Bachelor's project

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

Danielsen, Eirik Martin
Dyrkorn, Herman Andersen
Langlie, Anders Sundstedt
Magnussen, Andrea

Salamander Identification Application

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

Bachelor's project in Programming
Supervisor: Pedersen, Marius

NTNU
Kunnskap for en bedre verden

Danielsen, Eirik Martin
Dyrkorn, Herman Andersen
Langlie, Anders Sundstedt
Magnussen, Andrea

Salamander Identification Application

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

Bachelor's project in Programming
Supervisor: Pedersen, Marius
May 2021

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

NTNU
Kunnskap for en bedre verden

Sammendrag av Bacheloroppgaven

Tittel:	**Salamander Identification Application**
Oppgave nr.	23
Dato:	20.05.2021
Deltakere:	Eirik Martin Danielsen Herman Andersen Dyrkorn Anders Sundstedt Langlie Andrea Magnussen
Veileder:	Marius Pedersen
Oppdragsgiver:	Norsk Institutt for Naturforskning (NINA)
Kontaktperson:	Børre Dervo, borre.dervo@nina.no, 907 60 077
Nøkkelord:	Fullstack, Mobile, REST API, Datasyn, Kunstig Intelligens, Cross Platform
Antall sider:	103
Antall vedlegg:	12
Tilgjengelighet:	Åpen

Sammendrag: Salamanderartene storsalamander og småsalamander har hatt en dramatisk nedgang i populasjon i løpet av det siste århundret. I 2019 utlyste Norsk Institutt for Naturforskning (NINA) en oppgave som omhandlet å utvikle et system som kunne identifisere salamandere basert på det unike magemønsteret deres. Selv om det forrige prosjektet resulterte i en algoritme som klarte å identifisere salamandere, ble det ikke laget et grafisk brukergrensesnitt. Dette resulterte i at NINA ikke tok i bruk systemet. Derfor ønsket NINA denne gangen å få utviklet en applikasjon som benytter seg av algoritmen, slik at de kan bruke den i feltarbeid. Denne rapporten beskriver en fullstack applikasjonsutviklingsprosess, inkludert bildegjenkjenning og kunstig intelligens. Det endelige produktet består av en mobilapplikasjon, en REST API og en forbedret versjon av identifikasjonsalgoritmen.

Summary of Graduate Project

Title:	**Salamander Identification Application**
Project no.	23
Dato:	20.05.2021
Authors:	Eirik Martin Danielsen Herman Andersen Dyrkorn Anders Sundstedt Langlie Andrea Magnussen
Supervisor:	Marius Pedersen
Employer:	Norwegian Institute for Nature Research (NINA)
Contact Person:	Børre Dervo, borre.dervo@nina.no, 907 60 077
Keywords:	Full Stack, Mobile, REST API, Computer Vision, Artificial Intelligence, Cross Platform
Pages:	103
Attachments:	12
Availability:	Open

Abstract: During the last century, the salamander species northern crested newt and smooth newt, have dramatically declined in population. In 2019, the Norwegian Institute for Nature Research (NINA) issued a task to identify salamanders based on its unique abdominal pattern. Even though the previous project resulted in an algorithm that accomplished this, no graphical user interface was implemented for the researchers at NINA to interact with. This time, NINA requested an application that could incorporate this algorithm, so it could be used during field work. This thesis describes the process of a full stack application development, including image recognition and artificial intelligence. The finished system consists of a mobile application, a REST API and an improved version of the salamander identification algorithm.

Preface

We would like to thank everyone who has been involved in this bachelor's project. Thanks to our supervisor, Marius Pedersen, for actively supporting us and providing feedback throughout the entire project. We want to thank the Norwegian Institute for Nature Research for providing us with an interesting and unique task. We also want to thank our Product Owner, Børre Dervo, for his tremendous engagement and will to assist us in developing the best possible product. Thanks to Lars Erik Pedersen for quickly providing us with an OpenStack server, when we had to rethink our initial deployment plan.

Lastly we want to thank all friends and family who kept us motivated, helped us with testing the system, and provided feedback on our thesis.

Contents

Figures

Tables

Code Listings

Acronyms

AI	artificial intelligence. 3, 40, 43, 73, 103
API	Application Programming Interface. xviii, 25, 27, 31, 32, 42, 43, 57, 60
CPU	Central Processing Unit. 39, 78, 81
GDPR	General Data Protection Regulation. 4
GPU	Graphical Processing Unit. 39, 42, 66, 70, 73, 74, 81, 99
GUI	Graphical User Interface. 1, 5, 6, 20, 28, 46–48, 54, 64, 66
JWT	JSON Web Token. xvii, 19, 31, 39, 57, 61
LTS	Long term support. 78
NaN	Not a Number. 69, 73
NINA	Norwegian Institute for Nature Research. 1–5, 8, 9, 11, 18, 20–24, 33, 39, 41, 42, 46, 65, 66, 73, 76, 78, 79, 85, 86, 88, 97, 98, 100, 102, 103
NTNU	Norges teknisk-naturvitenskapelige universitet. 3, 5, 65, 78, 96
PIT	Passive Integrated Transponder. 2, 8, 100
SSL	Secure Socket Layer. 79
WIP	Work in Progress. 37

Glossary

Adobe XD A tool for creating digital prototypes. 28, 39, 46, 51, 52

Amazon Web Services Cloud service provided by Amazon. 3

Axios A javascript library that provides functionality to do HTTP requests. 27, 40, 42, 56, 57

Base64 In programming, Base64 is a group of binary-to-text encoding schemes that represent binary data in an ASCII string format by translating the data into a radix-64 representation. 31, 62

cloaca The opening for the amphibians that works as the outlet for feces, urine and reproduction. 41, 69, 73

Clockify A web-based time tracking tool for projects. 94

Cross-Site Request Forgery Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated [1]. 27

daily scrum 15 minute meeting held every day during a sprint. 21, 22, 38, 94

decorator A decorator pattern allows a user to add new functionality to an existing object without altering its structure. 61

DeepLabCut DeepLabCut is a python library that uses Tensorflow for pose estimation of animals and humans. 24, 32, 33, 39, 41, 42, 45, 64–66, 70, 71, 73, 93

Docker A tool that simplifies deployment of programs on cross platform at the cost of storage and performance. 79

DOS attack Denial of service (DOS) attack is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. 42, 63

dot product In mathematics, the dot product is an algebraic operation that takes two equal-length sequences of numbers, and returns a single number. 69

Eduroam A service used internationally by for example universities to exclude a network from the internet. 78

Fibonacci number In mathematics, the Fibonacci numbers form a sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. 36

Flask A python library that is used for web applications [2]. 24, 25, 30, 31, 39, 60, 62, 79, 93

Flask-Bcrypt A python library for hashing and salting passwords. 60

Flask-JWT-Extended An extension to handle JSON Web Token for authentication [3]. 60, 61

Flask-Limiter An extension that allows for easy implementation of rate limiters for endpoints [4]. 42, 63, 94

Flask-SQLAlchemy Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy. 31, 39, 94

GitLab A web-based DevOps lifecycle tool that provides a Git-repository manager providing wiki, issue-tracking and continuous integration and deployment pipeline features. 53, 94

ImageAI A library that focuses on computer vision using Tensorflow. 32, 33, 42, 72, 73

InceptionV3 A convolutional neural network model. 72

Kanban Agile software development method. 21, 94

Microsoft Azure Cloud service provided by Microsoft. 4

northern crested newt Red listed species of salamander in the subfamily Pleurodelinae. 1, 2, 41, 49, 65, 66, 72, 98, 100, 103

Openstack A platform on the internet where users can share questions, answers and discuss. 78

Overleaf A collaborative cloud-based LaTeX editor used for writing, editing and publishing scientific documents. 94

PIP PIP is a package manager for Python. 39

planning poker A gamified technique in Scrum (also called Scrum Poker) that helps a team estimate the required time it takes to complete the tasks in a sprint backlog. 36, 38, 94

Postman A tool for testing and documenting web APIs. 31

product backlog A structure of all the requirements for the end product. 22

product owner Term used in Scrum. Owner of the product being made. 22, 33, 34, 38, 42, 43, 46, 66, 78, 86, 88, 95, 98, 100, 102

RAW An uncompressed image format by Adobe. 1, 4

React React is a javascript framework for developing web applications. 24, 54, 58

React Native React Native is an open-source mobile application framework created by Facebook. It is used to develop applications for Android and iOS, among other things, by enabling developers to use React's framework along with native platform capabilities[5]. 24, 27, 28, 30, 53–56, 93

REST API An API that conforms to the constraints of REST architectural style. xi, 3, 20, 24–26, 30–32, 34, 37–42, 53, 57, 59–63, 70, 73, 74, 76, 79, 80, 93, 95, 97, 98, 103

Scrum Agile software development method. 21, 38, 92, 94, 102

scrum master Ensures the team follows agile values, principles and practices that the team agreed they would use. 22

semaphore A variable that controls the program/programs access to a set of memory. 42, 70

SHA-1 An algorithm that encrypts strings/character arrays. 78

smooth newt Species of salamander in the subfamily Pleurodelinae. 1, 2, 65, 66, 98, 100, 103

sprint backlog The Sprint Backlog is the set of Product Backlog items selected for the Sprint. 21, 36–38, 94

sprint burndown chart A diagram displaying for a set timestamps the amount of time left to complete the current sprint backlog. 36–38

sprint planning meeting A sprint planning meeting is an event that establishes the product development goal and plan for the upcoming sprint, based on the team's review of its product backlog. 21, 22, 36, 94

sprint retrospective meeting A meeting at the end of a sprint, where the purpose is to plan ways to increase quality, effectiveness and look back at the sprint.. 21, 22, 46

sprint review meeting A meeting at the end of a sprint where the purpose of the is to inspect the outcome of the Sprint and determine future adaptations. The Scrum Team presents the results of their work to key stakeholders and progress toward the Product Goal is discussed. 21, 22, 43

Tensorflow A library that focuses on machine learning, including deep learning. xvi, xvii, 33, 42, 66, 70, 73, 74

Trello Issue tracking software, styled as a kanban board. 22, 36, 37, 94

use case A use case is typically a list of actions defining the interactions between a role/actor and a system to achieve a goal. 11, 12, 29

VPN Virtual Private Network, is a service that allows a user to stay anonymous on the internet by sending their data to another computer that encrypts the origin of that data. 78

VRAM Memory on the GPU. 70

YoloV3 A convolutional neural network model user for object detection. 33, 69, 73

Chapter 1

Introduction

This bachelor's project was provided by the Norwegian Institute for Nature Research (NINA), and our contact person from the institute is Børre Dervo.

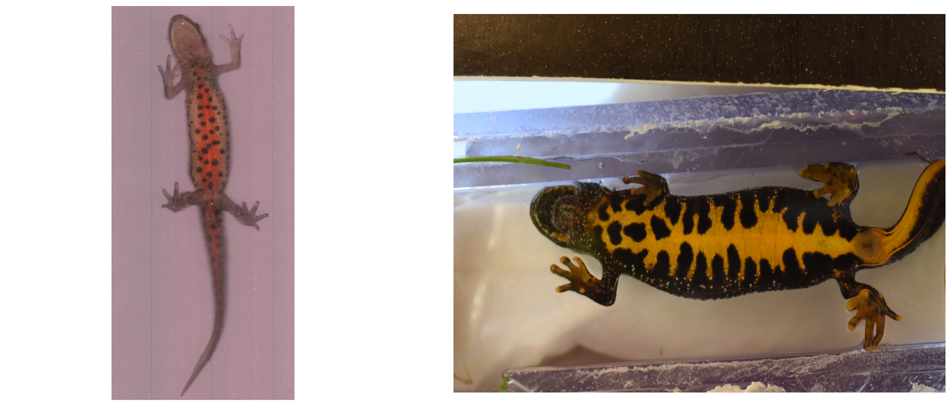
NINA is an independent foundation for nature research and research on the interaction between human society, natural resources, and biodiversity [6]. Back in 2019, NINA issued a bachelor's project where they requested a program that would be able to recognize a salamander based on a unique pattern located on its abdomen [7]. The previous bachelor's group was able to develop a system using machine learning and computer vision to match images of caught salamanders against a database of previously found salamanders. Unfortunately, they did not have enough time to make a proper Graphical User Interface (GUI) and instead resorted to using command line arguments. This resulted in it not being actively used by the scientists at NINA.

For this project, NINA want us to develop an easy to use GUI, using the existing system developed in the previous bachelor's project. They also wants us to make the system compatible with RAW images, and improve the algorithm overall. Additionally, NINA wants the ability to classify other traits eg. sex, weight and length. The full task description can be found in Appendix C.

1.1 Subject Area

One of the fields of research at NINA are salamanders. Salamanders are a group of amphibians, who thrive in marshlands, open woodlands, and cultural landscapes, with sufficient access to water and hiding places. In Norway, there are two different species of salamander; northern crested newt and smooth newt [8]. The northern crested newt is red listed [9], and the smooth newt while not red listed has declined in population [8]. It is therefore crucial to monitor changes in the population. By monitoring the number of individuals, one can get a good indicator of whether the active measures in place to aid in increasing the population are working.

Currently, NINA uses a Passive Integrated Transponder (PIT) placed under the skin of the salamander, to identify individuals that exist in Norway. This tagging method requires an invasive procedure, in addition to being expensive. By using computer vision to identify salamanders, this can be done in a non-invasive way, which is more humane and is less stressful to the salamanders. NINA is constantly looking for more ethical ways to conduct their research, making this a motivating factor for proposing this project. Figure 1.1a shows an image of a smooth newt and Figure 1.1b shows an image of a northern crested newt.



(a) A Smooth newt as seen from below. **(b)** A Northern crested newt as seen from below.

Figure 1.1: Images of the two salamander species.

1.2 Target Audience

The target audience for this thesis can be divided into two categories, one for the thesis and one for the end product.

1.2.1 Thesis

This thesis will first and foremost be interesting for anyone involved in the grading process, but also for others interested in the development process in regards to computer vision and mobile application development. The thesis requires that the reader has a basic understanding of programming, as well as some understanding of how software is developed.

1.2.2 Product

The target audience for the finished product is primarily the researchers at NINA. However, other researchers who are involved in the field of salamanders, may also find this product interesting.

1.3 Group Background

In this section, we will present our academic background and motivations for choosing this particular bachelor's project.

1.3.1 Academic Background

We are all bachelor's students studying programming at NTNU in Gjøvik [10]. The study program is divided into two slightly different paths. One focuses on application development, while the other focuses on game development. While many of the courses run for both fields of study, there are a few differences. Relevant experience for this thesis is artificial intelligence (AI) from the game development path, and web technologies from the application development path. Other relevant topics like software development, cloud technologies and mobile application development have been covered in both paths. Herman Dyrkorn, Andrea Magnussen and Anders Langlie followed the application path, whereas Eirik Danielsen chose game development.

1.3.2 Motivations

After reading several bachelor's proposals, the project description, as shown in Appendix C, was the one which gave us the best impression. It seemed interesting, exciting, and fun to work on. It also matched well with our line of study, as it involved full stack development of an application.

The project will consist of developing three different components, including a REST API, a mobile application, and an algorithm using computer vision with deep learning. This makes it quite easy for us to divide the different tasks between the group members based on interests and personal skills in the relevant subject areas. Another reason we desired this project was that it would allow us to learn about new exciting technologies and frameworks, like AI and cross platform mobile development.

We also got the impression that NINA was in a genuine need for a better solution than the one currently in use, which served as a significant motivating factor. Lastly, we got a great impression of our contact person at NINA, as he responded quickly to our emails. He also seemed motivated to follow up on the group throughout the project in order to get the best possible result.

1.4 Delimitations

In this thesis, we will only focus on making a system for NINA and not for an international user base. The software will be deployed on a server stationed locally at NINA, and will not be deployed on a cloud service e.g. Amazon Web Services or

Microsoft Azure. We will not implement measures to detect if an image actually contains a salamander or not, and we will primarily develop the system to handle images of good quality. By good quality, we do not necessarily mean good technical quality, but rather images where the salamander pattern is clearly visible and not obstructed.

We have decided that our application should require iOS 14.0 on an iOS device, and Android 8.0 Oreo on an android device. Lastly, due to our decision to use pictures taken with mobile cameras, we will only focus on using PNG and JPG image formats. Therefore, after discussions with our contact person at NINA, we will not optimize for the use of RAW images.

1.5 Constraints

The constraints are divided into three categories; time constraints, hardware- and software constraints, and legal constraints. These constraints will be taken into consideration when working on this project.

1.5.1 Time Constraints

- The product needs to be complete before 20th of May 2021.
- For the system to be deployed at NINA, we need a server at their location within April 2021, to do proper user testing.

1.5.2 Hardware and Software Constraints

- The software identifying salamanders and determining their sex is a demanding program, that requires a lot of processing power to run at sufficient speed. Therefore, it needs to run on a powerful computer.
- As the previous system was written in Python, we decided to continue using this language, as transferring the code to a different language would cost a considerable amount of time.
- Since the system will be utilized in fieldwork, the hardware needs to be lightweight and easily portable.
- For the software to function reliably, the image has to have a certain amount of sharpness and contrast. This might be limited by the quality of the camera taking the picture.

1.5.3 Legal Constraints

- Since users will store personal information, the application must conform to the General Data Protection Regulation (GDPR) [11].

1.6 Group Organization

Figure 1.2 shows an overview of our team structure and group responsibilities.

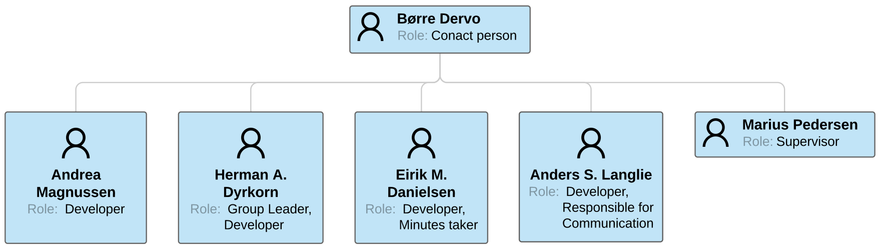


Figure 1.2: The map shows the different roles in the bachelor's project.

- **Contact person:** The contact person for this project is Børre Dervo. He represents NINA and provides us with information about their workflow, so we can fulfill the requirements in the project description. He will follow the project from start to finish and provide feedback regularly.
- **Supervisor:** Our supervisor is Marius Pedersen. He will provide guidance throughout the project with feedback and suggestions.
- **Developer:** Every group member will take part in developing the software. This includes taking part in all phases of the software development lifecycle [12].
- **Group Leader:** The group leader is responsible for making critical decisions and solving any potential conflicts or disagreements within the group.
- **Minutes Taker:** This role covers documenting all meetings with our supervisor and contact person.
- **Responsible for communication:** This role is responsible for email communication with the supervisor, contact person and other people involved in the project.

1.7 Thesis Structure

This thesis includes a list of acronyms and a glossary, which can be found above the introduction. We were provided with a LaTeX template from NTNU [13], made by Ivar Farup, and followed this as it is a baseline for computer science theses. In addition to the chapters the template suggested, we added a couple of additional ones: Chapter 3 Development Plan and Chapter 6 Graphical User Interface. Below is a listing of all chapters featured in this thesis, in chronological order.

1. **Introduction**: Contains a description of the task, as well as an introduction to the group.
2. **Requirements**: Covers the requirement specifications for the system.
3. **Development Plan**: Covers early choices we made before we started developing, in regards to both software development model and technologies.
4. **Technical design**: Presents an overall view of the final technical design.

5. **Development Process**: Covers the development process and tools used, including a short summary of all the sprints in a chronological order.
6. **Graphical User Interface**: Covers the GUI in the mobile application.
7. **Implementation**: Goes into detail about the implementation of the system's features.
8. **Deployment**: Covers how the mobile application was built and how the back end is deployed.
9. **Testing**: Covers testing of the system's performance and usability.
10. **Discussion**: Contains reflections on both the process and the final product.
11. **Conclusion**: Summarizes the final product's result, future work, and a few final words.

Chapter 2

Requirements

To make sure the product will satisfy the end users, it is important to spend a considerable amount of time on requirement specifications. Doing this will also help detecting possible challenges that may appear later on in the project.

2.1 Project Goals

The project goals are categorized into result goals and effect goals.

2.1.1 Result Goals:

- A working web-server that incorporates an improved version of the previous matching algorithm and will communicate with mobile phones used in the field.
- A cross-platform mobile application that will work as a client to the server.
- The system should have a better user experience than the existing system.
- The system should be able to classify the sex of the salamander with 95% accuracy.
- The system should be able to classify the salamander's species with 95% accuracy.
- Reduce the search time for matching salamanders by 50% on average by dividing the search by sex.
- Reduce the search time for matching salamanders by another 50% on average by dividing the search by species.
- Reduce the search time by 50-99% by dividing the search by geographical location.
- The server must support a capacity of at least five researchers concurrently.

The result goals will be revisited in Chapter 10 Discussion, to discuss to what degree these goals were met.

2.1.2 Effect Goals:

- Qualitative Goals:
  - The salamanders will be more carefully handled with the new system.
  - Increase the number of researchers tracking salamanders in the field, by eliminating the need for a special license for PIT.
  - The system should make it more efficient for researchers at NINA to conduct their research.
- Quantitative Goals:
  - Reduce the number of work hours needed to identify salamanders for the researchers at NINA by 75%.
  - The new system should replace the need for PIT-tagging within two years.
  - The system will reduce NINA's cost toward salamander research by 30% after five years.

These goals can only be measured over time, and will not be revisited in this thesis. However, they still might be useful for NINA to monitor over time.

2.2 PACT-Analysis

To define requirements in regards to user experience, we will conduct a PACT-analysis. A PACT (People, Activities, Contexts, and Technologies) analysis is a useful framework for thinking about human centered design [14].

2.2.1 People

Demography

The application is aimed towards researchers and others who might intern at NINA, who usually are in the range of 20-70 years old. This is quite a large span, which makes it important that the system is developed for a wide range of age groups. However, this is not an application that will be used by the general public, but rather a small group of researchers.

Computer Literacy

Most of the researchers at NINA use their mobile phones daily. To ensure that the researchers will be able to use the application without problems, the application should not require more technical skills beyond basic operations such as taking pictures and writing text into input fields.

Language

The system mainly targets Norwegian researchers at NINA, who also knows English. There may also be interns at NINA who only knows English. That is why we have decided that the default language for the application will be English. While we might add support for Norwegian at a later time, this will not be a priority.

Physical Abilities and Disabilities

Some of the researchers at NINA might have weak eyesight. This makes it necessary for the physical appearance of the application, such as the text size and buttons to be of large enough scale to accommodate these users.

2.2.2 Activities

Regular Operations

Functionality in the application should be designed intuitively, and operations should be logically performed. The application will have one primary goal, which is to aid the researchers in their fieldwork in regards to identifying caught salamanders. We will limit unnecessary features to avoid distracting from the main task.

Usage

The application will be designed to aid each researcher individually, and there will be no cooperation in the application other than the shared database with salamander images and locations.

2.2.3 Context

Physical Environments

The application will primarily be used outside, so it is important that the user can see the screen even when it is sunny. Since the researchers will use the application in their fieldwork, it is important to create an application that can be used on a portable device.

Social Environments

The application will be used in a professional setting, either individually or with other researchers.

2.2.4 Technologies

Network

Since the application will communicate with a web-server located elsewhere, the user has to have connection to the internet at all times when using the system.

View

As the users will interact with the system on a smart phone, the design will have to follow mobile design principles [15]. The application will also not support landscape mode, as this will not enhance the functionality.

Data Transmission

It will be necessary for the application to have access to the mobile phone's camera and the local camera storage on the device. The user must also accept that the application uses their position if they want to get access to location services.

2.3 Use Case

To capture core functionalities of our system and visualize the interactions of different actors, we have decided to utilize a use case diagram, with corresponding high level use cases and low level use cases. This makes it easier to keep the requirements of the system in mind, while in development. It will also show us the bigger picture in regards to important decisions we have to make in the requirement phase [16].

2.3.1 Use Case Diagram

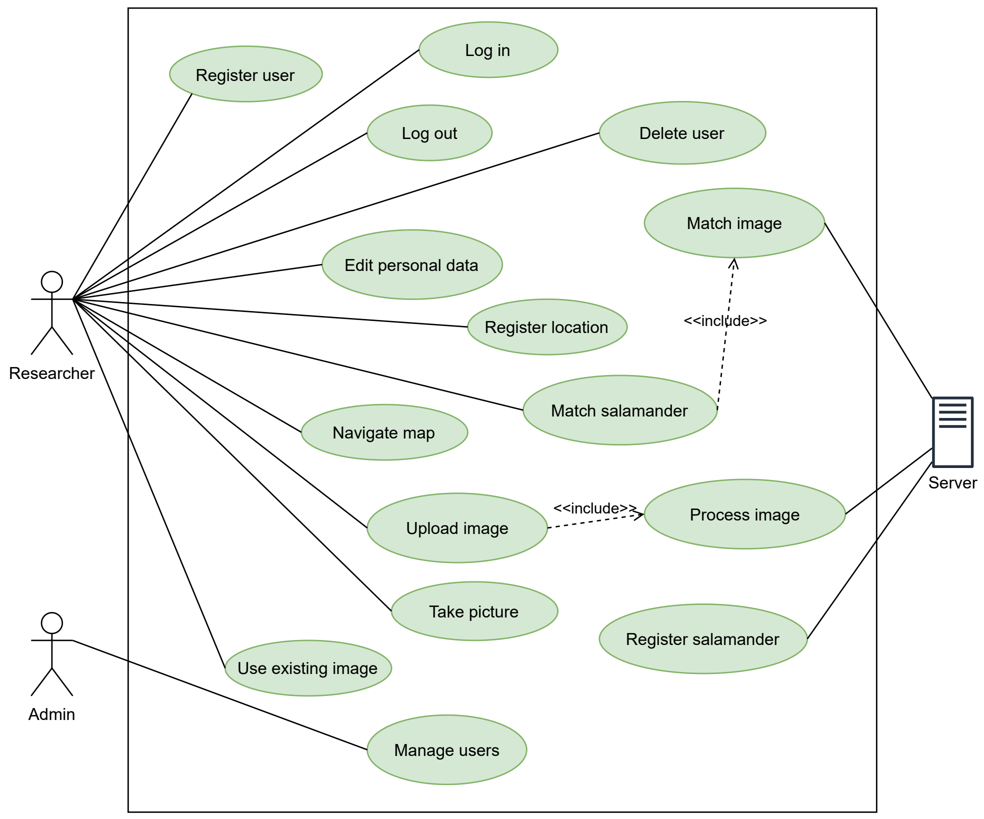


Figure 2.1: Use case diagram of the system.

Figure 2.1 shows a use case diagram of the system. To avoid too much clutter in the diagram, the admin actor only has a single connection to a use case, which is unique for the admin role. However, the admin will also be able to do everything a researcher can do, except for registering and deleting their user. Finally, all use cases, except for "Take picture", "Use existing image", and "Navigate map", involves server processing.

2.3.2 Actors

- **Researcher:** Works in the field, and will use the application for the individual detection of salamanders.
- **Admin:** Administrates the application by managing all users that request access to the server.
- **Server:** Will host all back end code and process requests from the mobile application. The server is planned to be located at NINA's headquarters in Trondheim.

2.3.3 High Level Use Case

The standard pre-conditions for all use cases is that the actor has to have an internet connection. In addition, in all use cases, except for register user (Table 2.1) and login (Table 2.2), the actor needs to be logged in.

Table 2.1: Use case for registering a user.

Use Case:	Register User
Actor(s):	Researcher.
Goal:	Create user with name, email and password.
Pre-condition:	The name, email and password is valid. The email must not exist in the database already.
Description:	The actor fills in name, email, password and a confirm password and clicks on the register button. Newly created users will not have access before they are approved by an admin.

Table 2.2: Use case for logging into the application.

Use Case:	Login
Actor(s):	Admin and Researcher.
Goal:	Login to the mobile application to gain access.
Pre-conditon:	Have an account that has the correct permissions.
Description:	The actor uses their email and password to login to the mobile application. The server checks their credentials and either accepts or declines their login request.

Table 2.3: Use case for logging out of the application.

Use Case:	Logout
Actor(s):	Admin and Researcher.
Goal:	Logout of the mobile application.
Pre-conditon:	Standard pre-conditions.
Description:	The actor navigates to the profile page and clicks on the logout button.

Table 2.4: Use case for editing personal data.

Use Case:	Edit Personal Data
Actor(s):	Admin and Researcher.
Goal:	Edit password, email or name.
Pre-condition:	The name, email, and password is valid. Email does not exist in the database already. The actor must be re-authenticated to edit password and email.
Description:	The actor navigates to the profile page. Here they can choose to edit their password, email or name.

Table 2.5: Use case for managing users.

Use Case:	Manage Users
Actor(s):	Admin.
Goal:	Administrate users.
Pre-condition:	Admin privileges are required and the actor must be re-authenticated.
Description:	The admin will have the ability to approve or decline new users. They will also be able to either make users admin, remove their admin access, revoke approval, or completely delete their account from the application.

Table 2.6: Use case for deleting a user.

Use Case:	Delete User
Actor(s):	Researcher.
Goal:	Delete own user from system.
Pre-condition:	The actor must be re-authenticated.
Description:	The actor navigates to the profile page where they are able to delete their user. This decision must be confirmed twice by the user.

Table 2.7: Use case for navigating the map.

Use Case:	Navigate Map
Actor(s):	Admin and Researcher.
Goal:	Navigate the map to see all registered locations.
Pre-condition:	Standard pre-conditions.
Description:	The user navigates the map by zooming or dragging their finger across it. The map will display all registered locations and the user can click on a location to display the name.

Table 2.8: Use case for registering locations.

Use Case:	Register Location
Actor(s):	Researcher and Admin.
Goal:	Add a new salamander location to the system.
Pre-condition:	Standard pre-conditions.
Description:	The actor creates a new location by registering position, create a name and set a radius.

Table 2.9: Use case for taking pictures.

Use Case:	Take Picture
Actor(s):	Researcher and Admin.
Goal:	Take picture of caught salamander with the mobile camera.
Pre-condition:	System must have access to mobile camera.
Description:	The researcher uses the camera function in the application to take a picture of the abdomen of a salamander.

Table 2.10: Use case for using existing image.

Use Case:	Use Existing Image
Actor(s):	Researcher and Admin.
Goal:	Choose an image from mobile storage.
Pre-condition:	System must have access to mobile storage.
Description:	The actor can select an image from their mobile storage to be used in the application.

Table 2.11: Use case for uploading an image.

Use Case:	Upload image
Actor(s):	Researcher and Admin.
Goal:	Receive a processed image from the server.
Pre-condition:	The actor either has to take a picture or use an existing one.
Description:	The actor uploads an image to the server, which processes it. After processing, the actor receives a response containing either the processed image or an error message.

Table 2.12: Use case for processing image.

Use Case:	Process image
Actor(s):	Server.
Goal:	The server computes salamander data and returns it.
Pre-condition:	The server must have received an image.
Description:	The server computes the salamanders sex and species. It also isolates the abdominal pattern of the salamander and returns the data to the client.

Table 2.13: Use case for registration of salamanders.

Use Case:	Register Salamander
Actor(s):	Server.
Goal:	Register a salamander into the system.
Pre-condition:	Salamander does not exist in the system.
Description:	The server registers a new salamander in the database with a unique ID, location, sex, species, date, and which user uploaded the image.

2.3.4 Low-level Use Case

We have made low level use cases for the use case Match image and Match salamander, which can be seen in Table 2.14 and Table 2.15. Our reason for choosing these specific requirements is because they are the most vital and substantial parts of the system. We wanted to cover the matching functionality of the system in both the client and the server.

Table 2.14: Low-level use case of matching image.

Use Case:	Match image
Actor(s):	Server.
Goal:	Match caught salamander against registered salamanders.
Description:	The server matches the incoming image against preprocessed images in the database.
Preconditions:	Server has received a request with an image and salamander data.
Post-conditions:	Sends a response back to client.
Special Requirements:	The request needs to come from an authenticated user with the correct access privileges.
Success Scenario:	1. The server receives a request from a user. 2. The server checks the access rights of the user. 3. The server validates the input data. 4. The server processes the image. 5. The server loads a subset of preprocessed images from the database, based on location, sex and species. 6. The server brute force matches the processed image against the subset. 7. The server returns a response back to the client.
Alternative Scenarios:	5-6. If the salamander is a juvenile, it will only be stored in the database and not matched against other salamanders. 7a. The server finds a match, and returns a confirmation message to the user. 7b. The server does not find a match, and register a new salamander in the database.
Fail Scenarios:	2. The user does not have the right privileges, and the server will abort the request. 3. The data sent to the server lacks important information, and the server will return an error response. 2-7. Internal server error while trying to process the request.

Table 2.15: Low-level use case of match salamander.

Use Case:	Match Salamander
Actor(s):	Researcher and Admin.
Goal:	To see if the salamander has been captured before.
Description:	The actor can match a salamander against the existing salamanders in a specific capture area, and get a response telling them if it was a match or not.
Preconditions:	The actor has to be logged in to the system. They also have to have cell coverage or WiFi on their mobile device.
Post-conditions:	The actor will always get feedback, either if it is a match, not match or server timeout.
Special Requirements:	There has to be a registered location in the system, for an actor to be able to match a salamander.
Success Scenario:	1. The actor navigates to the camera view. 2. The actor takes a picture of the salamander. 3. The actor confirms the image and fills in the required salamander information. This includes sex, species and location. 4. The actor clicks on the match salamander button, which sends a request to the server. 5. The server processes the request. 6. The actor receives feedback from the server.
Alternative Scenarios:	2. The actor can upload an existing image instead. 6a. The server finds a match, and sends a confirmation to the actor. 6b. The server does not find a match, and sends a message to the actor with the new entry information.
Fail Scenarios:	3a. The actor fills in wrong information about the salamander, and it will not match against the correct dataset. 3b. The image is not of sufficient quality, which will make the matching difficult. 6. The request times out, and an appropriate message will be displayed to the actor.

2.4 Performance

The systems performance relies on the cell coverage the user has when using the application in the field. By using Equation (2.1) [17], we can calculate the time it takes to upload an image.

- **Mb**: Megabit.
- **Mbps**: Megabit per second.

- **MB**: Megabyte.

$$Time(s) = \frac{FileSize(Mb)}{UploadSpeed(Mbps)} \tag{2.1}$$

According to Speedtest Global Index [18], the upload speed in Norway for cell coverage is 18.35 Mbps.

If we have an image of e.g 2 MB, which is equal to 16 Mb, we can calculate different scenarios based on the upload speed:

- **4G average (18.35 Mbps):** According to Equation (2.1) it will take 0.871 seconds to upload the image.
- **4G fast (50 Mbps):** According to Equation (2.1) it will take 0.320 seconds to upload the image.
- **Edge (0.2 Mbps):** An edge connection will only be able to upload 0.2 Mbps [19]. According to Equation (2.1) it will take 80 seconds to upload the image.

By looking at the calculations above, it is clear that the system will not perform sufficiently if there is bad cellular coverage. To ensure that the processing never takes too much time, there will be implemented a system timeout. If the system uses more than 50 seconds, the request will be aborted. However, our contact person at NINA said that most of their salamander locations have good cellular coverage. In addition, we will make it possible to upload images from the camera roll at a later point in time, if the cellular coverage is too slow at the exact location.

After the server receives an image from the client, the performance relies on how fast the matching algorithm is, which includes the pre-processing of the image and the brute force matching. The speed of the matching will slowly decrease as the database grows. However, by dividing the database into species, sex and location, the decrease in speed will happen at slower rate. This will also increase the accuracy of the matching as it only needs to match against the correct species, sex and location of the salamander.

2.5 Security

Security is an important part of the whole development process. Therefore, principal security measures will be kept in mind and followed when designing the system.

2.5.1 Functional Security Requirements

- When a new user wants to register, they have to:
    - Register a valid username.

    - Register a valid email.
    - Use a password with at least nine characters, letters, and at least one number.
- If the user tries to log in with an incorrect password, they will get a response like "username or password is incorrect".
- To use the system, a user has to be approved by an admin.
- The user will not be allowed to send a request to the server before they have provided all necessary data on the client side.
- Critical features will require re-authentication.

2.5.2 Non-functional Security Requirements

- The server should encrypt passwords before being stored in the database.
- Each user should have a role, which will decide what kind of functionality the user will have available.
- A JSON Web Token should be assigned to each user session in order to have safe user persistence across multiple requests without needing manual re-authentication.
- The client should validate all input from the user.
- The server should validate all input from the client.
- The server should have a rate limit to how many requests it can receive per minute.

While there is no dedicated security chapter in this thesis, security measures will be covered further in Chapter 4, Chapter 6, Chapter 7, and Chapter 8.

Chapter 3

Development Plan

This chapter will cover early decisions we made in regards to development. This includes which software development model we decided to use, as well as which tools we used throughout the project.

3.1 Software Development Model

In this section the characteristics of the project will be discussed, as well as our choice of development model and its use.

3.1.1 Characteristics of the Project

This project is a continuation of a previous bachelor's project from 2019 [7]. We decided to incorporate the code from the previous project, as well as improving and adding new functionality. The system will be developed independently as three components; an image recognition algorithm, a REST API, and a mobile application. Later in the development, the algorithm and the REST API will be merged together as one component.

Considering that the project deadline is the 20th of May, including thesis writing, the system development needs to be adapted to the given time frame. The requirements from NINA are also quite ambiguous, especially with regards to design. This means that we will have to refine the requirements during the development process.

One of the most important tasks in this project is to develop a Graphical User Interface (GUI), so that the algorithm can be used by the researchers at NINA. They are not software developers and some of the researchers may lack technical experience in computer science. Therefore, we need to establish a close collaboration between NINA and us, especially in regards to design and functionality.

The members of our group have limited experience when it comes to a project of this scale. This makes it difficult to plan the entire project before it starts and also hard to estimate the time needed to implement each task. However, if the final system lacks minor features, it can still be useful for the researchers at NINA.

All of these characteristics will need to be taken into consideration when choosing the software development model.

3.1.2 Software Development Model

Based on the characteristics in Section 3.1.1, we have concluded that we need an agile software development model [20]. Relevant software development models includes Kanban and Scrum. Both models have advantages and disadvantages in regards to a bachelor's project setting.

Kanban is an agile software development model where you visualize the whole project by dividing the project work into smaller tasks and placing them on a Kanban board [21]. The developers are free to choose tasks from the board and there are no specific roles in the developer team. This leads to Kanban being highly flexible [22]. Such flexibility could be an advantage for our group, considering the loose boundaries in the project description. On the other hand, this flexibility can lead to hard or boring tasks getting ignored, if none of the group members takes the responsibility to do it [23]. Besides, it might be more difficult for the group members to follow up on each other's contribution and workflow, if there are no set deadlines and planned meetings [24].

Scrum is an agile software development model based on incremental development, where the entire project is divided into sprints with set deadlines [25]. Each sprint will usually have a set length between two to four weeks. Roles, meetings, and other tools, aids this methodology in achieving structure and managing workload. Having proper structure and good routines could be especially helpful for us in our team, as we are quite inexperienced with projects of this scale. An important factor for us is that Scrum is suitable for smaller teams [25]. By utilizing daily scrums, all members of the group will get a good overview of what everyone is doing. On the other hand, we are currently in a pandemic, which means that we are not able to have physical meetings with our supervisor and contact person. This can hinder the full potential of having sprint planning meetings, sprint review meetings, and sprint retrospective meetings. Although fixed meetings contribute to maintaining structure, there may be occasions where they are not necessary. As a consequence, time may be wasted.

After looking at both of these models, we have concluded that Scrum is a fitting software development model for this project, as it can help our group maintain proper structure. We also want to incorporate some features from Kanban, like the Kanban board, for keeping track of the project backlog and sprint backlog. If necessary, we will bring in pair programming from eXtreme Programming [26]. Lastly, we concluded that the inability to have physical meetings with our supervisor and the product owner, does not significantly harm this project, as online meetings will be adequate.

3.1.3 Usage of Software Development Model

Andrea Magnussen will be the scrum master on our team. The product owner is Børre Dervo, as he is the contact person from NINA. Sprint lengths will be set to two weeks per sprint, as the project timeline is quite short. By splitting each sprint into two weeks, we will at least be able to finish four sprints during the project period. By having short sprints, it will keep our product owner frequently updated on the development of the system.

The sprints will start with a sprint planning meeting on the first Monday, and end on the second Friday with a sprint review meeting and a sprint retrospective meeting. The product owner will be included in the sprint review meeting. Every day, there will be a daily scrum. It will be timed and it will last 15 minutes, where each team member updates each other on what they have worked on. The scrum master will time the meeting. The entire product backlog will be tracked using Trello as a Kanban board. We will have one board containing the process of all the tasks in the entire backlog and one board for each sprint.

3.1.4 Plan for Meetings and Decision Points

By the end of April, we need to be finished with a product that satisfies NINA's needs. The final deadline for the entire project, software and thesis, is the 20th of May. The meeting with our product owner at the end of each sprint will be used to keep him updated on where we are in the process, and to get input towards the next sprint. Meetings with our supervisor on Tuesdays will be used to get guidance in progressing the product development, and also for asking questions about technologies, workflow, and thesis writing. See Table 3.1 for an overview of what our sprint schedule looks like.

Table 3.1: Sprint structure.

Week	Mon	Tue	Wed	Thu	Fri
1st	Sprint plan	Daily Scrum Supervisor-meeting	Daily Scrum	Daily Scrum	Daily Scrum
2nd	Daily Scrum	Daily Scrum Supervisor-meeting	Daily Scrum	Daily Scrum	Daily Scrum Sprint retro Sprint review

3.2 Gantt Diagram

Figure 3.1 shows how we have planned to have in total four sprints before Easter, where we will develop the application. We decided early on that we wanted to incorporate thesis writing during development, as shown with the light green color. The plan is to write roughly one or two chapters each sprint and send them to our supervisor for feedback. If everything goes according to plan, we will hopefully have written half of the report when we reach Easter.

With this in mind, developing the application and programming will have a larger priority than thesis writing during the sprints. It is important that we uphold the deadline of having developed the application before the break, so that we are ready for testing. This focus will shift after Easter, like Figure 3.1 shows. We will also conduct a user test to catch flaws, bugs and collect feedback, so that we can improve the system.

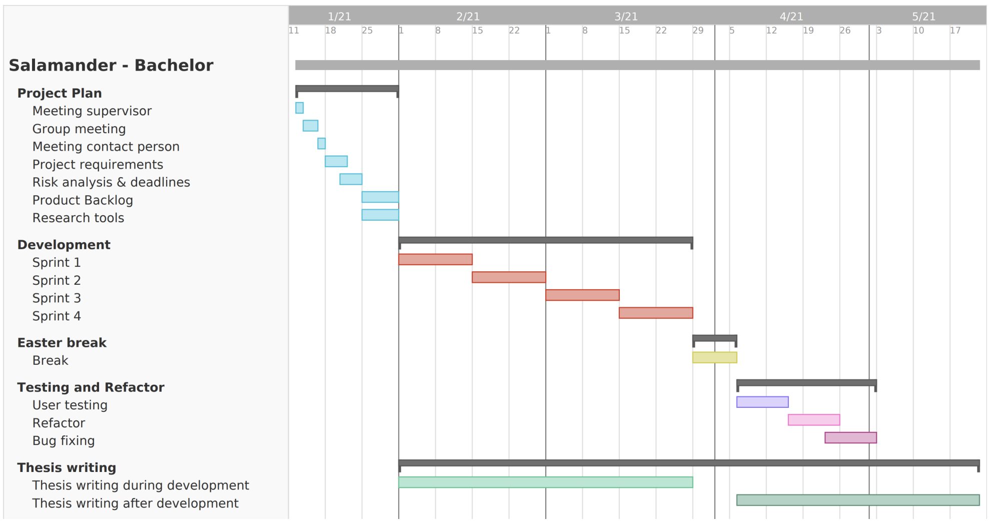


Figure 3.1: Gantt diagram for the project period.

3.3 Development Environment

In the beginning of the project period, we made multiple decisions in regards to development. This includes decisions about technologies and tools that we wanted to use to reach our end goal.

3.3.1 Technology Choices

Mobile Application

Given the desires from the researchers at NINA, we have decided to develop a mobile application. We considered the possibility to develop both a web application and a desktop application. However, the fact that the researchers at NINA wanted something that could aid in fieldwork weighed heavily in on our decision to develop for smart phones and tablets. At some point we considered doing both a mobile and a desktop application. However, due to the limited time frame, we decided to focus exclusively on a mobile application.

For developing the mobile application, we decided to use React Native. According to Statcounter [27], the mobile operating market share in Norway as of January 2021, is that 61.17% of mobile phones run on iOS and 38.61% run on Android. React Native allows us to develop an application that can run on both Android and iOS devices, which was a big factor as to why we decided to use this framework. It is also a framework that is rising in popularity [28]. In addition, we all have experience with native Android development from a previous course, and wanted to explore new technologies. Lastly, three out of four members of our team, use windows computers and are unable to develop native iOS applications easily [29].

We will be using Expo for testing and building our mobile application. Expo is a framework for universal React applications [30]. It is a set of tools and services that is built around React Native and native platforms. Expo aids in developing, building, deploying, and quickly iterating on iOS and Android applications, from the same JavaScript codebase.

Algorithm

The algorithm, developed by the previous group from 2019 [7], was written in Python. This is why we decided to continue working with the algorithm in this language, as it would be very time consuming and complicated to convert to another programming language. The previous group used DeepLabCut for finding the center line of the salamanders. DeepLabCut is an open source library made for pose estimation of humans and animals, and uses deep learning to accomplish this. As this library is made for Python 3.7, we will have to conform to this version [31]. For species and sex estimation we will be using a neural network for object recognition.

REST API

To make sure that the REST API, running on NINA's server, is compatible with the salamander matching algorithm, we decided to use the same programming language, which was Python. We also became restricted to using Python 3.7 quite early, as DeepLabCut needs this version to run properly for our application [31]. There are different frameworks for Python to develop REST API's. We mainly looked at two; Django and Flask.

In 2020, Michael Herman [32] looked at Flask and Django, and compared the two. They are both free open source Python frameworks for building web-applications and API's. Flask is a so called microframework, whereas Django provides more out of the box features. Django is better for large scale applications with bigger teams, while Flask is better for smaller and less complicated projects. Herman also stated in his comparison that Django is best for full web development with routing and templating, and that Flask is better for developing REST API's.

For our use case, which is to develop a REST API that serves a mobile application, we have decided to use Flask. Since Flask is a microframework, we can easily add more Flask-compatible libraries, instead of having to work with unnecessary features that comes with Django. Flask also provides an easy setup that makes it fast to start development.

3.3.2 Tool Overview

Figure 3.2 shows an overview of all the tools and frameworks we plan to use in this bachelor's project including development, documentation, deployment, and collaboration tools.

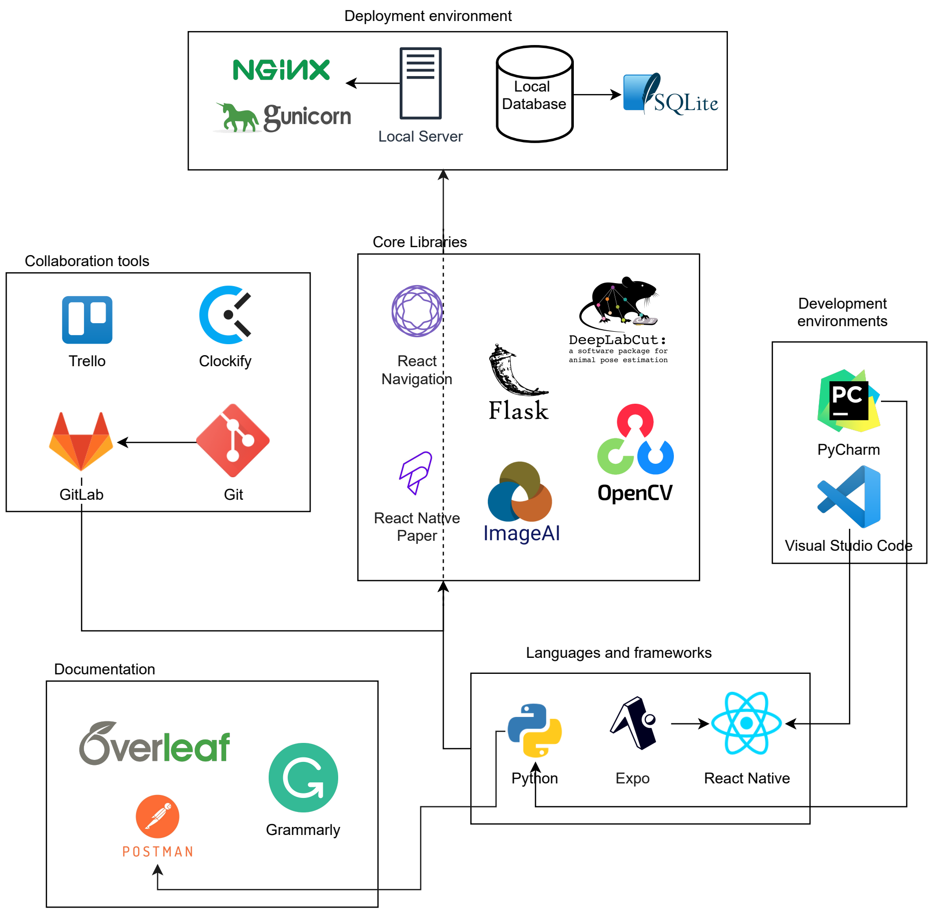


Figure 3.2: Different tools we are going to use in the project.

Chapter 4

Technical Design

In this chapter we will go through the technical design of the completed system. This includes the high level architecture of the entire system, as well as a deeper look at each of the main components.

4.1 System Architecture

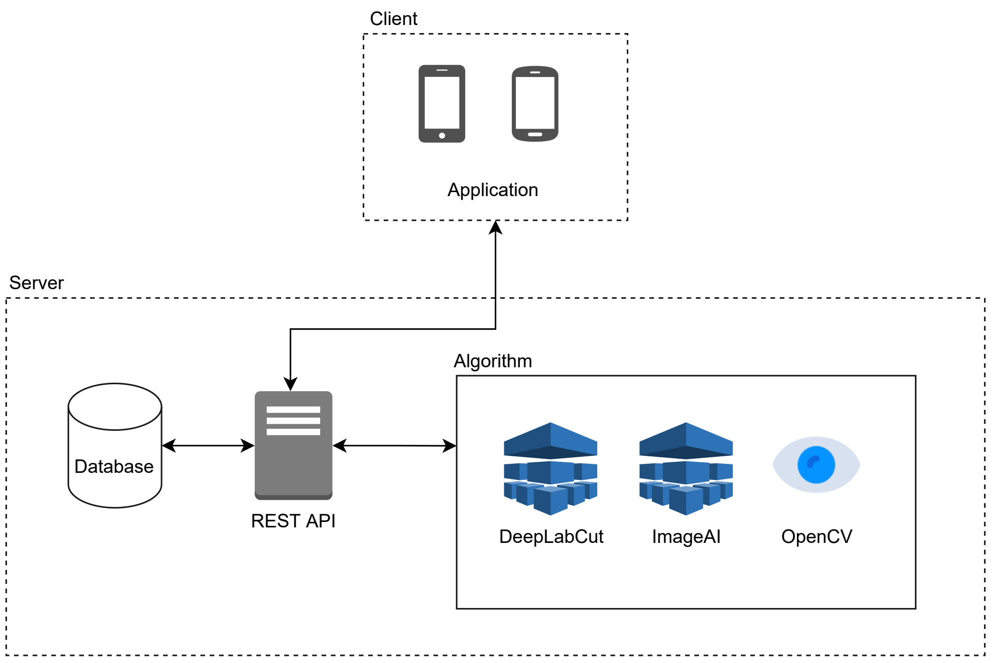


Figure 4.1: System architecture displaying the client/server interaction.

The system is divided into a client (mobile application) and a server (see Figure 4.1). The server consists of the REST API and the algorithm. We have decided to go for a thin client because the system will store a lot of data, primarily consisting of images. This will eventually require more storage space than what is available on a smart phone. In addition, the algorithm has to be as precise as possible to reduce incorrect identifications. Running this algorithm reasonably fast will therefore require significantly more processing power than what is available on a modern smart phone.

Due to our choice of developing a thin client, the mobile application relies on having a connection to the server to function properly. We also wanted to make sure that the data is always consistent across all devices at all times, which means the user needs to be connected to the server to register locations and images of caught salamanders.

4.2 Networking

There are several libraries that makes it possible to send and receive network requests and API calls from a client to a server. We mainly looked at two ways of achieving this; Axios and Fetch.

Fetch provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses [33]. However, compared to Axios, it requires more work by the developer to fully utilize the freedom it gives. Axios have several built in functionalities that would help reduce development time and unnecessary overhead. One of these functionalities is the ability to create one standard instance of Axios that could be utilized everywhere. In addition, Axios has built in Cross-Site Request Forgery protection [34]. If we were to justify the use of Fetch it would mean that we would have to provide solutions for Cross-Site Request Forgery that are better than what Axios's API has to offer.

All in all, we decided to use Axios to handle communication between the client and the server, as it seemed to fit our use case, and provided extra security.

4.3 Front End

In this section, a selection of the most important libraries used in the React Native application, and an overview over the navigation between the different screens are covered.

4.3.1 Libraries

We used several third-party libraries for creating this mobile application. Libraries worth mentioning are:

- **React Navigation**: A routing and navigation library for React Native [35], which makes navigation more elegant by removing boiler plate code and

taking care of the logic.
- **React Native Paper**: A standard-compliant Material Design library for React Native [36], which allows for importing already designed GUI components.
- **React Native Axios**: Discussed in Section 4.2, and handles network requests in the application.
- **React Redux**: An open-source JavaScript library for managing application state [37].

In addition to these libraries, we have used several smaller libraries to complement our application. These will be covered in further detail in Chapter 7 Implementation.

4.3.2 Navigation Overview

The purpose of the following diagrams is to give an overview of how the different screens are connected, and the different paths a user can take to navigate between them.

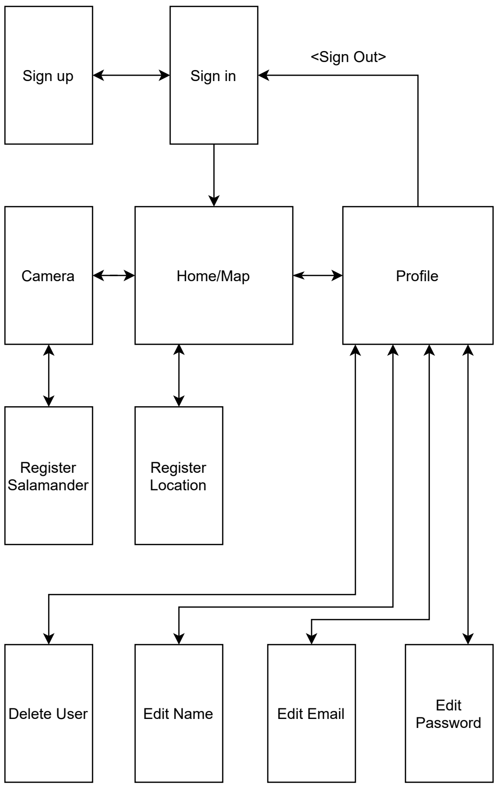


Figure 4.2: Initial navigation diagram.

Figure 4.2 shows how we initially planned to connect the various screens in the mobile application, and is based on the Adobe XD prototype (see Appendix G.2). There was no focus on implementing a feature for an admin to manage salamander data after registration. However, this use case was added during the last sprint (see Section 5.2.4).

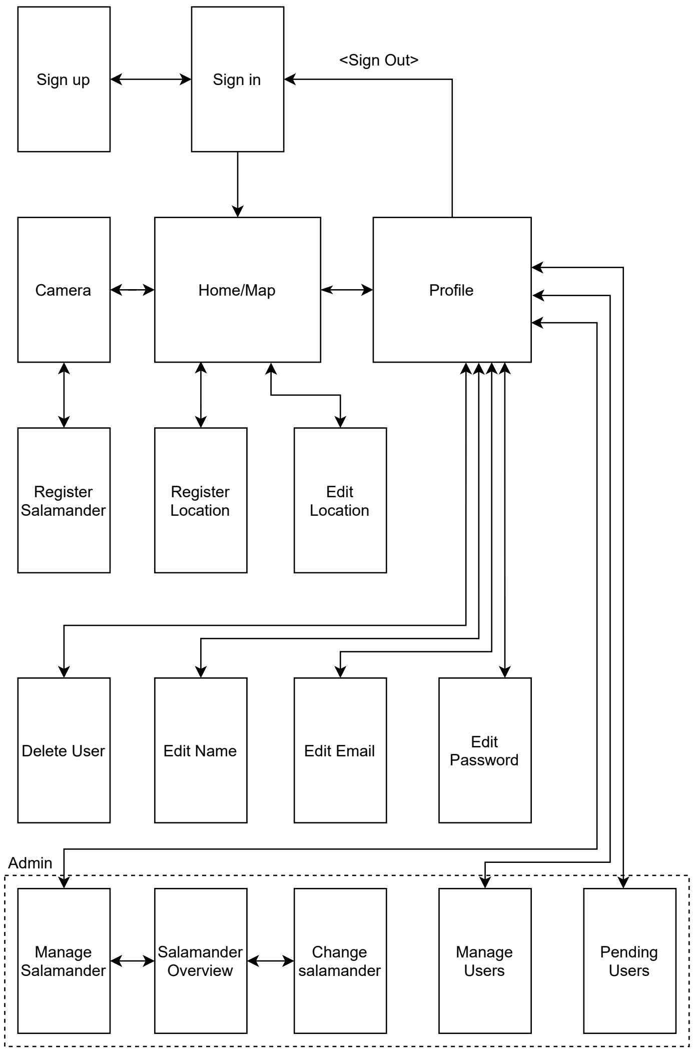
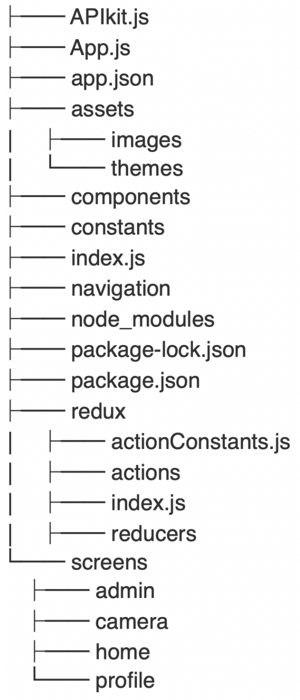


Figure 4.3: Final navigation diagram.

Figure 4.3 shows how the screens in the final product are connected. If we compare the final diagram to Figure 4.2, we added several more screens to the application. Primarily we added screens for an administrator allowing for management of users and salamanders. All of these screens are located inside the admin field at the bottom of the diagram. All admin related screens are located together under the same category in the profile settings. Lastly, we also wanted all users to have the ability to edit existing locations from the home screen.

4.3.3 File Hierarchy

In Figure 4.4 the overall file hierarchy of the mobile application is shown.



```
├── APIkit.js
├── App.js
├── app.json
├── assets
|   ├── images
|   └── themes
├── components
├── constants
├── index.js
├── navigation
├── node_modules
├── package-lock.json
├── package.json
├── redux
|   ├── actionConstants.js
|   ├── actions
|   ├── index.js
|   ├── reducers
└── screens
    ├── admin
    ├── camera
    ├── home
    └── profile
```

Figure 4.4: File hierarchy in the mobile application.

We divided the files based on their general purpose; assets, components, constants, navigation, redux and screens. Node modules has to be included in all React Native projects, and it includes all the imported libraries.

4.4 Back End

This section will cover the REST API and algorithm, and how they are structured to fit our case.

4.4.1 REST API

The REST API that runs on the server is written in Python. As discussed in Chapter 3, we chose this language so that the REST API and the algorithm could easily communicate together.

Libraries

The main library we used for creating the REST API is Flask. We also used other libraries, including Flask extensions. The libraries worth mentioning are:

- **Flask**: A web development library for creating API's [2].
    - **Flask-restful**: An extension which helps to create REST API's [38].
    - **Flask-jwt-extended**: An extension to handle JSON Web Token for authentication [3].
    - **Flask-sqlalchemy**: An extension to handle database functionality [39].
    - **Flask-bcrypt**: An extension to hash passwords and compare passwords to the hashed ones [40].
    - **Flask-limiter**: An extension that allows for easy implementation of rate limiters for endpoints [4].
- **Image-encoder**: Takes in a file path to an image and returns a string representation of Base64 encoded bytes [41].

Using Flask-restful allowed us to easily create different endpoints and use different HTTP methods on the same endpoint, such as POST, GET, PUT and DELETE.

Endpoints

All data is transferred as form data from the client and returned as JSON from the server. All HTTP requests, except for login and sign up, are authenticated using a JSON Web Token created by the server and transferred to the client. It serves as an identifier for each user by encrypting their unique user ID inside it. The token is sent with each request and is decrypted by the server (see Section 7.2.1 for further details).

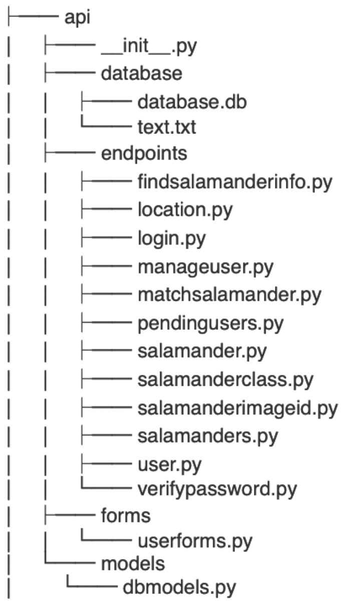
The API consists of 12 different endpoints. Some of these endpoints accepts different HTTP methods. In total, there are 21 ways of requesting the API. For example, the location endpoint has four allowed HTTP methods while the match salamander endpoint only has one. The different HTTP methods that we used throughout the project are GET, POST, PUT and DELETE. For instance, the location endpoint that accepts all HTTP methods, uses GET to get information about all the locations and POST for registering a new location. PUT is used for editing the location, which includes name and radius, and DELETE is used for deleting a location.

The API is documented using Postman. This documentation explains the usage of all of the endpoints and their different HTTP methods. The full documentation can be found in Appendix D.

For storing data on the server, we use a file structure (Figure 4.6) to store processed and original images, and a local database (Figure 4.7) to store all other data such as users, metadata for salamanders and locations. All processed images are always stored as JPG images, and the originals are stored in their original format. However, the server only accepts JPG, jpeg and PNG formats. For the local database we use Flask-SQLAlchemy which uses Sqlite dialect.

File Hierarchy

Figure 4.5 shows how we set up the project structure for the REST API.



```
├── api
|   ├── __init__.py
|   ├── database
|   |   ├── database.db
|   |   └── text.txt
|   ├── endpoints
|   |   ├── findsalamanderinfo.py
|   |   ├── location.py
|   |   ├── login.py
|   |   ├── manageuser.py
|   |   ├── matchsalamander.py
|   |   ├── pendingusers.py
|   |   ├── salamander.py
|   |   ├── salamanderclass.py
|   |   ├── salamanderimageid.py
|   |   ├── salamanders.py
|   |   ├── user.py
|   |   └── verifypassword.py
|   ├── forms
|   |   └── userforms.py
|   └── models
|       └── dbmodels.py
```

Figure 4.5: File hierarchy of the REST API.

The API folder is the root and is further divided into four sub folders. The database folder contains the local database for storing data about users, salamanders and locations. The endpoint folder contains all the different endpoints that the mobile application can request. The forms folder contains user related forms for input validation, and the models folder contains all the database models for the different tables in the database.

4.4.2 Algorithm

Libraries

- **DeepLabCut**: A library for pose estimation of humans and animals using neural networks [42].
- **Tensorflow**: A free open source library for machine learning [43]. Used in DeepLabCut and ImageAI.
- **ImageAI**: A library for generic categorization and object detection [44].
- **OpenCV**: A broad library with several functionalities for image manipulation [45].

The algorithm utilizes two separate neural networks to determine sex and to find the abdominal pattern of a given salamander. The previous bachelor's group used DeepLabCut to train a *ResNet_50* model in one of their attempts to estimate the abdomen of the salamander. After looking further into this library we decided to also use DeepLabCut for our project. DeepLabCut uses Tensorflow to create their models and they support models of various depths.

DeepLabCut is used to identify the abdomen, and ImageAI is used to determine the sex, as seen in Figure 4.1. Both of these libraries use Tensorflow for model generation and estimation [46] [42]. As precision is paramount we use *ResNet_152* for DeepLabCut and YoloV3 for ImageAI. These models produce great accuracy, but require more computation time than less accurate alternatives [47] [48].

4.5 Data Storage

In the project planning phase, we had a meeting with our product owner, where we began discussing how we were going to structure our database and storing the images. We also discussed how to best register the salamanders in the system, both for the researchers at NINA, and for the algorithm to be as effective as possible.

At first, we discussed dividing all captured salamander images into folders based on either county, municipality or city. This would be chosen by the researchers each time they would upload an image. This could reduce the time it would take for the algorithm to search through images, as it would only need to search through a single folder. The problem with this approach was that some of these ponds were located at the borders of the areas. There might be some confusion for the researchers to determine the right area for the pond. This would be critical, because the algorithm might search through the wrong area.

The next approach we looked at was to read the image metadata and extract the coordinates. According to our product owner, a salamander usually does not wander more than about 500 meters away from its birth pond. This means that there is no point in matching a salamander against others caught more than 500 meters away. When a salamander is caught we would then be able to calculate the coordinates and search through all the salamanders that are inside of a 500 meter radius from the image metadata. However, there are multiple issues with this approach. Based on the workflow of the researchers, this would not be suitable. Sometimes they will bring caught salamanders to a lab or another area, and if the system used this approach when matching salamanders, the coordinates would be incorrect. Another issue with this approach is that the coordinate of an image may not necessarily correspond to the salamanders birth pond, which violates the assumption that a salamander only wanders 500 meters away from the center of the pond.

After realizing that our approaches would not fit the needs of either the researchers or the algorithm, we had to look at other solutions. We have come up with a registration system, where the researchers at NINA can register locations on a map on their mobile device. When they register a location, they will give it a name and radius, and the coordinate will automatically be registered in the database. The purpose of the radius is only to visualize how big the area is. In this way, the researchers will have full control over which locations cover what ponds. When they upload images, they should easily be able to choose the right location from those which have been registered. When a location is selected and an image of a salamander is posted to the REST API, the server will automatically match the salamander with other salamanders that were caught at this location. Our product owner found this way of storing salamanders to be intuitive, easy to use, and effective for matching salamanders.

We created a folder structure that would accommodate this design, shown in Figure 4.6. The folder structure is divided into locations, species, and sex of the salamander. When registering a new location, the system will create a new folder with the location name, containing all sub folders.

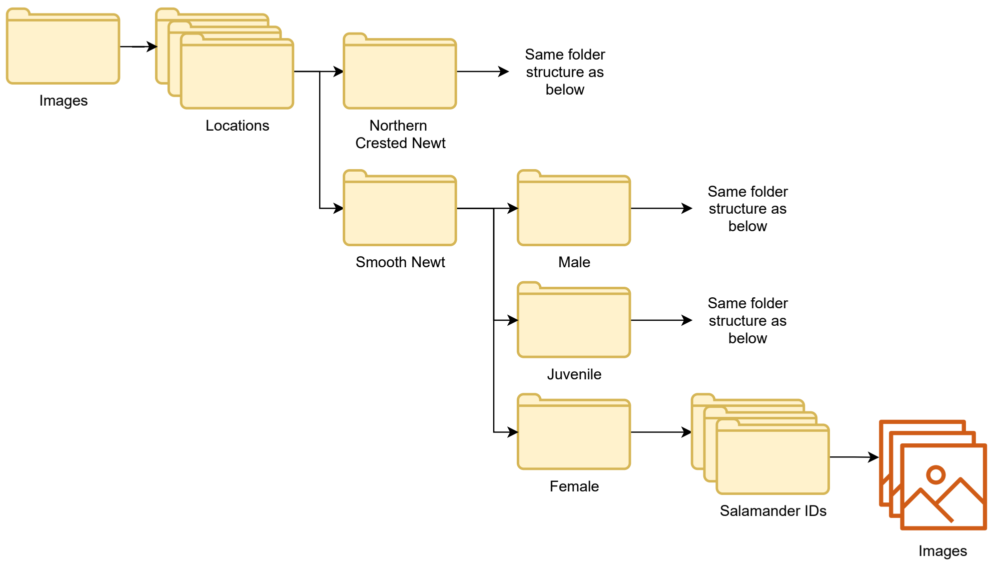

Images
Locations
Northern Crested Newt
Same folder structure as below
Smooth Newt
Male
Same folder structure as below
Juvenile
Same folder structure as below
Female
Salamander IDs
Images

Figure 4.6: How the images of the salamanders get stored in the folder structure.

Based on our final design choice, we made a database diagram, Figure 4.7, that shows how data is stored on the server and how this data is connected. When a salamander is registered in the database, it will get a foreign key to the user that registered it and a foreign key to where it was caught. The researchers will also be able to register the salamander's weight and length. The purpose of the SalamanderGrowth table is to track the individual salamander over time. The location table will include a unique ID, name, longitude, latitude and radius. With these data we can render all the locations in the mobile application. The date logged in SalamanderGrowth is not read from the metadata stored in the image file, but rather the date the image was uploaded to the server.

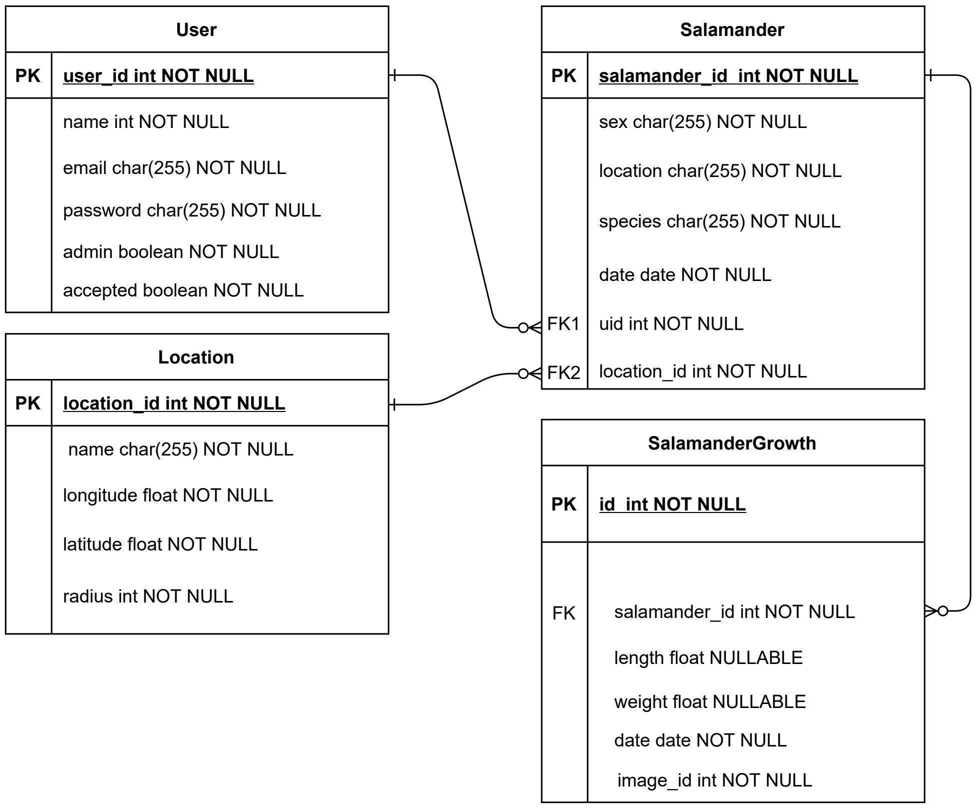


Figure 4.7: ER-diagram of our database.

Chapter 5

Development Process

This chapter will describe our usage of different tools during the development process, as well as what we did in each sprint. We had four sprints, where each sprint was two weeks long. This chapter will not go into detail about implementation, but simply give a summary of what was implemented in each sprint. Implementation details will be covered in Chapter 7.

5.1 Tools

We used various tools during development to keep track of our progress and to estimate workload. We used Trello and planning poker in all sprints. The plan was also to use a sprint burndown chart to keep track of our progress during the sprints.

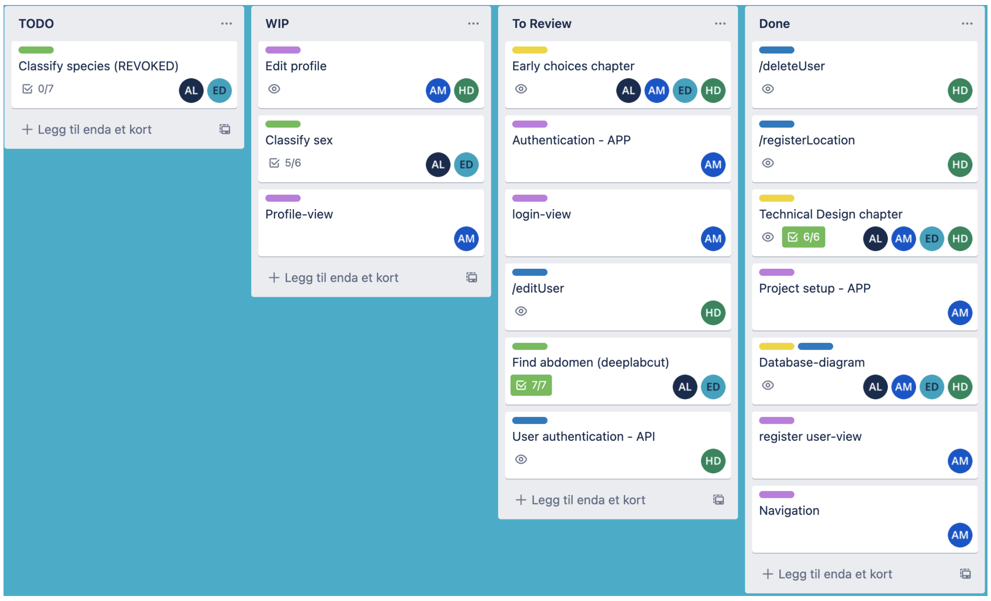
5.1.1 Planning Poker

During each sprint planning meeting, tasks from the product backlog were chosen and estimated using planning poker. For each sprint, we estimated 240 hours total workload, which is 30 hours per person per week, for two weeks. However, we decided to only estimate development related tasks when using planning poker, which means we would use the remaining time working on thesis related activities.

When one task was chosen from the product backlog, a short introduction of the task was given. Each group member then gave a time estimate in hours using Fibonacci numbers. A discussion followed after each time estimate, and the final estimate of the task was set. After all tasks were given an estimate, they were added to the sprint backlog. If the total estimate differed significantly from our estimated total time use per sprint, we would either remove or add tasks to the planning poker. The minimum amount of estimated workload was set to 150 hours and the maximum was set to 240 hours. For the actual number of hours spent each sprint, see Chapter 10.

5.1.2 Trello

We used Trello throughout the development process to keep track of the sprint backlog for each sprint. Figure 5.1 shows an example of how we used this tool for a single sprint in the development process.



TODO
- Classify species (REVOKED) — 0/7 — AL ED
- + Legg til enda et kort

WIP
- Edit profile — AM HD
- Classify sex — 5/6 — AL ED
- Profile-view — AM
- + Legg til enda et kort

To Review
- Early choices chapter — AL AM ED HD
- Authentication - APP — AM
- login-view — AM
- /editUser — HD
- Find abdomen (deeplabcut) — 7/7 — AL ED
- User authentication - API — HD
- + Legg til enda et kort

Done
- /deleteUser — HD
- /registerLocation — HD
- Technical Design chapter — 6/6 — AL AM ED HD
- Project setup - APP — AM
- Database-diagram — AL AM ED HD
- register user-view — AM
- Navigation — AM
- + Legg til enda et kort

Figure 5.1: Example of Trello usage in sprint 2.

We divided the board into four categories; TODO, Work in Progress (WIP), To Review and Done. As the sprint progressed the tasks would move from TODO, and eventually end up in the Done category. We color coded the tasks into four different categories. Purple was used for tasks regarding mobile application development, blue was used for REST API related work, green was used for the image recognition algorithm and yellow was used for tasks covering thesis related activities. We also assigned group members to each task.

5.1.3 Sprint Burndown Chart

We initially planned to draw a sprint burndown chart for every sprint. During the first two sprints this was done successfully, as seen in Appendix J. However, during the last two sprints we became increasingly less consistent with our updates to the diagram, eventually leading to the discontinuation of this tool in sprint 4.

One of the reasons for the discontinuation of sprint burndown charts, was that we got little value from using it. We realized we were able to maintain awareness of our progress without using this tool. This realization resulted in us prioritizing other work over the burndown chart. Due to our style of working, as we mostly worked together on campus, we could easily keep everyone up to date. We believe that using sprint burndown charts would have been more useful if the sprints were longer, and the group members worked remotely, and only communicated during the daily scrum meetings.

We also found it quite hard to estimate remaining workload, as we were inexperienced with the Scrum methodology. At times we would add more tasks to the sprint backlog, which made estimating challenging. Some of the larger tasks were also hard to accurately estimate as there was significant uncertainty regarding the time needed to complete the task. This was due to us not having experience with several of the technologies that was used in this project.

5.2 Sprint Overview

In this section we will cover the four sprints from this project. Each sprint will be given an overview, as well as a summary of the progress for the mobile application, REST API, and algorithm.

5.2.1 Sprint: Getting Started (01.02.2021 - 12.02.2021)

The first sprint started in the beginning of February, and was mostly focused on formalizing requirements and getting to know the technologies we decided to use. Table 5.1 shows the resulting estimates from this sprints planning poker.

Table 5.1: The tasks estimated during planning poker in sprint 1.

Task	Name	Time (hours)
1	/matchSalamander endpoint	8
2	Database and image storage	13
3	Database-design	13
4	Project setup - REST API	2
5	React Native research	8
6	Find abdomen - deeplabcut	89
7	Paper prototype	10
8	Adobe XD prototype	10
Sum:		**153**

In addition to the tasks estimated in Table 5.1, a lot of time during this sprint also went to writing the requirements.

Mobile Application

The main focus in this sprint was design and user experience in regards to the mobile application. A simple prototype was made and shown to our product owner (see Appendix G.1). We got feedback, and a more detailed prototype was made in

Adobe XD (see Appendix G.2). After receiving feedback on this as well, we were ready to develop the application.

REST API

The first goal for the REST API was to setup the project and get the library Flask to work with some basic endpoints. Initially there was some difficulties with Python, Pycharm, and PIP. However, after restructuring the project, the errors got sorted out.

The endpoints that were implemented in this sprint were; a salamander matching endpoint, a login endpoint with JWT authentication, a find salamander data endpoint, and a register user endpoint.

A database using Flask-SQLAlchemy was also implemented. It consists of three tables; user, salamander, and location. A lot of time went into researching authentication, database, and setting up the project structure to avoid errors.

Algorithm

The source code from the previous bachelor's thesis [7] was missing crucial functionality, such as automatically locating the abdomen of the salamander. Two members were therefore assigned the task to train a neural network model and change some of the source code to satisfy the requirements.

The images we received from our contact person at NINA was in JPG format and was taken in a high resolution. As our neural network works faster with images in lower resolutions, we had to downscale all the images. Additionally we had to convert the images to PNG format to be compatible with DeepLabCut.

We encountered an issue with getting DeepLabCut to use the GPU on our computers. This was problematic as using only the CPU would make the training and estimating take significantly more time. After four days we managed to solve this issue and was able to use the GPU.

5.2.2 Sprint: Accelerating (15.02.21-26.02.21)

The second sprint started in the middle of February, and marked the beginning of the development of the software solution. We spent time on designing our database, and implemented the various libraries and frameworks we chose in sprint one. Table 5.2 displays the resulting estimates from the planning poker session.

Table 5.2: Planning Poker for sprint 2.

Task	Name	Time (hours)
1	Register user-GUI	3
2	Profile-GUI	6
3	Login-GUI	2
4	/registerLocation endpoint	3
5	Authentication - REST API	2
6	Classify species and sex	89
7	Authentication - APP	3
8	Navigation - APP	8
9	Project setup - APP	2
10	System architecture	14
11	/editUser endpoint	4
12	/deleteUser endpoint	1
13	Find abdomen - deeplab	13
Sum:		**150**

In addition to the tasks estimated in Table 5.2, a lot of time during this sprint also went to documenting the technical design and the development plan.

Mobile Application

In the second sprint, all the screens designed in the first sprint were implemented, and the application was tested in the Expo development environment. Navigation was also implemented, and it became possible to navigate to all the different screens. We also started on networking with Axios, and tried to connect the application to the server. This was the main focus in the next sprint.

REST API

The second sprint mainly consisted of creating more endpoints, and adding more tables and entries to the database. We made an endpoint to verify passwords and an endpoint for a user to be able to edit personal information such as password, name and email. A location registration endpoint was also created, which lets a user register a location with a name, latitude and longitude, and a radius. These data would then be stored in the new database that was created, see Figure 4.7.

In this sprint we also merged some of the algorithm with the REST API. The find salamander data endpoint now used the image processing AI for straightening and cropping the abdominal pattern. We added the matching algorithm to the match salamander endpoint to match the processed image with other salamanders inside a specified folder in the hierarchy.

Algorithm

In this sprint we continued with the matching logic. Specifically the model made in DeepLabCut got improved. We added extra points in order to find the width of the abdominal pattern by locating the shoulders. The model was changed from *ResNet_50* to *ResNet_101*. In addition, two functions were made to communicate with the REST API to estimate and match a salamander.

We started to work on the automatic identification of sex. We decided to only classify males and females, as we got far too few images of juveniles to properly train a model. In the beginning we tried making an image classification model to classify the sex, but ran into precision issues. Because of this we went with an object identification method instead, which would identify the cloaca (the intestinal, urinary, and genital tracts), of a given salamander and identify the sex based on it.

Finally we also decided to not identify species, because the images we got from NINA was only of northern crested newts. Further discussion behind this decision will be covered in Chapter 7.

5.2.3 Sprint: Assembling (01.03.21-12.03.21)

The third sprint started the first day of March. It was mostly focused around the development of the mobile application. The REST API and algorithm was merged together and we continued on automatically identifying the sex of salamanders. This sprint was almost exclusively focused on software development, and thesis writing was put on hold. Table 5.3 shows the results of the planning poker session.

Table 5.3: Planning Poker for sprint 3

Task	Name	Time (hours)
1	Classify sex	10
2	Merge algorithm with REST API	3
3	Merge find abdomen and classify sex algorithms	1
4	Sign out - APP	3
5	Camera - GUI	80
6	Register salamander - GUI	35
7	Store data fetched from API - APP	6
8	Home - GUI	55
9	Edit Profile - APP	15
10	Visual feedback - APP	10
Sum:		**218**

Mobile Application

For the third sprint, a lot of time went to develop the mobile application. Everyone in the group started to work on different aspects of the application, as it at this point had to be connected to the REST API. The camera functionality was implemented, as well as the map functionality. An instance of Axios was also implemented, and was used to handle all the HTTP requests. We also implemented state management using Redux, to make the data in the application persistent over multiple screens.

Most of the originally planned features was implemented during this sprint, and we were able to give a demo to our product owner. During this meeting, our product owner wondered whether it was easy for the researchers at NINA to manage the registered salamanders. We came to the conclusion that it would not be that easy for them to manually move or edit salamander data on the server side, and decided that we should add functionality in the application that would cover this. This was the focus in the last sprint, as well as looking at deployment of the mobile application.

REST API

For this sprint we had not planned on more development of the REST API. Eventually, it became clear that further development was needed to fulfill the new functionality added in the front end.

The main feature that was implemented was an endpoint for administrators to manage users, by accepting or denying their access to the system. This endpoint is restricted to only admin users so that non-admin users will not have access to these features. We also implemented an endpoint for all users to change their passwords.

There were also some bug fixing and refactoring. We implemented a rate limiter to the REST API using Flask-Limiter, so that each endpoint has a set limit per minute. This was implemented so that the system would be less vulnerable to DOS attacks. Lastly we implemented status codes to the message that was returned to the client from the API.

Algorithm

In this sprint we continued on developing the sex identification algorithm. An issue we encountered was that DeepLabCut, and ImageAI used for sex identification, would not work together. This was because the two libraries would fight for GPU resources, and because of issues with Tensorflow. We also implemented a semaphore in the algorithm to control access to the GPU. Lastly we chose to run all code using Tensorflow in their own processes to free up GPU memory after use.

After testing the sex identification AI multiple times, we did not get acceptable accuracy, leading us to abandon the idea. Further discussion about this decision will be covered in Chapter 7 Implementation.

5.2.4 Sprint: Ragnarok (15.03.21-26.03.21)

The final sprint started in the middle of March and had a major focus on administrative features allowing for quality control and salamander data management. Several new API endpoints and mobile application screens were created. This was additional features we had not planned for, but decided to implement after the meeting with our product owner in the sprint review meeting of the previous sprint. We also began looking at deployment. Table 5.4 shows the results of the planning poker session for this sprint.

Table 5.4: Planning Poker for sprint 4

Task	Name	Time (hours)
1	Deployment - APP	34
2	Deployment - REST API	55
3	Sanitize input - APP	12
4	Move salamander endpoint	8
5	Salamander overview - GUI	7
6	Salamander list - GUI	8
7	Edit salamander - GUI	14
8	Get salamander images endpoint	3
9	Get salamanders endpoint	3
10	Delete salamander endpoint	10
11	Rematch salamander	3
12	Merge find abdomen and classify sex algorithms	2
Sum:		**159**

This sprint was focused on development, as this was our last sprint and the new features we wanted to implement were quite challenging and time consuming. To be able to implement these new features, we needed to conduct new requirement specifications and create a use case. Table 5.5 is a low level use case of the manage salamander requirement.

Table 5.5: Low-level use case for managing registered salamanders.

Use Case:	Manage salamanders
Actor(s):	Admin
Goal:	Ensure that registered salamander data is correct.
Description:	An admin can see a list of all caught salamanders based on a location, see images and data on one specific salamander, and change these data if necessary.
Preconditions:	There has to be registered salamanders in the database on the given location.
Post-conditions:	Server sends a response back to client.
Special Requirements:	The request needs to come from an authenticated user with the correct access privileges.
Success Scenario:	1. The admin navigates to the manage salamander screen. 2. The admin verifies their password. 3. The admin selects a location and fetches all salamanders from the server on the given location. 4. The admin scrolls through a list of salamanders and clicks on the one they want to look at. 5. The admin navigates to a new screen containing all the original images of the salamander. 6. The admin chooses one of the original images and gets navigated to a new screen containing all salamander data for the specific salamander. This includes weight, length, location, species and sex, in addition to the original and processed image. 7. The admin verifies the data. 8. The admin click the confirm button when data is verified and gets navigated back to the salamander list.
Alternative Scenarios:	4. If the admin does not want to manage the salamanders, they can click the done button instead. 7a. The admin may delete the salamander from the database. 7b. The admin may update any of the salamander data. This includes weight, length, location, species and sex. If the sex, location or species is changed, the back end will match the salamander against the salamanders new data.
Fail Scenarios:	2. The user does not have the right privileges, and the server will abort the request. 3. There are no registered salamanders on the location that the admin chooses. 2-8. Internal server error when trying to process the request.

Mobile Application

As mentioned earlier, this sprint was focused around implementing the new feature regarding management of salamanders. We used approximately one week to implement it, and the last week was used to debug and polish the application. We also added the ability to edit and delete a location, as well as making it possible for the admin to reset a users password.

REST API

During this sprint, the main tasks were to implement endpoints that would work with the new use case as seen in Table 5.5. This included an endpoint for getting all salamanders based on a location, getting all original salamander images based on its ID, and get the original and the processed image of a specific salamander. We also implemented an endpoint for deleting salamanders and changing their data. If data about the salamander's species, sex or location is changed, the salamander would need to be re-matched.

Algorithm

In this sprint, not much happened with the algorithm. The only notable change was moving from using *ResNet_101* to the deeper network *ResNet_152* for DeepLab-Cut. This was done to improve accuracy further, but required more powerful hardware to train on. See Section 7.2.2 for further details on this.

Chapter 6

Graphical User Interface

This chapter will cover choices we made in regards to colors, GUI-elements and visual feedback, in the mobile application.

6.1 Prototyping

During the first sprint we made a simple prototype of the application (see Appendix G.1) and presented it to our product owner. After receiving feedback, we reiterated and made a more complex and interactive prototype in Adobe XD. This was shown to our product owner during the sprint retrospective meeting of sprint 1, so he could get a refined impression of the functionality of the application, and its workflow. Screenshots of the Adobe XD prototype can be seen in Appendix G.2, in addition to a clickable link to the interactive version.

6.2 Aesthetics

This section covers choices in regards to colors, application icon and an overview of how the main screens are designed.

6.2.1 Colors

When deciding the colors in the application, we decided to use NINA's logo (see Figure 6.1) as a reference. Light blue is used as a background color in the sign in and sign up screens, and orange is used throughout the applications headers and the bottom navigation bar. For the buttons we used a dark blue color.

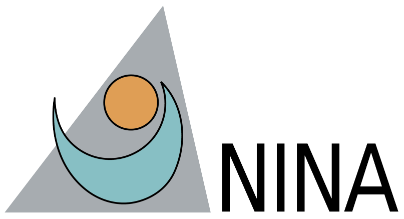


Figure 6.1: NINA's logo.

6.2.2 Application Icon

When drawing the icon for the application we designed a salamander (see Figure 6.6, and made it look like an *S*, as in *Salamander*.

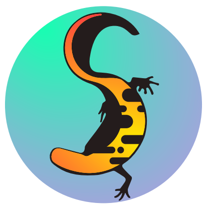
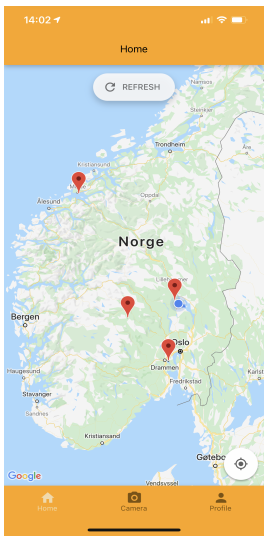


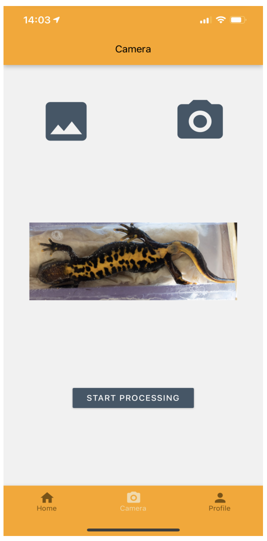
Figure 6.2: Salamander application icon.

6.2.3 Main Screens

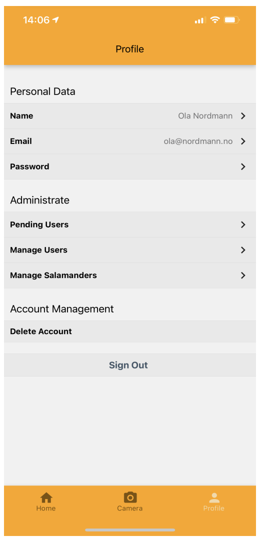
The finished application consists of three main screens; a home screen, a camera screen and a profile screen. The overall look and feel can be seen in Figure 6.3. The rest of the final GUI of the application is shown in Appendix H.



(a) Home screen.



(b) Camera screen.



Personal Data
Name Ola Nordmann
Email ola@nordmann.no
Password
Administrate
Pending Users
Manage Users
Manage Salamanders
Account Management
Delete Account
Sign Out

(c) Profile screen.

Figure 6.3: Images of the main screens.

6.3 GUI-elements

To avoid spending too much time on designing the GUI-elements ourselves, we decided to use React Native Paper. By using this, our application got a modern look and feel with minimal effort, as well as automatically adjusting the GUI to the operating system of the mobile device.

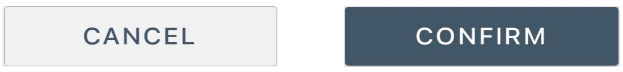
6.3.1 Navigation Bar

As seen in Figure 6.3, we decided to use a bottom navigation bar to handle navigation between the three main screens. According to Material Design [49], a bottom navigation bar should only be used if there are between three and five main destinations, which there are in our case. We also discussed using a drawer style navigation menu as well. However, this would only be useful if the application had more than five main screens to traverse between [50].

6.3.2 Buttons

When positioning buttons, we followed this article by Carlson [51]. Secondary buttons, typically representing cancellation of an action, were mostly placed on the left side of the screen. Primary buttons, typically representing an action, were placed on the right side. This is due to western style of reading going from left to right, which means the primary button will be the last destination.

In cases where the keyboard was triggered automatically, the buttons were placed vertically, as shown in Figure 6.8. This choice was due the keyboard creating noise when triggered, and we wanted to make sure the buttons were easily noticeable. The primary button was placed on top, as it is the most likely outcome when the user navigated to such a screen.

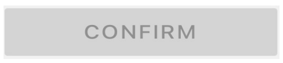


(a) Secondary button style. **(b)** Primary button style.

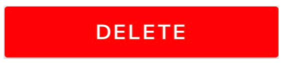
Figure 6.4: Buttons in the application.

We decided to use the mode "contained", seen in Figure 6.4b, for primary buttons. For secondary buttons we decided to go with the mode "outlined", as seen in Figure 6.4a. This button is less highlighted than the primary one, but still noticeable enough for a user to see it.

For special case buttons, such as disabled buttons or buttons for critical operations, the respective styles are shown below in Figure 6.5a and Figure 6.5b.



(a) Disabled button style.



(b) Delete button style.

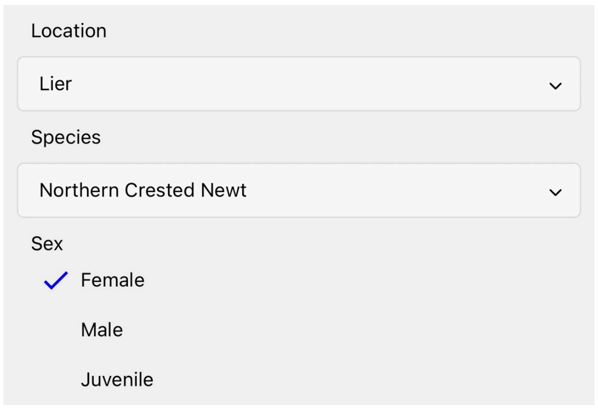
Figure 6.5: Special case buttons in the application.

6.3.3 Radio Buttons and Dropdown Menus

When a user is working with standardized data, we decided to implement a radio button group or a dropdown menu. By using a dropdown menu or radio button group, it is certain that the values the user chooses are correct. If the values could be entered through a text input field, the user could have typos or write values that do not exist.

For specifying sex, radio buttons were used, as it is only three options; female, male and juvenile. According to an article by Minhas [52], dropdown menus should only be used if there are more than five options, or if the default option is the recommended one. This is not the case for sex, as it can differ from case to case.

We used dropdown menus for locations due to the possibility of many locations being registered. In addition, the last selected location is always remembered, so the user will not have to select the location again. A dropdown was also used in the case of species. Currently, the application only works properly for northern crested newts, which is set as the default option. In addition, more species might be added in the future.


Location: Lier

Species: Northern Crested Newt

Sex: [✓ Female] Male Juvenile

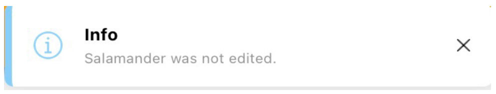
Figure 6.6: Our use of dropdowns and radio buttons.

6.3.4 Use of Modals

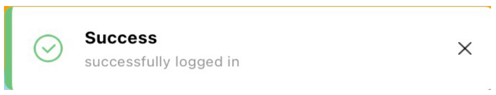
In each main screen there are tasks that require input from the user. In the home screen, the user may want to register a new location or edit an existing one. To do one of these tasks, we have decided to utilize a modal screen that slides up from below. This modal will not display the bottom navigation bar, and can be thought of as a sub screen from their main screen. The user will either have to cancel or complete the operation to return to the main screen. In this way, a user can focus on the task at hand, and not be distracted by other functionality in the application.

6.3.5 Visual Feedback

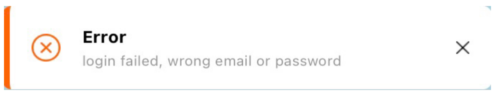
When users interact with the system they will expect some form of feedback. When they make requests, the server will send back a response, even if the response is an error. We decided to show this feedback in the form of a toast message. A toast provides simple feedback about an operation in a small popup, and only fills the amount of space required for the message [53]. The popup will disappear if the user dismisses it, or its timeout has passed. The different toasts a user can get is info, success, and error (seen in Figure 6.7). These toasts are displayed when a request has been processed.


Info
Salamander was not edited.

(a) Toast Info.


Success
successfully logged in

(b) Toast Success.


Error
login failed, wrong email or password

(c) Toast Error.

Figure 6.7: Images of the different toast messages in the application.

In addition to toast messages, we have also implemented responsive input validation in our application. This is showcased in Figure 6.8.

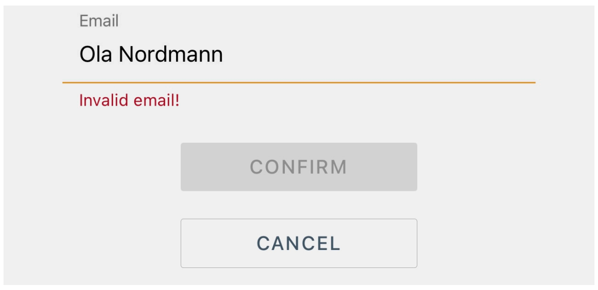
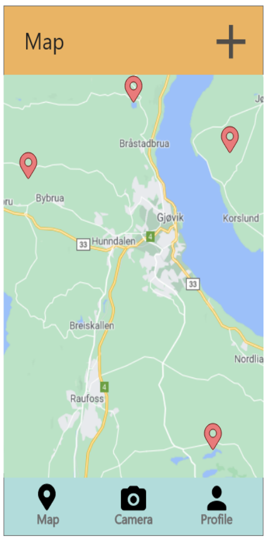

Email
Ola Nordmann
Invalid email!

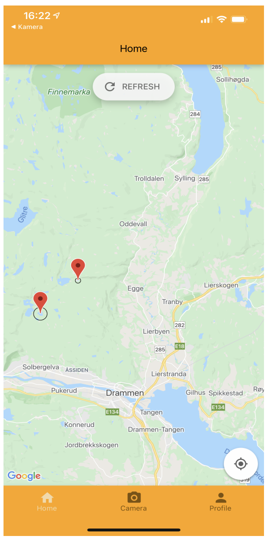
Figure 6.8: Helper text when typing in email.

The user will not be able to proceed if they have not provided the correct data, and they will be visually notified of what is wrong. We use regular expressions for validating text input. Each regular expression is shown in Appendix K.

6.4 GUI-evolution



(a) Prototype version.

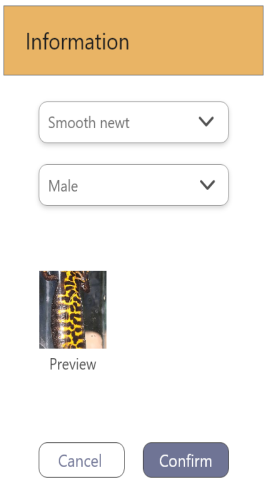


(b) Final version.

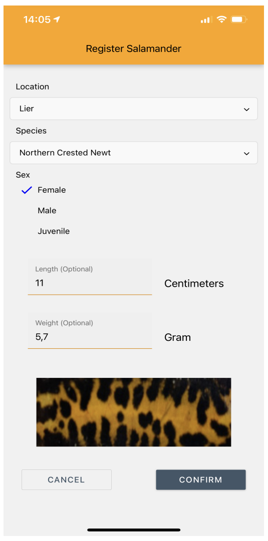
Figure 6.9: Home screen iteration from prototype to final product.

Figure 6.9 shows a comparison between the prototype we made in Adobe XD (see Figure 6.9a) and the final version of the application's home screen (see Figure 6.9b). The original plan was to have a "+" button in the top right corner that would be used to register locations. When the button was pressed the user would be able to press anywhere on the map to render a point. When the point was rendered they would have to confirm, and then they would be navigated to a new screen. There they could fill in the location name and radius to register the location.

The final version does this in a different way. For a user to register a location, they first need to long press the map to render a point. When the point is rendered, a button is shown, and the user can navigate to the register location screen. Here they can fill in location name and radius to register the location. We also added a refresh button in the final application.



(a) Prototype version.



(b) Final version.

Figure 6.10: Register salamander screen iterations.

Figure 6.10 shows a comparison between the prototype we made in Adobe XD (see Figure 6.10a) and the final version of the application's register salamander screen (see Figure 6.10b). The original plan was to only choose the sex and species for the salamander before registering it. The location would be chosen in the previous screen.

As we progressed in development, we decided it would be better to group all salamander data in one screen. We also added optional text inputs for weight and length. The user will also get a better overview of the salamander's abdominal pattern shown in Figure 6.10. The user would now be able to look at the image that the server has processed and ensure that it is not warped.

Chapter 7

Implementation

This chapter is divided into front end implementation and back end implementation. The back end implementation is again divided into a section about the REST API and the algorithm.

7.1 Front End

The front end code is written using the React Native framework and JavaScript. We will cover the most important modules implemented in the application and what they contain, as well as how navigation is implemented. We will also showcase how networking is handled, as well as state management. The code examples are excerpts from the source code, and some are shortened for example purposes. For a more detailed look into the source code, see the GitLab-repository in Appendix D.

7.1.1 Modules

We have implemented three main modules in our application; Home, Camera and Profile. Each module contains multiple screens and different functionalities that are linked with the respective module.

Home

The home screen is the first screen users see when they log into the application. In this module, users can see all registered locations, register new locations and edit locations. If users give the application permission to use their location, this will also be displayed on the map.

To add the map feature into our application, we imported the React Native Maps library[1].

[1]see, `https://github.com/react-native-maps/react-native-maps` date: 5.12.21

Camera

The second main module is the camera module. This module covers the ability to upload an image or take a picture, as well as registering and matching a salamander.

We imported two libraries into this module: expo-image picker and expo-camera. The image picker lets users pick and upload images from their local storage, and expo camera lets the application utilize the phone's camera to take a picture.

Profile

The profile module covers all user administrative features, such as the ability to change name, email and password, and deletion of the account. All operations, except for changing name, are password protected. This is an extra security implementation to protect administrative features in case an administrator's phone gets stolen.

In addition to the features above, admins will have the ability to manage users access rights and salamanders data.

7.1.2 Components

React Native is based on the React-framework, and the premise is to use components to make the code reusable, easy to read and maintainable [54]. We mostly imported GUI-components from the library React Native Paper. By doing this, we did not need to make all the GUI-components ourselves, as we could just import them from the library. We did, however, make separate components for larger components, like expo-camera. We also made components for functionality that would be used in multiple files, like the drop-down menu.

There are two ways to make components in React Native, class components and functional components. Class components have been used for a long time, and a lot of documentation and examples online use this. However, functional components have less boilerplate code, and most updated tutorials use this way of building components. The official React Native docs[2] also encourages new projects to use functional components, where they note that this is the future-facing way of using React [55]. This is why we decided to only use functional components in this project.

Code listing 7.1 shows an example of how we implemented our components. This component renders the screen where users can edit their profile name.

[2]see, `https://reactnative.dev/docs/getting-started` date: 5.12.21

Code listing 7.1: React Native Component example

```
const ChangeNameScreen = (props) => {

return (
    <SafeAreaView>
        <CustomActivityIndicator
            ...
        />
        <AccountData
            currentLabel="Current Name"
            ...
        />
        <TextInput
            mode="flat"
            ...
        />
        <CustomButton
            title="Confirm"
            ...
        />
    </SafeAreaView>
);

export { ChangeNameScreen };
```

The component consists of a *SafeAreaView*, which is a built in React Native component that makes sure the content is rendered within the safe area boundaries of a device [56]. This component wraps around our four custom components; the *CustomActivityIndicator*, *AccountData*, *TextInput* and *CustomButton*. The *CustomActivityIndicator* only appears on the screen if something is being processed. *AccountData* is a component that shows a user's current data, *TextInput* is a component that handles user input and *CustomButton* is a component that renders a button.

The content inside the components, for example "currentLabel", are called properties, and are passed down to the custom component. This is how we can reuse the component elsewhere in our application and customize it the way we want.

7.1.3 Navigation

We used the React Navigation library, and followed the official documentation when implementing navigation in the application. Code listing 7.2 shows how we implemented a stack navigation for the authentication screens.

Code listing 7.2: Stack navigation example

```
import React from "react";
import { createStackNavigator } from "@react-navigation/stack";
import { SignInScreen } from "../screens/SignInScreen";
import SignUpScreen from "../screens/SignUpScreen";

// Creating the navigator
const AuthStack = createStackNavigator();

const AuthStackScreen = () => (
  <AuthStack.Navigator>
    <AuthStack.Screen
      name="SignIn"
      component={SignInScreen}
    />
    <AuthStack.Screen
      name="SignUp"
      component={SignUpScreen}
    />
  </AuthStack.Navigator>
);
```

A stack navigator provides a way for an application to transition between screens where each new screen is placed on top of a stack [57]. We first defined the authentication stack, by calling a function from the React Navigation library. This stack contains the components SignInScreen and SignUpScreen. Like all components in our React Native application, this AuthStackScreen-component is exported and can be used elsewhere in the application.

In addition to the stack screens, we also implemented a bottom tab navigator. This allows users to switch between the home screen, camera screen and profile screen using a bottom tab bar.

7.1.4 Networking

Code listing 7.3: Axios instance

```
import axios from "axios";

// Create axios client, pre-configured with baseURL
let APIKit = axios.create({
  baseURL: "http://0.0.0.0:5000",
  timeout: 50000,
  headers: { "Content-Type": "multipart/form-data" },
});
```

As mentioned in Chapter 4, we used the library Axios for networking. As seen in Code listing 7.3, we implemented an instance of Axios in a variable called "APIKit".

The baseurl defines the address which will receive the requests. If the request goes on for more than 50 seconds, the request will time out. The instance also contains a set header, that allows the application to send forms and images. This variable is imported and used wherever there is a need for an API call.

When a user logs in successfully, a JSON Web Token will be generated by the REST API and returned back to the user. This token will be placed in the header of the Axios instance (see Code listing 7.4), and will grant the user access to other endpoints.

Code listing 7.4: Access Token.

```
APIKit.defaults.headers.common[
        "Authorization"
    ] = ‘Bearer ${response.data.access_token}‘;
```

Code listing 7.5 shows an example of a GET request. It shows how to access endpoints correctly, how response from the server is parsed, and applicable error handling.

Code listing 7.5: GET request example.

```
1  const getPendingUsers = () => {
2      setShowIndicator(true);
3
4      APIKit.get("/pendingUsers")
5        .then(function (response) {
6          setShowIndicator(false);
7
8          if (response.data.status === 200) {
9            setPendingUsers(response.data.users);
10         } else {
11           Toast.show(...);
12         }
13       })
14       .catch(function (response) {
15         setShowIndicator(false);
16         Toast.show(...);
17     });
18 };
```

When the function is triggered, the activity indicator is set to true, and will appear on the screen. To send the request to the correct endpoint, the get-function, on line 4, takes a string parameter. This parameter is appended to the baseurl of the Axios instance. When the server returns a response, the activity indicator will disappear. If it returns a 200 status response, the data is extracted into the "pending users" array as seen on line 9. If something goes wrong, an error message is shown in the form of a toast.

7.1.5 State Management

For state management we used *useState* from React, which is how functional components handles states. A state functions like a variable that triggers any component it is used in, whenever it receives a new value. As components were rendered on load, we had some difficulties when we wanted data to update dynamically across multiple screens. This is because states are intrinsic to their own component and cannot be passed to outside components. If we navigate back to a previously used component from another screen, it will not be rendered again, because it is already in memory. To handle this problem, we decided to use a third-party library.

At first, we looked into using AsyncStorage, and tried to dynamically update the data in the application. We quickly ran into problems, as the purpose of AsyncStorage is not to handle states, but rather to store data locally, causing us to scrap the idea.

After further research, Redux seemed like the best option. For Redux to work properly, we had to create actions, an object describing what happened, and a dispatch to update the redux store. The whole global state of the application is stored in an object tree inside a single store. Code listing 7.6 shows an action that dispatches the action to update the store.

Code listing 7.6: Redux example part 1. actions

```
/**
 * To edit the users name in the application and to change the state.
 *
 * @param {*} props conatains the name to be edited
 * @returns async dispatch from redux
 */
export const onEditName = (props) => {
  return async (dispatch) => {
    try {
      dispatch({ type: EDIT_NAME, payload: props.name });
    } catch (error) {
      dispatch({ type: ON_ERROR, payload: error });
    }
  };
};
```

This action receives the updated name from the user, and sends it to the reducer to update the global state. In the reducer, seen in Code listing 7.7, the updated state will overwrite the current state.

Code listing 7.7: Redux example part 2. reducer function.

```
case EDIT_NAME:
            return {
                ...state,
                user: {
                    ...state.user,
                    name: action.payload
                }
            };
```

This is an excerpt from the switch in the reducer, which has cases for each action. By using the spread-operator ("..."), the other states in the user-object will be kept, and only the specified state will be overwritten with the payload.

7.2 Back End

The back end section will cover the REST API and the image recognition algorithm for processing images and matching them.

7.2.1 REST API

This section will explain how the REST API for our system was implemented. It will include endpoint registration, user authentication, how files are returned to the client, rate limiting and the database.

Endpoint Registration

When implementing the REST API, we wanted to make it modular so that it would be easy to add more functionality as the project grows. By structuring the project, as the file hierarchy in Figure 4.5 shows, it is easy to add more endpoints, alter the database and adding different form validators. When adding a new endpoint, all the developer has to do is to create a new file in the endpoint folder, and register the endpoint in the __init__.py file, shown on line 10 and 17 in Code listing 7.8.

Code listing 7.8: __init__.py

```
from flask import Flask
from flask_restful import Api
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SECRET_KEY'] = 'randomgeneratedstring'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database/database.
    db'
api = Api(app)

from api.endpoints.login import Login
from api.endpoints.user import UserEndpoint
from api.endpoints.location import LocationEndpoint
from api.endpoints.salamanderclass import SalamanderClass
from api.endpoints.matchsalamander import MatchSalamander
from api.endpoints.findsalamanderinfo import FindSalamanderInfo

api.add_resource(Login, "/login")
api.add_resource(UserEndpoint, "/user")
api.add_resource(LocationEndpoint, "/location")
api.add_resource(SalamanderClass, "/editSalamander")
api.add_resource(MatchSalamander, "/matchSalamander")
api.add_resource(FindSalamanderInfo, "/findSalamanderInfo")
```

A file named run.py, located in the same folder hierarchy as the api folder, is used to run the API in development mode during development. Code listing 7.9 show how the app variable is imported from the __init__.py file and run in debug mode with a specific IP address. The default port that the Flask server runs the REST API on is 5000.

Code listing 7.9: Run.py

```
1  from api import app
2
3  if __name__ == "__main__":
4      app.run(debug=True, host='0.0.0.0')
```

User Authentication

User authentication is handled by using Flask-Bcrypt, combined with Flask-JWT-Extended. Flask-Bcrypt has two main functionalities that we used; hashing and salting the passwords, and comparing the hash to a string. In Code listing 7.10, the hash is generated before storing it in the database. After the hashed password is stored and the user is approved by an admin, the user can log in to the system. In Code listing 7.11 on line 2, we compare the hashed password against the password the user entered. This function will either return true if the passwords match, or false if they do not.

Code listing 7.10: Create user

```
1  password_hash = bcrypt.generate_password_hash(data.password)
```

When a user successfully types their email and password correctly and logs into the system, a JSON Web Token (JWT) will be generated, seen in Code listing 7.11 on line 4. The JWT that is generated also encrypts the user's ID. This token will be sent back to the user that requested the login, and will later be used to access all the other endpoints. In the mobile application the JWT that is returned will be placed in the header of each request and thereby granting users access to protected endpoints.

Code listing 7.11: Login.py

```
1  user = db.session.query(User).filter_by(email=data['email'].lower()).
       first()
2  if user and bcrypt.check_password_hash(user.pwd, data['password']):
3      if user.accepted:
4          ret = {
5                  'access_token': create_access_token(identity=user.id),
6                  'message': "successfully logged in",
7                  'admin': user.admin,
8                  'name': user.name,
9                  'email': user.email,
10                 'status': 200
11                 }
12             return jsonify(ret)
```

By using a simple decorator, provided by Flask-JWT-Extended, we can choose which endpoints that requires a JWT. Line 1 in Code listing 7.12 shows how this is implemented in one of the endpoints. The last feature the REST API uses from Flask-JWT-Extended is the get_jwt_identity() function. When a user requests an endpoint with a JWT, the function on line 3 in Code listing 7.12 decodes the user's ID from the JWT. This allows the system to verify users without the need for re-authentication. If users try to request an endpoint without the JWT, the server will respond with a 401 HTTP status code, and they will not be given access.

Code listing 7.12: JSON Web Token

```
1  @jwt_required
2  def get():
3      user_id = get_jwt_identity()
4      user = db.session.query(User).filter_by(id=user_id).first()
```

To further visualize how the authentication is handled in the system, see Figure 7.1.

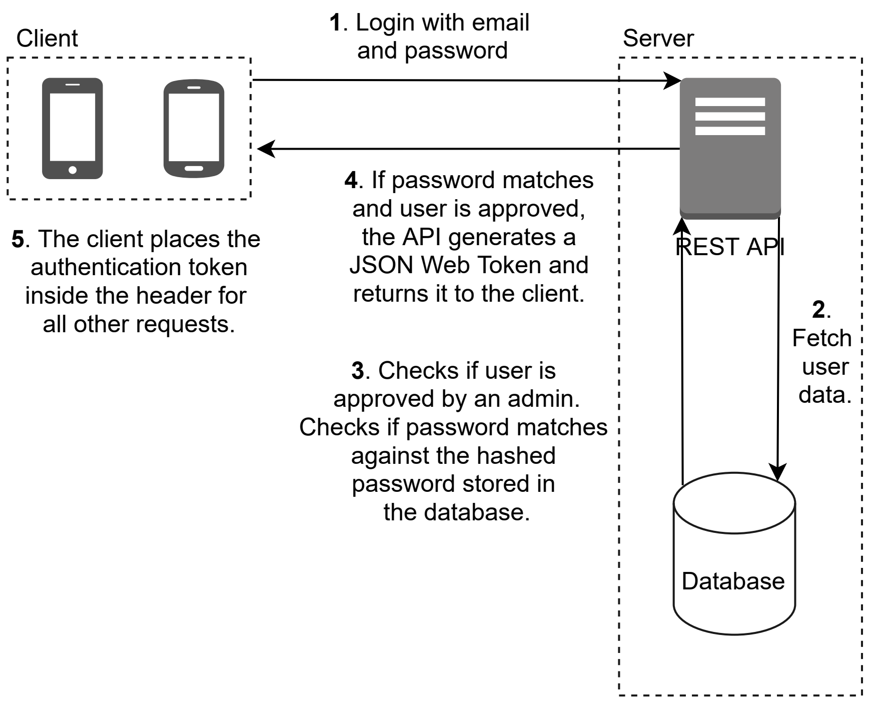

Client
1. Login with email and password
Server
4. If password matches and user is approved, the API generates a JSON Web Token and returns it to the client.
5. The client places the authentication token inside the header for all other requests.
REST API
2. Fetch user data.
3. Checks if user is approved by an admin. Checks if password matches against the hashed password stored in the database.
Database

Figure 7.1: Authentication diagram. The diagram should be read following the numbers in ascending order.

Image Encoder

During the salamander registration, the REST API will return the processed version of the originally uploaded image. This lets the user verify the quality of the processed image before submitting it to the matching algorithm. At first, we tried to use Flask's built in send_file() function. Even after testing and a deeper look into the documentation, we still struggled to get this to work when sending the file to the mobile application.

The send_file() function was then replaced with an image encoder instead. The encode() function on line 4 in Code listing 7.13 encodes the image to a Base64 string. This string could now be returned back to the client in a JSON object.

Code listing 7.13: Encode Images

```python
1  for path in list_of_paths:
2      basename = os.path.basename(path)
3      if basename.__contains__(str(image_id)):
4          encoded = encode(path)[2:-1]
5          image_data = {"url": "data:image/png;base64," + encoded}
6          salamander_images.append(image_data)
```

Rate Limiter

To improve security and performance of the REST API, we implemented a rate limiter. This is to prevent DOS attacks, and attempts to brute force passwords [58]. The library that handles the rate limiter is called Flask-Limiter. Code listing 7.14 shows an example of how the limiter is implemented and how each endpoint can have different rate limits. It is important to make sure the rate limit is not too low as being too strict can adversely affect the user experience. Also, by implementing a limiter, the system makes sure that no user can occupy the server's resources for too long.

Code listing 7.14: Python code example limiter

```python
decorators = [limiter.limit("60/minute")]
```

Database

The database was implemented by using the Flask-SQLalchemy library with SQLite dialect. Code listing 7.15 shows the implementation of the salamander table, also seen in Figure 4.7. When a new salamander is registered, the ID auto increments, and the date gets auto generated. The uid is a foreign key to the user that registered the salamander and the location_id is a foreign key to the location where the salamander is registered.

Database.db is the file that keeps the stored data about salamanders, users, locations and the salamander growth. It is located in the database folder, as shown in Figure 4.5. For storing all the images we use the folder structure as seen in Figure 4.6.

Code listing 7.15: Salamander Table

```python
class Salamander(db.Model):
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    sex = db.Column(db.String(255), nullable=False)
    species = db.Column(db.String(255), nullable=False)
    location_id = db.Column(db.Integer, db.ForeignKey('location.id'),
        nullable=False)
    uid = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=
        False)
    first_date = db.Column(db.DateTime, default=datetime.now)
```

7.2.2 Algorithm

The code from the previous bachelor's project, that identifies the salamander, needed significant changes to work properly. After looking into the previous source code, we realized that it is actually a collection of several individual scripts that could be run separately from the command line. From these scripts we kept the code that straightens an image and the code that matches two images.

Matching Process

It is important to mention that the matching algorithm requires a fixed image size to compare two straightened images. This means that all the straightened images have to be the same size. To straighten the abdomen in an image, the straightening algorithm needs to know how wide the salamander is in pixels. This width is a float, even though we are working with pixels, because it is only used as a factor for scaling the original image. We also need to know the position of the spine of the salamander. This will be represented by four evenly spaced out points along the salamander from the shoulders to the pelvis (see Figure 7.5b). These points will be generated by the model from DeepLabCut after feeding it an image.

These key points function as indicators, which are used by the straightening algorithm to interpolate a line down the spine to straighten the image. A single point on each shoulder of the salamander would be used to calculate the width of the salamander and how much the image needs to be scaled to cover the required fixed image ratio.

Once the image is straightened, it is returned back to the user. The processing will not continue if the user does not approve the straightened image via the mobile application, and ensures that it is not warped. If the user does not approve the image, a new one will have to be uploaded. When approving an image, the user also specifies how to match the salamander based on location, species, and sex. The matching algorithm will traverse a list of folders, each representing a salamander, and brute force match the incoming image using functionality provided by OpenCV[3]. If there is no match, a new salamander will be registered in the database. A folder will be created containing a copy of the original and processed image.

Dataset

To train a model we need a large set of sample images of salamanders, which we had to manually label using DeepLabCut's GUI based labeling tool. This tool lets us easily label the six key points we previously described in Section 7.2.2. We had to label the images twice, because after training the first model, we chose to add two shoulder points for estimating the width of the salamander. This was done to make sure the final cropped image containing the pattern would have as little background showing as possible in order to hopefully improve the matching accuracy. These points then got written to a CSV file, which made them ready to be used for training. Code listing 7.16 shows how the labeled points were saved to the file. To increase the size and variety of the training set, DeepLabCut uses Imgaug [59] to rotate some of the existing images. We do not know exactly how much larger the dataset became after augmentation, as DeepLabCut uses Imgaug

[3]see `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html` date: 5.12.21

by default. However, after looking through DeepLabCut's source code, we can estimate the total amount of images to be 2,688 original images and 1,344 rotated images.

Code listing 7.16: DeepLabCut CSV exerpt

```
bodyparts bodypart1 bodypart1 ...   shoulder_R  shoulder_R
coords  x y ...   x   y
labeled-data\1\img0.png  409.66   116.23   ...   306.45   138.19
labeled-data\1\img1.png  433.82   100.86   ...   306.45   129.41
labeled-data\1\img10.png 501.89   632.28   ...   398.68   568.60
```

Later in development, we also made a training set for object detection. We separated images based on male, female, and juvenile into three folders. There were in total 1,450 males, 1,076 females, 26 juveniles, and 139 with unidentifiable sex. To draw and store the bounding boxes (seen in Figure 7.9) we used Labelimg[4] which saved the labeling data into separate XML files.

Our contact person at NINA provided us with a large collection of 2,688 sample images of northern crested newts from 2019. It is important to note that there would never be more than one salamander per image. Not every image would contain a salamander in frame, and some images would not show the entire salamander.

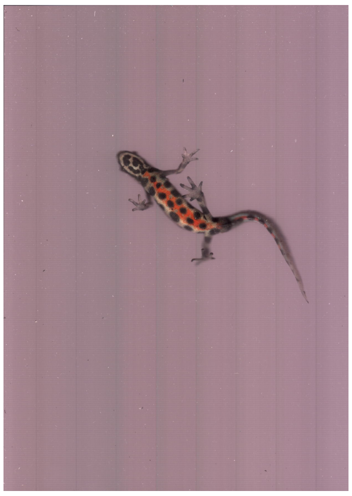


Figure 7.2: A sample from the older collection of smooth newt images.

Since we wanted a larger dataset of greater variety, our supervisor at NTNU provided us with an older collection of images of smooth newts taken using a passport scanner. Figure 7.2 shows an example of one of these images. The background and framing on these images differed heavily from the newer images in the collection

[4]see, `https://github.com/tzutalin/labelImg` date: 5.12.21

we got from our product owner.

At this point, we had two separate datasets. Due to them having little intrinsic variety, being of different species, and captured in completely different ways, we concluded to not use both of them in a single model. The neural network might locate the points or estimate species based on these factors, rather than the salamander itself. Therefore, we chose to not use the older set of smooth newts, because NINA no longer captures images with passport scanners. A consequence of this is that we loose the ability to determine species, as we are left with only the set containing northern crested newts.

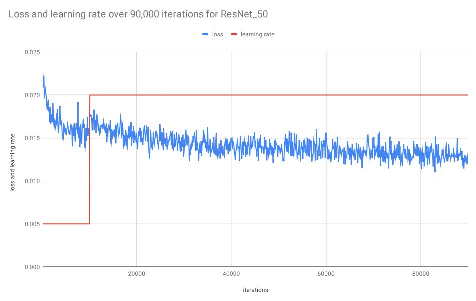
It is important to note that the exact impact of using both datasets for point estimation is not clear to us. Labeling, training and validating were quite time consuming, and we did not want to risk spending too much time on this issue. As the researchers at NINA knows the difference between species, we prioritized other more important requirements. A new model will have to be trained on a larger and more varied dataset if NINA wants it to perform better, or be able to recognize multiple species.

Models and Performance

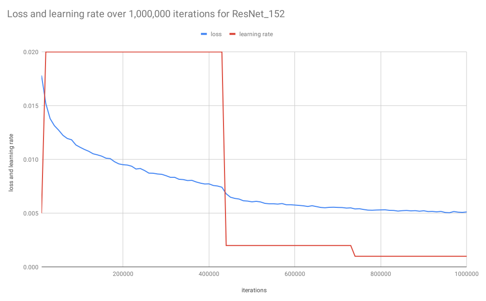
Training accurate models can take several days. It all depends on the complexity of the model, the hardware used, and the size of the dataset used for training. At least one strong GPU is needed to speed up the process. We specifically used DeepLabCut-core for training models and for estimating an image in the algorithm, because it supports the latest version of Python and Tensorflow. This was important as the GUI version of DeepLabCut used an outdated version of Tensorflow, which did not support the latest 30-series GPUs from Nvidia.

We trained several models, and at first we trained our models on an Nvidia RTX 2080 super. The final model was trained on an Nvidia RTX 3080. Tensorflow and DeepLabCut allows for dividable and continuous training, allowing us not having to train the complete model in one sitting. The longest training session took around 48 hours and trained a model for one million iterations.

Figure 7.3 shows the loss and the learning rate over iterations for our *ResNet_50* and *ResNet_152* models. The blue line represents loss, while the red line represents learning rate. Loss is a value that represents how poorly the model performed (lower is better). In Figure 7.3 it displays the average loss over 100 and 10,000 iterations. The learning rate is a value that affects how much the model should change its weights between each iteration. Overlapping loss and learning rate gives a good visualization of how learning rate impacts loss.



(a) ResNet_50 loss and learning rate over 90,000 iterations.



(b) ResNet_152 loss and learning rate over 1,000,000 iterations.

Figure 7.3: Loss and learning rate over iterations.

In Figure 7.4 the loss series from both models is overlayed. The loss from the *ResNet_50* model looks quite erratic compared to the *ResNet_152* model. It is important to note that the sample rate for the *ResNet_50* model is every 100 iterations, compared to 10,000 iterations for the *ResNet_152* model. Still, by taking into account the higher sample rate from *ResNet_50*, its loss still seems quite dispersed compared to the *ResNet_152* model. Despite this, both models follow a similar median slope. Ideally the loss should asymptotically flatten out at a specific value. Once the loss hits this "plateau" it is not usually needed to train further [60].

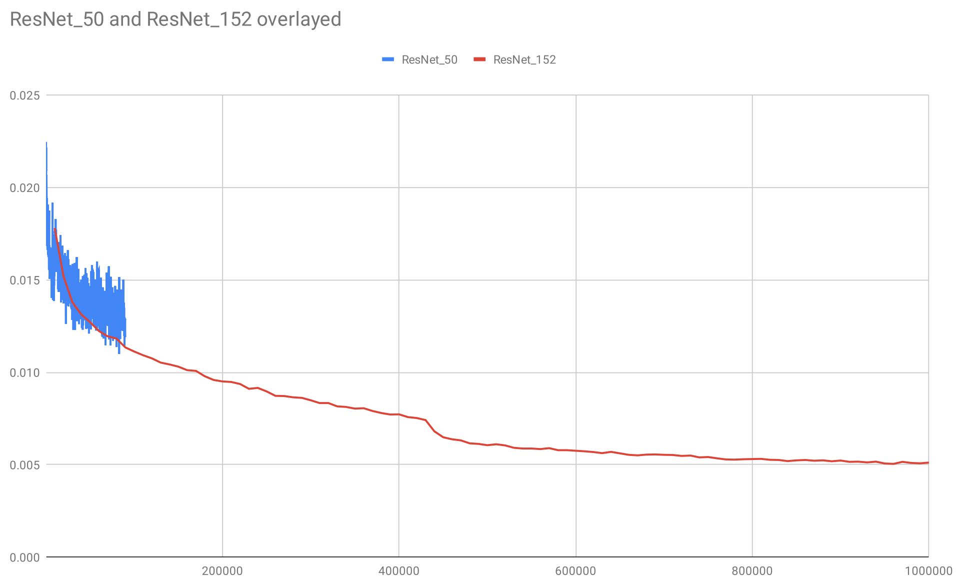


Figure 7.4: Loss and learning rate overlayed.

We trained various types of models to find the one with the best possible accuracy. At first we trained a *ResNet_50* model with 30 000 iterations. It did not perform well on most of the test images. The model continued training for another 20,000 iterations, but its performance still did not meet satisfaction. We moved on to a *ResNet_101* model with 100,000 iterations, and although it performed a lot better, some images were still too hard to estimate.

We were hesitant of using a deeper model because we did not want it to slow down the straightening process too much. That proved to be an unnecessary concern, because one estimation never took more than ten seconds. At that point we did not see any reason not to move onto *ResNet_152*. It also proved to significantly improve accuracy with only 50,000 iterations. For our final training session we decided to train the model for one million iterations.

As seen in Figure 7.5, the model has significantly improved estimating on that specific image after an increase in iterations and being upgraded to a deeper network. However, testing a models performance is not as straight forward. To say that a model is 90% accurate, does not tell us much about how the model performs. One could give a well trained model similar images to its training set, and it would probably score high. However, if the model receives an image which is significantly different from its training data, it will most likely score badly. Thus, highlighting the importance of a varied training set.

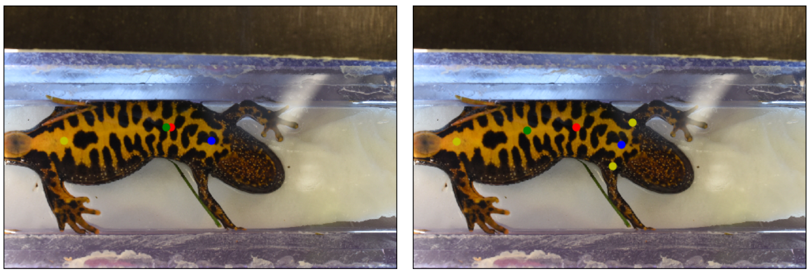
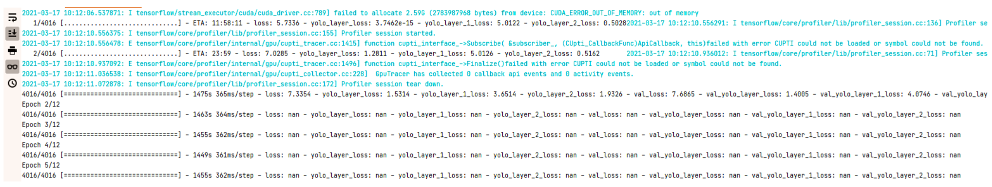

(a) ResNet_50 after 90K iterations. **(b)** ResNet_152 after 1M iterations.

Figure 7.5: A comparison between models with different amount of training.

A colored image is a large list of numbers, where each pixel is represented by three or four numbers. By looking at a single pixel, it is impossible to tell the context of that pixel. This is why a traditional simple neural network would typically not perform well on object recognition or determining points [61]. They are simply not complex enough, but due to how today's neural networks are designed to use backwards propagation to update each of the neurons weights, one cannot just increase the depth of a model in layers without consequences. Doing so can result in a vanishing gradient or an exploding gradient [62].

Vanishing or exploding gradients are problems encountered in the training of neural networks. These problems happen when the updates to the weights of a model under training either changes by exponentially increasing amounts, exploding gradient, or decreasing amounts, vanishing gradient. An exploding gradient will typically lead to an unstable model incapable of effective learning [63]. A vanishing gradient will result in a model that is incapable of further learning as its loss has stagnated [64].

If we were looking at an image of 2x2 pixels, a simple neural network would work fine, because there are only four dependent variables. Since most images are quite large in size, these problems become more apparent. As seen in Figure 7.6, we likely experienced a case of exploding gradients in a YoloV3 model, that was setup to locate the cloaca of the salamander. After looking at the results of a training session, all the weights had become NaN.

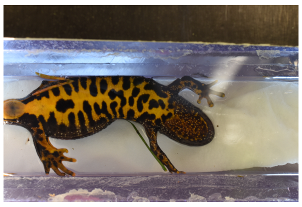


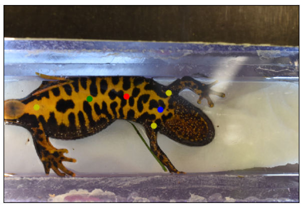
```
2021-03-17 10:12:06.537871: I tensorflow/stream_executor/cuda/cuda_driver.cc:789] failed to allocate 2.59G (2783987968 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
   1/4016 [..............................] - ETA: 11:58:11 - loss: 5.7336 - yolo_layer_loss: 3.7462e-15 - yolo_layer_1_loss: 5.0122 - yolo_layer_2_loss: 0.50282021-03-17 10:12:10.556291: I tensorflow/core/profiler/lib/profiler_session.cc:136] Profiler se
2021-03-17 10:12:10.556375: I tensorflow/core/profiler/lib/profiler_session.cc:155] Profiler session started.
2021-03-17 10:12:10.556478: E tensorflow/core/profiler/internal/gpu/cupti_tracer.cc:1415] function cupti_interface_->Subscribe( &subscriber_, (CUpti_CallbackFunc)ApiCallback, this)failed with error CUPTI could not be loaded or symbol could not be found.
   2/4016 [..............................] - ETA: 23:59 - loss: 7.0285 - yolo_layer_loss: 1.2811 - yolo_layer_1_loss: 5.0126 - yolo_layer_2_loss: 0.5162       2021-03-17 10:12:10.936012: I tensorflow/core/profiler/lib/profiler_session.cc:71] Profiler ses
2021-03-17 10:12:10.937092: E tensorflow/core/profiler/internal/gpu/cupti_tracer.cc:1496] function cupti_interface_->Finalize()failed with error CUPTI could not be loaded or symbol could not be found.
2021-03-17 10:12:11.036538: I tensorflow/core/profiler/internal/gpu/cupti_collector.cc:228]  GpuTracer has collected 0 callback api events and 0 activity events.
2021-03-17 10:12:11.072878: I tensorflow/core/profiler/lib/profiler_session.cc:172] Profiler session tear down.
4016/4016 [==============================] - 1475s 365ms/step - loss: 7.3354 - yolo_layer_loss: 1.5314 - yolo_layer_1_loss: 3.6514 - yolo_layer_2_loss: 1.9326 - val_loss: 7.6865 - val_yolo_layer_loss: 1.4005 - val_yolo_layer_1_loss: 4.0746 - val_yolo_lay
Epoch 2/12
4016/4016 [==============================] - 1463s 364ms/step - loss: nan - yolo_layer_loss: nan - yolo_layer_1_loss: nan - yolo_layer_2_loss: nan - val_loss: nan - val_yolo_layer_loss: nan - val_yolo_layer_1_loss: nan - val_yolo_layer_2_loss: nan
Epoch 3/12
4016/4016 [==============================] - 1455s 362ms/step - loss: nan - yolo_layer_loss: nan - yolo_layer_1_loss: nan - yolo_layer_2_loss: nan - val_loss: nan - val_yolo_layer_loss: nan - val_yolo_layer_1_loss: nan - val_yolo_layer_2_loss: nan
Epoch 4/12
4016/4016 [==============================] - 1449s 361ms/step - loss: nan - yolo_layer_loss: nan - yolo_layer_1_loss: nan - yolo_layer_2_loss: nan - val_loss: nan - val_yolo_layer_loss: nan - val_yolo_layer_1_loss: nan - val_yolo_layer_2_loss: nan
Epoch 5/12
4016/4016 [==============================] - 1455s 362ms/step - loss: nan - yolo_layer_loss: nan - yolo_layer_1_loss: nan - yolo_layer_2_loss: nan - val_loss: nan - val_yolo_layer_loss: nan - val_yolo_layer_1_loss: nan - val_yolo_layer_2_loss: nan
```

Figure 7.6: Possibly a case of exploding gradients

Today's feed-forward neural networks are not flawless, but there are workarounds to these problems. ResNet for example, use convolutional layers. Each convolution layer have neurons where their weights work more like filters [65]. The previous layer sends the image(s) to each filter. A filter can be looked at like a matrix that convolve over the image with a given stride and returns the dot product between that filter and the part of the image it is comparing to. These filters will give suggestions to where the correct spots lay based on an evolving pattern they are looking for.

ResNet also has residual blocks that works as a way to sum up the weights from a higher layer to a lower layer in the network [66]. Residual blocks help against vanishing or exploding gradients, because they use shortcuts and identity mappings, while propagating the gradient back up the network. A residual neural network can therefore send gradients long distances depending on how the residual blocks are made, and hopefully maintain a more reasonable gradient. This allows the earlier layers to learn faster [67].


(a) Unprocessed image of a Northern Crested Newt


(b) Image with labels from DeepLabCut

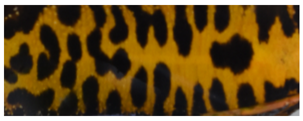
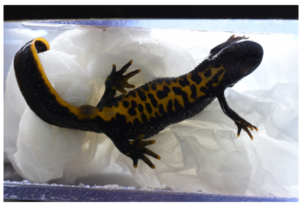
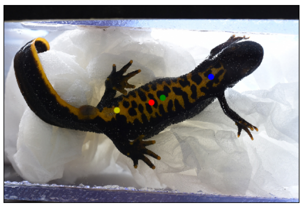

(c) Straightened and isolated abdominal pattern

Figure 7.7: Order of straightening process.


(a) Unprocessed image of a Northern Crested Newt


(b) Image with labels from DeepLabCut

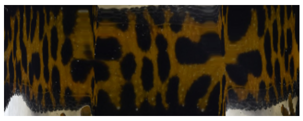

(c) Straightened and isolated abdominal pattern

Figure 7.8: Failed straightening process.

Figure 7.7 shows the different stages a raw unprocessed image goes through when the algorithm runs. Figure 7.7a displays an untreated image. Figure 7.7b shows the same image, but with the points from DeepLabCut overlaid. Here the four midline points, and the shoulder points are visible. These points are then used to create a cropped image containing a straightened version of the abdominal pattern, seen in Figure 7.7c. In Section 7.2.2, it is explained how the points are used. When matching two abdominal patterns the images are converted to gray scale. Figure 7.8 shows how an older model failed to straighten an image. The points were misplaced, causing the final pattern to be warped.

Incorporating the AI into the REST API

Several challenges occurred while trying to incorporate the algorithm into the REST API. One of the problems was that multiple users may try to process an image at once. We had to make sure that the server would not try to start several Tensorflow sessions at once while estimating. This is necessary because the server only has one GPU, and it can only handle one estimation at a time without running out of VRAM. Therefore, a semaphore was implemented to prevent this (see Code listing 7.17). Any thread that tries to access the straightening function while it is in use, will be blocked until the semaphore is freed. If the system were to be further developed to support several estimations at once, a system that assigns and frees several GPUs would be necessary.

Code listing 7.17: Semaphore Example

```python
def straighten(image):
  with _ACCESS_TF_AND_GPU_SEMA:
      # straightening...
```

Brute Force Matching

After an incoming image has been processed by DeepLabCut, and a straightened image is produced, it has to be matched against existing images. This is done using a brute force image matcher provided by OpenCV. Code listing 7.18 shows an example of this image matcher being used. The first two variables define the minimum requirements for a match to occur. When the brute force image matcher runs, it makes several comparisons, each with slightly different parameters. In our case the minimum number of successful matches is 15, which we found to be strict enough to avoid false positives, but not so strict to where the same pattern with differing contrast, lighting etc. would not match. The second value indicates how similar the images have to be in order to be considered the same. This value was set by the previous group from 2019, and since it gave good results we decided not to alter it.

The match_single_image() function on line 3, in Code listing 7.18, shows how we prepare the two images for matching by generating all the necessary data needed for the brute force matcher. After the match results have been generated, they are iterated through in a for loop, where they are validated against the minimum requirements. Finally, the function returns true or false depending on the final result.

Code listing 7.18: Match single image method

```python
 1  min_good_match = 15
 2  match_dist = 0.75
 3  def match_single_image(input_image, match_image) -> bool:
 4      bf = cv2.BFMatcher()
 5      input_salamander = create_salamander_image(input_image)
 6      match_salamander = create_salamander_image(match_image)
 7
 8      match = bf.knnMatch(input_salamander.descriptors, match_salamander
            .descriptors, k=2)
 9
10      goodmatch = []
11      for m, n in match:
12          if m.distance < match_dist * n.distance:
13              goodmatch.append([m])
14      if len(goodmatch) > min_good_match:
15          return True, len(goodmatch)
16      return False, 0
```

Code listing 7.19 shows how the prior function is used on images from several directories. The function is given an input image and a directory to iterate through. This directory would contain other directories containing salamanders of a particular species and sex, from the same location. The matching function would then be run against each straightened image inside the directories, storing the score and ID of any image which passes the matching requirements. If an image gets a better score than the previous best, its ID and score overrides the previous. When every image has been checked, the ID of the highest scoring image is returned. If no match is found the function returns -1, since no salamander can have a negative ID, to indicate that no match was found.

Code listing 7.19: Match over multiple directories

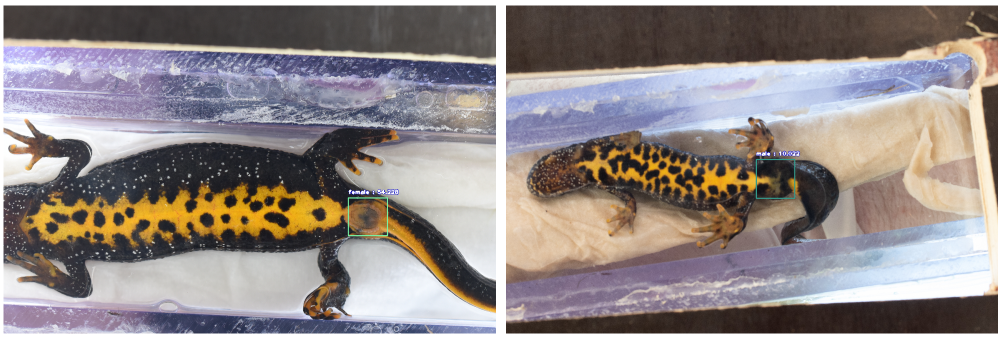
```python
def match_file_structure(input_image: str, match_directory: str):
    best_match = -1
    match_count = 0

    # check if input path is valid:
    if os.path.isfile(input_image):
        for folder in os.listdir(match_directory):
            name_list = glob.glob(os.path.join(match_directory, folder
                , "*_str.*"))
            for filename in name_list:
                res, num_matches = match_single_image(input_image,
                    filename)
                if res and num_matches > match_count:
                    match_count = num_matches
                    best_match = int(folder)
        return best_match
    else:
        return None
```

7.2.3 Abandoned implementations

Looking back at the requirements in Chapter 2, we were not able to implement a system that could automatically recognize the salamanders sex and species. Since the dataset we chose were specifically of northern crested newts, we were therefore not able to train a model to recognize species.

An attempt was made to implement sex recognition. All models trained for this purpose was done with ImageAI. At first we created a training set by dividing the images into folders based on sex. We trained an InceptionV3 model for image classification, and trained it for 80,320 iterations, but it performed poorly by often leaning towards guessing only female or male. We originally wanted to use *ResNet_152*, but we had some problems using some of the models in the ImageAI's library, and resorted to using what worked. Training the InceptionV3 model for another 48,192 iterations did not seem to improve its accuracy either.



(a) Female salamander. **(b)** Male salamander.

Figure 7.9: Successful sex estimations using YoloV3.

Another attempt was to use object detection by defining bounding boxes of where the cloaca was located (see Figure 7.9). This would specify to the AI what to look for, and possibly clarify what the difference between the two sexes are. A YoloV3 model was trained and performed better with fewer iterations, but as previously explained with Figure 7.6, after 204,816 iterations the weights turned NaN because of possibly exploding gradients.

Further investigation to fix this could have been carried out, but due to the sheer amount of time it took to do one estimation we decided to discontinue this. Setting up Tensorflow to estimate the points already takes around 10-20 seconds, and it would take another 10-20 seconds to start another session for sex estimation. The fact that the researchers at NINA already knows the salamander's sex before identification, means that it would be a waste of time to force them to wait for our system to assume it.

Another challenge was Tensorflow itself. We could not find a way to make it release the GPU and its allocated memory. Also, ImageAI required a different Tensorflow session from DeepLabCut. Running both AI's together caused a clash between the Tensorflow sessions. We discovered this problem when we were still trying to merge DeepLabCut with ImageAI's sex identification into the REST API. Also, several processes will run on NINA's server and they may also want to use the GPU. We tried multiple methods and functions from the Tensorflow documentation, but none of them actually freed up the memory from the GPU.

Once Tensorflow is given a key to the GPU, it will not release that key until the process is ended [68]. Our solution was to start a new process for every estimation, as seen in Code listing 7.20. In other words, the REST API starts a new process to use the model and then returns whatever the model finds back to the REST API before terminating the process.

Code listing 7.20: Creating a new process to run estimation

```python
def get_prediction_dlc(config,image,shuffle=1,
                trainingsetindex=0,gputouse=None):
    return_dict = Manager().list()
    args = (config, image, shuffle, trainingsetindex, gputouse,
        return_dict)
    p = Process(target=run_prediction_dlc, args=args)
    p.start()
    p.join()
    return return_dict[0], return_dict[1], return_dict[2], return_dict
        [3]
```

A downside to starting a new process and Tensorflow session for each estimation is that it takes the majority of the total estimation time. In addition, if the model is stored on a hard drive, rather than on an SSD, the total time will also be longer. If we were to actually incorporate sex recognition into the REST API, managing Tensorflow sessions and GPU access would have to be taken into consideration. In our case, the system is only given a partial amount of the servers hardware resources and the program is only using one Tensorflow session. The issue of the GPU not being released is therefore not a concern.

7.3 System Flow

The mobile application, the REST API, and the algorithm were all developed independently. They are not heavily dependent on each other, but they are intended and customized to work together. A user will use the mobile application to send requests to the REST API, and the REST API will call the algorithm when doing image processing.

To visualize how the different components in the system work together, we have created a sequence diagram of how the registration of a salamander works. We chose this example as it involves all sides of the system, from the mobile application, to the REST API and algorithm. Figure 7.10 illustrates the process from when an image is uploaded from the mobile application, approved by a user, and a result is returned from the algorithm.

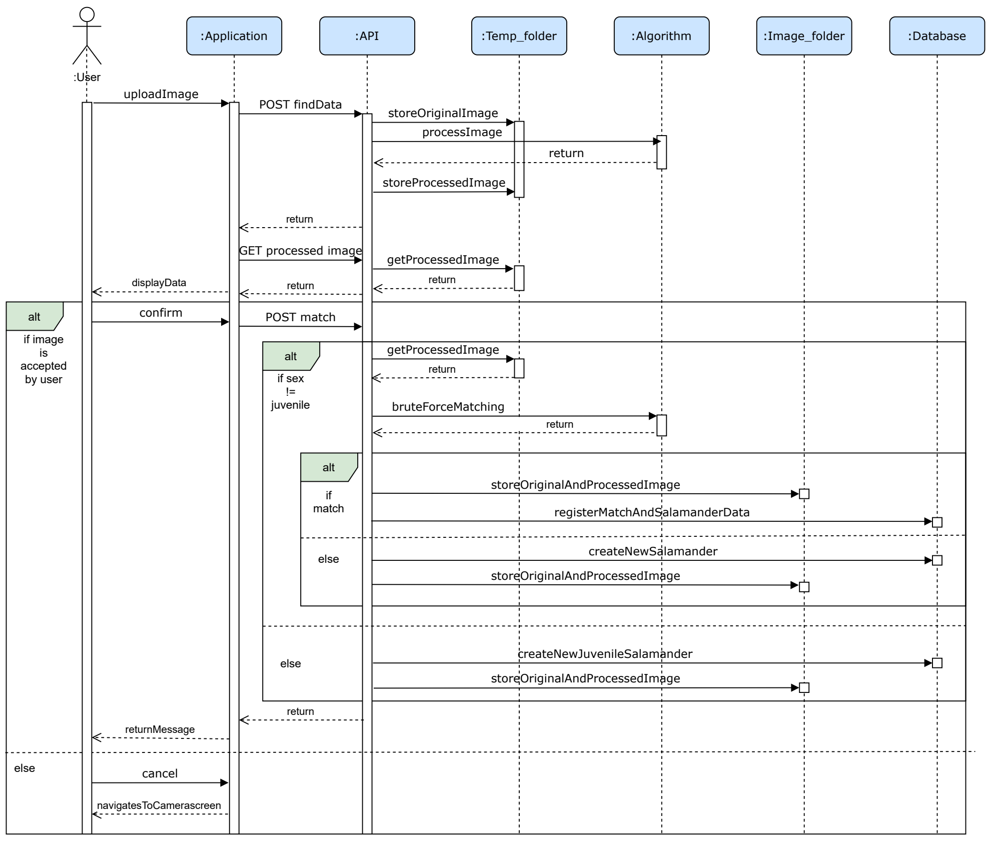


Figure 7.10: Sequence diagram of matching a salamander.

The *alt* boxes indicates alternative routes the system can take depending on certain conditions.

Chapter 8

Deployment

This chapter is divided into deployment of the mobile application and deployment of the REST API and algorithm.

8.1 Mobile Application

For the mobile application, we used Expo CLI throughout development. Expo CLI is a command line application that is the main interface between a developer and Expo tools [69]. It can be used for creating new projects, developing an application, publishing an application, building binaries, and manage Apple Credentials and Google Keystores.

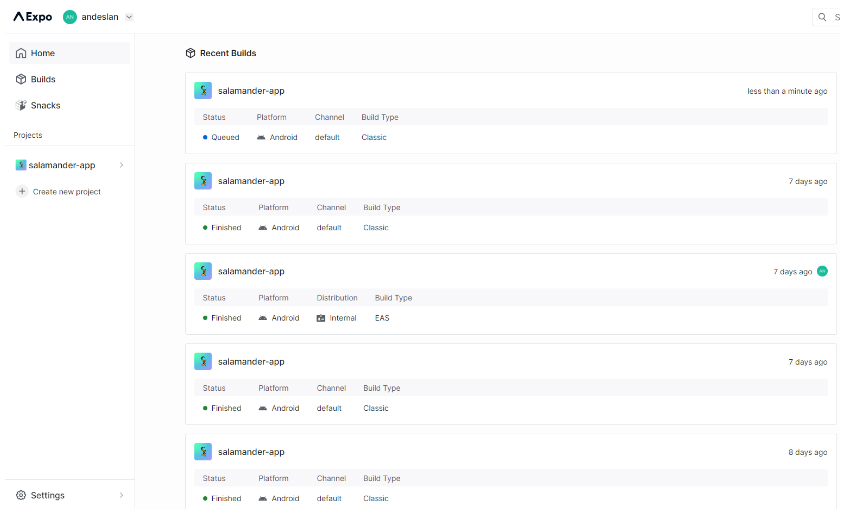
8.1.1 Development

During development, we continuously deployed our mobile application to the Expo servers, by using the command "expo start" in the terminal. We could then scan a QR-code, and test the application on either an emulator, or on our own device through the Expo Go-app. This made it easy for us to rapidly monitor any changes we did to the application.

8.1.2 Build

When building we used Expo's built-in tool. It allowed us to configure different build parameters inside a single JSON file. Using a single command, Expo can generate executable files for both Android and iOS. We focused on building for Android first, due to most of the researchers at NINA having Android phones. In addition, to build an iOS application, an Apple developer account is needed, which costs money. This will therefore not be done, unless it is financed by NINA.

We decided to use EAS Build to build our application in the beginning, as it allowed us to quickly and easily build and keep track of all builds and logs. Figure 8.1 shows how the different builds are listed based on creation date.



Recent Builds

salamander-app — less than a minute ago

Status	Platform	Channel	Build Type
Queued	Android	default	Classic

salamander-app — 7 days ago

Status	Platform	Channel	Build Type
Finished	Android	default	Classic

salamander-app — 7 days ago

Status	Platform	Distribution	Build Type
Finished	Android	Internal	EAS

salamander-app — 7 days ago

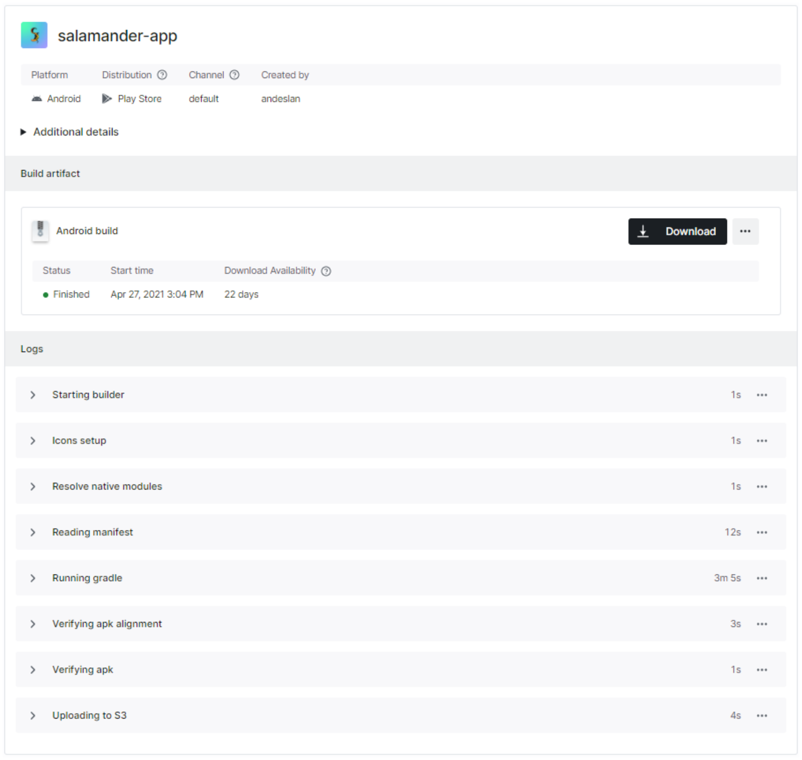
Status	Platform	Channel	Build Type
Finished	Android	default	Classic

salamander-app — 8 days ago

Status	Platform	Channel	Build Type
Finished	Android	default	Classic

Figure 8.1: Expo history.

EAS Build is a service provided by Expo which allows for building, signing, and distributing Expo applications [70]. This made it easy for us to debug any errors while building. Figure 8.2 shows how a single build is presented with its build details, runnable file, and build logs.


salamander-app

Platform	Distribution	Channel	Created by
Android	Play Store	default	andeslan

Additional details

Build artifact

Android build

Status	Start time	Download Availability
Finished	Apr 27, 2021 3:04 PM	22 days

Logs

```
Starting builder             1s
Icons setup                  1s
Resolve native modules       1s
Reading manifest             12s
Running gradle               3m 5s
Verifying apk alignment      3s
Verifying apk                1s
Uploading to S3              4s
```

Figure 8.2: Expo build.

EAS Build also generates a SHA-1 certificate fingerprint, which is needed for deploying the application to the GooglePlay Store or the App Store. In addition, this hash is important for the restriction of the Google Maps API key, as it needs to know the application package name and fingerprint. In the beginning we took advantage of EAS build's one month free trail for their premium subscription. This service gave us access to their priority builds feature, which made it easy to iterate and try several build configurations, as building with the free version can take a long time due to having to wait in a build queue. After the month had passed we went back to using the free version, using the working configuration we ended up with from earlier.

8.2 REST API and Algorithm

During development, we had close collaboration with NINA. According to our product owner, we could get access to one of their severs for deployment. However, during the end of the development period, it became clear that this would not be possible before the deadline. Therefore, we chose to deploy the back end on Openstack. In this way, we could test deployment on a similar server to NINA's.

8.2.1 OpenStack

NINA provided us with the specifications to their server. Although we did not get access to their server in time, we wanted to deploy the system to a similar server to NINA's. Thankfully, NTNU were able to provide us with access to a similar server on Openstack.

Unfortunately, the server on Openstack, running on NTNU's network, is blocked from outside access. This means that to access the server, we had to work on the Eduroam network or use a VPN. To use the application in the field, it is crucial that the server is able to receive requests from the outside.

Hardware

The server we got access to has 8 CPU cores, 90GB of RAM and has 40GB of storage on a hard drive. We also got access to 1/4th of an Nvidia Tesla V100 32GB graphics card.

Instance

Since NINA's server runs on Ubuntu 18.04 LTS, we chose to use the corresponding operating system on the Openstack server.

Security Groups

For testing the system with Flask's development server, we used port 5000. Port 80 and 443 was opened for HTTP/HTTPS requests on the production server.

Nginx and Gunicorn

For the production web server we used Nginx combined with Gunicorn to serve the REST API. Nginx is a free open source web server that is optimized for routing, handling incoming traffic, forward requests to Gunicorn, and terminating SSL. Gunicorn is a web server gateway interface, which allows for requests to be translated and makes sure that the correct piece of code is executed [71].

8.2.2 Future Deployment

In the middle of April we got contacted by NINA's IT department. They told us that the server we planned to deploy on was not open for outside access. However, they suggested that we could deploy on their server by splitting the back end on two different servers. The REST API would run on a server that is open for outside access and the algorithm would run on the internal server. The public server would then be able to forward the images for processing to the internal one.

They also suggested to create Docker containers for deploying on the servers. As we got noticed of this quite late in the project period, we decided that we did not have time to do this within the deadline. This will then become future work.

Chapter 9

Testing

This chapter is divided into testing we did during development, performance testing, and user testing.

9.1 Development Testing

During development we only did manual testing. Based on our Gantt-diagram, seen in Figure 3.1, we concluded that if we only had eight weeks to develop the entire system, it would be wiser to spend most of that time on developing, and not on making automated tests.

In retrospect, we should probably have had some form of unit testing on the REST API. Even though we would not have been able to implement as many features as we did, there were occasions where manual testing became tedious. In addition, automated testing would save us time in the long run, especially during implementation of new features. This was especially noticeable when we implemented the manage salamander feature, described in detail in Table 5.5.

Each time something crucial went wrong in the database or in the file hierarchy, it would have to be manually wiped and reset. An automated procedure that wiped and reset the database with new entries, would have saved us a lot of time. In addition, an automated procedure that sends requests to the REST API could be run after the reset. Having such a procedure would have saved time rather than doing it manually.

If we were to do this process all over again, unit testing for the back end is absolutely something we would have implemented, in addition to an automated script with test data.

9.2 Performance Testing

A series of performance tests were done to investigate the system's accuracy and its speed. We performed two different tests, one with images that differed heavily from the original training set, and one that was conducted during fieldwork testing. The purpose was to see how the system would work with images that varied in quality.

The hardware that was used in both tests for the back end was a Windows 10 computer with an Nvidia RTX 3080 10GB GPU, AMD Ryzen 3900X CPU, with 32 GB of RAM. The mobile phone we used to capture images for the second test was a Huawei p20 pro, capturing images in 10 megapixels.

9.2.1 First Test

The first test set was not from the original training set, and differed heavily from it. The images were characterized by the abdominal pattern of the salamander not being clearly visible, and captured using a different method than what is intended for this system. The current method is imaging salamanders inside a custom built enclosure with a transparent bottom. The method which was used in the first testing set, was by placing salamanders inside a curved glass bottle and photographing them from the outside.

Test set

This test set consisted of 100 images in total, divided into 10 images per individual salamander. Most of the images would not be considered to be of adequate quality, as the salamanders often lay sideways or in other challenging positions. The reason for doing such a test, was to see how the model would handle images captured in a different way from the images we used for training. Three different images from the test set can be seen in Figure 9.1.

Results

The results of the first test can be seen in Table 9.2.

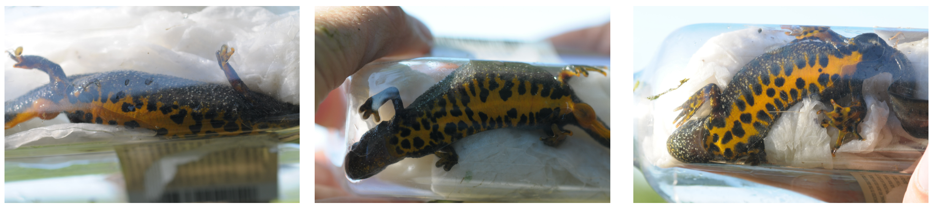
- **Failed to process**: Images that could not be straightened.
- **Managed to process**: Images that could be straightened, and was returned to the client.
- **Badly processed**: Images that were able to be straightened, but the returned abdominal pattern clearly showed signs of being warped and was not submitted. These images are a part of the "Managed to process" column.
- **Number of matches**: How many times match was registered per salamander. These images are a part of the "Managed to process" column.

- **False negatives**: Salamanders that were wrongly registered as a new salamander, even though they had been uploaded before. These images are a part of the "Managed to process" column.

Table 9.1: Results from the first test.

Number	Failed to process	Managed to process	Badly processed	Number of matches	False Negatives
1	5	5	0	3	2
2	10	0	0	0	0
3	6	4	2	2	0
4	5	5	0	3	2
5	1	9	2	5	2
6	5	5	0	4	1
7	3	7	1	6	0
8	5	5	2	2	1
9	3	7	5	2	0
10	2	8	1	7	0
Sum:	**45**	**55**	**13**	**34**	**8**

Out of the 100 images, 45 could not be processed. This means that the straightening algorithm was not able to process the image at all, and the user would have to upload a new image. These images were of too poor quality, as the user could not even proceed with these images. Figure 9.1 shows three different images used in the test, and explains what results they generated.



(a) Image could not be processed. **(b)** Image was processed, but produced a false negative. **(c)** Image was processed and matched correctly.

Figure 9.1: Images from the first test set.

Figure 9.1a was an image that could not be processed. The abdominal pattern in this image was not clear, and there were some reflection from the bottle that may have disturbed the processing. Figure 9.1b was processed, but resulted in a false negative. This image did not look like it would fail, but due to the salamanders position, some of its side was visible. The algorithm was likely unable to identify the mid line of the pattern, and probably used the mid line of the silhouette of the salamander as its reference point. This consequently caused the final pattern to be wrongly straightened. Different factors like lighting, background and dirt on the bottle may have contributed to cause this misjudgment. Figure 9.1c was processed and matched correctly. The abdominal pattern in this image was both clear and distinct.

The problematic part was the false negatives. In these cases, the image was able to be processed, which meant the user had to approve the straightened image they got back from the server. In most cases, it was clear that the images was badly processed (see the badly processed column in Table 9.2), and were not approved for further matching. However, there were a few that on first inspection did not look like badly processed images, and were wrongly matched by the algorithm. Figure 9.2 shows different processed images.

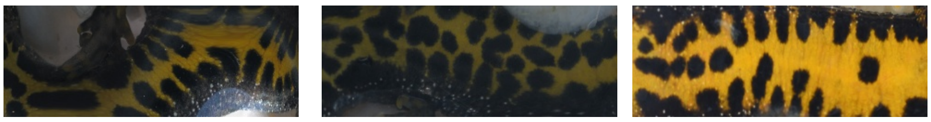

(a) Image is clearly malformed and would likely be discarded by a user. **(b)** Image might be accepted by a user, but produced a false negative. **(c)** Image is properly processed.

Figure 9.2: Images showing different results of the straightening process.

Figure 9.2a is clearly distorted, and would probably have been caught when the user were to confirm the processed image. Figure 9.2b is an image that might look like it is a good straightening on first impression, but does not work properly when it comes to matching. Figure 9.2c is a properly processed image, where the pattern is clear and distinct.

Out of the 100 images we tested, we had eight false negatives, and four of them were approved with reservations. If these images were captured in the way the system is intended for, the result would not be satisfactory. However, since these images were especially challenging and captured in a non intended way, these results does not reflect the systems capabilities when in actual use.

9.2.2 Second Test

The second test set was achieved from the field test we performed, with focus on capturing clear and precise images. This set of images was captured similar to the training set, in the way that the system is intended to be used.

Test set

This test set also consisted of 100 images in total, divided into 10 images per individual salamander. As in the first test, this test was not performed during fieldwork. This means that if some of the images were not able to be processed, there would be no way of taking new and better picture. The reason for performing this test, was to see how the model would handle images which was captured in the intended way. Still, it does not properly simulate the exact workflow, as the researchers in a fieldwork setting could just take a new picture if the straightening failed. Images from the test set can be seen in Figure 9.3.

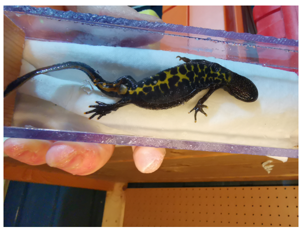
Results

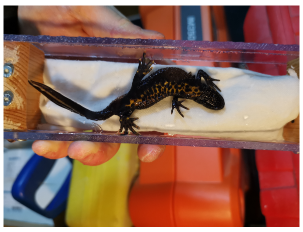
The results of the second test can be seen in Table 9.2. The same table descriptions as in the first test is used. Salamanders 1 and 2 were imaged extra carefully by three people. These images had a clear and uniform background compared to the rest. The latter images were taken by two people in a less precise and faster way, due us having little time at the test site. This might have affected the results.

Table 9.2: Results from the second test.

Number	Failed to process	Managed to process	Badly processed	Number of matches	False Negatives
1	1	9	2	7	0
2	1	9	0	9	0
3	4	6	0	6	0
4	0	10	0	10	0
5	1	9	6	3	0
6	0	10	1	9	0
7	1	9	1	8	0
8	0	10	0	10	0
9	1	9	7	2	0
10	6	4	2	2	0
Sum:	**15**	**85**	**19**	**66**	**0**

This time, out of the 100 images, only 15 could not be processed and there were no false negatives. This test produced much better results than the first test, both in terms of images that were able to be processed and number of images that were able to be matched. Figure 9.3 shows two different images used in the test, and explains what results they generated.


(a) Image was not able to be processed.


(b) Technically good image, but bad for the algorithm.

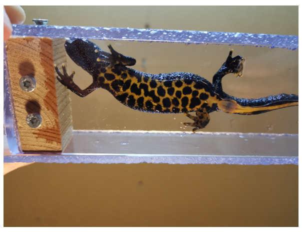

(c) Image was processed and matched correctly.

Figure 9.3: Images from the second test set.

Figure 9.3a was not able to be processed. This could be due to human fingers or excessive clutter in the background, the overall shape of the salamander, reflections or debris over the pattern, or other factors. Figure 9.3b is a technically good image with good contrast, lighting, and focus, but was not able to be processed properly. Figure 9.3c was able to be processed, likely due to its clear and distinct pattern.

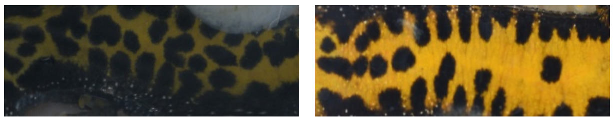

(a) Image might be accepted by a user, but produced a false negative. **(b)** Image is properly processed.

Figure 9.4: Images showing different results of the straightening process.

Figure 9.4 shows a badly processed image and a properly processed image from the field test.

9.2.3 Discussion

From the first test we can see that when the test data differ significantly from the training set, the algorithm often fails to process the images. However, from the second test, we can see that when given images taken in a similar way to the training set, it performs significantly better.

The positive we can extract from the first test, is that in 45 cases of 66 negative outcomes (45 failed to process + 13 badly processed + 8 false negatives), the algorithm refused to straighten the image, and the user was not be able to continue. This may sound contradictory, but it is better that the algorithm fails at processing a poor quality image, than to return a badly straightened image. The users may think that if the image is able to be processed, it is acceptable. This could then lead to the user accepting the badly processed image and result in a false negative.

In the second test, there were zero false negatives. In addition, only 15 images out of 100, failed to be processed. This is a significant improvement from the first test. However, there still were 19 images that were badly processed. This means that the researchers still have to be cautious before approving a straightened image, so that no false negatives will be registered.

9.3 User Testing

Our original plan was to perform user testing with researchers at NINA, while they were working in the field. The salamanders were expected to wake up from hibernation around 15th of April. However, due to no rain and low temperatures, this was postponed until 5th of May. As the deadline of the bachelor's thesis was 20th of May, we had to rethink our plans.

9.3.1 User Experience Testing

We decided to perform an overall usability focused test with both our product owner from NINA and non-researchers. This means, we tested the application in a controlled environment, instead of during fieldwork. The aim of this test was to see how intuitive the functionality of the application was, and to catch flaws we had yet to discover. There are not that many salamander experts in Norway, and we wanted to expand our pool of testers, which is why we included non-researchers to test the usability.

For this test, we made a guide for us to follow. This guide can be seen in Appendix I.1. We drew inspiration from the use cases (seen in Section 2.3), and observed the participants while they performed the tasks. After the test was over, the participants answered an anonymous Google survey, where they could rate their first impressions and provide additional feedback. This was to catch possible feedback the participants either forgot to provide, or if there were something they wanted to say that they did not want to say to us directly. In total, there were eight testers. The notes from the observation can be seen in Appendix I.2, and the results from the Google survey can be seen in Appendix I.3.

Results

The participants liked the color scheme and the overall simplicity of the application. We got feedback that some of the terminology used in the application, like 'confirm' was a bit confusing for some operations and 'choose image' was a bit ambiguous. In addition, there were some bugs like the default android back-button not functioning properly on some screens, and some regexes not working the way we intended.

The most crucial flaw, was that almost all participants seemed to struggle with registering a new location. For some, it took time to figure out that they had to long press the map to add a marker. The ones that did figure this out, struggled to understand that they had to click on the marker again to go to the register location screen.

Another flaw was that the buttons for accepting and denying users were too small and too light to easily tell the difference between the two. This was something that needed to be fixed, as the outcomes of the two buttons are critically different.

Reiteration

After the testing, we began to fix some of the issues we observed based on the feedback we received. We fixed the biggest flaw, which was the register location use case. The user still has to long press to add a location, but after performing that action, a button on the bottom part of the screen will appear (see Figure 9.5).

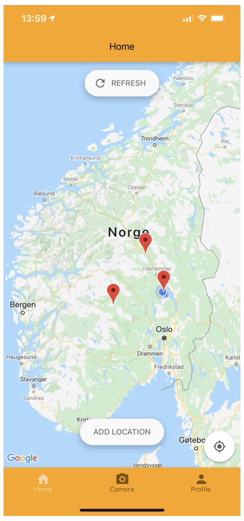
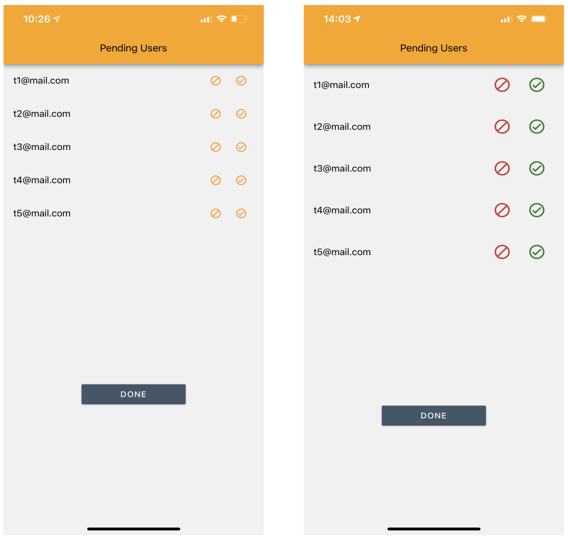


Figure 9.5: The final version of the add location functionality.

For pending users, we simply changed the color from yellow to red for denying users, and green for approving users. In addition, we made the buttons larger so they are more easily visible. This is shown in Figure 9.6.



t1@mail.com
t2@mail.com
t3@mail.com
t4@mail.com
t5@mail.com

(a) First version. (b) Final version.

Figure 9.6: Reiteration of pending users.

As shown in Figure 9.6a, it was quite hard to differentiate the two buttons when they had the same color, and the color blended in with the background. Looking back at the PACT-analysis, this was one area where we did not fulfill our requirement in regards to bad eyesight. Thankfully, this was caught during the user experience test.

Criticism

For this testing session, we did not have a large test group, as only eight testers is a limited amount. This was mostly due to Covid-19 restrictions at the time. In addition, there are limited salamander experts in Norway, and it was hard to find relevant testers. We did discuss to only test on NINA researchers, however we decided that it was not necessary to be acquainted with the subject to test the usability.

Besides our product owner, we mostly tested on family and friends. This probably generated some biases in our favor when getting feedback. To mitigate this, we tried to observe the participants reactions as best as possible. This would make it easier for us to catch flaws the testers either forgot to provide feedback on, or if they glossed over some of the struggles when reviewing the application.

9.3.2 Field Testing

For the field testing, we conducted a new test guide with a different focus compared to the user experience test. This time, the purpose was to observe and analyze how the researchers at NINA used the application during fieldwork. The test was conducted during night time, as the salamanders wanders to their birth pond at this time of day, after waking up from hibernation. The guide can be seen in Appendix I.4. Additional images from the fieldwork testing can also be seen in Appendix I.5.

Results

In the guide, we defined a set of questions we wanted answers to, when conducting the fieldwork test. The questions and answers are listed below:

How do the researchers interact with the application during field work?

The researchers would attempt to get a good shot of the salamander by storing it in a transparent container. They would keep the salamander in place with a wet piece of cloth and take an image with the application from outside of the container. To get the right brightness, a lamp was used and the researcher tried to make sure that there were no reflection from the light source in the captured image. In Figure 9.7, usage of the application and setup is shown.

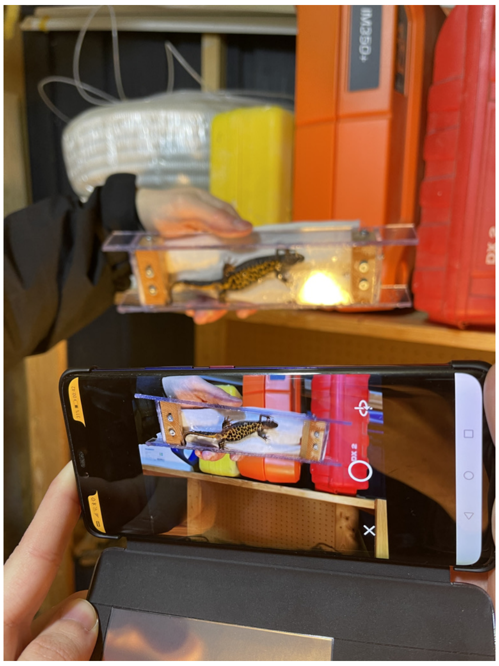


Figure 9.7: Capturing an image of a salamander with the mobile application.

Does lighting have any impact on taking pictures and processing?

Lighting was discovered to be a large factor as it is important to take images where the abdominal pattern is clear and distinct. As the test was conducted during the night, the light source came from a light bulb. The reflection of the light source some times disturbed the image. In addition, it was hard to take a good picture without assistance from another person, as one person would have to hold the mobile phone, light source and the salamander at the same time. A proper setup could fix this issue.

How does the outside environment affect the application?

Background and obstructions in front of the pattern seem to affect how the model will determine the points. Any form of obstruction between the pattern and the camera, affected the straightening and matching. Since it was dark outside, and no natural lighting, there were some difficulties regarding reflection from the light sources available. Future salamander registrations will be conducted during daytime as the salamanders are caught in the ponds, and not at night.

Is the user experiencing any annoyances during periods of waiting/bottlenecks in workflow?

The researcher found the application to be fast, as they were used to an even slower process. However, downloading images were the obvious sign of bottlenecks, and this needs to be addressed in a future release.

Does the researchers understand the feedback?

Whenever the researcher made a mistake, like trying to register an invalid password, the helper text helped guiding the user.

How does the mobile cellular coverage affecting the application?

The speed of the internet connection had a huge impact on the application's performance, mostly in regards to managing salamanders. When downloading images from the server, to be able to manage them, we experienced at one point that the request timed out. This was due to the slow 4G download speed. We did try to connect to a WiFi, which had faster network speed, and it resulted in the salamander being managed.

How is the mobile's battery life affected?

As the screen is always on and the outside temperature was low (2 degrees Celsius), the power consumption was rather high.

How is the workflow if there are several users at the same time?

Unfortunately, we did not have enough testers to test this.

Conclusion

After the test was finished and the data was gathered, we started processing it. The first thing that we noticed was that the setup for capturing images of salamanders was not ideal. At some point we needed three people for capturing the best possible image. One for holding a light source, one for holding the salamander and one for capturing the image. A better setup, which we discussed with the researcher, was to place the salamander enclosure on a table with enough space in between, where the transparent enclosure could be stationed.

The most significant flaw we discovered during the test was the slow performance when managing salamanders. This was mostly due to slow network speed. Future work will be to look at another way to send the images from the server to the client. The best option may be to compress the images before they are sent.

Criticism

Unfortunately, we were only able to get one researcher to test the system during fieldwork. While we did get useful information from this test, it would have been ideal to test on multiple people. This was due to the same reasoning as in the usability test, except for this time the tester was required to be acquainted with the subject to properly test the system.

Chapter 10

Discussion

This chapter will discuss and reflect on both the process and the product of the project.

10.1 Process

This section will discuss aspects regarding the bachelor's thesis process, including how the initial plan was followed, thoughts on the usage of Scrum, how the group worked together and critique of the process.

10.1.1 Project Plan

We were able to follow our initial project plan. We accomplished this by laying out a plan for each week, and we rarely diverged from what we agreed upon. During each sprint, we allocated a few hours each week for thesis writing. We feel this approach made the content more accurate, as we where able to write in real time and not in retrospect. In addition, we were able to get feedback from our supervisor on a regular basis. The downside was that we had to rewrite some of the early chapters as some of the requirements changed throughout the development. Still, we believe that the quality of the writing increased during the project period, and that by starting early, we had more time to get accustomed to English academic writing. We also had more time in the end to polish the thesis.

By Easter, the system was mostly finished, and the only thing we were not able to do before the break was deployment. However, our continuous writing on the thesis during the sprints gave us enough time to deploy the system after the break.

Another issue was that testing had to be postponed due to the dry, cold weather. The salamanders had not yet woken up from their hibernation to start migrating to their birth pond. We could not perform fieldwork tests on the planned date, so we decided to conduct a user experience test earlier in the process to make up for this hindrance. By doing this, we were able to do one iteration of the application.

However, due to the postponement of the fieldwork test, we did not have time to improve the system after the fieldwork test.

10.1.2 Technologies

In this section, the overall experience of our technology choices, including the front end, the back end, and other tools used in the bachelor's project will be discussed.

Front End

Our experience of using React Native to develop the mobile application was good. Expo made it easy to test during development and see changes live. Since we had some experience with web development from a previous course, it was easy to adapt to React Native as many of its aspects resemble HTML and CSS. Compared to development of Native applications in Android Studio, we all agree that using React Native gave a better development experience. This was due to multiple factors, with the most prominent factor being that it was easier to style the application by using a CSS-like grid system rather than the way done in Android Studio using XML.

Back End

As mentioned in Chapter 3, we chose to write the back end in Python primarily because the original source code was written in it. Considering the nature of what Python is and its flaws, there are many reasons we could have used another programming language. For instance, C++ is a faster language, and would most likely have performed better [72]. However, by not rewriting the source code to a different language, we saved a lot of time.

Our decision to use DeepLabCut for finding the salamander pattern was strongly driven by its previous usage in the bachelor's project from 2019 [7]. This tool proved to be reliable and easy to use. However, since it is made for specifically estimating the pose of animals and humans, and not for pattern recognition on salamanders, we had to make some adjustments for it to work properly in our case.

In the middle of the development period, we chose to discontinue our work on automatic sex identification. After getting undesirable results from several models of different types, we decided to stop working on this. We believe the time needed to get an accurate model, would be better spent on developing other crucial features.

Our decision to use Flask as the main library for creating the REST API proved to be good choice. Its built in development server helped us throughout the development period for both testing and debugging. Since Flask also allows for many different extensions, we were able to easily incorporate the ones we needed, like Flask-Limiter and Flask-SQLAlchemy.

Additional Tools

Our workflow in GitLab was to create merge requests of each issue, which was based on tasks from the sprint backlog. When a task was completed, it was merged into the master branch. We had two different code bases: one for the mobile application, and one for the server. We did not use GitLab to its full potential, and if we were to do this project over again, we would have utilized pipelines for automatic testing and quality control.

Clockify was used to track time continuously throughout the project. This was a useful tool, as we could go through the logs if we had forgotten when we did a certain activity or implemented a feature. This was especially helpful in regards to thesis writing, as we sometimes had to write about something we had done months prior. The interface was easy to use, and it also gave us statistics that we could use in the end of the project, which can be seen in Appendix F.

To write our thesis, we used a template made by Ivar Farup [13] in Overleaf. Overleaf made it easy for us to work on the thesis concurrently, and was a great tool to use for writing. The tips provided by Farup also aided us in writing this thesis. Still, we made slight modifications to the template to accommodate our thesis.

10.1.3 Scrum

During the development phase of the project we used Scrum as our software development model. This proved to be useful as our system requirements changed several times during development. However, we did not use Scrum to its full potential. In the beginning we scheduled daily scrum meetings. As development progressed, these meetings became less frequent and were only held when necessary. Although we used Scrum, we incorporated more features from Kanban than originally intended, and one could argue that our software development model was more similar to Scrumban [73].

Scrum Tools

The use of Trello differed between the group members. Some used it actively, while others did not. However, we found it to be a useful tool to keep track of what was remaining in each sprint. It might have been even more useful if we had performed code reviews regularly, as the "to review" column was rarely used.

A planning poker session was always carried out during each sprint planning meeting. Our estimations varied throughout each sprint, which we will go deeper into in the next section. One of the main problems was estimating tasks that involved unfamiliar technologies. They were often given high estimates, because of our inexperience and the tasks usually being large in scope. Ideally, the large tasks should have been divided into smaller, more manageable ones.

Sprints

Below is a comparison of our initial estimations in regards to development during each sprint, to the actual time spent on it. The numbers can be seen in Table 10.1.

Table 10.1: Table of hours used per sprint.

Sprint Number	Estimated Development (hours)	Actual Development (hours)	Total Available (hours)	Actual Time Spent (hours)
1	153	141	240	223
2	150	174	240	230
3	218	225	240	240
4	159	184	240	236
Sum:	**680**	**724**	**960**	**929**

When looking back at the estimations we made for each sprint, the estimated development hours and actual development hours are quite close. Sprint 1, 2, and 3 had at most a deviation of 24 hours in regards to development. Sprint 4 had the largest deviation with 25 hours. Although this seems acceptable, we actually failed to estimate development hours for this sprint correctly. Table 5.4, in Chapter 5 shows that we estimated 89 hours for deployment of the application and the REST API. However, deployment did not happen during that sprint, as we had to focus on implementing the manage salamander feature. This means that we actually underestimated time spent on development by 114 hours.

10.1.4 Communication

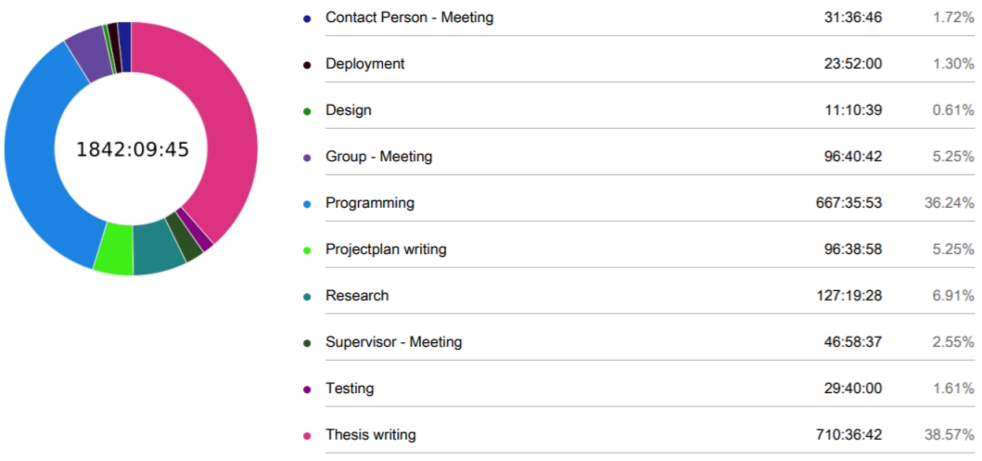
Overall, we had good communication during the whole process. We had weekly meetings with our supervisor, and meetings with our product owner every two weeks. This was maintained during the whole project. In addition, we continuously met at school, and everyone was kept in the loop. Our group communication was definitely a big factor as to how we were able to stick to schedule and reach our deadlines.

10.1.5 Group Work

The group worked really well together, as we all had similar expectations in regards to how much effort we wanted to put into this bachelor's project. We also had a similar approach regarding workdays, agreeing to meet at school at 9 am and work until 4 pm. This approach worked for us in earlier projects during our time at NTNU, and it worked well during this thesis.

As we all became invested in the project, we had several discussions regarding both the product and the thesis. Any disagreement were brought to the table and we had a group discussion. Even though our group leader had the ability to make the final decision, we often came to an agreement without this being necessary. We also made sure to do activities unrelated to the bachelor's project together, to boost team morale.

10.1.6 Work Allocation



●	Contact Person - Meeting	31:36:46	1.72%
●	Deployment	23:52:00	1.30%
●	Design	11:10:39	0.61%
●	Group - Meeting	96:40:42	5.25%
●	Programming	667:35:53	36.24%
●	Projectplan writing	96:38:58	5.25%
●	Research	127:19:28	6.91%
●	Supervisor - Meeting	46:58:37	2.55%
●	Testing	29:40:00	1.61%
●	Thesis writing	710:36:42	38.57%

Figure 10.1: Work allocation.

Figure 10.1 shows how we spent our time during the entire project period. We grouped similar activities using tags, so we could easily track time spent per activity. Most of the time was used on thesis writing and programming. We also wanted to track time used on smaller activities like design, deployment, and testing, which did not fit into the bigger categories.

Looking at the end product and thesis, we are satisfied with how we allocated our available time. We managed to complete our product in time, and we also got to use a considerable amount of time writing our thesis. If we were to do this project again, we would choose a similar approach.

10.1.7 Covid-19 Situation

As we were aware of the ongoing Covid-19 pandemic in the planning phase, we tried to plan and organize to accommodate the situation. We made several plans based on the possibility of new and stricter Covid-19 restrictions, in addition to keeping the number of personal interactions to a minimum. We also made sure we had an additional physical place to work together in case of a new lockdown on campus, as this was our preferred style of working. Additionally, we had digital communication tools like Discord, Messenger, and Microsoft Teams that could be utilized in the case of potential quarantines.

10.1.8 Critique of Process

During the development process we did not take advantage of automated unit tests. We focused on getting the functionality done and only did manual testing. In retrospect, we see that it would have been helpful to have automated tests for the REST API, as this was where the main functionality was. This was also where we spent most time manually testing. If we had planned accordingly, we could have been able to go for a more test driven development approach. However, we realized this too late in the process, and decided not to implement it just for the sake of adding it. In addition, the project period was rather short, and it would have made sense to use more time on testing if the project had a longer time frame.

We also did not use code reviews regularly. Several bugs and issues could have been avoided by reviewing the code in addition to improving the code quality. However, as we often worked together in the same room or did pair programming, issues were often caught during implementation.

We originally planned to deploy our system on one of NINA's servers. This proved to be problematic as the server we were given access to did not allow for communication with clients outside NINA's network. Since we were made aware of this late in the process, we did not have enough time to solve the issue. A contributing factor to this was that we got in contact with the server technicians at NINA too late. If we had started on deployment sooner, we might have gotten this information early enough to accommodate the situation.

10.2 Product

In this section, we will revisit our result goals, discuss how choices we made during development affected the end product, the product's ethical and societal outcomes and a critique of the product.

10.2.1 Revisiting Result Goals

We will revisit the result goals from Chapter 2, and discuss to what degree these goals were met. A goal will be listed first in bold, with our assessment below.

A working web-server that incorporates an improved version of the previous matching algorithm and will communicate with mobile phones used in the

field.

This goal was met as our end product includes a mobile application that communicates with a REST API that has incorporated an improved version of the matching algorithm from 2019.

A cross-platform mobile application that will work as a client to the server.

We managed to develop an application which is able to connect to a remote server on a different network. Even though we used a cross platform development framework, we only built the mobile application for Android since the researchers we tested with only had Android phones. However, building it for iOS in the future is possible, but will require an Apple developer account.

The system should have a better user experience than the existing system.

The old system was never a standalone application. It was instead a collection of files which could be run sequentially from the command line. We, together with our contact person at NINA, concluded that our system delivered a significantly better user experience than the old system.

The system should be able to classify the sex of the salamander with 95% accuracy.

Although we got close to completing this feature, we concluded with our product owner not to incorporate it. We determined that due to the increased time a sex estimation took, and how imprecise the model was, it was not worth including in the finished product. In addition, this feature was non-critical as the researchers at NINA are able to recognize both sex and species without problems.

The system should be able to classify the salamander's species with 95% accuracy.

We did not have the ability to implement this, because we only used images of northern crested newts to train a model. Even if we had a dataset of smooth newts captured in a similar way to our training set, we would have come to the same conclusion as we did with sex estimation. Similarly to sex identification, the researchers at NINA know what species they are identifying, and this feature was deemed non-critical.

Reduce the search time for matching salamanders by 50% on average by dividing the search by sex.

In theory, the search time should be reduced by 50% on average because of how we have structured the database. In hindsight, this is not a good quantitative goal, considering that we could have calculated the outcome of the planned implementation.

Reduce the search time for matching salamanders by another 50% on average by dividing the search by species.

In theory, the search time should also be reduced by another 50% after the previous goal because of the way we have structured the database.

Reduce the search time by 50-99% by dividing the search by geographical location.

We were able to structure the salamander data in a way that allowed us to separate it by location. Assuming each location has the same number of salamanders associated with it, we can calculate the estimated time savings in matching compared to not dividing by location. The more locations the system has, the more time is saved.

The server must support a capacity of at least five researchers concurrently.

For most of the requests, the user does not have to wait more than a second to get a response. The only request that takes a considerable amount of time is the straightening request. This operation uses a semaphore to make sure only a single user has access to the GPU at a given time. When multiple users try to use it they will be put in a queue, which can cause delays and timeouts.

10.2.2 Consequences of Design Choices

In this section, design choices that especially affected the end product will be discussed.

Folder Structure

Our decision to divide salamanders into species, location, and sex proved to be useful for registering salamanders. Although we never registered enough salamanders in our tests to measure any meaningful data on time taken by the matching function or its accuracy, we still found this way of grouping salamanders intuitive and orderly. Being able to look up salamanders based on these aspects proved to be useful regardless. In addition, it makes it easier to estimate the salamander population based on locations.

Confirmation of Processed Image

Early in the process, we decided to return the processed image to the user, so they could confirm that it had been straightened properly. Even though we thought this feature would be useful for quality control, we did not imagine how important this feature proved to be. As the performance tests showed, while we tried to capture images to the best of our ability, there were still some badly processed images. If the system had not returned these images to the user, the user would have had no way of knowing if the image was acceptable or not. This would most certainly have led to faulty matching.

Manage Salamanders

In the end of sprint 3, we had a meeting with our product owner and decided to implement the ability to manage salamanders. Even though this led to a delayed deployment, it made the end product more refined. This made the user able to correct any mistakes made when registering salamanders, which would not have been possible without manually editing data on the server. Due to the modular design of the system, this proved easy to incorporate.

Sex Estimation

Our decision to end our attempt at automatic sex identification proved to be a good idea. There is a chance we would have been able to make it work before the end of development, but it most likely would have taken time away from other more important tasks. We also observed how simple it was for the researchers from NINA to identify the sex manually and enter it into the mobile application during our field test.

10.2.3 Ethical and Societal Outcomes

The system will hopefully reduce the need for PIT-tagging, and other invasive procedures. In addition, by helping the researchers track measures implemented to increase the salamander population, we hope this tool will aid in increasing the population of the northern crested newts, and eventually the smooth newts.

10.2.4 Critique of Product

One criticism of the product is how we handle image transfer between the mobile application and the server. Downloading images from the server takes a long time. This was discovered during the field test we conducted, as we had mainly used internet with high speed during development. The reason for the slow download is still not clear.

One hypothesis is that the way the system encodes images on the server and is sent to the client is inefficient. Another hypothesis is that the images sent to the mobile application are too large, which affects the time it takes to download them. For example, when managing salamanders, the original image is sent back in its original resolution. If a salamander has several high resolution images attached to it, downloading all of them can take a long time. This is also a concern when using mobile data, as most people have strict data quotas which will quickly be exhausted. This is something we should have considered when implementing the manage salamanders feature.

The way we chose to implement the "forgot password" feature is not ideal. Only an administrator can reset a user's password. The new password will be set to a random string of characters which the admin has to relay to the affected user. This system puts a tremendous amount of trust on the administrators as they can easily abuse the system. In retrospect, we should have set up a mail server that could send a reset password link to the user who needs to reset it. The reason we chose our approach is because we did not have time to learn and set up a mail server. A mail server also requires a domain which often costs money. Since our system will only be used by a small group of people, we do not consider this feature a significant security concern.

Chapter 11

Conclusion

This chapter will conclude our thesis providing a summary of our process and end product, in addition to cover future work and our final words.

11.1 Summary

This section will provide a summary of our bachelor's thesis process and the final product.

11.1.1 Process

We are pleased with how we planned and structured this project. By choosing Scrum as our software development model and having consistent meetings with both our supervisor and product owner, we were able to complete both the product, and our thesis in time. However, we should have clarified deployment with NINA earlier in the process, which could have resulted in the system being deployed on their server. Overall, despite the ongoing Covid-19 pandemic, we were able to support and motivate each other throughout this period.

11.1.2 Product

The final product is in essence a tool with a single purpose, which we hope will satisfy NINA's needs. With proper instructions, researchers will find this tool useful and easy to use. During the field test, the application proved its usefulness, and our product owner was pleased with the results. With limited changes and training of new models, this system can serve as a template for other species with unique patterns, like trouts and frogs.

11.2 Future Work

Below is a list of features we did not have time or resources to implement, and should be implemented in the future.

- Collect more images of northern crested newts and smooth newts with varying backgrounds in order to create a larger and more varied data set. This will allow for the creation of better training sets resulting in better models, and the possibility of automatic species estimation.
- Implement automatic identification of the salamander's sex.
- More efficient file transfer between the server and client; particularly downloading images from the server as downloading took significantly longer than uploading.
- Allow for straightening salamander images concurrently, as they currently happen sequentially. Adding this feature would significantly reduce waiting times when multiple users try to straighten images.
- Implement a feature for automatic population estimation at locations.
- Deployment of the back end on NINA's server.
- Build the mobile application for iOS.

11.3 Final Words

During this bachelor's thesis, we have acquired new knowledge of working on large group projects and several technologies. Developing a system meant to aid the researchers at NINA in preserving, monitoring, and increasing the population of an endangered species has been both exciting and rewarding.

This bachelor's project gave us the opportunity to not only dive into new fields like artificial intelligence and image recognition, but also expand our knowledge in fields like full stack application development and REST API design. We were able to dive deeper into new frameworks and libraries, and have improved our ability to take new challenges head on.

In conclusion, we are very satisfied with how this project turned out. Due to great teamwork and a strict schedule, we were able to develop a complete system and a thesis we are proud of.

Bibliography

[1] KirstenS and contributors, *Cross site request forgery (csrf)*, Aug. 2021. [Online]. Available: `https://owasp.org/www-community/attacks/csrf`.

[2] F. community, *Flask, web development, one drop at a time*, 2010. [Online]. Available: `https://flask.palletsprojects.com/en/1.1.x/` (visited on 02/26/2021).

[3] F. community, *Flask-jwt-extended*, 2020. [Online]. Available: `https://flask-jwt-extended.readthedocs.io/en/stable/` (visited on 02/26/2021).

[4] A.-A. Saifee, *Flask-limiter 1.4*, Aug. 2020. [Online]. Available: `https://pypi.org/project/Flask-Limiter/` (visited on 05/15/2021).

[5] Wikipedia contributors, *React native — Wikipedia, the free encyclopedia*, 2021. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=React_Native&oldid=1005998180` (visited on 02/15/2021).

[6] *Nina - the norwegian institute of nature research*. [Online]. Available: `https://www.nina.no/english/Home` (visited on 01/15/2021).

[7] J. Bakløkken, F. Schoeler, and H. Nørholm, *Automated salamander recognition using deep neural networks and feature extraction*, May 2019. [Online]. Available: `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2617897` (visited on 01/15/2021).

[8] B. K. Dervo, *Salamander*. [Online]. Available: `https://www.nina.no/salamander` (visited on 01/15/2021).

[9] Artsdatabanken, *Norsk rødliste for arter*, Mar. 2020. [Online]. Available: `https://artsdatabanken.no/Rodliste` (visited on 11/24/2021).

[10] N. U. of Science and Technology, *Programmering*. [Online]. Available: `https://www.ntnu.no/studier/bprog` (visited on 02/02/2021).

[11] *Lov om behandling av personopplysninger (personopplysningsloven)*, 2018. [Online]. Available: `https://lovdata.no/dokument/NL/lov/2018-06-15-38` (visited on 01/15/2021).

[12] A. ALTVATER, *What is sdlc? understand the software development life cycle*, Apr. 2020. [Online]. Available: `https://stackify.com/what-is-sdlc/` (visited on 03/22/2021).

[13] I. Farup, *Thesis-ntnu*. [Online]. Available: https://github.com/COPCSE-NTNU/thesis-NTNU (visited on 02/02/2021).

[14] V. Trulock, *Pact analysis*. [Online]. Available: http://hci.ilikecake.ie/requirements/pact.htm (visited on 02/09/2021).

[15] C. Jensen, *Learn from the best: Mobile design principles*, Sep. 2020. [Online]. Available: https://uxdesign.cc/learn-from-the-best-mobile-design-principles-f1bdc46bc016 (visited on 05/12/2021).

[16] Usability, *Use cases*, Unknown. [Online]. Available: https://www.usability.gov/how-to-and-tools/methods/use-cases.html (visited on 02/10/2021).

[17] Egnyte, *How long will it take to upload, backup, or download my files?* [Online]. Available: https://helpdesk.egnyte.com/hc/en-us/articles/201637064-How-Long-Will-it-Take-to-Upload-Backup-or-Download-my-Files- (visited on 02/08/2021).

[18] Ookla, *Speedtest global index*, Dec. 2020. [Online]. Available: https://www.speedtest.net/global-index/norway (visited on 02/08/2021).

[19] Willip, *Is 2g data still usable?* Jul. 2019. [Online]. Available: https://www.giffgaff.com/blog/is-2g-data-still-usable/ (visited on 02/08/2021).

[20] Wikipedia contributors, *Agile software development — Wikipedia, the free encyclopedia*, Jan. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=999545034 (visited on 01/26/2021).

[21] Wikipedia contributors, *Kanban (development) — Wikipedia, the free encyclopedia*, Jan. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Kanban_(development)&oldid=1002634912 (visited on 01/26/2021).

[22] M. REHKOPF, *Kanban vs. scrum: Which agile are you?* [Online]. Available: https://www.atlassian.com/agile/kanban/kanban-vs-scrum (visited on 01/26/2021).

[23] M. Gorman, *Scrum vs kanban: Weighing their pros and cons*. [Online]. Available: https://www.kovair.com/blog/scrum-vs-kanban-pros-and-cons/ (visited on 01/26/2021).

[24] T. Hristovski, *Agile methodologies: Kanban vs scrum – advantages and disadvantages*, Dec. 2017. [Online]. Available: https://iwconnect.com/agile-methodologies-scrum-vs-kanban-advantages-disadvantages/ (visited on 01/26/2021).

[25] Wikipedia contributors, *Scrum (software development) — Wikipedia, the free encyclopedia*, Jan. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Scrum_(software_development)&oldid=999915311 (visited on 01/26/2021).

[26] L. C. Team, *What is extreme programming? an overview of xp rules and values*. [Online]. Available: `https://www.lucidchart.com/blog/what-is-extreme-programming` (visited on 01/26/2021).

[27] StatCounter, *Mobile operating system market share norway*, Jan. 2021. [Online]. Available: `https://gs.statcounter.com/os-market-share/mobile/norway` (visited on 02/15/2021).

[28] Techliance, *Unraveling the growing popularity of react native*, 2021. [Online]. Available: `https://blog.techliance.com/unraveling-the-growing-popularity-of-react-native/` (visited on 02/26/2021).

[29] F. Faisal, *How to develop ios apps on windows*, Jan. 2021. [Online]. Available: `https://mindster.com/how-develop-ios-apps-windows/` (visited on 05/14/2021).

[30] Expo, *Introduction to expo*, 2021. [Online]. Available: `https://docs.expo.io/`.

[31] D. contributors, *How to install deeplabcut*. [Online]. Available: `https://github.com/DeepLabCut/DeepLabCut/blob/5154c336f81ca57a05409c90a08aac7a9451eef9/docs/installation.md` (visited on 05/03/2021).

[32] M. Herman, *Django vs. flask in 2020: Which framework to choose*, Oct. 2020. [Online]. Available: `https://testdriven.io/blog/django-vs-flask/` (visited on 02/15/2021).

[33] M. contributors, *Https://developer.mozilla.org/en-us/docs/web/api/fetch*$_a$*pi/using*$_f$*etch*, Apr. 2021. [Online]. Available: `https://arxiv.org/abs/1804.02767`.

[34] V. Anushka, *Difference between fetch and axios.js for making http requests*, Aug. 2020. [Online]. Available: `https://www.geeksforgeeks.org/difference-between-fetch-and-axios-js-for-making-http-requests/`.

[35] R. N. Community, *React navigation*, 2021. [Online]. Available: `https://reactnavigation.org/` (visited on 02/24/2021).

[36] Callstack, *Making your react native apps look and feel native*, 2019. [Online]. Available: `https://reactnativepaper.com/` (visited on 03/25/2021).

[37] Wikipedia contributors, *Redux (javascript library)*, Mar. 2021. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Redux_(JavaScript_library)&oldid=1012646179`.

[38] K. Burke, K. Conroy, R. Horn, F. Stratton, and G. Binet, *Flaskrestful*, 2020. [Online]. Available: `https://flask-restful.readthedocs.io/en/latest/` (visited on 02/26/2021).

[39] A. Ronacher, *Flask-sqlalchemy*, 2020. [Online]. Available: `https://pypi.org/project/Flask-SQLAlchemy/` (visited on 02/26/2021).

[40] Flask-Bcrypt contributers, *Flask-bcrypt*. [Online]. Available: `https://flask-bcrypt.readthedocs.io/en/latest/` (visited on 02/26/2021).

[41] J. J. Kumar, *Image-encoder 0.0.8*, May 2020. [Online]. Available: https://pypi.org/project/image-encoder/ (visited on 05/15/2021).

[42] DeepLabCut contributors, *Deeplabcut*, Mar. 2021. [Online]. Available: http://www.mackenziemathislab.org/deeplabcut (visited on 03/24/2021).

[43] W. contributors, *Tensorflow — Wikipedia, the free encyclopedia*, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=TensorFlow&oldid=1018433675.

[44] I. contributors, *Imageai*, 2021. [Online]. Available: https://github.com/OlafenwaMoses/ImageAI.

[45] O. contributors, *Open source computer vision*, 2021. [Online]. Available: https://docs.opencv.org/4.5.2/d1/dfb/intro.html.

[46] M. Olafenwa and J. Olafenwa, *Imageais repository*, 2020. [Online]. Available: https://github.com/OlafenwaMoses/ImageAI (visited on 02/26/2021).

[47] J. Redmon and A. Farhadi, *Yolov3: An incremental improvement*, Mar. 2021. [Online]. Available: https://arxiv.org/abs/1804.02767 (visited on 03/24/2021).

[48] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. [Online]. Available: https://arxiv.org/abs/1512.03385 (visited on 02/24/2021).

[49] Google, *Bottom navigation*. [Online]. Available: https://material.io/components/bottom-navigation#usage (visited on 03/25/2021).

[50] Google, *Drawer navigation*. [Online]. Available: https://material.io/components/navigation-drawer#usage (visited on 03/25/2021).

[51] B. Carlson, *Designing for action: Best practices for effective buttons*, Unknown. [Online]. Available: https://balsamiq.com/learn/articles/button-design-best-practices/ (visited on 05/06/2021).

[52] S. Minhas, *7 rules of using radio buttons vs drop-down menus*, May 2018. [Online]. Available: https://blog.prototypr.io/7-rules-of-using-radio-buttons-vs-drop-down-menus-fddf50d312d1 (visited on 05/06/2021).

[53] A. developers, *Toasts overview*, Apr. 2021. [Online]. Available: https://developer.android.com/guide/topics/ui/notifiers/toasts (visited on 05/12/2021).

[54] Wikipedia contributors, *React (javascript library) — Wikipedia, the free encyclopedia*, 2021-04-19, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=React_(JavaScript_library)&oldid=1018689613.

[55] F. Inc., *Function components and class components*, 2021. [Online]. Available: https://reactnative.dev/docs/getting-started (visited on 04/19/2021).

[56] F. Inc., *Safeareaview*, 2021. [Online]. Available: https://reactnative.dev/docs/safeareaview (visited on 04/19/2021).

[57] R. N. Community, *Createstacknavigator*, 2021. [Online]. Available: https://reactnavigation.org/docs/stack-navigator/ (visited on 04/19/2021).

[58] J. Simpson, *Everything you need to know about api rate limiting*, Apr. 2019. [Online]. Available: https://nordicapis.com/everything-you-need-to-know-about-api-rate-limiting/ (visited on 04/20/2021).

[59] imgaugs contributors, *Imgaug*. [Online]. Available: https://github.com/aleju/imgaug (visited on 04/27/2021).

[60] C. Versloot, *Getting out of loss plateaus by adjusting learning rates*, 2020. [Online]. Available: https://www.machinecurve.com/index.php/2020/02/26/getting-out-of-loss-plateaus-by-adjusting-learning-rates/ (visited on 05/05/2021).

[61] C. Szegedy, A. Toshev, and D. Erhan, *On the difficulty of using simple neural network*. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1809/1809.09645.pdf (visited on 04/23/2021).

[62] R. Pascanu, T. Mikolov, and Y. Bengio, *On the difficulty of training recurrent neural networks*. [Online]. Available: https://arxiv.org/abs/1211.5063 (visited on 04/19/2021).

[63] K. Pykes, *The vanishing/exploding gradient problem in deep neural networks*, May 2020. [Online]. Available: https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-191358470c11 (visited on 05/14/2021).

[64] C.-F. Wang, *The vanishing gradient problem*, Jan. 2019. [Online]. Available: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484 (visited on 05/14/2021).

[65] Z. Li, W. Yang, S. Peng, and F. Liu, *A survey of convolutional neural networks: Analysis, applications, and prospects*. [Online]. Available: https://arxiv.org/abs/2004.02806 (visited on 04/19/2021).

[66] K. He, X. Zhang, S. Ren, and J. Sun, *Identity mappings in deep residual networks*. [Online]. Available: https://arxiv.org/abs/1603.05027 (visited on 04/19/2021).

[67] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*. [Online]. Available: https://arxiv.org/abs/1512.03385 (visited on 04/19/2021).

[68] gustavla, *Gpu resources not released when session is closed*. [Online]. Available: https://github.com/tensorflow/tensorflow/issues/1727 (visited on 04/23/2021).

[69] E. community, *Expo cli*. [Online]. Available: https://docs.expo.io/workflow/expo-cli/ (visited on 04/26/2021).

[70] Expo, *Eas build*. [Online]. Available: https://docs.expo.io/build/introduction/ (visited on 04/28/2021).

[71] vsupalov, *Gunicorn and nginx in a nutshell*. [Online]. Available: https://build.vsupalov.com/gunicorn-and-nginx/ (visited on 04/28/2021).

[72] D. Zarya, *The best programming language for ai: Your ultimate guide*, Jun. 2020. [Online]. Available: https://www.zfort.com/blog/best-programming-language-for-ai (visited on 05/15/2021).

[73] Savita Pahuja, *What is scrumban*, May 2017. [Online]. Available: https://www.agilealliance.org/what-is-scrumban/ (visited on 05/10/2021).

Appendix A

Group Rules

Group rules

Andrea Magnussen, Anders S. Langlie,
Eirik M. Danielsen and Herman A. Dyrkorn

January 2021

1 Rules

- Herman A. Dyrkorn is the group leader.
- Decisions in the group are done in a democratic way where the majority wins.
- The group leader has the ability to decide the outcome of a tied group vote, meaning that Herman A. Dyrkorn has two(2) votes in a tied vote.
- Each member has to work roughly 30 hours a week at least (this corresponds to the expected workload for a 20 study points course).
- We expect that all members show up to scheduled meetings and arrangements, unless it is notified beforehand.
- If a member refuses to work, he/she will first get a written warning. If this does not better the situation, we will try to get external help from the department. In an extreme case, the member can be removed from the group.
- Everyone has to show respect towards one another.
- Communication between members needs to be open and clear. This means that if there is a problem that prevents a given task from being completed, then every group member needs at the very least to be informed.
- We all have agreed to work towards a top grade, A, B or C.

This agreement is between Andrea Magnussen, Herman A. Dyrkorn, Eirik M. Danielsen and Anders S. Langlie.

Students signature:

[illegible]	13/01-21
Herman A. Dyrkorn	Date
[illegible]	13/01-21
Andrea Magnussen	Date
[illegible]	13/01-21
Eirik M. Danielsen	Date
[illegible]	13/01-21
Anders S. Langlie	Date

Appendix B

Project Agreement

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato   Vår referanse

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

NORSK INSTITUTT FOR NATURFORSKNING (NINA) (oppdragsgiver), og

HERMAN A. DYRKORN, EIRIK M DANIELSEN, ANDERS S. LANGLIE OG ANDREA MAGNUSSEN (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 12.01.21 til 20.05.21.

Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon, reiser og nødvendig overnatting på steder langt fra NTNU i Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle beståtte bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv NTNU Open.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): MARIUS PEDERSEN

Oppdragsgivers kontaktperson (navn): BØRRE DERVO

Student(er) (signatur): [illegible] dato 12.01.21

[illegible] dato 12.01.21

[illegible] dato 12.01.21

[illegible] dato 12.01.21

Oppdragsgiver (signatur): [illegible] dato 18/1-21

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): ______________________ dato __________

Appendix C

Task Description

Salamanders have a uniquely identifiable pattern located on their abdomen, which makes it possible to recognize each individual, similar to a human fingerprint. Currently, the Norwegian Institute for Nature Research (NINA), uses a Passive Integrated Transponder (PIT) placed under the skin, to identify individual salamanders that exist in Norway. This tagging method requires an invasive procedure, in addition to being expensive. Since the abdominal pattern is unique, it is possible to use it for identification instead. Our two species of salamander are red-listed, which makes it important to be able to monitor their population robustly and easily.

In the spring of 2019, a bachelor thesis was completed at NTNU, where a tool for the recognition of salamanders based on their abdominal pattern was developed[7]. The tool is based on machine learning, pattern recognition, and is written in Python. The tool was tested on a small data set and looks promising for the researchers at NINA. At this point, the tool is only usable through the command line, which is not an optimal solution with user experience in mind. It should also support the RAW image format. Additionally, the current matching algorithm should be improved as it is not robust. The ability to classify the salamanders based on sex and species is also wanted. Despite these weaknesses, the tool has gotten both national and international attention from researchers who wish for a tool to aid in fieldwork. Therefore, NINA wishes for a continuation of development for the tool, so it can be usable in the field. This project will result in software that can be used by the researchers at NINA.

The project consists of creating a graphical user interface (GUI), optimize usage of RAW pictures, a faster and better way to match salamanders, and classification of the salamander's sex and species.

Appendix D

Documentation

Front-end: https://git.gvk.idi.ntnu.no/eirikmda/salamander-app

Back-end: https://git.gvk.idi.ntnu.no/HermanDyrkorn/salamander-api

API-Documentation: https://documenter.getpostman.com/view/8860712/TzJvcbk9

Appendix E

Project Plan

Project Plan

Eirik Martin Danielsen
Anders Sundstedt Langlie
Andrea Magnussen
Herman Andersen Dyrkorn

May 6, 2021

Contents

Figures

Tables

Project Goals and Constraints

1.1 Background

This project was provided by Norwegian Institute for Nature Research (NINA), and our contact person from the institute is Børre Dervo. NINA is an independent foundation for nature research and research on the interaction between human society, natural resources, and biodiversity[1].

One of the fields of research at NINA is salamanders. Salamanders are a group of amphibians, who thrive in marshlands, open woodlands, and cultural landscapes, with sufficient access to water and hiding places. In Norway, there are two different species of salamander; northern crested newt and smooth newt[2]. Both of these species are red-listed, and NINA's goal is to monitor and ultimately increase their population.

Back in 2019, NINA issued a bachelor project where they requested a program that would be able to recognize a salamander based on a unique pattern located on its abdomen[3]. The previous bachelor group was able to make a system that allowed the user to match images of caught salamanders against a database of previously found animals. Unfortunately, they did not have enough time to make a proper graphical user interface and instead resorted to using command line arguments. This resulted in it not being actively used by the scientists at NINA.

For this project, NINA wants us to develop an easy to use- graphical user interface, using the existing system developed in the previous bachelor project. Additionally, NINA wants the ability to classify other traits eg. sex, weight, length, etc.

1.2 Project Goals

The goals are divided into result and effect goals, with effect goal again being divided into qualitative and quantitative goals.

1.2.1 Result Goals:

- The system should make it easier for scientists at NINA to conduct their research.
- A working web-server that uses the previous system and will communicate with portable devices used out in the field (mobile phones).
- A cross-platform mobile application that will work as a client to the server.
- The system should be able to classify the sex of the salamander with 95% accuracy.
- The system should be able to classify the salamander's species with 95% accuracy.
- Reduce the search time of an animal by 50% by dividing the search by sex.
- Reduce the search time by 5-50% by dividing the search by geographical location.
- The system must support a capacity of 5 scientists concurrently.
- The system has to work as long as there is an internet connection for both the client and the server.

1.2.2 Effect Goals:

- Qualitative goals:
  - The system should have a better user experience than the existing system.
  - The salamanders will be more carefully handled with the new system.
  - Increase the number of researchers tracking salamanders in the field, by eliminating the need for a special license.
- Quantitative goals:
  - Reduce the number of work hours for the researchers at NINA by 75%.
  - The new system should substitute the amount of PIT-tagging by 20% each year.
  - The system will reduce NINA's cost toward salamander research by 30%.

1.3 Constraints

1.3.1 Time Constraints:

- The product needs to be complete before 20.5.2021.

- If the system will run at NINA, we need a server at their location within April, to do some proper user testing.

1.3.2 Hardware and Software Constraints:

- The software identifying the animals and determining their sex is a heavy program running on a heavy language that takes a lot of electricity and processing power.
- The previous system was written in Python, so we need to continue using this language to improve the algorithm.
- Since the image processing algorithm requires quite powerful hardware, it will be limited to only run on a dedicated server.
- Since the system will be utilized in fieldwork, the hardware needs to be lightweight and easily portable.
- The light outside needs to be bright enough for a mobile camera to take a good quality picture.
- Poorly taken photos where there are particles between the camera and the animal, the animal is not straightened or the picture was taken with flash enabled will inhibit the matchmaking.

1.3.3 Legal Constraints:

- If a user can store any personal information that operation must apply to General Data Protection Regulation (GDPR)[4].

Scope

2.1 Subject Area

The salamander species that NINA are researching are the northern crested newt and the smooth newt. The northern crested newt is red listed[5], and the smooth newt while not red listed has decreased in population[2]. It is therefore crucial to monitor changes in the population. By monitoring the number of individuals, one can get a good indicator on whether the active measures in place to aid in increasing the population is working. In order to estimate the population the capture-mark-recapture method is used. We can look at the ratio between the individuals that have been caught previously and the ones that have not. By using the Lincoln-Peterson method[6] we can then estimate the population.

- item N is the total number of salamanders in the population.
- N is the total number of salamanders in the population.
- n is the number of salamanders that were marked on the first visit.
- k is the number of recaptured salamanders that were marked.

$$\frac{N}{n} = \frac{K}{k} \tag{2.1}$$

If we rearrange, we see that we can estimate the total population N̂ using the following equation:

$$\hat{N} = \frac{Kn}{k} \tag{2.2}$$

2.2 Task Description

Salamanders have a uniquely identifiable pattern located on their abdomen, which makes it possible to recognize each individual, similar to a human fingerprint. Currently, the Norwegian Institute for Nature Research (NINA), uses a Passive Integrated Transponder (PIT) placed under the skin, to identify individual salamanders that exist in Norway. This tagging method requires an invasive procedure, in addition to being expensive. Since the abdominal pattern is unique, it is possible to use it for identification instead. Our two species of salamander are red-listed, which makes it important to be able to monitor their population robustly and easily.

In the spring of 2019, a bachelor thesis was completed at NTNU, where a tool for the recognition of salamanders based on their abdominal pattern was developed[3]. The tool is based on machine learning, pattern recognition, and is written in Python. The tool was tested on a small data set and looks promising for the researchers at NINA. At this point, the tool is only usable through the command line, which is not an optimal solution with user experience in mind. It should also support the RAW image format. Additionally, the current matching algorithm should be improved as it is not robust. The ability to classify the salamanders based on sex and species is also wanted. Despite these weaknesses, the tool has gotten both national and international attention from researchers who wish for a tool to aid in fieldwork. Therefore, NINA wishes for a continuation of development for the tool, so it can be usable in the field. This project will result in software that can be used by the researchers at NINA.

The project consists of creating a graphical user interface (GUI), optimize usage of RAW pictures, a faster and better way to match salamanders, and classification of the salamander's sex and species.

2.3 Delimitation

In this thesis, we will only be focusing on making a system for NINA and not for an international user base. The software will be deployed on a server stationed locally at NINA, and will not be running in a cloud service e.g. AWS or Azure. We will not implement measures to detect if a taken image is an image of a salamander. The program will not be able to work with pictures of poor quality. The react native framework's minimum requirements are iOS 10.0 or Android 4.1 (API 16)[7]. However, we have decided that our application should require iOS 14.0 on an iOS device, and Android 8.0 Oreo on an android device. Lastly, due to our decision to use pictures taken with mobile cameras, we will only focus on using PNG and JPG/JPEG image formats.

Project Organization

3.1 Responsibilities and Roles

Figure 3.1 shows an overview of our team structure and group responsibilities.

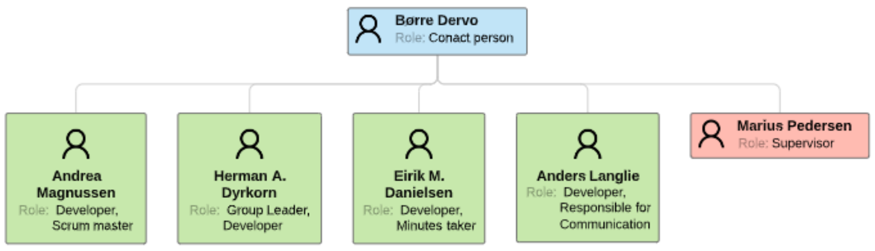

Børre Dervo
Role: Conact person
Andrea Magnussen
Role: Developer, Scrum master
Herman A. Dyrkorn
Role: Group Leader, Developer
Eirik M. Danielsen
Role: Developer, Minutes taker
Anders Langlie
Role: Developer, Responsible for Communication
Marius Pedersen
Role: Supervisor

Figure 3.1: The map shows the different roles in the bachelor project.

3.2 Rules and Routines

- Herman A. Dyrkorn is the group leader.
- Decisions are done in a democratic way where the majority wins.
- The group leader has the ability to decide the outcome of a tied group vote, meaning that Herman A. Dyrkorn has two(2) votes in a tied vote.
- Each member has to work roughly 30 hours a week at least (this corresponds to the expected workload for a 20 study points course).
- We expect that all members show up to scheduled meetings and arrangements, unless it is notified beforehand.
- If a member refuses to work, he/she will first get a written warning. If this does not better the situation, we will try to get external help from the department. In an extreme case, the member can be removed from the group.
- Everyone has to show respect towards one another.
- Communication between members needs to be open and clear. This means that if there is a problem that prevents a given task from being completed, then every group member needs at the very least to be informed.
- We all have agreed to work towards a top grade, A, B or C.

Planning, Followup and Reporting

4.1 Development Process

4.1.1 Characteristics of Project

This project is a continuation of a previous bachelor's project from 2019[3]. We will have to incorporate the code from the previous bachelor, as well as improving and adding new functionality to it in this project. The system will consist of independent components: The image recognition Algorithm, a web server API, and a mobile app connected to the server. These components can be developed independently in our team of four people.

Considering the project deadline is the 20th of May, including thesis writing, the system needs to be developed in a short amount of time. The requirements from NINA are also quite ambiguous, especially in regards to design. This means that we will have to refine the requirements during the development process.

One of the most important tasks in this project is to develop a graphical user interface (GUI), so that the algorithm can be used by the scientists at NINA. They are not software developers and some of the scientists may lack technical experience in regards to computer science. Therefore we need to establish a close collaboration between NINA and us, especially in regards to design and functionality.

The members of our group do not have that much experience when it comes to a project of this scale. This makes it hard to plan the entire project before it starts and also hard to estimate the time needed to implement each task. However, if the final system lacks minor features, it can still be useful for the researchers at NINA.

All of these characteristics will need to be taken into consideration when choosing a software our development model.

4.1.2 Software Development Model

Based on the characteristics in Section 4.1.1, we have concluded that we need an agile software development model (SDM)[8]. Relevant SDM's includes Kanban and Scrum. Both models have advantages and disadvantages in regards to a bachelor's project setting.

Kanban is an agile SDM where you visualize the whole project by dividing the project work into smaller tasks and placing them on a Kanban board[9]. The developers are free to choose tasks from the board and there are no specific roles in the developer team. This leads to Kanban being highly flexible[10]. Such flexibility could be an advantage for our group, considering the loose boundaries in the project description. On the other hand, this flexibility can lead to hard or boring tasks getting ignored, if none of the group members takes the responsibility to do it[11]. Besides, it might be more difficult for the group members to follow up on each other's contribution and workflow, if there are no set deadlines and planned meetings[12].

Scrum is an agile SDM that is based on incremental development, where the entire project is divided into sprints with set deadlines[13]. Each sprint will usually have a set length between two to four weeks. Roles, meetings, and other tools aids this methodology in achieving structure and managing workload. Having proper structure and good routines could be especially helpful for us in our team, as we are quite inexperienced with projects of this scale. An important factor for us is that Scrum is suitable for smaller teams[13]. By utilizing daily Scrum, all members of the group will have a good overview of what everyone is doing. On the other hand, we are currently living in a pandemic, which means that we are not able to have physical meetings with our supervisor and product owner (PO). This can hinder the full potential of having sprint planning meetings, sprint review meetings, and sprint retrospective meetings. Although fixed meetings contribute to maintaining structure, there may be occasions where they are not necessary. As a consequence, time may be wasted.

After looking at both of these models, we have concluded that Scrum is a fitting SDM for this project, as it can help our group maintaining proper structure. We also want to incorporate some features from Kanban, like the kanban-board, for keeping track of the project backlog and sprint backlog. If necessary, we will bring in pair programming from eXtreme Programming[14]. Lastly, we concluded that the inability to have physical meetings with our supervisor and the PO, does not significantly harm this project, as online meetings will be adequate.

4.1.3 Usage of Software Development Model

Andrea Magnussen will be the scrum master on our team. The PO is Børre Dervo, as he is the contact person from NINA. We have decided on sprint lengths of two weeks per sprint, as the project timeline is quite short. By splitting each sprint into two weeks, we will at least be able to finish four sprints during the project period. By having short sprints, it will keep our PO frequently updated on the development of the system. The sprints will start with a sprint planning meeting on the first Monday, and end on the second Friday with a sprint review and retrospective meeting. The PO will be included in the sprint review meeting. Every day, there will be a daily scrum. It will be timed and it will last 15 minutes, where each team member updates each other on what they have worked on. The scrum master will time the meeting. The entire product backlog will be tracked using Trello as a Kanban board. We will have one board containing to process of all the tasks in the entire backlog and one board for each sprint.

4.2 Plan for Meetings and Decision Points

By the end of April, we need to be finished with a product that satisfies NINA's needs. The final deadline for the entire project, software and thesis, is the 20Th of May. The meeting with Børre Dervo before every sprint will be used for making user stories and get a picture of what needs to be made during the upcoming sprint. Meetings with Marius Pedersen on Tuesdays will be used to get guidance in progressing the product development, and also for asking questions about technologies, workflow, and report writing. See Table 4.1 for an overview of what a sprint looks like.

Table 4.1: Sprint structure

Week	Mon	Tue	Wed	Thu	Fri
1	Sprint plan	Daily scrum Meeting	Daily scrum	Daily scrum	Daily scrum
2	Daily scrum	Daily scrum Meeting	Daily scrum	Daily scrum	Daily scrum Sprint retro Sprint review

Quality Assurance

5.1 Documentation and Source Code

5.1.1 Documentation

After completing our bachelor's, there exists a possibility that someone will have to re-visit our code. It is therefore important that we document the functionality of the system, as well as our process a long the way. For code documentation, we will mostly use commits is GitLab. Therefore, it is imperative that each member commits regularly with satisfactory commit messages. All meetings will be documented by writing a small report that explains what the meeting was about, its participants, and the duration of the meeting. Finally we will create a manual for deploying and using the server, as well as a manual for use of the mobile application.

5.1.2 Source Code

All source code needs to follow code standards for their respective language. For python we will be using PEP 8 code standard[15]. By following these standards, it will make it easier for others to read the code. All written code should have sufficient commenting, so other members in the group can read and understand it, as well as for documentation purposes. It is required that all functions have a comment explaining what the function does.

5.2 Configuration Management

Standard Tools

- Overleaf will be used for document writing in the bachelor's thesis. This includes the project plan and the main thesis.
- Trello will be used as a Kanban board for visualizing the product backlog. The board will also work as an issue tracker, where we can keep track of the status of each task.
- We will be using GitLab for version control. There will be a total of three separate GitLab projects. One for the API, one for the mobile application,

and one for the salamander matching algorithm.
- We will be using Clockify for logging hours. This will give us a good indication if we have worked enough or not. This will also give us an overview of what we are spending our time on.
- For the salamander matching algorithm we will be using Python. Python will also be used to create the REST API, with the Flask library. The IDE for Python development will be PyCharm.
- React Native will be used to develop the mobile application. The IDE for developing the mobile app will be Visual Studio Code, with appropriate addons and extensions.
- We will be using APIDOC for documenting the REST API.
- For the AI we will use OpenCV, Tensorflow and Deeplabcut because that is what the previous group used in their program. We might have to use other libraries as well.
- Grammarly will aid us in our writing, and ensure the written report reads well.

Figure 5.1 visualizes how all the tools will be working together in our development environment.

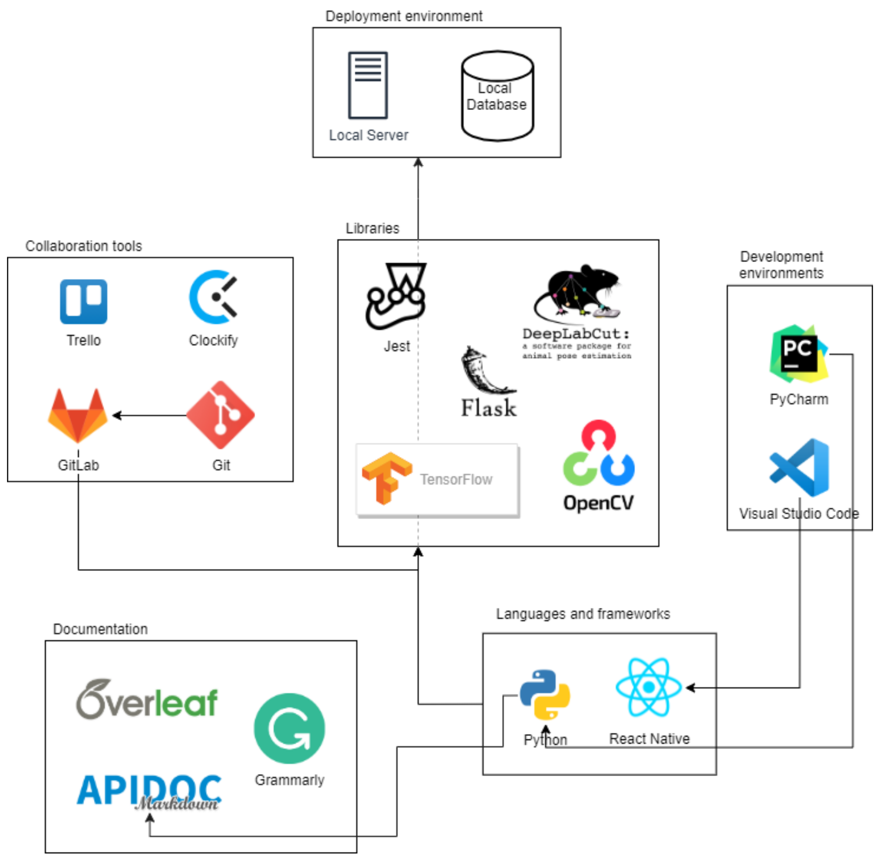


Figure 5.1: An overview of the different tools we will utilize in the project.

5.3 Risk Analysis

Table 5.1 shows the different degrees of likelihood and consequence for all risks. In Table 5.2 we have described eight risks, and given them a degree of likelihood and consequence. In Table 5.3, we have made a prioritization of the risks, and also come up with a mitigation that will lower the risk.

Table 5.1: Risk standardization

Likelihood/ Consequence	**Minimal**	**Minor**	**Moderate**	**Significant**	**Critical**
highly likely	5	10	15	20	25
likely	4	8	12	16	20
probable	3	6	9	12	15
unlikely	2	4	6	8	10
highly unlikely	1	2	3	4	5

Table 5.2: Risks

Risk	**Description**	**Likelihood/ Consequence**
1	We will not be able to do any physical user testing due to covid-19.	**Likely/ Significant**
2	The system performs badly and has a lot of bugs	**Probable/ Significant**
3	Overestimating our capabilities in terms of project size, tools or complexity	**Likely/ Moderate**
4	NINA does not provide any images for the project	**Unlikely/ Critical**
5	We are not able to get the AI to work.	**Unlikely/ Critical**
6	One of the group members catches Covid-19	**Probable/ Moderate**
7	Group member gets critically sick or injured and will not be able to contribute to the project	**Unlikely/ Significant**
8	Loss of important data	**Unlikely/ Significant**
9	Someone beats us to market with a better solution	**Highly unlikely/ Minor**

Table 5.3: Mitigation

Priority	Risk	Mitigation
High	1	In case of travel restrictions or other problems stopping us from conducting physical user testing, we can make the testing phase digital. This can allow us to observe the user without being physically present.
High	2	We need to cover all important functions with sufficient unit testing. We should include NINA in the testing phase, to ensure that the product performs as anticipated.
High	3	We need to uphold deadlines to the best of our ability, and hold frequent meetings.
High	4	We need to set requirements for NINA, as well as maintaining good communication and frequent meetings.
High	5	Allocate and prioritize enough workload and members towards the AI development.
Medium	6	If a member catches covid-19, the member needs to work from home as long as the member is in good physical shape.
Medium	7	Hard to mitigate this risk. Good documentation and daily scrum will aid the others in stepping in for the member affected.
Medium	8	Backups and online storage of files and project, with systems like git and overleaf.
Low	9	Hard to mitigate, but very unlikely. We also have a signed contract with NINA.

Development Plan

6.1 GANTT - Diagram

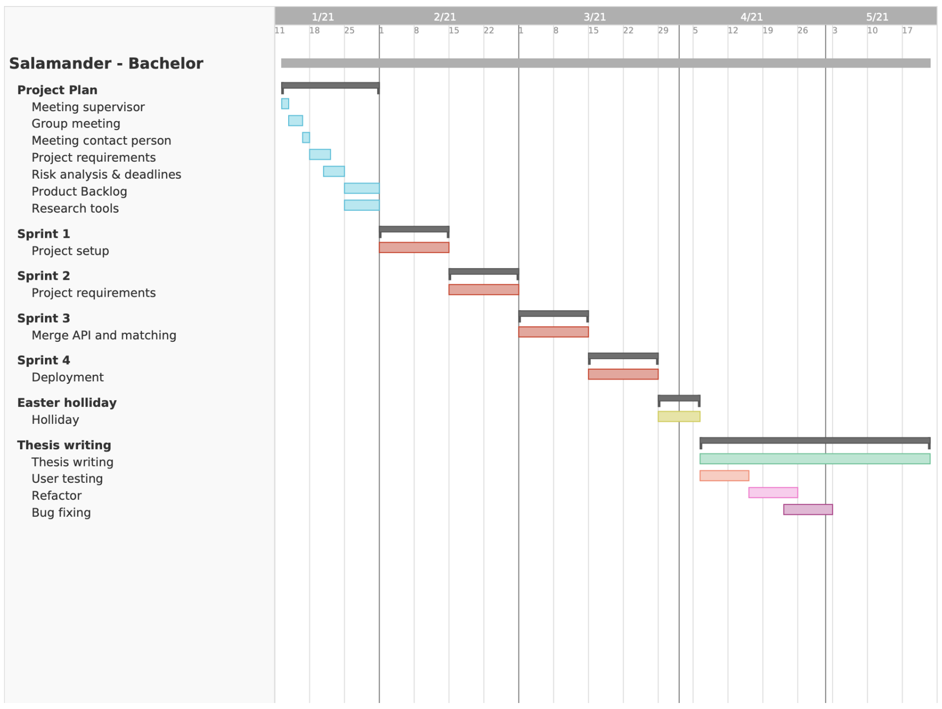


Figure 6.1: Gantt diagram.

In Figure 6.1, the upper gray bar represents the whole project period. During this time we will be developing our software solution, and also writing our report. During the last milestone we will be focusing more on thesis writing than software development. The reason we have chosen to not extend this last period (particularly the green bar) is to emphasize the increase in the work on writing over development. By the 3rd of May, we will be completely finished with development and only focus on finalizing the written report.

6.2 Milestones, Decision Points and Activities

6.2.1 Milestones

1. 31/1: Finish project plan, product backlog, and project agreement.
2. 12/2: Finish sprint 1.
3. 26/2: Finish sprint 2.
4. 12/3: Finish sprint 3.
5. 26/3: Finish sprint 4.
6. 29/3: Easter break.
7. 14/4: Completed user testing in the field.
8. 3/5: Finished with developing the system.
9. 20/5: Bachelor delivered in Inspera.
10. June: Bachelor presentation.

6.2.2 Decision points

The beginning and end of each sprint are the most significant decision points throughout the project. If we progress faster than planned, we will have to add more tasks to the current sprint backlog. On the other hand, if we do not manage to complete all tasks in the current sprint backlog, we will have to prioritize and choose tasks from the sprint backlog and include them in the next one. Also, the tasks themselves, and their priority might be changed depending on new requirements and problems that might occur.

6.2.3 Activities

There will be three crash courses during the bachelor´s project period. All group members are obligated to join these.

During sprint planning meeting, we will use planning poker to estimate how much time each task will take. This will give us a good indication on what to include in the sprint backlog.

To boost motivation and team morale, we have decided that the team should do something unrelated to work together, at least once a month. This can include activities like bowling, field trips etc.

Bibliography

[1] *Nina - the norwegian institute of nature research*. [Online]. Available: `https://www.nina.no/english/Home` (visited on 01/15/2021).

[2] B. K. Dervo, *Salamander*. [Online]. Available: `https://www.nina.no/salamander` (visited on 01/15/2021).

[3] J. Bakløkken, F. Schoeler, and H. Nørholm, *Automated salamander recognition using deep neural networks and feature extraction*, May 2019. [Online]. Available: `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2617897` (visited on 01/15/2021).

[4] *Lov om behandling av personopplysninger (personopplysningsloven)*, 2018. [Online]. Available: `https://lovdata.no/dokument/NL/lov/2018-06-15-38` (visited on 01/15/2021).

[5] Artsdatabanken, *Norsk rødliste for arter*, Mar. 2020. [Online]. Available: `https://artsdatabanken.no/Rodliste` (visited on 11/24/2021).

[6] Wikipedia contributors, *Lincoln index — Wikipedia, the free encyclopedia*, [Online; accessed 26-January-2021], Nov. 2020. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Lincoln_index&oldid=991124768` (visited on 01/26/2020).

[7] NPM, *React-native - npm*, Nov. 2020. [Online]. Available: `https://www.npmjs.com/package/react-native` (visited on 01/26/2021).

[8] W. contributors, *Agile software development — Wikipedia, the free encyclopedia*, Jan. 2021. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=999545034` (visited on 01/26/2021).

[9] Wikipedia contributors, *Kanban (development) — Wikipedia, the free encyclopedia*, Jan. 2021. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Kanban_(development)&oldid=1002634912` (visited on 01/26/2021).

[10] M. REHKOPF, *Kanban vs. scrum: Which agile are you?* [Online]. Available: `https://www.atlassian.com/agile/kanban/kanban-vs-scrum` (visited on 01/26/2021).

[11] M. Gorman, *Scrum vs kanban: Weighing their pros and cons*. [Online]. Available: `https://www.kovair.com/blog/scrum-vs-kanban-pros-and-cons/` (visited on 01/26/2021).

[12] T. Hristovski, *Agile methodologies: Kanban vs scrum – advantages and disadvantages*, Dec. 2017. [Online]. Available: `https://iwconnect.com/agile-methodologies-scrum-vs-kanban-advantages-disadvantages/` (visited on 01/26/2021).

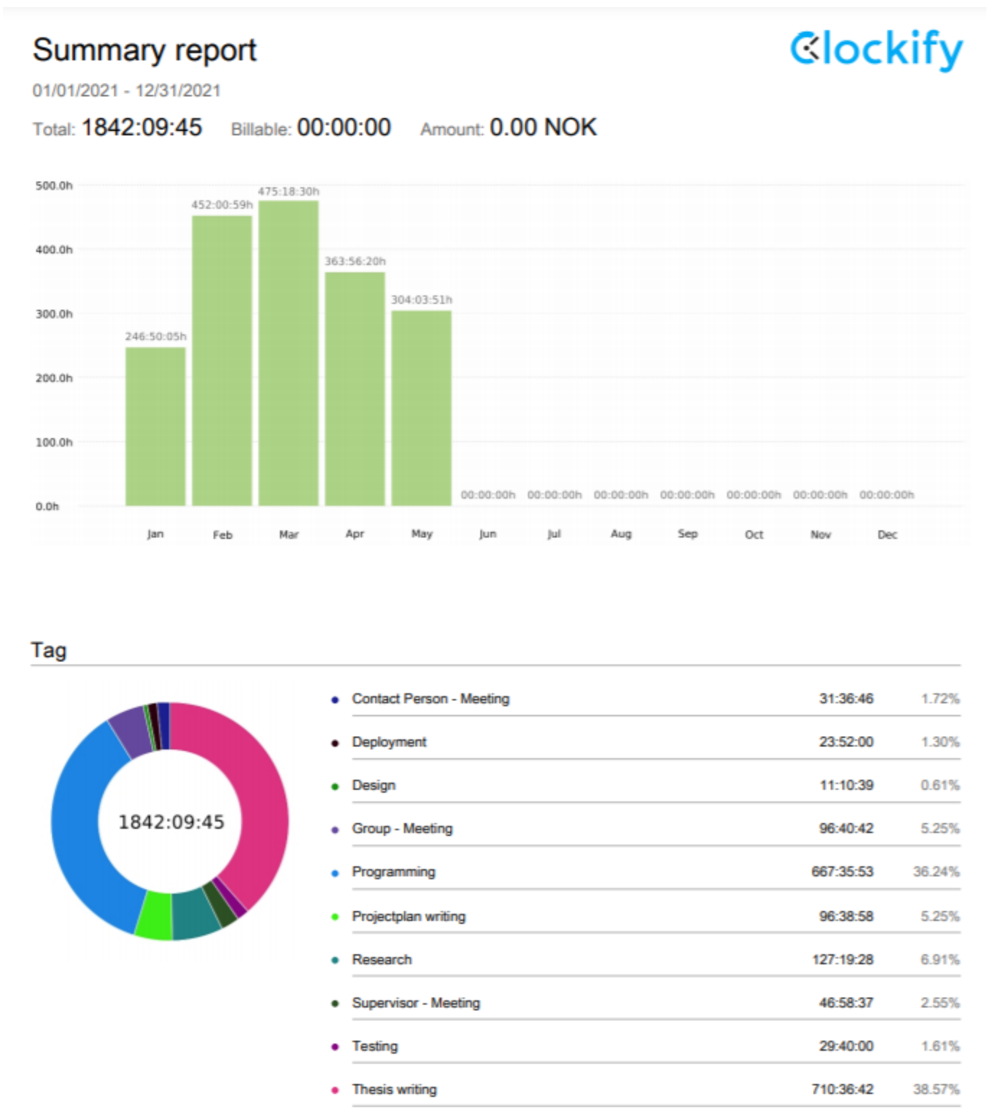
[13] Wikipedia contributors, *Scrum (software development) — Wikipedia, the free encyclopedia*, Jan. 2021. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Scrum_(software_development)&oldid=999915311` (visited on 01/26/2021).

[14] L. C. Team, *What is extreme programming? an overview of xp rules and values*. [Online]. Available: `https://www.lucidchart.com/blog/what-is-extreme-programming` (visited on 01/26/2021).

[15] G. van Rossum, B. Warsaw, and N. Coghlan, *Pep 8 – style guide for python code*, Aug. 2013. [Online]. Available: `https://www.python.org/dev/peps/pep-0008/` (visited on 01/26/2021).

Appendix F

Detailed Work Allocation



Summary report

01/01/2021 - 12/31/2021

Total: 1842:09:45 Billable: 00:00:00 Amount: 0.00 NOK

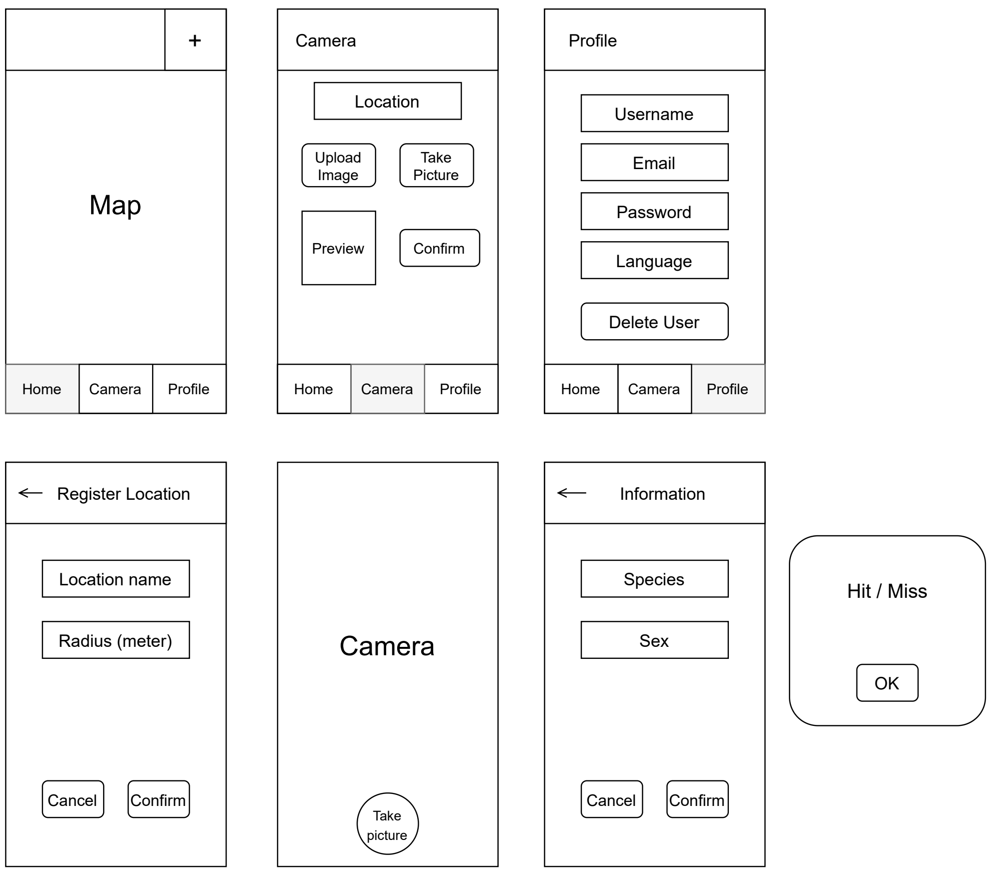
Tag

●	Contact Person - Meeting	31:36:46	1.72%
●	Deployment	23:52:00	1.30%
●	Design	11:10:39	0.61%
●	Group - Meeting	96:40:42	5.25%
●	Programming	667:35:53	36.24%
●	Projectplan writing	96:38:58	5.25%
●	Research	127:19:28	6.91%
●	Supervisor - Meeting	46:58:37	2.55%
●	Testing	29:40:00	1.61%
●	Thesis writing	710:36:42	38.57%

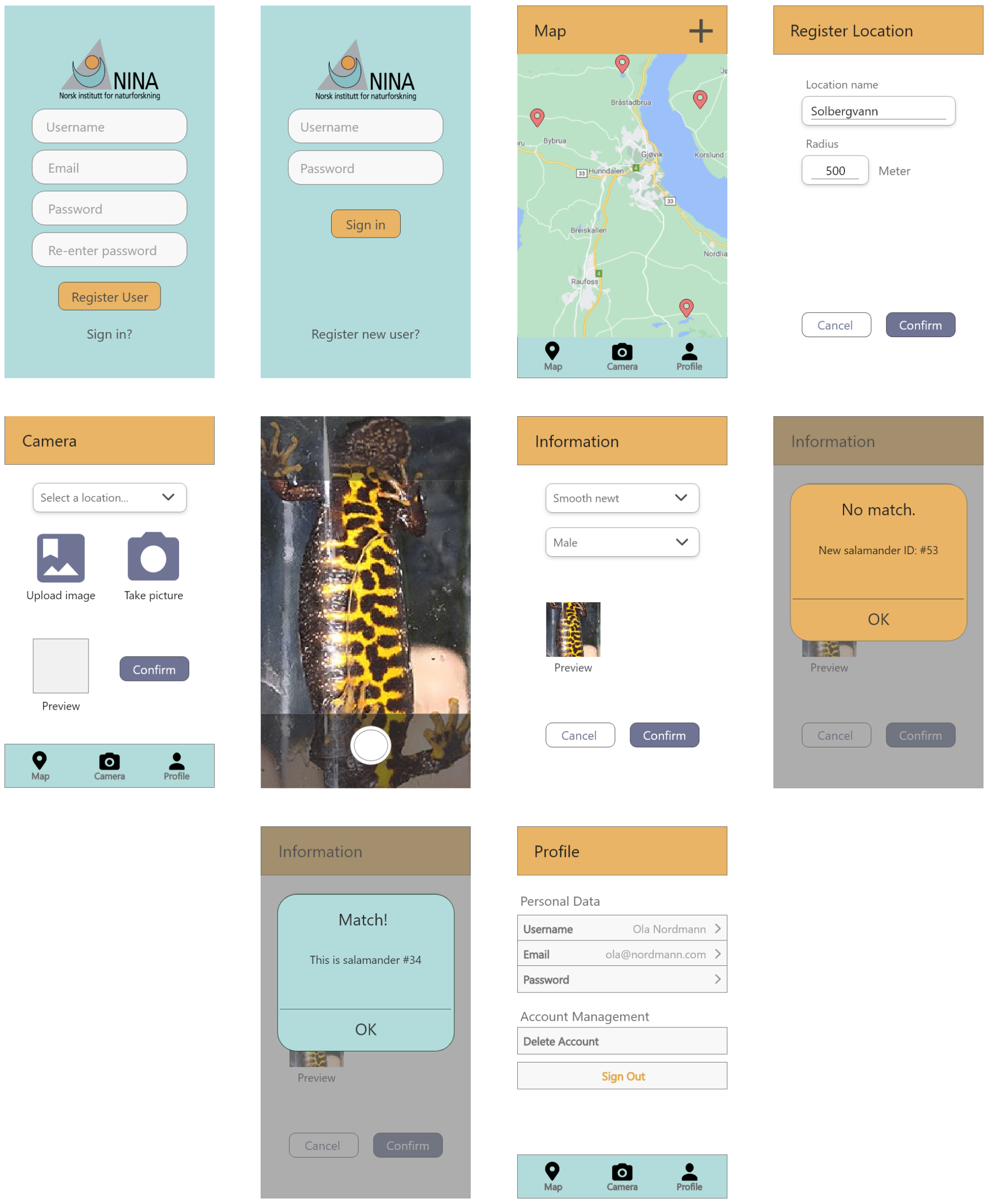
Appendix G

Prototypes

G.1 Simple Prototype



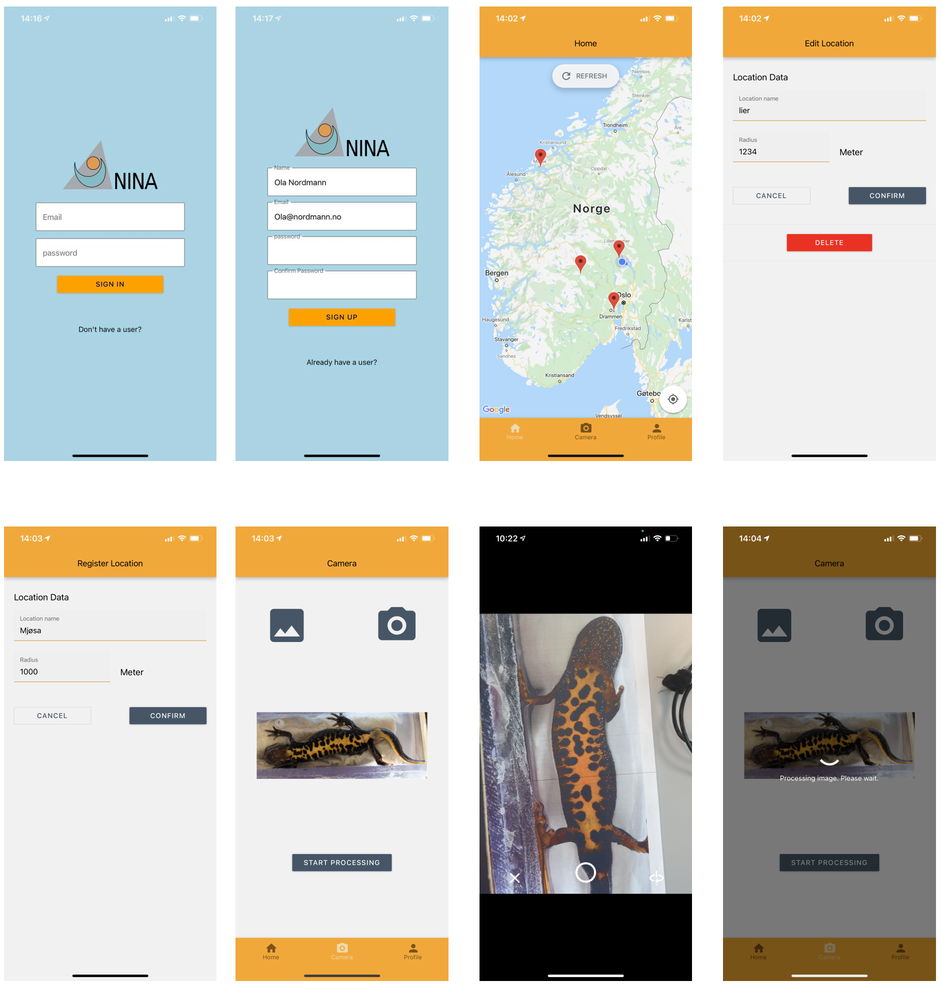
G.2 Adobe XD Prototype

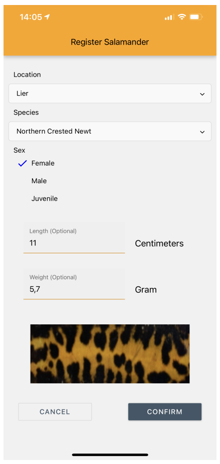


The digital interactive prototype can be found at `https://xd.adobe.com/view/f0b5d44c-1b8f-41e4-ba80-79c6c41d332d-a5e6/`.

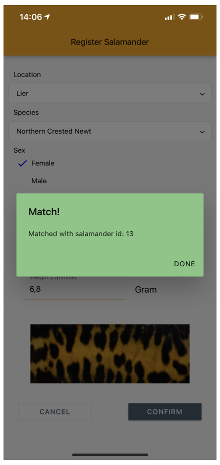
Appendix H

Final Graphical User Interface

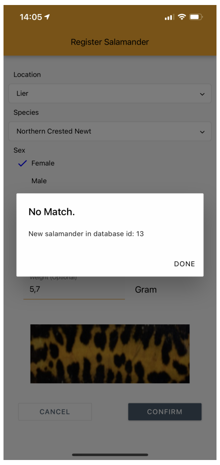




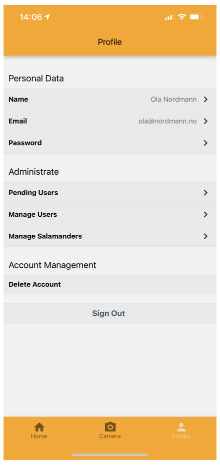
Location
Lier
Species
Northern Crested Newt
Sex
✓ Female
Male
Juvenile
Length (Optional)
11 Centimeters
Weight (Optional)
5,7 Gram



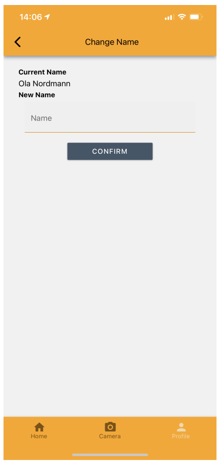
Location
Lier
Species
Northern Crested Newt
Sex
✓ Female
Male
Match!
Matched with salamander id: 13
Weight (Optional)
6,8 Gram



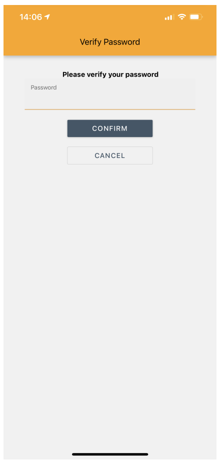
Location
Lier
Species
Northern Crested Newt
Sex
✓ Female
Male
No Match.
New salamander in database id: 13
Weight (Optional)
5,7 Gram



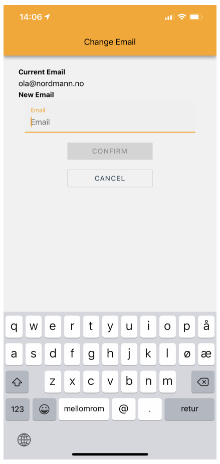
Personal Data
Name Ola Nordmann
Email ola@nordmann.no
Password
Administrate
Pending Users
Manage Users
Manage Salamanders
Account Management
Delete Account
Sign Out



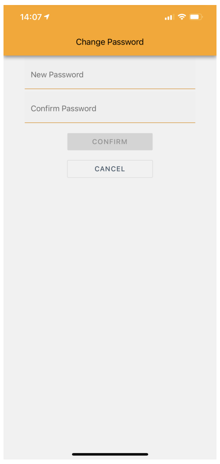
Current Name
Ola Nordmann
New Name
Name



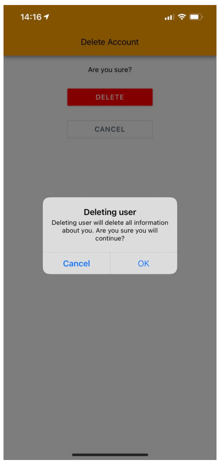
Please verify your password
Password



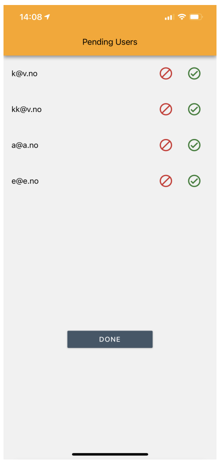
Current Email
ola@nordmann.no
New Email
Email



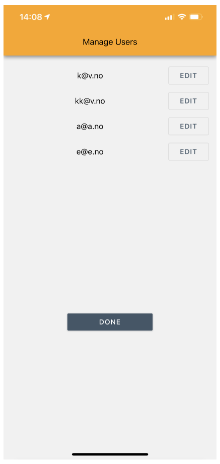
New Password
Confirm Password



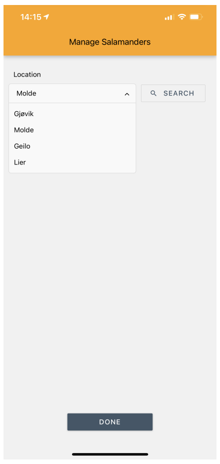
Are you sure?
Deleting user
Deleting user will delete all information about you. Are you sure you will continue?



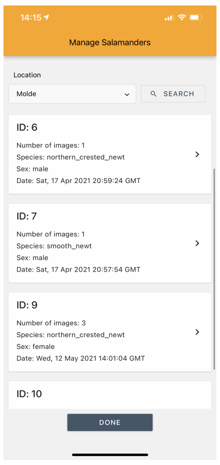
k@v.no
kk@v.no
a@a.no
e@e.no



k@v.no
kk@v.no
a@a.no
e@e.no



Location
Molde
Gjøvik
Molde
Geilo
Lier


14:15

Manage Salamanders

Location

Molde

SEARCH

ID: 6

Number of images: 1
Species: northern_crested_newt
Sex: male
Date: Sat, 17 Apr 2021 20:59:24 GMT

ID: 7

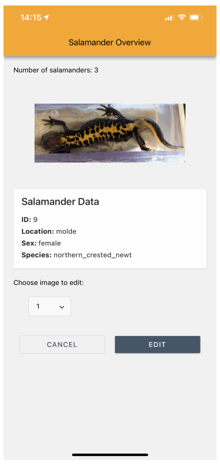
Number of images: 1
Species: smooth_newt
Sex: male
Date: Sat, 17 Apr 2021 20:57:54 GMT

ID: 9

Number of images: 3
Species: northern_crested_newt
Sex: female
Date: Wed, 12 May 2021 14:01:04 GMT

ID: 10

DONE


14:15

Salamander Overview

Number of salamanders: 3

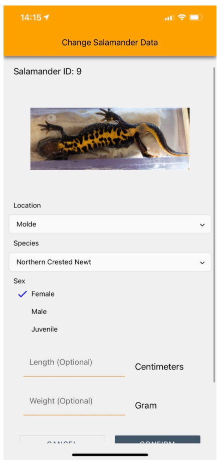
Salamander Data

ID: 9
Location: molde
Sex: female
Species: northern_crested_newt

Choose image to edit:

1

CANCEL EDIT


14:15

Change Salamander Data

Salamander ID: 9

Location

Molde

Species

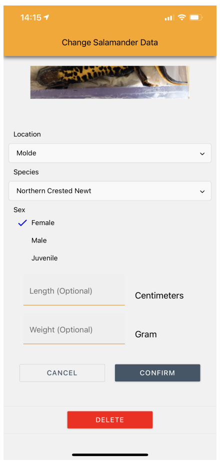
Northern Crested Newt

Sex

✓ Female
Male
Juvenile

Length (Optional) Centimeters

Weight (Optional) Gram


14:15

Change Salamander Data

Location

Molde

Species

Northern Crested Newt

Sex

✓ Female
Male
Juvenile

Length (Optional) Centimeters

Weight (Optional) Gram

CANCEL CONFIRM

DELETE

Appendix I

Testing

I.1 User Test Guide

Usability Testing Guide

Scope	Testing the salamander application. This test will cover the usability and first impression of the application.
Purpose	Questions: • Can the user easily register a user? • Can the user add a new location without problems? • Is it easy to find the upload image/take picture buttons? • Is the user able to navigate the application with ease? • Is the user experiencing any annoyances during periods of waiting? • Does the interface feel natural/intuitive? • Does the user feel unnecessarily hindered by security checks? • Does the user understand the feedback? • Can the admin accept pending users easily? Goals: • Get feedback from the user for development iteration. • Understanding how the user interacts with the app. • To find major flaws with the user interface. • Discover potential overlooked bugs or flaws of any level of concern. • Uncover security flaws of any level of concern.
Schedule & Location	Various
Sessions	Maximum 1 hour per participant
Equipment	Android/iOS phone
Participants	Researchers at NINA, regular users
Scenarios	• Register User • Log In • Add Location • Edit Location • Take Picture • Match Salamander • Edit Personal Data • Log Out • Delete User • Manage Users • Manage Salamanders
Metrics	Qualitative survey after testing
Quantitative metrics	Time on task, successful completion rates
Roles	Researcher, Non-researcher

I.2 Usability Tests

User Tests

Test: 1

Role	Non-Researcher
Place (city)	Molde
Date	17.04.2021
Age	55

Name	Register User
Successful Task Completion	
Critical errors	None
Non-critical errors	• small text (phone setting had small text) • Server wasn't running and therefore the user couldn't register. User didn't understand what to do when the server wasn't running.
Time On Task	4 minutes including server restart.

Name	Log in
Successful Task Completion	
Critical errors	None
Non-critical errors	Email input was cap sensitive, and the user didn't know how to change cap size to lower case.
Time On Task	2 minutes

Name	Add location
Successful Task Completion	
Critical errors	None
Non-critical errors	Tapped on the map instead of holding the finger on the map. Should have been given a toast/hint asking if the user would want to add a location. Navigated to several screens before receiving a hint.
Time On Task	02 minute and 18 seconds

Name	Edit Location
Successful Task Completion	
Critical errors	None
Non-critical errors	None
Time On Task	15 seconds

Name	Take Picture
Successful Task Completion	
Critical errors	None
Non-critical errors	User pressed “confirm” because they thought the app asked if they really wanted to use that image. Being left with a buffering screen without any feedback was confusing. Should have said “straighten image” rather than “confirm”
Time On Task	8 seconds

Name	Match Salamander
Successful Task Completion	
Critical errors	None
Non-critical errors	User was confused with the toast message. When registering a salamander “No match” could be misunderstood, because to register a new one. This confusion could have been because the task was to “Match a salamander” rather than “Register salamander”. The toast color was orange and the user thought that something wrong had happened.
Time On Task	1 minute

Name	Edit Personal Data
Successful Task Completion	
Critical errors	None
Non-critical errors	None
Time On Task	10 seconds

Name	Log Out

Successful Task Completion	
Critical errors	None
Non-critical errors	There should be a confirmation that you really want to log out.
Time On Task	3 seconds

Name	Manage Users / Pending Users
Successful Task Completion	
Critical errors	Why can't you delete other users.
Non-critical errors	Hard to see accept/decline icons. Could have been text rather than icons. Both icons were light orange, but they could have been different colors (green/red) Manage Users were confusing because there were no users in the system. There could have been a text that states that there are no users. Annoying to write your password each time. Unnecessary when you have a session/token. confusing that there were 3 radial buttons for changing the user permission and then "reset password" was just a yellow button.
Time On Task	1 min / 1 min

Name	Manage Salamanders
Successful Task Completion	
Critical errors	None
Non-critical errors	It was confusing when a salamander was moved because the user didn't understand that a salamander was registered. The fact that the system started matching was confusing. There should have been a clearer message rather than "salamander moved".
Time On Task	1 min

Name	Delete User
Successful Task Completion	
Critical errors	None
Non-critical errors	Because the user was admin it was confusing to understand who you were deleting.
Time On Task	30

Test: 2

Role	Non-Researcher
Place (city)	Molde
Date	17.04.2021
Age	57

Name	Register User
Successful Task Completion	
Critical errors	None
Non-critical errors	None
Time On Task	1 minute and 38 seconds

Name	Log in
Successful Task Completion	
Critical errors	None
Non-critical errors	None
Time On Task	1 second

Name	Add location
Successful Task Completion	
Critical errors	None
Non-critical errors	The was not able to add a location without explanation because the user experience is not self explanatory. There should be some form of instruction on how to add a location. The rest was okay.
Time On Task	1 minute and 49 seconds

Name	Edit Location
Successful Task Completion	
Critical errors	
Non-critical errors	None
Time On Task	10 seconds

Name	Take Picture
Successful Task Completion	
Critical errors	The user thought the “Confirm” button was meant to confirm that the image was the one they wanted to keep. They didn’t know that it started the straightening process.
Non-critical errors	None
Time On Task	

Name	Match Salamander
Successful Task Completion	
Critical errors	None
Non-critical errors	“No match” message confused the user. They thought it meant that the image they sent in didn’t match a male, northern crested newt. Doesn’t explain what no match means.
Time On Task	20 seconds

Name	Edit Personal Data
Successful Task Completion	
Critical errors	None
Non-critical errors	None
Time On Task	10 seconds

Name	Log Out
Successful Task Completion	

Critical errors	None
Non-critical errors	None
Time On Task	5 seconds

Name	Manage Users
Successful Task Completion	
Critical errors	None
Non-critical errors	Annoyed that they have to enter their password again.
Time On Task	20 seconds

Name	Manage Salamanders
Successful Task Completion	
Critical errors	None
Non-critical errors	Annoyed that they have to enter their password again.
Time On Task	20 seconds

Name	Delete User
Successful Task Completion	
Critical errors	None
Non-critical errors	Annoyed that they have to enter their password again.
Time On Task	10 seconds

Test: 3

Role	Non-Researcher
Place (city)	Oslo
Date	18.04.2021
Age	23

Name	Register User
Successful Task Completion	Managed to create a user.
Critical errors	None
Non-critical errors	Tried to create a user in login.
Time On Task	1 min, 14 sec

Name	Log in
Successful Task Completion	Success
Critical errors	none
Non-critical errors	none
Time On Task	3 sec

Name	Add location
Successful Task Completion	Understood long press.
Critical errors	None
Non-critical errors	Took some time after double press to press it again.
Time On Task	40 sec

Name	Edit Location
Successful Task Completion	Success

Critical errors	Changed the location name to: :-). This could be an critical error
Non-critical errors	None
Time On Task	23

Name	Take Picture
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	6 sec

Name	Match Salamander
Successful Task Completion	Success, no match on first salamander. Second salamander was a match.
Critical errors	None
Non-critical errors	None
Time On Task	58 sec

Name	Edit Personal Data
Successful Task Completion	Success. The helper text helped during the test.
Critical errors	None
Non-critical errors	Struggled to get the passwords matching.
Time On Task	2 min, 30 sec

Name	Log Out
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	1 sec

Name	Manage Users
Successful Task Completion	Success at accepting users. Made uses administrators.

Critical errors	None
Non-critical errors	None
Time On Task	1 min

Name	Manage Salamanders
Successful Task Completion	Edited weight and length of the salamander. Indication that there are more images.
Critical errors	None
Non-critical errors	None
Time On Task	1 min, 16 sec

Name	Delete User
Successful Task Completion	Success, but the tester was an admin so the user could not be deleted.
Critical errors	None
Non-critical errors	None
Time On Task	14 sec

Test: 4

Role	Non-Researcher
Place (city)	Oslo
Date	18.04.2021
Age	55

Name	Register User
Successful Task Completion	Success after some failed attempts in login
Critical errors	None
Non-critical errors	Tried to register a user on the login screen. Found the “dont have a user” after a little while.
Time On Task	2 min, 37 sec

Name	Log in
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	8 sec

Name	Add location
Successful Task Completion	Success
Critical errors	None
Non-critical errors	Used some time after long press to press the location again.
Time On Task	1 min, 51 sec

Name	Edit Location

Successful Task Completion	Success, chose to delete the location.
Critical errors	None
Non-critical errors	None
Time On Task	20 sec

Name	Take Picture
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	12 sec

Name	Match Salamander
Successful Task Completion	Found camera storage and chose a salamander.
Critical errors	None
Non-critical errors	Did not select location, let the dropdown stay open. Therefore could not confirm.
Time On Task	5 min, 30 sec

Name	Edit Personal Data
Successful Task Completion	Changed name, email and password
Critical errors	None
Non-critical errors	Struggled first attempt to verify password.
Time On Task	2 min

Name	Log Out
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	4 sec

Name	Manage Users
Successful Task Completion	Success, managed to accept 2 new users.
Critical errors	None
Non-critical errors	None
Time On Task	1 min, 51 sec

Name	Manage Salamanders
Successful Task Completion	Success after some time.
Critical errors	Choose to upload an salamander image instead of managing it.
Non-critical errors	Zoomed in on the salamander image to make it bigger. Maybe thought that this was about managing it.
Time On Task	4 min, 40 sec

Name	Delete User
Successful Task Completion	Success after clicking through pending and managing.
Critical errors	None
Non-critical errors	None
Time On Task	1 min, 14 sec

Test: 5

Role	Non-Researcher
Place (city)	Oslo
Date	18.04.2021
Age	59

Name	Register User
Successful Task Completion	Found dont have a user in the beginning. Success
Critical errors	None
Non-critical errors	None
Time On Task	50 seconds

Name	Log in
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	35 seconds

Name	Add location
Successful Task Completion	Success
Critical errors	None
Non-critical errors	Started navigating the app. Struggled pressing the location after long press
Time On Task	2 min, 2 seconds

Name	Edit Location

Successful Task Completion	success
Critical errors	None
Non-critical errors	None
Time On Task	28 seconds

Name	Take Picture
Successful Task Completion	Success.
Critical errors	None
Non-critical errors	None
Time On Task	6 seconds

Name	Match Salamander
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	1 min, 10 seconds

Name	Edit Personal Data
Successful Task Completion	Success, changed name and password.
Critical errors	None
Non-critical errors	Wrong password on verify.
Time On Task	1 min, 54 seconds

Name	Log Out
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	1 seconds

Name	Manage Users

Successful Task Completion	Found pending users after 2 min. Success.
Critical errors	None
Non-critical errors	Went into managing users instead of pending users.
Time On Task	2 min, 20 seconds

Name	Manage Salamanders
Successful Task Completion	Annoying that edit fields are under the keyboard.
Critical errors	None
Non-critical errors	None
Time On Task	1 min, 33 sec

Name	Delete User
Successful Task Completion	Success after entering manages users first.
Critical errors	None
Non-critical errors	None
Time On Task	1 min, 53 seconds

Test: 6

Role	Non-Researcher
Place (city)	Gjøvik
Date	20.04.2021
Age	30

Name	Register User
Successful Task Completion	Success, used some time to match passwords
Critical errors	None
Non-critical errors	Wanted email fields to auto lower case
Time On Task	1:32 seconds

Name	Log in
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	10 seconds

Name	Add location
Successful Task Completion	Success
Critical errors	None
Non-critical errors	• Hard to add locationsLitt vanskelig å vite at man skal trykke på lokasjonen, • Æøå is missing • Wants to be able to use special characters
Time On Task	2:18 seconds

Name	Edit Location
Successful Task Completion	
Critical errors	None
Non-critical errors	None
Time On Task	45 seconds

Name	Take Picture
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	55 seconds

Name	Match Salamander
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	30 seconds

Name	Edit Personal Data
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	1:10 seconds

Name	Log Out
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	1 seconds

Name	Manage Users
Successful Task Completion	Success
Critical errors	None
Non-critical errors	• Had somewhat of a hard time to understand pending users
Time On Task	30 seconds

Name	Manage Salamanders
Successful Task Completion	Success
Critical errors	None
Non-critical errors	• It was hard to go out of the image modal, was not able to use the android back button • Wanted to og back to profile screen when using back button on android, while in manage salamanders. It went back to the password verification screen. • Comment: Use ‘choose observation’ instead of ‘choose image’.
Time On Task	5:30 seconds

Name	Delete User
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	30 seconds

Test: 7

Role	Researcher
Place (city)	Gjøvik
Date	20.04.2021
Age	62

Name	Register User
Successful Task Completion	Success after trying to create user in login.
Critical errors	None
Non-critical errors	Tries to create user in sign in screen
Time On Task	2 min, 32 seconds

Name	Log in
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	26 seconds

Name	Add location
Successful Task Completion	Found double press and pressed it again fast. Success.
Critical errors	None
Non-critical errors	None
Time On Task	1 minute

Name	Edit Location

Successful Task Completion	Success.
Critical errors	None
Non-critical errors	None
Time On Task	19 seconds

Name	Take Picture
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	22 seconds

Name	Match Salamander
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	2 min, 21 seconds

Name	Edit Personal Data
Successful Task Completion	Changes name and email.
Critical errors	None
Non-critical errors	Was unsure why he had to verify the password/confused. Understood it after 4 minutes.
Time On Task	4 min, 18 seconds

Name	Log Out
Successful Task Completion	Success.
Critical errors	None
Non-critical errors	None
Time On Task	4 seconds

Name	Manage Users
Successful Task Completion	Choose to make a user administrator and reset users password.
Critical errors	None
Non-critical errors	None
Time On Task	2 min, 38 seconds

Name	Manage Salamanders
Successful Task Completion	Success after some struggle
Critical errors	None
Non-critical errors	Clicked done after a location was chosen. Did not see that you can click on the images. Not intuitive that it was more salamanders in one id. Unsure how to navigate after the app died to start up the task again.
Time On Task	5 min, 36 seconds

Name	Delete User
Successful Task Completion	Success after struggling and navigating wrong.
Critical errors	None
Non-critical errors	Went into managing users and then pending users. Entered a lot of wrong passwords in all password verifiers. Had to get help to find the delete account button.
Time On Task	4 min, 1 seconds

Test: 8

Role	Non-Researcher
Place (city)	Oslo
Date	25.04.2021
Age	27

Name	Register User
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	50 seconds

Name	Log in
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	9 seconds

Name	Add location
Successful Task Completion	Success, after som time
Critical errors	None
Non-critical errors	Did not see the add button clearly
Time On Task	47 seconds

Name	Edit Location

Successful Task Completion	Success
Critical errors	None
Non-critical errors	Did not see the add button clearly
Time On Task	30 seconds

Name	Take Picture
Successful Task Completion	Was able to upload, but not take picture
Critical errors	Fikk no access selv om han ga access
Non-critical errors	Litt for kort knappe size
Time On Task	1, 30 seconds

Name	Match Salamander
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	20 seconds

Name	Edit Personal Data
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	50 seconds

Name	Log Out
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	1 second

Name	Manage Users

Successful Task Completion	Success
Critical errors	None
Non-critical errors	• Should have an added confirmation on deny access on pending users • Should notify that a user was moved from manage to pending
Time On Task	1,25 seconds

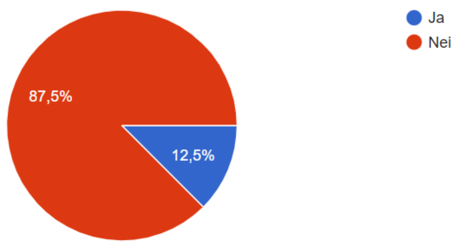
Name	Manage Salamanders
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	1,30 seconds

Name	Delete User
Successful Task Completion	Success
Critical errors	None
Non-critical errors	None
Time On Task	13 sek

I.3 Google Form Answers

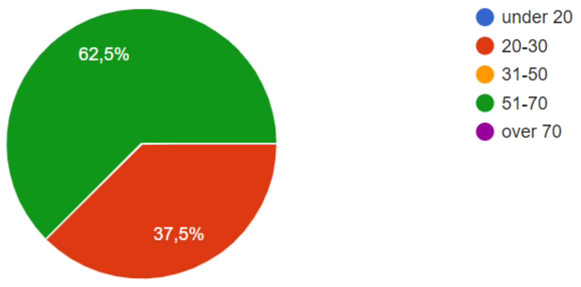
Er du forsker hos NINA?

8 svar



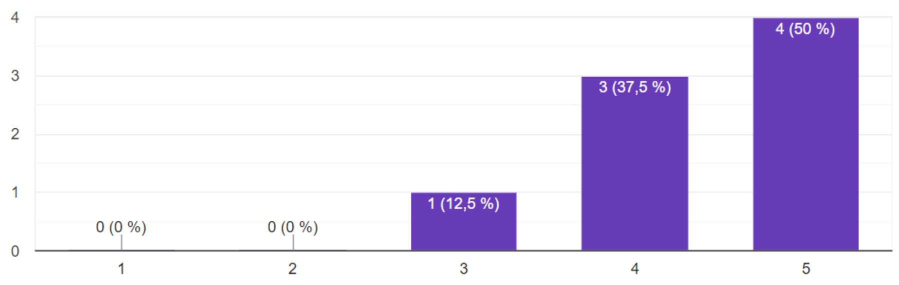
Oppgi aldersgruppe

8 svar



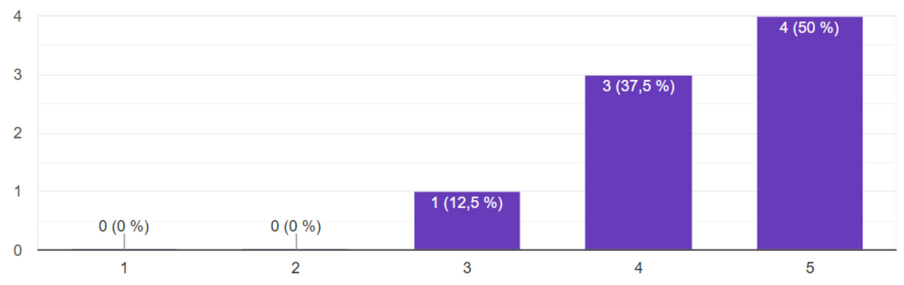
Hvordan opplevdes brukergrensesnittet i appen (knapper, menyer, farger, etc.)?

8 svar



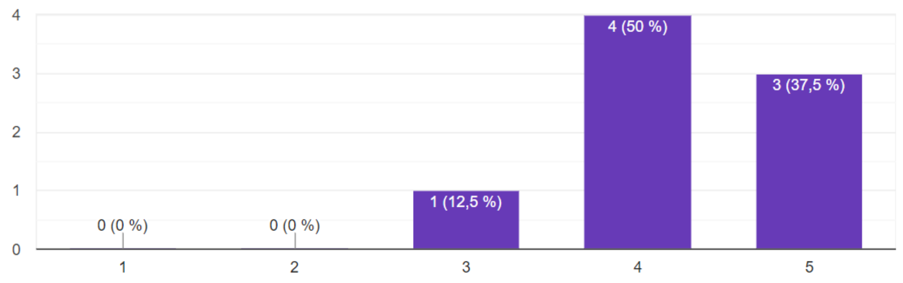
Hvordan opplevdes navigasjonen i appen?

8 svar



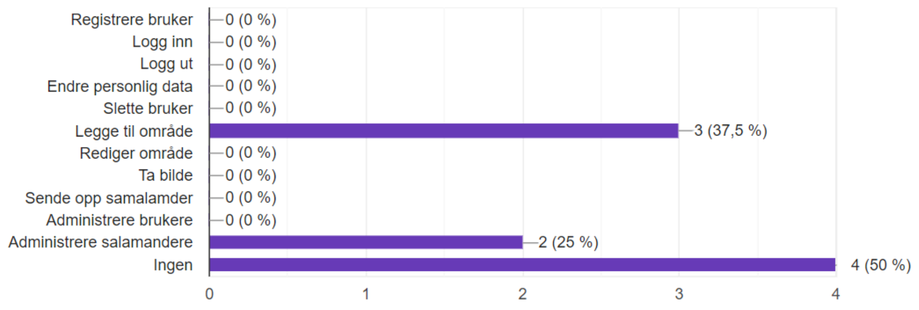
Hvordan opplevdes brukervennligheten?

8 svar



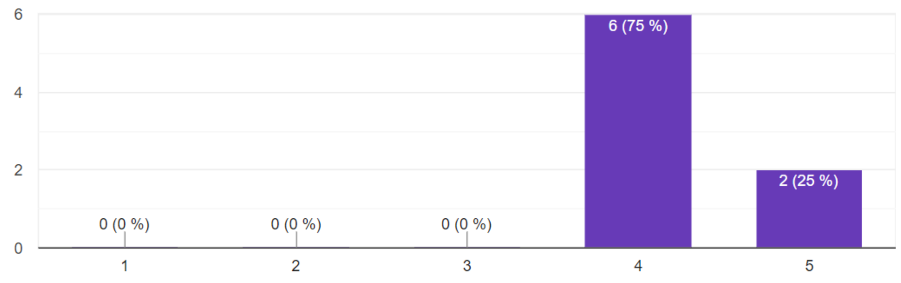
Var det noen spesielt vanskelige oppgaver?

8 svar



Hvordan var helhetsintrykket av appen?

8 svar



Hva likte du med appen? (optional)

7 svar

Simple to enter salamander information

fargevalg

lett forståelig

Effektiv og brukervennlig

Veldig brukervennlig kart. Responsiv på alle trykk.

layouten er enkel og fin. Kartet er veldig bra, spesielt at den viser hvor du er nå.

Fine farger. Enkle ledetekster.

Hva likte du ikke med appen? (optional)

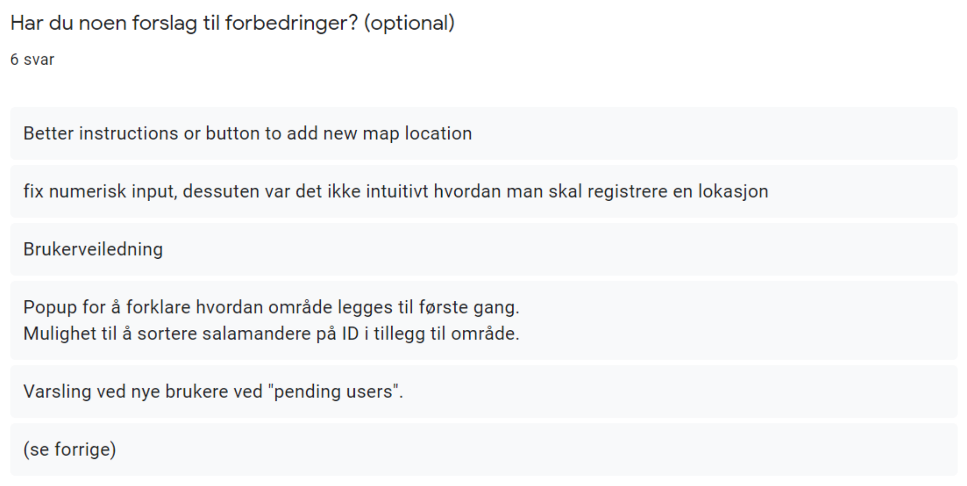
4 svar

Had to enter uid and password too often

numerisk input la seg over feltet for inntasting (redigering av salamander)

Ingen ting

Forvirrende da jeg la til salamander. Skjønte ikke at jeg også måtte vente på matching (snurrehjulet). En forklaringstekst hadde hjulpet.



Har du noen forslag til forbedringer? (optional)

6 svar

Better instructions or button to add new map location

fix numerisk input, dessuten var det ikke intuitivt hvordan man skal registrere en lokasjon

Brukerveiledning

Popup for å forklare hvordan område legges til første gang.
Mulighet til å sortere salamandere på ID i tillegg til område.

Varsling ved nye brukere ved "pending users".

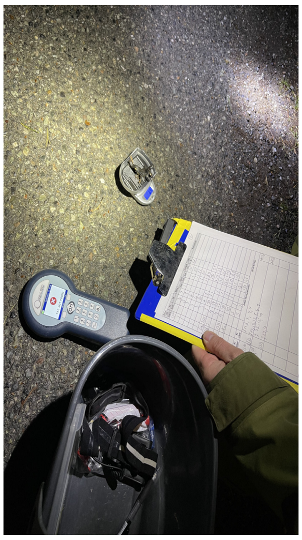
(se forrige)

I.4 Field Test Guide

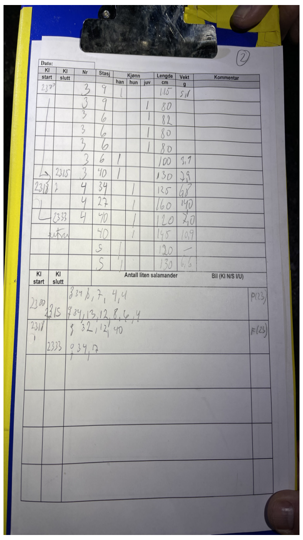
Field Test Guide

Scope	Testing the salamander application during fieldwork. The aim is to observe how the researchers use the application while working.
Purpose	Questions: • How do the researchers interact with the application during field work? • Does lighting have any impact on taking pictures/matching? • How does the outside environment affect the application? • Is the user experiencing any annoyances during periods of waiting/bottlenecks in workflow? • Does the user understand the feedback? • How is the mobile cellular coverage affecting the application? • How is the mobile's battery life affected? • How is the workflow if there are several users at the same time? Goals: • Get feedback from the user for development iteration. • Understanding how the user interacts with the app. • To find major flaws with the user interface. • Discover potential overlooked bugs or flaws of any level of concern. • Uncover security flaws of any level of concern. • Observe how the application is utilized in field work. • Discover potential bottlenecks in workflow. • Observe how climate and environment affects the application.
Schedule & Location	Lier, Norway
Sessions	No time limit
Equipment	Android phone
Participants	Researchers at NINA
Scenarios	• Log In • Add Location • Take Picture • Match Salamander • Manage Salamanders
Quantitative metrics	Number of successfully processed images vs not successful. Battery life. Number of critical mistakes.
Roles	Researcher

I.5 Pictures From the Field Test



(a) Manual registration.



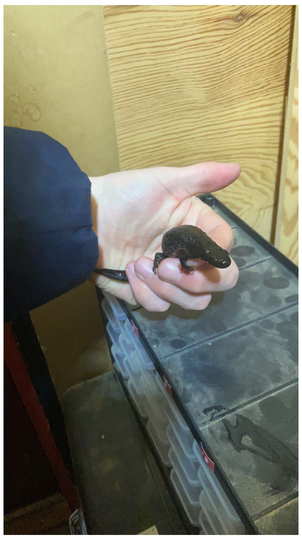
2

Dato:

Kl start	Kl slutt	Nr	Stasj	Kjønn han	Kjønn hun	Kjønn juv	Lengde cm	Vekt g	Kommentar
2300		3	9	1			115	5,1	
		3	9			1	80		
		3	6			1	82		
		3	6			1	80		
		3	6			1	80		
		3	6	1			100	3,7	
	2315	3	40	1			130	7,9	
2318	?	4	34		1		125	6,8	
		4	27		1		160	14,0	
	2333	4	40		1		120	7,0	
	ekstra		40		1		145	10,9	
			5	1			120	–	
			5	1			130	6,6	

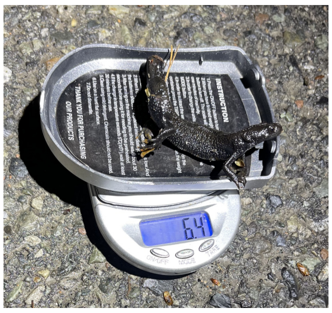
Kl start	Kl slutt	Antall liten salamander	Bil (Kl N/S I/U)
2300	2315	♂ 34, 8, 7, 4, 4 ♀ 34, 13, 12, 8, 6, 4	P(23)
2318	2333	♀ 32, 12, 40 ♀ 34, 17	F(23)

(b) Notes.



(c) Salamander.

Figure I.1: Images from the field test.



(a) Salamander being weighed.

(b) Field work.

Figure I.2: Images from the field test.

Appendix J

Burndowncharts

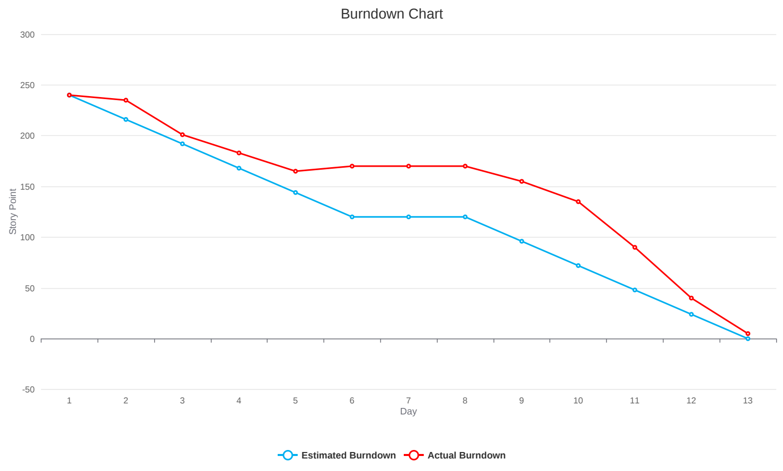


Figure J.1: Sprint burndown chart sprint 1

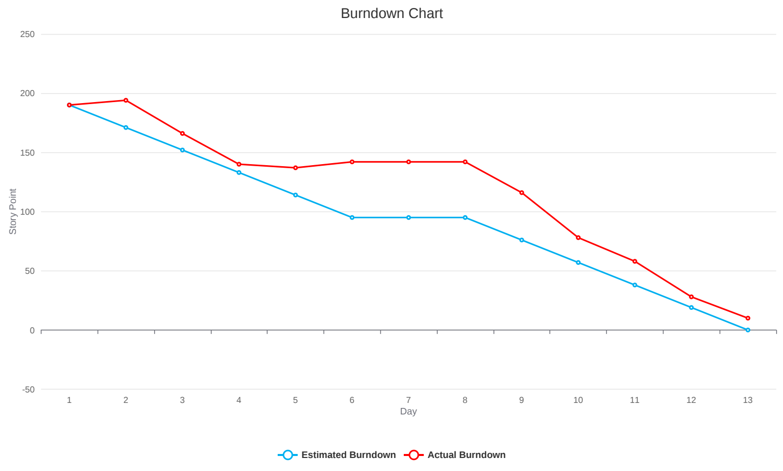


Figure J.2: Sprint burndown chart sprint 2

Appendix K

Regular Expressions

Regular expressions used in the mobile application for input validation.
Email regex:

```
^[Æ  Åæøåa-zA-Z0-9_.+-]+@[Æ  Åæøåa-zA-Z0-9-]+\.[Æ  Åæøåa-zA-Z0-9-.]+$
```

Password regex:

```
/^(?=.*\d)(?=.*[a-zA-ZÆ  Åæøå]).{9,}$
```

Measurement regex:

```
^(\d+(?:[\.\,]\d{1,2})?)$
```

Integer regex:

```
^[0-9]*[1-9][0-9]*$
```

Location name regex:

```
^[Æ  Åæøå\w\-]+$
```

Appendix L

Meeting Logs

L.1 1st meeting

Date: 12.01.2021
Participants: Bachelor group and Marius Pedersen
Agenda: First meeting with supervisor
Place: Gjøvik, Norway
Duration: 11:00 - 12:00

L.2 2nd meeting

Date: 15.01.2021
Participants: Bachelor group and Børre Dervo
Agenda: First meeting with contact person
Place: Gjøvik, Norway
Duration: 10:00 - 11:30

L.3 3rd meeting

Preparation:

- How do we write proper paragraphs? (indentation vs. empty line)
- Where do we find the proper template for the front page?
- How do we measure effect goals?

Date: 19.01.2021
Participants: Bachelor group and Marius Pedersen.
Agenda: Project plan guidance
Place: Gjøvik, Norway
Duration: 11:00 - 12:00

Meeting content:

- **Summarize:** We summarized our previous work and our meeting with Børre Dervo.
- **Raw image:** questioning if raw images are necessary. The application should not store raw images, but it should be able to receive raw images. Later in the meeting we reconsidered this topic and concluded that it might not be all that important after all, but the app might need to tell the user to retake the photo.
- **Tagging:** We discussed about the opportunity to give each individual animal metadata about it such as length, sex, weight, location.
- **Tracking:** A way to track projects? A specific project the current recording is connected to.
- **Development process:** discussing how we should hold group meetings.
- **BF matching:** should be divided into gender and location.
- **User stories:** Extract the most essential features the scientists needs in the program.
- Discussed how the process of taking these images.
- **Paragraph:** Both indentation and a clear line work but not at the same time.
- **Front page template:** Free real estate.
- **Measure effect goals:** Measure the amount of time the process it takes to do it manually and guesstimate how much faster an application with our system would make it.

L.4 4th meeting

Preparation:

- Effective goals.
- Raw images.
- How far does a salamander travel.
- How much time does it take to do the process manually.
- How should user-authentication work?
- Hardware (if they can provide a server to run at their office)
- Subject area. How they calculate population. How do they know if some salamanders are missing.
- Lincon Peterson method?
- When should we have a finished prototype.

Date: 21.01.2021
Participants: Bachelor group and Børre Dervo.
Agenda:
Place: Gjøvik, Norway
Duration: 14:00 - 15:00
Meeting content:
Effect goals

- Compared to spending approximately an hour per animal our system save a lot of time:
- Each session/pond can take around a week depending on the population in that pond.
- NINA looks at around 10-20 ponds a year.

A salamander:

- A salamander can live around 12 - 18 years.
- Most salamanders in the wild survives around 10 years.
- They are mature after three years and aren't easily individually recognizable in their youth.

Location: Børre was very positive to the idea and meant that it would be very time saving. It would also reduce the chance of mismatching. NOTE! They sometimes bring the animals to a different place to work on. They will therefore need to be able to manually input the proper salamander location. A salamander doesn't often travel further than 500 meters away from their breeding pond. **User authentication:**

- Børre wanted a user system.
- A way to set a signature on a piece of work

Hardware: They have 2 alternatives for at the time. Google or a server they just ordered/recieved.
Lincon Peterson method: Discussed the method of estimating population.
Deadline:

- Salamanders wake up around April.
- They end their migration around May the 10Th-15Th.

AI: We need pictures for training a model.

L.5 5th meeting

Preparation:

- Project plan.

Date: 26.01.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place: Gjøvik, Norway
Duration: 11:00 - 12:00
Meeting content:
General notes:

- footnote: include date. additional information which isn't a reference.
- More than one reference.
- Reference to online articles with "misc".

- Include page number when using quotes from a book.

Project plan: Chapter1:

- Result goal is what we achieve when we are done.
- Effect goal are the longlasting effects that occur from the usage of a system.
- Effect goals have to be measurable.
- Raw images shouldn't be a goal. it could be in delimitation.
- Authentication.
- Specify android and iOS versions that will be supported (probably a newer version).

Chapter2:

- Reference. Lincon Peterson method.
- GDPR as a reference.
- GDPR as an acronym.
- Always reference to pictures in text.

Chapter3: Chapter4:

- Add sources. '

Chapter5:

- Code standard.
- Discussed deployment.
- Add * to subsection.
- Typo, "meduim"
- Not being able to test the application could be a risk considering the situation.

Chapter6:

- What we think will be done in that specific sprint.
- We should specify that we will write continuously on the thesis.
- Date for specific draft.
- If people are to read a draft there needs to be time for them to read it and for us to fix potential problems
- Add presentations and lectures.

L.6 6th meeting

Preparation:

- Do they have specific ponds they track? (related to DB structure discussion). Present potential problems if we are to automatically sort salamanders by location (fylke/kommune).
- Distance between tracked ponds. Do they separate by ponds or general locations?
- Pond registration. Would Børre be able to use such a system?

Date: 12.01.2021
Participants: Bachelor group and Børre Dervo.
Agenda:
Place: Gjøvik, Norway
Duration: 10:00 - 11:00
Meeting content:
Geotagging Map system

- Very positive to the idea.
- Some ponds are very small, but it is fine as long as you know the rough location.
- Salamanders can theoretically wander between ponds.

Wishes from Børre

- To be able to set where the group of pictures are going to be taken/ have been taken.

Mark. programmed in R. For population calculations.

L.7 7th meeting

Preparation:

- Requirements, use case, misuse case, high-level + low-level?, security, risk analysis.
- Introduction.
- Subdivide our development chapter (sprinter).
- Should we have color for acronym and glossary? blue vs black.
- Acronyms, full or always acronym with link.
- Ask supervisor if he can read chapters every week going forward.
- What are necessary words to use acronyms and glossary for.

Date: 02.02.2021
Participants: Bachelor group and Marius.
Agenda:
Place: Gjøvik, Norway
Duration: 11:00 - 12:00
Meeting content:
General

- Showing our sprint plan.
- Explaining the current issue concerning missing source code.

Requirements

- (high-level + low-level) Separating use cases in high and low level is good if it helps with structure and saving time, but it is an investment. It's not necessary if you don't gain on it.

- requirements are necessary to hone the development towards the end goal. If it doesn't do then it's meaningless.

Sending small drafts Marius was very positive to the idea: "If you are able to do that I'll be very impressed". Note! this means that the report is very process based. It won't be product based. Which fits the task given from NINA.
Glossary, and acronym: Not so important. Don't use too many acronyms. link to acronym in glossary is fine. Are acronyms are always necessary by definition, but not practically (jpg). Don't use acronyms if it's not used enough.
Login discussion: Marius sad: If someone else will be using this application it should work for them as well without affecting NINAs server. Potentially future work.

L.8 8th meeting

Preparation:

- Show simple prototype to Børre for feedback.
- Get database information.

Date: 05.02.2021
Participants: Bachelor group and Børre Dervo.
Agenda:
Place: Gjøvik, Norway
Duration: 11:00-12:00
Meeting content:

Showing prototype:
Database: The server Vegard has setup might be viable. It runs most likely on linux.
new requirements? Wants to be able to tell the system that the new images aren't to be added to the system, but rather just stored?

L.9 9th meeting

Date: 09.02.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: 11:00-12:00
Meeting content:
Deeplabcut: Requirements

- A lot of storage.

Discussing the current difficulties with Deeplabcut, Tensorflow and Python. **Thesis**

- Paper or thesis.
- We need to write about individual knowledge. Not necessarily all the subjects, but everything that is deemed important. (image processing, ai, web, application development)

L.10 10th meeting

Preparation:

- How much time do they want to wait before a request is processed.
- Ask about nina and the people who will use the app for the PACT analysis.
- Show XD prototype.

Date: 12.02.2021
Participants: Bachelor group and Børre Dervo.
Agenda:
Place:
Duration: 10:00-12:00
Meeting content:
Security

- A session will last between 1-2 hours. 4-5 hours in some cases.

PACT

- They are willing to wait some (??) seconds, but not too many. They usually have 4G. They are willing to wait a while.
- English version is OK.
- NB! han nevnte 2.
- Smooth Newt etc.
- Storing weight and length for each catch (this requires date to make sense). each salamander will need an individual table in that case.

Prototype

- Use GPS to create new location.
- Add weight, length to DB (manually).
- Ruler shouldn't break the AI.

'goal

- If the app works then they might completely stop pit tagging.

L.11 11th meeting

Preparation:

- Go through Introduction Chapter and Requirements Chapter
- Show prototype
- Past, Present, Future? In regards to writing

Date: 15.02.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: 11:00 - 12:00
Meeting content:

Questions

- vector graphics is a way to draw something with math. That means that you never lose "quality" no matter how much you zoom in on the drawing. EMS, EPS
- how is grade given? They prepare some questions after reading the report. they ask the after the performance.
- Should we be consistent with time? Keep it consistent in a chapter.

Impression looks good.

- First point on result and effect goal should be swapped maybe?
- why did we pick reducing pit tagging by 50
- have images of salamander to mix it up.
- always use salamander to prevent confusion.
- you didn't HAVE to use python. It is not a constraint
- Prevent that the referance is on a different page.
- use case: log in, but not log out
- reset password should be under edit personal data
- should internet connection should be a COMMON pre condition

L.12 12th meeting

Date: 19.02.2021
Participants: Bachelor group and Børre Dervo.
Agenda:
Place:
Duration: 10:00 - 11:00
Meeting content:

Database Meeting with Vegard Bakkestuen and Børre Dervo. **Specs**

- Graphics card Nvidia Tesla T4
- Lagring = infinite

L.13 13th meeting

Preparation: UTGÅTT

- cite when there isn't one author.

Date: 26.02.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: kl, 11:00 - 12:00
Meeting content:

L.14 14th meeting

Date: 26.02.2021
Participants: Bachelor group and Børre Dervo.
Agenda:
Place:
Duration: kl, 10:00 - 11:00
Meeting content:

L.15 15th meeting

Date: 30.03.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: kl, 11:00 - 12:00
Meeting content:
Reviewing chapter 4 Discussing why a thin client is necessary.

- Are there any cases where the date isn't valid? In case they take a picture and then add them to the system at a later date. Potentially manually add dates.
- is length and weight optional? yes
- Sequential diagram may be misleading (not null)
- Having a db structure can have more advantages then just reducing the search time
- Proved to be reliable after testing ourselves. What does that mean?

L.16 16th meeting

Date: 05.03.2021
Participants: Bachelor group and Børre Dervo.
Agenda:

Place:
Duration: kl, 10:00 - 11:00
Meeting content:

- Button to save the picture is fine as long as it's not too hard to implement.
- Børre question: Possibility to use this app for other animals.

L.17 17th meeting

Date: 05.03.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: kl, 11:00 - 12:00
Meeting content:

- We discussed the possibility of extending our project. There was a though from the product owner to expand this system to other species, but that is something we will have to add to future work.

L.18 18th meeting

Date: 12.03.2021
Participants: Bachelor group and Børre Dervo.
Agenda:
Place:
Duration: kl, 10:00 - 11:00
Meeting content:

- discussing mail about server
- Børre would really like an easy way to delete a salamander.
- some date in april we will try out the system for real.

L.19 19th meeting

Date: 12.03.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: kl, 11:00 - 12:00
Meeting content:

- display date for manage salamander.
- should we display how many salamanders that are registrered to a location.
- user has to implement password a lot, but is it too much?

- for future work possibility to edit image before matching? manually set points. train a better model.
- overlapping locations.

L.20 20th meeting

Date: 23.03.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: kl, 11:00 - 12:00
Meeting content:

- demo of system.

L.21 21st meeting

Date: 26.03.2021
Participants: Bachelor group and Børre Dervo.
Agenda:
Place:
Duration: kl, 12:00 - 13:00
Meeting content:

- Demo for Børre

L.22 22nd meeting

Date: 06.04.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: kl, 09:00 - 10:00
Meeting content:

- Notified that the model had been trained for a million iterations. All the versions of the model should be mentioned in the report.

- The acronyms and bookmarks(?) are not consistently capitalized.
- 2.3 PACT analysis: explain what PACT is even in the report.
- 2.5 there are only bullet points on this page. Maybe have some text over the bullet points.
- Is there any place to get the official colors of NINAs web page or did you just inspect their web page to get the colors.
- Referring to figure 5.3.b before figure 5.3.a.

- In the section about Visual feedback with toasts use one word to describe message and response.
- For input sanitation you can mention if input is sanitized.
- Should Figure 6.10 (maybe Figure 6.9 as well) be separated into two figures?

L.23 23rd meeting

Date: 09.04.2021
Participants: Bachelor group and Børre Dervo.
Agenda:
Place:
Duration: kl, 10:00 - 11:00
Meeting content:

- Planning the testing. aiming for April the 20th.

L.24 24th meeting

Date: 13.04.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: kl, 11:00 - 12:00
Meeting content:

- User interface: In the entire chapter 5 there is a consistent lack of reference to why we do the things we do. A reference to what you should use in terms of guidelines from big people. Apples have a guideline. The drop down option for the salamander rather than a radial button. Find a reference for this.
- Don't explain something too late. Or don't introduce something that needs explaining too early.
- Chapter 6. Too vauge. page 40: specify things. The fact that this is further explained in a later chapter should be specified. Specify what resolution, specify what "The source code was missing crucial functionality, such as automatically locating"
- Move the group roles from introduction of the group members to 3.1.3 where the first sentence talks about it. PO should be changed to customer/-client.
- Group leader isn't related to scrum
- change figure 1.2
- What will you do if you are not able to test the system?
- How to verify that the system is if there is only one tester. More quality

testing.
- Quantitative testing data can be obtain by testing the system on family members. Prepare some paper that with measurable points.
- Why is user interface before development process and implementation?
- Talk about security continuously throughout the thesis and then a specific section about it in discussion.
- Should we add a screenshot of the Trello board.
- Show what we have learned in a discussion chapter. Results should be shown before discussion.

L.25 25th meeting

Date: 15.04.2021
Participants: Bachelor group and Børre Dervo.
Agenda:
Place:
Duration: kl, 10:00 - 11:00
Meeting content:

- Snow
- Pond by lahell (lahelldammen). Jan lolann.
- new box
- Børre will include some more participants for the testing.

Meeting with Robert Zahl Dybvad and Vegard Bakkestuen:

- The server is setup on the inside of NINAs servers.
- DMZ is a server for test and development. It is not internally.
- The concluded plan is to seperate the server into the API and a another server that does the actual estimation.

L.26 26th meeting

Date: 20.04.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: kl, 11:00 - 11:20

L.27 27th meeting

Date: 20.04.2021
Participants: Bachelor group and Marius Pedersen.

Agenda:
Place:
Duration: kl, 11:00 - 12:00
Meeting content:

- 1.1 and 1.2 is good
- Axios why do we use it?
- Why is redux the best option?
- 1.1 vector graphics
- Encryption of user ID (user authentication). Why did you choose to do it the way you did?
- 1.2.2 There should be an image at the point you talk about the points (even if it is displayed below).
- What does it mean when "it did not perform well on the test images" (visualize and a graph)
- Explain image augmentation.
- 1.2.3 abandoned implementation "it performed poorly on male/female implementation. What does that mean? How many images in each class?
- Tell how you did the labeling. numbers of each class. How did you ensure that you labeled on the same way? If you don't write it means that you haven't done it.
- Tell how many iterations you trained yolov3.

L.28 28th meeting

Date: 04.05.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: kl, 11:00 - 11:20
Meeting content: Deployment chapter feedback:

- Liked that Expo CLI was explained.
- Maybe there should be an explanation of what Gunicorn does.
- Add what you could have done to prevent the failed deployment in discussion.
- We should have understood that there was a higher level of security and that we should have looked further into dockers. To think that they were going to create a VM was a bit too naive and we should have done better communication with the correct people.

Testing

- How big are the images.
- Why are we testing on such a different dataset than what we trained on.
- Variation in quality. What is a high quality image in our context? Compare

it to technically good images. That should be clarified.
- Figure 2.1 make it clearer. Add the sum.

Discussion

- How much should we analyze statistical graphs? You have to understand what the graphs are telling you and explain what they mean, but you shouldn't have to analyze too much on it.
- Gantt diagram displaying writing? You should have that in the diagram. The visual is also okay, but have it displayed on the same line?
- Chapter 4: Should we talk about networking between the components (client and server)
- Chapter 4: Libraries are mentioned in implementation. Is this redundant? Move it to implementation? Marius: "this should be moved to development. These are the methods that we have and we should discuss what/why we should use them"
- Should we talk about the libraries that we will use in technical design?
- Does file structure fit in chapter 4? yes, rather refer to it in the later chapters.
- Graphical design. interaction design: should this be moved to technical design? It is a bit weird that the interaction comes at the very end of the chapter.

L.29 29th meeting

Date: 11.05.2021
Participants: Bachelor group and Marius Pedersen.
Agenda:
Place:
Duration: kl, 11:00 - 11:20
Meeting content:

- chapter 04. Talking about saving images. (what type of images)?
- visit dates on footnotes.
- Figure 6.1, but it may be too small. Add some more image description.
- Figure 6.10 is hard to read if you print the pdf. Maybe rotate it?
- chapter 7. why not display the final screens in the thesis rather than in an appendix? possibly display the main three screens and then referring to the rest.
- summary and abstract:
- Images from the testing that shows that we're out doing field tests are good!
- The image of Børre taking notes should be shown in the introduction.
- Second sentence in introduction refers to appendix C.
- acronyms should be used consistent. If you only use it once or twice don't acrshort it. Acrfull it once. use acrlong for sdm.

NTNU
Kunnskap for en bedre verden