projects during our bachelor's have been for creating programs from scratch, this project will give us experience in reading existing code and understanding it to continue developing it.

1.6 Group organization

Both group members have the role of developer, which entails working on developing the product and documentation. Other parts are delegated as well, such as;

Group

Group leader, appointed to Fredrik Lønnemo. In addition to the leader role, Fredrik is a developer responsible for the back end and server-side aspects.

Nora Hønnåshagen Hansen is a developer on the team responsible for the UI design and development of the front end.

Supervisor

Mariusz Nowostawski is our appointed supervisor, giving advice and feedback on our progress throughout the project.

Gjøvik Municipality

Pål Godard is our contact person representing the client.

Geir Karsrud is the original developer of the project working for the municipality and acts as a secondary contact person for advice and feedback.

1.7 Thesis structure

The thesis is divided into chapters as described here;

- 1. Introduction - An introduction to the thesis.
- 2. Theory - Theory related to UX, an important aspect of our project.
- 3. Requirements - Different requirements related to the project.
- 4. Development Process - Details about the development process.
- 5. Graphical User Interface - Detailed explanation of the design of the user interface.
- 6. Technical Design - Details about the technical design.
- 7. Implementation - Detailing what we have implemented.

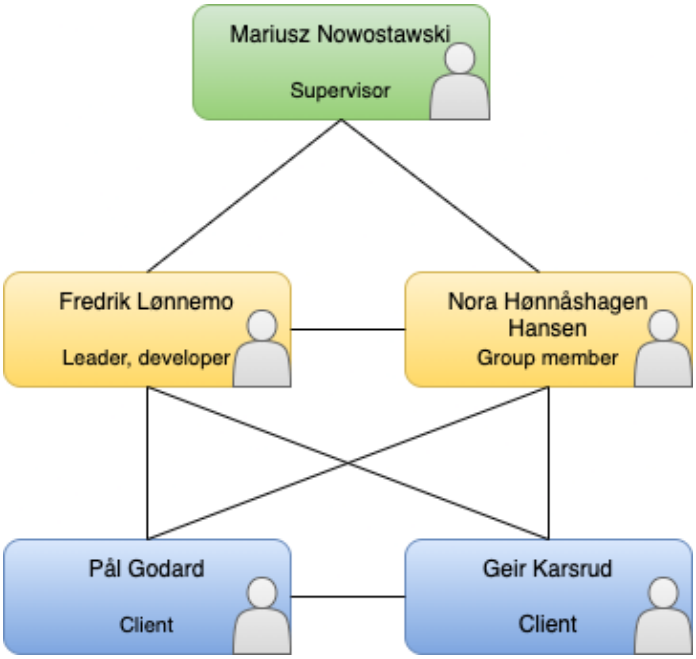


Figure 1.1: Group Organization diagram.

- 8. Quality Assurance - Details about the testing process, both on code and users.
- 9. Deployment - Details about the deployment of the project during development and after.
- 10. Evaluation - Evaluation of the final product and the process.
- 11. Conclusion - Concluding words and future developments.

Chapter 2

Theory - User Experience

A big part of developing a product is ensuring a good User Experience, UX. This chapter will discuss what UX is, its differences from UI, and what makes it important in the software development process.

2.1 Domain

The domain being discussed in this chapter is UX. We've chosen this topic as it is a vital part of our project, aiming to achieve a user-friendly product as possible, as well as being a vital part of most software development projects. All projects where we are developing a product for a market will need a good UX.

2.2 Purpose

The purpose of this chapter will be to research and understand the core aspects of UX and what makes it an essential part of development. A good UX can be quite difficult to achieve, depending on the broadness of the target audience. Designing a good product for a small segment of people is easier than a large segment with more significant variations.

2.3 What is User Experience?

2.3.1 User Experience vs. User Interface

The terms UI and UX are often used interchangeably; however, they are fundamentally different.
UI largely entails a product's overall look and design. It focuses on making a visually pleasing product through color, layout design, branding, etc. The look of the UI may negatively or positively influence the UX.
UX includes the UI design but also covers other important areas that should be satisfied to create an enjoyable experience, leading to a positive impression of the

brand. Key elements needed to be fulfilled to achieve a positive UX are described by Peter Morville in the UX Honeycomb shown in Figure 2.1 [7].
These elements include;

- **Useful**
  Ask yourself, "Is this product useful?". The users need to find a use for the product to succeed. This element is subjective, as products may be useful to one person but not another.
- **Usable**
  Usability is important but not sufficient on its own. If the users are unable to use the product, they will not see a reason to give it a chance, and it will fail.
- **Desirable**
  Make a product desirable through images, branding, identity, etc.
- **Findable**
  Users should be able to find what they need as easily as possible. Using too much time to find what they need can be off-putting and feel like a waste of time for the users.
- **Accessible**
  Making the product accessible for people with disabilities is important, as they make up a good portion of the world's population [8].
- **Credible**
  The design can potentially influence users' trust and belief in us and the content we offer.
- **Valuable**
  The product should provide value. If not for profit, it should bring value to its mission.

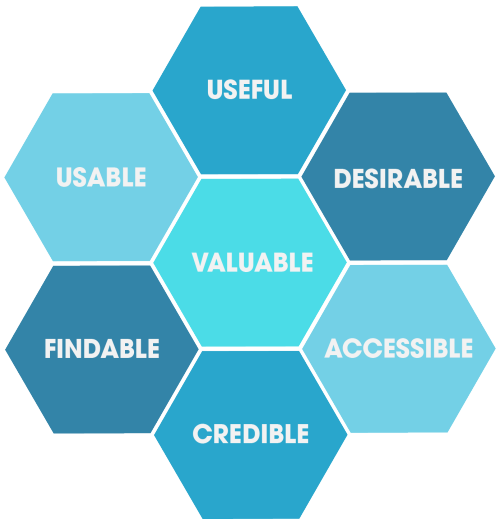


Figure 2.1: The UX Honeycomb graph created by Peter Morville.

2.3.2 Achieving a Positive User Experience

As developers, it is easy to get used to the layout and experience of our product, making us less suitable for judging the quality of its experience as an end-user. Some developers may simply rely on a gut feeling, which can result in a UI, which is confusing to navigate and less appealing for the end-users. Thus it is important to seek feedback from actors outside the software development team, with members of the product's target audience being ideal.

There are multiple possible methods of gathering user feedback, and we will now describe some of them.

The Guerilla testing method
The Guerilla is a testing method in which you bring a prototype to a public space, such as a shopping mall or a cafe, and seek out people to receive feedback from in exchange for a small gift. This test is better suited for the early stages of development when a prototype or MVP is ready and can be presented.

Surveys
A survey consists of a form containing questions, which can be, for example, sent to the users and other testers or posted publicly online for anyone to participate. However, this could result in many answers at the cost of the quality of the answers. With surveys, it becomes difficult, if possible, to observe the users' interactions with the product directly, and one can only rely on the honesty of their written answers. Additionally, the quality of the answers themselves may vary; some users may take the survey very seriously, taking their time and giving well-thought-out answers, while some may rush through it and give lackluster answers. Still, surveys are an easy way of getting many users' opinions if one can see past the issue with quality.

Usability tests
Usability tests are ideally performed in person but can also be conducted remotely, for example, over digital platforms. In these tests, the user is given a set of specific tasks to perform, where their actions and interactions can be observed. After the user has completed the tasks, they may share their opinions in a post-test interview or write them down in a form given to the developers. The advantage of these tests is quality assurance. Testers may not always be truthful, even if unintentionally. For example, a tester may claim to have had no issue finding a feature, even if you observed their struggle, simply out of embarrassment. In such a case, the credibility of the users' answers can be questioned, and those answers that reflect the user's actions will be more valuable.

Unmoderated remote usability tests
With unmoderated remote usability tests, the user is given a set of tasks to per-

form on the product without a moderator. The session is recorded for review by the team. When the user can use the product independently, it will feel more natural than in a moderated environment, leading to more "honest" behavior.

2.3.3 Using testing data to improve product

Once user testing has been conducted, it is time to use the data collected to improve the UX. When analyzing the data, one can look for what feedback is the most common. The most repeated answers will be more critical, while other, more specific ones will be down-prioritized. It is also important to consider the possibility you have in achieving their request if possible. Some feedback and requests may be repeated multiple times but are impossible to achieve in your specific product.

Chapter 3

Requirements

This chapter will discuss what this project should accomplish for the client and users.

3.1 Use case

We created the Use Case diagram based on what the client wanted the program to be able to do. The user in this diagram is a construction worker employed by the municipality of Gjøvik, chosen as they are the main target audience in this project. However, all actions in this diagram may be performed by anyone regardless of status or occupation since the solution will be available to everyone.

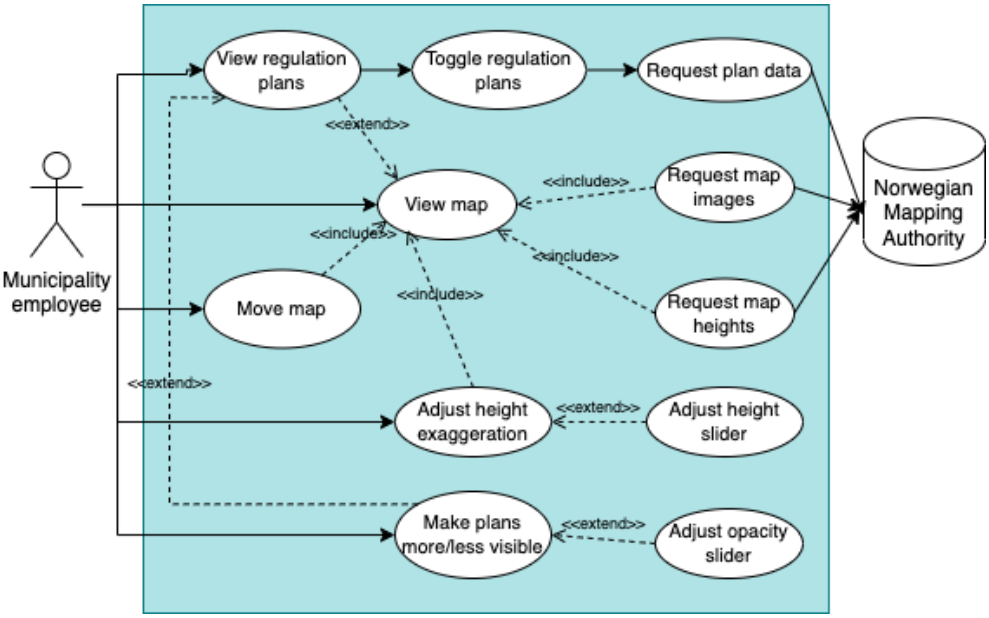


Figure 3.1: Use Case diagram. Made in Diagrams.net.

3.1.1 Actors

Construction workers – construction workers working for the Municipality of Gjøvik. The construction worker uses the 3D map to see a 3D visualization of regulation plans in the terrain.

3.1.2 User Stories

Below are more detailed descriptions of the user stories.

User Story	As a construction worker, I want to see a map of Gjøvik in 3D
Actor	Construction worker
Goal	See a map of Gjøvik in 3D
Description	The user enters Gjøvik Regionen's website. They click on the suitcase icon and find the button saying 3D. The map is loaded, and they view it.

User Story	As a construction worker, I want to see regulation plans in Gjøvik
Actor	Construction worker
Goal	See regulation plans
Description	The user enters the 3D map through Gjøvik Regionen. They then click the burger icon in the upper left corner. They check the checkbox next to the text label saying "Reguleringsplaner."

User Story	As a construction worker, I want to see height differences more clearly
Actor	Construction worker
Goal	Adjust the heights of the 3D map
Description	The user enters the 3D map through Gjøvik Regionen. They then click the burger icon in the upper left corner. They find the slider under the text saying "Høydeoverdriving" and slide it further to the right.

3.2 Backlog

We kept track of all tasks that needed to be done using Gitlab issues in our project repository. Here we would make one new issue per task, where the person whose responsibility it was to complete would be assigned to the task within the issue. We could comment on these issues with information related to the progress or others. Once the task is completed, its corresponding issue is closed.

User Story	As a construction worker, I want to make the regulation plans more visible
Actor	Construction worker
Goal	Make the regulation plans more visible
Description	The user opens the 3D map through Gjøvik Regionen. They then click the burger icon in the upper left. They find the slider under the text saying "Opasitet" and slide further to the right.

3.3 Performance

Most modern computers and mobile devices can run the web solution. Being a project heavily reliant on 3D graphics processed on the client side, it benefits from a dedicated graphics card performance-wise. Additionally, more modern hardware has proven to improve the experience.

We have worked out these possible minimum requirements for computers:
Graphics processor - Intel Iris Plus Graphics
Processor - 2,3GHz Four Core Intel Core i5
Memory - 8 GB 2133 MHz LPDDR3
OS Version - Windows 7+, OS X 10.6+

The version requirements for popular web browsers can be found in section 1.4.2.

3.4 Functional

The project should provide a 3D map covering the municipality of Gjøvik. It should allow the user to see a map of the municipality in a new way by displaying heights in the terrain, accurate to the real-world environment. The user should be able to view regulation plans in the municipality on the map. The plans should be accurately placed (no artefacts should be displaced). The map should be adjustable in ways such as rotating it to see from different angles, changing the exaggeration of heights of the terrain higher or lower, and seeing the regulation plans or underlying map more clearly by adjusting the opacity of the plans, all with controls readily available within the web interface.

3.5 Operational

The product should be deployed on Geoinnsyn by Gjøvik Municipality on their servers, making it available for everyone through this web portal. It will be able to run on all the popular web browsers running a version compatible with WebGL, as listed in 1.4.2 and on desktop computers, tablets, and mobile devices running one of the compatible web browsers. It should perform just as well on a computer as a mobile device and on as low-end hardware as possible, as mentioned in 3.3.

The solution should get its map data from the Norwegian Mapping Authority, which includes data about the heights with WCS and images of the Topo4 map colors and regulation plans with WMS.

3.6 Security

Regarding the data the map is constructed with, security is not our primary responsibility as the government provides this and is thus expected to be correct.

The dependencies should be current to ensure potential security holes have been patched. There should be no possibility of disruption in the loading of the dependencies.

3.7 UI

The UI should be intuitive and have both universal and responsive design. It should be easy to navigate and find the desired functionality on all platforms by avoiding cluttering and intruding on the layout. The UI should enhance the UX, giving the user the best experience possible.

Chapter 4

Development process

In this chapter, we will go through the details of the overall process of this project, how the initial plan was thought out, and how the actual process turned out. We will review the software development model used, what project management tools were utilized, look at the Gantt diagram, and talk about how we conducted meetings with the group, the supervisor, and the client.

4.1 Project characteristics

Although we have been through software development in many projects during our study programme, this project was of a much bigger size than what we have done in previous courses.

Already from the beginning, we were aware that we would be working with libraries and frameworks largely unfamiliar to us, concerning both the map technologies and 3D graphics library in use, and should therefore keep in mind that there could emerge a need to spend extra time studying these to gain a better understanding of what we were to work with in the coming time.

4.2 Software development model

At the beginning of the project, we looked through multiple possible software models we could utilize in this project and considered both benefits and drawbacks. Most of the models we reviewed were better suited for projects where one is developing a brand new product from scratch, where having a project which is not complete is part of the process. Although a model of these characteristics could be utilized, we wanted a model more fitting for developing further on a project already started.

After reviewing the models, we settled for the incremental software development model for this project. The biggest reason behind this is the model's character-

istic, which entails having a functional base that is always in a working state. This base project will then have new features and additions, making it more and more fleshed out as the development process goes on. This way, the product can technically be released at any time, although how rich it is in features will depend on how far it has been in development.

The characteristics of this model were the best fitting with our project being based on working on the already in-development project provided by the municipality, being the base. The initial project given was already being hosted on their website and technically functioning; however, lacking features the client wished to have added.

At the beginning of this project, we were given some starting features requested by the client to work with. We created issues for each of these, giving us a great number of tasks to choose from from the start. At the beginning of each sprint, we would review the issues at hand, judging their weight and each team member would take on one of the most important ones. In some instances, other less important issues could be prioritized if they were expected to take a short time to implement. When a team member has finished their task early on in the sprint, they would inform the other member of the progress and take on additional tasks by picking the next most important one again.

In the next sprint, we could show off the progress made during the past week. If the task has not been completed, the current process would be shown, and we could give each other feedback. In some cases, if much time spent on a task had shown little or no progress over time, the task would be put on hold, and the group member would move on to another task to ensure progress is still being made on the project, and then being able to move back to the task at a later time.

As our supervisor meetings were on the same days as our sprint meetings, we would also be able to discuss the feedback given during this meeting and take in the suggestions given in these meetings. On days when we also had a meeting with our client, the feedback from these would also be discussed, and any further requests given by them would be added as issues and, if critical, would become the focus of the upcoming sprint. This allowed us to go into detail about the feedback and the time to come.

In addition to weekly sprints, we would keep in touch regularly through our chosen communication platform Discord, where we would give each other updates about progress made or when important decisions regarding the task being worked on have been made, for example, once someone has moved from one task to another. This would help keep each other always informed on the current state and progress of the project.

4.3 Project management tools

We used a shared folder in `Google Drive` to share documents to be worked on together and keep backups of these. This contained documents such as meeting minutes, surveys and their results, diagrams, and other images.

Other resources, such as useful links and resources, were shared in dedicated text channels in the group Discord server for easy access. In addition, digital meetings and other communications were largely held in Discord with the group, as well as meetings with the supervisor, should these meetings need to be held digitally. Meetings with the client were held in `Microsoft Teams`.

4.4 Version control and code organization

For version control of the code, we used NTNU in Gjøvik's Gitlab instance, where we kept a repository to which the project's source code was stored and changes to it were committed. Minor changes with little impact on the code could be pushed to the main branch directly without worries, while significant changes and feature additions are pushed to a new branch until it is ready for merging. This ensures the main branch is always functioning and prevents a new feature from introducing unwanted problems or bugs, which could take too much time trying to fix without a backup. Whenever a group member has finished a new addition and is ready to merge it with the main branch, a merge request is made, which the other group member has to approve before it can be merged. Once merged to the main branch, the newly merged branch is deleted, and the associated issue or issues are closed.

4.5 Gantt diagram

In the Gantt diagram in figure 4.1, you can see how we initially planned the semester work to look while creating the project plan at the beginning of the year. As you can see, we had it set up based on the possibility of adding multiple data visualizations in our project with multiple back ends. However, the actual development process of the project turned out differently than we had initially planned.

4.5.1 Planning phase

During this phase, we set up our project plan for the semester. With the open nature of the task description, we spent much of this time defining and narrowing down our project scope to be more specific and to find out what aspects we would prioritize and which we could make optional. We ended up settling for one main goal to complete, being the further development of the 3D map itself and making it more complete by adding the regulation plans and other UI elements, with adding

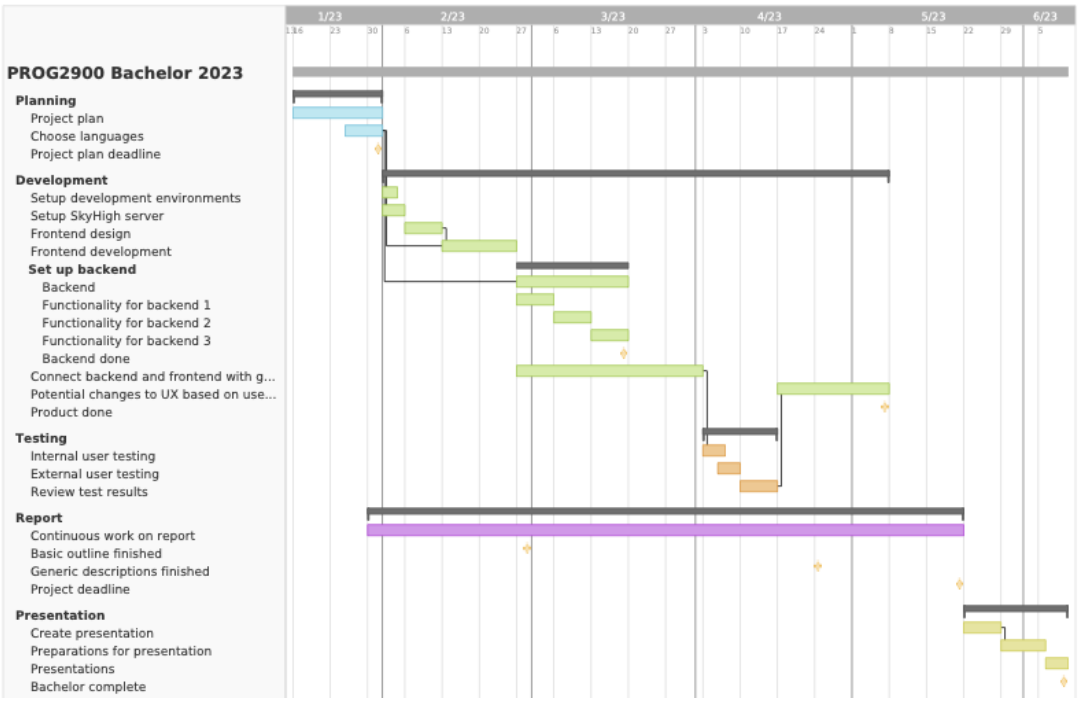


Figure 4.1: Gantt chart.

additional visualizations to it being optional and could be discarded if the time constraints showed this not to be possible.

4.5.2 Front end design

Originally, this phase was planned to be dedicated to the design of the front end. However, as we quickly realized, this phase would instead be dedicated to reading, studying, and understanding the source code we had been given, as well as researching and learning about the frameworks and libraries in use, including the WCS and WMTS services.

As neither of us had used the Three.js and OpenLayers libraries before working on this project, we decided to dedicate time to looking at the documentation of these, in addition to studying how these were used in the project itself, to make working with them an easier process in the long run.

4.5.3 Front end development

The front-end development process mainly consisted of making adjustments to the visual aspects of the project, including the 3D map and UI elements. We would take one piece of the design at a time, starting with work on the UI elements first, then moving on to the basic shape of the map, and then the additional features

requested by the client, like the regulation map and further adjustments to the map. We would take care of issues in the GitLab first, then receive new additions from the client during meetings.

4.5.4 Back end development

The pre-existing code's back end was not much developed, as most of it worked with map-specific technologies unfamiliar to us, so a lot of time was spent figuring out how this worked. For delivering the website to users, we used Apache because Fredrik had previous experience with deploying on it in DCSG2003 Robust and scalable services. Caching was implemented on the test-server, to aid performance, but does not feature in the final product as the municipality will host the project themselves and have explicitly stated they will prefer a simple "plug and play" approach where minimal configuration has to be done to the server.

4.5.5 User testing

Some user feedback was given on request sporadically during development, specifically when developing the design layout. Our client contact person would arrange user tests with the internal employees of the municipality. Once we had implemented a sufficient amount of features as we had wanted, we sent the code to our client, who would host the project on their internal servers and ask employees to test it out. Other user testers not employed in the municipality, representing the general public, were also asked to test the project. More details about the user testing are described in section 8.2.

4.5.6 Changes based on user feedback

This period was dedicated to reviewing the feedback given by the client. We sent the project to our client first and quickly received some feedback to address. We then spent a week fixing some important things before sending an updated version of the code to the client, which was then hosted, and a survey we created was sent to workers within the municipality. More details about the user tests are described in section 8.2.

4.6 Meetings

All meetings were held on Thursdays. Weekly meetings were held with the supervisor and group, and meetings with the client were held every other week.

4.6.1 Group meetings

Internal group meetings were held on Thursdays, where we would discuss what had been done and determine the focus areas of the upcoming sprint. These were largely held online through our chosen communication platform Discord.

4.6.2 Supervisor meetings

We held physical meetings on campus with our supervisor weekly on Thursdays. We would start the meetings by showing off the project in its current state, followed by our supervisor giving feedback and coming up with suggestions.

4.6.3 Client meetings

We set up the possibility of meeting with the client every two weeks, depending on whether each party found it necessary. These meetings were held on Microsoft Teams. We would use these meetings to show off the progress made on the project, and they were given the opportunity to give feedback. In addition, we communicated with the original developer of the 3D map Geir Karsrud who would, from the 23rd of March, join our client meetings and to whom we could ask for advice and clarifications if needed.

Chapter 5

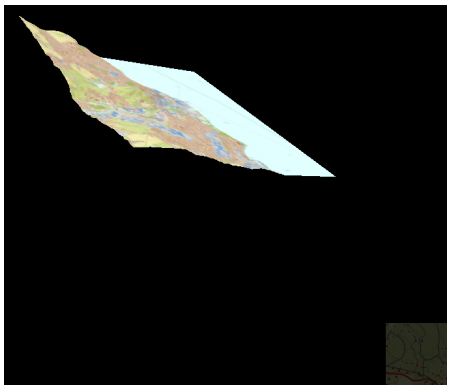
Graphical user interface

This section will discuss UI development process.

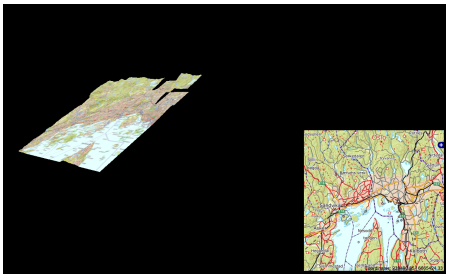
5.1 Initial Design

The 3D map of the project are created using the Three.js library, a high-level cross-browser library for creating 3D graphics in web browsers, using WebGL.

The user interface of the initial code we were given can be seen in Figure 5.1a. It consisted of a square-shaped map centered on the screen, a small button in the upper right reading "3D" for reloading the map geometry. Moving the map around to where no part of the map was located in the center of the screen would cause additional tiles of the map to be loaded in that location. A semi-transparent mini-map was situated in the lower right corner, which moves into the screen and gains opacity on mouse hover, shown in Figure 5.1b. Unlike the 3D map, the mini-map is made using the OpenLayers library.



(a) The map is centered in the middle on an empty background.



(b) The mini-map in the lower right corner pops into the screen when hovered over.

Figure 5.1: The Initial User Interface.

5.2 Design Changes

5.2.1 Camera Controls

TrackballControls are used for controlling the camera in the Three.js Scene. This allows the user to move the map infinitely in all directions using a mouse on computers and touch gestures on mobile devices or other devices with a touch screen. The amount of freedom in the movement resulted in users having the ability to turn the map all the way around, viewing the backside. This could be avoided by changing the camera controls to OrbitControls, with which we could limit rotation on the x-axis and y-axis. However, one feature the Municipality needs is the ability to turn the map on the z-axis, which could not be accomplished with the OrbitControls library as it keeps a constant up-vector. Thus camera controls were switched back to TrackballControls.

5.2.2 User Interface Changes

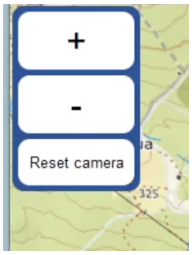
The UI of the initial project was very simple, with few controls available, the only button added being used to reload the 3D geometry. We started out by adding essential button controls, such as for zooming. The buttons were added using HTML tags with CSS styling. The original design of these buttons made them big in size and placed them in a blue-colored container in the upper left corner, as shown in Figure 5.2a. The initial size of the map created much unwanted empty space on the screen, so the map was changed to fill out the entire screen from the beginning.

During the design development, we sought feedback on the layout from external actors who had no connection to the project. Some feedback was given regarding the buttons' placement, as some found it more natural to place these in one of the bottom corners.

To gather more inspiration, we looked at various successful web mapping platforms such as Google Maps [9], Apple Maps [10], and Kommunekart [11], whose UIs can be seen in Figure 5.5 . Common design traits found were; small conjoined zoom buttons, often with rounded edges and in white or dark grey color, depending on whether the device was using light or dark mode, placed in one of the lower corners of the screen. Additionally, the map takes up most of the view, with few elements obstructing the view. Maps with many features leave these hidden in a collapsible menu.

Based on these traits, we changed the button design to be smaller and white, with a slight blue hue when hovering over it with the mouse and darker blue when it is clicked, to make it clear to the user that the actions are being performed. To avoid obstructing too much of the map with the buttons, the buttons' color was given some transparency but is still opaque enough to be visible and easy to find.

(a) Initial look of the UI buttons.



(b) Close-up of the button layout.

Figure 5.2: Interface with buttons added.

Although all the desktop maps we viewed had optional buttons for zooming in or out of the map, the mobile versions most often omitted these buttons in favor of touch gestures. We adopted this feature by not displaying the buttons if the site detects it is being run on a mobile device, keeping as much as possible of the small screens obstruction-free, shown in figure 5.3.

One of the main features requested by the Municipality was the ability to visualize regulation plans on the map as an overlay. The plans were fetched from the Norwegian Mapping Authority's WMS APIs and loaded as a texture for the 3D-Map. We combined the topographic map with the regulation plan map 50/50 using a custom shader written in GLSL to display both map textures simultaneously. Both the topographic and combined map textures were loaded into the 3D object, with only one being shown on the map at the time. A collapsible menu was added in the upper left corner where the regulation plans could be toggled on and off.

5.2.3 Changes Based on User Feedback

After meetings with the Municipality, we received feedback on the UI.

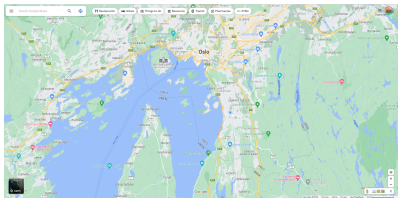
They suggested adding a message in the map which would display while the map is loading to give the user some indication of whether or not the map has finished loading. Using a *LoadingManager* from the Three.js library, a message is displayed while the textures are being loaded, with a counter showing progress.

Figure 5.3: No zoom buttons on mobile layout.

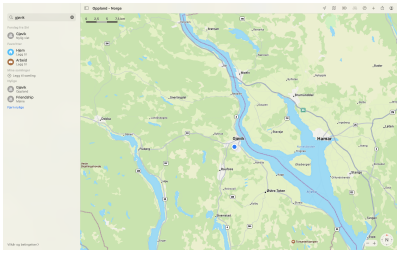
(a) Updated look of the UI buttons.

(b) Close-up of the updated button layout.

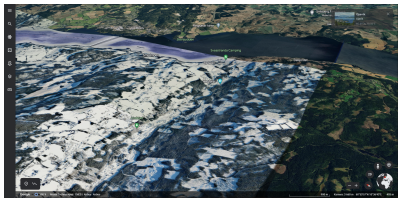
Figure 5.4: Updated layout.



(a) Google Maps desktop layout

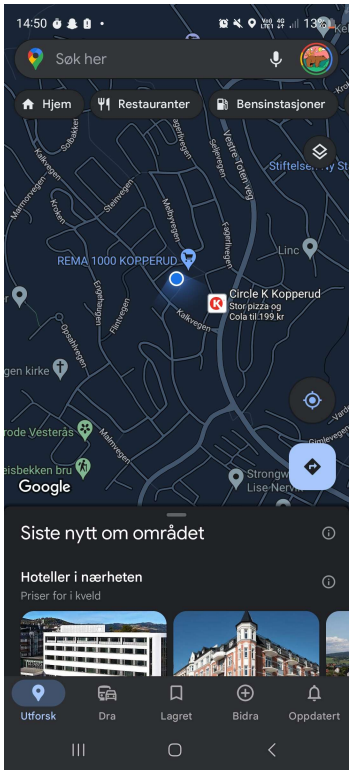


(b) Apple Maps desktop layout



(c) Google Earth desktop layout

(d) Kommunekart desktop layout



(e) Google Maps mobile layout

Figure 5.5: The layouts of widely used map services. The top four images show how a map may be designed for desktop use, while the bottom image shows a layout design for mobile devices.

The client also requested the ability to adjust features in the browser, such as height exaggeration in the 3D geometry and the opacity of the overlay. Using ranges, these were added to the menu for easy access.

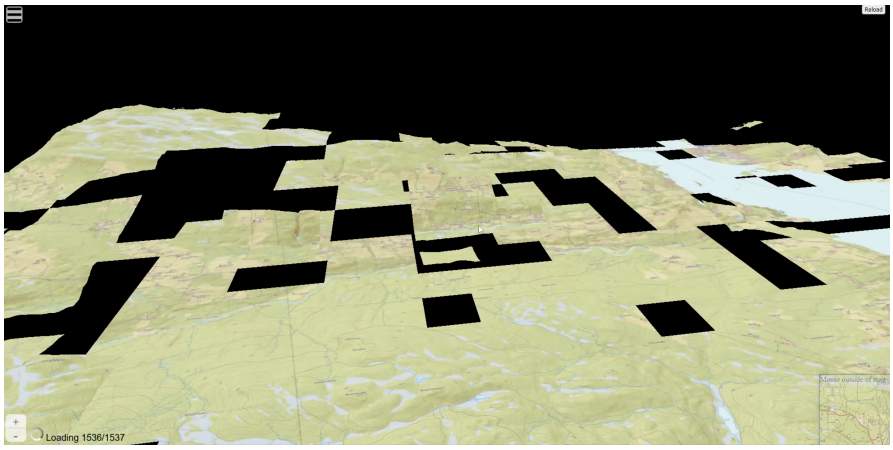
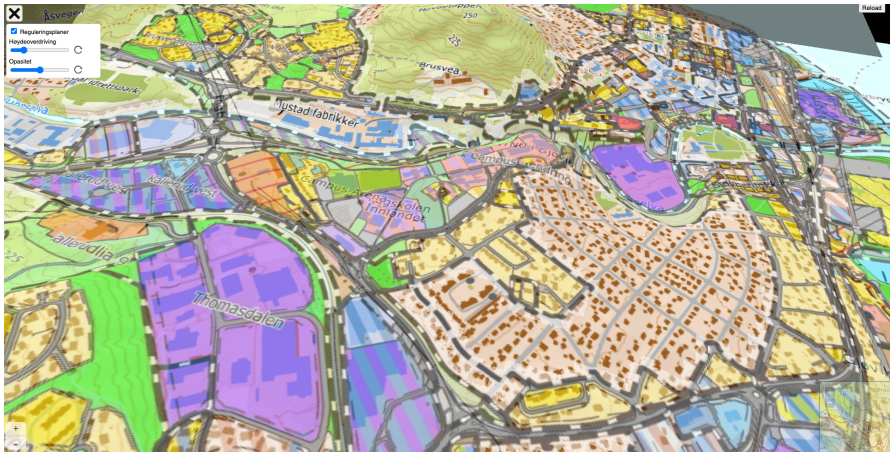


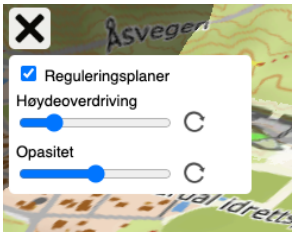
Figure 5.6: The menu is added to the upper left corner, and the rendering distance is shortened.

5.2.4 Final Design

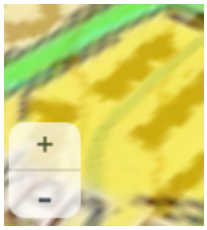
In Figure 5.7a, you can see the final layout of the UI as it was delivered.



(a) The final layout of the UI.



(b) Close-up of the menu.



(c) Close-up of the final layout of the buttons.

(d) The final UI on mobile.

Figure 5.7: Final UI layout.

Chapter 6

Technical design

In this chapter, we will describe the overall system architecture, what alternatives we considered during the planning phase, and go into detail about the front end and back end.

6.1 System architecture

In Figure 6.1, you can see a simple diagram of the overall design of the resulting system architecture. The project is accessible through the Municipality's Geoinnsyin portal, which loads the HTML entry point, which leads the user to the 3D map, which fetches data from the Norwegian Mapping Authority's endpoints.

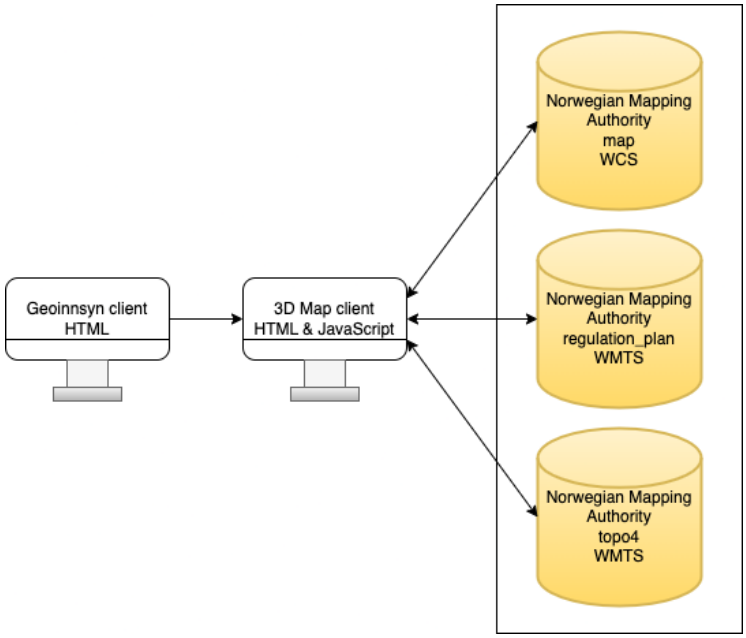


Figure 6.1: Overall system architecture.

Although using only one web page, the architecture can be characterized as a Multi-Page web application as it is structured in a way that does not use technologies used by common Single-Page web applications. The languages used in the solution are HTML, used to construct the basic page layout and loading JavaScript files, CSS to add styling to the HTML elements, and Vanilla JavaScript, which is used for the fetching of data and creation of the 3D map using various libraries.

The processing of the 3D geometry and rendering is done on the client side due to the nature of the Three.js library, which is built on WebGL, a graphics library for web browsers allowing for rendering without the need for the user to install additional plugins but in turn, performs its processing on the end-user's GPU, or graphics processing unit. As a result, most of the architecture is focused on the front end of the solution, with the back end aspects referring to anything the user cannot see, such as the fetching of data from the Norwegian Mapping Authority's endpoints and processing of these. In 6.2, you can see a more detailed system architecture diagram showcasing the program's structure.

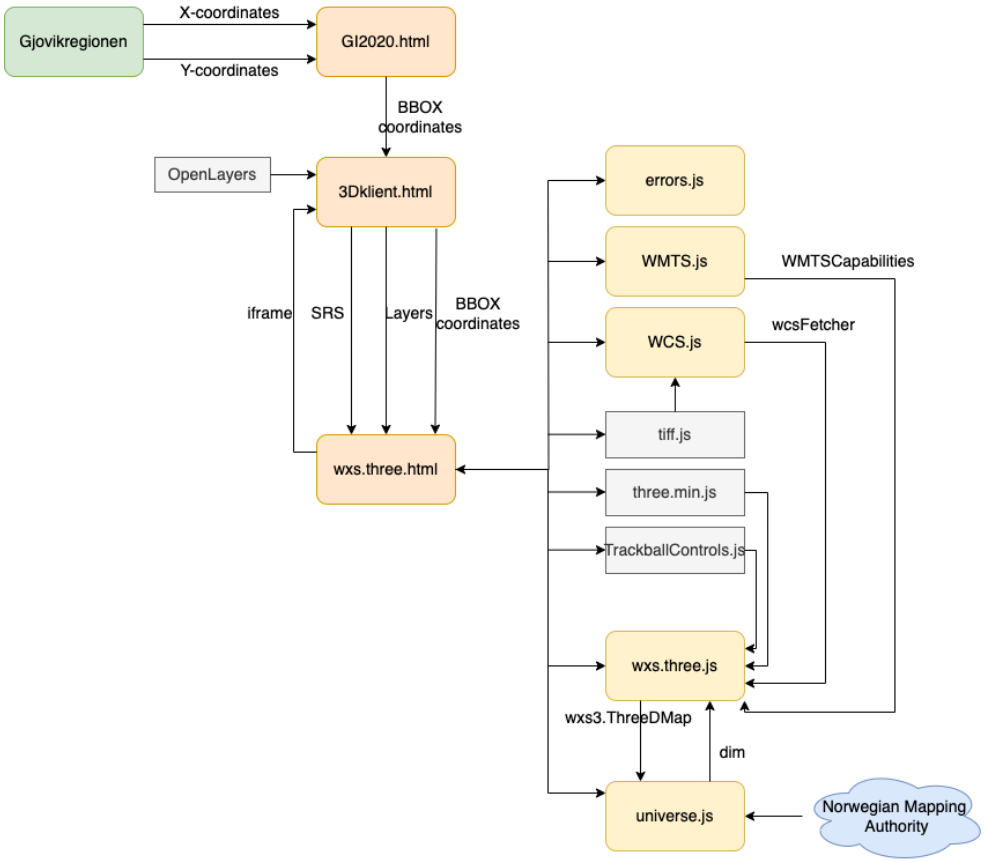


Figure 6.2: Detailed system architecture.

6.2 Architecture Alternative

While considering how to continue work on this project, we looked at alternative architectures different from the one already established. As mentioned in the previous section, the project was developed using Three.js to create 3D graphics. Besides this, there exist other alternatives useful for achieving similar results. We will in this section discuss the alternative we considered.

A different possibility we found was to convert the project to using Unity with WebGL rather than Three.js. Our client showed us a similar project called 3D Amsterdam during one of our initial meetings, which web solution can be found here, with its code repository found here. Unlike the project in progress by the municipality, this project uses Unity for its 3D graphics processing.

While using Three.js would let us simply pick up where the last development had stopped and continue work right away, had we chosen to use Unity, we found we would need to rewrite the entire project from scratch for the other engine. How Three.js and Unity work and are developed are fundamentally different. Three.js is a 3D engine specifically used for web development, where everything needs to be coded by the developer, but it is well suited for web development and integrates gracefully with web applications, being able to run right on the web page using JavaScript. In contrast, Unity is a game engine that, while most commonly used for developing desktop applications and video games, can also build programs for web pages using extensions for building for WebGL. However, unlike Three.js, which can be loaded right into the web browser, the Unity application must be compiled before being uploaded to the web application. It takes, in general, longer to load than Three.js, and often is bigger in size once compiled.

Ultimately, we chose not to go with this model as we were gonna be developing a project already using Three.js. We would need to read and understand the Three.js code regardless, as understanding what the program was accomplishing would be necessary to convert the project to use Unity, especially finding out how to use the data from the Norwegian Mapping Authority in Unity. To spend time doing this and then converting this code to Unity would be very time-consuming and leave little time to develop the map solution further, so we decided to continue the development using Three.js.

6.3 Front end

The source code we were given was written in HTML, CSS, and JavaScript. Three HTML files make up the entry point to the 3D map, styled with CSS within each

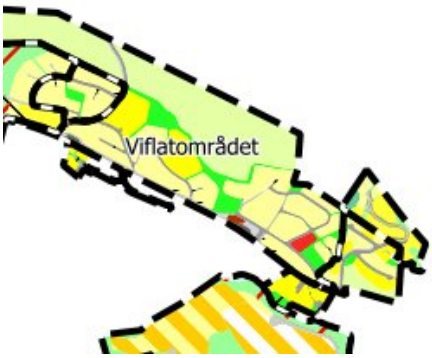
file. The solution is entered through GI2020.html, which loads 3Dklient.html with parameters in the URL as an *<iframe>* element. 3Dklient processes and loads the mini-map using OpenLayers, and loads the wxs.three.html file with the parameters in the URL as well, also as an *<iframe>* element. Within the wxs.three.html file, the local JavaScript files are loaded, as well as the libraries used in the solution, including both Three.js and the camera controls, TrackballControls. As camera control libraries are not included in the Three.js library, these need to be loaded separately.

The 3D map is created using the Three.js library, which is used to define all the elements of the 3D scene and display them. The 3D map receives data from the back end to form the 3D objects, as section 6.4 explains. This includes WMTS images in both PNG format for the Topo4 images, and JPEG for the regulation plan images, and WCS for the height information.

6.4 Back end

This project's front end and back end are tightly tied together. In this section, we will discuss the parts of our project which are not directly visible to the user. The data is as mentioned in section 6.1 fetched from the Depth DTM WCS endpoint, Topographic Map 4 cache endpoint and Zoning plans WMS endpoint. In figure 6.3, you can see a pair of tiles as the endpoints deliver them. The GeoTIFF tiles fetched from the WCS can be opened as they have the *.tif* file extension. However, their appearance is simply pure white, as the data embedded in the file is not visible in the image itself.

(a) Tile from the topo4 layer.



(b) Tile from the regulation_plan layer.

Figure 6.3: Example tiles from the Norwegian Mapping Authority.

When entering via Geoinnsyn, X coordinates, and Y coordinates are passed as URL

parameters to the HTML files, which are turned into BBOX coordinates, which are used when calling the Norwegian Mapping Authority's endpoints, in addition to a SRS code, a Spatial Reference System code, given in the URL. The SRS code is used when measuring coordinates, which is necessary to ensure the BBOX coordinates correspond with the location in the world that should be shown.

Chapter 7

Implementation

This chapter will go into more detail about the methods used to achieve the resulting solution.

7.1 Already implemented

As this project is one where we have further developed an existing project, we will talk about the parts of this project we did not implement in this section.

7.1.1 wxs.three.js

The original code the project is based on is the open-source project wxs.three.js [5]. This project was developed by Sverre Iversen, Atle Frenvik Sveen, Carsten Mielke, and Jarle Pedersen. The repository can be found on Github here. It is a 3D solution for visualizing data from the Norwegian Mapping Authority.

This project works as the base of the web solution, where the functionality for creating the map itself is created. As explained in chapter 6, the project calls the Norwegian Mapping Authority's WCS depth endpoint for getting the heights of the terrains and calls their WMTS endpoints for textures. The 3D map in this project is made using *THREE.PlaneGeometry* objects from the Three.js library. These are, by default, flat square objects whose shape can be manipulated, which is changed to have the form of heights in the terrain as seen on the map.

The depth endpoint serves data about heights in the terrain in multiple formats, where this project takes use of the data found in GeoTIFF format, an extension of the TIFF file format modified to contain georeferencing and geocoding data. The height data in the GeoTIFF files are parsed using the *TIFFParser* found in the tiff.js library. The heights in a *THREE.PlaneGeometry* object is changed to be equal to the values found in the GeoTIFF, which ensures the 3D aspect of the map is achieved.

The textures used to paint the map are fetched from the Norwegian Mapping Authority's WMTS endpoints, which serve multiple possible image formats. This project uses the images in the PNG format, a widely used image format benefiting from lossless compression resulting in high-quality images. When creating the *THREE.PlaneGeometry* objects, a material must be created to give the Mesh color or texture. In this project, the developers made use of *THREE.MeshBasicMaterial* to color the map, a type of material that will always be visible even if there is no light source in the 3D Scene. This material is given a texture from the WMTS endpoint in PNG format and combined with the height data from the WCS to create the 3D map we can see in the project.

Information about the depth WCS endpoint can be found on the Norwegian Mapping Authority's website here, and information about the Topo4 and regulation plan WMTS endpoint can be found here.

7.1.2 Modifications by the municipality

Modifications were made by Geir Karsrud, who worked on making the wxs.three.js project centered on Gjøvik and making it suitable for use by the municipality. Geir created the GI2020.html file, which acts as the entry point for the 3D solution, which would be opened through their Geoinnsyn site. The URL used by Geoinnsyn to enter the web solution contains query data: an X-coordinate, a Y-coordinate, an SRS code, and a multiplier for the heights. The X and Y-coordinates are used to calculate the BBOX, or bounding box, which is an area in a map defined by two Latitudes and two Longitudes, done by taking the x-coordinate and creating two new x values; one by adding 2000 to the x-coordinate, and one by subtracting 2000. The same process is done to the y-coordinate. The BBOX and SRS values are used in the calls to the Norwegian Mapping Authority's endpoints, while the multiplier exaggerates the height differences in the 3D map, with *2* being the default value sent in the URL.

Geir Karsrud also created the 3Dklient.html file, a client in which the mini-map is created using OpenLayers. The map was initialized and loaded multiple layers from endpoints provided by the Norwegian Mapping Authority. Using CSS styling, the mini-map was loaded in the lower right corner, partially hidden off-screen, with some transparency set, and would move further into the screen and gain full opacity when hovered over with the mouse pointer. The mini-map loaded multiple layers from the Norwegian Mapping Authority, which could be switched between using controls within the mini-map.

7.2 Dependencies

The original source code given had multiple dependencies, most of which were older than the newest available versions. One of the first things we did was up-

date the Three.js library to the most recent version as of January 2023. After the library had been updated, we needed to make changes to the existing code, which was still using deprecated functions related to loading and creating textures, and replaced these with the currently supported functions. The dependencies were also switched from being loaded from online sources to being stored locally in the repository to improve security. Doing this reduces the risk of attacks such as spoofing, as loading dependencies from online sources may result in the connection rerouting to a different or potentially hostile source.

7.3 3D Map

The map images are loaded onto the 3D Meshes as using *THREE.MeshBasicMaterial*, as mentioned in 7.1.1. A Mesh in Three.js can natively only show one texture material at a time, whereas the map originally only showed the Topo4 texture. One of the main requests from the municipality was the visualisation of regulation plans on the map. When researching possible ways of displaying the regulation plans on the map as an overlay on top of the Topo4 texture, we found two possible options for consideration;

One of the options considered was to create two identical 3D map objects. One would be given the normal Topo4 texture and be displayed at full opacity, and a second identical map object would be given the regulation plan texture. The map object with the regulation plan texture could then be placed slightly above the Topo4 textured object and have some transparency set to it, making the colors from both objects visible at the same time, as well as avoiding any obstruction of the information contained in the Topo4 map texture, which already displays some information such as location names. However, we considered the performance issues this would introduce, as the number of tiles rendered on-screen would be doubled due to needing to display two objects instead of only one. As a result, we came to the conclusion that the performance cost would be too great on an already graphics-heavy project and looked to the other option.

The second option we considered and ultimately ended up implementing was using custom shaders to combine the two textures into one before assigning it as material to the 3D map Mesh. As Three.js is based on WebGL, which is a low-level graphics language for creating and displaying graphics in web browsers, we had the possibility to create a custom shader for this purpose using the GLSL shader language, a language made specifically for writing shaders in 3D programming with a syntax similar to that of the C/C++ languages. This shader language is used also in OpenGL, another low-level graphics language used for creating 3D graphics for desktop applications, which we had experience using from the Graphics Programming course.

This shader would take the Topo4 images and regulation plan images, both of

PNG format, as texture uniforms, which were mixed using the *mix()* function in the fragment shader, which takes three values; The first two being two different values to interpolate, and the third being the weight between them. The regulation plan images consisting of only the regulation plans filled the empty space between these with a pure white color, which showed up on the mixed texture, making the map around the regulation plans look faded. To combat this, a check of the pixels was added, which, if the color was found to be pure white, was replaced with the corresponding pixel color on the Topo4 texture. The fragment shader used to achieve this can be seen in 7.1.

Code listing 7.1: The fragment shader used to mix textures into one.

```
varying vec2 vUv;
uniform sampler2D u_texture;
uniform sampler2D u_overlay;
uniform float u_opacity;
void main() {
  vec4 color1 = texture2D(u_texture, vUv);
  vec4 color2 = texture2D(u_overlay, vUv);

  if (color2.r == 1.0 && color2.g == 1.0 && color2.b == 1.0) {
    color2.rgb = color1.rgb;
  }

  gl_FragColor = mix(color1, color2, u_opacity);
}
```

The last value in the *mix()* function, determining the amount the second texture should be mixed with the first texture, was originally set to a hard-coded value of 0.5, making each texture appear 50/50. A useful feature requested by the client was the ability to change the opacity within the program. To do this, we set the last value of the *mix* function to a uniform with a default value of 0.5, which gives us th ability to change it dynamically. We created a slider with the *<range>* tag in the map's UI, which would call a function from the wxs.three.js file *oninput*. The reason we are using *oninput* rather than *onchange*, is the latter will only send the values to the function once the user stops moving the nob and lets go of it, while *oninput* keeps sending the value while the nob is still being moved. The scale change function gets the range element by ID and reads the current value of the slider somewhere between 0 and 10, divides it by 10 to get a decimal number less than 1, and sets the value of the opacity uniform to be this value.

Each 3D object is given the two created textures, the normal Topo4 and the mixed texture, in an array, with the texture at index 0 being the one displayed on the map, with the Topo4 texture being the default. When toggling the regulation plans to be on, the locations of the textures are swapped by placing the mixed texture in index 0. Although a possibility could be to have the texture with no overlay simply be the mixed texture with the mix value set to 0, we wanted the project to have the possibility of adding other overlays in the future by adding these to the array of textures and giving the ability to switch between these.

The source code we were given used a Raycaster object to determine whether or not a tile has been processed and, if so, should be loaded. It was used to cast a ray, which, if hitting a tile belonging to the *backgroundGroup*, which consists of tiles that are not currently on-screen, would call the function for processing tiles and placing them in the *foregroundGroup* matrix containing tiles that have been processed and are now visible to the user. As the original map object was one small square, the user would have to use their mouse or touch gestures to drag the map around the screen, forcing it to load more tiles if they wanted more of the map to be loaded on the screen.

As mentioned in chapter 5, the map was changed from a small centered square to filling out the screen better. To start with, we would use a check of the screen aspect ratio and increase the number of tiles loaded at the beginning depending on the screen size, where a wide aspect ratio would load extra tiles horizontally, and a tall aspect ratio would load extra tiles vertically. However, we had no way of loading additional tiles when the map was moved, so moving the map would reveal the empty backside. The original project already had the *raycaster* function implemented which would load new tiles where the ray hit, so we wanted to see if we could do this over the entire screen instead of just one point.

After looking into the Three.js documentation, we discovered we could achieve this by creating a Frustum object from the Three.js library, an object used to detect what elements are on-screen, commonly used to elegantly exclude objects on-screen from rendering. We used this to detect if a tile belonging to the *backgroundGroup* has entered the view of the camera and is within the view of the screen, in which case it should be processed and added to the *foregroundGroup* matrix once the process is finished. The Frustum object is given the same size as the Scene's camera object when created, meaning only the tiles currently in view are rendered at the time. Thus, tiles that have been processed and lay within the *foregroundGroup* matrix will not be rendered once their position is outside the screen and outside of the Frustum.

During development, we were given feedback from our client, who could not run the solution properly due to performance issues. To combat this, we moved the far-plane, or the back, of the *PerspectiveCamera* object used by the Scene closer to the near-plane, or the front, changing from its original distance value of 5000000 down to only 10000. Since the Frustum size is set to the size of the camera, the number of tiles within the Frustum at once was drastically reduced, and new tiles stopped loading much earlier than before, where the far-plane position was so great the tiles did not stop loading. We found this to be the best solution, as it did not hinder the experience, with the tiles loading further towards the far plane not in focus and would be too far away to see the information on. Fewer tiles on-screen once greatly helped reduce the performance issues, and the users need

not worry when tilting the map, as tilting would lead to the biggest number of tile loads.

The option to edit the exaggeration of heights was achieved by manipulating the scale.z values of the map tile objects, multiplying their heights equally. The default value already set by the original developers was 2.0.

7.4 Lighting

We looked at implementing different lighting in the Scene to create shadows. While the 3D aspect is more visible when tilting the map, the heights are not as obvious when the camera is pointed straight down onto the map. Adding a light source able to cast shadows could make terrain differences easier to see and give some realism to the map.

The Scene already had *AmbientLight* added, which lights up all objects equally, although unnecessary as the type of material used in the 3D objects, *MeshBasicMaterial*, will appear evenly lit regardless of lights in the Scene, in addition to not being affected by shadows. Thus, before looking into the lighting, we changed the material used to *MeshStandardMaterial*, which can cast and receive shadows. We experimented with dimming the ambient light and adding different types of additional lights to the Scene, for example, *DirectionalLight*, where all light hits the objects in the same direction specified, and *PointLight*, in which light is cast in all directions from a specified point.

Both of these lights can be used to cast shadows on the 3D object, which could make the heights more obvious. However, casting shadows comes at the cost of performance, as calculating the shadows is costly. We were already seeing how poorly the 3D map was performing at the time, and with the shadows on top of that, it most certainly would worsen performance. Therefore, as it was not necessary as much as a cosmetic addition, the idea was scrapped, and the project would keep being lit by *AmbientLight* only.

7.5 User Interface

In chapter 5, you can see the changes and additions we have made to the solution's UI. The collapsible menu in the top left corner was created using a *<input type="checkbox">* HTML element for the icon, which makes it easy to create a collapsible menu using only CSS, avoiding the need to use JavaScript functions. With this method, we could create a menu * * element and change its properties depending on whether the checkbox has been checked or not, using the *:checked* pseudo-class on the checkbox, which activates when the checkbox with the specified ID has been checked. By default, the menu is hidden outside the screen by

setting its *margin-left* property to be -500px moving it in its entirety to the left as it has a width of just 200px. Its opacity is set to 0, this is to achieve a slight fade in effect when the menu enters the screen and fade out when leaving the screen. When the checkbox is checked, these properties are changed to *margin-left: 5px;* and *opacity: 1;*. This, combined with the *transition: 0.2s* element, makes the menu slide into the screen when the checkbox element is checked.

The menu icon itself is created as a *<label>* element for the checkbox, with the checkbox itself being hidden from view. A checkbox can be checked by clicking its label, which is useful for creating custom-looking menu icons since the original checkbox can be hidden entirely without removing its functionality. We could then use custom-made images as menu icons, one of a burger menu and one of an X, and again have these change depending on the checkbox *:checked* pseudo-class. The menu icon, when displaying a burger menu icon, is slightly transparent and gains opacity when displaying the X to make it clear that the menu is active.

Within the menu in the top left corner, we added a slider for adjusting the height exaggeration using a HTML *<input type="range">* tag, as mentioned in chapter 5. The minimum value was set to 0.1, and the maximum value was set to 10.0. When the nob of this slider is moved, the current value is sent to the function using the *oninput* property, which takes this value, and changes the scale.z value of all tiles to be this value. The user may reset the value back to 2.0 by clicking the reset button next to the slider.

The alternative zoom buttons were, as mentioned in chapter 5, created using HTML *<input type="button">* tags. The styling of these buttons was, as with the other UI elements mentioned so far, applied using CSS. To give the rounded look, the *border-radius* properties were set to 10px in the outer corners of each button. Making the buttons clearer when hovered over was achieved using the *:hover* pseudo-class, and making the buttons appear bluer when clicked was achieved using the *:active* pseudo-class. The zoom buttons, when clicked, will call a function defined in the wxs.three.js file to change the camera's position. This is accomplished by using the built-in function in the Three.js library for translating positions on the z-axis, *translateZ*, which performs this action using a value specified by the developer. The function is called in this project, given a value of -500 when zooming in and 500 when zooming out. To prevent the camera from zooming in too far, the camera's position on the z-axis is checked every time the user attempts to zoom in, and if it is less or equal to 500, the function does not proceed.

7.6 Loading Message and Error Message

With the larger amount of tiles being loaded on screen, it became less clear when the solution finished loading all tiles, and the user may be unsure if the program is still loading or, worse case, has become stuck. Geir Karsrud suggested displaying a message to the user to inform them if the program is still loading. We used a *LoadingManager* object from the Three.js library to achieve this. This object allows programmers to define behavior to execute while the desired textures; start loading, are loading, and once the loading process has ended.

We created a HTML * * containing the loading message, consisting of an image of a spinning circle, a common element of loading messages, and text. This element would not be shown while nothing is loading by setting its default CSS *display* property to being *hidden* and changing this property to *inline-block* to display it when loading is happening. The *LoadingManager* reports three values; The current URL from which the texture is being loaded, the number of currently loaded textures, and the total textures. To make the loading message more informative to the user, the number of loaded textures and the total amount of textures were included in the message. The message itself would then be: *Loading <Amount loaded>/<Total amount>*.

In addition to letting the user know the map has finished loading, during the development of the map, we found that there was no information displayed to the user if an error had occurred preventing the map from initializing. We implemented a simple error check function in its own separate file to help keep the program sufficiently informative for the user. This could be given two parameters, *obj* and *msg*; *obj* being an object or value of any kind, and *msg* either a message index or a specific error message. The function checks if *obj* is a *null* value, making it a tool for ensuring an error message is shown when an important value required by the program is *null*. If the given object or value is found to be *null*, an error message is displayed to the user, as shown in figure 7.1. In cases where a critical error may occur but is unrelated to a value being null, the function can be called by simply giving it *null* as a value in place of *obj*, which forces the function to display an error message. The *msg* can be either a string or a number. If a string is passed, it will be displayed as an error message. Alternatively, it is possible to predefine error messages in the error.js file where the function is located, and the function can be called with a number representing the message index in an array of messages.

7.7 Caching

As the program relies on a lot of external geographic data, which does not change frequently, caching is highly desirable. The initial plan was to implement a server-side service that would primarily store cached data but could theoretically do some

Figure 7.1: This banner will be displayed on the page when an error prevents the map from loading.

data-processing, as that is currently all client-side. However, this would require major rewrites and increase the complexity of the project, leading to possibly more bugs that are harder to track down. We ended up using Apache's built-in caching module, which, unfortunately we later learned the municipality will not use, nor will they use an equivalent in their own server-hosting software.

7.8 Code Metrics

This section will show some stats about the code itself, including how many new lines were written and how many were edited, how much time was spent on certain tasks, and some details about the issues and merge requests.

7.8.1 Code and languages

Approximately 446 new lines of code were written for this project, including comments. These consist of 78 lines of HTML, 142 lines of CSS, and 244 lines of JavaScript, as shown in 7.2a. In addition to these, 34 existing lines of code were edited. Of these, 7 were HTML, 8 were CSS, and 10 were JavaScript, as shown in 7.2b.

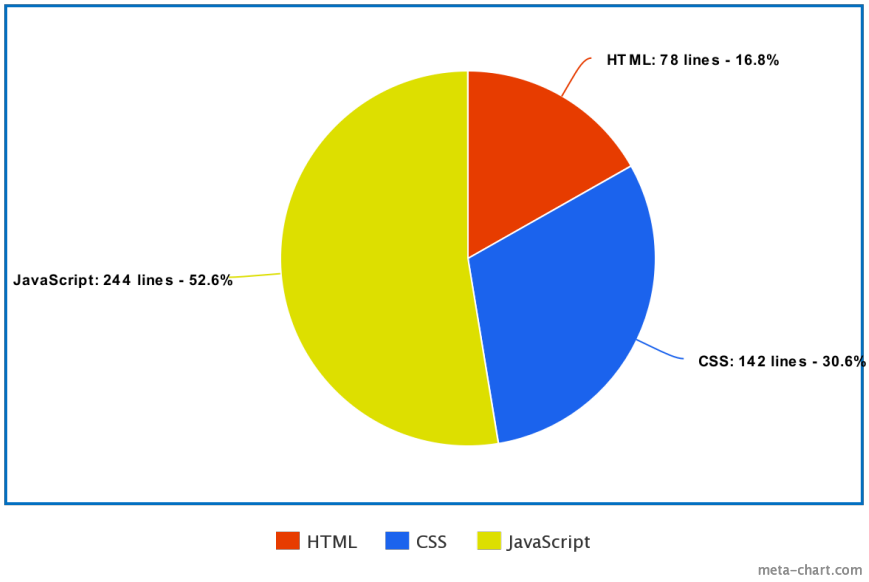
7.8.2 Issues and merge requests

In total, 15 issues were closed during this project. Some additional issues were made, which were optional if time allowed. 8 merge requests were made and merged with a total of 54 commits, and all were approved by the other group member.

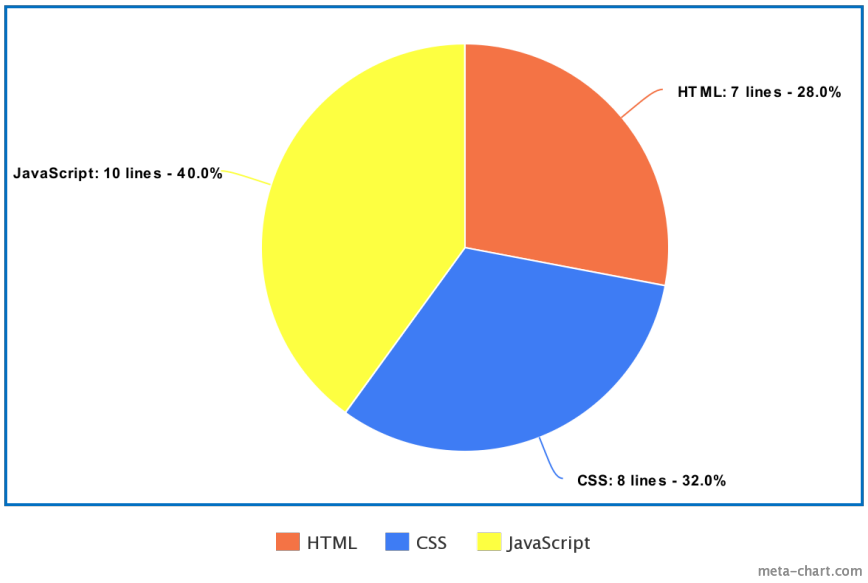
7.8.3 Time usage

The amount of time spent on each task has varied a lot due to their scope in general and unforeseen delayers such as bugs. In this subsection we will show some examples of how much time certain features took to implement.

As mentioned earlier in section 4.5.2, much time was spent in the beginning on studying the code and its libraries, including OpenLayers, Three.js, and the protocols used; WMS, WMTS, and WCS. During this project, Nora spent around



(a) Distribution of new code lines per language. Created with meta-chart.com.



(b) Distribution of edited code lines per language. Created with meta-chart.com.

Figure 7.2: Code written in this project, divided by language.

22.5 hours doing this to gain a better understanding of the project, Fredrik spent around 30 hours. Most of this was at the beginning of the project, with some of it also happening during the rest of the semester as new unknown aspects popped up. Also, during this time, some time was spent refreshing web development, such as JavaScript programming.

One of the features that took the most time to implement was the addition of the regulation plan overlay. This process took approximately 22 hours, including finding the correct endpoint, fetching the tiles and loading them onto the map, and creating the custom shaders. Loading the map from the endpoint itself took some time to figure out, with finding the right way the URL should be formatted. Once the plans could be fetched, the next step was finding the best way to show them on the map object, as explained in section 7.3.

Due to us being inexperienced with the Three.js library, finding a way to load more tiles took some time, as we spent some time researching how to detect the edges of the screen before finding out about the Frustum object, which luckily was quick to implement once discovered. Limiting the number of tiles to improve performance was a quick task, taking only two hours. Lighting was worked on for three hours before being scrapped.

Designing the UI and implementing it took approximately 16 hours. We spent time first implementing basic elements of the UI and making constant design adjustments over the course of these hours. Implementing the zoom buttons took around four and a half hours, and adding and styling the error codes took around seven and a half hours. During the UI design, around 1 hour was spent studying similar web solutions, which are shown in figure 5.5. Editing the GI2020.html file and 3Dklient.html file to work together took around eight and a half hours.

Chapter 8

Quality Assurance

This chapter will detail the methods we have used to ensure the quality of our project.

8.1 Testing

This section details the steps we have taken toward ensuring the quality of the code itself.

This project, mainly centered around graphical and other visual aspects, proved challenging to create proper automated tests for. To ensure core functionality still worked as it should during the development of the code, a workaround for the lack of automated tests was to create a comprehensive list of basic tasks, such as button clicks, scrolling the wheel on a computer mouse while using the map, etc., which are all tasks the user will perform while using the software.

In this list, the tasks listed are coupled with a description of the expected outcome next to an empty field where we may fill in the actual outcome if it should differ. When performing these tests, we could compare the actual result with what is expected and troubleshoot the code should the actual result differ. While not as fast as automated tests are, this way, we could quickly verify each core function in an orderly manner. The full list used for manual testing can be found in table 8.1.

For performance, we would utilize built-in FPS meters in our web browsers. While not perfect, it can indicate how well the program is performing and during what events the performance drops most.

8.2 User feedback

This section will detail the steps we performed to ensure the quality of the UX.

Task no.	Task	Expected outcome	Actual outcome
1	Open map	The map loads, with Gjøvik in center	Functional
2	Click [+]	The camera zooms in on map	Functional
3	Click [-]	The camera zooms out of map	Functional
4	Click [Reload]	The map reloads	Functional
5	Click on map with the right mouse button, drag towards the right	The map moves to the left	Functional
6	Click the map with the left mouse button, drag upwards	The map is tilted backwards	Functional
7	Move the mouse pointer to the lower right corner	The mini-map pops into the screen	Functional
8	Click the burger menu icon in the upper left corner	The menu enter the screen	Functional
9	Click the checkbox next to the text reading "Reguleringsplaner"	Regulation plans are shown on the 3D map	Functional
10	Drag the nob on the slider for "Høydeoverdriving" to the right	The hills on the map are taller	Functional
11	Drag the nob on the slider for "Opasitet" to the right	The regulation plans are more opaque	Functional
12	Click the spiral arrow next to the sliders	Nob and map property returns to default value	Functional

Table 8.1: The action list used to perform manual tests

8.2.1 During development

While designing the layout of the basic UI, we regularly seek feedback from external actors with no connection to the project. While the construction workers of the municipality are our primary target audience, seeking this feedback would help universalize the design of the layout, given it would be available to everyone. The feedback received here was generally not regarding the regulation plans, as this was found not to be a priority by most people outside of the construction profession.

8.2.2 Feedback from client

To ensure the project was satisfactory for our target audience, once we had implemented most of the features requested by the client, we sent the project over to Geir Karsrud, who put the files on their servers for hosting. However, we quickly received feedback we needed to look into before we could perform user tests. The first feedback we received was they had difficulties running the project, as the number of tiles led to severe performance issues. The then-current version of the project would load tiles infinitely when tilted, which led to the web browsers inevitably crashing. As there was no need to render tiles too far back to see, to combat the performance issues, the far-plane of the camera Frustum was moved closer to the screen, changing from a value of 5000000 down to 10000. This greatly improved the workload, as the number of tiles on-screen was reduced.

During development, we worked with the 3Dklient.html file in mind as the entry point. After we had sent the project, we were informed about the GI2020.html file being preferred as the entry point. This file would take in x-coordinates and y-coordinates through the URL, which would convert these values to BBOX coordinates for use in the wxs.three.html file. Based on this new information, we modified the 3Dklient.html file, which up until this point had worked with hard-coded coordinates as a starting point, and changed it to take in the values through the URL and work with these instead, with default values being used should no values be passed through the URL.

8.2.3 User Feedback

We sent a survey of basic tasks to perform on the map to our secondary contact person Geir, who sent it to construction workers within the municipality. This survey asked them to perform tasks similar to those in our manual testing list. they were then free to add comments they had regarding the map layout. They were also asked to name the type of device they had utilized and their choice of web browser, as these details could impact their experience. With the construction workers and general public as target audiences, we sent the survey to some actors outside the municipality employees. The answers we received from both

these audiences differed, giving valuable insight.

The answers given by the construction workers were mostly positive. Most of the tasks were reported as easy to perform, with few exceptions. One of the testers reported they could not rotate the map, stating they could not find buttons for doing so. In the map, the rotation is meant to be performed by dragging the map in certain directions. However, as dedicated buttons exist for zooming, we see how this could lead to confusion, as it could be assumed that buttons for rotating the map should exist. This could be fixed by implementing such buttons or adding instructions on controlling the map within the solution itself. Besides the rotation feedback, one tester reported turning on regulation plans as OK rather than easy. However, no further comments on that task were given. It could be assumed that the menu icon is slightly too hidden, and the toggle location for the regulation plans is not very obvious. Of all these testers, all performed the tasks on a computer.

The results from the testers from the general public were quite different, as most of these reported difficulty performing a larger number of tasks. Few of the testers could toggle the regulation plans, and some had difficulties moving the map. Most of these testers used a mobile device, which indicates flaws in the design across device types. We found that even though development browsers showed equal sizes to the buttons on desktop screens and in Responsive Design Mode, as well as our own SkyHiGh deployment, once the finished solution was deployed to the Municipality's website, the website was shown as a down-scaled version of the desktop version, making the buttons appear much smaller than planned. Usually, to combat this, a line is added to the HTML code reading *<meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0">*, which is commonly used in websites which are optimized for use on mobile devices, as it prevents the website from displaying as a down-scaled version of the desktop layout. This line was added to the 3Dklient.html file. However, it is not present in the GI2020.html, which could be the reason for this issue.

Users on mobile devices also found some difficulty moving the map. However, rather than having trouble rotating the map, they found it difficult to move it. Moving the map with TrackballControls is performed using three fingers, which may be a touch gesture not many casual mobile users are familiar with. A possible solution could be to add instructions on the movement of the map on mobile devices. Another comment we received was some users tried to zoom in on different parts of the map using touch gestures. However, the map only zoomed in on the center regardless of what part of the screen the gesture was performed. Although most of the testers could run and test the program, some users with older phones still found issues regarding performance and ran into crashes due to the heavy load.

Out of all the feedback we received, most of the problems testers had seemed related to map movement, as can be seen from the feedback received in figure 8.2. While 2D maps and their controls are common and more users have experience with this, when it comes to navigating maps in a 3D space, the movement shows to introduce challenges. While most movement gestures on a computer using a mouse are easier to figure out, mobile devices relying on the user finding out how to use touch gestures to perform certain movements proved a bigger challenge. While the main target audience is construction workers operating on desktop computers, it can be a good idea to adjust the mobile design to make the experience as universal as possible.

I felt like I was stuck at one point of the map, I could not move to another position to zoom in and out there.
When I turned the map to get it horizontally, it over-rotated the map.
I did not think it was possible to move the map, but only having ability to zoom as I was unable to move it.
It was difficult to move the map as I used a mobile device and expected to be able to move it with 1 or 2 fingers. Needed to use 3 fingers to move it instead of rotating. I still found it out relatively quick.
Could not move the map. I tried arrow keys, clicking and dragging with the mouse and all visible buttons on the side.
There was a square in the lower right corner which did not load, I do not know what +/- would to to the square. But it reloaded the map with new zoom.
After having tried it a bit more, I saw there was a picture in the corner which did not load. I tried clicking and dragging this, and the map moved. But it was quite uncontrollable.
Rotating the map felt very sensitive, anything more than a small swipe would rotate the map around much more than desired.
Could not find the buttons to rotate the map.

Table 8.2: Answers about what made certain actions more difficult.

The button for regulation plans is easy to find, but quite small on mobile. The menu itself should have been a bit bigger. Otherwise, everything was quite easy, clear and user friendly
Four buttons do the same thing? Scrolling with a mouse, +/- in the corner, the menu and +/- down in the lower left corner. The page takes a long time to load. After messing around with it for a little, all I got was a black map, even after a reload. But, I got back to the map eventually.
The map crashed a few times when trying to load too much, usually when the map was on its side after rotating it took far.

Table 8.3: General feedback from the testers.

Chapter 9

Deployment

9.1 SkyHigh

During development, we deployed our project to an OpenStack instance, on NTNU's SkyHiGh installation, letting us test its behavior when running from a server. And ensure the project ran on a non-development computer. We attempted to have the deployment automatically update every time the main branch of our Gitlab repo was updated. If it worked, GitLab would have simply sent a request to the server on its internal network, triggering a script to run the command "git pull". Unfortunately GitLab, due to security reasons, does not allow registering an internal IP for webhooks. Ultimately we decided to manually update the deployment, as updates were infrequent and it was quick to do manually.

9.2 Gjøvik Kommune's website

The deployment of the final solution is taken care of by the municipality, who are hosting it on their servers. The solution can be found via their Geoinnsyn portal here, as of May 2023.

Chapter 10

Evaluation

10.1 Development process

The development process was acceptable, although it could have been executed better in some areas. For example, we developed the product by choosing issues from our product backlog and doing them. What would have made this process better would be setting more concrete deadlines, like finishing the UI by a certain date, finishing the 3D map by a certain date, etc. Additionally, although we both kept it in mind throughout the project, we could have utilized the project plan better. We also probably should have prioritized differently initially, as the importance of the regulation plan overlay became apparent after our first meeting with Geir Karsrud.

Even though the development process could be better, we are happy to have implemented the highest-value requests we were given. The map has seen progress from the state it originally was, with a UI more suitable for being publically available, and regulation plans being implemented as an overlay.

10.2 Quality of what was achieved

The quality of the project did not turn out as good as we would have hoped, as the final project had issues with performance which should have been improved to better the overall experience. One of the features we wanted to implement, which was increasing fidelity when zooming in on the map, had to be scraped as we saw we could not implement it in time. The endpoint for fetching regulation plans seems to be rate limited, which results in black squares on the map, which should have been fixed. Lastly, a last-minute bug prevented the mini-map from loading as it should. We tried looking into it together with Geir Karsrud, but it remains unfixed. It remains somewhat functional in that clicking and dragging on the map does what it would normally do besides being invisible. This is something that could be looked into by future developers.

Besides these downsides, we are quite happy with many aspects of the finished product. The most important feature requested was the regulation plan overlay, and while the endpoint is limited, the plans are functional and visible on the map. The overall UI has also been improved, with the map fully loading on start and including alternative controls, which may be helpful for those who find zooming with a scroll wheel difficult, and an error message is displayed when the map fails to load. We have also increased security by hosting the dependencies locally rather than loading them from a remote source and updating the outdated Three.js dependency. However, we recommend switching to a package manager like Npm.

10.3 Project and Code

The source code we were given was, in our opinion, poorly documented. There were very few comments in the code itself, which led to us spending more time than anticipated finding out how the code worked. While reading the code, we decided to add some comments, mostly to keep track of the functionalities. Any new functions made by us have been given comments in an effort to make the development easier for future developers.

Although a function for this has not been made, the project has the ability to be given more optional layers. The texture for the regulation plan overlay is given to the shader as a uniform, meaning that it should be possible to load multiple overlays as textures and change the uniform to others.

10.4 Stakeholder

We had good communication with our stakeholder, although ideally, we would have wanted to meet with Geir Karsrud earlier in the process, as he provided valuable feedback and advice, having worked on the same project earlier. Other than that, we are happy with the collaboration. We implemented the core features requested, and our contact persons seemed pleased with the result.

10.5 User studies

Although we did receive useful feedback, the user studies of this project could have been conducted better. The number of results gathered from the main target audience, construction workers employed in the municipality, was quite lacking. Due to this, we sought testers from the secondary target audience, the general public. Here, luckily, we received a larger number of detailed responses. We would have liked to have sought out more feedback earlier, allowing us to work on the feedback given, and while we were able to work on the feedback given by Geir

Karsrud, the results of the user surveys were received late in the project. These results can be found in the appendix, which may be of use for future developers on this project.

10.6 Project plan

Looking back at the project plan, we find some deviations from how the semester was originally planned out.

We made plans to potentially utilize the Jest framework to conduct unit testing. However, this did not happen. We created manual tests for checking the 3D graphics of the project, but no tests of other functionality were implemented, as we should have. We considered using Flutter for UI development early in the project. While we spent some time looking into it, we ultimately did not use it for this project, as we did not see an elegant way of integrating the preexisting project with it.

Chapter 11

Conclusion

The project's development process could be improved. At the end, we succeeded in implementing what we wanted and what the specification requirements were. We prioritized the highest-value features and managed to deliver what the client wanted the most.

The product, although not performing as well as we would want, contains the features we wanted to implement by the project's end. The UI looks much cleaner and more user-friendly compared to the initial look.

As we had seen in the 3D Amsterdam example discussed in chapter 6, both the client and the group showed interest in the possibility of displaying 3D objects, such as buildings, as these would also be quite helpful for construction workers in getting a better overview for construction planning. This should be possible by utilizing additional endpoints provided by the Norwegian Mapping Authority.

The current map will get its location from the location it receives from Geoinnsyn. As is a feature in many other mapping services, one could add functionality for searching for specific addresses once inside the map. Such functionality does exist in the OpenLayers library, and it should then be possible to tie this together with the 3D map by searching for an address with OpenLayers and using the response to move the position of the camera to focus on that specific area on the 3D map. Alternatively, the Norwegian Mapping Authority provides a RESTful API for providing addresses, which could potentially be used to create a search function and tie it to the mini-map.

As discussed in section 8.2, we received a lot of user feedback from our surveys, which mostly concerned the movement of the map. The map uses TrackballControls. However, many camera control libraries have been made for Three.js, so there is a possibility that a library more suitable for this project exists, maybe even solving the current problems with navigating the map on mobile. If not, alternatively, it is possible to create a new camera control library or make edits to

an existing one. If none of these options is doable, adding instructions on moving the map to the users is also possible, be it within the program or as documentation.

The project has gotten essential features. But it still needs more development, this could make a good task for a future bachelor group. Overall, we have acquired a lot of knowledge about 3D graphics in web development and mapping services, specifically working with the Three.js library, and WCS and WMTS protocols, and we are happy with the final product.

Bibliography

[1] R. 1. GeoTIFF. 'Geotiff, revision 1.0.' (2020), [Online]. Available: https://www.loc.gov/preservation/digital/formats/fdd/fdd000279.shtml (visited on 10/05/2023).

[2] NASA. 'Hierarchical data format (hdf).' (2023), [Online]. Available: https://www.earthdata.nasa.gov/technology/hierarchical-data-format-hdf (visited on 10/05/2023).

[3] N. Snow and I. D. Center. 'What is netcdf?' (2023), [Online]. Available: https://nsidc.org/data/user-resources/help-center/what-netcdf (visited on 10/05/2023).

[4] L. Vårdal, J.-A. Overland and E. S. Zakariassen. 'Kommunene.' (2020), [Online]. Available: https://ndla.no/subject:1:470720f9-6b03-40cb-ab58-e3e130803578/topic:1:3d26f57e-a8c9-45e5-bc57-2d31df53f969/topic:1:27f37dfa-783e-4c3e-a52e-406ab0741b25/resource:ad4432c2-4f71-4007-a512-613dafa6f61f (visited on 10/05/2023).

[5] A. F. Sveen, C. Mielke, J. Pedersen and S. Iversen. 'Wxs.three.js.' (2016), [Online]. Available: https://github.com/jarped/wxs.threejs (visited on 09/05/2023).

[6] A. Enterprise. 'Wcs services.' (2022), [Online]. Available: https://enterprise.arcgis.com/en/server/latest/publish-services/linux/wcs-services.htm (visited on 09/05/2023).

[7] Peter Morville. 'User experience design.' (2004), [Online]. Available: http://semanticstudios.com/user_experience_design/ (visited on 28/03/2023).

[8] World Health Organization. 'Disability.' (2023), [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/disability-and-health (visited on 28/03/2023).

[9] G. LLC. 'Google maps.' (2023), [Online]. Available: https://www.google.com/maps/ (visited on 05/05/2023).

[10] A. Inc. 'Apple maps.' (2023), [Online]. Available: https://www.apple.com/maps/ (visited on 05/05/2023).

[11] N. AS. 'Kommunekart.' (2023), [Online]. Available: https://kommunekart.com/ (visited on 05/05/2023).

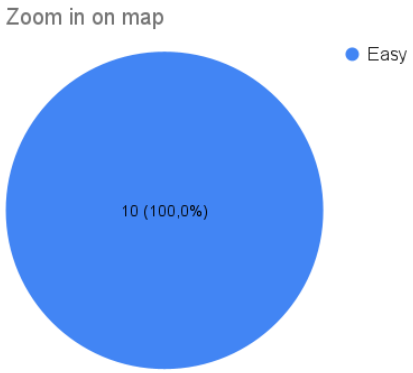
Appendix A

Additional Material

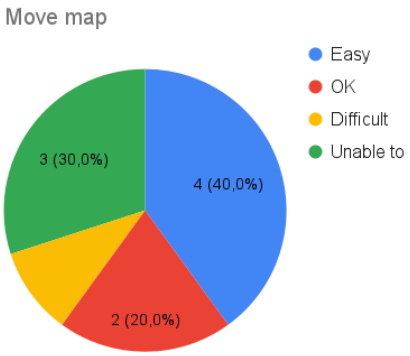
A.1 User Test Results

Følte jeg sto fast på et punkt på kartet, klarte ikke flytte meg til en annen posisjon for å zoome inn og ut der. Skjønte ihvertfall ikke
Når jeg snudde på kartet å ville ha det i horisontal så overroterte kartet
trodde ikke det var mulig å flytte rundt på kartet, men at det bare hadde zoom pga at jeg ikke fikk det ilt
Det var vanskelig å flytte på kartet da jeg brukte mobil fordi jeg forventet å kunne flytte det ved hjelp av 1-2 fingre på skjermen. Måtte ha 3 fingre for å flytte det i steden for å rotere. Fant likevel ut av det relativt raskt.
Fikk ikke til å flytte på kartet. Jeg prøvde piltaster, klikke og dra med musen og alle synlige knapper på siden.
Det var en firkant nede i høyre hjørne som ikke lastet inn, jeg vet ikke hva +/- skulle gjøre i forhold til forkanten. Men for meg reloadet de siden med ny zoom.
Etter å ha prøvd litt mer så jeg at det var et bilde i dette hjørnet som ikke lastet inn. Jeg prøvde å klikke og dra på dette og kartet flyttet seg. Men dette var veldig ukontrollerbart.
Rotating the map felt very sensitive, anything more than a small swipe would rotate the map around much more than desired.
Fant ikke symbolet/tegnet som viser at kartet kan roteres.

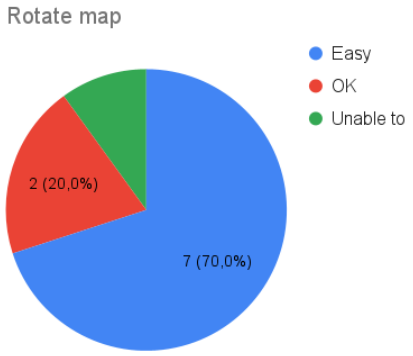
Table A.1: Answers about what made certain actions more difficult.



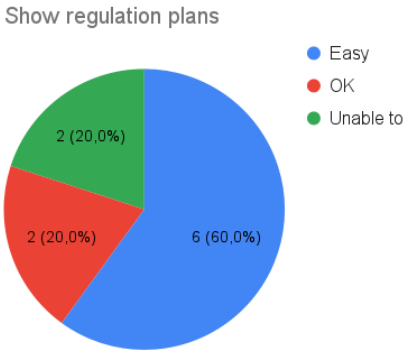
(a) Difficulty of zooming on the map.



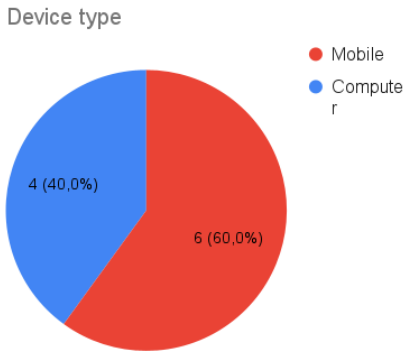
(b) Difficulty of moving the map.



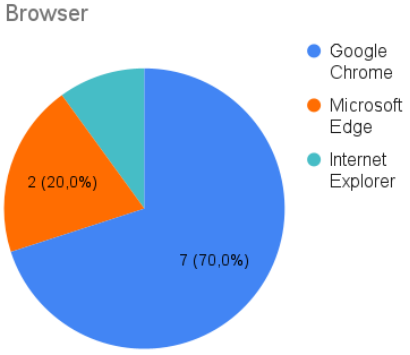
(c) Difficulty of rotating the map.



(d) Difficulty of showing regulation plans on the map.



(e) Type of device used.



(f) Web browser used.

Figure A.1: User testing results.

Vet ikke hva reguleringsplaner er :*-)
Knappen for reguleringsplaner er lett å finne, men blir svært liten på mobil. Selve menyen burde ha vært litt større. Ellers var alt veldig lett, oversiktlig og brukervennlig.
4 knapper som gjør det samme? scolling på mus, +/- i hjørnet, "menyen" og +/- nede i venstre hjørne Siden bruker veldig lang tid på å laste inn. Etter å ha tuklet litt med den får jeg kun svart kart selv etter reload. Men jeg kom til slutt tilbake til kartet.
The map crashed a few times when trying to load too much, usually when the map was on its side after rotating it took far.

Table A.2: General feedback from the testers.

Nora Hønnåshagen Hansen, Fredrik Lønnemo

Project plan

Innovative presentation of open data

Project plan
Supervisor: Mariusz Nowostawski
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU

Project Plan

1.1 Background

Gjøvik municipality gathers data from several sources through different development projects. Some of these experimental datasets are available as open data. They do not have the time and resources themselves to create good visual presentations of these data sets, and would like students to create educational and informative presentations of these datas for the relevant target audiences. An example that is relevant is water temperatures and air quality readings.

Within Gjøvik municipality, work on creating a 3D map of the city has been started, however it is not finished. This map would ideally give the user the ability to view area plans by clicking on a building, and potentially other information. They are interested in having this project further developed, by adding the mentioned functionality, as well as improve upon the user interface.

1.2 Project goals

- Result goals
We aim to develop a solution that will visualize the town of Gjøvik in 3d and make the viewing of data from the municipal government easier so municipal workers can make decisions with clearer/more digestible information.
If time allows we will add visualization of some open data to give information to citizens more easy
- Effect goals
Municipal workers spend less time on looking at maps and are able to reach decisions faster. And more easily see who is affected when they draw up area plans
- Learning goals
Web-development and deployment, integration of data from the "kommune", managing a larger project that spans months of work. Integrating UX feedback.

1.3 Framework

- The webservice should work on all web browsers with WebGL support, which ensures compatibility with ThreeJS as well. Some examples of browsers with WebGL support are the latest versions of Chrome, Firefox, Safari, and Edge

- The website should have independent design, meaning it can be viewed on both computer screens as well as mobile phone screens, without the need to develop two separate versions of the product

- The data used to make the 3D map will be fetched from the Norwegian Mapping Authority's APIs. The open-source library OpenLayers will be used to display maps, and ThreeJS is used for creating 3D graphics.

- Additional data to be added, for example area plans or air quality, will be fetched from the appropriate APIs hosting said data.

- The map should cover the city of Gjøvik only

Scope

2.1 Domain

When developing a product which will be available to the public, in order to reach as many of them as possible, it is important to create an independent design. To achieve this, user centered design becomes essential, as a product which has a well developed back end but an inconvenient front end will not be as successful. To avoid a lacking user interface, one must take multiple aspects of design into consideration such as user tests, accessibility, studying examples of good user centered design, and reiterations based on user feedback.

The primary target audience of the project is the internal audience of Gjøvik municipality, meaning employees who might use it for construction planning, among other things. These will be primarily using desktop computers, however with the growing prevalence of mobile devices, we will focus on keeping the design independent. This way, employees in the municipality can use the map on a computer, just as well as on a portable device such as a mobile phone or tablet. It will also ensure reaching a wider audience, such as the general public of the city.

- Frontend web design using HTML, CSS and JavaScript
- Graphics for web, using ThreeJS
- UI design with Flutter

2.2 Delimitation

The data used to create the map will be obtained from external services, including The Norwegian Mapping Authority's endpoints. Other endpoints will be used to collect the additional data to be displayed in the map, which may include Nilu, The Norwegian Public Roads Administration, and internal data from Gjøvik municipality. We will not be responsible for the accuracy of the data from the mentioned APIs.

2.3 Case description

This project will be using the existing in progress map solution, and develop it further. The main focus will be to achieve an intuitive user interface, and adding the features as wished by the client. The map's target audience is primarily the internal employees of the municipality, who will mostly be using the solution on desktops for purposes such as viewing area plans for construction work. The solution will be deployed as a service on the web. Although it is primarily targeted

at the employees of Gjøvik municipality, it will be open for anyone to view and use without the need for authentication.
The solution will use data from the Norwegian Mapping Authority to build the map itself. To add more information to be viewed in the map, we will use the APIs hosting the relevant data, these could include Nilu, Yr, and the Norwegian Public Roads Administration.

Project Organization

3.1 Responsibilities and Roles

Group roles

Client
Pål Godard is our client contact person representing Gjøvik Municipality. We will have meetings every other Thursday 12.00.
We will also with Geir Karsrud who developed the current 3d map solution and is part of it's intended audience

Supervisor
Mariusz Nowostawski is our designated supervisor. We will have weekly meetings in person on Thursdays 13.00, with the possibility of digital meetings if needed.

Group Leader
Fredrik Lønnemo is the group leader, which entails having the responsibility of making sure the project moves forward as it should, as well as solving internal conflicts.
Fredrik will also act as a developer

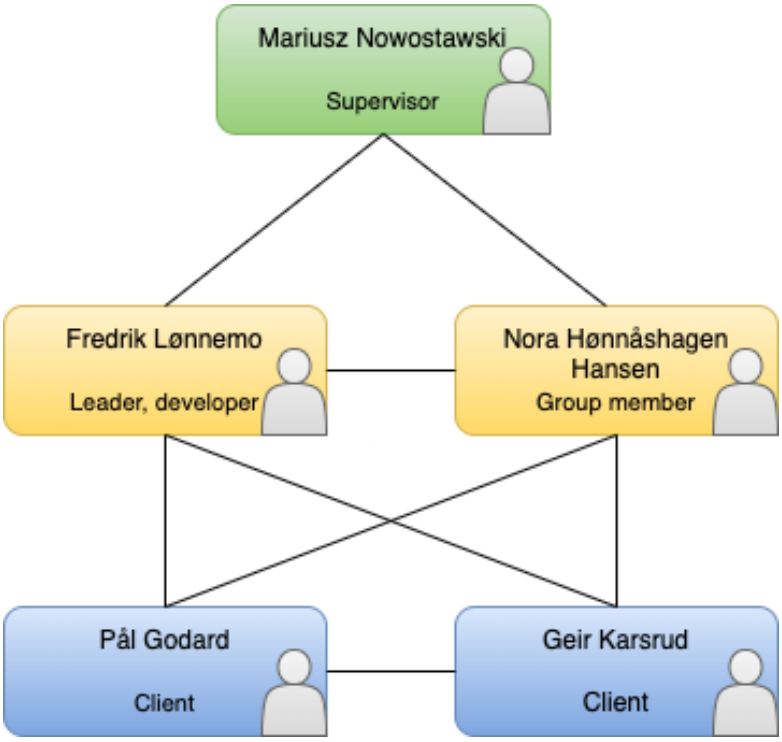
Developer
Nora Hønnåshagen Hansen is a developer on the team, with the shared responsibility of fulfilling tasks and documenting decisions made.

3.2 Routines and group rules

We will as a group hold weekly meetings, where we discuss what has been worked on, as well as what tasks should be worked on the following week.

Group rules

- Each group member should log approximately 30 hours a week.
- All hours spent on the project should be logged, with descriptions telling what these hours have been spent on



- We will have at least one group meeting each week, with the possibility of supplementing with more meetings if needed.

- If a group member is unable to attend a meeting, they are expected to notify the other group member of this

- Should a group member be unable to attend a meeting in person, they should alternatively attend the meeting digitally.

Planning, Follow up, and Reporting

4.1 Software Development Model

The model we have chosen to use for this project is the incremental sequential model. This entails starting with the base we have been provided, in which we will develop and add new additions piece by piece, with each piece being added to the working product. We have chosen this model as our task's scope is very open and has the ability to be expanded where desired. As we have a core component that needs to be functional (the 3d display of terrain and buildings) and then features that would be good to have, but not necessary, using the incremental sequential model is a good fit as it will let us discard, if necessary, or add on more increments, if possible.

4.2 Plan for Status Meetings and Decision-Making

We will have progress meetings on Thursdays with our client, our supervisor, and a group meeting, one after the other. After we have completed user testing, we will have a meeting where we discuss the results and figure out what changes should be made to the product.

Organization of Quality Assurance

5.1 Documentation, Standards, Configuration Management, Tools

5.1.1 Documentation

For this project we will be building upon pre-existing open-source code, and it is desired that our end result also be available to the public. Therefore it is important our project is well documented both with external documentation, and documentation in the form of comments in the code itself. The project should also be one that can be easily expanded further, making documentation important to ensure the source code is easy for the potential developers to understand.

We will be using Git for version control. The code should be committed frequently, with descriptive messages telling what changes have been made. The project repository will contain a comprehensive README with information on how to use the project.

5.1.2 Standards

All documentation and commenting will be done according to the relevant standards and best practices, in order to maintain tidy and easily readable code both for the group member and future developers working on the code.

5.1.3 Tools

Name	Type	Usage
OverLeaf	Collaborative cloud-based LaTex editor	Writing documents, project plan, final report, and other deliverables
TeamGantt	Gantt chart maker	Create the Gantt chart
SkyHigh	Cloud service	Hosting the solution on the internal NTNU network while in development
Discord	Communication platform	Hosting online group meetings and general communication
Flutter	Open-source UI software development kit	Creating the independent user interface for the solution
Google Drive	Cloud file storage	Sharing and keeping backups of documents
Visual Studio Code	Source code editor	Developing and editing code
Sublime Text	Text and source code editor	Developing and editing code
Firefox Developer Edition	Web browser with additional developer tools	Local testing
Gitlab	Version control system	Working together on the code and keeping track of changes

5.2 Testing (Plan for inspections and Testing)

As this project will rely heavily on requests made to different APIs, we will utilize testing methods such as mocking and stubbing. To do this we might utilize the JavaScript testing framework Jest, which comes with built-in mocking functions which can be overwritten to fit our needs. We aim for at least 50% test coverage, we do not feel confident in aiming higher as a lot of the code will likely be focused on graphical output which we find easier to test manually than programatically.

5.3 Risk analysis

Risks

Table 5.3.1: Risk Standard

	Severity					
Likelihood	Risk Matrix	Insignificant	Minor	Moderate	Major	Severe
	Almost certain	Medium	High	Very high	Very high	Very high
	Likely	Medium	High	High	Very high	Very high
	Possible	Low	Medium	High	High	Very high
	Unlikely	Low	Low	Medium	Medium	High
	Rare	Low	Low	Low	Low	Medium

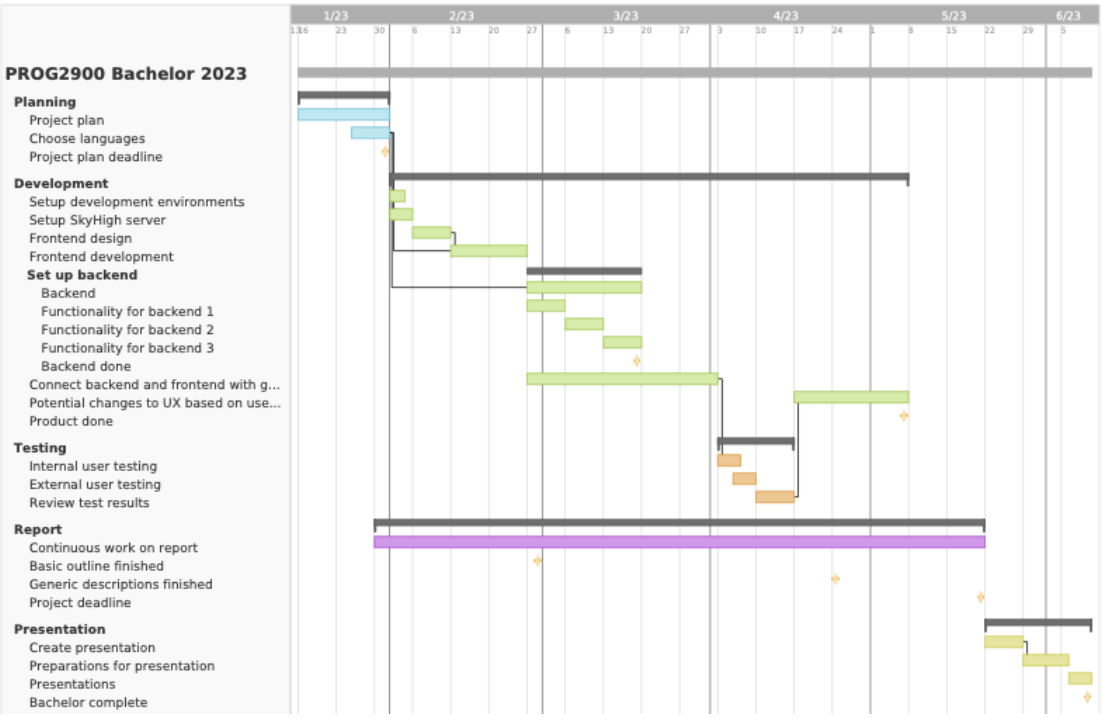
Number	Description	Likelihood/Consequence
1	Data endpoints become unavailable	Unlikely/Major
2	Group member is absent due to illness/other reasons	Possible/Moderate
3	Client is missing	Rare/Moderate
4	Product not finished by deadline	Possible/Major
5	Lose access to SkyHigh instance	Unlikely/Minor
6	User testing cannot be completed	Possible/Major
7	Source code lost	Rare/Major
8	The web solution works well on only some devices...	Possible/Major

6.1 Implementation Plan

Number	Mitigation plan
1	If the desired data cannot be fetched, we will seek out the providers to find ways the data can still be
2	All group members will be informed on what work has been done, to ensure they are able to pick up where the absent group member left off. As the scope is very open, adjustments can be made
3	We will keep regular contact with the client. If contact is lost, the supervisor will be asked to help to get in touch again. If necessary, we will ask for help from a student counselor.
4	Both members will work regularly on the project from the beginning. As the project is a very open one, the scope can be adjusted to ensure the goal is reachable
5	If access to the SkyHigh server is lost, we will apply for access again. Should our application be denied, we will find an alternative cloud service.
6	If user testing cannot be completed in any circumstance, alternatively we will have to look to other sources for inspiration, such as studying cases of good design.
7	Regular commits will be made to the git repo, ensuring that any data lost locally can be found remotely.
8	The product will be tested regularly on different device types and sizes to avoid dependent design.

6.1.1 Gantt chart

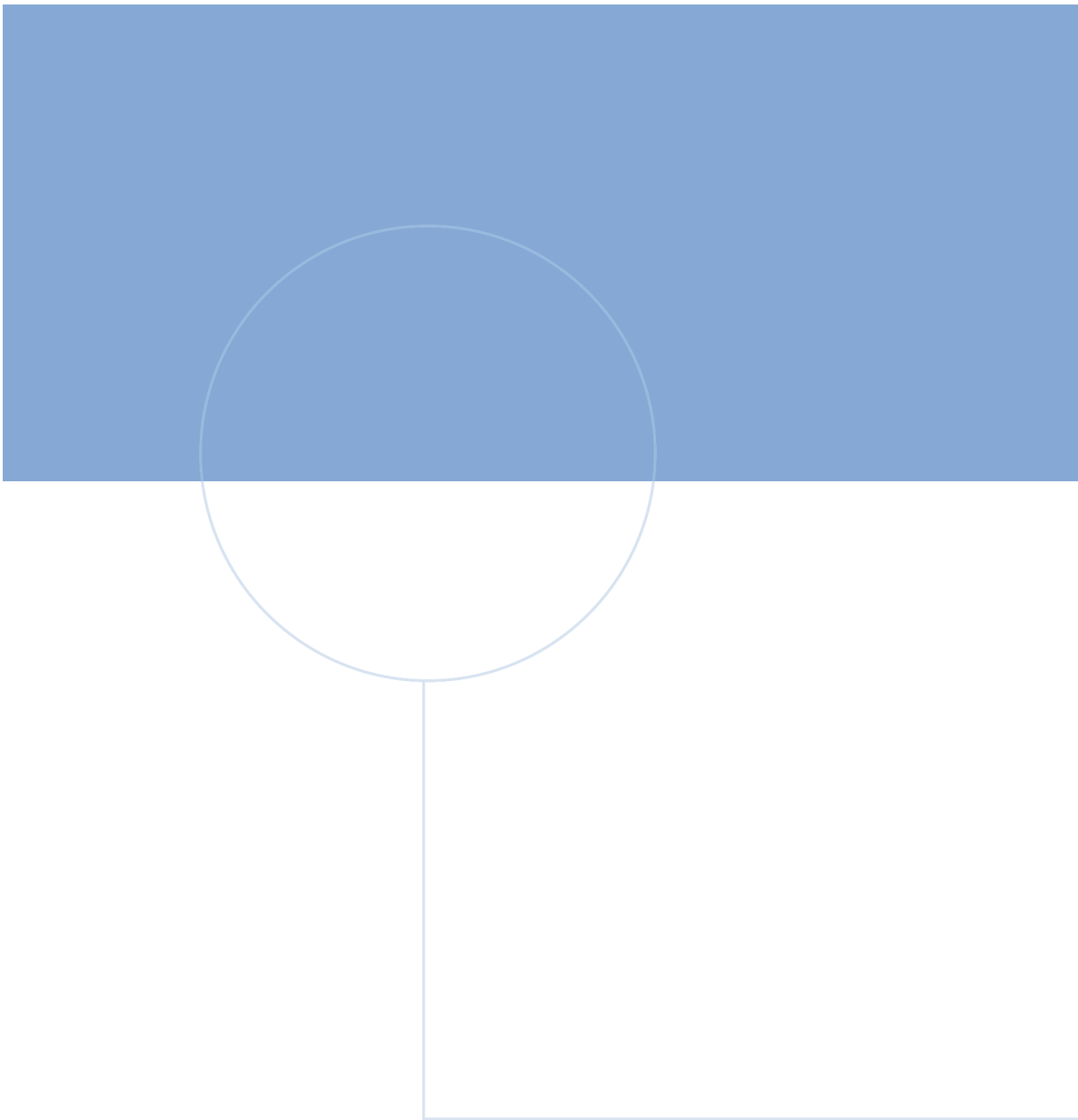
Gantt chart



6.1.2 Milestones

- 31. January: Project plan ready for delivery
- 31. January: Languages chosen
- 3. February: SkyHigh server and development environments set up
- 17. March: Backend done
- 5. May: Product done
- 31. March Product ready for user testing
- 22. May Final report and product deadline
- 6. - 8. June Presentations

References https://www.projectmanager.com/training/how-to-analyze-risks-project



NTNU

Norwegian University of
Science and Technology