Bachelor thesis Cryogenetics 2023

Axel Elias Wollebekk Jacobsen
Håvard Bø
Lars Lahlum Ruud
Matthias David Greeven

Summary

Title	Cryogenetics logistics solution
Project No	202
Date	22.05.2023
Authors	Axel Elias Wollebekk Jacobsen Håvard Bø Lars Lahlum Ruud Matthias David Greeven
Supervisor	Frode Haug
Client	Cryogenetics
Contact Person	Steffen Wolla
Keywords:	Cryogenetics, logistics, full-stack development
Pages	93
Attachments	13

Abstract Cryogenetics is the leading provider of technology and services to the aquaculture industry. In 2022, Cryogenetics submitted a task to Norwegian University of Science and Technology (NTNU) with a request for a new logistics solution. For years they have been logging everything from transactions to repair logs manually, using spreadsheet software. As their operation has expanded they needed a new and more efficient system. During this project, we have built a full-stack solution, which utilizes a mobile application for warehouse operators, a web application for administrators, and a backend server to store information. In this thesis we will discuss the development process from start to finish. We will also analyse our final product, to help our client in further development of the product.

Oppsummering

Tittel	Cryogenetics logistikkløsning
Prosjekt No	202
Dato	22.05.2023
Forfattere	Axel Elias Wollebekk Jacobsen Håvard Bø Lars Lahlum Ruud Matthias David Greeven
Veileder	Frode Haug
Klient	Cryogenetics
Kontakt person	Steffen Wolla
Nøkkelord:	Cryogenetics, logistikk, full-stack utvikling
Pages	93
Attachments	13

Sammendrag Cryogenetics er en ledende levrandør av teknologi og tjenester innen akvakultur bransjen. I 2022 sendte Cryogenetics en oppgave til Norges Teknisk-Naturvitenskapelige Universitet (NTNU) med forespørsel om en ny logistikkløsning. De har i flere år loggført alt fra transaksjoner til reprasjonslogger manuelt ved bruk av regnearkprogrammer. Ettersom at bedriften deres har blitt større, trengte de et mer effektivt system. Gjennom prosjektet har vi bygget en helhetlig løsning som innebærer en mobilapplikasjon for lageropratører, en nettapplikasjon for administratorer og en serverapplikasjon med database for lagring og håndtering av data. I denne bachelorteksten skal vi gjennomgå og diskutere utviklingsprosessen fra start til slutt. Avslutningsvis analyserer vi sluttproduktet for å se hva som kunne vært bedre, samt hjelpe klienten vår med videreutviklingen av produktet.

Preface

We would first like to express our gratitude to everyone who helped us complete this project. We thank our client, Cryogenetics, who provided us with an intriguing task which enabled us to further our knowledge on full stack development. Thank you Frode, for offering guidance and indispensable feedback during every step of the project. Finally, a special thanks to Steffen Wolla who provided feedback and insight during the development process.

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Project background

1.1.1 Client background

Cryogenetics AS, henceforth known as the client, is a Norwegian biotech company that provides services and products for cryopreservation of milt and the fertilization of fish. The company is based in Hamar, but maintains an international presence with labs in Chile, Canada, the United States, and Scotland[1].

Steffen Wolla, in his role as production manager and business developer for the client, has requested the development of a logistical system to register the movement of their liquid nitrogen tanks. The intention for this new system is to let employees of the company have an easier time managing the tanks, without having to manually log all transactions using spreadsheets.

1.1.2 Subject area

The main subject area for the project is logistics. Logistics deals with the movement of materials and products towards facilities, in order to sell or produce materials and services. Logistics are a part of the company's operational costs. As companies grow and expand, it gets increasingly more complex to acquire, store and transport resources. The digitalization of logistics has made many logistics processes more precise and manageable. By using specialized tools, companies are able to handle larger amounts of transactions, directly increasing profits and efficiency. Features like tracking and estimating delivery times allow companies to acquire more precise information surrounding their products, improving the experience for employees and customers.

1.1.3 Delimitation

We are going to focus on the logistics of liquid nitrogen tanks, the contents of said tanks will not be registered in our system. The client does not require the inclu-

sion of order fulfillment, labor management, or integration with other logistical systems.

1.2 Task description

The objective of the project is to create an inventory control system that can track and update the movement, status, and other details about their liquid nitrogen tanks.

Mobile Application for Operators	**Web Application for Administration**
User identification, with a numeric code of at least 3 digits.	Ability to get a report of changes in a specified format between two desired points in time.
A QR-Code scanner, to scan tank identifiers.	Authentication of administrators through email, password, and two factor authentication.
A user-centered design to limit the need for training and improve accessibility.	A page for transaction logs.
A page for transaction logs of each tank individually.	An inventory page for the tanks.
An inventory page for the tanks.	A sort, search and filter feature for transactions, locations, employees, tanks and customers.
A sort, search, and filter feature for transactions and tanks.	Provide detailed logs over the history of a specific tank, filtered by locations.
The ability to scan multiple tanks in a “filling menu”, and update the server with all newly filled tanks.	Functionality to add new customers and update existing ones.
The date of when a tank was last refilled to be visible for operators, so they are able to locate tanks that need to be refilled before the contents are spoiled.	Retrieval of a detailed change log within a time period.
	Administration of: - Laboratory locations in the database. - Clients in the database. - Operator’s information including their authentication code. - Container models in the database.

Table 1.1: Expected features for the mobile application for operators and web application for administration.

The project requires two separate User Interface (UI)s as well as a server and database. The final product has to include a mobile application that has to accommodate warehouse operators of varying technical skill levels. The application has to be able to update information about the tanks, and then update the database with this information. Furthermore, the product has to provide a web application. This will be used by administrators to print monthly logs containing information on all operations performed on locally stored tanks. The website should also provide the ability to alter tank information in case of a user error.

1.3 Target audience

Our target audience is split between the two primary products of this project, the logistics product and the thesis.

1.3.1 Logistics product

The product we developed is primarily directed towards our client. By extension, our target audience is both the administrators and operators at Cryogenetics, due to them being the primary users of the applications.

1.3.2 Project Thesis

The target audience for our thesis are all future developers looking to develop a full-stack logistics solution. Our goal is that this thesis will provide knowledge to any interested developer, such that they can use this as reference material.

1.4 Project goals

1.4.1 Result goals

Our project should produce a logistics solution which will keep track of the location of liquid nitrogen containers, as well as all internal and external transactions in which they are involved. Additionally, the solution should aid in tracking relevant cryopreservation processes. The logistics solution will be a custom designed solution for the clients needs. It will include:

- A mobile tool for operators.
- A desktop tool for administrators.
- A server and database.
- Deployment on Microsoft Azure.

1.4.2 Effect goals

By the projects end, the client will have:

- Better control and traceability over their liquid nitrogen containers.

- Digitized and simplified their workflow.
- Warnings for when containers require maintenance.
- Improved accountability.

1.4.3 Learning goals

During the development of our bachelor thesis, we will be exploring and learning about our client's subject area. The knowledge we will focus on learning include, but is not limited to:

- Learning about the practices of the logistics business.
- Designing an efficient and user-friendly logistics system.
- Creating a product with sound foundations for use in a real work environment.
- Establishing a server-based database that is nationally available and secure for company wide use.
- Using User Centered Design principles and techniques to develop a design which is intuitive to use by staff of different demographics.
- Working efficiently and professionally in a group together with a third-party client
- Improving our mobile programming skills by developing a professional application.
- Creating an efficient database using data modeling.
- Developing a web application with the use of ReactJS.
- Creating a modern and fast backend using Golang.

1.5 Project rules & requirements

1.5.1 Time Constraints

The project's final delivery deadline is 22.05.2023 12:00 PM, at this point the product, final report and documentation needs to be complete.

1.5.2 Technological requirements

The solution should be user friendly and usable by employees with different levels of technical experience. The end product must utilize a mobile device. The solution has to register and monitor the movement and maintenance of nitrogen containers. Additionally, there must be a system for QR- or bar-code to swiftly register movement and maintenance of containers. Finally, a web application for administrative tasks is also required.

1.5.3 Other requirements

The final thesis will be written in Overleaf, using NTNUs provided LaTeX format. Additionally, due to members of the group living far apart, we will conduct weekly

meetings online when it is not possible for all members to meet in person.

1.6 Group organization

1.6.1 Member experience

The group consists of four "Bachelor in programming" students, each with roughly the same academic experience. However, in the course PROG2052 - Integrasjonsprosjekt, we worked on two separate groups, gaining expertise in different fields. Håvard and Matthias developed an Android application in Kotlin, while Lars and Axel made a Golang backend.

1.6.2 Responsibilities

Each group member has an area of responsibility based on their previous experience. Table 1.6.2 provides overview of the initial role distribution. In addition, each group member has a role, as shown in figure 1.1.

Member	Role	Responsibilities
Håvard	Documentation Manager	Android development
Matthias	Communications Manager	Website development
Axel	Group Leader	Backend, Initial thesis
Lars	Maintenance Manager	Backend, Deployment

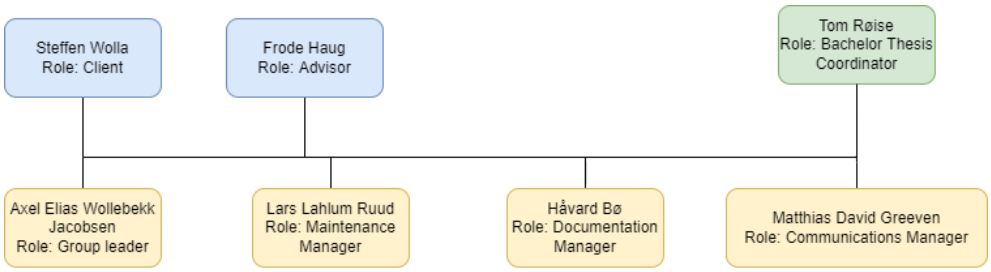


Figure 1.1: A simple overview of important people involved in the thesis.

1.6.3 Group agreement

Notification in case of absence or other incidents

Team members are expected to communicate as soon as possible if they are unable to attend a meeting with the client or are running late. If a meeting cannot be rescheduled, it will be held without the absent team member.

Expected effort

Team members are expected to spend approximately 30 hours per week on the project, but this is not strictly enforced as long as their tasks and responsibilities are completed. Team members are also expected to be available from 12:00 to 15:00 on days when a Scrumban meeting is held. If a team member cannot be available during this time, they must notify the group at least 12 hours prior.

Disagreement

If there is an academic disagreement within the group, the team should attempt to find a solution that the majority agrees on. If exactly half of the group disagrees with the other, the Group Leader will make the final decision. Once a decision has been made, all team members are expected to adhere to it.

Documents

Discord and Git are used to share files, with Google docs and Overleaf as collaborative writing tools. When writing with Overleaf we will be utilizing LaTeX to format the text. The documentation manager is responsible for meeting reports from meetings. These reports give a short description of which decisions have been made and what has been done during the meeting, the full log can be found in Appendix G.

Policy for monitoring tasks

The group leader is responsible for monitoring tasks and ensuring that everyone has something to do. If you are struggling with completing your tasks, you should notify the group leader so that your tasks can be more evenly distributed.

Submission of groupwork

The group leader is responsible for ensuring that deadlines are kept, and files are submitted.

Maintenance / Services

The Maintenance Manager will be responsible for ensuring that necessary services, such as cloud hosting services for backend, are available. This will be done in cooperation with the client, who has agreed to provide these services.

1.7 Thesis organization

Chapter 1: Introduction A short description and elaboration on the thesis as a whole.

Chapter 2: Theory Information on the subject area and client expertise.

Chapter 3: Requirement specification Descriptions on what is required of our final product.

Chapter 4: Graphical Design Descriptions of the visual aspects of our applications.

Chapter 5: System design Overview of each individual application and their component's functions.

Chapter 6: Development process A broad overview of the development process from start to finish.

Chapter 7: Implementation Thorough information and how our product is structured and why we chose to build it this way.

Chapter 8: Testing An overview of the testing performed on and in our product.

Chapter 9: Security An in-depth view on how we have implemented security into our product, as well as additional plans for more security.

Chapter 10: Reflection A reflection on the strengths and weaknesses of our final product.

Chapter 2

Theory

In this chapter, we will introduce the subject area and provide information on our client's area of expertise. We will also provide more specific requirements for each individual application as well as a list of tools we intend to use during the project.

2.1 Subject area

The primary subject area for the project is logistics. According to the New Oxford Dictionary of English[2], logistics is the coordination of complex operations involving many people, facilities, or supplies. However, the US-based Council of supply chain management professionals [3] defines logistics as "The process of planning, implementing, and controlling procedures for the efficient and effective transportation and storage of goods including services, and related information from the point of origin to the point of consumption for the purpose of conforming to customer requirements. This definition includes inbound, outbound, internal, and external movements."

Logistics can also be explained by giving an understanding of what it involves. We will use the explanation from the book "Global Logistics & supply chain management" [4]. "Logistics involves getting the right product, in the right way, in the right quantity and right quality, in the right place at the right time, for the right customer at the right cost." Getting all these points right is a challenging task, but for most businesses getting most of them right is good enough.

Our project will focus on the digitalization of warehouse management systems. As companies grow and expand, it gets increasingly more complex to acquire, store, and transport resources effectively. Warehouse management systems can be utilized to improve coordination, organization, and supervision of storage, movement, handling, and other logistics operations. The digitization of logistics has made many logistics processes more precise and manageable. By using

specialized tools, companies can handle larger amounts of transactions, directly increasing profits and efficiency.

2.2 Task Analysis

Our client has requested a Warehouse Management System (WMS) to keep better track of their liquid nitrogen tanks. Their current inventory control is a manual system, where warehouse operators enter data into a spreadsheet. Our client has chosen to not utilize an existing solution, we will identify the unique challenges which our client wants to solve. We will also discuss the scope and criteria of our task which is to produce a WMS for our client.

2.2.1 Commercially available solutions

Most modern commercially available WMS are complex and consist of tools and features which may not be useful for all clients. Some clients may also require special adaptations for it to fit their needs, while others might not find a system that is able to suit their needs at all. For example, our client needs to be able to register the refilling of liquid nitrogen to multiple tanks at the same time. This is unlikely to be a readily available feature in most WMS. Other features that may not be common in WMS are being able to maintain the stocked items, and keeping track-record of items that are sent to customers and then returned.

2.3 Definitions

Below we have provided a small list of commonly used words used in this thesis for easier reading.

- **Frontend** - The parts of the application the user directly interacts with, UI.
- **Backend** - The parts of the application the user never directly interacts with. Deals with storing and manipulating data.
- **Server / API** - A computer system that responds to other devices through a network.
- **Component** - A smaller part of a larger program, which has its own defined functionality.
- **Tank** - A thermo-isolated container capable of holding biological content at extreme freezing temperatures for extended periods of time.

2.4 Tools

Name	**Usage level**	**Reason**
GitHub	High	We are familiar with using GitHub for version control. By utilizing trunk-based development we are guaranteed code insurance.
Golang	Medium	Golang is an efficient language with great external libraries. Due to it being a young language with tremendous support, it is highly unlikely that it will become deprecated any time soon.
ReactJS	Medium	ReactJS (also known as React.js) is a free and open-source front-end JavaScript library for building user interfaces based on components.
Visual Studio code	High	Our main development IDE will be Visual Studio Code due to its simplicity and large library integration. This will be used to develop the backend and the web application.
Android studio	Medium	Android Studio will be used for developing the mobile application aspect of our project. This is mostly due to previous experience with the IDE as well as its capabilities for quickly testing code with the built-in Android emulator.
Discord	High	For day-to-day communication and meetings within the group we will be using Discord. Discord is the group's preferred tool for text and video communication as it allows us to divide text and voice communication into different channels.
Figma	Medium	Figma will be used to make low and high-fidelity prototypes of the Graphical User Interface (GUI) of our web and mobile applications. It will also be utilized for user testing and as a visual representation of the desired outcome.
Microsoft Teams	Low	Teams will be used for digital meetings with our client and counselor since it is their preferred tool for this purpose.
Microsoft Outlook	Low	Text communication by Mail will be used to send meeting notices, questions, and more to our client and advisor.
Microsoft Azure	Low	Microsoft Azure will be used to deploy our backend server, database, and web service.
Toggl	Low	To track time spent working we will be using Toggl. Toggl makes it easy to either track time spent live while working or insert worked hours afterwards.
Overleaf	Medium	Overleaf will be used to write our final thesis using LaTeX.

Table 2.1: List of tools we intend to use

2.4.1 Microsoft Azure

Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers [5]. It is trusted by many companies and governments to provide data security and provides services that tailor to the customer's needs, and some may prefer it to other competitors due to its scalability and reliability. We opted to utilize Azure since the client is already using their services for other purposes.

Azure offers an extensive array of services that cover various categories, catering to diverse business needs. Some of the key categories include databases, networking, analytics, Artificial Intelligence (AI), and Internet of Things (IoT). For our project, interest falls into the Azure SQL Database and Azure Portal services.

Azure SQL Database [6] is a platform that provides high-performance, secure and scalable storage. It serves as a highly dependable storage service, with features like automatic backups and seamless integration with other Azure services.

Azure Portal [7] is a central monitoring hub to survey the resources Microsoft Azure has to offer. This service offers extensive functionality for resource management, deployment, access control, and monitoring through the web-based interface.

2.4.2 Golang

Golang is an open-source programming language developed by Google. It was designed to be simple, efficient, and reliant, and can be used to make a wide variety of applications. It has a minimalistic syntax and a strong focus on performance. An important feature of Golang is its concurrency support which allows for "goroutines" to be created and managed quickly and easily. Golang is also statically typed, memory safe, and features automatic garbage collection. It is a compiled language rather than interpreted, which improves its performance. Some notable applications made partially with Golang include Docker, Dropbox, Twitch, and Uber [8].

2.4.3 Android studio & Kotlin

Android Studio

The official Integrated Development Environment (IDE) for Android app development is Android Studio [9]. It is based on the IntelliJ IDEA Integrated Development Environment (IDE), with specialized features for mobile app development. These are the following features that this IDE has to offer:

Version Control Integration with popular systems like Git. All changes are automatically tracked and allow for easy management of source code.

Emulator and device support are integrated with Android Studio, and allow developers to emulate different Android devices on their workstations. This allows developers to test their performance on different types of devices. The IDE also allows for physical connections between Android devices, meaning that developers can download their applications on a remote device for debugging and testing purposes.

Layout Editor is a visual design tool that simplifies and visualizes the creation of user interfaces through real-time previews and drag-and-drop functionality.

Kotlin Language Support is a crucial part of developing an Android app. Android Studio includes comprehensive support for Kotlin code, making it the preferred choice for Android developers when combined with the features described previously.

Kotlin

Kotlin is a general-purpose programming language that is designed to fully interoperate with Java code/codebases [10]. This open-source language has a growing community for Android app development and is capable of using existing Java libraries due to the seamless integration with the Java Virtual Machine. Security features like null safety reduce the risk of crashes more than Java does, while still running as efficiently.

2.4.4 REACT

ReactJS (also known as simply React) is a JavaScript library [11], and is used to create User Interfaces using components. A React component is a module that houses a specific set of functions. Utilizing components is an essential part of creating modular and cohesive software. Each component should be independent and reusable, meaning that they can quickly be replaced. The replacement component must meet the requirement of the application interface. These components are rendered to the root element in the Document Object Model (DOM), receiving any necessary values through Props.

To render and update components efficiently, ReactJS utilizes a Virtual DOM. The Virtual DOM is a lightweight representation of the actual DOM, and ReactJS compares it to make minimal updates when changes occur. This approach enhances performance when rendering and updating components.

The library is free, open source and maintained by Meta, along with its community of companies and individual developers.

What intend to achieve by using ReactJS is a fully functional and deployable Single-Page Application (SPA). The ReactJS Router framework allows us to do this. Router allows for navigation through various components by changing the URL and keeps the Graphical User Interface (GUI) in sync with the URL.

2.5 Digitalization & Digitization

The topic of digitization & digitalization stands at the core of this thesis. Our client has for a long time been using both pen and paper as well as digital spreadsheets to manually track information about their tanks. With our solution, the goal is to change that and merge all their processes into one product.

2.5.1 Digitization

Due to the confusing nature of these words, it is necessary to have a solid definition. "Digitization describes the conversion of continuous analog, noisy, and smoothly varying information into clear bits of 1s and 0s" [12]. In other words, it refers to the transition from pen and paper to electronic solutions. During our visit to the client, we found that the warehouse operators would write down information in a list when a tank is refilled with liquid nitrogen. This along with a few other operations is still done using a physical solution. Our product will take part in the digitization of the client as they no longer have to use pen and paper to track this information.

2.5.2 Digitalization

Digitalization on the other hand, "describes the social implications of increased computer-assistance, new media and communication platforms for economy, society and culture" [12]. In essence, digitalization refers to the further development of electronic services to better improve the profitability or efficiency of a business. The way we fill this role is primarily in how we improve on their previous digital solutions. They used to manually enter information about tanks and transactions into spreadsheets. Our solution intends to perform this work semi-automatically and thus, more efficiently.

2.6 Monolithic architecture

When creating any backend there is a decision to be made regarding if it should be made *monolithic* or as a set of *microservices*. A monolithic application is "a single unified software application which is self-contained and independent from other applications" [13]. This is in contrast to a microservice architecture, wherein the software is divided into fine-grained, loosely coupled, services.

Creating an application as a set of microservices has some advantages. Since each microservice can be scaled on its own, it gives more options when scaling the product. The product as a whole becomes more resilient as the failure of one microservice does not bring the whole system down. Lastly, it is easier to maintain than a monolithic application, since each component can be updated separately and faults are usually easier to isolate.

However, there are also disadvantages to using a microservice architecture. With more parts come more problems, testing, and maintenance, which may require additional effort depending on the scale of the architecture. Additionally, the need for communication between each microservice may lead to higher latency and worse overall performance.

Monolithic applications on the other hand are simpler to test and maintain and may have better performance. While monoliths have a greater risk of downtime, we believe that our utilization of Golang's exceptional error handling will be sufficient for this project. Given the scale of the project, microservice architecture is not necessary. For these reasons, we have decided to go with a monolithic structure for our backend.

Chapter 3

Requirement Specification

We will go further into detail on what is expected of our final product in this chapter. Additionally, we will provide use cases to better explain some possible operations and show how the programs will respond. We will also cover what is required of us as a group with respect to development procedures.

3.1 Functional requirements

The functional requirements entail what the project requires in order to work the way we intended it to. The following sections will go through what operations and functions the program is expected to perform.

Web application

New admins must be registered by existing admin users. The admin user has high authority when managing digital resources. They are entrusted with keeping the system running with provided tools. The admin application's features are as follows:

- Two factor authentication.
- Inventory overview.
- Transaction logs.
- Filter and search all data.
- Generate monthly reports about the registered movement of the containers.
- Generate and print Quick Response Code (QR-Code) identifier for a liquid nitrogen tank.

Mobile application

New mobile operators & devices must be registered through the admin application. Operators have access to the following features:

- Sign into the application using a personal 3-digit code.

- Receive the latest version of the transaction logs for their workplace.
- See which containers require maintenance based on data from the transaction log.
- Register when a container has been refilled.
- Filter and search all data.
- Register a container into the system when received from the client.
- When a user interacts with a container, a log must be sent to the server for record-keeping, these will appear in the transaction log.

Backend server

The server must be able to handle the following:

- Multiple requests from all over the world and respond accordingly.
- Store incoming transactions in the SQL database.
- Create / Edit / Delete elements in the SQL database.

3.1.1 Use case model

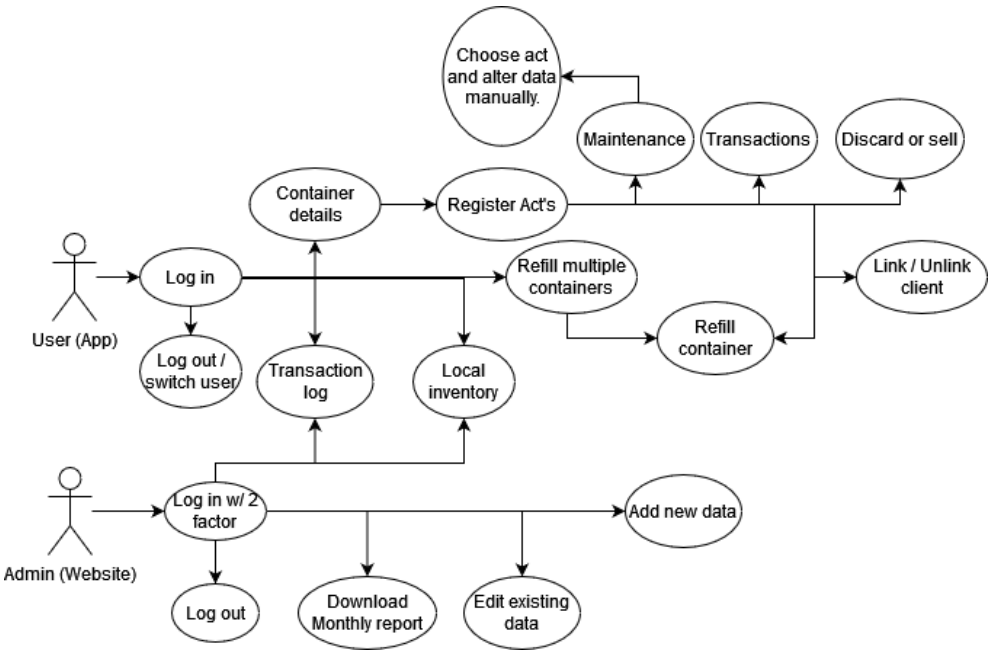


Figure 3.1: Use case model for the mobile- and web applications

3.1.2 High level Use cases

In figure 3.1, our Use case model shows what the different actors are able to do in their respective programs, and when they are allowed to do so.

The following tables show what the most important user-available features must be capable of doing. The following points on the Use case model are not included due to redundancy: Log in, log out, and alter data manually.

Use case	Transaction log
Actor	User (App) & Admin (Website)
Purpose	Retrieve an overview of the transactions regarding local labs
Description	A user should be able to view the recent transactions from / to their laboratory. The user can filter the selection of results to their specific needs.

Table 3.1: Transaction log High level Use case

Use case	Local inventory
Actor	User (App) & Admin (Website)
Purpose	Retrieve an overview of the containers currently in the local labs inventory.
Description	The user can see a list of all the containers they currently have in their storage, and an overview of that containers' current status.

Table 3.2: Local inventory High level Use case

Use case	Download Monthly report
Actor	Admin (Website)
Purpose	Produce a CSV file containing customer, start and end timestamps, location, and container for the chosen month.
Description	The report must contain the data above, categorized by customer. The purpose of the report is to track where the customers' containers have been moved within the last month.

Table 3.3: Monthly report High level Use case

Use case	Log out / switch user
Actor	User (App)
Purpose	Remove/replace what user is responsible for transaction logging
Description	Users can log out / switch users when the current logged in user no longer holds responsibility for the Acts that are being recorded. Users can quickly switch between Logged in users by typing their authentication code.

Table 3.4: Log out / Switch User High level Use case

Use case	Register Acts
Actor	User (App)
Purpose	Register a container Act to the transaction log
Description	Maintenance, Transactions, Discard or Sell, Link / Unlink client and Refill container are all marked as “Acts”. This means that this operation on the app is to record an activity that the container has gone through.

Table 3.5: Register acts High level Use case

Use case	Maintenance (Act)
Actor	Send in a maintenance update about the scanned container to the backend.
Purpose	Send in a maintenance update about the scanned container to the backend.

Description	Users have three options when assigning maintenance:
	• This container needs maintenance.
	  ◦ The user specified what is required in the comment field.
	• Assign a custom act to this container. This is necessary in order to clear up human error by letting users “overwrite” their previous transactions.
	  ◦ Assign container model, status, act, location, address, date of last refill, invoice date, and serial number.
	Assigning maintenance is tracked on the transaction log, and changes the status from “maint needed" to "maint compl”.

Table 3.6: Maintenance act High level Use case

Use case	Discard or sell (Act)
Actor	User (App)
Purpose	Remove the container from the database
Description	The user chooses a container to be sold or discarded, adding a comment with the relevant information. When a container is sold or discarded, it is no longer the property/interest of Cryogenetics, and is therefore no longer needed to track through the database. If the container is sold it no longer shows up on the laboratory's inventory.

Table 3.7: Discard of sell act High level Use case

Use case	Transactions (Act)
Actor	User (App)
Purpose	Register where the container is being sent to / when it arrives at the user's workplace.
Description	Users have three options when registering Transactions: • Container is being sent out to customer ◦ User has to input the address of the destination in order for our system to keep track. • Container has been returned to us • Container is being sent to an affiliate ◦ Users must select which affiliate the container will be sent to. This movement is tracked on the transaction log.

Table 3.8: Transactions act High level Use case

Use case	Link / Unlink client (Act)
Actor	User (App)
Purpose	Change ownership of a container.
Description	If the container does not have a client registered to it, connect an existing client with the scanned / selected container. Else if the container already has a customer, remove that customer from the container's data. Linking a container to a client results in the customer taking ownership of the container. The change in ownership is saved on the transaction log of the container

Table 3.9: Link / Unlink client act High level Use case

Use case	Refill container (Act)
Actor	User (App)
Purpose	Change the "Last_Filled" date for this container to the current date
Description	Users select which container they physically refilled with nitrogen, which will update the container "Last_Filled" date in order to keep track of which containers need to be refilled at what date.

Table 3.10: Refill container act High level Use case

Use case	Refill multiple containers
Actor	User (App)
Purpose	Scan a single / multiple container(s) to register them as refilled.
Description	Users can register that they have refilled one or multiple containers with nitrogen. This will set the last filled date to the current date. This is saved in the transaction logs of the affected containers.

Table 3.11: Refill multiple containers High level Use case

Use case	See Container details
Actor	User (App)
Purpose	Reveal the retrieved data about the scanned container.
Description	After scanning / selecting a container, the application must show the current status of the container from the server, what actions the user can register from here, and what transactions this container already has registered.

Table 3.12: See container details High level Use case

Use case	Add new data
Actor	Admin (Website)
Purpose	Input new data for the users to retrieve.

Description	Admins can add new data objects to the database by navigating to the desired data type they want to add. The following data can be added:
	• Acts
	• Affiliates
	• Customers
	• Containers
	  ◦ Container models
	  ◦ Container statuses
	• Users / employees
	Each of these data types require their specific contents filled before they are sent to the backend.

Table 3.13: Add new data High level Use case

Use case	Edit existing data
Actor	Admin (Website)
Purpose	Edit existing data in case of errors / changes.

Description	Admins can edit existing data objects by navigating to the desired data type they want to edit. The following data can be edited:
	• Acts
	• Affiliates
	• Customers
	• Containers
	  ◦ Container models
	  ◦ Container statuses
	• Users / employees
	Each of these data types require their specific contents filled before they are sent to the backend.

Table 3.14: Edit existing data High level Use case

3.1.3 Low level Use cases

Due to the quantity and size of all potential low-level use cases we decided to only add one. However, more can be found in the original draft for this chapter see appendix C

Use case	See Container details
Actor	User (App)
Purpose	Reveal the retrieved data about the scanned container.
Description	After scanning / selecting a container, the application must show the current status of the container from the server, what actions the user can register from here, and what transactions this container already has registered.
Precondition 1	User must be authenticated on the application.
Precondition 2	The container must be in a non-sold/discarded state.
Post-condition	The container data and history is shown, along with options for next acts.
Detailed course of action:	1. User locates the desired container. 2. User clicks the Camera icon to activate the QR scanning / Selects container from the inventory list. a. User scans the QR code located on the container. 3. Container data is fetched from the backend. 4. User sees the container data / history. 5. Users can Register Acts like Maintenance, Transactions, Refilling, Discard / Sell and Link / Unlink the container to customers.
Alternative scenarios	*Errors:* • The API fails to load the container data • Loss of internet connection

Table 3.15: See container details Low level Use case

3.2 Sequence diagram

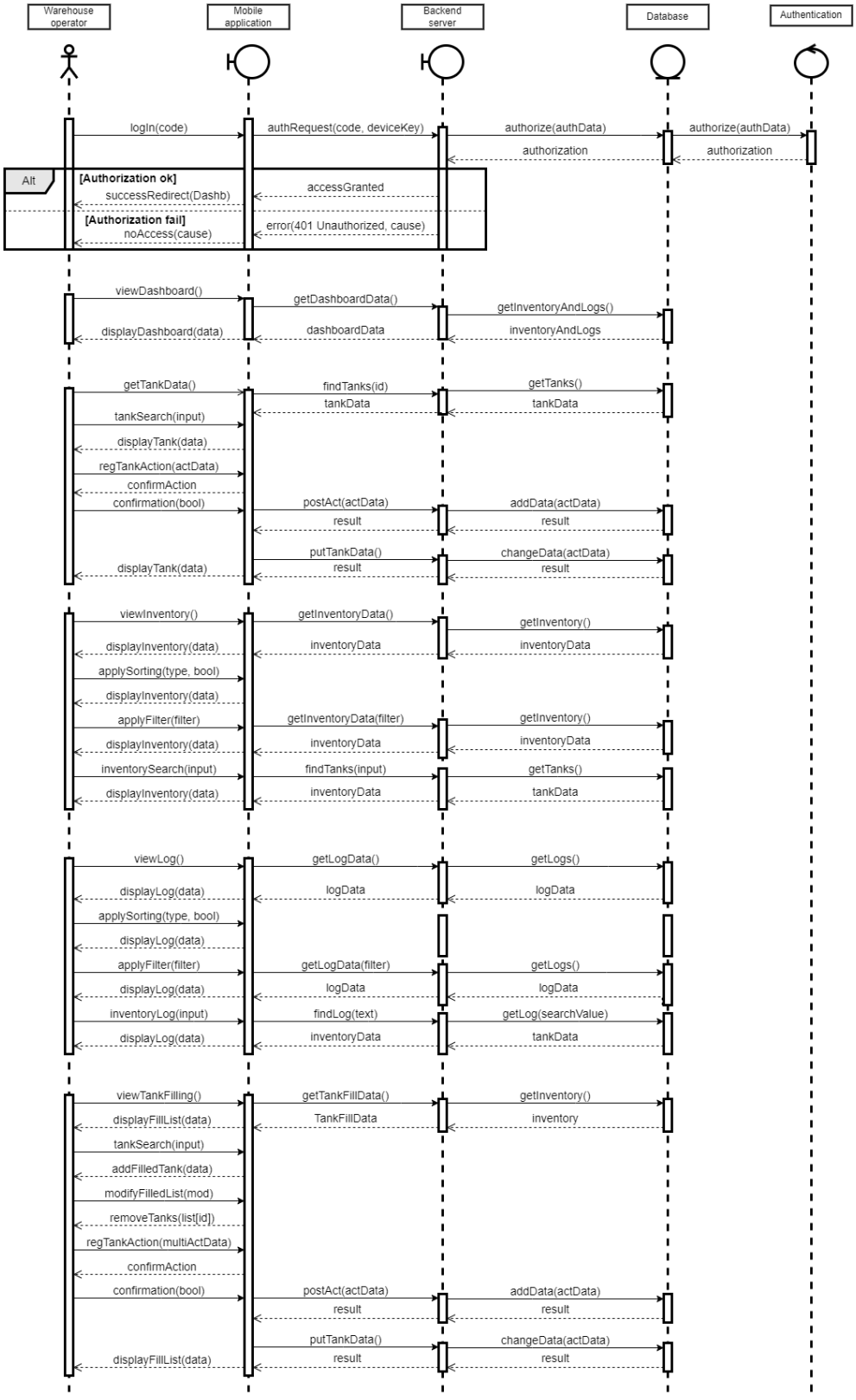


Figure 3.2: Sequence diagram showing how the different components of the mobile app cooperates

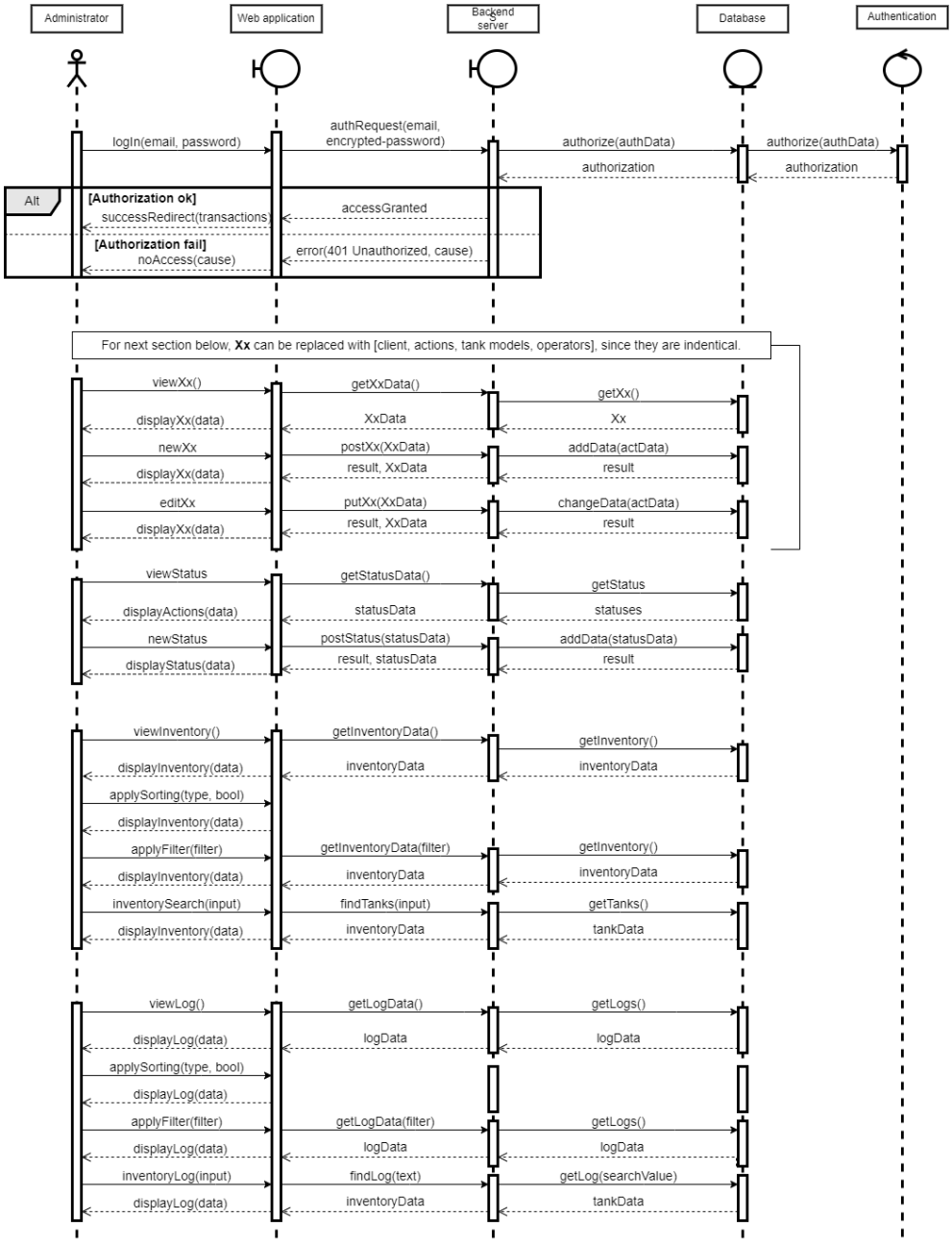


Figure 3.3: Sequence diagram showing how the different components of the web app cooperates

3.3 Product backlog

As we are using Scrumban, a Kanban board will be created to enhance workflow. This Kanban board is going to be on GitHub. To do so, a new project with a backlog will be set up and attached to the project repository. Here, issues can be categor-

ized and moved between columns, each will represent a step in the workflow. The first column will be “Backlog”. Issues here have no set start or end date and are not currently being worked on. The second column will be “In progress”, which represents issues that are ongoing but have no set end date. Then a column named “Deadline [DD/MM/YY]” will be added. We will update the column name each meeting to reflect the next deadline. Issues in this column should be time-limited, and be completed before the set date. When an issue is completed, it will be moved to the “In review” column, which we will review at the end of each meeting. Then, the reviewed issues will either be closed or put back into the backlog if they are not complete.

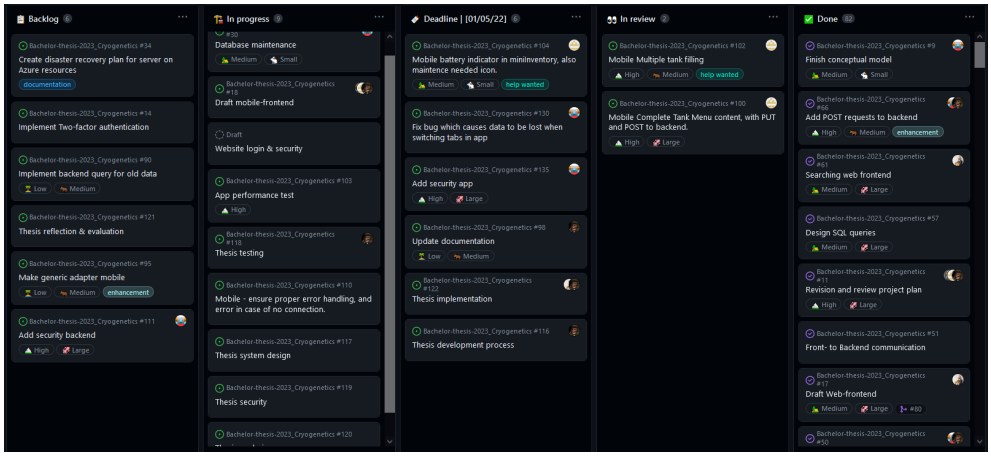


Figure 3.4: An example of how our issue board looked with deadline 1st of May

New issues will be made during meetings when there are few issues left. These new issues should typically be motivated by the GANTT 6.2 diagram, and be aligned with the current development stage. Issues are to be designated labels, size, priority, and milestone, in addition to responsible group members.

3.4 Domain model

The system mainly revolves around containers and how they are acted upon. A container has a significant number of attributes, but only the ones that are used in transactions with other tables are shown in the domain model. These are “serial_number”, “id”, and “model”, which are used when selling, moving, filling, or discarding a container. A container can be moved or filled as many times as needed, but only sold or discarded once, which is shown in the domain model as cardinality. Containers, clients, and employees are registered through the admin web page. An admin can register as many of these as needed, but only one admin is responsible for each registration. Employees may log on to either the admin web page or the app, but not both at the same time. When an employee is using the app their actions are logged. Each container sale, check-out, check-in, filling

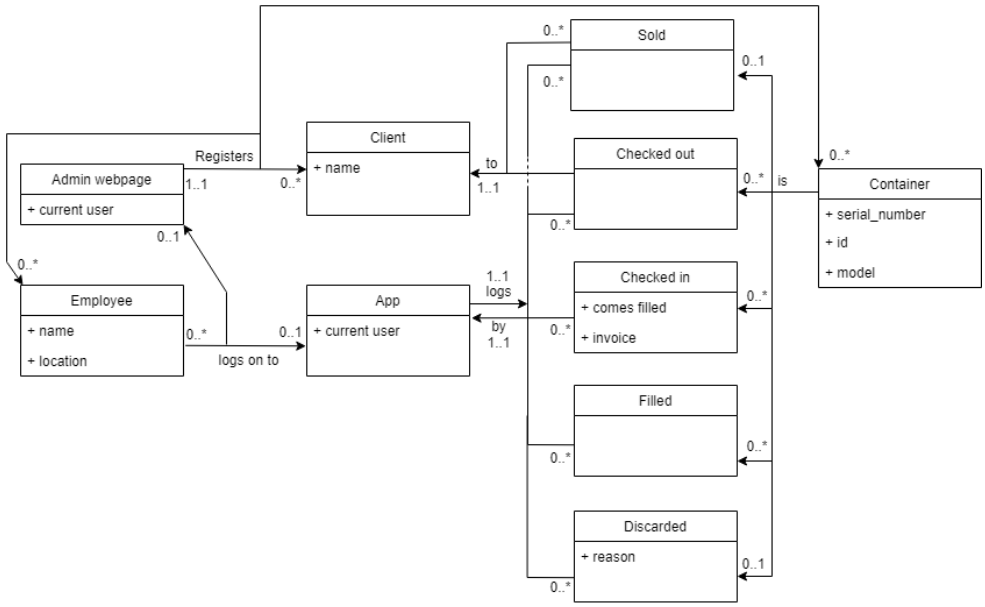


Figure 3.5: The domain model.

and discarding has a responsible app user, and containers are sold or checked out to a client who is registered in the database.

3.5 Operational requirements

3.5.1 Mobile application

The mobile application is intended to run on an Android tablet with camera access. The devices running the mobile application must:

- Have an Android operating system, version 10 - API 29 - Quince Tart or above.
- Have at least 128MB of free local storage.
- Connect to a Cryogenetics WiFi network.
- A camera capable of reading QR-codes.

3.5.2 Web application

The web application will require an internet connection and a common desktop browser. Users must login with a Microsoft account and an authentication application on a mobile device for Microsoft two-factor authentication.

3.5.3 Server

Sending and storing data should be as efficient as possible to reduce the pressure on the servers. Successful retrieval of data from the server must be in a reasonable time, which is at least within 3 seconds. The server must be deployed on Cryogenics' Microsoft Azure resources, alongside a disaster recovery plan to correct problems that may occur.

3.5.4 Technical requirements

Mobile application

The app is to be developed for Android devices using the Android Studio development environment. To utilize the latest features of Android Studio, the last stable release available to the devices should be used. The app will be developed primarily with Kotlin, the official language for Android App Development. To save development resources the application will require a common tablet used in landscape orientation for an optimized experience. The device will also need a camera capable of scanning QR codes.

Web application

To render the web application for administrators a browser will be required, as well as an internet connection. To print QR-Code for tanks, a label printer with a 4 by 6-inch output label will be required. Users will be responsible for ensuring connection to a printer, as well as installing the necessary drivers and changing printer settings for successful printing. For quick and clean web development, ReactJS will be used with the component library Material-UI. Communication between the website and the backend server will utilize the Fetch API - a native browser API that provides a low-level interface for making HyperText Transfer Protocol (HTTP) requests. Fetch API is built into modern browsers, requiring no additional libraries or dependencies, in addition to providing support for asynchronous requests and responses through the use of Promises. Promises make it possible to register callbacks that are executed once the response is received, making it easier to write cleaner and more maintainable code.

Server

The backend will be deployed on the client's Microsoft Azure instance. Git actions will be set up to automatically push updates to the Microsoft Azure cloud when changes are made to the server. The server might need to process up to 600 bytes of data per transaction, which for a typical processor (1,9GHz), takes about 40 nanoseconds, meaning processing power won't be an issue for any modern processor. Storing one million transactions in the database takes about 0,6GB of storage space, which in addition to project- and XAMPP files adds up to about 2GB of required storage space. Golang will be the primary programming language

used for the backend. It is fast and well integrated with existing technologies such as Structured Query Language (SQL).

3.5.5 Interface requirements

To ensure that our product operates nominally, we have set a few interface requirements in place. Since our product consists of two separate applications as well as a backend, the requirements have been split into three groups, mobile, web, and shared. Since we will be developing the mobile application for Android devices, the mobile device has to support at least Android version 10, Quince Tart. This is so that we can ensure the longevity of the product as an older device might cause deprecation problems. Additionally, we require the device to be a tablet, 8 inches or larger. To ensure the usability of our application we have selected colors that contrast each other, and make the app easy to use even for those with special requirements. In addition to the colors, almost all buttons have text and icons which represent them.

For the admin web application, our priority is ensuring solid contrasts. The website will have a simpler color palette. Thus, we have prioritized working with different levels of saturation to increase contrast. The website will be largely text-based. This ensures efficient communication of information. Since the application is web based there are no specific device limitations. As for shared requirements, the largest one is access to the internet. Both mobile- and web applications are reliant on communication with a backend that interacts with the database. Thus, if they lose internet access, neither application will be able to perform any operations. Additionally, both the mobile- and web applications should be able to filter and sort all available data using the different categories present, such as container size or location. Finally, the backend has to function as a server that can be run on Microsoft Azure, in accordance with the client's wishes.

3.5.6 Testing

Unit testing

From an early stage of development, we will include unit tests in our backend. By using the test-driven development style we aim to create robust and stable backend code which will act as a foundation for our client's potential continual development.

Testing during development

We will be using two main ways of testing our applications during development. Postman to test backend HTTP requests, and Android Studio's built-in emulator to test the mobile application. Postman lets us send HTTP requests to specific URLs

with payloads. Using this we can test endpoints to see if they are returning the expected data in the correct format. Android Studio's emulator lets us test how the mobile application will function on a "real" device, without having to export it as an application every time a new feature is implemented.

User testing

During the development of the whole project we will be performing user testing. Initially, we will test the low- and high-fidelity models with our client. The feedback we receive here will help us shape the application visually, in addition to adding or removing functionality from the applications. In the later stages of development, we will test the application to receive more fine-tuned feedback, while also keeping our client up-to-date on the progress of the project.

3.5.7 Security requirements and abuse handling

Our primary security concern for our program is access. We wish to restrict access so that only authorized personnel can interact with the application. For the mobile frontend this involves limiting access to the application itself. The intention behind this is to stop threat actors from exploiting potential weak points in the application through brute force. Additionally, we plan to implement device attestation to ensure no new devices can connect to the server without permission from an administrator. This implies that the web frontend will have a higher access level than the mobile application. Thus, we will implement two layers of protection, an email login and two-factor authorization to secure the website.

To tackle abuse of our system we have decided to simply utilize regular backups of the database. Since there are few things a user can do other than alter the database, it is our primary concern to ensure that all data is not lost. Thus the backups will be completely inaccessible from the program and will have to be manually accessed in Microsoft Azures cloud storage. Additionally, we will implement an automatic and comprehensive logging system. With this we can see every change made to the database and the tanks in the system. This way, if a threat actor decides to alter data to sabotage our client, we will see where changes were made, and can therefore fix them quickly.

3.5.8 Authentication

To authenticate administrators in our web application, we intend to use Microsoft's two factor authentication, which requires a Microsoft account, a mobile device and access to the client's Microsoft Azure instance.

The mobile application may only be used from a known network and device, since it uses simple 3-digit authentication for quick log in. Administrators can add networks and devices to the list of known networks and devices. When the connec-

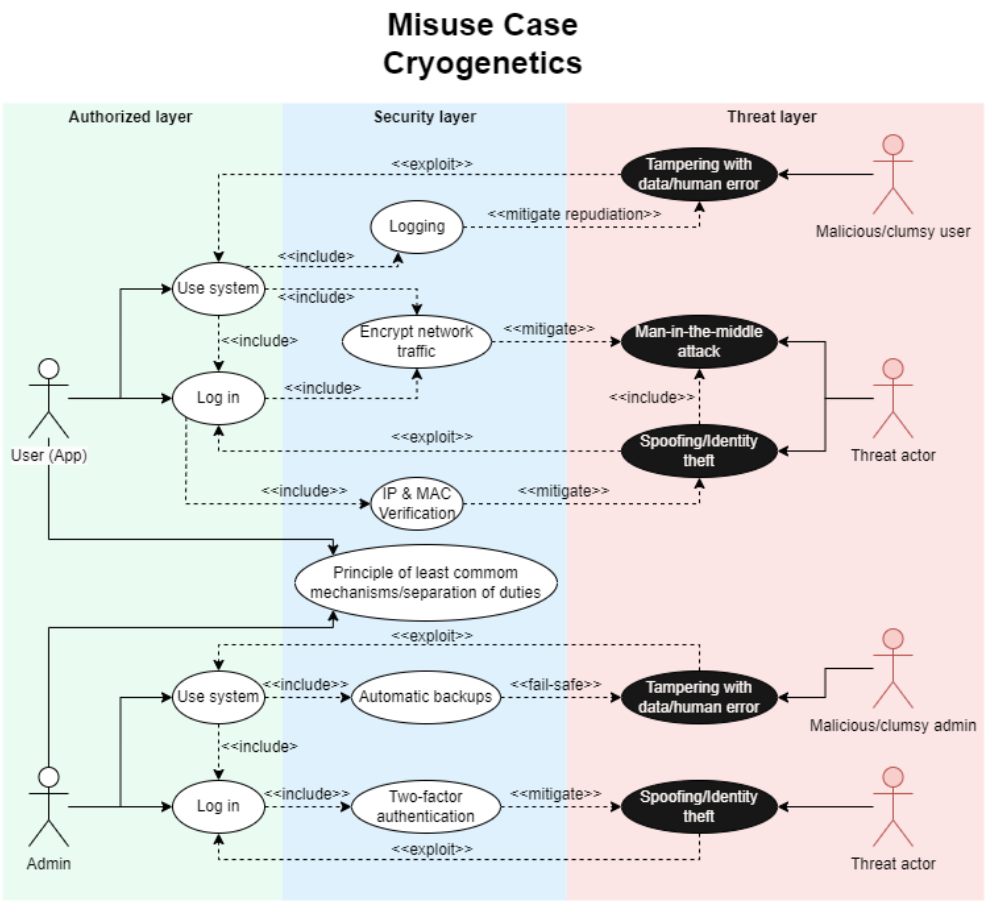


Figure 3.6: Missuse case model, displaying potential threat actors and how we deal with them.

ted network and the device is verified, the application can be used by employees associated with the location.

3.5.9 Encryption

To protect against man-in-the-middle attacks, sensitive network traffic should be encrypted. In our case we will mostly handle client- and employee information. Passwords will be stored and handled by Microsoft's two factor authentication, so salt and hashing will not be necessary. Lastly, android automatically encrypts data located in the internal storage, so the devices used by the employees will not need any special encryption.

3.6 Project requirements

This project is required to be organized and well documented, since other developers will have the responsibility for maintaining and possible further development. It is also necessary for our application to be user friendly for people of all ages, without training the users to use the applications.

For the project to be successful for our client it must:

- Offer better tracking and organization of liquid nitrogen tanks.
- Offer reports that can be used for invoicing customers.
- Offer functionality to register actions that affect the tanks.
- Be user-friendly and internationalized for use in multiple countries.

The project must be completed and delivered to both the client and NTNU before 22.05.2023 kl 12:00. The group must present the project on NTNU campus when requested, the date could be 6-8. June 2023.

3.6.1 Documentation

Documentation is an important aspect of developing software. Good planning and detailed documentation will often save time and resources in the future. Since this project will not be maintained by the same team developing it, the importance of good documentation is even greater. We will follow best practices for commenting code, which will apply to our backend API, website- and mobile frontend.

Frontend - Low and High-level design

Web- and application design will be documented with the final iteration of both our low- and high-level design. The final high-level designs will be the desired result of our development and will be used as guidance to implement the intended functionality.

Backend - Database

The database will be documented with a conceptual model and a logical model. These models will show the thoughts behind the database implementation and will help us design an optimized database with minimal oversight.

Backend - API

We will document the API by creating a plan for each endpoint, which will be helpful when developing and maintaining the API. We believe it will also increase our productivity during development since we will have an overview of the required functionality.

Backend - Deployment

We will need a disaster recovery plan before production deployment, to account for unplanned incidents which can shut down our service. One reason to plan for these incidents is to keep recovery time and data loss minimal. Another is to implement preventive measures and correctly detect issues that need to be corrected before it greatly affects the availability.

User Testing Report

User testing will be documented with a report about the suggestions, results, and conversations we found during user testing. This is to gather all the information from the user testing session in one place. This will also make it easier for us to make changes according to user feedback.

Meeting Notes

We will document all meetings by taking notes and writing a short summary for each meeting. Keeping track of all our meetings will allow us to keep track of decisions and what we have already discussed. This will allow us to have more efficient meetings and therefore get more done with fewer resources.

3.6.2 Internationalization

We have decided to solve Internationalization in two ways, by operating exclusively in English and utilizing symbols in the mobile application. We initially discussed with our client whether or not they wished for multiple languages to be available. However, they preferred to keep it simple and only use English. Since the mobile application will be used by most workers, we concluded that if Internationalization could be relevant, it would be there. This is part of the reason why we have decided to use many small icons for important and significant buttons. The web application does however, not include as many icons as the mobile app.

3.6.3 User friendliness

To create user-friendly interfaces for our applications, we will utilize User Centered Design principles and user testing to create a custom-tailored and intuitive design. Since the mobile application will be used during warehouse-related tasks, it needs to be convenient and practical to use. Since the web application is considered more of an additional tool for administrators, we will dedicate more resources to the design of the mobile application. The mobile application will feature various icons in combination with text and color, to help users navigate the user interface quickly and precisely. It will also feature common functionality which most users are already familiar with, for example changing the sorting in the log will be inspired by Microsoft's file explorer.

3.6.4 Versioning

For our project, we will be using GitHub for versioning. We will use branches to keep track of different versions of our code, and we use pull requests to merge changes from different branches. This allows us to develop more efficiently together by ensuring that everyone is working on the same version of the code. We also use GitHub's issue tracker to track bugs and feature requests. We have set up a GitHub project where we store our issues, and quickly move them between "backlog", "in progress", "next deadline", "in review" and "done". Additionally, we can assign size, priority, and assignee's as well as link the issues to specific commits or pull requests to provide context and traceability. Overall, GitHub provides us with a powerful tool for developing efficiently and transparently, in case the client wishes to further develop our product in the future.

3.6.5 Logging

During our project we will log hours using Toggl. Toggl lets us begin a timer when we start working which helps us keep track of how long we have worked. When we are done we can see how much we have worked in a certain time period, as well as what we were working on through the use of tags attached to each work session. In addition to tracking time spent, we also have a designated note-taker who writes short summaries of our group, client, and advisor meetings.

Chapter 4

Graphical Design

In this chapter we will cover the graphical design of our product's two frontends. We will cover the design process and discuss why certain decisions were made.

4.1 Graphical User Interface (GUI) development

The mobile and web application were designed using principles from the user-centered design methodology. The applications were developed together with our contact person at Cryogenetics. We also user tested and discussed the app design with the warehouse operators. Working together with the client, we have created an app for their operators and a web page for their administrators. This will help the company transition their logistics workflow from their previous solution to the new purpose-built product.

User testing the design and creating multiple iterations are essential steps to user-centered design. Therefore, we have created multiple design prototypes for both applications and conducted user testing with eight participants. Five of the participants were the client's employees, who will be using the applications if implemented. The participant's age ranged from young adults to middle-aged adults, with varying levels of digital proficiency. To obtain comprehensive data, physical user testing sessions were conducted for all participants except for one, who was tested digitally due to their geographical location.

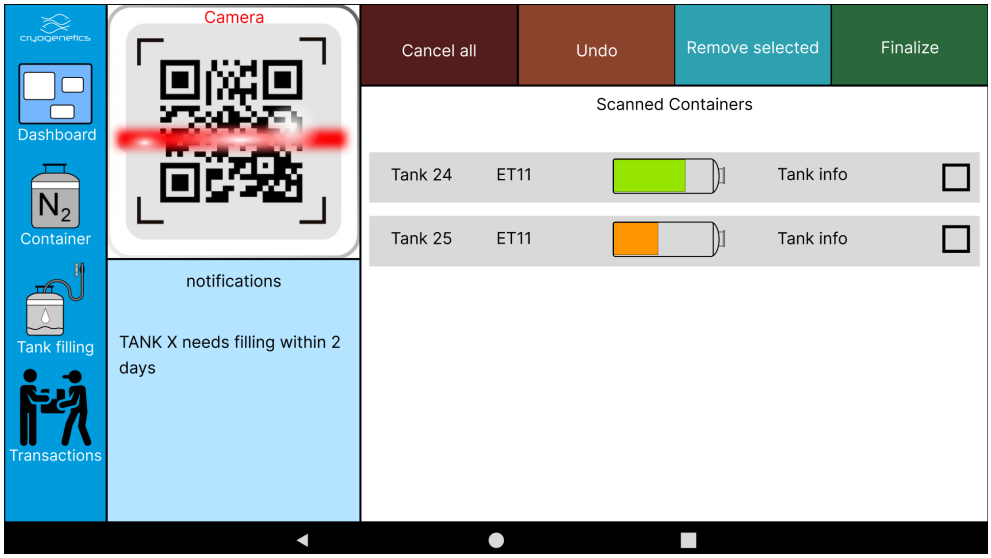


Figure 4.1: Screenshot of the fill multiple tanks page, from the first design iteration of the mobile application.

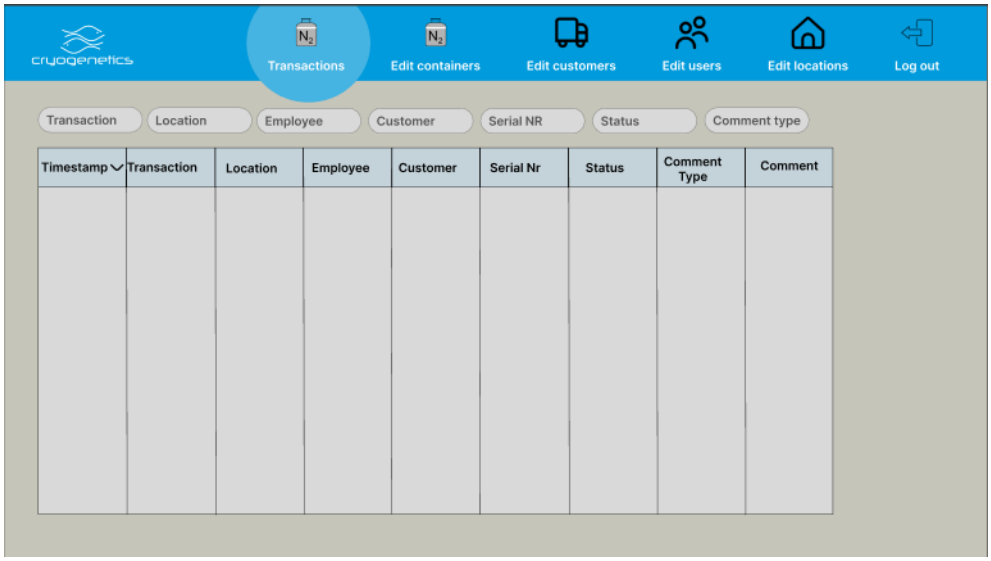


Figure 4.2: First iteration of the website transaction page and navbar.

For our initial design iterations we created two low-fidelity prototypes to save resources and keep the cost of changes low. The first drafts were created during a meeting together with the whole group. Afterwards, updates to each model were performed by the group members in charge of the respective application. The application designs were then discussed in multiple meetings with the client, which resulted in a better understanding of the operator's workflow. When the low-fidelity prototypes included all strictly necessary aspects and functionalities,

we decided it was time to move away from low-fidelity design.

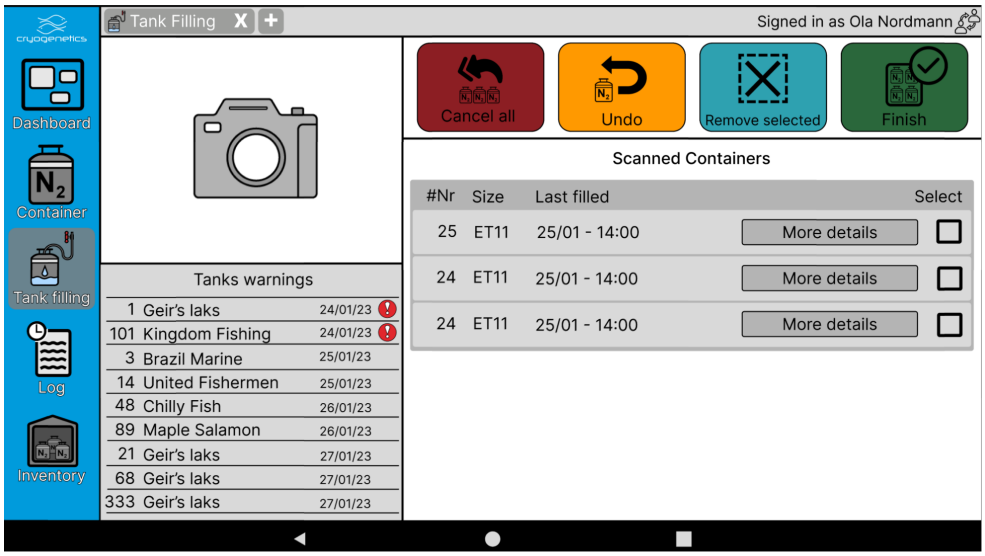


Figure 4.3: Screenshot of the fill multiple tanks page, from the second design iteration of the mobile application.

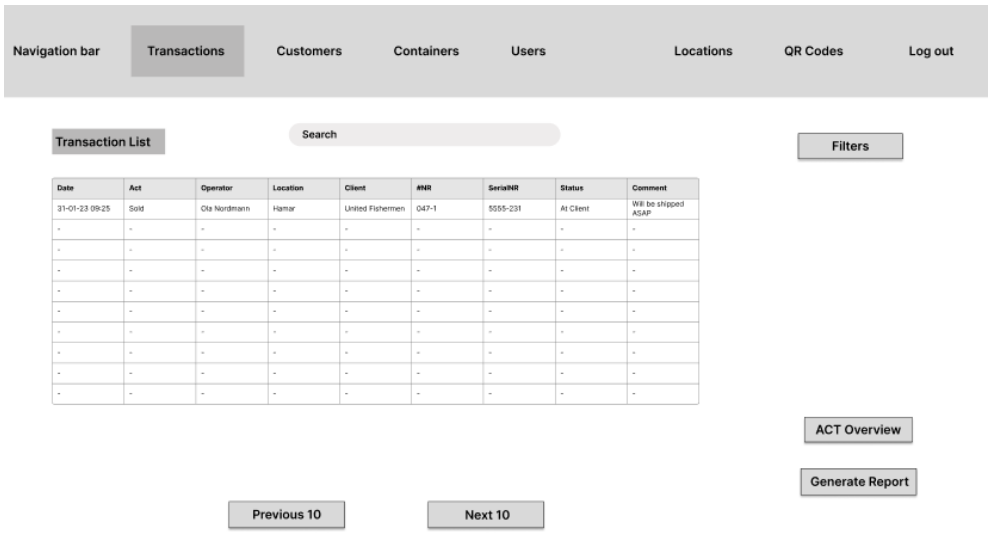


Figure 4.4: Second and final iteration of the website transaction page and navbar.

In the second iteration of the application designs, our objective was to develop high-fidelity prototypes that could be used for effective user testing. The navigation system of the mobile application was changed to suit a fast-paced logistics setting. We integrated the ability to accommodate multiple tabs and multitasking. By enabling operators to multitask, their workflow can be greatly enhanced

for certain tasks. This feature aims to provide flexibility and efficiency to the app, without making it harder to use. We also added a menu to the tank page, where operators can perform various actions on a tank. Additionally, we reworked the look and feel of the application, adopting a contemporary and minimalistic design. We decided to keep the web application strictly functional, due to the limited usage area of the application. The web application got more features, to better manage the administrator role of the mobile application. This included management of clients, tank models, lab locations, and a finalized layout of the printable container labels.

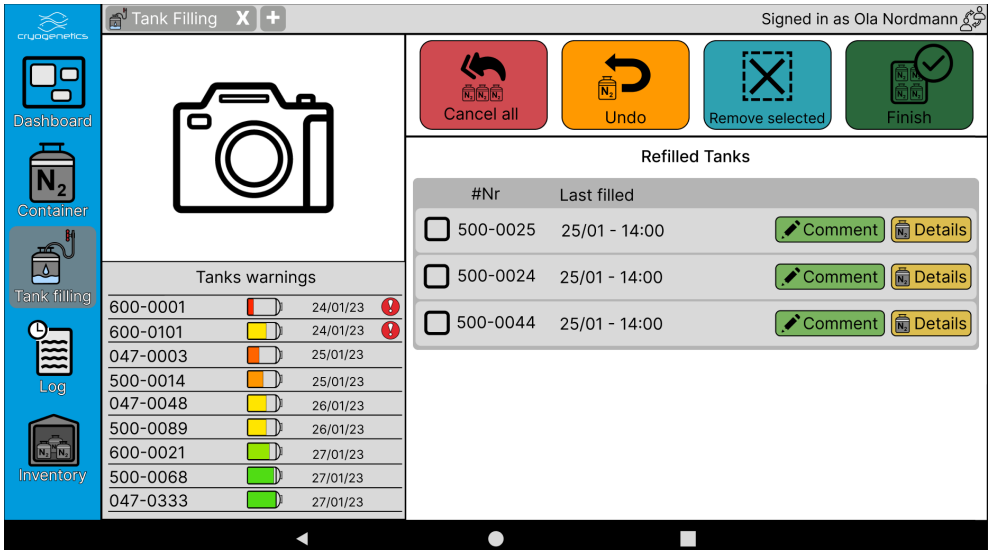


Figure 4.5: Screenshot of the fill multiple tanks page, from the third design iteration of the mobile application.

In the third and final design iteration [14], we aimed to enhance the User Interface (UI) by integrating minor additional features and altering certain elements to improve usability and ease of use. Through user testing, we observed that certain users encountered difficulties when carrying out specific tasks within the tank menu, and that the act layout appeared awkward when exceeding the menu's height. To alleviate these issues, we reconfigured the menu from a vertical to a horizontal layout, while retaining the 2 layered menu design. This was received positively by the users in the previous iteration. We also made the menu option´s text more visible, instead of only showing the icons of the menu options when a choice has been made.

4.2 Icons

An important aspect of GUI development is readability. "Information is represented by icons semasiographically. They convey semantic information in a nonverbal

manner and do not rely on a set of clear rules to convey meaning, as do written words" [15]. The intent behind adding icons to our applications was to improve the readability and accessibility of applications. Being able to see an icon and immediately know what the buttons do improves the efficiency of both the administrators' and the operators' work.

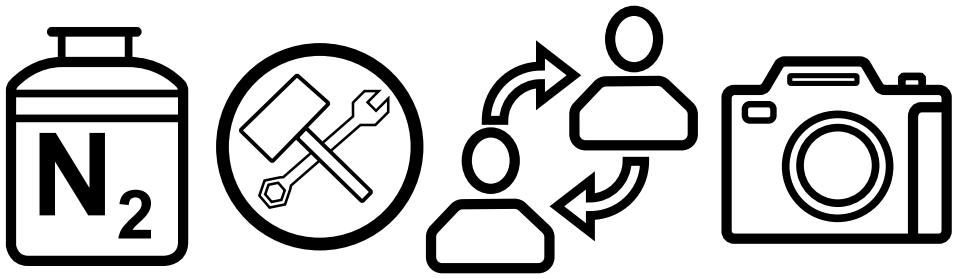


Figure 4.6: An example selection of icons from the applications. From left to right, nitrogen tank icon, maintenance icon, swap user icon and camera icon.

Figure 4.6 shows a few of the over 25 icons we made for the applications. The reason for making all of these icons instead of finding them on the internet was copyright. Although there exists free icons available for commercial use, we decided that we wanted to avoid any potential copyright problems by simply making them ourselves. Each icon has been made using vector lines such that they could be exported in any resolution without losing quality. We have made all of these icons available to our client as some of the icons we made were not used in the final product.

4.3 Color choice

When developing an application an important accessibility factor is the choice of colors[16]. Before picking colors for the program, we researched common forms of Color Vision Deficiency (CVD) and found that "Protanopia and deuteranopia, the two most common forms of inherited color blindness, are red-green color vision defects caused by the absence of red or green retinal photoreceptors"[17]. With this in mind, we ensured that any green and red color we chose had to be significantly different. To do this we performed tests using a color-blindness visualizer. The final colors of our application, tested on every kind of available CVD are visualized in Figure 4.7. Though some of the text is difficult to read if you are suffering specific kinds of CVD, we hope that context together with the icons will counteract this problem.

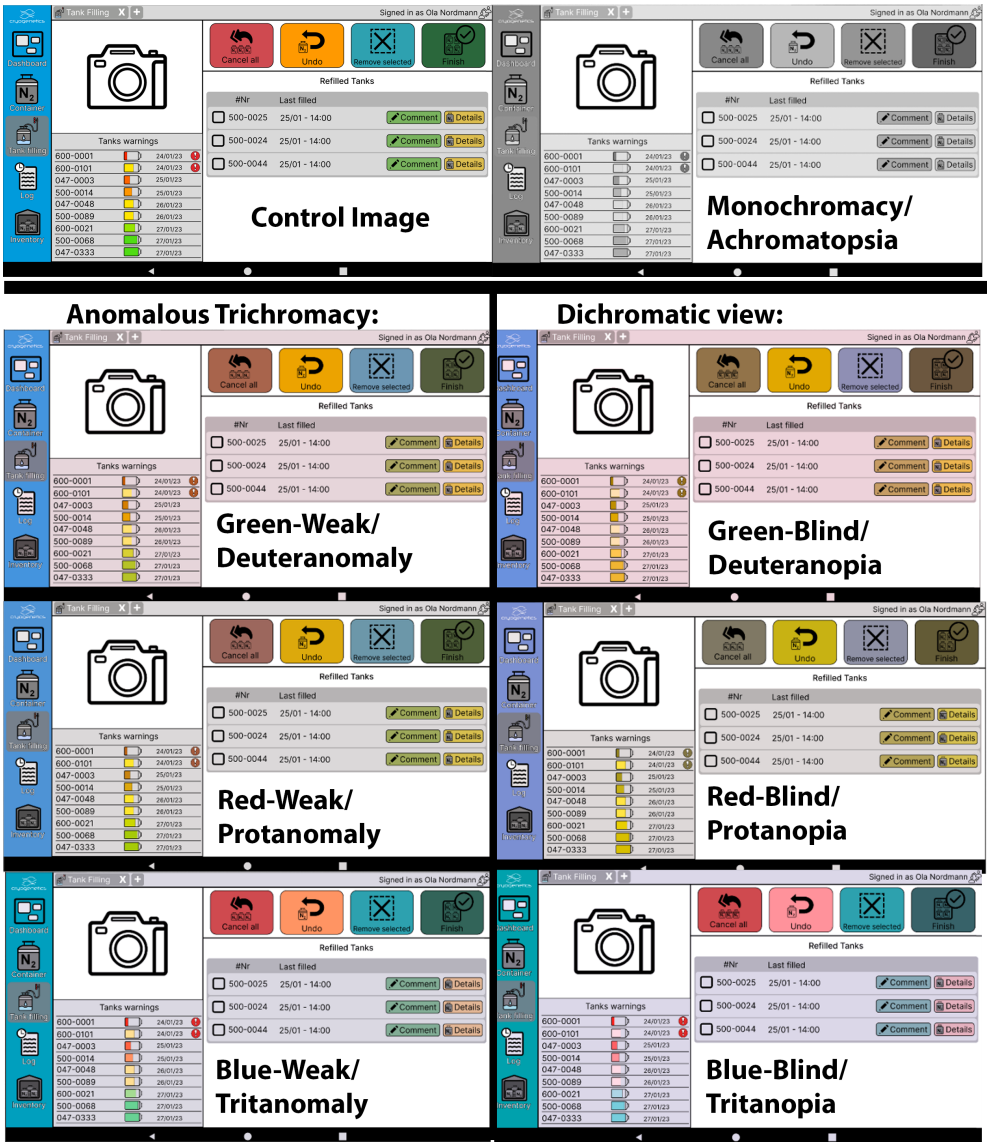


Figure 4.7: Tests performed for CVD

4.4 Admin website - UI

Since the web application is only used by administrators we decided that it should be kept clean. This was so that we could maximize the amount of information displayed without adding unnecessary noise to distract the user. Figure 4.8 shows how the main page of the web application looks. We wanted to ensure that the different function tabs on the top did not blend into the background. For this reason, we decided to add color to it. Our goal was for it to fit the theme of Cryogenetics while also being a suitable color that wouldn't demand too much attention. Luck-

ily, Cryogenetics' own blue color, which they use on their website [1], fits these criteria [18]. We also decided to utilize an F-shaped layout to ensure the readability of the website [19].

The website is divided into two major sections: The Navigation bar and the main content screen. The buttons on the navigational bar give the user access to the website, as well as indicate what page the user is on at a quick glance. The navigational bar always stays on top of the screen, except when the user is logged out. The main content screen varies based on the currently selected URL. On 4.8 we see an example of the User screen, containing a table and additional action buttons. This is standard for all the components showing data from the database.

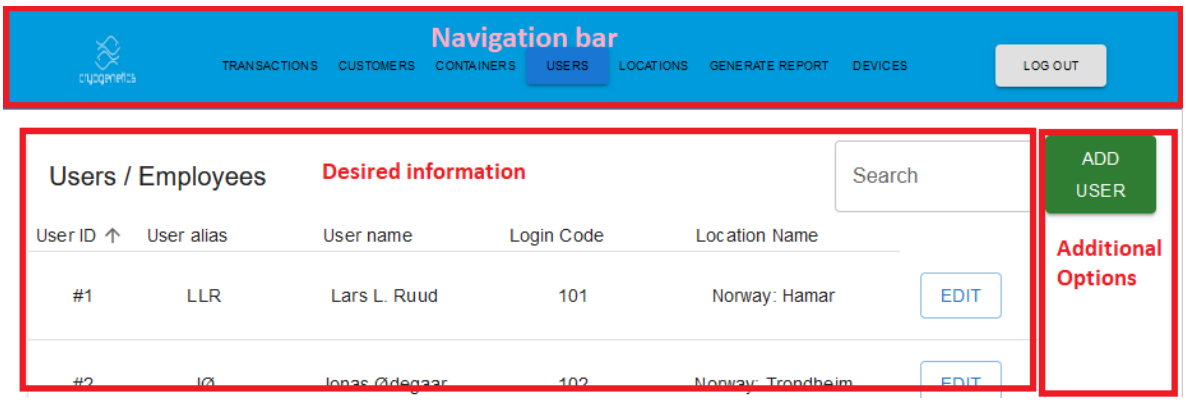


Figure 4.8: Screenshot of the layout on the website.

4.5 Mobile application - UI

Since the intent behind the mobile application is to be used in all of Cryogenetics warehouses, we wanted to ensure that all important functions were visible and easy to both access and use. Additionally, due to the small size of the tablet we wanted to avoid cluttering the screen too much. For this reason, we have allocated clear boundaries for functionalities and information to avoid blending. We also wanted to ensure that the application was accessible to all our client's operators. For this reason, we carefully picked and arranged colors to ensure even colorblind operators would be able to use it without confusion. In addition to this, we also ensured that most primary functions were labeled with both an icon and a text name.

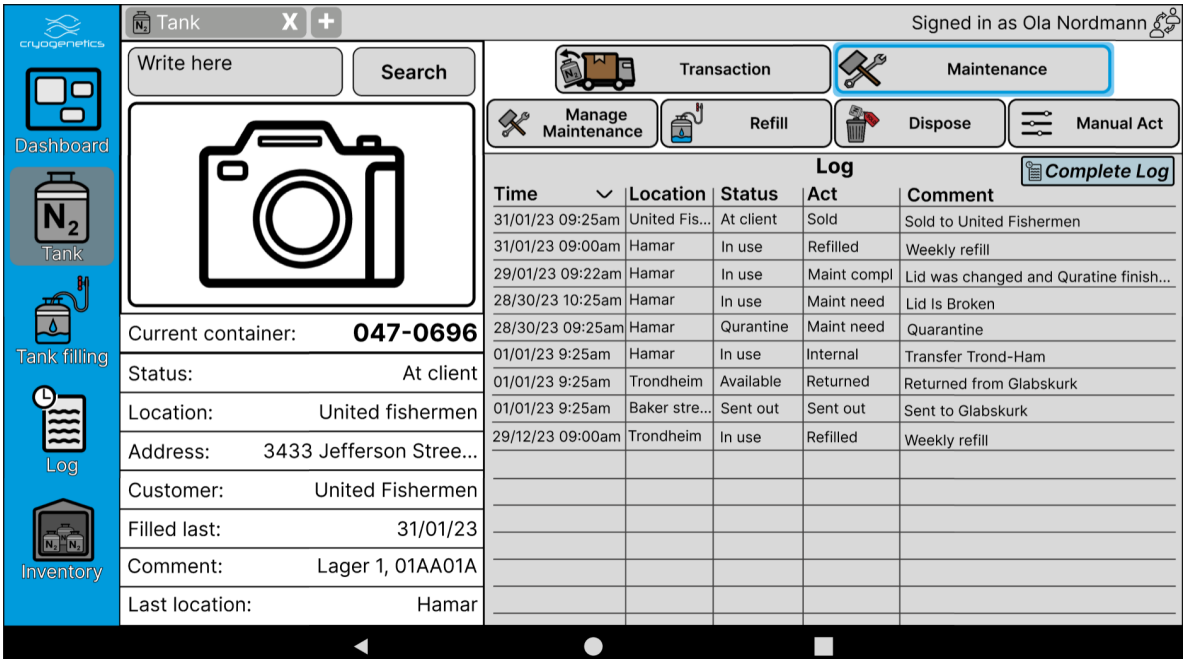


Figure 4.9: Figma design of the tank page, maintenance menu options.

The layout chosen for the mobile application is optimized for tablets in landscape orientation. By designing the user interface for this layout, we can fit more information on the screen without compromising usability. The user interface is strictly optimized for the client and their needs, to keep better track of their liquid nitrogen tanks. We decided that adding multitasking capabilities for the tablet would be beneficial for warehouse operators. The addition of multitasking allows users the ability to utilize the application in ways that are hard to consider and facilitate by introducing other features. Multitasking was introduced by creating a tabbed interface, like in a web browser. This will allow users to create different tabs of content, which they can utilize in various ways to improve productivity.

The mobile application consists of five main pages which we will refer to as dashboard, tank, multiple tank filling, act log and inventory. The menu on the left side (blue) is used to navigate to the pages, each page has different features to facilitate the client's needs. The **Dashboard** is an informative page that consists of a small version of the act log and inventory tables. The **Tank** page displays the information of the tank chosen by scanning QR-Code or searching (green), the values of the tank are below (yellow). The menu (brown) is a layered menu with a total of 8 menu options divided into two categories, "Transaction" and "Maintenance". The sub-options of the "Transaction" category can be found in figure 4.10, sub-options of "Maintenance" can be found in figure 4.9. The sub-options reveals a layout of user input fields (pink) in between the menu and the small version of the act log below (teal).

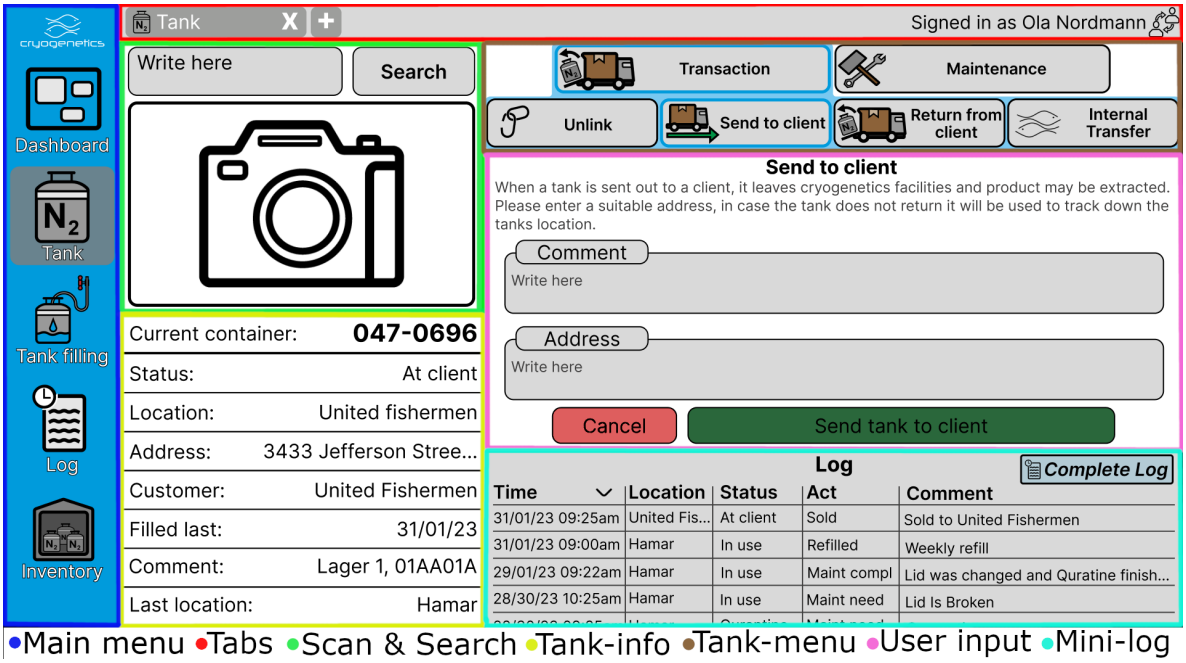


Figure 4.10: Figma design of the tank page when performing send to client action.

A central part of our logistics solution is the act log and inventory table, both tables also have a smaller version which is used on the dashboard, tank, and tank filling page. We tried to make the tables more modern by removing the border surrounding the table, making it blend more into the background of the page. The "Filter" button on the top left will open a pop-up window to filter the table displayed, the data can also be searched in the top right corner. To sort the table by different values the box containing the name of the column can be pressed, if it is pressed twice the order will be reversed. The **Tank filling** (figure 4.5) page can perform multiple refill actions simultaneously and will be used for weekly refilling of the liquid nitrogen tanks. The act **Log** (figure 4.11) page displays the values we will store when the different actions are performed. The **Inventory** (figure 4.12) page displays the values of the tanks currently in use.

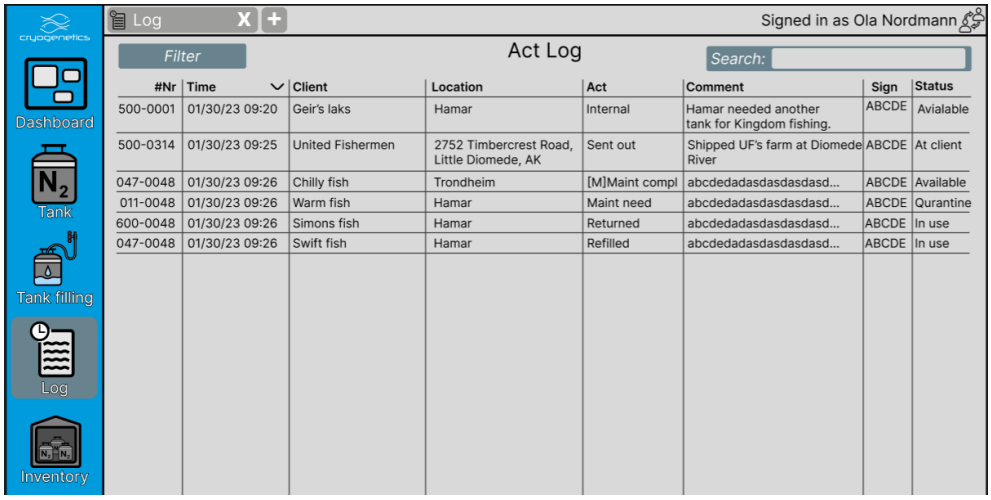


Figure 4.11: Figma design of the act log page.

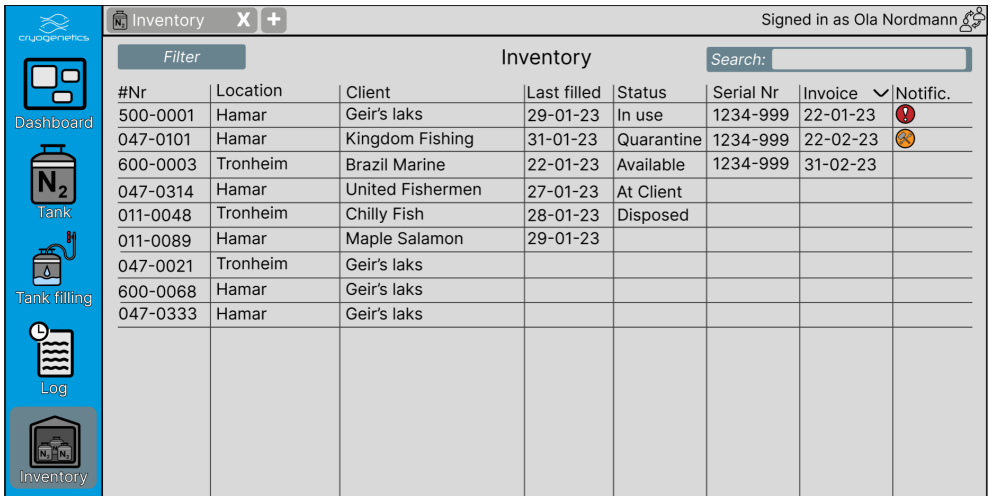


Figure 4.12: Figma design of the inventory page.

Chapter 5

System Design

In this chapter we will cover the structural design of our product as a whole. As mentioned in the earlier chapters our product exists of four parts, two frontend applications, mobile and web, as well as two backend components, the server application and the database. We will take a look at each primary component and introduce its functions.

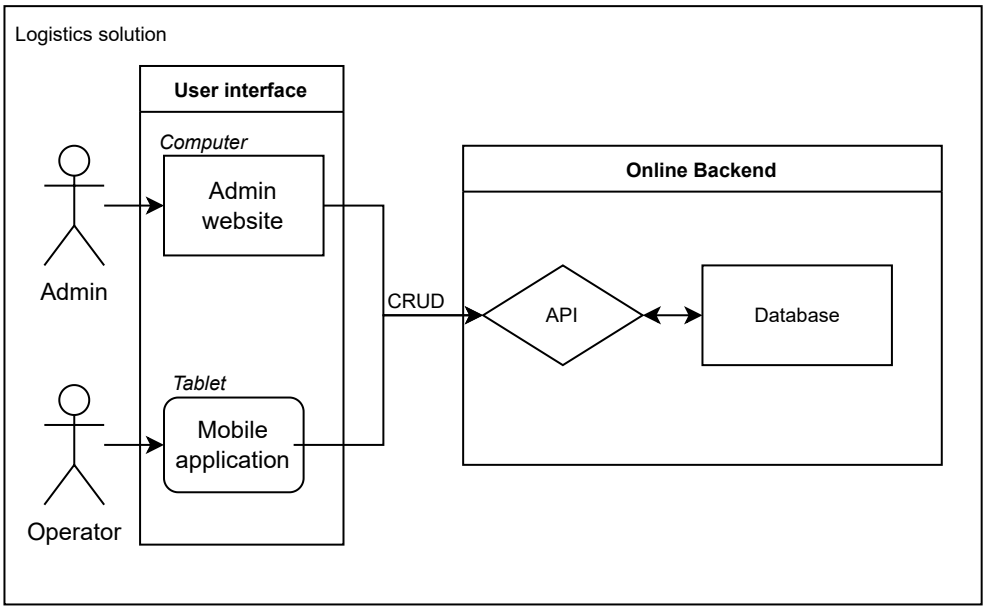


Figure 5.1: Simplified solution architecture

5.1 Frontend

Figure 5.1 provides a rough overview of how the individual components are connected. The product provides two separate ways for a "user" to interact with the

product. We have separated users into two categories, where "admins" use the web frontend, and "operators" use the mobile frontend.

5.1.1 Web application architecture

The web application is a simple ReactJS website meant for admins to have full control over the database without having to manually edit it. As the website will have a high level of access, we decided that it needed to be secure enough to deter most simple assaults. We will cover how we did this in chapter 9.

By using React and most importantly, React Router, we have been able to make the website into a Single-Page Application (SPA). This means that the main page (`App.js`) always stays loaded on our screen, importing components into itself to the "change page". In the `App.js` file, we define all the components that are available to the user, as long as they are logged in. This ensures that the application is not accessible without the proper credentials.
The structure we define in this main file is reflected in every "page" we set up in the future, so we have designed it as follows: First is the Navigation bar (NavBar), a panel of buttons leading to different pages in the program and the logout button. Second, is the page the user has selected, with all its content and buttons delivered to the main page through the returns of those page functions. Last is the copyright notice, which serves as a demonstration to the client on where they could include it in the future.

5.1.2 Mobile application architecture

The mobile application has a relatively simple structure. To accommodate for the multitasking feature, all "pages" had to be made as fragments instead of activities due to how Android handles memory. This means that the only activity we have is the initially loaded "MainActivity", which immediately loads a fragment presenting the login screen. Most other functionalities are contained within other fragments. The fragments present data dynamically using "recyclerviews". Some significant components are not bound by this structure. This includes features such as the API calling functions and all code related to QR scanning. These are generic and accessible to all fragments.

5.2 Backend

5.2.1 Server API

Due to all database interaction being performed by the server API we decided that it should be built as robustly and dynamic as possible. A central component to this is the SQL statement constructors. We built four separate constructors, one for each of the "GET", "PUT", "DELETE" and "POST" request types. We will go more

in-depth on how these were made in chapter 7. To simplify further development we built the API such that all tables can have their data altered using endpoints that are dynamically created based on a list of table names on the server. This means that if a developer wants to add another table of data, they only need to add it to the database and then add it to the list of table names. After this, an endpoint will be accessible for interaction by the frontend.

Endpoints

At time of delivery our API provides standard interaction for all tables from our database. List 5.2.1 provides an overview of all standardized endpoints, all are prefixed by the base URL which Cryogenetics have yet to decide.

- /api/transaction/
- /api/client/
- /api/container/
- /api/handler/
- /api/act/
- /api/container_model/
- /api/container_status/
- /api/employee/
- /api/location/

However, there are some endpoints that don't utilize the standardized functions. Each of these endpoints have unique functions tied to them that are not used by the standardized endpoints. These are displayed below in list 5.2.1, with a short description of their function. Additional information on how each endpoint works is covered in chapter 7.

- /api/user/login/
    - Handles the identification of operators using their login-code.
- /api/user/admin/login/
    - Handles authentication encryption and decryption of an admin's login information.
- /api/create/
    - container/
    - transactions/
    - employee/
    - Endpoints built to fetch data from each table, and data from each of their respective Foreign Key-tables.

5.2.2 Database

The database is where the whole logistics system stores its information. Due to the importance of this component, we decided to spend significant time ensuring

that it was robust and well-structured.

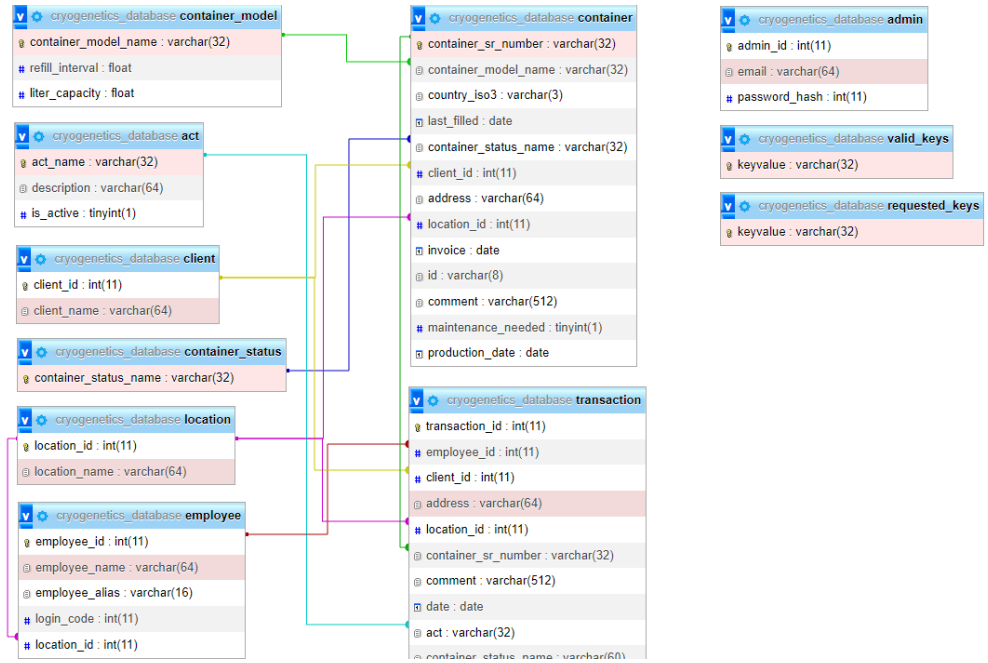


Figure 5.2: A visual representation of the database structure.

Figure 5.2 shows how the different tables of the database are connected through the use of Foreign Keys. By using Foreign Keys and tables with static information such as status or client name, we can construct new tables that hold data that gets updated dynamically when the Foreign Keys data are updated. Two significant tables are container and transaction. In the container table, we hold all data related to a single container using its serial number as the primary key. Since we want to save every alteration of a container done by the mobile frontend, we have the "transaction" table to handle this. The transaction table holds information on which container is altered, who did it, and when it was done. Additionally, it stores what was done and potential comments added by the operator, as the container is updated. This is the information displayed in the transaction report printable from the admin application.

Chapter 6

Development process

In this chapter we will cover our development process, including the choice of development method and meeting schedules. We will also elaborate further on our use of GitHub as a version control tool.

6.1 Development Model

Due to previous experience with different development frameworks, the process of deciding on one was not long. Due to the rigidity of standard frameworks like Rapid Application Development (RAD) or waterfall, we decided that an Agile framework would better suit our needs [20]. For our bachelor project, we decided to work using the Scrumban agile framework. Scrumban, as the name implies, takes aspects of Scrum and Kanban to create a highly adaptive agile framework[21] benefiting from the advantages of both, with less of the negatives. Additionally, since we are familiar with Scrumban it is easier for us to effectively utilize it instead of spending valuable time learning a new framework.

To make our Scrumban board more effective we decided to use GitHubs own solution called "Github project" [22]. By using GitHubs own issue board instead of an external solution, we can update the board using Git while referencing specific issues in our commit messages. We have also added a "workflow" which automatically moves issues to the "done" column when they are closed.

6.1.1 Meetings

Weekly meetings

As mentioned, weekly meetings are held according to the Scrumban framework. Each Monday and Thursday at 12:00pm a Scrumban meeting is held physically or over Discord. The Scrumban leader is responsible for incorporating proper Scrumban technique, such as the use of the Github project issue board. Meetings with

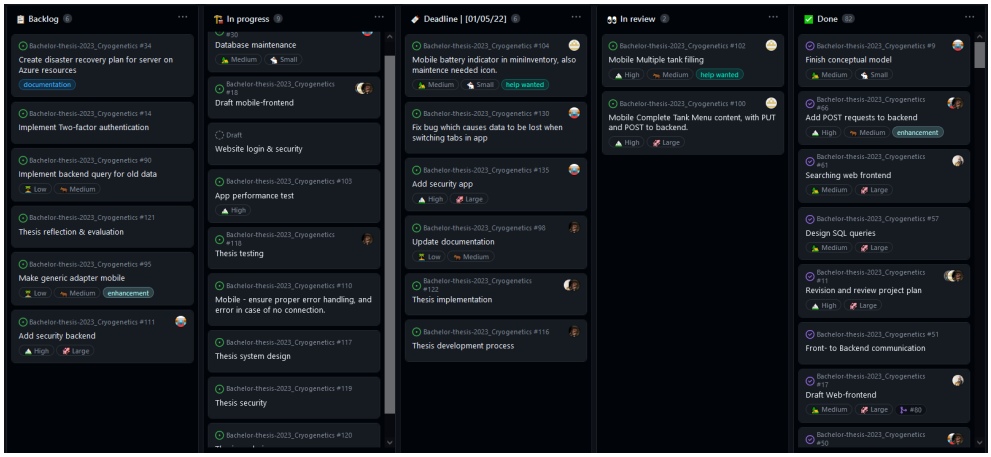


Figure 6.1: An example of how our Github project looked during development with a deadline of the 1st of May

the client are held on Mondays at 15:00 over Teams, if necessary. These are arranged at least 96 hours prior by the Communication manager and may be moved to a more suitable time if needed.

Client meetings

To ensure close cooperation with our client we decided to commit to weekly meetings in the startup period of the project. During these meetings, we asked questions regarding the task to our client's spokesperson, Steffen Wolla. He would provide feedback on the planned features to ensure that we had not misinterpreted the task. When we later moved to bi-weekly meetings, we would perform small-scale user tests. Here we would receive feedback on whether or not he found them user-friendly and in line with his idea of the product.

Advisor meetings

We decided early that the most efficient way to utilize our advisor was for him to assist us with the writing aspect of the thesis. This meant that instead of having weekly scheduled meetings, we rather contacted our advisor as we approached significant development milestones. Our advisor requested that if he was to review anything, it had to be submitted 24 hours ahead of the meeting. This proved essential, as his feedback was indispensable.

6.2 GANTT chart

As part of our initial project planning, we decided to set up a GANTT chart. Having a GANTT chart assists us in maintaining steady work progression while not

being fully binding. We benefited greatly from this when we decided to alter our priorities a few weeks in. We will cover this more in-depth in Chapter 10.

ACTIVITY	START	DURATION	END
Planning	2	4	5
Project plan draft	2	2	3
Project plan review	4	1	4
Project plan deadline	5	1	5
Development start	5	6	10
Physical database	5	2	6
Backend draft	6	3	8
Website draft	7	2	8
First status report	9	1	9
App draft	5	5	9
User testing	8	1	8
User test revision	10	3	12
Website revision	10	2	11
App revision	10	2	11
Backend complete	10	3	12
Thesis start	12	6	17
Thesis outline	12	2	13
Second status report	14	1	14
Thesis draft	14	4	17
Development end	15	2	16
User test final round	14	1	14
Website complete	15	2	16
App complete	15	2	16
Thesis end	18	3	20
Third status report	18	1	18
Thesis review	18	2	19
Thesis complete	20	1	20

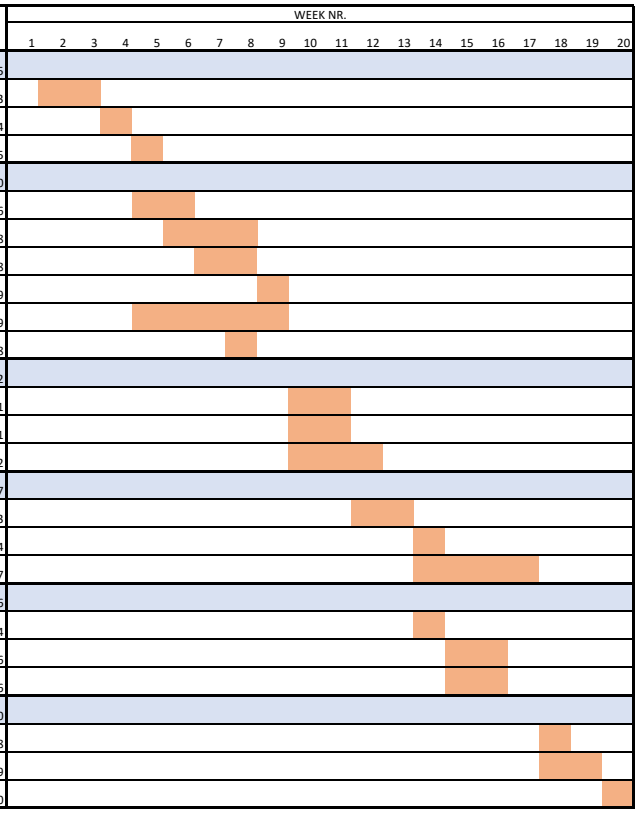


Figure 6.2: The chart displays our initial progression plan.

6.3 Organization of quality assurance

6.3.1 Documentation and standards

During the development process we put an emphasis on client-friendly development practices. Though not a requirement, our client suggested that if our product provided a solid foundation, then they would consider developing it further to add more features. For this reason, we used the commenting and documentation practices of each individual coding language respectively. Additionally, we have included links to the documentation practices of every language we used, in our GitHub repository. To further ensure consistency we will be developing and documenting exclusively in English.

6.3.2 Standardized workflow

To simplify the workflow of our project, we have decided that all tasks will go through the same development steps. This ensures that all work is tracked properly, and follows a clear step-by-step path. A task begins its life cycle as a feature desired by our client. We will then discuss specifications around how this feature operates before we break it down into smaller, more manageable tasks. These tasks are then added to the backlog of our Github project issue board. Our issue board consists of five sections, “Backlog”, “In progress”, “Deadline [XX.XX.XXXX]”, “In review” and “Done”.

When a team member starts working on an issue, they will move it into "In progress" and assign it a size as well as a priority tag. However, since most issues are designed to be small, and are assigned twice a week during our meetings, it is likely that the issue will be moved straight from the backlog to the "Deadline" category. A task in this category is expected to be completed by the updated date in the category title, see figure 6.1 for an example. When a task has been completed, it will be moved to "In review," and the team will review the task during the next Scrumban meeting, to decide if the task has been completed up to the group’s standards. If yes, the issue is closed and moved to "Done". However, if a task needs more work, it will be moved back into either "In progress" or "Deadline," depending on the priority and size of the task.

Chapter 7

Implementation

In this chapter, we will elaborate on how our product works. We will first dissect our backend and provide detailed information on how it is structured, as well as some important functions. We will then do this for each of the frontends and the database as well.

7.1 Backend

Since our product utilizes two separate frontends, both needing to display accurate and updated information, we decided to link them both up to a unified backend. The backend handles all database interactions and performs most of our product's security functions. We will elaborate further on the security aspects of our product in chapter 9.

7.1.1 Overview

The backend consists of the files listed in Appendix L.
At the root level there are Golang configuration files responsible for managing dependencies and storing essential project information. Each folder represents a package and has files containing source code and tests within. These tests are described in chapter 8. The main function is located within `cmd\server.go` and starts the server. The server routes each endpoint to an appropriate "handler" function based on the Uniform Resource Locator (URL).

```
// Route
routes := map[string]func(http.ResponseWriter, *http.Request){
    constants.BASE_PATH:                shared.EndpointHandler,
    constants.SHARED_CREATE_PATH:       shared.CreateDataHandler,
    constants.WEB_LOGIN_PATH:           web.HandlerWebLogin,
    constants.CRYPTOGRAPHY_PATH:        shared.CryptographyHandler,
    constants.MOBILE_VERIFICATION_PATH: mobile.HandlerMobileVerification,
    constants.ADMIN_VERIFICATION_PATH:  web.HandlerVerification,
    constants.PUBLIC_STATUS_PATH:       status.HandlerStatus,
}

for route, routeTo := range routes {
    http.HandleFunc(route, routeTo)
    http.HandleFunc(route+"/", routeTo)
}
```

Figure 7.1: Server routing

7.1.2 Endpoints

Each endpoint has a specific function, but they all have something in common. They read the URL arguments, marshal the URL body, and write back a suitable response. The contents of this suitable response is calculated by functions defined in internal packages.

7.1.3 Internal packages

Constants

The "Constants" package defines terms and functions that are unchanging across each session. These can be changed outside of runtime and dictate things such as database credentials, database metadata, and endpoint URLs.

```
package constants

// Port
const PORT = "8080"

//Path components
const BASE_PATH = "/api"
const USER_PATH = "/user"
const BASE_USER_PATH = BASE_PATH + USER_PATH

// Endpoint paths
const SHARED_CREATE_PATH = BASE_PATH + "/create"                  // "/api/create/"
const PUBLIC_STATUS_PATH = BASE_PATH + "/status"                  // "/api/status"
const CRYPTOGRAPHY_PATH = BASE_PATH + "/cryptography"             // "/api/cryptography"
const MOBILE_VERIFICATION_PATH = BASE_USER_PATH + "/verification"  // "/api/user/verification"
const WEB_PRIMARY_PATH = BASE_USER_PATH + "/admin"                // "/api/user/admin"
const ADMIN_VERIFICATION_PATH = WEB_PRIMARY_PATH + "/verification" // "/api/user/admin/verification"
const WEB_LOGIN_PATH = WEB_PRIMARY_PATH + "/login"                // "/api/user/admin/login"

// File paths
const KEY_FILEPATH = "./keyfile.txt"
```

Figure 7.2: Constants

Request

The "Request" package provides functions for reaching external resources such as the database. Additionally, it handles common exceptions that might arise from missing data and SQL errors.

Cryptography

The "Cryptography" package contains functions for encoding, decoding, encrypting, and decrypting data - in addition to communicating such data to the database and managing encryption keys.

Figure 7.3: Cryptography outline

Globals

The "Globals" package consists of functions that might be required in any context. For this reason, functions here are made as pure functions - so that they don't rely on any specific external components and can work independently. Data treatment, parsing, and database string generation happens here. This package is also the one that is tested the most, as it is responsible for some of the backend's most important tasks.

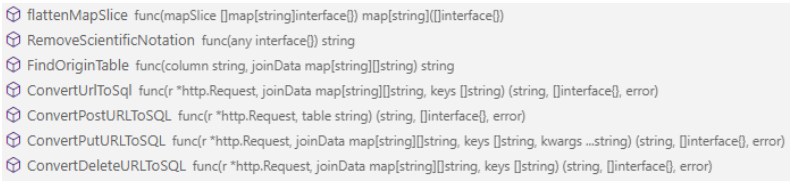


Figure 7.4: Globals outline

7.1.4 External packages

Go standard library

The Golang standard library is plentiful and used all through the backend. It contains functions for handling strings, keeping track of time, marshalling to JSON,

and much more.

MySQL driver

To handle interactions between Golang and SQL, the "Go-MySQL-Driver" is used. It is a commonly used open source driver that formats raw SQL data into more usable Golang data types. It also establishes a connection with the database and automatically handles interrupted connections.

7.1.5 Modularity and expandability

As previously mentioned, every function handling database string generation is a pure function located within the "Globals" package. These functions provide a modular way to create SQL statements without having any prior knowledge of SQL. Additionally, they provide automatic protection against SQL injections, which is further explained in chapter 9.

```
/**
 *  Takes an http request and returns an SQL query with values sepperate
 *  The SQL query PUTS(updates) entries in the given table.
 *
 *  @param r - a pointer to the http request
 *  @param joinData - a map where the key is a string representing the table name and any foreign key references, and the value is a slice containing:
 *       - the name of the table
 *       - the name of the primary key for that table
 *       - any data values requested from that table
 *  @param keys - a slice of strings representing the keys in the joinData map
 *  @param kwargs - Additional arguments presented as strings:
 *      "alterForeignTables" - Alters foreign table values rather than the main table values whenever possible
 *
 *  @returns - an SQL string with placeholders
 *  @returns - a list of values to fit the SQL query
 *  @returns - any potential errors thrown
 */
func ConvertPutURLToSQL(r *http.Request, joinData map[string][]string, keys []string, kwargs ...string) (string, []interface{}, error) {
```

Figure 7.5: The header of the ConvertPutUrlToSql function

Imaged in Figure 7.5, is the header of the `ConvertPutURLToSQL` function. Taking in a wide array of parameters, this function generates a SQL statement that updates the contents of one or more tables in the database. It automatically locates which table each field originates from, joins the required tables for the operation, and covers edge cases such as missing values. If the optional argument `alterForeignTables` is included. It attempts to avoid change to the main table, instead changing the values of foreign fields whenever possible.
This function is one of four string generation functions. The others generating SQL queries for getting, posting, and deleting content. To add a new endpoint that automatically utilizes these functions, the client can simply add the name of the table they wish to expose to a list in the `EndpointHandler` function. Additionally, if the client wishes to join more tables when making database requests, they can simply add the names of the tables in the `SetJoinData` function.

7.1.6 Safety measures

To minimize risk, multiple safety measures are taken advantage of in the backend. The purpose and logic behind these measures are explained in chapter 9, while this section only describes their implementation.

Error handling

Every place an error might occur is immediately followed by a check to see if something went wrong. If everything went as expected, the program continues as normal. Otherwise, an error is generated and returned with an empty payload. Meaningful error messages are used to make debugging easier.

```
// Create the keyfile
file, err := os.Create(constants.KEY_FILEPATH)
if err != nil {
    return nil, errors.New("error creating key file")
}
defer file.Close()

// Generate the key
key, err := generatePrivateKey(KEY_BITS, rand.Reader)
if err != nil {
    return nil, errors.New("error generating key")
}

// Write marshaled key to file
keyMarshalled := x509.MarshalPKCS1PrivateKey(key)
_, err = file.Write(keyMarshalled)
if err != nil {
    return nil, errors.New("error writing to key file")
}
```

Figure 7.6: Examples of error handling

Encoding and encryption

Vulnerable data is encrypted, then encoded before sent. The `PKCS1v15` algorithm is used for the public-key cryptography, while `base64` encoding is used to reduce the size of the encrypted payload. Due to the nature of JSON, special precautions must be taken when encoding data that is going to be marshalled as such. Certain symbols, such as the newline symbol, are temporarily replaced with something that can be marshalled without error. The replaced symbols are put back as they were when decoded.

```
/**
 *  Encodes a set of bytes using base64.
 *  "\n" and "+" are replaced.
 *
 *  @param data - The data to encode.
 *
 *  @return The encoded data.
 */
func EncodeBase64(data []byte) string {
    encoded := base64.URLEncoding.EncodeToString(data)
    encoded_fixed := strings.ReplaceAll(encoded, "+", "{plus}")
    encoded_fixed = strings.ReplaceAll(encoded_fixed, "\n", "{newline}")
    return encoded_fixed
}
```

Figure 7.7: The Encode function

Automatic key handling

The management of encryption keys is automatic and never exposed to any other packages. When a key is requested the Cryptography package first checks if a key file already exists, reading the key from file if it does, as seen in Figure 7.8.

```
// If key file already exists, attempt fetching its key
if _, err := os.Stat(constants.KEY_FILEPATH); err == nil {
    // Open the keyfile
    bytes, err := os.ReadFile(constants.KEY_FILEPATH)
    if err != nil {
        return nil, errors.New("error reading key file")
    }

    // Parse the key
    key, err := x509.ParsePKCS1PrivateKey(bytes)
    if err != nil {
        return nil, errors.New("error parsing key file")
    }

    return key, nil
```

Figure 7.8: The first part of the FetchPrivateKey function

Otherwise, a new key is generated and stored. The key is generated using the Rivest-Shamir-Adleman (RSA) algorithm, as seen in Figure 7.9

```
// If the key file doesn't exist, create it and generate a key
} else {
    // Create the keyfile
    file, err := os.Create(constants.KEY_FILEPATH)
    if err != nil {
        return nil, errors.New("error creating key file")
    }
    defer file.Close()

    // Generate the key
    key, err := generatePrivateKey(KEY_BITS, rand.Reader)
    if err != nil {
        return nil, errors.New("error generating key")
    }

    // Write marshaled key to file
    keyMarshalled := x509.MarshalPKCS1PrivateKey(key)
    _, err = file.Write(keyMarshalled)
    if err != nil {
        return nil, errors.New("error writing to key file")
    }

    return key, nil
}
```

Figure 7.9: The second part of the FetchPrivateKey function

7.1.7 Database

The database consists of SQL code and was made with phpMyAdmin. Details on the database's design can be found in chapter 5. This section describes practical measures that was taken during the implementation of the database.

Location

To reduce the amount of fields in the `container` table, the following assumption is made: If `container.address` is `NULL`, the location of the `container` is instead `container.location_id` - which means the tank is currently not at a `client`.

Transaction history integrity

Once an entity has been mentioned in a `transaction`, it cannot be deleted. This is done by setting every Foreign Key in the `transaction` table to `RESTRICT` mode, and ensures log integrity.

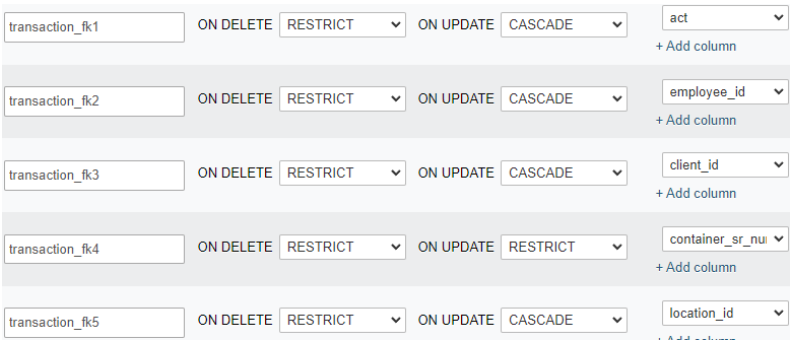


Figure 7.10: The 'transaction' table's foreign keys, all set to RESTRICT on deletion

Imaged below is the same Foreign Keys, but in the `container` table. If a `client` or `location` is deleted from the database, the `client_id` and `location_id` fields of `container` are respectively set to `NULL`.

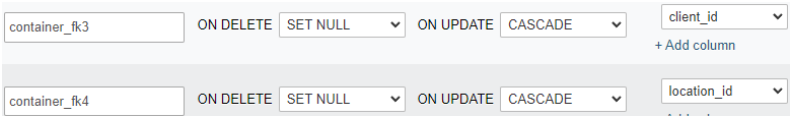


Figure 7.11: Some of the 'container' table's foreign keys, which are set to SET NULL on delete

Minor practical measures

- The tables `requested_keys` and `valid_keys` are separated to allow for different privileges.
- `transaction.employee_id` is set to `NOT NULL` to ensure accountability for each transaction. This was requested by the client.
- `act` entries cannot be deleted once referenced, instead, the `is_active` field is set to `false`.

7.2 Mobile application

The mobile application was a large part of the project, which demanded the attention of multiple group members. To facilitate clear and efficient teamwork a development plan was made, see Appendix J. In this section we will go into further detail on how the mobile application was implemented.

7.2.1 API Communication

The mobile communication with the API consists of 4 major functions. 2 for GET calls, and 2 for PUT and POST. To perform a GET request to the backend you first have to call the `fetchJsonData` function. This function utilizes the `url` package

to open a connection with a provided URL, then send a request with the provided method. In this case, we use the URL provided as a parameter to perform a GET request.

```
/**
 * Gets json string from url
 *
 * Performs simple Get request to provided url and returns a string reading of the data
 *
 * @param urlString the complete url to query
 * @return the json data in string format, send directly to parseJson
 */
Axel Jacobsen
fun fetchJsonData(urlString: String): String {
    //Update internet policy to perform request
    val policy = StrictMode.ThreadPolicy.Builder().permitAll().build()
    StrictMode.setThreadPolicy(policy)
    //Convert url string into url object
    val url = URL(urlString)
    //Open connection on the url string
    val connection = url.openConnection() as HttpURLConnection
    //Set connection method
    connection.requestMethod = "GET"
    //Perform API request
    connection.connect()

    val respCode = connection.responseCode
    if (respCode != 200) {
        Log.e(TAG, msg: "Error in feching data, returned code: $respCode")
        return ""
    }

    val inputStream = connection.inputStream
    return inputStream.bufferedReader().use(BufferedReader::readText)
}
```

Figure 7.12: Function responsible for fetching data from a given endpoint and returning the JSON data.

`fetchJsonData` shown in Figure 7.12, returns a `jsonString` which the next function `parseJsonArray` receives and turns into a `List<Map<String, Any>>`. A `Map<String, Any>` provides the ability to store any type of variable and have it coupled to an identifying string. Storing this in a list lets us contain multiple, unre-

lated chunks of data in one place. This is beneficial considering that JSON utilizes a similar storage format, with a JSON string containing one or multiple objects - each containing a list of Key Value pairs. To summarize, the data is fetched from the API, turned into a string, and then reorganized into a list of many pairs of data. This data can then be displayed by the mobile application.

To PUT or POST, these process has to be done in reverse. For this, we have `generateJson` which takes a `List<Map<String, Any>>` and turns it into a JSON formatted string. To simplify usage, the `generateJson` function is called from within the `makeBackendRequestWithResonse` function, where the actual request is performed. This function takes three parameters, an endpoint to contact, the list of data to send, and which method to perform. By opening a connection the same way as in `fetchJsonData`, we then "write" the JSON string to the `OutputStreamWriter` which handles sending the data itself. Upon completing the request, we receive an updated version of the data as well as a response code in case of errors.

7.2.2 Multitasking

We decided that we wanted to create a multitasking layout for the mobile application. This decision was made because it gives the users the ability to utilize the application in ways that are hard to consider and facilitate by introducing other features. The multitasking feature is implemented using a tabbed interface similar to how web browsers do it. The addition of multitasking has caused some limitations for the development of the mobile application, the largest limitation being that we cannot change activity as easily. To mitigate this, we decided to only use one activity, which is the host for all fragments. The fragments have a host fragment which is the parent of all other fragments, the children are modular pieces of UI. All fragments are hosted and managed by the parent, this results in a modular GUI with different layers of layouts.

The illustration in figure 7.13 shows four different fragments inside our GUI, the host is green, tank is red, camera is blue and mini-act-log is yellow. The host fragment's purpose is to control the menu navigation and be a parent for various child fragments. In this example the tank fragment is a child of the host, tank fragment is also the parent of camera and mini-act-log. By dividing the different layouts into fragments, they can easily be reused, replaced and changed. To achieve the multitasking GUI we designed, it was strictly necessary to make a modular application. By making the application modular we have made the application more complex, by introducing more files and cross communication between them.

7.2.3 Recycler views

To create different lists of elements, we have utilized Android's recycler views to create these lists in a dynamic manner. This allows us to make modular and

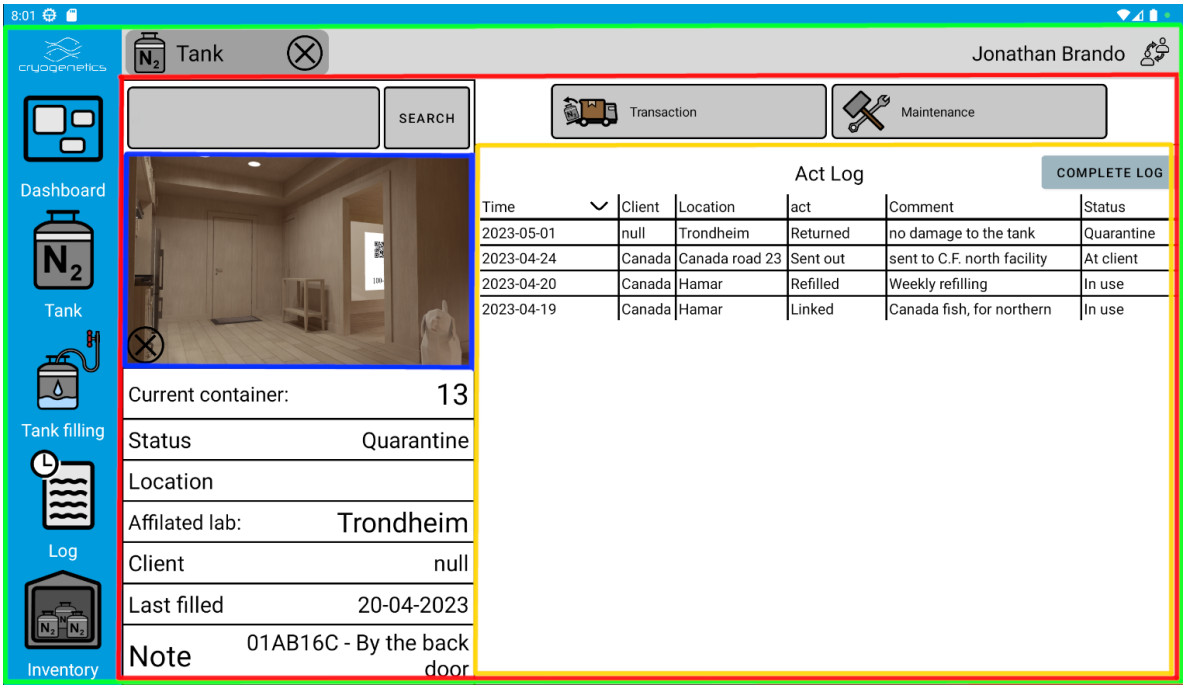


Figure 7.13: Screenshot of tank page in Android emulator, lines have been added to illustrate different fragments.]

interactable lists of elements with different elements to serve various purposes. For example, on the multiple tank filling page we have a list of all tanks which have been scanned or added by search. This list can be modified by checking boxes, clicking buttons in the list or by clicking the tool options above the list. We also utilize recycler views to create inventory and act log tables, the benefit of using a recycler view instead of Android's built in table tools here is flexibility of design choices. This was the best solution we found to create the custom design which we had planned for. This comes with the benefit of being able to listen for clicks in the rows of the table, for example to show the complete data in the table or to navigate to the tank which was pressed.

7.2.4 Layouts and drawables

To make a resizable Android application we utilized constraint layouts, constraint layout allows us to make complex layouts with various tools like weighted chains. Constraint layouts are resizable, since they can be constrained to neighboring views, layouts, the parent or invisible guidelines. When a view is constrained between other elements, the width and height values can for example be set to use the "0dp" to utilize the space given or to "wrap-content" to use the minimum amount of space. To customize some elements to a greater extent we have utilized Android shapes to achieve rounded corners and borders on the background of views and layouts. Utilization of weighted chains is crucial to achieve the de-

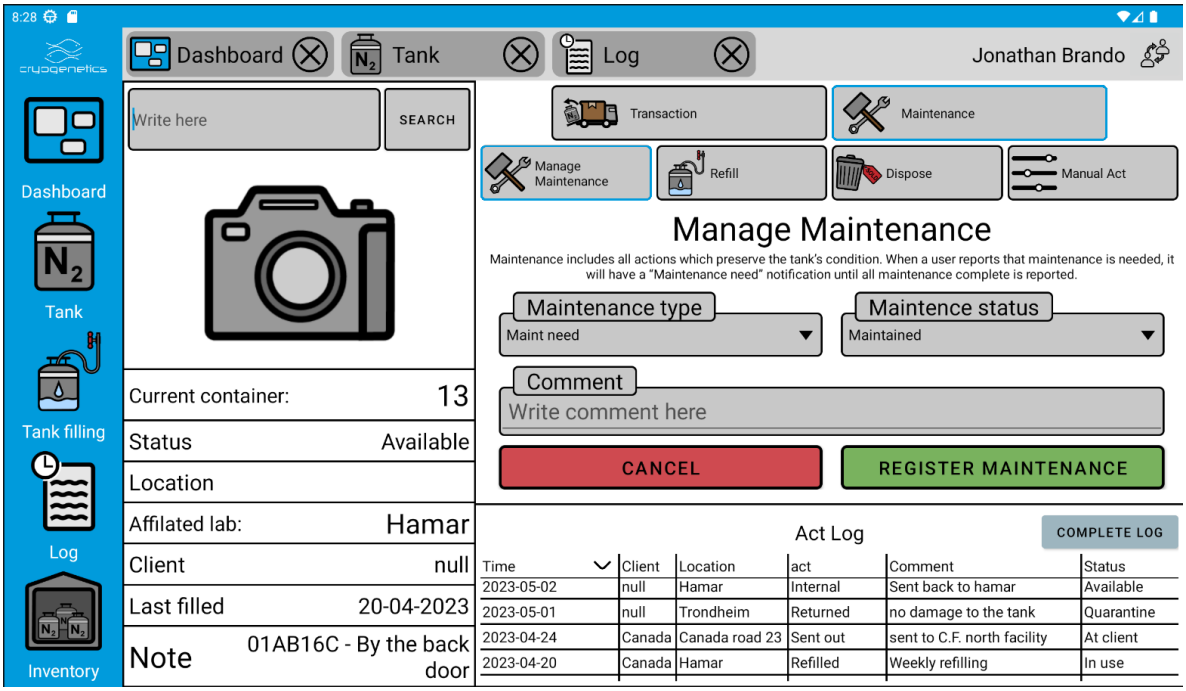


Figure 7.14: Screenshot of tank page in an Android emulator, the manage maintenance option has been chosen in the menu.

sired result, for example, in the various tables all the columns of each row are constrained to their neighbors. By using a weighted chain, the elements of the chain can have different weight relative to each other, which will result in a different utilization of space.
The layout and drawable files can be found in zip attachment, "Code/Frontend/-Mobile application/Logistics/app/src/main/res/".

7.2.5 External dependencies

We utilize mostly the stock Android library, which is included in the android software development kit. In addition to this, we also utilize dependencies from AndroidX, Google and Journeyapps. AndroidX is intended to be an improvement and replacement for the original Android support library, and it contains regular support functionality for Android applications. An example of the functionality offered by AndroidX is the camera functionality we have utilized, which is imported from the Androidx.camera. Utilizing this framework has allowed us to create a live preview of the camera view and to analyze each frame. From Journeyapps, we have imported ZXing Android Embedded, which is a library dedicated to barcode and QR-Code scanning. Utilizing the functionality from AndroidX and Joruneyapps we have implemented quick and modular QR-Code scanning capabilities to our app. We have also utilized Google's Android Material library for the

theme of the application. This allows us to define more colors for the theme of the application. For example, we use Cryogentetics blue for the background on the status bar inside the application, and then we have made the icons like the wifi symbol white to match.
The full list of dependencies can be found in zip attachment, "Code/Frontend/-Mobile application/Logistics/build.gradle".

7.3 Web application

The purpose of the web application in this project is to act as an administrative space. The client requested the domain to be capable of the following:

- Reveal data that is stored on the database.
- Manipulate data.
- Print out QR-Codes linked to specific containers.
- Generate a report to track what locations the containers have been in.
- Authorize mobile devices.

Our previous experience with webapp development came from the PROG2053 course: Web-technologies. Here we learned how to create a webapp using JavaScript and the React library, widely known and used tools in the industry. We decided to use these tools again, as they are well-known and documented, allowing us to utilize many comprehensive guides on how to use them most efficiently.

We used the React Router feature to make a Single-Page Application (SPA). This reduced the loading and development time by separating the backend from the webapp. ReactJS also allows us to utilize Material User-Interface (M-UI), a component library which simplifies the Cascading Style Sheet (CSS) and HyperText Markup Language (HTML) aspect of the webapp. These are respectively used for defining the style and layout of the web application.

We chose to use M-UI due to its vast documentation and popularity, the style fitting the client's request, and its flexibility. React uses the Fetch API to send its HTTP calls. Fetch is built into the ReactJS framework, and provides highly flexible construction of Representational State Transfer (REST) calls, as long as a URL to the API is included.

7.3.1 Show data

Retrieving data

When the admin navigates to a page where the data is displayed (e.g. /transaction), we call a GET function to that item type's URL (e.g server-name/api/transaction). The JSON data that is inside the response is saved in an array.

```
React.useEffect(() => {
  async function fetchRowData() {
    try {
      const response = await fetchData('/api/transaction', 'GET');
      if (response == null){
        console.log('No data available.'); // Log error message instead of returning JSX
      }
      else {
        setRows(response); // Update "rows" state with fetched data
      }
    } catch (error) {
      console.error(error);
    }
  }
  fetchRowData();
}, []);
```

Figure 7.15: The fetchData function is called once when the component is rendered (indicated by the empty array at the end of the useEffect)

Table structure

The table is designed and provided by M-UI, and includes a table head, table content and toolbar. The table head is where we define what columns will be displayed, through headCells. The ID assigned to the headCells is what binds the JSON data to the table when the rendering begins, so it is important that they share the same ID/name (e.g. both the headCell and JSON data name for the container name field are called `container_name`). Should the values be renamed on the database or on the server, these IDs must be renamed as well.

```
return (
  <TableHead>
    <TableRow>
      {headCells.map((headCell) => (
        <TableCell
          key={headCell.id}
          align='center'
          padding={headCell.disablePadding ? 'none' : 'normal'}
          sortDirection={orderBy === headCell.id ? order : false}
        >
          <TableSortLabel
            active={orderBy === headCell.id}
            direction={orderBy === headCell.id ? order : 'asc'}
            onClick={createSortHandler(headCell.id)}
          >
            {headCell.label}
            {orderBy === headCell.id ? (
              <Box component="span" sx={visuallyHidden}>
                {order === 'desc' ? 'sorted descending' : 'sorted ascending'}
              </Box>
            ) : null}
          </TableSortLabel>
        </TableCell>
      ))}
    </TableRow>
  </TableHead>
);
```

Figure 7.16: The EnhancedTableHead returned values

Local sorting

The table head allows users to sort the locally stored data in ascending or descending order. By clicking on the names of the desired field, the attribute of which to sort by is saved, and the table is rearranged. When sorted in ascending order, the smallest number, earliest date or earliest letter of the alphabet will appear at the top of the list. This is reversed for descending sorting order. The table head only allows for one value to be set as the sorted value, as sorting multiple values through this interface was disorienting and confusing.

```
const handleRequestSort = (event, property) => {
  const isAsc = orderBy === property && order === 'asc';
  setOrder(isAsc ? 'desc' : 'asc'); //What way is the table sorted?
  setOrderBy(property); //What are we sorting by?
};
```

Figure 7.17: The handler for the Sort functionality

Local searching

We placed the local search feature in the top right corner of the table Toolbar. An admin can search by entering a query into the text box. When changed, the data will then be filtered through the `filterRows` value, where the output is all the rows that have matching data. The values the data is filtered through are determ-

ined by the `const filterRow`.

```
//DEFINE WHAT THE COLLUMNS ARE FILTERED IN SEARCH
const filterRows = (row) => {
  return (
    row.container_model_name.toLowerCase().includes(searchTerm.toLowerCase()) ||
    row.container_status_name.toLowerCase().includes(searchTerm.toLowerCase()) ||
    row.client_name.toLowerCase().includes(searchTerm.toLowerCase()) ||
    row.address.toLowerCase().includes(searchTerm.toLowerCase()) ||
    row.comment.toLowerCase().includes(searchTerm.toLowerCase()) ||
    row.location_name.toLowerCase().includes(searchTerm.toLowerCase())
  );
};
const filteredRows = rows.filter(filterRows);
```

Figure 7.18: The filterRow const in the Container component

7.3.2 Manipulate data

All types of data manipulation are done in their own "popup", called a Modal. A Modal is a message box that appears in front of all other content on the screen, allowing the user to temporarily commit operations without losing their location on the previous screen. The Modal used in this project are provided by M-UI, as they are very simple to use. The Modal is rendered when its parent component is rendered, but remains invisible until it is "opened". By passing a Boolean value `openModal` into the Modal component's "props", we can later reveal the Modal by changing this value to `true`.

```
return (
  <Modal open={Boolean(props.selectedRow)} onClose={handleCloseModal}  aria-labelledby="modal-modal-title" aria-describedby

    <Box sx={style}>
      <Typography id="modal-modal-title" variant="h6" component="h2" >
        Print QR code
      </Typography>

      <canvas ref={canvasRef} width={512} height={700} />

      <Button variant="contained" sx={{ m: 2 }} color="error" onClick={handleCloseModal}>Cancel</Button>

      <Button variant="contained" sx={{ m: 2 }} color="success" onClick={handleConfirmModal}>Confirm and Download</Button>
      </Box>

  </Modal>
```

Figure 7.19: Example of an M-UI Modal object

All Modals are given Props, which is data that the parent component provides to its child. This works much like arguments do in functions, as it allows us to customize the children in our desired way. Props like `openModal` and `handleCloseModal` are used in every Modal in this project. `handleCloseModal` is the handler for when the user closes the Modal, either through clicking on the outside of the Modal or pressing a button that would close it. This handler calls the parent's `onCloseModal` function, which makes another HTTP GET call, refreshing the page with the most

recent data. `handleCloseModal` also "cleans" all the input fields the user can access, emptying them for the next time the Modal opens. If a Modal is intended to use or alter an existing object, the user selected row from the table is also passed through the props.

Creating an item

If an admin wants to add an item to the database, they click the green "Add item" button on its respective page. The admin is presented with `TextFields` and Lists to fill out the information about the specified item. If said item must contain an SQL foreign key in the database, then the list shows the name of that foreign key ID. To accomplish this, we use the Create endpoint of our backend, which gives the names, IDs, and other values that the foreign keys have (e.g. Client names and IDs). When we receive that data, we put them into `MenuItems`, which are shown in Lists. This is demonstrated in Figure 7.20. When an item in that list is selected, we store its ID in order to use them when sending the HTTP request.

```
<TextField
    select
    required
    label="Container model"
    id="container-model"
    fullWidth
    value={containerModel}
    onChange={(event) => setContainerModel(event.target.value)}
  >
  <MenuItem value={""} disabled>Select a container model</MenuItem>
  {containerModelOptions.map(option => (
    <MenuItem key={option.value} value={option.value}>{option.label}</MenuItem>
  ))}
</TextField>
```

Figure 7.20: Here the data gathered from another GET call (containerModelOptions) is inserted into the list

When all of the required fields are filled, the admin is allowed to press the Confirm button. This creates a JSON array containing the data that was input, before sending it to the server as a POST request.
Should an error occur, the admin will receive an Error popup with the error code. If the error code is 409, the popup reveals that the identification digit is already occupied by a different object. This means that the object that was attempted to be created had the same Primary Key as a different object in the database, causing an SQL error.

```
const handleConfirmModal = async () => {
  try {
    const data = [{
      act_name: actName,
      description: actDescription,
      is_active: 1
    }];
    await fetchData("/api/act", 'POST', data)
    handleCloseModal()
  } catch (error) {
    alert(`Error: ${error.message}`);
  }
}
```

Figure 7.21: The HTTP structure of an Add Modal

Editing an item

Each row on each table has an edit button ,except for Transaction, as they are not to be altered to keep accountability on the users and admins. When pressed the Modal is given the selected rows' data and inserts them into `TextFields` and Lists. The admin can then change any non-primary value of that item. The updated values are inserted into a JSON array, where the `primary` field is always the primary key of the given database table.

```
const handleConfirmModal = async () => {
  try {
    const data = [{
      primary: "container_model_name",
      container_model_name: props.selectedRow.container_model_name,
      refill_interval: refill,
      liter_capacity: liter
    }];
    await fetchData("/api/container_model", 'PUT', data);
    handleCloseModal()
  } catch (error) {
    alert(`Error: ${error.message}`);
  }
}
```

Figure 7.22: A Modal with the Edit feature

Deleting an item

For an item to be deleted, the item in question must not have been used or registered on any object or transaction in the database. If this is not the case, it simply will not be deleted, throwing an SQL error. The Admin receives this warning in the Modal when the Modal opens. If they press the Delete button in the Modal, the items' ID is sent through a DELETE HTTP call, deleting it from the database.

7.3.3 Print out Tank Labels

In our original design we imagined having a separate page for Quick Response Code (QR-Code) generation, where the admin will receive a list of the tanks in their storage. This idea proved to be more complicated than necessary, and instead, we went with the following design. On the Container page, we added another button behind the objects' Edit button on the table. This button opens a Modal called `PrintModal.js`, taking the item data as a prop. The Modal will instantly generate a QR-Code using the qrcode-generator library, and display it to the admin on a canvas, with the tank ID and model liter capacity underneath it. The QR-Code contains a singular value, the container serial number, which is universally unique.

We chose this third-party library as it was the simplest to understand for such a

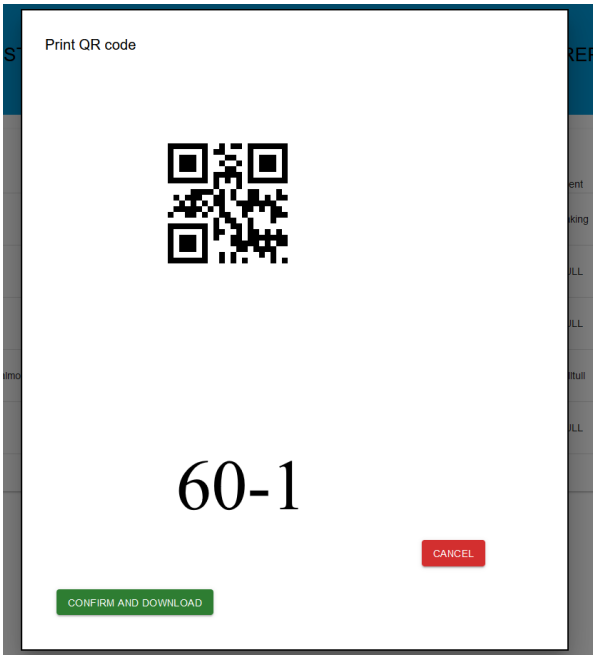


Figure 7.23: Example of the generated Label

complex task, more so than its alternatives. When the admin confirms that this is the correct QR-Code, they can press the generate function. This downloads the

canvas with its QR-Code and text as a Portable Network Graphics (PNG) file that they can print out on a sticky surface and attach to a tank.

7.3.4 Generate a monthly report

Our client requested the ability to generate monthly reports that track the locations and movement of their tanks in the selected time frame. This feature would allow them to more easily create invoices for their clients, as they would see exactly which tanks are in their storage, and which ones are shipped out to other places. Our solution accomplishes this by retrieving all the recorded transactions from the selected month and filtering out those that have no client attached to them. This is important, since if no client is attached to the tank, then there is nobody to direct the invoice to in that time period. The remaining transactions are sorted by tank and date before we begin assembling the Comma Separated Values (CSV) file. We chose to use a CSV file as the output of this functionality, as they are simple to create and easy to port over to other programs like Microsoft Excel, which is what the client has been using to keep track of their containers movement so far.

```
const fullReport = clientNames.map((clientName) => {
    // Call createSectionPerClient function for each clientName
    const filteredDataForClient = filteredData.filter(transaction => transaction.client_name === clientName);
    return createSectionPerClient(filteredDataForClient);
});

//Removes the commas between the array indexes
let reportString = ""
fullReport.map((table)=>(
    reportString+=table
))

    //add the header to the final report
const finalReport= csvHeader + reportString

// Download CSV file using CSVLink component
const csvData = 'data:text/csv;charset=utf-8,' + encodeURIComponent(finalReport);
const downloadLink = document.createElement('a');
downloadLink.href = csvData;
downloadLink.download = selectedMonth+'report.csv'; // Set the filename for the downloaded CSV file
document.body.appendChild(downloadLink);
downloadLink.click();
document.body.removeChild(downloadLink);
```

Figure 7.24: The process of creating the CSV file

For each client we retrieve, we create a "section" of transactions that only includes that client. In this section, we track the movement of the containers based on the `act` in the transaction, as only the acts "Returned" and "Sent Out" correlate with a change in location in the transaction. Any container that does not have one of these two acts, is counted as being stationary for the duration of that month and receives a single row with the location being the last transaction's location name. When a location has the act "Sent Out", the location field changes from the current location name to "Sent Out", meaning that the container is at a client. This is vice versa when the act is "Returned". In order to register the "end date", we needed to

catalog when the next mention of the "Sent out", "Returned" or "Discarded" acts, and write their date as the end date. This means that the action ended on the date when the location was changed again, or the container was discarded.

```
// Create empty row with client name
const clientRow = `${client[0].client_name},,,,`;

// Create CSV data rows
const csvData = sortedTransactions.map((transaction, index) => {
  const capacityTempId = transaction.liter_capacity + '-' + transaction.id;
  const date = transaction.date;
  const endDate = SetEndDate(transaction.transaction_id,transaction.container_sr_number, transaction.date);
  const location = transaction.container_status_name === 'At client' ? 'At client' : transaction.location_name;
  const endDatefromString = `${endDateString.slice(8, 10)}/${endDateString.slice(5, 7)}/${endDateString.slice(0, 4)}`;
  const startDatefromString = `${startDateString.slice(8, 10)}/${startDateString.slice(5, 7)}/${startDateString.slice(0, 4)}`;
```

Figure 7.25: The client row displays only the name of the client, the csvData scrolls through all the transactions and returns the correct format

This functionality works with the expectation that each container (that is connected to a client) in the storage of Cryogenetics has at least one transaction registered in the span of the month. Should the container stay "Idle" for the whole month, then we assume it is not being used.

7.3.5 Authentication keys

When loading the `devices.js` component, the API is called with the `admin/verification` URL to GET all the values stored in the `requested_keys` table. These values are stored in an array and displayed on a toggle-able list, made using the M-UI List components. Each item fetched from the API is placed in a list item, along with a checkbox that acts as a button. We used check-boxes to allow the admin to select multiple keys at the same time, should they so desire. All items that have an active checkbox are stored in an array, which is sent in an HTTP POST request to the API. When the request has been completed, the list is refreshed with the remaining keys from the API.

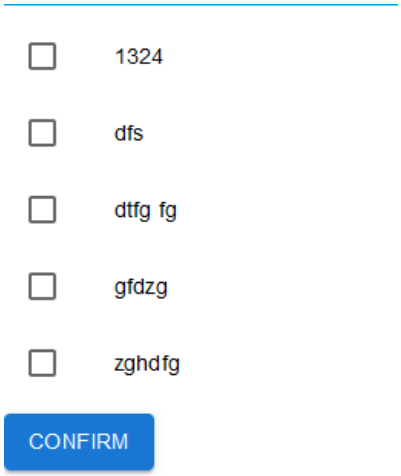


Figure 7.26: List of key names with example data

Chapter 8

Testing

In the testing chapter, we will look at the different testing methods we have applied during this project. This includes both in-person tests and unit tests for the backend functions.

8.1 User testing

8.1.1 Plan for Inspections and Testing

We have been granted permission to conduct testing with some staff members from Cryogenetics. This was a great help for our user tests, as the employees are our main target for users in the finished product (Adults ranging between the ages of 25- to 60). We conducted these tests on both our low-fidelity mobile application model as well as our first mobile- and web-application draft. With the data we received from the tests, we can add, remove or alter functionality to better serve our client's needs.

User testing the design and creating multiple iterations are essential steps to user-centered design, therefore we have created three design prototypes and conducted user testing with eight participants. Five of the participants were Cryogenetics employees, who will be using the application if implemented. The participants' ages ranged from young adults to middle-aged adults, with varying levels of digital proficiency. To obtain comprehensive data, physical user testing sessions were conducted for all participants except for one, who was tested digitally due to geographical location.

8.1.2 Testing at Cryogenetics

The 17th of February we visited Cryogenetics in Hamar to user test 4 of their employees. In preparation for this user testing session, we made an intractable prototype of the mobile app. To give the most authentic user experience, we utilized an older tablet to run the prototype in Figma and then gave the participants the tablet. We then asked the participants to explore the prototype on their own,

before we gave them tasks and asked questions. The data gathered was useful and can be found in Appendix I, but we were hoping for more new ideas and ways to improve the program.

8.1.3 Additional user testing in Trondheim

To gather more data, we performed user tests on our advisor, a Cryogenetics employee in Trondheim and two fellow students. To user test the employee in Trondheim, we chose to send the prototype and ask the employee to open it on their computer and share their screen with us. The other participants were tested physically with the prototype on a computer. Both testing procedures were successful and gave valuable insight. The employees were more concerned with functionality, while the others were more focused on design.

8.1.4 Feedback from User Testing

- Most participants praised the design and color choices, which proved to be satisfactory.
- Cryogenetics employees were thrilled to see a personalized app, which included custom icons and features the company's colors and logo.
- The inclusion of comments when performing actions proved to be a significant feature, as it allows supplementary information to be provided.
- Users found the app easy to navigate, and the interface was user-friendly and intuitive.
- All preconfigured acts were useful, including manual acts which gave the option to make more complex changes.
- The participants had no suggestions for actions to add as a preconfigured option in the menu on the tank page.
- The possibility of adding clients, operators, and tank models proved to be necessary, adding actions and statuses was only nice to have.
- The participants appreciated the effort we had made to display the right information at the right time.
- The addition of multitasking proved to be a valuable feature some operators found useful.
- The menu on the tank page left dead space and was not informative.
- Participants indicated that the serial number is rarely used, and should only be displayed on the tank page.
- One user complained that black text on a red button was hard to read.
- Users expressed the need for an additional value for tanks, which would express the rate liquid nitrogen evaporates.
- It could be beneficial to add more features to separate operations from different laboratories.
- Production date for each tank could be beneficial, as tanks get worn out with time.

- Features to display and change the time format could be beneficial for international use in Cryogenetics.
- While we got some praise from participants for the battery indicator, which predicts the level of liquid nitrogen in the tanks, some said it was not strictly necessary, others said it would be nice to have several places.
- We got a suggestion to add a preview of the act before it is confirmed, which would make the app more transparent for the operators.

8.1.5 Actions taken after user testing

- The menu on the tank page was redesigned to be more informative and to utilize the space more efficiently.
- An option to change the "Cryo dissipation rate" was added to the design to express the rate of liquid nitrogen evaporation.
- Production date was added to the web application to manage worn-out tanks.
- Buttons with red backgrounds were changed to a lighter red.
- An option to press a row in a table to display the information in a pop-up was added, which allows users to see information that cannot fit in the table layout.
- A preview of the act is now provided before it is confirmed, which gives the operators the opportunity to preview the change before it is made.

8.1.6 User testing conclusion

The product was deemed successful after user testing, according to the client's employees, all necessary features were included. Multitasking, comments when performing actions and various other features mentioned above proved to be desirable or necessary. The design and color choices were adequate, and the GUIs were user-friendly and intuitive.

8.2 Backend tests

8.2.1 Method

We chose unit testing to be our main method of testing the backend. The main workload of the backend is done by a few pure functions located within the `Globals` package. Testing these functions with a broad array of different subtests was prioritized over integration tests and End-to-End (E2E) tests. Integration tests are useful for finding errors that might arise when different components interact with each other. The backend does not have much interaction between different components, instead, it has a flat - but modular - layout. The exact architecture of the backend is described in chapter 6. E2E tests are helpful in verifying that the entire system works from the end user's perspective, but are time-consuming to

make. Focus was instead put into making sure the core functions worked exactly as intended, regardless of the parameters given. This way the client may choose to continue with an incremental test approach, building more complex tests on the solid foundation of tests we made.

8.2.2 Code

Before testing could begin the functions had to be made entirely pure. Since Golang's `map` data type is fundamentally unordered, special measures have been taken to order it so that no factors are unaccounted for. First, an alphabetically sorted list of the `map`'s keys was made.

```
urlData := r.URL.Query()
urlDataKeys := make([]string, 0, len(urlData))
for k := range urlData {
    urlDataKeys = append(urlDataKeys, k)
}
sort.Strings(urlDataKeys)
```

Figure 8.1: Alphabetically sorting a map's keys

This sorted list is then iterated rather than the `map` itself, and the value of that key is fetched for each iteration.

```
for _, k := range urlDataKeys {
    v := urlData[k]
```

Figure 8.2: Iterating a sorted map

To mock HTTP requests, the `httptest` package from Golang's standard library was used. This package provides methods for constructing fake requests as if they were made by an external actor.

```
// No body supplied
  name: "No body",
  r: httptest.NewRequest(
      http.MethodPost,
      "localhost:8080/api/container",
      strings.NewReader(``),
  ),
```

Figure 8.3: Mocking an HTTP request

A testing environment was created for each function tested, allowing for rapid development of new subtests. These subtests require a name, input, and expected output. The expected output includes any error which may arise.

```
},
{ // No body.
    name: "no body",
    r: httptest.NewRequest(
        http.MethodPost,
        "localhost:8080/api/container",
        strings.NewReader(``),
    ),
    kwargs:           make([]string, 0),
    expectedSqlQuery: "",
    expectedSqlArgs:  []interface{}{},
    expectedErr:      "EOF",
},
{ // Invalid body.
    name: "invalid body",
    r: httptest.NewRequest(
        http.MethodPost,
        "localhost:8080/api/container",
        strings.NewReader(`this is an invalid body since it doesn't follow JSON formatting :(`),
    ),
    kwargs:           make([]string, 0),
    expectedSqlQuery: "",
    expectedSqlArgs:  []interface{}{},
    expectedErr:      "invalid character 'h' in literal true (expecting 'r')",
},
```

Figure 8.4: Two subtests for the function 'ConvertPostURLToSql'

8.2.3 Results

Most edge cases are covered by three or four tests per function, as well as one test testing the way the function is normally used. In total there are 13 tests divided among four functions, resulting in a 91.3% test coverage of all `global` functions. See Appendix K for the complete test coverage report of the "Globals" package.

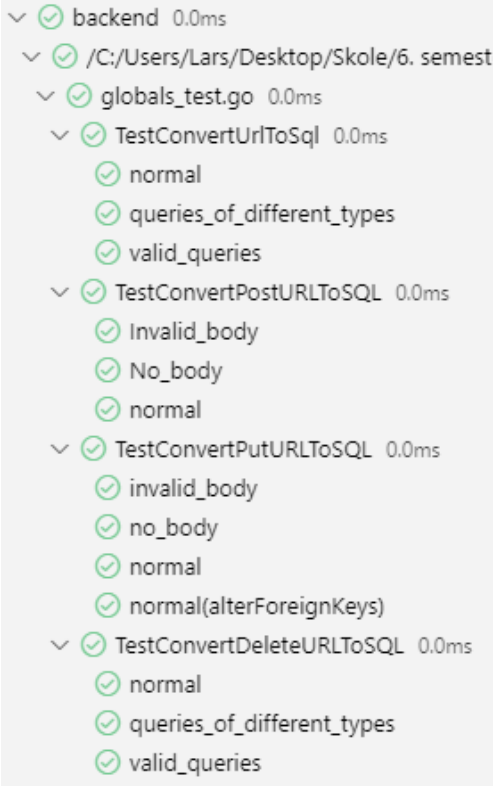


Figure 8.5: All tests executing properly

Chapter 9

Security

Since our product will be handling information our client has deemed sensitive, we decided to prioritize security. Our security is mainly focused on two areas, authorization and identification. This chapter goes into detail about the decisions and measures we have taken in order to secure our client's assets.

9.1 Initial risk analysis

Before starting our project we performed an initial risk analysis. This analysis was performed based on preconceptions and predictions we had for the future of our product, before starting development.

- **Overestimated development speed:**
  As a group we overestimated our own skill and development speed, causing us to fall behind on the development plan.
- **Late and sudden changes in desired product or client wishes:**
  The client realizes late into development that there had been significant miscommunication leading us to develop a product that doesn't suit their needs.
- **Loss of documentation or source code:**
  Somehow all code or documentation is lost or becomes inaccessible for a longer period of time, causing a halt in development.
- **Leaked customer data:**
  An insufficient security protocol causes client data to be accessible to anyone.
- **Loss of group members:**
  A group member becomes inaccessible or decides that they do not wish to work together, causing them to leave the group.
- **Conflicts within the group:**
  A tear in the group causes work to slow down or grind to a halt.
- **Lack of competence:**
  Insufficient competence leads to an unfinished or inferior product.

- **Server crash:**
  Temporarily stored data gets erased or server access is lost.
- **Local data is lost:**
  Local progress is lost causing a setback to previous Git commit.

			Probability	
		Low	Medium	High
Impact	Low	5, 6		9
	Medium	1, 4	7	
	High	2, 3	8	

Table 9.1: Risk analysis, Red: Dangerous, Has to be handled, Impact, Yellow: Medium priority, should be handled or Impact, Green: Unimportant or not dangerous

9.2 Mobile security

9.2.1 Framework

> Rich Internet mobile apps are deployed on the client device but they leverage the backend support infrastructure extensively using communications technologies [23]

When creating the mobile application we decided on making it a *Rich Internet Mobile App*. The option of creating it as a *Browser Based Mobile App* using React Native was considered. This would have the benefit of being cross-platform and utilizing the same programming language as the admin web page. However, it quickly became clear that there was no necessity for cross-platform, as our client preferred Android over alternatives such as iOS. Mainly due to the cost difference, but also due to security concerns.
IOS has a multi-tasking feature known as *backgrounding*. "In iOS backgrounding, iOS takes a screenshot of the application before minimizing it to run in the background" [23]. This could lead to potential vulnerabilities in the application and be a threat to confidentiality.
There are additional security issues that come with creating a browser-based mobile app. "Browser-based mobile applications are relatively more susceptible to traditional web vulnerabilities in addition to threats that come with mobile use cases and device weaknesses" [23]. For these reasons, we ended up scrapping the idea of a browser-based app, because although it would have been simpler, it would also be less secure.

9.2.2 Challenges

When creating mobile applications there are multiple factors to keep in mind which are easily avoidable on desktop devices. For example, data storage is not as secure. This is in part due to the mobile nature of the mobile device, but also because mobile databases are not as mature yet as their counterpart desktop ones. In the words of Paul Mano (2013), "Local databases on most mobile operating systems are not as mature as their desktop counterparts when it comes to the confidentiality assurance capabilities such as encryption" [23]. To mitigate the risk of an employee losing a device containing sensitive information, or a fundamental security flaw leading to such information being leaked, we decided to not keep sensitive information stored on the mobile application.

Another factor we considered when creating the mobile application was that of encryption. "Lack of end-to-end encryption and data-in-motion cryptographic protection between the mobile device and the carrier network or between one device and another can lead to sniffing and tapping attacks, which have been known to lead to information disclosure from mobile applications" [23]. To ensure integrity in the verification process, the devices' identifying information is encrypted before it is sent to the backend, in addition to being communicated with Hypertext Transfer Protocol Secure (HTTPS).

Using a debugger or disassembler, a malicious actor can easily recover hardcoded strings from the mobile application [23]. For this reason, the application is designed in such a way that even if the original source code is retrieved, it poses no threat. This security practice is known as open design, which is the opposite of security by obscurity.

9.2.3 Device attestation

The first time a device starts the app, a unique number is generated and stored in the device's app preferences, which is inaccessible from other apps. Each time the device opens the app it sends an encrypted version of this number to the backend, which then decrypts it and checks if it has been verified before. If it has not been verified the key is added to a list, which an admin on the website can verify at their earliest convenience. The exact details of how the key is encrypted and decrypted is described in section 7.1.3. This is a form of *ownership-based* authentication which does not add any accountability.

9.2.4 Employee codes

When an admin adds an employee, they decide on their personal code. This code is used as a form of knowledge-based identification. The code is only three to four digits long, which means it is not secure enough for proper authentication. It is simply in place to increase accountability and does not prevent a malicious

employee from posing a threat to the system. It does however work as both a deterrent and detective control to accidents which are prone to happen as a result of carelessness.

9.3 Admin security

9.3.1 Two-factor identification

Our wish was to implement Microsoft Multifactor Authentication (MFA), where admins could log into the website using their work-related email. MFA introduces more layers of security past just using an email and password. Our plan was to utilize the Microsoft Authenticator app, which would make users provide a fingerprint and personal code in order to identify and authenticate themselves. Should this solution be implemented in future versions, the risk of outsider attacks would be nullified, as only trusted individuals would gain access. Unfortunately we received complications surrounding the access to our client's Azure network, which is where these features need to be implemented. These complications have resorted in us postponing MFA in the finished prototype.

In place of this feature, as a temporary security measure, we added a simple login page where the admin has to log in with an email and password registered into the database. There is no way to register another email/change password on the website. Before the login request is sent, the password is encrypted with SHA265 hashing. This type of hashing cannot be "decoded", and brute-forcing the password has astronomically low odds of succeeding.

Our server receives the password and email, and checks if the email exists in the database. If it does not, the user must submit another email to proceed. If the email exists, the server hashes the password connected to the email with SHA265, and checks if the hashed passwords are the same. Should they match, then the password is correct, and the user is granted access to the website. This serves as a layer of authentication, making it more difficult for a malicious actor to gain Authorization to the system and reducing risk.

9.3.2 Authentication keys

As described in subsection 9.2.3, new mobile devices are given a code that identifies them. That code is sent to the database as a "requested key". The device cannot access the application until this key is registered in the "valid keys" by an administrator. The administrator gets a list of the devices that want to access the application and can accept them, or leave them be.

Chapter 10

Reflection

In the final chapter of our thesis we will be reviewing our project retrospectively. We will discuss what is good and what could be better. Finally, we will discuss potential future improvements.

10.1 Product analysis

Our final product has been completed according to the primary requirements we defined at the beginning of this project. In collaboration with the client, we carefully developed two graphical user interfaces tailored to their administrators' and warehouse operators' needs. The product is an inventory control system that can help the client keep better track of their liquid nitrogen tanks.

10.1.1 GUI

Initially, this project was not graphically demanding, but we decided to spend significant time planning the graphical design. The reason behind this was to establish the necessary functionality and develop a user-friendly solution for the client's needs. We started the process by making a low-fidelity model which we could pitch to our client, then we made higher-fidelity models to further develop the design. After the initial design and pitch, we continued updating the models to better reflect the desired product. By over-engineering the models of the applications, we may have spent more time than necessary on the design process. The time schedule was also affected by the client's desired date to perform user testing at Cryogenetics in Hamar. However, the lost time was made up by having an accurate model we could discuss with our client during progression meetings. A visual representation of the desired outcome would prove itself to be a great asset for the group as it aligned our goals, helping us work together in an effective manner.

Our mobile application was built around the multi-tasking feature. However, from a graphical design standpoint, this feature is questionable. Although it serves

a practical purpose in case there is high workload and an operator needs to do multiple tasks at once. However, it can be seen as an unnecessary complication and as visual clutter. Had it not been for our client liking the feature, we could have considered not including it to simplify the mobile application. Other than that the mobile application conveys information effectively, considering the small format of the screen. Overall, the design, color choice and layout supports its functions effectively.

The web application's simple visuals are both its greatest strength and weakness. The application is only intended for infrequent operations, such as adding new employees, clients, tanks or mobile devices. For this reason, the priority is to effectively convey functions and make these processes as quick and easy as possible. However, this results in an uninteresting design. There is a lot of deadroom on the website that could be filled, either using notifications or by displaying important notifications. This would, however, go past the scope of what this application's intent is. In the end, keeping it simple and effective has taken priority over making it visually intriguing.

10.1.2 Backend

Choice of programming language

During the planning phase we considered using Node.js instead of Golang. This would have the benefit of using the same programming language as the frontend website, making development faster as functions could have been reused. Node.js is also great for building performance-friendly websites with heavy visuals. However, due to the simplistic design of the website this was not necessary. Additionally, Golang is a compiled language while Node.js is interpreted. This makes Golang's performance slightly better than Node.js' when doing data-heavy calculations, which lead to it being chosen instead of Node.js.

JoinData

The `joinData` function was implemented in order to allow for easy change of "joins" between tables, the exact details of how this is achieved is described in subsection 7.1.5. However, this also means that every time the database is updated, the joinData section must be changed as well. This makes altering the database more troublesome than it should be, and is in retrospect something we should have done differently. For example, we could have used SQL's "views". This is a feature which allows for the creation of virtual tables which remain inactive until invoked with a given input. This could have removed the need to change the backend when updating the database.

10.1.3 Database

Choice of database

During the planning phase noSQL alternatives such as mongoDB and firebase were considered. These would have been simpler to design, completely removing the need for conceptual modeling and greatly reducing the amount of work required to create and update the database. However, we decided to use SQL as it is more structured and vertically scalable. In retrospect, this was the right choice as it leaves the client with a product which is easier to expand on.

Normalization

When normalizing the database there were several challenges. Many columns are weakly connected and have to be separate even though they are directly dependent most of the time. This has lead to some recurring columns in separate tables. Additionally, some tables had to be un-normalized to allow for separate permissions. The end result was a very mixed database, with some normalized parts and some that were not. As with `joinData`, we should have taken better use of SQL's view feature to reduce the amount of clutter.

Wide vs narrow tables

To reduce the amount of database joins, we decided on making *wide* tables rather than *narrow* tables. In retrospect, some tables should have been made more narrow by dividing them into multiple sub-tables. For example, `container` and `transaction` contains almost half of all columns. This leads to worse performance when querying, but reduces the total amount of data stored. For our project a more narrow approach would be better suited.

10.1.4 Web application

We are satisfied with how the admin application have turned out. Especially the color scheme, functionality, and general layout. However, there are a few parts of our program that we would like to change to boost code accessibility and improve the overall structure. The first and biggest issue is the presence of database value names in the code. As described in chapter 7 Implementation, should a value change its name on the database, it would need to be renamed across the entire website code base as well. This could be fixed by utilizing the data structures of the desired objects more efficiently, generating the tables and Modal based on the names and values by passing them throughout the program as props, instead of the hard-coded way it stands now.

As described in chapter 9 Security, we wanted to implement Microsoft Azure Multifactor Authentication (MFA) as the primary authorization method. Complic-

ations regarding access to Microsoft Azure delayed this plan to the point of replacement. The structure is in place to implement this feature, to hopefully ease the method for our client, but we are disappointed by the missed opportunity to work with this technology. The team is satisfied with the resulting website and has learned a lot from the improvements we wish to make.

10.1.5 Mobile application

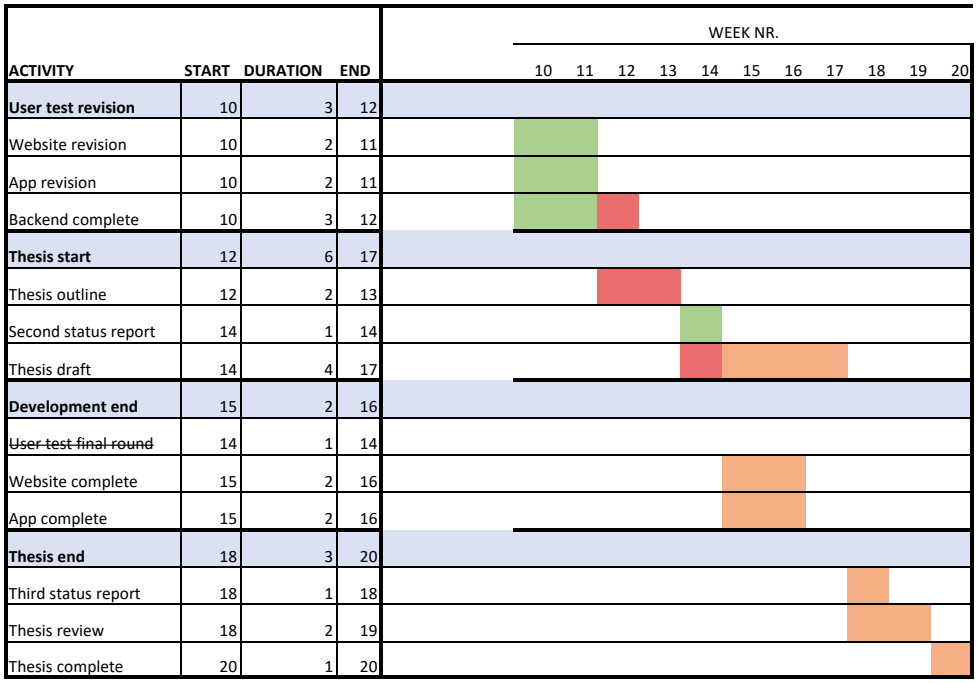
The structuring of the mobile application had one major constraint during development, the multitasking feature. A feature we proposed to our client was the ability to swap between "tabs" inside the app, like in a web browser. Since we got positive feedback for this feature, we decided to implement the tabbed interface which we had designed. To make a multitasking interface in Android we utilized fragments and various other methods to divide the application into modular pieces. This helped us minimize the files and lines of code required for the desired solution while creating a responsive and detailed GUI. We were able to implement all strictly necessary features and make the application functional.

To achieve a seamless multitasking user experience we choose to utilize fragments, which allowed us to seamlessly switch between different pages without loading. The alternative to using multiple fragments to achieve this layout would be to use Androids Jetpack Compose with an activity. Jetpack Compose would introduce a sharp learning curve to the project since we do not have much experience using it to create GUI. Compose would also allow us to generate the GUI using Kotlin instead of XML, and therefore allow us to create more intricate code. We tried out a couple of samples from the official Jetpack Compose samples, and we were not impressed by the performance of the applications we tried. Multiple people would also have to learn the new technology to contribute to the app. Based on the arguments mentioned, we came to the conclusion that utilizing Jetpack Compose was not the right choice for this project. We decided it was best to stick with eXtensible Markup Language (XML) since it would allow us to achieve more progress quickly. We were also quite familiar with XML, as it is the standard for Android UI before Jetpack Compose was introduced.

10.1.6 Project schedule changes

Figure 6.2 shows our initial plan in the form of a GANTT chart. We followed this while maintaining steady progress for the first 6 weeks. However, after a meeting with our advisor we decided to complete the requirement specification part of the final thesis early. Since the requirement specifications cover the goals for our project, writing this early let us measure our progress not only in time but also milestones that we set in the text. Since this had not been included in our initial time estimates for the project, it ended up causing a delay for all ongoing work. We included a chart to show how much time we spent across the project

where we can see the increased workload as the project progressed M. To ensure that we could meet future goals without causing large delays, we revisited the GANTT chart. Figure 10.1 shows how we updated the GANTT to compensate for the delays.



ACTIVITY	START	DURATION	END
User test revision	10	3	12
Website revision	10	2	11
App revision	10	2	11
Backend complete	10	3	12
Thesis start	12	6	17
Thesis outline	12	2	13
Second status report	14	1	14
Thesis draft	14	4	17
Development end	15	2	16
~~User test final round~~	14	1	14
Website complete	15	2	16
App complete	15	2	16
Thesis end	18	3	20
Third status report	18	1	18
Thesis review	18	2	19
Thesis complete	20	1	20

Figure 10.1: The revised GANTT chart for the affected weeks, changes are marked in red

Another unforeseen delay arose around the same time. To access Microsoft Azure we needed to be given access by the client. The request for access began almost as soon as we started work, but there were potential security risks in giving us access to their own server instance. The process of gaining access took time and had to go through multiple confirmation steps within Cryogenetics. Around March we should have been granted access, but when we tested it there were still problems. In the end we found that there had simply been a delay before we were granted full access. With the problem solved, we managed to host a version of our product on an enclosed Azure instance. This was a relief as it made handing over the product to our client easier, allowing us to deploy it for them on their Microsoft Azure instance.

The final delaying factor was the other subject we had this semester, IØ-2000. Before the final semester started we asked around trying to gauge the workload required for this subject. However, it seemed that no one was quite sure. In the end, we were under the impression that the subject was largely practical and would not demand too much time outside of the allocated classes. Though as

we would find out, that was not the case. The subject involved three separate assignments which demanded significant time, causing minor delays throughout the semester. We understand that this is a delay that most groups likely suffered, but as it was an outside factor of our initial time estimates, we found it necessary to bring up.

10.2 Project results

10.2.1 Group results

At the beginning of this project, we set primary roles and responsibilities for each of the group members. In our initial plan, we intended for these roles to help guide progress and ensure that we always had a "leader" for each product component. This turned out to be a successful arrangement that was helpful during development. Since we had also agreed on a time of day where all members had to be available, there was never an issue reaching a "leader" in the case of development-related questions. This meant that we had effectively consolidated knowledge of each major component into a group member each. In addition to this, we also worked across components. Thus, if a component ended up behind schedule we could delegate more group members to help. The ability to perform these swaps was primarily due to our choice of development method 6. With an Agile development method we had significantly more flexibility than we otherwise would have. In the end, we cooperated well as a group. If someone was struggling, we would help, and if someone was slacking, we were not afraid to give them a warning. This helped us keep a balance in the group which kept everyone motivated to finish this thesis together.

10.2.2 Client reception

During this project, we have maintained steady communication with our client through bi-weekly meetings. During these meetings, we have received feedback on each feature we developed during the project. This ensured that we never strayed off the course our of our client's brief. Thus, when the time came to deliver the product, we gave them a short demo of all the features and agreed upon a date to deploy the product on their servers. The only issue was our lack of time. As the project was approaching the end, we could not spare the time to deploy it for them locally as it would demand a whole day, travel included. Considering that we managed to host a version on a separate Microsoft Azure instance they had access to, we decided to instead perform the exchange digitally on the 1st of June. This way we will be in much less of a rush and can go through the features and product structure at a slower pace.

10.2.3 Future potential

Our program shows significant potential for future development. Even though it is functional in its current form, it still requires some polishing. As mentioned, security is the main issue. Though we implemented significant measures to stop potential malicious actors, there is still room for improvement. An example of an additional security feature would be linking the application up to Microsoft such that a Microsoft account can be used to log in to the admin website. This would decrease the amount of sensitive data that has to be sent through our own API while utilizing Microsoft's own security. Another feature that would improve our product is the implementation of automatic database backups. Though not a large feature, it would provide increased availability of their data in the case of an error on Microsoft's side. Finally, as mentioned earlier in Chapter 5, we have built the server and database with high modularity in mind. This will simplify expansion for our client if they wish to store additional data in our product.

10.3 Conclusion

During the duration of this thesis project we have learned a lot about working as a group, working with a client, and setting realistic expectations. We managed to build a product to our client's specifications. We cooperated, managed our time, and also completed this thesis. Although our product is finished, it is not perfect. As a high quality MVP it is great. However, it still needs more work before it can compete with the industry standard with regard to security and efficiency. Through our weekly meetings and unexpected delays, we have learned a lesson in time management. Overall, we consider this bachelor thesis to be a success. Our client has received a product that we are proud of, and we have gained invaluable experience.

Bibliography and External sources

[1] *Cryogenetics about us*, https://www.cryogenetics.com/, Accessed: 2023-05-4.

[2] New Oxford Dictionary, *Definition of "logistics"*, 2023.

[3] *Supply chain management terms and glossary*, https://cscmp.org/CSCMP/Educate/SCM_Definitions_and_Glossary_of_Terms.aspx, Accessed: May 20, 2023.

[4] J. Mangan, C. Lalwani, T. Butcher and R. Javadpour, *Logistics and Supply Chain Management*, 2nd. John Wiley & Sons Ltd, 2012.

[5] *Microsoft azure*, https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/, Accessed: May 5, 2023.

[6] *Azure sql database*, https://learn.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview?view=azuresql, Accessed: May 5, 2023.

[7] *Azure portal*, https://learn.microsoft.com/en-us/azure/azure-portal/azure-portal-overview, Accessed: May 5, 2023.

[8] *Golang*, https://go.dev/, Accessed: May 5, 2023.

[9] *Android studio*, https://developer.android.com/studio, Accessed: May 5, 2023.

[10] *Kotlin*, https://kotlinlang.org/, Accessed: May 5, 2023.

[11] *React website*, https://reactjs.org/, Accessed: May 5, 2023.

[12] A. Schumacher, W. Sihn and S. Erol, 'Automation, digitization and digitalization and their implications for manufacturing processes,' 2016, Theresianumgasse 27, 1040 Vienna, Austria.

[13] 'Cracking the monolith: Challenges in data transitioning to cloud native architectures,' 2018. DOI: 10.1145/3241403.3241440.

[14] *Cryogenetics logistical solution design prototypes*, https://www.figma.com/community/file/1242273867569143764, Accessed: May 20, 2023.

[15] T. H. Carr, 'Perceiving visual language,' in *Handbook of Perception and Human Performance: Vol. 2. Cognitive Processes and Performance*, K. R. Boff, L. Kaufman and J. P. Thomas, Eds., Accessed: May 17, 2023, New York: Wiley, 1986, pp. 29–92.

[16] L. Zhou, V. Bensal and D. Zhang, 'Color adaptation for improving mobile web accessibility,' in *2014 IEEE/ACIS 13th International Conference on Computer and Information Science (ICIS)*, 2014, pp. 291–296. DOI: 10.1109/ICIS.2014.6912149.

[17] B. Wong, 'Color blindness,' *nature methods*, vol. 8, no. 6, p. 441, 2011.

[18] R. M. Rider, 'Color psychology and graphic design applications,' *Senior Honors Theses*, no. 111, 2010, Accessed: May 10, 2023. [Online]. Available: https://digitalcommons.liberty.edu/honors/111/.

[19] N. Babich, '"f-shaped pattern for reading content",' *https://uxplanet.org/f-shaped-pattern-for-reading-content-80af79cd3394*, 2017, Accessed: May 10, 2023.

[20] T. Thesing, C. Feldmann and M. Burchardt, 'Agile versus waterfall project management: Decision model for selecting the appropriate approach to a project,' *Procedia Computer Science*, vol. 181, pp. 746–756, 2021, CENTERIS 2020 - International Conference on ENTERprise Information Systems / ProjMAN 2020 - International Conference on Project MANagement / HCist 2020 - International Conference on Health and Social Care Information Systems and Technologies 2020, CENTERIS/ProjMAN/HCist 2020, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2021.01.227. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921002702.

[21] N. Ozkan, S. Bal, T. Gurgen Erdogan and M. Gok, 'Scrum, kanban or a mix of both? a systematic literature review,' Aug. 2022. DOI: 10.15439/2022F143.

[22] *Github project*, https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects, Accessed: May 5, 2023.

[23] M. Paul, 'Official (isc)2 guide to the csslp cbk,' 2013.

Acronyms

API Application Programming Interface. 30, 34, 48, 49, 62, 67, 75, 93

CSS Cascading Style Sheet. 67

CSV Comma Separated Values. 19, 74

CVD Color Vision Deficiency. 41, 42

DOM Document Object Model. 13

E2E End-to-End. 79

GUI Graphical User Interface. 11, 14, 40, 64, 79, 90, 102

HTML HyperText Markup Language. 67

HTTP HyperText Transfer Protocol. 30, 31, 67, 70, 71, 73, 75, 80

IDE Integrated Development Environment. 12, 13, 99

JSON JavaScript Object Notation. 57, 59, 63, 64, 67, 68, 71, 72

M-UI Material User-Interface. xiv, 67, 68, 70, 75

MFA Multifactor Authentication. 86, 89

MVP Minimal Viable Product. 93

NTNU Norges Teknisk-Naturvitenskapelige Universitet. vi

NTNU Norwegian University of Science and Technology. iv, 4, 34

PNG Portable Network Graphics. 74

QR-Code Quick Response Code. 17, 30, 44, 66, 67, 73, 74

RAD Rapid Application Development. 51

REST Representational State Transfer. 67

RSA Rivest-Shamir-Adleman. 60

SPA Single-Page Application. 14, 48, 67

SQL Structured Query Language. 12, 31, 48, 57, 58, 61, 71, 73, 88, 89

UI User Interface. 3, 10, 13, 40

URL Uniform Resource Locator. 14, 43, 49, 55, 56, 63, 67, 75

WMS Warehouse Management System. 10

XML eXtensible Markup Language. 90

Glossary

accountability The principle that an individual is entrusted to safeguard and control equipment, keying material, and information and is answerable to proper authority for the loss or misuse of that equipment or information. 85

activities In Android, an activity is a single, focused thing that a user can do. It is a class that typically corresponds to a single screen with a user interface. 48

Agile A set of values and principles for software development that prioritize customer satisfaction, teamwork, and flexibility, with a focus on delivering working software early and often. 51, 92

Android An open-source operating system used for smartphones and tablet computers. 13, 29–31, 48, 65, 66, 84, 90

Android Studio The official Integrated Development Environment (IDE) for Android app development. 11, 13, 30–32

authentication Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system. 85, 86

Authorization In a security setting, authorization refers to the process of granting or denying access rights to resources or actions based on established policies and privileges.. 86

backend The "Backend" of our project refers to our server application as well as our database . 4, 14, 15, 30, 31, 34, 55, 57, 59, 67, 71, 79, 85, 88

confidentiality The security concept that has to do with protection against unauthorized information disclosure. 84

configuration The way something is arranged. 55

cross-platform Able to be used on different types of operating systems or with different software packages. 84

database A centralized location for the storage, retrieval, and modification of data. 29, 30, 55–58, 61, 72, 88, 89

debug The process of identifying and removing errors from software. 59, 85

decode The process of transforming encoded data back into its original form. 57, 59

decrypt The process of transforming encrypted data back into its original form. 57, 85

dependency A component required for something to work. 55

disassembler A program for converting machine code into a low-level symbolic language. 85

Discord A proprietary, all-in-one voice and text chat application. 6, 11, 51

encode The process of putting data into a specialized format for efficient transmission or storage. 57, 59

encrypt The process of putting data into a specialized format to prevent unauthorized access to it. 57, 59, 60, 85

endpoint A digital location in software where in- and outgoing requests are made to an external actor. 55, 56, 58

exception An abnormal or unprecedented event that occurs after the execution of a software program or piece of code. 57

field A set of values that have the same data type and are arranged in a table. 58, 61, 62

firebase A set of backend cloud computing services and application development platforms provided by Google hosting databases, services, authentication, and integration for a variety of applications. 89

folder A container for one or several files. 55

Foreign Key One or multiple fields in a database table that refers to the primary key of another table.. 49, 50, 61, 62

fragments In Android, a fragment is a modular section of an activity's UI that can be combined with other fragments to create a flexible and responsive UI. 48

frontend The "Frontend" of our project refers to our mobile- and web application. 37, 47, 55, 88

function A sequence of program instructions that performs a specific task. 55–58, 79–81

function header The piece of a function containing its name and what type of data it expects to receive and return. 58

GANTT Gantt chart, a bar chart used to illustrate a project schedule. 28, 52, 90, 91

Git A free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. 6, 13, 30, 51, 84

GitHub A web-based hosting service for version control using Git. 11, 27, 36, 51, 53

Github project A project is an adaptable spreadsheet that integrates with your issues and pull requests on GitHub to help you plan and track your work effectively. 51, 52, 54

Golang A statically typed, compiled programming language designed at Google. 4, 5, 11, 12, 15, 30, 55, 57, 58, 80, 88

Google docs Google docs is a cloud-based word processing software developed by Google that allows users to create, edit, and collaborate on documents in real-time. 6

hardcode (Adjective) Data or parameters being made in such a way that they cannot be altered without modifying the program. 85

Hypertext Transfer Protocol Secure (HTTPS) An extension of the Hypertext Transfer Protocol (HTTP), using encryption for secure communication over a computer network. 85

incremental test The process of iteratively making more complex tests than the previous ones. 80

integration test A software testing method by which individual software units are combined and tested as a group. 79

integrity The measure of software resiliency and it has to do with the alternation or modification of data and the reliable functioning of software. 85

Internationalization Internationalization is the process of designing and developing software products or applications that can be adapted to various languages, regions, and cultures without requiring changes to the source code. 35

iOS An operating system used for mobile devices manufactured by Apple Inc. 84

issue A feature of the GitHub platform that allows users to track and manage tasks, bugs, and feature requests for a project. 51, 54

Kanban A visual management method used to track and manage work, originally developed for manufacturing but now used in many fields. 27, 51

Key In the context of maps, a key is a unique identifier that is used to access a corresponding value. 64

Kotlin A cross-platform, statically typed, general-purpose programming language with type inference. 13, 30

LaTeX A document preparation system for high-quality typesetting, widely used in academia and the scientific community. 4, 6, 11

marshal The process of transforming data into a format suitable for storage or transmission. 56, 57, 59

Material-UI A component library for ReactJS for GUI development. 30

Meta A multinational technology based company, formerly Facebook Inc. 13

metadata Data providing information about data. 56

Microsoft Azure A cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers. 3, 11, 12, 30–32, 89–92

Microsoft Excel Microsoft Excel is a spreadsheet developed by Microsoft. It features calculation or computation capabilities, graphing tools, and pivot tables. 74

milt In fishing, milt refers to the seminal fluid or sperm of male fish. It is released during the spawning process. 1

Modal A user interface component or pattern that is used to display information, notifications, or additional content on top of the main application screen. 70–73, 89

mongoDB A source-available cross-platform document-oriented database program. 89

Node.js A cross-platform, open-source server environment which runs on the V8 JavaScript Engine, and executes JavaScript code outside a web browser. 88

normalize the process of organizing data into tables in such a way that the results of using the database are always unambiguous and as intended. 89

noSQL Databases that are non-tabular and store data differently than relational tables. 89

open design A security principle which states that the implementation details of the design should be independent of the design itself. 85

open source Software for which the oroginal source code is made freely available and may be redistributed and modified. 58

Overleaf Overleaf is an online collaborative LaTeX editor that allows users to create, edit, and share LaTeX documents. 6, 11

package A group of components licensed, downloaded or subscribed to as a bundle of related products. 55–58, 60, 79, 80

parse The process of transforming data from one format to another. 57

phpMyAdmin A free software tool written in PHP, intended to handle the administration of MySQL over the Web. 61

Props React properties (more commonly known as Props) are like function arguments used to pass attributes from a parent component to its child. 13, 70

public-key cryptography The field of cryptographic systems that use pairs of related keys. 59

pure function A function which does not rely on anything else than its declared inputs. 57, 58, 79, 80

React Native An open-source UI software framework used to develop cross-platform applications. 84

ReactJS A JavaScript library for building user interfaces, created by Facebook and maintained by Meta. 4, 11, 13, 14, 30, 48, 67, 102

repository In the context of Git and GitHub, a repository (or "repo") is a collection of files and version control information stored on a server. 27

risk The probability of exposure or loss resulting from a cyber attack or data breach on an organization. 59, 85, 86

root The top-level directory of a file system. 55

runtime The time when a program is running. 56

Scrum An agile framework for developing and sustaining complex products, with a focus on team collaboration and iterative progress. 51

Scrumban The Scrumban framework merges the structure and predictable routines of Scrum with Kanban's flexibility to make teams more agile, efficient, and productive. 6, 27, 51, 54

security by obscurity A security principle wherein the security of the software is dependent upon the obscuring of the design itself. 85

server A computer or computer program which manages access to a centralized resource or service in a network. 55

session The time frame between program- start and interruption. 56

source code Program instructions stored in a format that humans can read. 55, 85

SQL injection A form of attack in which a SQL query is inserted or "injected" via the input data of a function. 58

string A data type representing a sequence of characters. 57, 58, 85

table A database object containing organized data. 58, 61, 62, 72, 88, 89

test A method to check if a component acts as expected. 55, 57, 79–81

test A software testing method by which individual units of source code are tested separately. 79

threat The possibility of an unwanted, unintended or harmful event occurring. 85, 86

Toggl A time tracking tool that offers both web-based and desktop-based applications as well as mobile apps for iOS and Android. 11, 36

Value In the context of maps, a value is the data associated with a key. It can be accessed using the key in a key-value pair. 64

verification The process of evaluating software to determine whether the products of a given development phase satisfies the conditions imposed at the start of the phase. 85

Visual Studio Code A free source-code editor made by Microsoft for Windows, Linux and macOS. 11

vulnerability A weakness or flaw that could be accidentally triggered or intentionally exploited by an attacker, resulting in the breach or breakdown of the security policy. 84

waterfall A linear sequential approach where the project is divided into distinct phases, such as requirements gathering, design, implementation, testing, and maintenance. 51

XAMPP XAMPP is a free and open-source cross-platform web server solution stack, which includes Apache HTTP server, MariaDB database, and PHP scripting language . 30

Appendix A

Project agreement

NTNU

Vår dato Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

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

_____

_____ (oppdragsgiver), og

_____

_____

_____ (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 _____ til_____ .

   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/fax, reiser og nødvendig overnatting på steder langt fra NTNU på 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 bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

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): _____

Oppdragsgivers kontaktperson (navn): _____

Student(er) (signatur): _____ dato _____

_____ dato _____

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur):_____ dato _____

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 B

Project plan

Project plan Cryogenetics

Authors:
Lars Lahlum Ruud
Axel Elias Wollebekk Jacobsen
Matthias David Greeven
Håvard Bø

1 Background and Scope

1.1 Background

Cryogenetics AS, henceforth known as the client, is a Norwegian biotech company that provides services and products for cryopreservation of milt and the fertilization of fish. The company is based in Hamar, but maintains an international presence with labs in Chile, Canada, the United States and Scotland.
Steffen Wolla, in his role as Production Manager and Business Developer for the client, has requested the development of a logistical system to register the movement of their liquid nitrogen containers. The ambition of this new system is that employees of the company would have an easier time managing the containers.

1.2 Subject area

The main subject area for the project is logistics. Logistics deals with the movement of materials and products towards facilities, in order to sell or produce materials and services. Logistics are a part of the company's operational costs. As companies grow and expand, it gets increasingly more complex to acquire, store and transport resources.

The digitalization of logistics has made many logistics processes more precise and manageable. By using specialized tools, companies are able to handle larger amounts of transactions, directly increasing profits and efficiency. Features like tracking and estimated delivery times allow companies to acquire more precise information surrounding their products, improving the experience for employees and customers.

1.3 Delimitation

We’re going to focus on the logistics of liquid nitrogen containers, the contents of said containers will not be registered in our system. These containers are designed to keep its contents below -140°C for extended durations, which removes the biological decay from its content. To ensure this they have to be regularly refilled with liquid nitrogen. The frequency of refilling depends on the size and model of the container, where larger containers often require recurrent refilling. Therefore it is crucial to document when refilling happens, as temperature increases could lead to the biological content to contaminate or expire.

1.4 Task description

The task is to develop an application to track and register the movement and status of their liquid nitrogen tanks. The project requires user interfaces, a server and a database. Operators and administrators will have two different user interfaces, which will communicate with a common server and database.

The tablet application must be able to do the following:

- Provide authentication through a 3 digit code.
- Designed with the principles of user centered design to limit the need for training.
- Scan QR codes on containers to identify the container.
- Sort transactions based on date, location, employee, container type and customer.
- Show transaction logs for the scanned container.
- Scan multiple containers in a filling menu", and update the server with all newly filled containers.
- Display when specific tanks were last filled with nitrogen, and give an approximation of when they should be refilled, based on tank size and model.

The administration application is requested to be designed for desktop environment, and must do the following:

- Deliver a report of change logs in between two desired time points.
- Authenticate users through email, password and two factor authentication.
- Sort on transactions based on date, location, employee, container and customer.
- Provide detailed logs over the history of a specific tank, filtered by locations.
- Administrate client locations in the database.
- Administrate operators authentication codes.
- Administrate container models in the database.
- Register new customers and update existing ones.
- Retrieve a detailed change log of all desired containers.

2 Goals and frames

2.1 Project goals

These are the goals we want to achieve during this project.

2.1.1 Result goals

Our project should produce a logistics solution which can be used to keep track of liquid nitrogen containers location and internal processes used for cryopreservation. The logistics solution will be custom designed solution for the clients needs, it will include:

- A mobile tool for operators.
- A desktop tool for administrators.
- A server and database.
- Deployment on Microsoft Azure.

2.1.2 Effect goals

By the projects end, cryogenetics will have:

- Better control and traceability over their liquid nitrogen containers.
- Digitized and eased their workflow.
- Warnings for when containers require immediate or scheduled maintenance.

2.1.3 Learning goals

During the development of our bachelor thesis we will be exploring and learning about our clients subject area. The knowledge we will focus on learning include, but is not limited to:

- Learning about the practices of the logistics business.
- Designing an efficient and user friendly logistics system.
- Creating a product with sound fundations for use in a real work environment.
- Establishing a server-based database that is internationally available and secure for company wide use.
- Using User Centered Design principles and techniques to develop a design which is intuitive to use by staff of different ages.
- Working efficiently and professionally in a group together with a third-party client
- Improving our mobile programming skills by developing a professional application.
- Creating an efficient database using data modeling.
- Developing a web application with the use of ReactJS.
- Creating a modern and fast back-end using Golang.

2.2 Constraints

2.2.1 Time Constraints:

- The project's final delivery deadline is 22.05.2023 12:00 PM, at this point the product, final report and documentation needs to be complete.

2.2.2 Technological constraints

- User friendly solution, which can be used by employees with different prerequisites.
- Application for a mobile device.
- Register and monitor movement and maintenance of nitrogen containers.
- Use of QR- or bar-code for swiftly registration of movement and maintenance.
- A web application for administrative tasks.

3 Project organization

The group consists of four BPROG students, each with roughly the same academic experience. However, in the course PROG2052 - Integrasjonsprosjekt, we worked on two different groups,

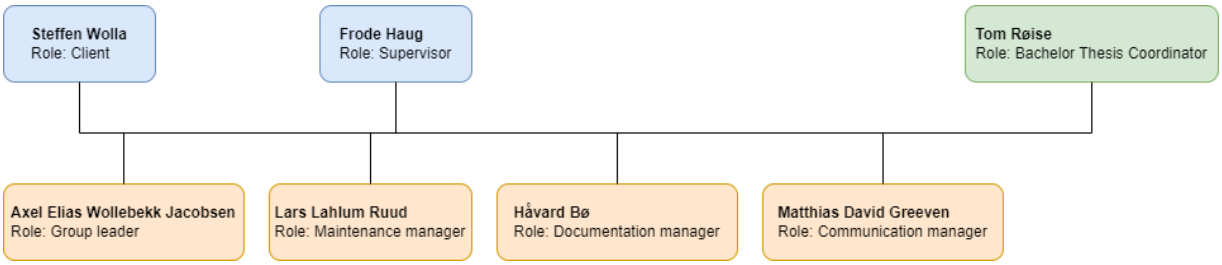
gaining expertise in different fields. Håvard and Matthias developed an android application in kotlin, while Lars and Axel made a GoLang backend.

3.1 Responsibilities and roles

Each group member has an area of responsibility based on their previous experience,

- Håvard is responsible for the Android application.
- Matthias is responsible for the website.
- Axel is responsible for the Golang backend.
- Lars is responsible for the SQL database and its deployment.

In addition each group member has a role, as shown in the image below.



3.2 Routines and group rules - set up a contract

Members: Axel Elias Wollebekk Jacobsen, Matthias David Greeven, Lars Lahlum Ruud, Håvard Bø

Procedures for the teamwork

A. *Meetings*

Meetings are held according to the Scrumban agile framework. Each Monday and Thursday 12:00 a Scrumban meeting is held physically or over Discord. The scrumban leader is responsible for incorporating proper scrumban technique, such as the use of the Github Project issueboard. Meetings with the client are held on mondays at 15:00 over Teams, if necessary. These are arranged at least 96 hours prior by the Communication manager and may be moved to a more suitable time if needed.

B. *Notification in case of absence or other incidents*

If you are late to- or cannot attend a meeting with the client, this should be communicated as soon as possible so that the meeting can be moved. If the meeting cannot be moved, it is held without you.

C. *Expected effort*

It is expected that you spend about 30 hours a week on this project, but as long as you complete your tasks and responsibilities it is not strictly enforced. In addition, you should be available from 12:00 to 15:00 each day a Kanban meeting has been held. If you cannot be available during this time, you should notify the group at least 12 hours prior.

D. *Disagreement*

If there is an academical disagreement the group should try to find a solution which the majority agrees on. If exactly half the group disagrees with the other, the Group leader decides. After the solution has been set, everyone must be loyal to it.

E. *Documents*

Discord and GIT are used to share files, with Google Docs as a collaborative writing tool. The Documentation Manager is responsible for meeting reports from meetings, these reports give at least a short description of which decisions have been made and what has been done during the meeting.

F. *Policy for monitoring tasks*

The Group Leader is responsible for monitoring tasks and ensuring that everyone has something to do. If you are struggling with completing your tasks, you should notify the group leader so that your tasks can be more evenly distributed.

G. *Submission of teamwork*

The Group Leader is responsible for ensuring that deadlines are kept and files are submitted.

H. Maintenance / Services

The Maintenance Manager is responsible for ensuring that the necessary services are available, such as cloud hosting services for backend. This is done in cooperation with the client, who has agreed to provide such services.

Breach of contract

If a group member does not follow the agreed upon procedures, a meeting can be held with every group member present about the breach of contract. If this does not solve the issue, a written warning is created by the other group members. The written warning should contain: An explanation of how the contract was breached, requirements for solving the issue, and what action will be taken if the group member does not fulfill the requirements. If the breach is severe enough, this action should be holding a meeting with the group and advisor about solving the issue. If none of this solves the issue, the group member who breached the contract will be expelled from the group.

4 Planning, follow-up & reporting

4.1 Main division of the project

For our bachelor thesis we decided to utilize the scrumban SDLC framework. Initially, our plan was to just use scrum. By having daily meetings we could ensure that everyone had work to do, and that they got it done. However, an issue we found with this was that the daily meetings ended up consuming a significant portion of extra time, even if the meeting itself only lasted 15-30 minutes. After consulting our advisor who suggested we tried an alternative approach, we decided to try using both kanban and scrum. Kanban would offer some flexibility in the beginning of the project where members can work on their parts when they have time, while scrum would ensure efficient progress later when we are mostly coding. After some discussion however, we realized that moving from one framework, to another mid-project could pose unwanted delays. Due to this we found that scrumban would suit our needs. By having two scrumban meetings a week with specific tasks distributed from a backlog we can ensure both progress and flexibility. Additionally, altering the length of sprints would permit us to optimize the framework to our needs as we progress in the project, while only utilizing one development framework.

4.2 Plan for status meetings and milestones during the project

During the initial design phase of the project we will have weekly meetings with our Cryogenetics representative, and NTNU Advisor, Frode Haug. This will help us ensure that the project's design models are correctly designed and up to the desired standards of both the school and our client. When the design gets accepted, the group will shift its focus from designing to development. Since the design is already decided on, we will reduce the amount of meetings with our representative and councilor. This will permit us to work faster since we can divide larger tasks between meetings. The meetings will also transition into a presentation format where we present our work so far, and receive feedback for improvement or alterations.

To improve progress tracking through the project, we have placed progress milestones. Setting dates for major progress checkpoints will help us change our focus from one work pattern to another. Since we do not have perfect foresight these dates will be subject to change as we progress in the project. However, we initially have chosen to place our milestones at these dates:

- 25.01 : Project plan
- 02.02 : Physical database set up
- 13.02 : First backend application draft
- 16.02 : First web application draft
- 20.02 : First status report
- 25.02 : First mobile application draft
- 01.03 : Main user testing round

- 13.03 : Web application mostly finished
- 17.03 : Mobile application mostly finished
- 22.03 : Backend completed
- 27.03 : Thesis outline completed
- 03.04 : Second status report
- 21.04 : Mobile & web application completed
- 28.04 : First rough draft of thesis
- 01.05 : Final status report
- 12.05 : Thesis mostly done
- 17.05 : Thesis completed

5 Organization of quality assurance

5.1 Documentation and standards

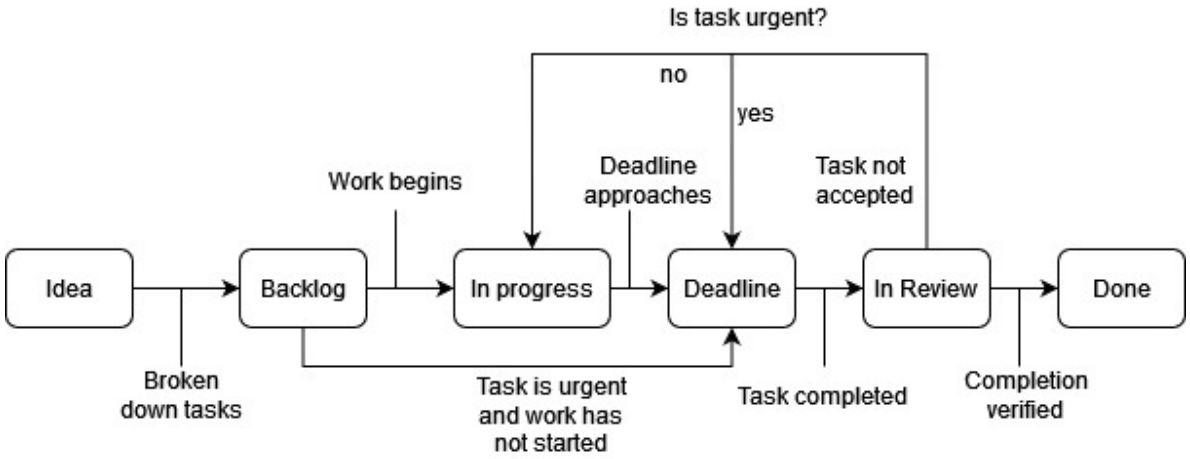
During the development process we will put an emphasis on client-friendly development practices. The intent behind this is to make it easier for our client to further develop our project after its completion without it demanding significant work with code and structural interpretation. For this reason we will be using the commenting and documentation practices each individual coding language uses respectively. Additionally we will include links to documentation practices in our GitHub repository to further simplify the process. To further ensure consistency we will be developing and documenting exclusively in english.

5.2 Standardized workflow

To simplify the workflow of our project, we have decided that all tasks will go through the same development steps. This ensures that all work is tracked properly, and follows a clear step-by-step path. A task begins its lifecycle as a feature desired by our client. We will then discuss specifications around how this feature operates before we break it down into smaller, more manageable tasks. These tasks are then added to the backlog of our GitHub-project issue board. Our issue board consists of five sections, “Backlog”, “In progress”, “Deadline [XX.XX.XXXX]”, “In review” and “Done”.

As mentioned, a new task is initially added to the backlog. When a group member starts work on a task they move it into “In progress”. Usually this is done during a scrumban meeting, but if someone decides they want to work more, then they can move it themselves to signify that it's being worked on to the rest of the group. At the same time as moving it, a task is assigned to whoever is working on it as well as being given a size and priority tag. More often than not we will move a task straight from the backlog into the “Deadline” category. The reason for this is that “Deadline [XX.XX.XXXX]” signifies when the next meeting, and delivery date is. We update the x’es to display the date of our next meeting. Thus a task in this category is expected to

be completed by that date. When a task has been completed it will be moved into “In review”. In our scrumban meetings we go through all tasks in “In review” and decide whether or not they have been completed up to the groups standards, if yes, the task is closed and moved to “Done”. However, if a task needs more work it will be moved back into either “In review” or “Deadline”, depending on the priority and size of the task.



Visualized progression of a feature’s life cycle

5.3 Tools

Name	Usage level	Reason
Github	High	We are familiar with using github for version control as well as utilizing trunk based development offers code insurance.
Golang	Medium	Golang is an efficient and fast language with great external libraries. Due to it being a young language with tremendous support it is highly unlikely that it will become deprecated any time soon.
Visual Studio Code	High	Our main development IDE will be VS code due to its simplicity and large library integration. This will be used to develop the backend and the web application.
Android studio	Medium	Android studio will be used for developing the mobile application aspect of our project. This is mostly due to previous experience with the IDE as well as its capabilities for quickly testing code with its built in android emulator.
Discord	High	For day to day communication and meetings within the group we will be using discord. We have chosen to work mostly online due to living far apart and most of us developing on a stationary computer. **NOTE:** We make sure to not share any sensitive client information via normal communication channels to ensure confidentiality of the data.
Microsoft Azure	Low	For hosting our backend and database we will be using Microsoft Azure in accordance with our clients wishes.
Toggl	High	To track time spent working we will be using Toggl. Toggl makes it easy to either track time spent live while working or insert worked hours afterwards.

5.4 Plan for Inspections and Testing

We've been granted permission to conduct testing with some staff members from Cryogenetics. This is a great help for our user tests, as the employees are our main target for users in the finished product (Adults ranging between the ages of 25-60). We will conduct these tests on both our low-fidelity mobile application model as well as our first mobile- and web application draft. With the data we receive from the tests we can add, remove or alter functionality to better serve our client's needs.

During development we'll utilize Postman to test our API. With postman we can send specific requests to a server endpoint with both SQL and JSON. Postman will then display the resulting response data and code in a structured fashion which will help identify bugs more efficiently. We will also write some simple tests that can be run to check that the endpoints are functioning as expected during development.

5.5 Risk analysis at project level (identify, analyze, measures, follow-up)

We have performed an overarching risk analysis for the entire project where we have identified how likely an event is, as well as the impact of each event.

1. **Overestimated development speed**
   As a group we overestimated our own skill and development speed, causing us to fall behind on the development plan.
2. **Late and sudden changes in desired product or client wishes**
   The client realizes late into development that there has been significant miscommunication leading us to develop a product that doesn't suit their needs.
3. **Loss of documentation or source code**
   Somehow all code or documentation is lost or becomes inaccessible for a longer period of time, causing a halt in development.
4. **Leaked customer data**
   An insufficient security protocol causes client data to be accessible to anyone.
5. **Loss of group members**
   A group member becomes inaccessible or decides that they do not wish to work together, causing them to leave the group.
6. **Conflicts within the group**
   A tear in the group causes work to slow down or grind to a halt.
7. **Lack of competence**
   Insufficient competence leading to an unfinished or inferior product.
8. **Server crash**
   Temporarily stored data gets erased or server access is lost.
9. **Local data is lost**
   Local progress is lost causing a setback to previous Git commit.

		Probability		
		Low	Medium	High
Impact	Low	5 & 6		9
	Medium	1 & 4	7	
	High	2 & 3	8	

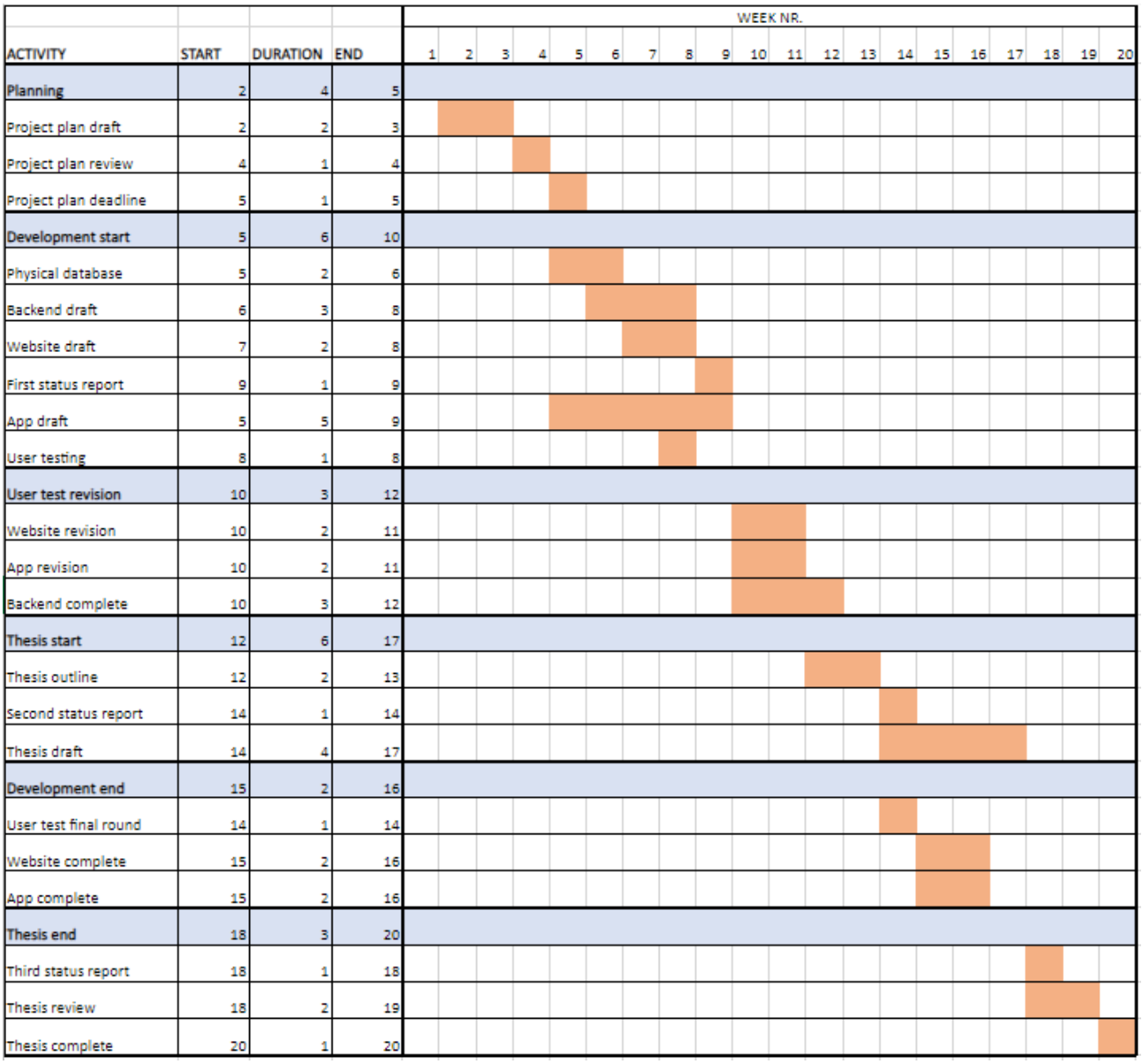
5.6 Plan for risk handling through preemptive and corrective measures

Nr	Preemptive	Corrective
1	By estimating progress based on our milestone estimates we can quickly see if we are falling behind.	Working longer hours or assigning more issues per sprint should help catch up to our desired schedule. Worst case, discuss with the client about decreasing the project's scope.
2	Through early user testing and prototyping we can clearly convey our intentions with our client to avoid miscommunication.	Initiate an emergency meeting with our client to correct our misinterpretations, and specify customer desires. Then salvaging what we can from the product and attempt to meet the client's wishes.
3	By storing backups of the source code locally on each developer's computer, we ensure that there is always an additional copy either locally or in Github's commit history.	Utilize a previous backup of the project locally stored on a group member's computer. Worst case, if all is lost contact supervisor and client in a joint meeting to discuss a solution.
4	By utilizing hashed passwords and individual ID's to access the sensitive endpoints we prevent unauthorized access. Additionally, perform regular tests to see if we can breach our own security.	First contact our client to alert them. Then, identify and fix the security flaw. Afterwards, host a meeting with our client to resolve the situation diplomatically.
5	By dividing work equally and ensuring that work is done cooperatively, there will always be someone who knows what the missing person was doing, and can	Communicate with the missing member to see if they will be gone permanently or temporarily. Worst case, reconfigure our work schedule and increase the workload of each individual member.

	thus fill their position.	
6	Regular and clear communication about issues ensures that there is no festering disagreements among group members that could escalate further.	In the case of a full blown conflict among group members, we will host a special meeting to discuss the problem and clear the air between the offending group members.
7	Tight knit cooperation and development with other group members will ensure that knowledge and expertise is shared in the case of a group member struggling with progressing their appointed task.	Lack of competence will hopefully be noticed during a weekly scrumban meeting when a member's work is not on par with expectations. The task will then be made cooperative so that knowledge can be shared. Worst case, a member will be given more time to research and acquire the needed expertise
8	Microsoft Azure permits saving temporary database backups on their server. Enabling this will grant us previous versions of the server available for rollback scenarios.	If the server crashes we should be able to restore the database using a backup file. Worst case, due to the clients wishes of not having a regular database backup system, we would have to rely on the client themselves having saved a backup.
9	Frequent and atomic sized git commits will ensure that even if local data is lost, the time since the last commit is minimal.	Restoring from the previous git commit, or previous git push should not cause a huge set back. Worst case, a group member has to start working from the beginning of their work session.

6 Gantt diagram & Implementation plan

ACTIVITY	START	DURATION	END
Planning	2	4	5
Project plan draft	2	2	3
Project plan review	4	1	4
Project plan deadline	5	1	5
Development start	5	6	10
Physical database	5	2	6
Backend draft	6	3	8
Website draft	7	2	8
First status report	9	1	9
App draft	5	5	9
User testing	8	1	8
User test revision	10	3	12
Website revision	10	2	11
App revision	10	2	11
Backend complete	10	3	12
Thesis start	12	6	17
Thesis outline	12	2	13
Second status report	14	1	14
Thesis draft	14	4	17
Development end	15	2	16
User test final round	14	1	14
Website complete	15	2	16
App complete	15	2	16
Thesis end	18	3	20
Third status report	18	1	18
Thesis review	18	2	19
Thesis complete	20	1	20

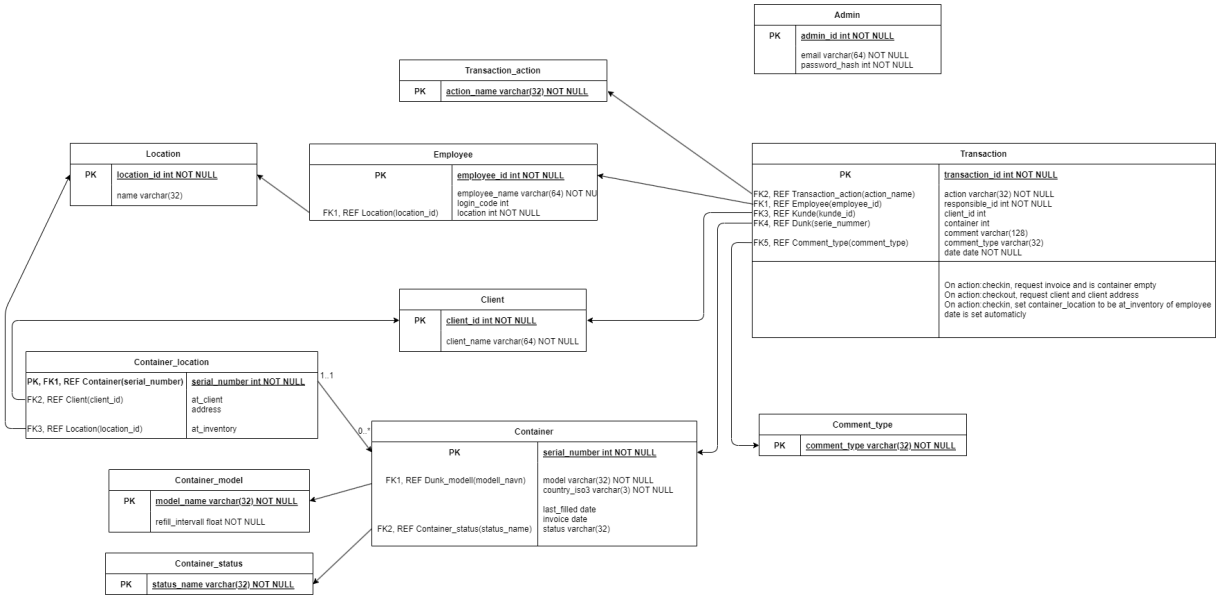


GANTT Chart. Blue, bold lines indicate the start of a new development phase.

i	Activity	Start Date	End Date
1	**Planning**	09.jan	31.jan
1.1	Project plan draft	09.jan	22.jan
1.2	Project plan review	23.jan	25.jan
1.3	Project plan deadline		31.jan
2	**Development start**	26.jan	28.feb
2.1	Physical database	26.jan	02.feb
2.2	Backend draft	02.feb	13.feb
2.3	Website draft	06.feb	16.feb
2.4	First status report	17.feb	20.feb
2.5	App draft	26.jan	25.feb
2.6	User testing	20.feb	20.feb
3	**User test revision**	01.mar	22.mar
3.1	Website revision	01.mar	13.mar
3.2	App revision	01.mar	13.mar
3.3	Backend complete	01.mar	22.mar
4	**Thesis start**	14.mar	12.mai
4.1	Thesis outline	14.mar	27.mar
4.2	Second status report	28.mar	03.apr
4.3	Thesis draft	28.mar	12.mai
5	**Development end**	10.apr	21.apr
5.1	User test final round	03.apr	09.apr
5.2	Website complete	10.apr	21.apr
5.3	App complete	10.apr	21.apr
6	**Thesis end**	22.apr	17.mai
6.1	Third status report	22.apr	01.mai
6.2	Thesis review	22.apr	12.mai
6.3	Thesis complete	13.mai	17.mai

GANTT Chart as text, with more detailed start- and end dates.

7 Conceptual model



Appendix C

Requirement specification

Requirements specification Cryogenetics

Table of contents:

3.1 Functional Requirements

The functional requirements entail what the project requires in order to work the way we intended it to. The following sections will go through what operations and functions the program is expected to perform.

Web Application

Admins **must** be registered by existing admin users. The admin user has higher authority when managing digital resources, entrusted with keeping the system running with provided tools. The admin features are as follows:

- Two factor authentication.
- Inventory
- Transaction logs
- Filter and search
- Generate monthly reports about the registered movement of the containers.
- Generate and print QR Code identifier for a liquid nitrogen tank.

Mobile Application

Users **must** have been registered through the admin webpage. Users have access to the following features:

- Sign into the application by using a personal 3 digit code.
- Receive the latest version of the transaction log for their workplace.
- See which containers require maintenance based on data from the transaction log.
- Register when a container has been refilled.
- Register a container into the system when received from the client.
- When a user interacts with a container, a log must be sent to the server for record keeping, these will appear in the transaction log.

Backend

The server **must** be able to handle the following:

- Multiple requests from all over the world and respond accordingly.
- Store incoming transactions in the SQL database.
- Create / Edit / Delete elements in the SQL database.

3.1.1 Use case Model

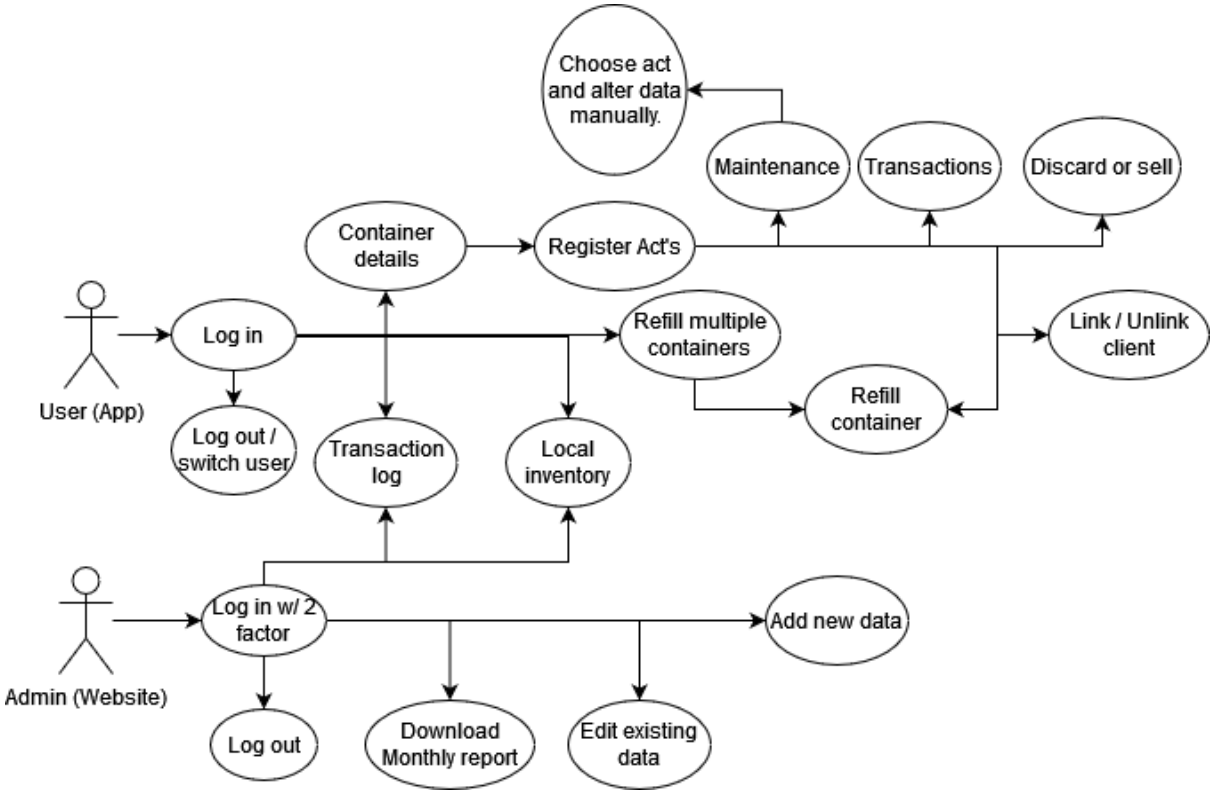


Fig 1: Use case model

3.1.2 High level Use case

In figure 1, our Use case model shows what the different actors are able to do in their respective programs, and when they are allowed to do so.

The following tables show what the most important user available features must be capable of doing. The following points on the use case model are not included due to redundancy: Log in, log out and alter data manually.

Use case	Transaction log
Actor	User (App) & Admin (Website)
Purpose	Retrieve an overview of the transactions regarding local labs
Description	A user should be able to view the recent transactions from / to their laboratory. The user can filter the selection of results to their specific needs.

Use case	Local inventory
Actor	User (App) & Admin (Website)

Purpose	Retrieve an overview of the containers currently in the local labs inventory.
Description	The user can see a list of all the containers they currently have in their storage, and an overview of that containers' current status.

Use case	Download Monthly report
Actor	Admin (Website)
Purpose	Produce a CSV file containing customer, start and end timestamps. location and container for the chosen month.
Description	The report must contain the data above, categorized by customer. The purpose of the report is to track where the customers' containers have been moved within the last month.

Use case	Log out / switch user
Actor	User (App)
Purpose	Remove/replace what user is responsible for transaction logging
Description	Users can log out / switch users when the current logged in user no longer holds responsibility for the Acts that are being recorded. Users can quickly switch between Logged in users by typing their authentication code.

Use case	Register Acts
Actor	User (App)
Purpose	Register a container Act to the transaction log
Description	Maintenance, Transactions, Discard or Sell, Link / Unlink client and Refill container are all marked as "Acts". This means that this operation on the app is to record an activity that the container has gone through.

Use case	Maintenance (Act)
Actor	User (App)
Purpose	Send in a maintenance update about the scanned container to the backend.

Description	Users have three options when assigning maintenance: ● This container needs maintenance. ○ The user specified what is required in the comment field. ● This container has completed its maintenance. ● Assign a custom act to this container. This is necessary in order to clear up human error by letting users “overwrite” their previous transaction. ○ Assign container model, status, act, location, address, date of last refill, invoice date and serial number. Assigning maintenance is tracked on the transaction log, and changes the status to “maint needed / maint compl”.

Use case	Transactions (Act)
Actor	User (App)
Purpose	Register where the container is being sent to / when it arrives at the users workplace.
Description	Users have three options when registering Transactions: ● Container is being sent out to customer ○ User has to input the address of the destination in order for our system to keep track. ● Container has been returned to us ● Container is being sent to an affiliate ○ Users must select which affiliate the container will be sent to. This movement is tracked on the transaction log.

Use case	Discard or sell (Act)
Actor	User (App)
Purpose	Remove the container from the database
Description	The user chooses a container to be sold/discarded, adding a comment with the relevant information. When a container is sold or discarded, it is no longer the property/interest of Cryogenetics, and is therefore no longer needed to track through the database. If the container is sold it no longer shows up on the laboratory's inventory.

Use case	Link / Unlink client (Act)
Actor	User (App)

Purpose	Change ownership of a container.
Description	If the container does not have a client registered to it, connect an existing client with the scanned / selected container. Else if the container already has a customer, remove that customer from the container's data. Linking a container to a client results in the customer taking ownership of the container. The change in ownership is saved on the transaction log of the container

Use case	Refill container (Act)
Actor	User (App)
Purpose	Change the *Last_Filled* date for this container to the current date
Description	Users select which container they physically refilled with Nitrogen, which will update the container Last_Filled date in order to keep track of which containers need to be refilled at what date.

Use case	Refill multiple containers
Actor	User (App)
Purpose	Scan a single / multiple container(s) to register them as refilled.
Description	Users can register that they have refilled one or multiple containers with nitrogen. This will set the last filled date to the current date. This is saved in the transaction logs of the affected containers

Use case	See Container details
Actor	User (App)
Purpose	Reveal the retrieved data about the scanned container.
Description	After scanning / selecting a container, the application must show the current status of the container from the server, what actions the user can register from here, and what transactions this container already has registered.

Use case	Add new data
Actor	Admin (Website)
Purpose	Input new data for the users to retrieve.
Description	Admins can add new data objects to the database by navigating to the

	desired data type they want to add. The following data can be added: ● Acts ● Affiliates ● Customers ● Containers ○ Container models ○ Container statuses ● Users / employees Each of these data types require their specific contents filled before they are sent to the backend.

Use case	Edit existing data
Actor	Admin (Website)
Purpose	Edit existing data in case of errors / changes.
Description	Admins can edit existing data objects by navigating to the desired data type they want to edit. The following data can be edited: ● Acts ● Affiliates ● Customers ● Containers ○ Container models ○ Container statuses ● Users / employees Each of these data types require their specific contents filled before they are sent to the backend.

3.1.3 Low Level Use Cases

Use case	See Container details
Actor	User (App)
Purpose	Reveal the retrieved data about the scanned container.
Description	After scanning / selecting a container, the application must show the current status of the container from the server, what actions the user can register from here, and what transactions this container already has registered.

Precondition 1	User must be authenticated on the application
Precondition 2	The container must be in a non-sold/discarded state.
Post-condition	The container data and history is shown, along with options for next acts.
Detailed course of action:	1. User locates the desired container. 2. User clicks the Camera icon to activate the QR scanning / Selects container from the inventory list. a. User scans the QR code located on the container. 3. Container data is fetched from the backend. 4. User sees the container data / history. 5. Users can Register Acts like Maintenance, Transactions, Refilling, Discard / Sell and Link / Unlink the container to customers.
Alternative scenarios	*Errors:* ● The API fails to load the container data ● Loss of internet connection

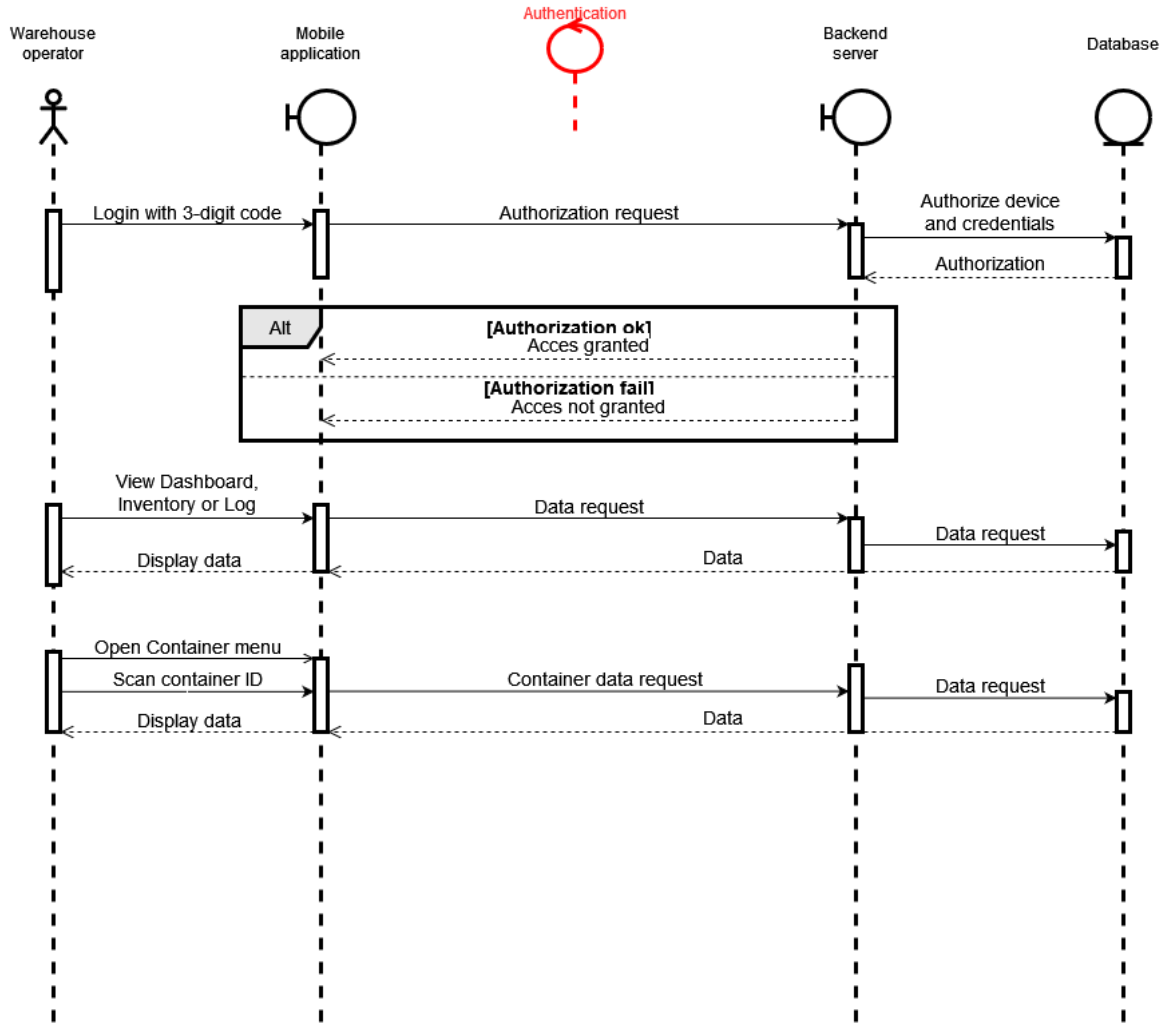
Use case	Add new data
Actor	Admin (Website)
Purpose	Input new data for the users to retrieve.
Description	Admins can add new data objects to the database by navigating to the desired data type they want to add. The following data can be added: ● Acts ● Affiliates ● Customers ● Containers ○ Container models ○ Container statuses ● Users / employees Each of these data types require their specific contents filled before they are sent to the backend.
Precondition 1	Admin must be authenticated on the website.
Precondition 2	The data must not exist beforehand.
Post-condition	The new data is sent to the backend, and updated on the Admin's page.
Detailed course of action:	1. Admin locates the desired type of data to add 2. A table of existing data of that type is retrieved from the backend.

	3. Existing data is revealed on that data's page 4. Admin clicks the "Add Data" button 5. A popup appears with the required fields of input for that data type. 6. Admin inputs the data in the required fields. 7. Admin presses the "Confirm" button. 8. New data is sent to the backend 9. New data is added to the table on the page.
Alternative scenarios	*Errors:* ● The added data already exists within the database ● Not all required fields are filled. ● Loss of internet connection.

Use case	Edit existing data
Actor	Admin (Website)
Purpose	Edit existing data in case of errors / changes.
Description	Admins can edit existing data objects by navigating to the desired data type they want to edit. The following data can be edited: ● Acts ● Affiliates ● Customers ● Containers ○ Container models ○ Container statuses ● Users / employees Each of these data types require their specific contents filled before they are sent to the backend.
Pre-condition	Admin must be authenticated on the website.
Post-condition	The new data is sent to the backend, and updated on the Admin's page.
Detailed course of action:	1. Admin locates the desired type of data to edit 2. A table of existing data of that type is retrieved from the backend. 3. Existing data is revealed on that data's page 4. Admin clicks the "Edit" button for the object they want to edit 5. A popup appears with the previous inputs that the admin can change as they please. 6. Admin edits the data. 7. Admin presses the "Confirm" button. 8. New data is sent to the backend. 9. New data is added to the table on the page.

Alternative scenarios	*Errors:* ● The added data already exists within the database (No duplicates) ● Not all required fields are filled. ● Loss of internet connection.

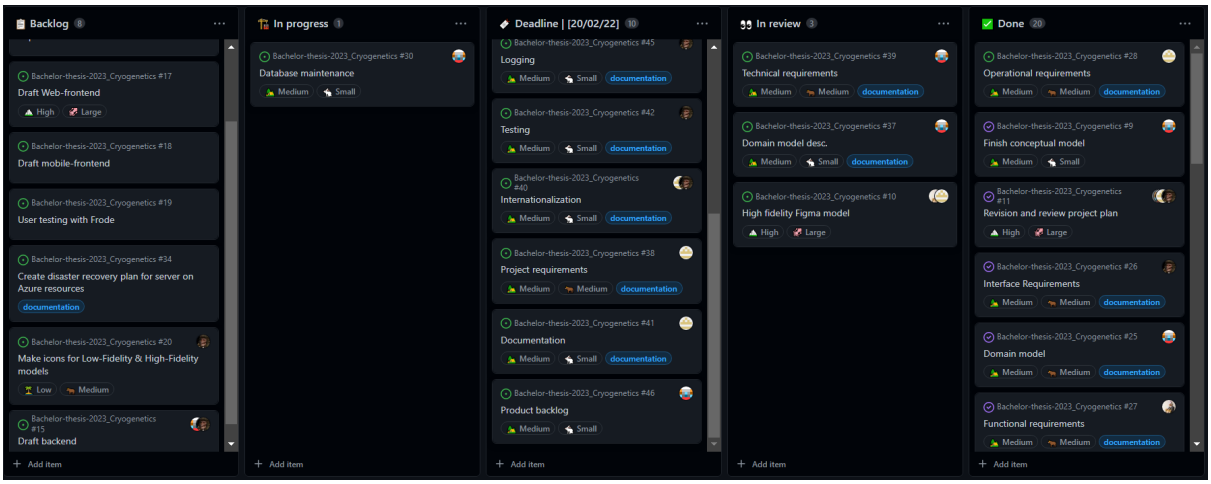
3.2 Sequence Diagram



3.3 Product backlog

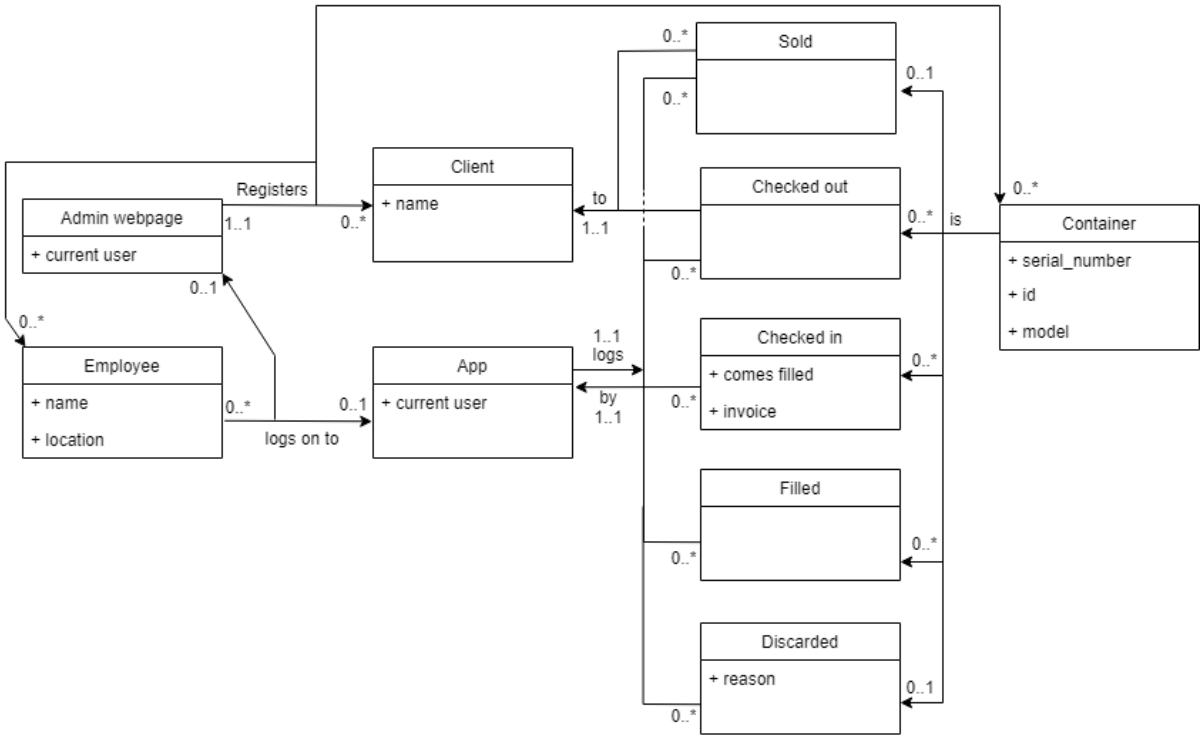
As we are using scrumban, a kanban board will be created to enhance workflow. This kanban board is going to be on Github. To do so, a new project with a backlog will be set up and attached to the project repo. Here, issues can be categorized and moved between columns, which each will represent a step in the workflow. The first column should be “Backlog”,

issues here have no set start or end date and are not currently being worked on. The second column will be “In progress”, which represents issues that are ongoing, but have no set end date. Then a column named “Deadline [DD/MM/YY]” will be added, in which we will change the name of each meeting to represent the next deadline. Issues in this column should be time-limited, and be completed before the set date. When an issue is completed, it will be moved to the “In review” column, which we will review at the end of each meeting. Then, the reviewed issues will either be closed or put back into the backlog if they were not complete.



New issues will be made during meetings when there are few issues left. These new issues should typically be motivated by the GANTT diagram, and be aligned with the current development stage. Issues are to be designated labels, size, priority, and milestone, in addition to responsible group members.

3.4 Domain model



The system mainly revolves around containers and how they are acted upon. A container has a lot of attributes, but only the ones that are used in transactions with other tables are shown in the domain model. These are “serial_number”, “id”, and “model”, which are used when selling, moving, filling or discarding a container. A container can be moved or filled as many times as needed, but only sold or discarded once, which is shown in the domain model as cardinality. Containers, clients and employees are registered through the admin webpage. An admin can register as many of these as needed, but only one admin is responsible for each registration. Employees may log on to either the admin webpage or the app, but not both at the same time. When an employee is using the app their actions are logged. Each container sale, check out, check in, filling and discardment has a responsible app-user, and containers are sold or checked out to a client who is registered in the database.

3.5 Operational Requirements

Mobile Application

The mobile application is intended to run on an android tablet with camera access.
The devices running the mobile application must:

- Have an Android operating system, version 10 - API 29 - Quince Tart or above.
- Have at least 128MB of free local storage.
- Connect to a Cryogenetics wifi network.
- A camera capable of reading QR-codes.

Web Application

The web application will require an internet connection and a common desktop browser. Users must login with a microsoft account and an authentication application on a mobile device for microsoft two factor authentication.

Server

Sending and storing data should be as efficient as possible to reduce the pressure on the servers. Successful retrieval of data from the server must be in reasonable time, which is at least within 3 seconds. The server must be deployed on Cryogenics' Microsoft Azure resources, alongside with a disaster recovery plan to correct problems that may occur.

3.5.2 Technical Requirements

Mobile Application

The app is to be developed for android devices using the Android Studio development environment. To utilize the latest features of Android Studio, the last stable release available to the devices should be used.. The app will be developed primarily with Kotlin, the official language for Android App Development. To save development resources the application will require a common tablet used in landscape orientation for an optimized experience. The device will also need a camera capable of scanning QR codes.

To render the web application for administrators a browser will be required, as well as an internet connection. A Microsoft account and a mobile device is required to use two-factor authentication. To print QR codes for tanks, a label printer with a 4 by 6 inch output label will be required. Users will be responsible for ensuring connection to a printer, as well as installing the necessary drivers and changing printer settings for successful printing. For quick and clean web development, React will be used with the component library Material UI. Communication between the website and the backend server will utilize the Fetch API - a native browser API that provides a low level interface for making HTTP requests. Fetch API is built into modern browsers, requiring no additional libraries or dependencies, in addition to providing support for asynchronous requests and responses through the use of Promises. Promises make it possible to register callbacks that are executed once the response is received, making it easier to write cleaner and more maintainable code.

The backend will be deployed fast and easily with Docker, which allows for cross-compatibility across windows, linux, mac, and more. The server might need to process up to 600 bytes of data per transaction, which for a typical processor (1,9GHz) takes about 40 nanoseconds, meaning processing power won't be an issue for any modern processor. Storing one million transactions in the database takes about 0,6GB of storage space, which in addition to project- and XAMPP files adds up to about 2GB of required storage space. GoLang will be the primary programming language used for the backend. It is fast and well integrated with existing technologies such as Docker and MySQL.

Github will be used for versioning, in addition to keeping track of issues, CI/CD, and milestones.

3.5.3 Interface Requirements

To ensure that our product operates nominally, we have set a few interface requirements in place. Since our product consists of two separate applications as well as a backend the requirements have been split into three groups, mobile, web and shared. Since we will be developing the mobile application for android devices, the mobile device has to support at least android version 10, Quince Tart. This is so that we can ensure the longevity of the product as an older device might cause deprecation problems. Additionally, we require the device to be a tablet, 8 inches or larger. To ensure the usability of our application we have selected colors that contrast each other, and make the app easy to use even for those with special requirements. In addition to the colors almost all buttons have text and icons which represent them.

For the admin web application our priority is ensuring solid contrasts. The website will have a simpler color palette. Thus, we have prioritized working with different levels of saturation to increase contrast. The website will be largely text based, this ensures efficient communication of information. Since the application is web based there are no specific device limitations. As for shared requirements, the largest one is access to the internet. Both the mobile- and web application are reliant on communication with a backend that interacts with the database. Thus, if they lose internet access, neither application will be able to perform any operations. Finally, the backend will be run on Microsoft Azure, in accordance to the clients wishes.

3.5.4 Testing

Mocking
From an early stage of development we will include mocking in our backend. Mocking the backend endpoints will allow us to perform tests without it influencing our database as well as. Additionally it will let us perform tests which give a standardized result regardless of the contents of the database. This way we can ensure that any failed tests are attributed to the application in development, and not the backend. Finally, through mocking we can see if a design choice is efficient and will grant the result we want before implementing

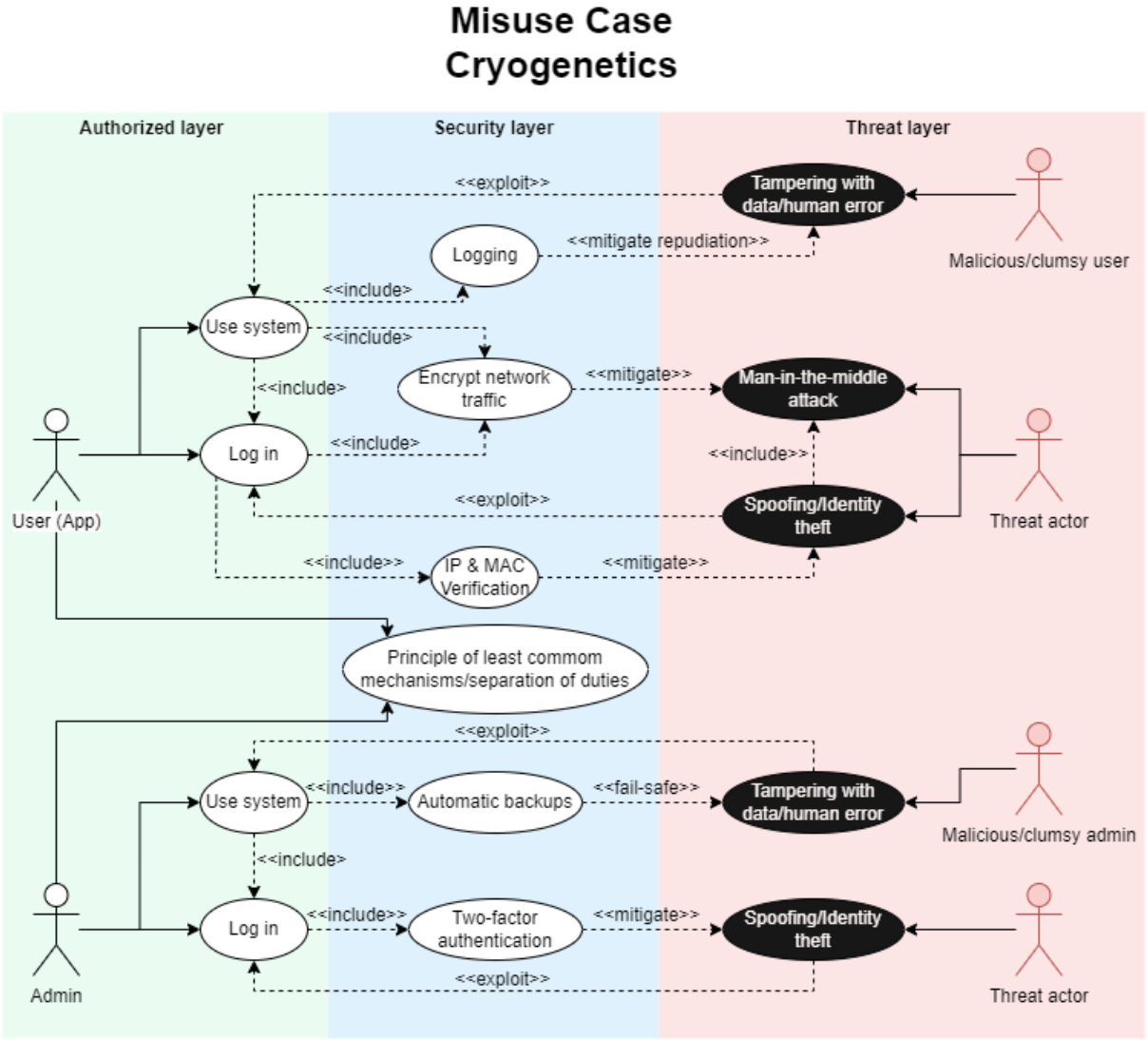
Testing during development
We will be using two main ways of testing our applications during development. Postman to test backend HTTP requests, and android studio’s built in emulator to test the mobile application. Postman lets us send HTTP requests to specific URLs with payloads. Using this we can test endpoints to see if they are returning the expected data in the correct format. Android studio's emulator lets us test how the mobile application will function on a “real” device, without having to export it as an application every time a new feature is implemented.

User testing
During the development of the whole project we will be performing user testing. Initially we will test the low- and high fidelity models with our client. The feedback we receive here will help us shape the application visually, in addition to adding or removing functionality from the applications. In the later stages of development we will test the application to receive more fine tuned feedback, while also keeping our client up-to-date on the progress of the project.

3.5.5 Security Requirements and Abuse Handling

Our primary security concern for our program is access. We wish to restrict access so that only authorized personnel can interact with the application. For the mobile frontend this involves limiting access to the application itself. The intention behind this is to stop threat actors from exploiting potential weak points in the application through brute force. Additionally, we plan to implement MAC and IP authorization to ensure no new devices can connect to the server without permission from an administrator. This implies that the web frontend will have a higher access level than the mobile application. Thus, we will implement two layers of protection, an email login and two-factor authorization to secure the website.

To tackle abuse of our system we have decided to simply utilize regular backups of the database. Since there are few things a user can do other than alter the database, it is our primary concern to ensure that all data is not lost. Thus the backups will be completely inaccessible from the program and will have to be manually accessed in the Azure cloud storage. Additionally we will implement an automatic and comprehensive logging system. With this we can see every change made to the database and the tanks in the system. This way, if a threat actor decides to alter data to sabotage our client, we will see where changes were made, and can therefore fix them quickly.



3.5.6 Authentication

To authenticate administrators in our web, Microsoft's two factor authentication will be used, which requires a Microsoft account and a mobile device. Super User and administrator can grant administrator access to other employees within the company.

The mobile application may only be used from a known network and device, since it uses simple 3-digit authentication for quick log in. Administrators can add networks and devices to the list of known networks and devices. When the connected network and the device is verified, the application can be used by employees associated with the location.

3.5.7 Encryption

To protect against man-in-the-middle attacks, sensitive network traffic should be encrypted. In our case we will mostly handle client- and employee information. Passwords are stored and handled by Microsoft's two factor authentication, so salt- and hashing will not be

necessary. Lastly, android automatically encrypts data located in the internal storage, so the devices used by the employees won't need any special encryption.

3.6.1 Project requirements

This project is required to be organized and well documented, since other developers will have the responsibility for maintaining and possible further development. It is also necessary for our application to be user friendly for people of all ages, without training the users to use the applications.

For the project to be successful for our client it must:

- Offer better tracking and organization of liquid nitrogen containers.
- Offer reports that can be used for invoicing customers.
- Offer functionality to register actions which affect the containers.
- Be user friendly and internationalized for use in multiple countries.

The project must be completed and delivered to both the client and NTNU before 22.05.2023 kl 12:00, the group must present the project on NTNU campus when desired, date could be 6-8. June 2023.

3.6.2 Documentation

Documentation is an important aspect of developing software, good planning and detailed documentation will often save time and resources in the future. Since this project will not be maintained by the same team developing it, the importance for good documentation is even greater. We will follow best practices for commenting code, which will apply to our backend API, website- and mobile frontend.

Frontend - Low- and High-Level Design

Web- and application-design will be documented with the final iteration of both our low- and high-level design. The final high-level designs will be the desired result of our development, and will be used as a guidance to implement intended functionality.

Frontend - Implementation Plan

Web- and application-design will be documented with the final iteration of both our low- and high-level design. The final high-level designs will be the desired result of our development, and will be used as a guidance to implement intended functionality.

Backend - Database

The database will be documented with a conceptual model and a logical model. These models will help show the thoughts behind the database implementation, and will help us design an optimized database with minimal oversight.

Backend - API

We will document the API by creating a plan for each endpoint, which will be helpful when developing and maintaining the API. We believe it will also increase our productivity during development, since we will have an overview over the required functionality.

Backend - Deployment

We will need a disaster recovery plan before production deployment, to account for unplanned incidents which can shut down our service. One reason to plan for these incidents is to keep recovery time and data loss minimal. Another is to implement preventive measures and correctly detect issues that need to be corrected before it greatly affects the availability.

User Testing Report

User testing will be documented with a report about the suggestions, results and the conversations we had during user testing. This is to gather all the information from the user testing session in one place, this will make it easier for us to make changes according to user feedback.

Meeting Notes

We will document all meetings by taking notes, and writing a short summary for each meeting. Keeping track of all our meetings will allow us to keep track of decisions and what we have already discussed. This will allow us to have more efficient meetings and therefore get more done with less resources.

3.6.3 Internationalization

We have decided to solve internationalization in two ways, by operating exclusively in english and utilizing symbols in the mobile application. We initially discussed with our client whether or not they wished for multiple languages to be available. However, they preferred to keep it simple and only use english. Since the mobile application will be used by most workers, we concluded that if internationalization could be relevant, it would be there. This is part of the reason why we have decided to use many small icons for important and significant buttons. The web application does, however, not include as many icons as the mobile app. Regardless, since it will only be used by administrative personnel, we concluded with our client that keeping English would be satisfactory.

3.6.4 User friendliness

To create user friendly interfaces for our applications we will use User Centered Design principles and user testing to create a custom-tailored, intuitive and convenient design. Since the mobile application will be used during warehouse related tasks, it needs to be convenient and practical to use. Since the web application is considered more of an additional tool for administrators, will we dedicate more resources to the design of the mobile application. The mobile application will feature various icons in combination with text and color, to help users navigate the user interface quickly and precisely. It will also feature common functionality which most users are already familiar with, for example changing the sorting in the log will be inspired by Microsoft's file explorer.

3.6.5 Versioning

For our project, we will be using GitHub for versioning. We will use branches to keep track of different versions of our code, and we use pull requests to merge changes from different branches. This allows us to develop more efficiently together by ensuring that everyone is working on the same version of the code. We also use GitHub's issue tracker to track bugs and feature requests. We have set up a GitHub project where we store our issues, and quickly move them between “backlog”, “in progress”, “next deadline”, “in review” and “done”. Additionally, we can assign size, priority and assignees as well as link the issues to specific commits or pull requests to provide context and traceability. Overall, GitHub provides us with a powerful tool for developing efficiently and transparently, in case Cryogenetics wishes to further develop our product in the future.

3.6.6 Logging

During our project we will log hours using Toggl. Toggl lets us begin a timer when we start working which helps us keep track of how long we have worked. When we are done we can see how much we have worked in a certain time period, as well as what we were working on through the use of tags attached to each work session. In addition to tracking time spent, we also have a designated note-taker who writes short summaries of our group-, client- and advisor meetings.

Appendix D

Gantt chart final

Requirements specification Cryogenetics

Table of contents:

3.1 Functional Requirements

The functional requirements entail what the project requires in order to work the way we intended it to. The following sections will go through what operations and functions the program is expected to perform.

Web Application

Admins **must** be registered by existing admin users. The admin user has higher authority when managing digital resources, entrusted with keeping the system running with provided tools. The admin features are as follows:

- Two factor authentication.
- Inventory
- Transaction logs
- Filter and search
- Generate monthly reports about the registered movement of the containers.
- Generate and print QR Code identifier for a liquid nitrogen tank.

Mobile Application

Users **must** have been registered through the admin webpage. Users have access to the following features:

- Sign into the application by using a personal 3 digit code.
- Receive the latest version of the transaction log for their workplace.
- See which containers require maintenance based on data from the transaction log.
- Register when a container has been refilled.
- Register a container into the system when received from the client.
- When a user interacts with a container, a log must be sent to the server for record keeping, these will appear in the transaction log.

Backend

The server **must** be able to handle the following:

- Multiple requests from all over the world and respond accordingly.
- Store incoming transactions in the SQL database.
- Create / Edit / Delete elements in the SQL database.

3.1.1 Use case Model

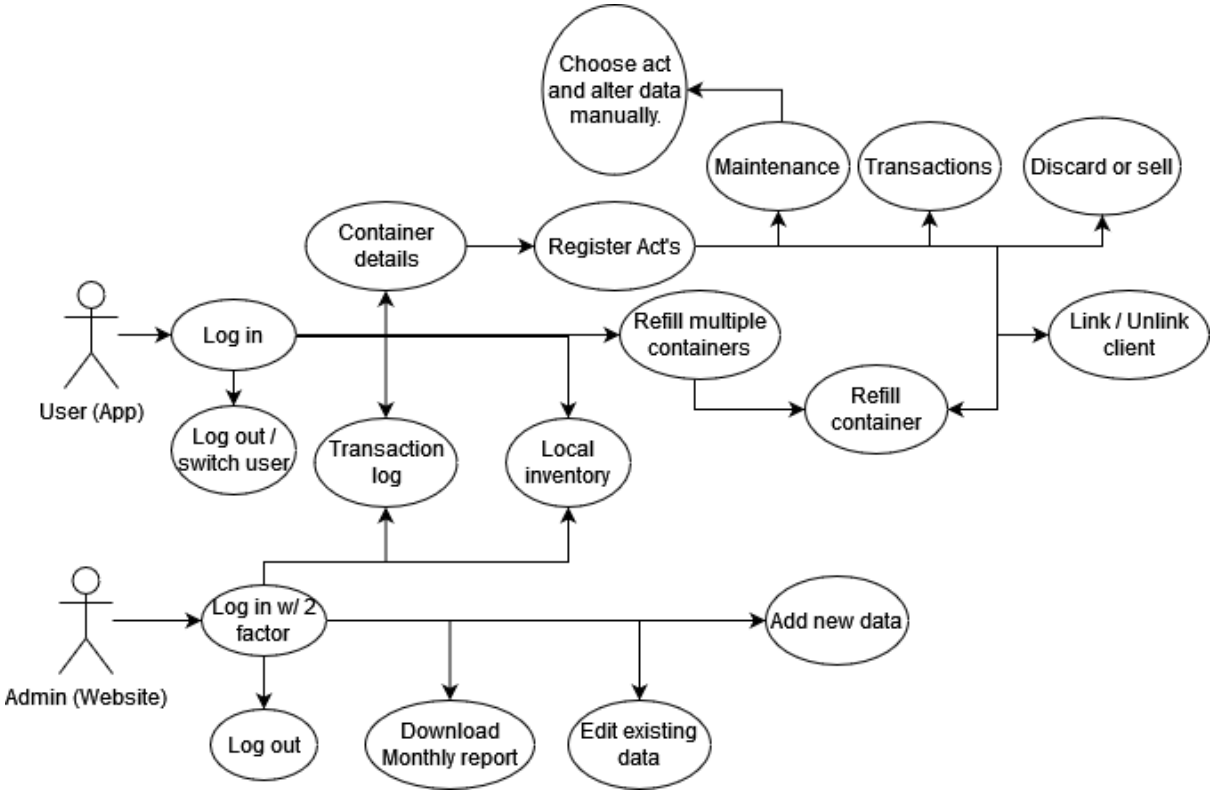


Fig 1: Use case model

3.1.2 High level Use case

In figure 1, our Use case model shows what the different actors are able to do in their respective programs, and when they are allowed to do so.

The following tables show what the most important user available features must be capable of doing. The following points on the use case model are not included due to redundancy: Log in, log out and alter data manually.

Use case	Transaction log
Actor	User (App) & Admin (Website)
Purpose	Retrieve an overview of the transactions regarding local labs
Description	A user should be able to view the recent transactions from / to their laboratory. The user can filter the selection of results to their specific needs.

Use case	Local inventory
Actor	User (App) & Admin (Website)

Purpose	Retrieve an overview of the containers currently in the local labs inventory.
Description	The user can see a list of all the containers they currently have in their storage, and an overview of that containers' current status.

Use case	Download Monthly report
Actor	Admin (Website)
Purpose	Produce a CSV file containing customer, start and end timestamps. location and container for the chosen month.
Description	The report must contain the data above, categorized by customer. The purpose of the report is to track where the customers' containers have been moved within the last month.

Use case	Log out / switch user
Actor	User (App)
Purpose	Remove/replace what user is responsible for transaction logging
Description	Users can log out / switch users when the current logged in user no longer holds responsibility for the Acts that are being recorded. Users can quickly switch between Logged in users by typing their authentication code.

Use case	Register Acts
Actor	User (App)
Purpose	Register a container Act to the transaction log
Description	Maintenance, Transactions, Discard or Sell, Link / Unlink client and Refill container are all marked as "Acts". This means that this operation on the app is to record an activity that the container has gone through.

Use case	Maintenance (Act)
Actor	User (App)
Purpose	Send in a maintenance update about the scanned container to the backend.

Description	Users have three options when assigning maintenance: ● This container needs maintenance. ○ The user specified what is required in the comment field. ● This container has completed its maintenance. ● Assign a custom act to this container. This is necessary in order to clear up human error by letting users “overwrite” their previous transaction. ○ Assign container model, status, act, location, address, date of last refill, invoice date and serial number. Assigning maintenance is tracked on the transaction log, and changes the status to “maint needed / maint compl”.

Use case	Transactions (Act)
Actor	User (App)
Purpose	Register where the container is being sent to / when it arrives at the users workplace.
Description	Users have three options when registering Transactions: ● Container is being sent out to customer ○ User has to input the address of the destination in order for our system to keep track. ● Container has been returned to us ● Container is being sent to an affiliate ○ Users must select which affiliate the container will be sent to. This movement is tracked on the transaction log.

Use case	Discard or sell (Act)
Actor	User (App)
Purpose	Remove the container from the database
Description	The user chooses a container to be sold/discarded, adding a comment with the relevant information. When a container is sold or discarded, it is no longer the property/interest of Cryogenetics, and is therefore no longer needed to track through the database. If the container is sold it no longer shows up on the laboratory's inventory.

Use case	Link / Unlink client (Act)
Actor	User (App)

Purpose	Change ownership of a container.
Description	If the container does not have a client registered to it, connect an existing client with the scanned / selected container. Else if the container already has a customer, remove that customer from the container's data. Linking a container to a client results in the customer taking ownership of the container. The change in ownership is saved on the transaction log of the container

Use case	Refill container (Act)
Actor	User (App)
Purpose	Change the *Last_Filled* date for this container to the current date
Description	Users select which container they physically refilled with Nitrogen, which will update the container Last_Filled date in order to keep track of which containers need to be refilled at what date.

Use case	Refill multiple containers
Actor	User (App)
Purpose	Scan a single / multiple container(s) to register them as refilled.
Description	Users can register that they have refilled one or multiple containers with nitrogen. This will set the last filled date to the current date. This is saved in the transaction logs of the affected containers

Use case	See Container details
Actor	User (App)
Purpose	Reveal the retrieved data about the scanned container.
Description	After scanning / selecting a container, the application must show the current status of the container from the server, what actions the user can register from here, and what transactions this container already has registered.

Use case	Add new data
Actor	Admin (Website)
Purpose	Input new data for the users to retrieve.
Description	Admins can add new data objects to the database by navigating to the

	desired data type they want to add. The following data can be added: ● Acts ● Affiliates ● Customers ● Containers ○ Container models ○ Container statuses ● Users / employees Each of these data types require their specific contents filled before they are sent to the backend.

Use case	Edit existing data
Actor	Admin (Website)
Purpose	Edit existing data in case of errors / changes.
Description	Admins can edit existing data objects by navigating to the desired data type they want to edit. The following data can be edited: ● Acts ● Affiliates ● Customers ● Containers ○ Container models ○ Container statuses ● Users / employees Each of these data types require their specific contents filled before they are sent to the backend.

3.1.3 Low Level Use Cases

Use case	See Container details
Actor	User (App)
Purpose	Reveal the retrieved data about the scanned container.
Description	After scanning / selecting a container, the application must show the current status of the container from the server, what actions the user can register from here, and what transactions this container already has registered.

Precondition 1	User must be authenticated on the application
Precondition 2	The container must be in a non-sold/discarded state.
Post-condition	The container data and history is shown, along with options for next acts.
Detailed course of action:	1. User locates the desired container. 2. User clicks the Camera icon to activate the QR scanning / Selects container from the inventory list. a. User scans the QR code located on the container. 3. Container data is fetched from the backend. 4. User sees the container data / history. 5. Users can Register Acts like Maintenance, Transactions, Refilling, Discard / Sell and Link / Unlink the container to customers.
Alternative scenarios	*Errors:* ● The API fails to load the container data ● Loss of internet connection

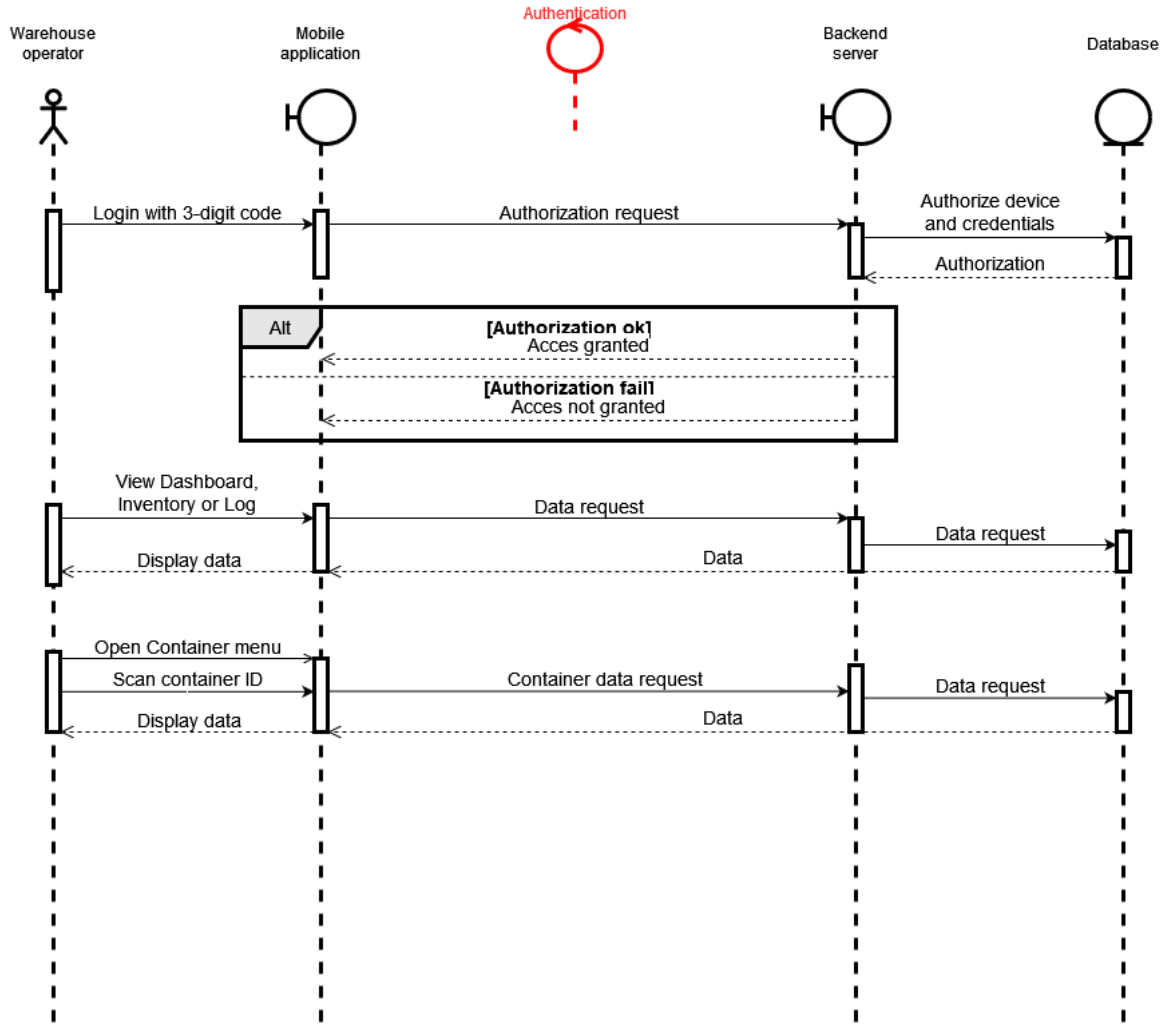
Use case	Add new data
Actor	Admin (Website)
Purpose	Input new data for the users to retrieve.
Description	Admins can add new data objects to the database by navigating to the desired data type they want to add. The following data can be added: ● Acts ● Affiliates ● Customers ● Containers ○ Container models ○ Container statuses ● Users / employees Each of these data types require their specific contents filled before they are sent to the backend.
Precondition 1	Admin must be authenticated on the website.
Precondition 2	The data must not exist beforehand.
Post-condition	The new data is sent to the backend, and updated on the Admin's page.
Detailed course of action:	1. Admin locates the desired type of data to add 2. A table of existing data of that type is retrieved from the backend.

	3. Existing data is revealed on that data's page 4. Admin clicks the "Add Data" button 5. A popup appears with the required fields of input for that data type. 6. Admin inputs the data in the required fields. 7. Admin presses the "Confirm" button. 8. New data is sent to the backend 9. New data is added to the table on the page.
Alternative scenarios	*Errors:* ● The added data already exists within the database ● Not all required fields are filled. ● Loss of internet connection.

Use case	Edit existing data
Actor	Admin (Website)
Purpose	Edit existing data in case of errors / changes.
Description	Admins can edit existing data objects by navigating to the desired data type they want to edit. The following data can be edited: ● Acts ● Affiliates ● Customers ● Containers ○ Container models ○ Container statuses ● Users / employees Each of these data types require their specific contents filled before they are sent to the backend.
Pre-condition	Admin must be authenticated on the website.
Post-condition	The new data is sent to the backend, and updated on the Admin's page.
Detailed course of action:	1. Admin locates the desired type of data to edit 2. A table of existing data of that type is retrieved from the backend. 3. Existing data is revealed on that data's page 4. Admin clicks the "Edit" button for the object they want to edit 5. A popup appears with the previous inputs that the admin can change as they please. 6. Admin edits the data. 7. Admin presses the "Confirm" button. 8. New data is sent to the backend. 9. New data is added to the table on the page.

Alternative scenarios	*Errors:* ● The added data already exists within the database (No duplicates) ● Not all required fields are filled. ● Loss of internet connection.

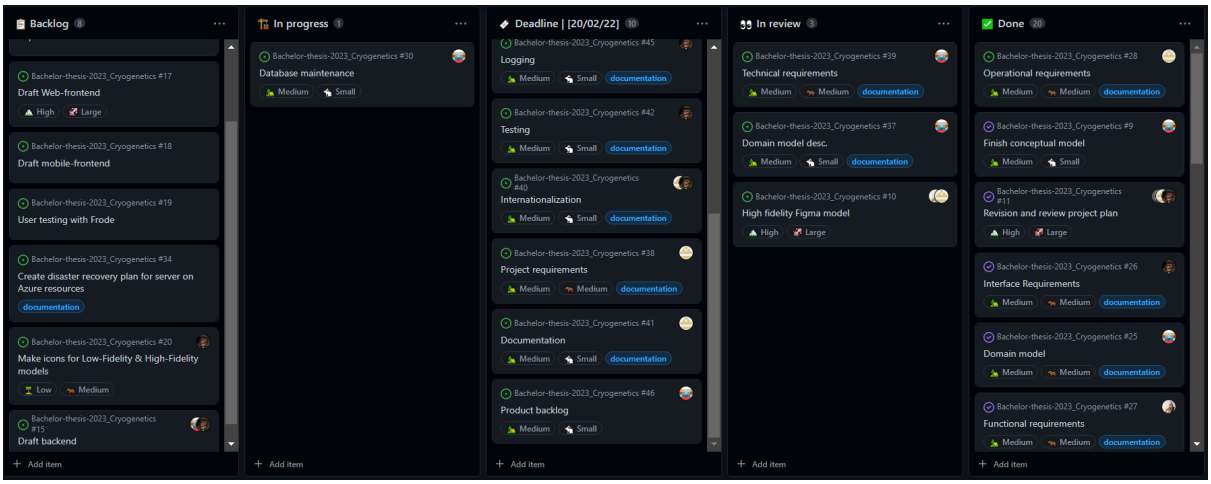
3.2 Sequence Diagram



3.3 Product backlog

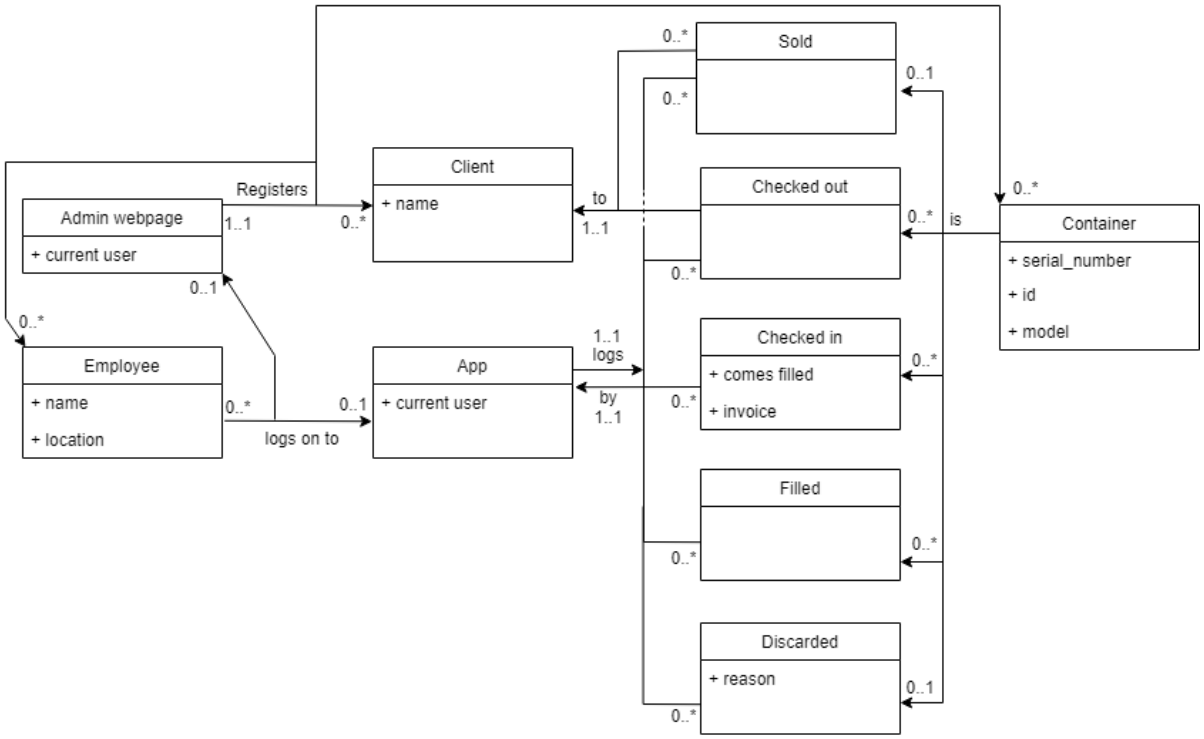
As we are using scrumban, a kanban board will be created to enhance workflow. This kanban board is going to be on Github. To do so, a new project with a backlog will be set up and attached to the project repo. Here, issues can be categorized and moved between columns, which each will represent a step in the workflow. The first column should be “Backlog”,

issues here have no set start or end date and are not currently being worked on. The second column will be “In progress”, which represents issues that are ongoing, but have no set end date. Then a column named “Deadline [DD/MM/YY]” will be added, in which we will change the name of each meeting to represent the next deadline. Issues in this column should be time-limited, and be completed before the set date. When an issue is completed, it will be moved to the “In review” column, which we will review at the end of each meeting. Then, the reviewed issues will either be closed or put back into the backlog if they were not complete.



New issues will be made during meetings when there are few issues left. These new issues should typically be motivated by the GANTT diagram, and be aligned with the current development stage. Issues are to be designated labels, size, priority, and milestone, in addition to responsible group members.

3.4 Domain model



The system mainly revolves around containers and how they are acted upon. A container has a lot of attributes, but only the ones that are used in transactions with other tables are shown in the domain model. These are “serial_number”, “id”, and “model”, which are used when selling, moving, filling or discarding a container. A container can be moved or filled as many times as needed, but only sold or discarded once, which is shown in the domain model as cardinality. Containers, clients and employees are registered through the admin webpage. An admin can register as many of these as needed, but only one admin is responsible for each registration. Employees may log on to either the admin webpage or the app, but not both at the same time. When an employee is using the app their actions are logged. Each container sale, check out, check in, filling and discardment has a responsible app-user, and containers are sold or checked out to a client who is registered in the database.

3.5 Operational Requirements

Mobile Application

The mobile application is intended to run on an android tablet with camera access.
The devices running the mobile application must:

- Have an Android operating system, version 10 - API 29 - Quince Tart or above.
- Have at least 128MB of free local storage.
- Connect to a Cryogenetics wifi network.
- A camera capable of reading QR-codes.

Web Application

The web application will require an internet connection and a common desktop browser. Users must login with a microsoft account and an authentication application on a mobile device for microsoft two factor authentication.

Server

Sending and storing data should be as efficient as possible to reduce the pressure on the servers. Successful retrieval of data from the server must be in reasonable time, which is at least within 3 seconds. The server must be deployed on Cryogenics' Microsoft Azure resources, alongside with a disaster recovery plan to correct problems that may occur.

3.5.2 Technical Requirements

Mobile Application

The app is to be developed for android devices using the Android Studio development environment. To utilize the latest features of Android Studio, the last stable release available to the devices should be used.. The app will be developed primarily with Kotlin, the official language for Android App Development. To save development resources the application will require a common tablet used in landscape orientation for an optimized experience. The device will also need a camera capable of scanning QR codes.

To render the web application for administrators a browser will be required, as well as an internet connection. A Microsoft account and a mobile device is required to use two-factor authentication. To print QR codes for tanks, a label printer with a 4 by 6 inch output label will be required. Users will be responsible for ensuring connection to a printer, as well as installing the necessary drivers and changing printer settings for successful printing. For quick and clean web development, React will be used with the component library Material UI. Communication between the website and the backend server will utilize the Fetch API - a native browser API that provides a low level interface for making HTTP requests. Fetch API is built into modern browsers, requiring no additional libraries or dependencies, in addition to providing support for asynchronous requests and responses through the use of Promises. Promises make it possible to register callbacks that are executed once the response is received, making it easier to write cleaner and more maintainable code.

The backend will be deployed fast and easily with Docker, which allows for cross-compatibility across windows, linux, mac, and more. The server might need to process up to 600 bytes of data per transaction, which for a typical processor (1,9GHz) takes about 40 nanoseconds, meaning processing power won't be an issue for any modern processor. Storing one million transactions in the database takes about 0,6GB of storage space, which in addition to project- and XAMPP files adds up to about 2GB of required storage space. GoLang will be the primary programming language used for the backend. It is fast and well integrated with existing technologies such as Docker and MySQL.

Github will be used for versioning, in addition to keeping track of issues, CI/CD, and milestones.

3.5.3 Interface Requirements

To ensure that our product operates nominally, we have set a few interface requirements in place. Since our product consists of two separate applications as well as a backend the requirements have been split into three groups, mobile, web and shared. Since we will be developing the mobile application for android devices, the mobile device has to support at least android version 10, Quince Tart. This is so that we can ensure the longevity of the product as an older device might cause deprecation problems. Additionally, we require the device to be a tablet, 8 inches or larger. To ensure the usability of our application we have selected colors that contrast each other, and make the app easy to use even for those with special requirements. In addition to the colors almost all buttons have text and icons which represent them.

For the admin web application our priority is ensuring solid contrasts. The website will have a simpler color palette. Thus, we have prioritized working with different levels of saturation to increase contrast. The website will be largely text based, this ensures efficient communication of information. Since the application is web based there are no specific device limitations. As for shared requirements, the largest one is access to the internet. Both the mobile- and web application are reliant on communication with a backend that interacts with the database. Thus, if they lose internet access, neither application will be able to perform any operations. Finally, the backend will be run on Microsoft Azure, in accordance to the clients wishes.

3.5.4 Testing

Mocking
From an early stage of development we will include mocking in our backend. Mocking the backend endpoints will allow us to perform tests without it influencing our database as well as. Additionally it will let us perform tests which give a standardized result regardless of the contents of the database. This way we can ensure that any failed tests are attributed to the application in development, and not the backend. Finally, through mocking we can see if a design choice is efficient and will grant the result we want before implementing

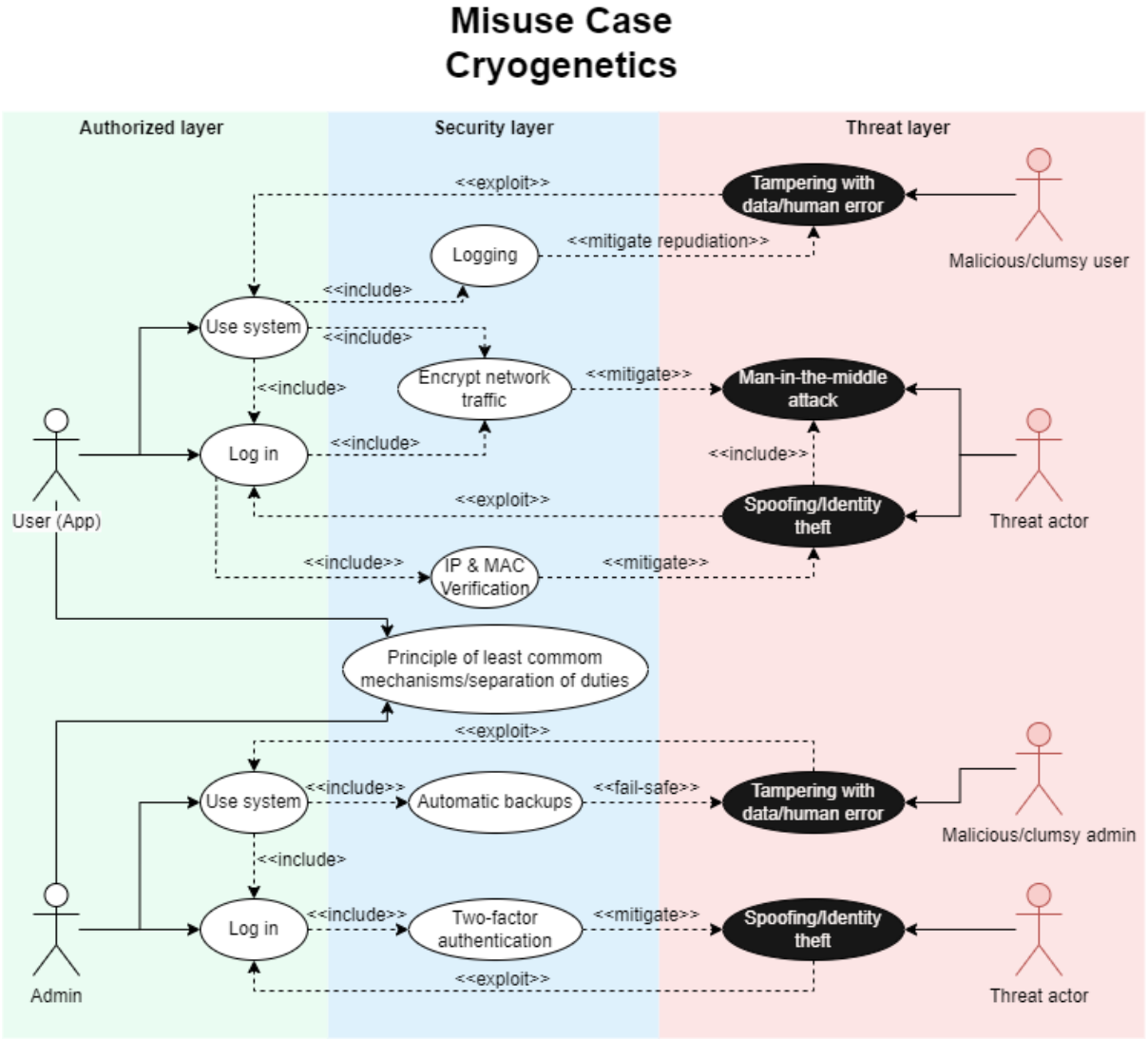
Testing during development
We will be using two main ways of testing our applications during development. Postman to test backend HTTP requests, and android studio’s built in emulator to test the mobile application. Postman lets us send HTTP requests to specific URLs with payloads. Using this we can test endpoints to see if they are returning the expected data in the correct format. Android studio's emulator lets us test how the mobile application will function on a “real” device, without having to export it as an application every time a new feature is implemented.

User testing
During the development of the whole project we will be performing user testing. Initially we will test the low- and high fidelity models with our client. The feedback we receive here will help us shape the application visually, in addition to adding or removing functionality from the applications. In the later stages of development we will test the application to receive more fine tuned feedback, while also keeping our client up-to-date on the progress of the project.

3.5.5 Security Requirements and Abuse Handling

Our primary security concern for our program is access. We wish to restrict access so that only authorized personnel can interact with the application. For the mobile frontend this involves limiting access to the application itself. The intention behind this is to stop threat actors from exploiting potential weak points in the application through brute force. Additionally, we plan to implement MAC and IP authorization to ensure no new devices can connect to the server without permission from an administrator. This implies that the web frontend will have a higher access level than the mobile application. Thus, we will implement two layers of protection, an email login and two-factor authorization to secure the website.

To tackle abuse of our system we have decided to simply utilize regular backups of the database. Since there are few things a user can do other than alter the database, it is our primary concern to ensure that all data is not lost. Thus the backups will be completely inaccessible from the program and will have to be manually accessed in the Azure cloud storage. Additionally we will implement an automatic and comprehensive logging system. With this we can see every change made to the database and the tanks in the system. This way, if a threat actor decides to alter data to sabotage our client, we will see where changes were made, and can therefore fix them quickly.



3.5.6 Authentication

To authenticate administrators in our web, Microsoft's two factor authentication will be used, which requires a Microsoft account and a mobile device. Super User and administrator can grant administrator access to other employees within the company.

The mobile application may only be used from a known network and device, since it uses simple 3-digit authentication for quick log in. Administrators can add networks and devices to the list of known networks and devices. When the connected network and the device is verified, the application can be used by employees associated with the location.

3.5.7 Encryption

To protect against man-in-the-middle attacks, sensitive network traffic should be encrypted. In our case we will mostly handle client- and employee information. Passwords are stored and handled by Microsoft's two factor authentication, so salt- and hashing will not be

necessary. Lastly, android automatically encrypts data located in the internal storage, so the devices used by the employees won't need any special encryption.

3.6.1 Project requirements

This project is required to be organized and well documented, since other developers will have the responsibility for maintaining and possible further development. It is also necessary for our application to be user friendly for people of all ages, without training the users to use the applications.

For the project to be successful for our client it must:

- Offer better tracking and organization of liquid nitrogen containers.
- Offer reports that can be used for invoicing customers.
- Offer functionality to register actions which affect the containers.
- Be user friendly and internationalized for use in multiple countries.

The project must be completed and delivered to both the client and NTNU before 22.05.2023 kl 12:00, the group must present the project on NTNU campus when desired, date could be 6-8. June 2023.

3.6.2 Documentation

Documentation is an important aspect of developing software, good planning and detailed documentation will often save time and resources in the future. Since this project will not be maintained by the same team developing it, the importance for good documentation is even greater. We will follow best practices for commenting code, which will apply to our backend API, website- and mobile frontend.

Frontend - Low- and High-Level Design

Web- and application-design will be documented with the final iteration of both our low- and high-level design. The final high-level designs will be the desired result of our development, and will be used as a guidance to implement intended functionality.

Frontend - Implementation Plan

Web- and application-design will be documented with the final iteration of both our low- and high-level design. The final high-level designs will be the desired result of our development, and will be used as a guidance to implement intended functionality.

Backend - Database

The database will be documented with a conceptual model and a logical model. These models will help show the thoughts behind the database implementation, and will help us design an optimized database with minimal oversight.

Backend - API

We will document the API by creating a plan for each endpoint, which will be helpful when developing and maintaining the API. We believe it will also increase our productivity during development, since we will have an overview over the required functionality.

Backend - Deployment

We will need a disaster recovery plan before production deployment, to account for unplanned incidents which can shut down our service. One reason to plan for these incidents is to keep recovery time and data loss minimal. Another is to implement preventive measures and correctly detect issues that need to be corrected before it greatly affects the availability.

User Testing Report

User testing will be documented with a report about the suggestions, results and the conversations we had during user testing. This is to gather all the information from the user testing session in one place, this will make it easier for us to make changes according to user feedback.

Meeting Notes

We will document all meetings by taking notes, and writing a short summary for each meeting. Keeping track of all our meetings will allow us to keep track of decisions and what we have already discussed. This will allow us to have more efficient meetings and therefore get more done with less resources.

3.6.3 Internationalization

We have decided to solve internationalization in two ways, by operating exclusively in english and utilizing symbols in the mobile application. We initially discussed with our client whether or not they wished for multiple languages to be available. However, they preferred to keep it simple and only use english. Since the mobile application will be used by most workers, we concluded that if internationalization could be relevant, it would be there. This is part of the reason why we have decided to use many small icons for important and significant buttons. The web application does, however, not include as many icons as the mobile app. Regardless, since it will only be used by administrative personnel, we concluded with our client that keeping English would be satisfactory.

3.6.4 User friendliness

To create user friendly interfaces for our applications we will use User Centered Design principles and user testing to create a custom-tailored, intuitive and convenient design. Since the mobile application will be used during warehouse related tasks, it needs to be convenient and practical to use. Since the web application is considered more of an additional tool for administrators, will we dedicate more resources to the design of the mobile application. The mobile application will feature various icons in combination with text and color, to help users navigate the user interface quickly and precisely. It will also feature common functionality which most users are already familiar with, for example changing the sorting in the log will be inspired by Microsoft's file explorer.

3.6.5 Versioning

For our project, we will be using GitHub for versioning. We will use branches to keep track of different versions of our code, and we use pull requests to merge changes from different branches. This allows us to develop more efficiently together by ensuring that everyone is working on the same version of the code. We also use GitHub's issue tracker to track bugs and feature requests. We have set up a GitHub project where we store our issues, and quickly move them between “backlog”, “in progress”, “next deadline”, “in review” and “done”. Additionally, we can assign size, priority and assignees as well as link the issues to specific commits or pull requests to provide context and traceability. Overall, GitHub provides us with a powerful tool for developing efficiently and transparently, in case Cryogenetics wishes to further develop our product in the future.

3.6.6 Logging

During our project we will log hours using Toggl. Toggl lets us begin a timer when we start working which helps us keep track of how long we have worked. When we are done we can see how much we have worked in a certain time period, as well as what we were working on through the use of tags attached to each work session. In addition to tracking time spent, we also have a designated note-taker who writes short summaries of our group-, client- and advisor meetings.

Appendix E

Status report 1

28/2/23

Cryogenetics status report 1

This is the first status report for our bachelor thesis project. In this report we will cover our progress as of february 28th.

Project plan

Before we started work on the product itself, we set up a comprehensive project plan which covered our plans for the completion of the project itself. We detailed rules for the group as well as important milestones in the form of a Gannt timeline. This was to ensure that we could keep a steady workflow while also knowing what to do if a group member either failed to accomplish what they were supposed to, or if an unexpected obstacle occurred. Additionally, we outlined what technologies we planned on using for the project, and what we hoped to gain by completing this project from an educational standpoint.

Knowing what technologies we plan to utilize will grant our client more insight into the workings of our product, as well as giving them an overview of potential expenses for when they ultimately receive the product. Finally, setting educational goals will help us stay on track to accomplish our goal using the methods we set out to learn.

Requirement specification

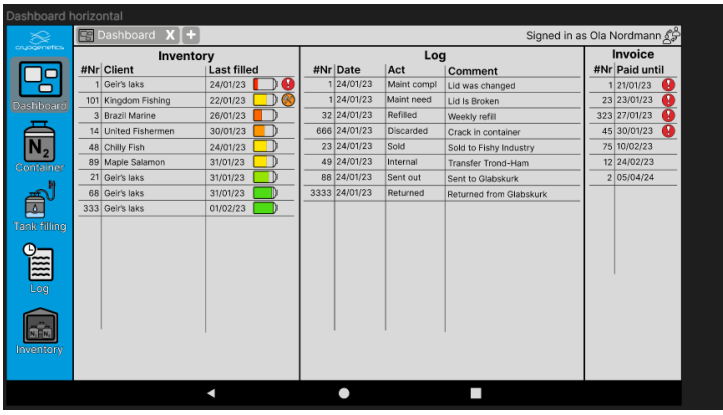
Since requirement specifications are an integral part of developing a successful product, we decided to do this early in development. In the document we detailed our use and misuse cases, operational- and functional requirements, as well as how we intended to test our programs. In addition to this we covered our work methods, such as issue board utilization, documentation, time tracking and versioning. Our use cases are split into two categories, high- and low level use cases. In the high level use cases we cover some of the overlying functions that a user can operate, these include critical actions such as retrieving an overview and downloading a monthly report. In the low level use cases we bring up examples of more complicated and niche actions, an example of this would be a detailed description of what happens when the user tries to reveal the retrieved data about a scanned container.

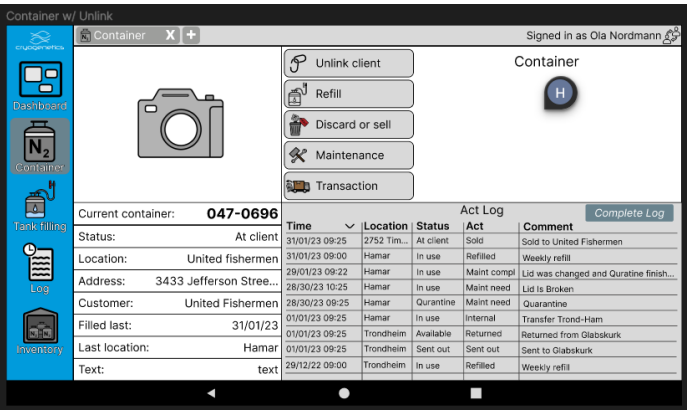
When it comes to our work methods, they have proven to be quite effective in ensuring proper and in-depth documentation. Github’s project function has worked seamlessly, partially due to our experience with using it in past projects. We have utilized toggl time track to track our time spent, and it has worked as intended. By giving us a clear overview of the time spent in any given time period, we can easily tell if we need to work more or if we are on schedule. Finally, Github has functioned as it should so far, and by using trunk-based development we have avoided any merge error so far.

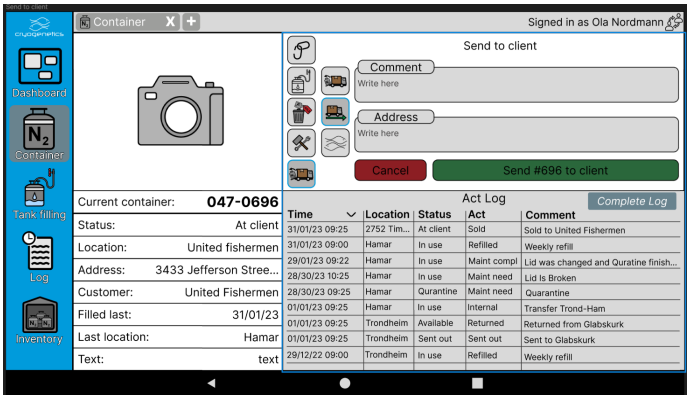
28/2/23

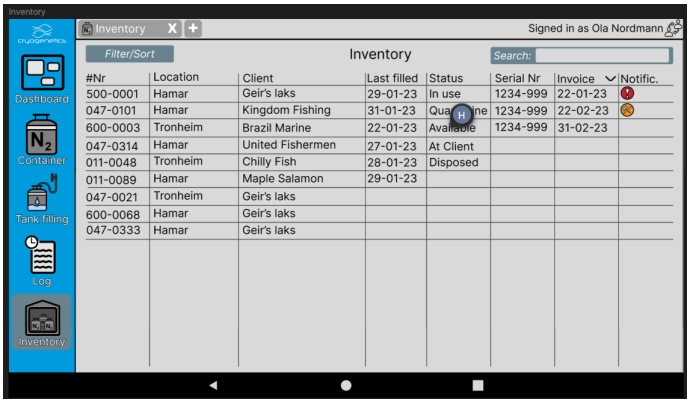
Models

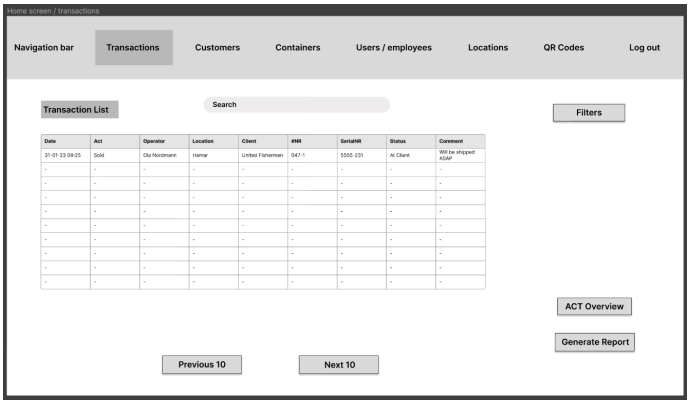
[Link to figma workspace of the Low + high fidelity models]

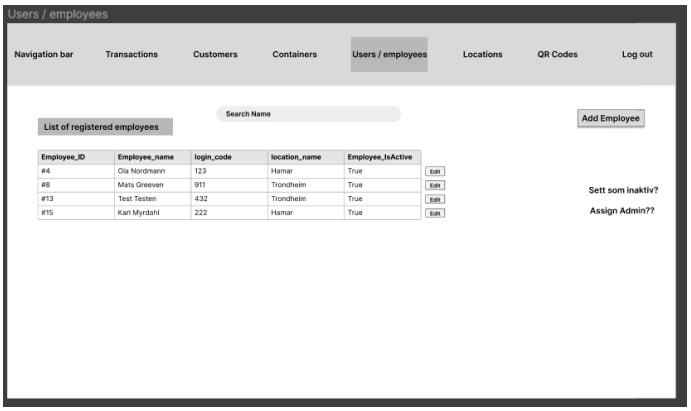












28/2/23

Program progress

Frontend

The mobile application design is finished, most if not all content and functionality is implemented in the design model. We have used the model to create an interactable prototype that Cryogenetics staff got to explore during usertesting. We got mostly positive feedback from the users, but some aspects had room for improvement. The application was intuitive and easy to use according to the results of the usertesting, most users said this was due to simplicity, icons and color choices. Because we did not want to use icons from various suppliers, we have created icons using adobe illustrator to be used for mobile- and web-application.

We did not conduct user testing on the administration website as the main relevant feedback is gathered with our meetings with our Cryogenetics Correspondent. Because of the limited users, we decided it wasn't important to customize the website too much. The main appeal should be simplicity and usability, much like the app. Due to the larger screen size on monitors compared to tablets, we are able to use more text instead of images, reducing the amount of time needed to design the layout.

Backend

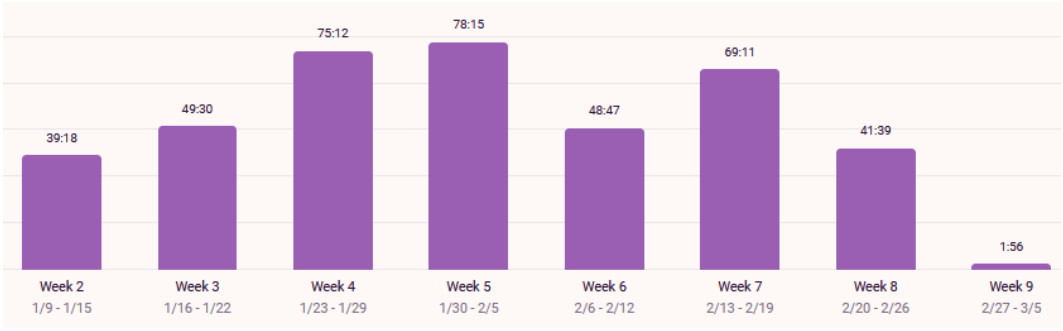
For the backend, we started by creating the conceptual model of the database. This conceptual model was made in multiple iterations, the first being made during the second week of development. For the first couple of weeks a new iteration was made after each meeting with the client, adding and changing content based on the client’s feedback. Eventually, we felt satisfied with the conceptual model and decided to make it physical. The physical model was made in phpMyAdmin, and each time the physical model was changed afterwards, the conceptual model was too. This was done to ensure clear and correct documentation.

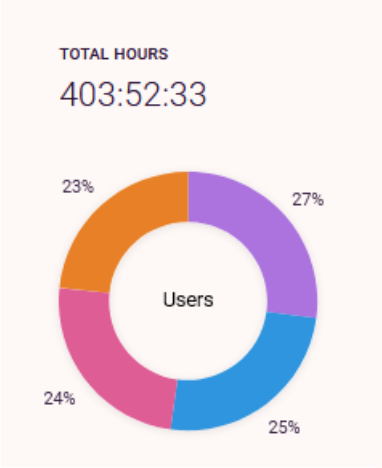
After user testing and some quick updates to the database, we began working on the GoLang server. A list of endpoints with URIs was created and agreed upon by the group so that both front- and backend could work simultaneously without confusion. For the backend, setting up the server with empty endpoints which returned status code OK was a high priority so that the back- and frontend could be connected as soon as possible.

Finally, a list of structures and functions for the server was made, as well as code which connects the server to the database and runs an example query.

28/2/23

Time tracking [february 28th]





	NAME	DURATION	PERCENTAGE
15	Axel Jacobsen	94:23:53	23.37%
29	Havard_ski	109:35:15	27.13%
19	Lars L R	98:46:45	24.46%
22	Mats Greeven	101:06:40	25.04%

Appendix F

Status report 2

Cryogenetics status report 2

This is the second status report for our bachelor thesis project. This report covers our progress as of April 10th.

1. Progress Status

Since the last status report major progress has been made on the app, website, and server.

1.1 Android application

1.1.1 Android design / figma prototype

Android design is close to being completed for functionally, but some minor aspects are not implemented yet and the design is not finished visually. Some fine tuning and design work is still needed to make it look closer to the design prototype.

1.1.2 Navigation

Navigation is about close to completed, the menu navigation is implemented, but we still have to achieve intuitive navigation. Intuitive navigation refers to the ability to navigate the

application in different ways, for example the ability to navigate to a tank that is in the inventory-table.

1.1.3 Multitasking

Multitasking is close to complete, but it still needs some fine tuning to make it work as intended. The management of different android fragments is important to make a seamless application which does not require loading of new activities when the user navigates to a different page. Instead of loading a new activity we switch out the child fragment and delete it from memory when it is no longer needed.

1.1.4 Api calls

API GET calls have been implemented and functions as they should. Currently only implemented in the inventory fragment. The data is initially fetched as a string and we have another function to convert from string into a List<Map<String,Any>> that can be used to access the data. The inventory fragment currently gets data from these functions and assigns it to the recyclerview in the adapter. PUT and POST requests are under development and testing, we will not utilize any other types of requests.

1.2 Web application

1.2.1 SSL Certificate

We've been in contact with Cryogenetics and their IT partners to get a SSL certificate for the admin website. This is required in order to use Azure's multi factor authentication, which will allow admins to log in using their company email. The purchase has been accepted by Cryogenetics, and the issue has been escalated to the IT company's consumer department. We are waiting for them to get back to us.

1.2.2 Website layout

The layout is around 90% complete, the only thing still required are the add/edit buttons and the QR code screen. Using the Figma models shown in the previous status report, the layout has stayed exactly the same.

12.3 HTTP calls

GET requests have been fully implemented along with the backend, and we need to implement the POST and PUT requests on the backend before we do anything else. When we add that functionality to the server, the website's development will be extremely simple: Open a Modal that allows the user to input fields of information. Generate a HTTP request around that data. Display the returned data to the tables.

1.3 Server

Development on the server began by designing the server architecture, coming up with a list of endpoints, and setting up the file structure. After this was done, essential functions such as querying the database and formatting data was prioritized. With these functions in place the first endpoint was made - the GET endpoint for containers. After bug testing and modularizing, safety was our next concern. Research was put into mitigating and protecting against common vulnerabilities such as SQL injections using features of the official Go SQL package. Filters were also added in order to avoid sending unnecessary data to the frontend.

Once the first GET endpoint was complete and secure we began work on the first POST endpoint. There was some difficulty reformatting the data while at the same time protecting against SQL injections due to how the Go SQL package works, but it did not take long before this endpoint was complete too.

Lastly, the first PUT endpoint was developed. A discussion was held about how to format the input data in order to have it consistent, but easy to use. Some of the code from the POST endpoint was reused as most of the work revolved around data reformatting. The endpoint was SQL injection-proofed and modularized, and we were done.

With one endpoint of each type (GET, POST, and PUT) created in a modular fashion, it was easy to create the rest of the endpoints (transaction, client, admin, etc.). However, before we did so, a discussion was held about changing the endpoint URLs. Additionally, functions for joining table data would have to be made in order to reduce the amount of requests from the frontend and redundant data.

These changes were made as well as some changes to the database in order to simplify the new functions, and the backend was declared complete in terms of functionality. Device verification and testing was put on hold so that the group’s entire focus could be used on the app and website.

1.4 Deadlines

We have tried our best to keep up with the deadlines we agreed upon, but when the original GANTT scheme was made the requirements specification was not put into consideration. Due to this oversight we had to make space for writing the requirement specification, which pushed most deadlines back by 1-2 weeks.

Work on the thesis outline has been moved from week 12 to 14. Focus was put into creating core functionalities for the app early, which have made it easier to integrate without unexpected issues.

				WEEK NR.										
ACTIVITY	START	DURATION	END	10	11	12	13	14	15	16	17	18	19	20
User test revision	10	3	12											
Website revision	10	2	11											
App revision	10	2	11											
Backend complete	10	3	12											
Thesis start	12	6	17											
Thesis outline	12	2	13											
Second status report	14	1	14											
Thesis draft	14	4	17											
Development end	15	2	16											
~~User test final round~~	14	1	14											
Website complete	15	2	16											
App complete	15	2	16											
Thesis end	18	3	20											
Third status report	18	1	18											
Thesis review	18	2	19											
Thesis complete	20	1	20											

The image above shows the original GANTT scheme for weeks 10 to 20, it is currently week 14. Completed items are colored green, uncompleted items that have crossed the deadline are red, and not yet met deadlines are orange. The final round of user testing was completed before week 10 instead of week 14, due to involvement of the clients employees.

				WEEK NR.													
ACTIVITY	START	DURATION	END	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Kravspek	7	2	12														
Kravspek draft	7	2	8														
Kravspek ver. 2	9	2	10														
User test revision	10	3	12														
Website revision	10	2	11														
App revision	10	2	11														
Backend complete	10	3	12														
Thesis start	12	6	17														
Thesis outline	12	2	13														
Second status report	14	1	14														
Thesis draft	14	4	17														
Development end	15	2	16														
Backend complete	14	1	14														
Website complete	15	2	16														
App complete	15	2	16														
Thesis end	18	3	20														
Third status report	18	1	18														
Thesis review	18	2	19														
Thesis complete	20	1	20														

The image above shows what the *updated* GANTT scheme looks like. We worked with requirements specification for two weeks before sending a draft, which we then worked another two weeks on before completion. During this time there was little development on all other parts. The other deadlines have been moved to compensate for the time utilized writing the requirements specification.

1.5 Planning, organization & responsibilities

Our organization & planning method has stayed the same; online meetings twice per week to discuss our progress and set new goals until the next meeting. We use our issue board to keep track of our tasks, and we evaluate what works and what needs improvement in our social and work dynamics. In case of any roadblocks or issues, we aim to address them immediately and find solutions before the next meeting.
This has worked well for us, as we have gotten better at breaking down the goals ahead of us to make achievable goals. This has resulted in more effective development, where we achieve great progress as individuals and solve the bigger challenges as a team.

Under a short period we had three people working on the mobile application, this was mostly to give development a speed boost towards the final weeks. However we see that this has caused a slight delay for other tasks, for this reason we are splitting up again. We will be dividing work as shown below until the last few weeks.

Responsibility	Person
Mobile	Håvard
Website	Mats
Deployment Backend Security	Lars
Beginning thesis	Axel

1.6 Report progress

Chapters that have been written

- Requirements specification
- The project plan
- Software development plan
- Contract with client
- Progress plan
- Gantt chart
- Milestones
- Responsibility map

2 Summary of the above points

We have held steady progress on the project overall and are where we are expecting to be. The web application is slightly ahead of schedule, it's mostly finished and we are only missing authentication and some minor things. The mobile application is slightly behind however, but the team is confident it will be finished in time. The cause of the delay is an oversight of not taking into account the time it takes to write the requirement specification. This put us slightly behind on the backend as well. However, since the requirement specification is ready to be put straight into the final thesis we haven't ended up significantly behind on the project overall.

3 Possibilities, threats and problems

The main threats are related to time and security. While we would prefer to have started on the thesis outline earlier, being finished with the requirement specifications means we have some leeway on the rapport side of things. However, to be sure that we have the time needed for the rapport we will be starting on the thesis outline week 14. This way we gain a better understanding of how much work there is ahead of us and mitigate the risk of not completing the rapport.

Additionally, we will put two weeks of work into verification and security for the backend so that we aren't caught off guard in case an issue arises. This means the group is spread thin, each member working on their own, which could become an issue if someone becomes stuck. However, we are in tight communication and will be doing this for about three weeks, so this should not be an issue.

Lastly, there is a possibility that we might not be able to add Microsoft's two factor authentication to our program. This was requested by the client as a method for authentication by the client. The reason for this threat is that documentation related to Microsoft's two factor authentication is sparse, and using it might require a license which we depend on the client to provide. As part of development on the backend's final security features during week 14-16, more research will be put into this. If it is found to be unrealizable or too time consuming, a discussion will be held with the client in regards to alternative solutions to Microsoft's two factor authentication.

4 Motivation and group relations

Currently our group shows good motivation and cooperation in mobile development, making the work efficient and successful. Regardless, there have been challenges with motivation due to inexperience with large development products. In addition, we have not maintained a common working time, which may have hindered efficiency, collaboration and caused more independent choices. Despite these challenges, the team has been able to collaborate, organize and make great progress.

5 Contact with client and advisor

We have made sure to keep in contact with both our advisor and our client during the whole project. However as we have only been developing for the last few weeks we reduced the amount of meetings to suit our and the clients needs. This worked out for us as we had more continuous work sessions, then we would normally have when we had meetings. Now we have less frequent regular meetings, and we reach out to the client when we need to. We will likely communicate more with our advisor to get feedback on our thesis as more parts are added. So far we are satisfied with both our client's and advisor's contributions to our thesis, and we will continue to utilize their skills and knowledge.

Appendix G

Meeting notes

Meeting Notes Cryogenetics Bachelor Thesis

09.01.23 Meeting with client

We had a meeting with Steffen Wolla from Cryogenetics, from 14.00 to 16.00. We went through the project at their offices in Hamar and got a deeper understanding of the project as a whole. During the meeting we got to see their warehouse and how they currently handle the logistics of their liquid nitrogen tanks. We also discussed which key functionality the client needs, and the data we need to store in the database. We also discussed how we could choose to solve different tasks and what technology we should use.

16.01.23 Meeting with client

We had a meeting with Steffen Wolla from Cryogenetics, from 15.00 to 15.30. We went through our low fidelity design, which is still a work in progress at this moment. The client gave us feedback on our design, and we for example got a better understanding of what we should display on the dashboard. We also got a moment to go over our transaction table and database model, where we had missed some categories which we needed to include. We also had some icons to show off, where the client had the option to choose between two iterations of an icon. To finish the meeting the client gave us some positive feedback and guidance for further developing the project.

17.01.23 Meeting with counselor

We had a meeting with Frode Haug from NTNU, from 14.30-15.00. We started the meeting by getting more familiar with each other and the project. Then we got a crash course on how he wants us to write the project plan, which changes the original template. He also gave us insight into what he wants us to write about under the different categories. We also discussed development methods, working remotely and the work in progress project plan.

23.01.23 Meeting with client

We had a meeting with Steffen Wolla from Cryogenetics, from 15.00 to 15.30. We went through some questions we had gathered from the last week, with regards to storage and functionality. Because there was a discussion regarding how we should implement locations

in our storage model, but after understanding the clients needs, we found a solution we could get behind. We also discussed other subjects, like deployment on Microsoft Azure, because we need access to the clients Azure resources.

24.01.23 Meeting with counselor

We had a meeting with Frode Haug from NTNU, from 14.00-14.30. Frode had given us some feedback, by commenting on our WIP project plan. So we collectively went through each subject in the project plan and discussed changes that we should make. The meeting helped us improve our project plan, and was greatly appreciated.

30.01.23 Meeting with client

We had a meeting with Steffen Wolla from Cryogenetics, from 15.00 to 15.30. Where we showcased some progress on our low-fidelity design-iteration of the web-page and mobile-application. When we went through the design on figma, we had many questions about how they wanted the application to work. We also had some questions about the data set, because we discovered it would be beneficial to have an address field, instead of writing this in the comment.

31.01.23 Meeting with counselor

WC, Axel and Mats met physically with Frode, while I was available for comments on MS-Teams. This was the first time Frode got to see the application and got a bigger picture of the functionality we are working towards. He had many questions and gave us mostly positive feedback, even though it is not really his expertise or responsibility for his role.

13.02.23 Meeting with client

We had a meeting with Steffen Wolla from Cryogenetics, from 15.00 to 15.30. Where we showcased some progress on our high-fidelity design-iteration of the web-page and mobile-application. The web page prototype was highly functional, and included all aspects that the client had asked for. We also discussed the addition of tank statuses and acts. In addition we discussed the new tank menu, which got positive feedback from the client.

14.02.23 Meeting with counselor

We had a meeting with Frode Haug from NTNU, from 14.00-14.30. Frode had given us some feedback, by commenting on our requirement specification. The feedback was not comprehensive because the format was not correct, and we had to add much more to the

document. We also discussed low- and high-level use cases, since we had only made a use case model, which was not sufficient.

21.02.23 Meeting with counselor

We had a meeting with Frode Haug from NTNU, from 14.00-14.15. Frode had given us feedback on our requirement specification, by commenting on the document. The feedback was mostly positive, but we still had some format problems and missing categories. We also had to write the text in the future time form, instead of the mix we currently have of different timeforms.

22.02.23 Meeting with client

We had a meeting with Steffen Wolla from Cryogenetics, from 15.00 to 15.30. Where we discussed the feedback from usertesting, and the solution to solve concerns and wanted functionality. We also discussed the opportunity to contact an employee of Cryogenetics in Trondheim which will use the system extensively, to get more data. We also discussed MS Azure access, which is taking longer than expected due to confidentiality concerns.

06.03.23 Meeting with client

We had a meeting with Steffen Wolla from Cryogenetics, from 15.00 to 15.30. Where we discussed the feedback from usertesting with an employee from Trondheim, and we discussed some solutions to a concern and some additional functionality. We also discussed some changes to the tank-menu, and some minor changes to act-table. Steffen had arranged azure access for the group, and we could confirm it was working as intended. We also mentioned some of the progress regarding the documentation, and the future plans for the next two weeks.

07.03.23 Meeting with counselor

We had a meeting with Frode Haug from NTNU, from 14.00-14.15. Frode had given us some more feedback on our requirement specification, by commenting on the document. The feedback was mostly positive, but we still had some format problems and Frode had made some changes to the expected format of the document. We also briefly discussed some questions we had regarding the changes he wanted us to add.

14.03.23 Meeting with counselor

We had a meeting with Frode Haug from NTNU, from 14.00-14.30. Axel and Mats visited Frode's office on campus in Gjøvik, they brought a laptop and a tablet to usertest Frode. The remaining group members joined the meeting digitally on teams.
We also briefly discussed some questions about the requirement specification, and the results of usertesting at Cryogenetics.

20.03.23 Meeting with client

We had a meeting with Steffen Wolla from Cryogenetics, from 15.15 to 15.45. Where we discussed the feedback from usertesting at Cryogenetics, and the changes we wanted to make to the application. We also asked to usertest another employee in Trondheim to get more data, this employee is likely to be the most active user of the application. Steffen asked the employee for a date and time of the meeting, and we later agreed on the time by mail.

12.04.23 Meeting with counselor

We had a meeting with Frode Haug from NTNU, from 14.00-14.30. We discussed the second status report, which we had sent to Frode by mail prior to the meeting. Frode was thrilled to see a detailed status report, but he also said it was longer than necessary. We discussed the project's progress and he answered some questions from the group. We also discussed the thesis and the structure of the document and the attachments.

14.04.23 Meeting with client

We had a meeting with Steffen Wolla from Cryogenetics, from 13-13.30. We showed a demo of the web and mobile application, Steffen was excited to see the applications coming together. We were missing some minor features, which we discussed and explained to the client. We also had some minor bugs occur during the demo, which we took note of.

05.05.23 Meeting with client

We had a meeting with Steffen Wolla from Cryogenetics, from 13-13.30. We showed a demo of the web and mobile application, unfortunately we had some issues with "null" values and a bug occurred. But overall the demo was successful and the participants were excited to see

the working applications. We also agreed to another meeting where we will run the demo on their Azure resources and give them the resources to test it out for themselves.

09.05.23 Meeting with counselor

We had a meeting with Frode Haug from NTNU, from 14.00-14.30. Where we discussed the deployment of the application for Cryogenetics, because they want to start using the product. We agreed that it is not our responsibility to give the client a production environment for production use, but we still wish to give them a test environment. It will be the client's responsibility to set up a production environment and ensure that the software is maintained and secure to store confidential data. We also discussed the bachelor thesis document, we had many questions regarding the structure. After the meeting we were able to restructure the document and get a better understanding of what we needed to work on.

15.05.23 Meeting with counselor

We had a meeting with Frode Haug from NTNU, from 14.00-14.30. Prior to the meeting we sent Frode the latest version of our thesis, which he read and commented on. We discussed comments which we wanted him to elaborate on, we also discussed the structure of the document. We also got answers for questions regarding where we should place different chapters or sub-chapters in our document. Frode provided more insight into how we can improve the text, and we got to thank him for his contributions to our thesis.

Appendix H

User testing Cryogenetics employees

Cryogenetics feedback form

Explore the prototype on your own, and write a short summary of your first impression. *

Very nice design, intuitivt design, fine symboler, veldig oversiktlig.

Navigate to "Dashboard", what do you think about this feature?

Very nice to have features, can be useful to get an overview, hard to understand that the mens are split.

What did you like about the Dashboard? (color, layout, content, icons, features etc.)

Like the design, like the battery tank indicator for nitrogen liquid.

What don't you like about the Dashboard? (color, layout, content, icons, features etc.)

How would you rate the Dashboard?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

Is there anything you want to add or remove from the Dashboard?
Any text that could be changed?

Navigate to the "Log", what do you think about the Act Log?

DD/MM/YY for other countries, looks nice.

What did you like about the Act Log? (color, layout, content, icons, features etc.)

Balansed color choices.

What did you not like about the Act Log? (color, layout, content, icons, features etc.)

How would you rate the Act Log?

	1	2	3	4	5	6	
Very poor	○	○	○	○	○	◉	Very good

Is there anything you want to add or remove from the Act Log?
Any text that could be changed?

Navigate to the "Inventory", what do you think about the Inventory?

What did you like about the Inventory? (color, layout, content, icons, features etc.)

What did you not like about the Inventory? (color, layout, content, icons, features etc.)

How would you rate the Inventory?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

Is there anything you want to add or remove from the Inventory?
Any text that could be changed?

Wish to add battery indicator to last filled column, serial number is not strictly neccecjary here. Store date of container use, yearly service for tanks older than about 8-10 year and approval.

Navigate to Tank Filling for multiple containers, what do you think about how this feature works?

Looks nice, If I am in a room I can scan them all, and then quickly get the job done.

Mark buttons/features which you think are necessary?

- [x] Cancel all
- [x] Undo
- [x] Remove selected
- [x] more details
- [ ] Andre: .....

What would rate the multiple Tank filling menu?

	1	2	3	4	5	6	
Very poor	○	○	○	○	○	◉	Very good

Is there anything you want to add or remove from the Inventory?
Any text that could be changed?

.....

Navigate to "Container", and press camera icon to open a container. What do you think about this layout?

Since there is space, i think it is nice to have.

How would you rate the menu inside the container window?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

What do you think about the menu inside the container window?

The icons are nice, i can understand the functionality behind, I like nested menus because it is easier to use..
I would like the order of the table to have Time, location, Act, status, comment.

Any comments about Link and Unlink client?

Looks alright.

Any comments about individual refill?

I like how it is.

Any comments about Discard or sell?

Looks okay.

Any comments about Maintence need and complete?

What do you think about the manual act feature?

It looks handy, but I can not see any specific use case for myself.

Any comments about Transaction?

Looks nice.

Go to the Transaction menu, and press the square on the bottom navigation bar, which icon would you rather use on the internal transfer menu?

- (●) Cryogentics fish
- ( ) N^2 Tank and office/lab
- ( ) Andre: _____

How would you rate the functionality of the container menu?

	1	2	3	4	5	6	
Very poor	( )	( )	( )	( )	(●)	( )	Very good

Navigate to the dashboard and press the square on the bottom naigation bar, do you prefer the main menu on the side or the bottom?

- ( ) Side menu
- ( ) Bottom menu

(Navigate to the dashboard and) press the triangle on the bottom naigation bar, do you prefer a verticale layout?

- ○ Landscape
- ○ Portrait
- ○ Both

How would you rate the draft? *

	1	2	3	4	5	6	7	8	9	10	
Very poor	○	○	○	○	○	○	○	○	◉	○	Completly perfect

If there are any, what changes would you make to the program?

If there are any, what additional functionalities would you like to see in the program?

How user friendly was the program? (How hard was it to navigate, understand content etc.) *

	1	2	3	4	5	6	7	8	9	10	
Impossible to navigate and use	○	○	○	○	○	○	○	○	○	◉	Intuitive and easy to use

Were the colors used in the draft to your liking?

- (●) Yes
- ( ) No
- ( ) Andre: _____

Additional Feedback

_____

If we have time left, explore the web application and please give your feedback.

Do not see a use for the zeros in running number.

Cryogenetics feedback form

Explore the prototype on your own, and write a short summary of your first impression. *

Oversiktlig, intuitive menu.

Navigate to "Dashboard", what do you think about this feature?

Looks quite usefull, nice to have.

What did you like about the Dashboard? (color, layout, content, icons, features etc.)

I like the balanced color choices, text size is nice, icons are good.

What don't you like about the Dashboard? (color, layout, content, icons, features etc.)

I would like to have the feature to see the complete comment for example.

How would you rate the Dashboard?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

Is there anything you want to add or remove from the Dashboard?
Any text that could be changed?

I would like a popup to see a whole field that does not fit.

Navigate to the "Log", what do you think about the Act Log?

OPER is not self explaintory.

What did you like about the Act Log? (color, layout, content, icons, features etc.)

Nice simplified, color for statuses, green for available for example.

What did you not like about the Act Log? (color, layout, content, icons, features etc.)

How would you rate the Act Log?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

Is there anything you want to add or remove from the Act Log?
Any text that could be changed?

Instead of code it would be nicer to have initials, so i can easier see which is which.

Navigate to the "Inventory", what do you think about the Inventory?

Could be nice to separate the screen into two, drag and drop to move tanks internal. Could be split horizontally.
I would love to have a feature to hide columns or add columns.
I do not think serial number is not neccessary, but it strictly neccessary to store.

What did you like about the Inventory? (color, layout, content, icons, features etc.)

Simple style, intiuitive.

What did you not like about the Inventory? (color, layout, content, icons, features etc.)

Unneccessary columns,

How would you rate the Inventory?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

Is there anything you want to add or remove from the Inventory?
Any text that could be changed?

instead of #NR it could be tank id.
Specify invoice means, what does this date mean.

Navigate to Tank Filling for multiple containers, what do you think about how this feature works?

Really cool, undo is a nice feature.
Cancel all, the whole list removes.

Mark buttons/features which you think are necessary?

- [x] Cancel all
- [x] Undo
- [x] Remove selected
- [x] more details
- [ ] Andre:

What would rate the multiple Tank filling menu?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

Is there anything you want to add or remove from the Inventory?
Any text that could be changed?

I would like to have the battery indicator and last refilled date.

Navigate to "Container", and press camera icon to open a container. What do you think about this layout?

Layout is nice

How would you rate the menu inside the container window?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

What do you think about the menu inside the container window?

Very good, easy to use, icons are nice.

Any comments about Link and Unlink client?

It works

Any comments about individual refill?

Easy and nice

Any comments about Discard or sell?

Good, we need to know who it is sold to.

Any comments about Maintence need and complete?

What is maintence, yearly maintence?
It would be nice to have production year to be extra causus with these tanks.

What do you think about the manual act feature?

Could be handy to have, but i think it is very easy to make human errors with this feature.

Any comments about Transaction?

One customer may have many address.
I want be able to change linked client easier, since

Go to the Transaction menu, and press the square on the bottom navigation bar, which icon would you rather use on the internal transfer menu?

- (●) Cryogentics fish
- ( ) N^2 Tank and office/lab
- ( ) Andre:

How would you rate the functionality of the container menu?

	1	2	3	4	5	6	
Very poor	○	○	○	○	●	○	Very good

Navigate to the dashboard and press the square on the bottom naigation bar, do you prefer the main menu on the side or the bottom?

- ○ Side menu
- ○ Bottom menu

(Navigate to the dashboard and) press the triangle on the bottom naigation bar, do you prefer a verticale layout?

- ○ Landscape
- ○ Portrait
- ○ Both

How would you rate the draft? *

	1	2	3	4	5	6	7	8	9	10	
Very poor	○	○	○	○	○	○	◉	○	○	○	Completly perfect

If there are any, what changes would you make to the program?

Small changes I would like to add

If there are any, what additional functionalities would you like to see in the program?

Some tanks dont hold nitrogen as efficiently, It could be nice with a yellow indicator for tanks that do not hold nitrogen as well, red for really badly.
Can you have a login option for trondheim/hamar?

How user friendly was the program? (How hard was it to navigate, understand content etc.) *

	1	2	3	4	5	6	7	8	9	10	
Impossible to navigate and use	○	○	○	○	○	○	○	○	◉	○	Intuitive and easy to use

Were the colors used in the draft to your liking?

- ◉ Yes
- ○ No
- ○ Andre: _____

Additional Feedback

Colors, menus and content
Multitasking, I do not think it is strictly neccessary. But there is some use cases where i think it quite handy.
We use tank mostly, not container.

If we have time left, explore the web application and please give your feedback.

Cryogenetics feedback form

Explore the prototype on your own, and write a short summary of your first impression. *

Not answered.

Navigate to "Dashboard", what do you think about this feature?

What did you like about the Dashboard? (color, layout, content, icons, features etc.)

What don't you like about the Dashboard? (color, layout, content, icons, features etc.)

How would you rate the Dashboard?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

Is there anything you want to add or remove from the Dashboard?
Any text that could be changed?

We would like to #NR to be changed to tank id.
Maybe it would be better to only show critical information and
I think the invoice features would only be suited for the admin page/web-app, or a specific menu option for this feature.

Navigate to the "Log", what do you think about the Act Log?

Oper could be replaced with initials.
Nice to have

What did you like about the Act Log? (color, layout, content, icons, features etc.)

What did you not like about the Act Log? (color, layout, content, icons, features etc.)

How would you rate the Act Log?

	1	2	3	4	5	6	
Very poor	○	○	○	○	○	◉	Very good

Is there anything you want to add or remove from the Act Log?
Any text that could be changed?

Navigate to the "Inventory", what do you think about the Inventory?

We would like to have the battery nitrogen level indicator to be on the separate inventory.

What did you like about the Inventory? (color, layout, content, icons, features etc.)

What did you not like about the Inventory? (color, layout, content, icons, features etc.)

We need to have seperate icons for not paid and needs to be filled.

How would you rate the Inventory?

	1	2	3	4	5	6	
Very poor	○	○	○	○	○	◉	Very good

Is there anything you want to add or remove from the Inventory?
Any text that could be changed?

Add battery nitrogen level indicator.
Different icon for invoice not paid.

Navigate to Tank Filling for multiple containers, what do you think about how this feature works?

Mark buttons/features which you think are necessary?

- [x] Cancel all
- [x] Undo
- [x] Remove selected
- [x] more details
- [ ] Andre:

What would rate the multiple Tank filling menu?

	1	2	3	4	5	6	
Very poor	○	○	○	○	○	◉	Very good

Is there anything you want to add or remove from the Inventory?
Any text that could be changed?

Navigate to "Container", and press camera icon to open a container. What do you think about this layout?

Looks good, nice to have the Log on the button. We need to have warning symbols for each seperate container.

How would you rate the menu inside the container window?

	1	2	3	4	5	6	
Very poor	○	○	○	○	○	◉	Very good

What do you think about the menu inside the container window?

Intuiative to use.

Any comments about Link and Unlink client?

Good

Any comments about individual refill?

Any comments about Discard or sell?

Any comments about Maintence need and complete?

What do you think about the manual act feature?

I think it is fine to have this feature.

Any comments about Transaction?

Go to the Transaction menu, and press the square on the bottom navigation bar, which icon would you rather use on the internal transfer menu?

- ○ Cryogentics fish
- ○ N^2 Tank and office/lab
- ○ Andre:

How would you rate the functionality of the container menu?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

Navigate to the dashboard and press the square on the bottom naigation bar, do you prefer the main menu on the side or the bottom?

- ○ Side menu
- ○ Bottom menu

(Navigate to the dashboard and) press the triangle on the bottom naigation bar, do you prefer a verticale layout?

- ○ Landscape
- ○ Portrait
- ○ Both

How would you rate the draft? *

	1	2	3	4	5	6	7	8	9	10	
Very poor	○	○	○	○	○	○	○	◉	○	○	Completly perfect

If there are any, what changes would you make to the program?

If there are any, what additional functionalities would you like to see in the program?

We would love to have an option to add comment to multiple fill menu, and to each seperate icon.

How user friendly was the program? (How hard was it to navigate, understand content etc.) *

	1	2	3	4	5	6	7	8	9	10	
Impossible to navigate and use	○	○	○	○	○	○	○	○	◉	○	Intuitive and easy to use

Were the colors used in the draft to your liking?

- ○ Yes
- ○ No
- ○ Andre: .....

Additional Feedback

Multitasking would be nice to have to quickly check and do different actions at the same time, would defintly be used in some operations.

Indicator for bigger tanks 5-600 which is approved, organized innside before link to client.

If we have time left, explore the web application and please give your feedback.

.....

Cryogenetics feedback form

Explore the prototype on your own, and write a short summary of your first impression. *

Looks really nice, cool to have all the information about our product in our.

Navigate to "Dashboard", what do you think about this feature?

Handy to get an overview of contents.

What did you like about the Dashboard? (color, layout, content, icons, features etc.)

What don't you like about the Dashboard? (color, layout, content, icons, features etc.)

How would you rate the Dashboard?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

Is there anything you want to add or remove from the Dashboard?
Any text that could be changed?

Navigate to the "Log", what do you think about the Act Log?

What did you like about the Act Log? (color, layout, content, icons, features etc.)

What did you not like about the Act Log? (color, layout, content, icons, features etc.)

How would you rate the Act Log?

	1	2	3	4	5	6	
Very poor	○	○	○	○	○	◉	Very good

Is there anything you want to add or remove from the Act Log?
Any text that could be changed?

I would like to have initials, instead of operator codes.

Navigate to the "Inventory", what do you think about the Inventory?

What did you like about the Inventory? (color, layout, content, icons, features etc.)

What did you not like about the Inventory? (color, layout, content, icons, features etc.)

How would you rate the Inventory?

	1	2	3	4	5	6	
Very poor	○	○	○	○	○	○	Very good

Is there anything you want to add or remove from the Inventory?
Any text that could be changed?

Navigate to Tank Filling for multiple containers, what do you think about how this feature works?

Mark buttons/features which you think are necessary?

- [x] Cancel all
- [x] Undo
- [x] Remove selected
- [x] more details
- [ ] Andre: .....

What would rate the multiple Tank filling menu?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

Is there anything you want to add or remove from the Inventory?
Any text that could be changed?

Looks like it have all the information

Navigate to "Container", and press camera icon to open a container. What do you think about this layout?

How would you rate the menu inside the container window?

	1	2	3	4	5	6	
Very poor	○	○	○	○	◉	○	Very good

What do you think about the menu inside the container window?

Any comments about Link and Unlink client?

Any comments about individual refill?

Any comments about Discard or sell?

Good that we can comment on discard or sell, so we can write the reason for discardment.

Any comments about Maintence need and complete?

It is really helpful to know when maintence is needed for a tank, since we often forget to make strictly neccessary.

What do you think about the manual act feature?

Sometimes we have to make changes to correct specific issues, which makes this

Any comments about Transaction?

It is very handy to have comments on each Transactions.

Go to the Transaction menu, and press the square on the bottom navigation bar, which icon would you rather use on the internal transfer menu?

- (●) Cryogentics fish
- ( ) N^2 Tank and office/lab
- ( ) Andre: _____

How would you rate the functionality of the container menu?

	1	2	3	4	5	6	
Very poor	( )	( )	( )	( )	(●)	( )	Very good

Navigate to the dashboard and press the square on the bottom naigation bar, do you prefer the main menu on the side or the bottom?

- (●) Side menu
- ( ) Bottom menu

(Navigate to the dashboard and) press the triangle on the bottom naigation bar, do you prefer a verticale layout?

- ○ Landscape
- ○ Portrait
- ◉ Both

How would you rate the draft? *

	1	2	3	4	5	6	7	8	9	10	
Very poor	○	○	○	○	○	○	○	○	◉	○	Completly perfect

If there are any, what changes would you make to the program?

If there are any, what additional functionalities would you like to see in the program?

It could be nice with an option to fill all tanks, but i see that i

How user friendly was the program? (How hard was it to navigate, understand content etc.) *

	1	2	3	4	5	6	7	8	9	10	
Impossible to navigate and use	○	○	○	○	○	○	○	○	○	◉	Intuitive and easy to use

Were the colors used in the draft to your liking?

- (●) Yes
- ( ) No
- ( ) Andre: _____

Additional Feedback

Everything is here, but i think I might have more feedback after mor
It would be cool to have a record of the production date of tanks, because tanks get "worn" and cannot be used after 10 years.

If we have time left, explore the web application and please give your feedback.

Appendix I

Notes from User Testing

Elin Bergsett

* La merke til tankene først

* "Tanknivået er ut ifra når den var fylt sist"

    * Likte batterimoddelen, slipper å se på datoene, raskere å se

* Prøvde å trykke på en spesifikk dunk

    * Antok at man kan se historikken bak hver tank

* Antok at man kan bla ned tankene i overblikk-siden (scroll/recycler-view)

* Liker symbolene

    * Ikke like selvforklarende: Link/Unlink

* Antok at man kan søke etter dato på log

    * Filtrere etter både kunde og lokasjon

* Foretrekker dd/mm/åå, bekymret for misforståelse i USA og andre land

    * Står "dato/måned/år som i dmy", vil ha det forklart et sted

* Liker godt fargene og symbolene, synes det er oversiktlig

* "Hvor nyttig er dashboard"

    * Ikke klart at log og fylling er uavhengige tabeller, trodde comment var forrige handling på tank 1

    * Spurte om maintenance merke og rødt utropstegn.

* Trodde Internal transfer ikonet var shorthand for hjemaddressen til kunden

* Forstod return symbolet

* Dashboard rating 1-6: 5

* Ingen kommentar på ting å legge til/fjerne fra dashboard

* Log: Skal være presist hvilket dato-format som brukes

    * Passe mengde farger, likte fargene

    * Log rating 1-6: 6

* Inventory:

    * "Går det an å ha fyllestatus her også (batteri-indikatoren), ellers synes jeg det ser veldig bra ut"

    * "Alle tanker over en viss alder skal kontrolleres en gang i året, kanskje ha et varsel for visse eldre dunker"

        * => Age variabel for tanker. Tanker over 8-10 år gamle burde få årlig service.

        * Warning for de som er over alderen og skal godkjennes

    * Inventory rating 1-6: 5

* Tank filling:

    * "Det er bra"

    * Noen knapper som er overflødige?: "Vil tro at det er bruk for alle"

    * Trodde man måtte hake av før cancel all. Klargjør?

    * Likte undo, "alltid kjekt med å kunne gå tilbake"

    * Tank filling rating 1-6: 6

* Likte nested menus måten å gjøre ting på "jeg liker at det er få valg til å begynne med"

    * terningskast: 5

* "Men ein ting her: At en ACT kanskje kunne ha tenkt meg den mellom time og location"

    * "Status etter ACT"

        * => time, location, act, status

* Likte hvordan unlink og comment funker sammen

* Discard or sell:

    * Likte det

* Manual act:

    * Kan ikke se for seg når hun vil bruke det, "men sikkert fint med muligheten"

    * "Ingen ting er mer frustrerende enn når man vil velge en meny men ikke får valgt det man skal"

* Transaction menu:

    * "Synes det ser fint ut"

* Firkant-knapp-meny

    * Ikonene: Skjønte fabrikken, "internal er jo cryo, kanskje kult å bruke logoen", 5/6 terningskast

* Hele skissen:

    * "Synes det ser veldig fint ut"

    * 1-10: 9

* Admin page:

    * Foretrekker QR-koden med tallet til høyre uten null

* Dunker som kommer fra kunde tomme burde stå tomme i 1 uke uten nitrogen

    * Alltid 1 uke i kvarantene, litt usikker på om de klarer å holde det i gang selv

Arthur Zolothare

* Førsteinntrykk

    * "Det jeg liker veldig godt er at det er veldig enkelt, ikke for komplisert layout, og det er veldig oversiktlig"

Dashboard: 5/6

* Likte å kunne se hva som skal fylles først

* "Hvordan skal selve invoice delen oppdateres"

    * Viste mye bekymring for hvordan fakturering gjøres. F.eks. hvis tanken har vært hos kunde i bare 2 uker, men er fakturert for en måned.

    * Likte hvordan månedlige rapporter kunne genereres

        * Ville at man skal kunne selektere dato fra til for månedsrapport

* "Liker ikke alt for flashy farger, så liker disse, fargene til cryogenics"

    * Likte tekststørrelse, font, ikoner

    * Vil at man skal kunne trykke på en comment for å utvide den og se hele samtidig

* Lurte på: Hva er poenget med faner når man kan bytte meny på venstre-siden

    * Multi-tasking feature: Virket ikke veldig positiv.

        * Kan hende man vil se info om én tank i spesifikk imens man fyller for å se om den har begynt å lekke.

* Vil at man skal ha forskjellige grader av advarsel, likt som en bil.

    * Dårlig kapasitet -> gul, burde ikke sende til kunde i lange perioder

    * Foreslår: Ikon av tank som viser hva som er galt.

* Forventer ikke så mye av dashboardet , vil bare ha en generell oversikt

Log: 5/6

* "Hva er opper." -> kort for operator. "Hva er det?" -> hvem som har gjort aktiviteten. "Ja det er bra."

* "I utgangspunktet er det nok info å begynne med, på loggen, så det er bra"

* Ville ha fargekode på status

    * Available -> grønn.

* Istedenfor operasjon nr: Bruk initsialer

    * => Navn/initsialer i databasen

Inventory: 5/6

* Det hadde gått an å dele skjermen i to: hamar og trondheim

* Mange sendinger mellom hamar og trondheim, hadde likt om man kunne sveipet over skjermen raskt og lett, siden det bare er internal transfer

* Invoice ikke nødvendig

* "Hadde det vært mulig å ha vis/gjem kolonner"

    * Ikke alltid man er interessert i å se noe annet enn tank nr og én ting til

* Liker den enkle stilen

* Ville likt farge på øverste rad

* Serienr er ikke så nødvendig her, men veldig viktig på container details

    * "Bare f.eks. hvis det skjer noe og man skal informere leverandøren om at tanken er ødelagt"

* Kunne kalt nr for "tank ID, det er det jeg bruker når jeg snakker med kunde"

* Bruker "tank", ikke "container"

* "Går det an å på en måte bare spesifisere hva invoice betyr, om det er invoiced UNTIL eller noe annet"

Tank filling menu for multiple containers: 5/6

* "Det er kjempekult"

* Ingen misforståelse med "cancel all" knappen her

* Likte "more details" knappen godt

* "Hvor kan man f.eks. se tank filling history for den spesifikke knappen?"

    * F.eks. hvis de scanner tank 24 vil de se tank filling history for å se om det har blitt hoppet over. Kan hende noen glemte, da legger de merke til at de må fylle mer.

    * Etter å ha blitt forklart hvordan det kan gjøres: "Det er OK da"

* Tenker på human error:

    * Hvis man f.eks. er lat og fyller først så scanner alle, det kan hende man har hoppet over en tank

* Tolket at datoene på oversikten er sortert etter når de ble fylt (halvveis sant)

    * Vil ha litt farge, ikonene på toppen mindre, gjøre bildene av batteriene større

        * Eller erstatte med: En ekstra kolonne med nr(#), client, og "refill date"

Container menu:

* "Layouten er veldig bra"

* ville at deadspacen skulle bli brukt

* Manuel act er human error mode xD

* "Icons er veldig fine"

* "Hvordan er det når man får tanken tilbake, trykker på unlink client, affiliasjonen til tanken"

    * "Går det an at når man logger inn, velger man affiliation hamar/trondheim"?

Individual refill:

* "Som sagt, alt er veldig enkelt å greit"

* "Det viktige med refill er registrering av tank, tidspunkt, og evt. noen kommentarer om tanken er veldig tom, og se loggen av tanken"

* Ville ikke ha feature for å fylle i går

Discard and sell:

* "Det som kunne vært alright er om man MÅ legge til hvem det er solgt til"

* "Comment section er bra, da kan man skrive nødvendige ting der, det viktigste er hvem man har solgt det til"

Maintenance needed:

* "Eneste jeg kan tenke på er: Man får tanken tilbake av en kunde, har vært på anlegg med unik helsestatus der, disinfeksjon/vasking av tanken er veldig viktig"

* "Produksjonsår er del av serienummer, typ 2022XYZ"

    * Vil ha mulighet til å se production year for å sammenlikne med hvor fort nitrogenen fordamper

Manual act:

* Problem med human error? "Over alt. Kan skje hvor som helst, når som helst."

Transaction: 5/6

* Antok at man bruker det når man får tanken tilbake fra kunde

* Viste bekymring for hvor man registrerer kunde, viste forvirring om link/unlink/send/recieve

* "{om link/unlink} Det som kan skje, det vil ikke alltid være mulighet for å kjapt registrere en kunde, når man sender ut"

    * Vil at det skal være mulig å registrere kunde imens man sender ut, uten å gå unlink->link først

* Viste bekymring om kunde->kunde transfer

* Internal transfer icon: Foretrekker cryogenetics logo, den andre ser ut som en "ukrainian bunker"

Hele prototypen: 7/10 på farger, ikoner, meny, funksjoner. Enkelt å forstå

Inger og Anelene

Log:

* Vil at man skal kunne velge kategori(er) å se

* Forvirret over "oper." loginkode, vil ha initsialer også

Inventory: 6/6

* Antok at inventory viser tankene på de forskjellige beholdningene

    * Antok at man kan søke lokasjon: Hamar, så kommer kun Hamar opp

        * Gjøre filtrering og søking til samme? Typ discord "channel:DnD Dolbus"

* Fylling > Invoice, vis den tidligere

* Snakk om testing av tanker for å sjekke om de lekker

    * Hvert år? At man kan manuelt legge til at en tank burde sjekkes?

* Vil ha batteri-ikonet her

* Notification: Forskjell mellom fylling og fakturering; vil ha forskjellige ikoner

    * Gjem fakturering litt fra appen, ikke av interesse for dem

        * "Viktig at det som er rødt er krise"

Tank filling:

* Vil at man skal kunne legge til kommentarer for både enkeltdunker og alle dunker under fylling

    * => Tank comment? Vil ha for spesifikke tanker hvis de er sus

* Vil at man skal kunne skrive kommentar på spesifikke dunker under "more details"

* Vil at "man skal kunne gå rett tilbake til der man var"

* Likte de fire knappene

Dashboard:

* Tank nr -> tank ID

* "Sånn det her har nå er det et lite utvalg av tanker, vi har mange fler"

* Vurderte at det hadde vært bedre med å se én kategori om gangen på dashboard

* Tenker at invoice er viktig, men det er ofte andre som gjør det enn de som fyller

    * Flytt invoice vekk fra dashboardet

    * Tenkte at de som bruker admin pagen kan gjøre det

        * Faktureres ofte én gang i året per tank bare

* "Kanskje hvis man hadde hatt at den loggen lå inne på inventory, ikke at det er to separate lister"

* Litt "meh" om loggen på dashboard? "Det er viktig at den er der, men det er ikke det vi har fokus på på daglig drift liksom"

* En "logg"-knapp på bunnen av inventory

Container:

* Syntes det var greit at "maintenance need, maintenance complete"

* Antok at når man trykker unlink får faktorerings-folka beskjed om det

* Vil ha varsling for tanker som er maintenance need men ikke maintenance complete, og at det skal ligge på dunk info.

    * Vil ha grønn fargekode når den er fiksa, og viktigst at det er rød fargekode på venstre bunn når den ikke er vedlikeholdt

Manuell act:

* "Ser greit ut"

* Vil ha rød varsling hvis man prøver å sende ut maintenance needed tank til kunde

    * Samme med årlig checkup

* Vil ha mulighet for å legge til maintenance varsel for dunker som ikke er riktig organisert på innsiden, slik at de ***må*** fikses før de sendes(linkes) til kunde

    * Spesifikt 500-literne

Appendix J

Mobile Software Development Plan

Software Development Plan for mobile application

To create our Android application for mobile devices we decided it would be beneficial to create a software development plan. The android application is quite large and complex, to develop it effectively our team will work closely together and utilize the plan to divide tasks between us. Effective dividing of tasks has many benefits, but we are mostly interested in keeping merge errors to a minimum and to make swift progress.

Activities or Fragments

To create a multitasking android application we will only use 2 activities, this is to create a seamless experience where the user can rapidly navigate and switch fragments without loading a new activity. Separate activities can be imagined as separate processes, and when starting a new activity there will be a delay while the new activity is loaded and the old one is finished. This delay will create a poor user experience if we were to create our multitasking UI with mostly activities. Therefore we will create the UI using mostly fragments, since they are modular and we can quickly switch between them without delay.

Activities:

- Authentication
- Main activity

Fragments:

- Log
    - Header with title, search and filter/sort.
        - Search implementation
        - Filter implementation
        - Sort implementation
    - Tablenames
        - display for each column of th
    - Recycleview
- Partial log for tank view and dashboard
    - Header with title, search and filter/sort.
    - Tablenames
    - Recycleview
- Inventory
    - Header with title, search and filter/sort.
    - Tablenames
    - Recycleview
- Partial Inventory for dashboard and tank filling menu

- Tank filling
    - Menu for tank filling
    - List of Scanned tanks
- Tank
    - Tank menu
    - Tank menu content
        - Link or unlink
        - Send to client
        - Return from client
        - Internal Transfer
        - Refill
        - Dispose
        - Manual Act
        - Maintenance
- Camera for Tank, and Tank filling
- Tab manager
    - Tabs (5 menu options)
    - Display name and switch user button
- Sidebar

The main tasks of development is:

1. Build Android UI
    - **Purpose**
        - The purpose of this task is to create all the UI elements for the application, everything that the user can see.
    - **Approach**
        - Build the Android UI in XML layout files, replicate the final Figma design using Android Studio's Layout Editor. The design may be slightly different to avoid spending unnecessary time making big changes to Android elements.
        - All visible elements should be created with the use of XML, unless it is strictly necessary to create the element with other techniques.
        - All layouts should be created using constraint layout to optimize for different screen sizes, unless it is strictly necessary to use other layouts.
    - **Goals**
        - Replicate all elements from Figma layout within reasonable similarity.
        - Create a scalable layout for different screen sizes of tablets, only for landscape orientation.
2. Navigation
    - **Purpose**
        - The purpose of navigation is to intuitively let the user navigate the application. This task includes all aspects of navigation from log in to log out, except the multitasking functionality.

- **Approach**
  - Implement the sidebar navigation, which navigates to the 5 main menu options.
  - Implement intuitive navigation, when a tank is in a list, it can be pressed to open that specific tank, the header can be pressed to open a complete log or inventory. This includes the log, partial log, partial inventory, see complete log etc.
- **Goals**
  - Navigation works similar or better than the figma prototype.
  - Handle inaccurate touches correctly, like long presses, swipes and others.
  - All elements that could potentially be used as buttons to navigate the user are implemented, within reason.

3. Login, authentication and switch user
   - **Purpose**
     - The main purpose of this login and authentication system is to identify the users which interact with the tanks.
     - We also want to provide a level of authentication, this may stop a bad actor with access to an authorized device on the clients warehouse networks.
     - Another possible purpose is to also provide different levels of authorization for users. Even though this is not a requirement for this project, it can be added at a later stage with little effort.
   - **Approach**
     - Create an authentication activity which only navigates to the main activity when a valid login code is submitted.
     - Get the name of the user to display in the top right corner, and store the identifier locally.
   - **Goals**
     - Provide a level of authentication, identification and authorization.
     - Display the users name, and allow the user to log out/switch users.
     - Log out users when inactive for more than 10 minutes.
4. Log
   - **Purpose**
     - The log shows all actions which have affected the tanks.
     - The user can use sorting, filter and search to find the changes they are looking for.
   - **Approach**
     - Display the data inside the table UI.
     - Implement sorting, by category rising and falling.
     - Implement filtering, for
     - Implement smart search functions.
   - **Goals**
     - Display the data correctly, similar to the figma model.

        - Provide tools for users to sort in alphabetical and non alphabetical order for each column in the table, this also includes ascending and descending order for dates and numbers.
        - Provide tools for users to filter the data for different set values, for example acts, statuses, dates etc.
        - Provide a smart search function, which lets users search for all different values in the table.
5. Inventory
    - **Purpose**
        - The inventory shows all tanks which are stored in the clients facility and the tanks which are sent out to the customers of the client.
6. Tank
    - **Purpose**
        - Search for a tank or scan QR-code and get related information
        - Display all info related to a tank
        - Let users perform various acts
            1. Transaction
                a. Link/unlink client
                b. Send to client
                c. Recieve from client
                d. Internal transfer
            2. Maintence
                a. Manage maintence
                b.
                c. Dispose/sell
                d. Manual act
    - **Approach**
        - Create tank fragment, which will be “parent” to various fragments.
        - Implement Tank menu and related functionality
            1. POST -> /transactions
            2. PUT -> /container
            3. Show and hide second row of menu, and change menu-content based on if user choose transaction or maintenance.
            4. Change menu-sub-content for different text and inputs for the 8 different act categories.
        - Implement search, let users search for any value they wish.
        - Implement camera fragment, with QR-scanning functionality.
    - **Goals**
        - Create a similar GUI to the Design prototype with satisfactory result.
        - Scan or search, and open correct tank.
        - Perform acts successfully by updating both transaction and container tables correctly.
7. Tank filling
    - **Purpose**

        - Provide functionality to be able to quickly scan and register fill act to multiple tanks at the same time.
        - Be able to see the tanks which needs to be refilled.
    - **Approach**
        - Create the basis of the UI based on the Figma design prototype.
        - Create a recyclerview of the list of scanned tanks, then add functionality like adding, deleting, etc.
        - Put the updates to the tank table
        - Post the new acts to the act table
        - Make a confirm pop up which allows for double checking before entries are sent.
    - **Goals**
        - Create a similar GUI to the Design prototype with satisfactory results.
        - Make all lists tools work correctly.
        - Put and post data to the backend.
        - Give the users a view of the tanks which needs to be refilled.

Appendix K

Globals test coverage report

backend/globals/functions.go (91.3%) not tracked not covered covered

```go
package globals

import (
	"encoding/json"
	"errors"
	"fmt"
	"math"
	"net/http"
	"sort"
	"strings"
)

/**
 *	Flattens a map slice.
 *
 *	@param mapSlice - The map slice
 *
 *	@return The map slice, flattened.
 */
func flattenMapSlice(mapSlice []map[string]interface{}) map[string][]interface{} {
	// Reorder into format: ["propertyName":[value1, value2, value3]]
	// (And assembly prop query)
	props_values := make(map[string][]interface{})

	var props []string
	for _, kvp := range mapSlice {
		for k, v := range kvp {
			// Check if prop already exists in props
			propExists := false
			for _, prop := range props {
				if prop == k {
					propExists = true
					break
				}
			}
			if !propExists {
				props = append(props, k)
			}

			props_values[k] = append(props_values[k], v)
		}
	}

	return props_values
}

/**
 *	Parses a value, removing the in-built scientific notation of GoLang.
 *	For example: Parsing 123456789 becomes "123456789" instead of 1.23456789e8, and 12345678.9 becomes "12345678.90000" instead of 1.23456789e7
 *	Non-numbers are parsed like normal.
 *
 *	@param any - Value to parse.
 *
 *	@return The parsed value, devoid of scientific notation.
 */
func RemoveScientificNotation(any interface{}) string {
	parsed := ""
	switch any.(type) {
	case float64, float32:
		// Parse as float...
		parsed = fmt.Sprintf("%f", any)

		// ...Then convert the float to int if it doesn't change it's value
		if argVal64, ok := any.(float64); ok {
			if argVal64 == math.Floor(argVal64) {
				parsed = fmt.Sprint(int(argVal64))
			}
		} else if argVal32, ok := any.(float32); ok {
			if float64(argVal32) == math.Floor(float64(argVal32)) {
				parsed = fmt.Sprint(int(argVal64))
			}
		}

	default:
		parsed = fmt.Sprintf("%v", any)
	}

	// Return
	return parsed
}

/**
 *	Finds the origin table of a column, if given by joinData.
 *
 *	@param column - The column name.
 *	@param joinData - The joinData to search through.
 *
 *	@return The table which the column originates from.
 */
func FindOriginTable(column string, joinData map[string][]string) string {
	// Find which table the given field belongs to
	belongsToTable := ""
	for t, tf := range joinData {
		for _, s := range tf {
			if column == s {
				belongsToTable = t
				break
			}
		}
		if belongsToTable != "" {
			break
		}
	}

	// If not found, assume the field belongs to the main table
	if belongsToTable == "" {
		belongsToTable = joinData["main"][0]
	}

	// Return
	return belongsToTable
}

/**
 *	ConvertUrlToSql takes an HTTP request and generates an SQL query with values separated based on the parameters provided.
 *	The SQL query GETS entries from the provided activeTable and its related foreign key tables.
 *
 *	@param r: a pointer to the HTTP request
 *	@param joinData: a map where the key is a string representing the table name and any foreign key references, and the value is a slice containing:
 *		- the name of the table
 *		- the name of the primary key for that table
 *		- any data values requested from that table
 *	@param keys: a slice of strings representing the keys in the joinData map
 *	@param table: The main table which data is being queried from
 *
 *	@returns: a string representing the SQL query with placeholders
 *	@returns: a slice of interface{} containing the values to fit the SQL query
 *	@returns: an error if there are any issues with the request or query construction
 */
func ConvertUrlToSql(r *http.Request, joinData map[string][]string, keys []string) (string, []interface{}, error) {
	var (
		sqlArgs   []interface{}
		sqlSelect string
		sqlJoin   string
	)

	table := joinData["main"][0]

	for _, key := range keys {
		// Extract values from data
		data := joinData[key]
		tableName := data[0]
		primaryKey := data[1]
```

```
                dataIWant := data[2:]

                // Extract target table name from key
                targetTableName := strings.Split(key, ":")[0]

                // Construct SQL JOIN statement
                if key != "main" {
                        sqlJoin += fmt.Sprintf("LEFT JOIN %s ON %s.%s = %s.%s ", targetTableName, tableName, primaryKey, targetTableName, primaryKey)
                }

                // Construct SQL SELECT statement
                for _, val := range dataIWant {
                        if key == "main" {
                                sqlSelect += fmt.Sprintf("%s.%s, ", joinData[key][0], val)
                        } else {
                                sqlSelect += fmt.Sprintf("%s.%s, ", key, val)
                        }
                }
        }

        // Construct SQL WHERE statement
        urlData := r.URL.Query()
        urlDataKeys := make([]string, 0, len(urlData))
        for k := range urlData {
                urlDataKeys = append(urlDataKeys, k)
        }
        sort.Strings(urlDataKeys)

        var queryWhere strings.Builder

        // Declare start- and end date for later
        var (
                startDates []string
                endDates   []string
        )

        // Iterate each key (field name) and value (filter after)
        for _, k := range urlDataKeys {
                v := urlData[k]

                // Save and skip over start- and end date fields
                if k == "start_date" {
                        startDates = v
                        continue
                } else if k == "end_date" {
                        endDates = v
                        continue
                }

                // Find which table the given field belongs to
                belongsToTable := FindOriginTable(k, joinData)

                // If found, add field and table to query string
                if belongsToTable != "" {
                        for _, vd := range v {
                                if queryWhere.String() != "" {
                                        queryWhere.WriteString(" OR")
                                }
                                // Check if the value says to exclude values rather than include
                                if len(vd) > 4 && vd[:4] == "not_" {
                                        vd = vd[4:]
                                        if vd == "null" || vd == "NULL" || vd == "" {
                                                queryWhere.WriteString(fmt.Sprintf(" %s.%s IS NOT NULL", belongsToTable, k))
                                        } else {
                                                queryWhere.WriteString(fmt.Sprintf(" %s.%s != ?", belongsToTable, k))
                                                sqlArgs = append(sqlArgs, vd)
                                        }
                                } else {
                                        if vd == "null" || vd == "NULL" || vd == "" {
                                                queryWhere.WriteString(fmt.Sprintf(" %s.%s IS NULL", belongsToTable, k))
                                        } else {
                                                queryWhere.WriteString(fmt.Sprintf(" %s.%s = ?", belongsToTable, k))
                                                sqlArgs = append(sqlArgs, vd)
                                        }
                                }
                        }
                }
        }

        // Remove trailing comma from SELECT statement
        sqlSelect = sqlSelect[:len(sqlSelect)-2]

        // Combine all SQL statements into one
        SQL := fmt.Sprintf(
                "SELECT %s FROM %s %s ",
                sqlSelect, //what we want
                table,     //what is the main table
                sqlJoin,   //where do we get extra data
        )

        // Append filters to SQL query
        if queryWhere.String() != "" {
                SQL += " WHERE " + queryWhere.String()
        }

        // Append start- and end date to SQL query
        if startDates != nil && endDates != nil {
                // Ensure the "WHERE" part has been added...
                firstWhere := queryWhere.String() == ""

                // ...and add the ranges
                for i, startDate := range startDates {
                        // (Stop if the current startDate doesn't have a corresponding endDate)
                        if i >= len(endDates) {
                                break
                        }

                        endDate := endDates[i]
                        if firstWhere {
                                SQL += " WHERE "
                                firstWhere = false
                        } else {
                                SQL += " OR "
                        }

                        SQL += table + ".date BETWEEN '" + startDate + "' AND '" + endDate + "'"
                }
        }

        return SQL, sqlArgs, nil
}

/**
 *        Takes an http request and returns an SQL query with values sepperate.
 *        The SQL query POSTS entries to the given table.
 *
 *        @param r - a pointer to the http request
 *  @param table - name of the relevant table, (should be added as request header or in the url instead)
 *
 *        @returns - an SQL string with placeholders
 *        @returns - a list of values to fit the SQL query
 *  @returns - any potential errors thrown
 */
func ConvertPostURLToSQL(r *http.Request, table string) (string, []interface{}, error) {
        // Decode body
        var data []map[string]interface{}
        err := json.NewDecoder(r.Body).Decode(&data)
        if err != nil {
                println(data)
                return "", nil, err
        }
        // Get props string and create a sorted list of its keys
        props_values := flattenMapSlice(data)
        props := make([]string, 0, len(props_values))
        for k := range props_values {
                props = append(props, k)
        }
        sort.Strings(props)

        // Iterate props_values in order and append to propsQuery
        var propsQuery strings.Builder
        for i, k := range props {
                if i > 0 {
                        propsQuery.WriteString(",")
                }
                propsQuery.WriteString(k)
```

```
		propsQuery.WriteString(k)
	}

	var args []interface{}

	// Assemble values string
	var valuesQuery strings.Builder
	for i, kvp := range data {
		if i > 0 {
			valuesQuery.WriteString("), (")
		}
		for j, prop := range props {
			if j > 0 {
				valuesQuery.WriteString(",")
			}
			v := kvp[prop]
			if v == nil {
				valuesQuery.WriteString("NULL")
				continue
			}
			args = append(args, fmt.Sprintf("%v", v))
			valuesQuery.WriteString("?")
		}
	}

	// Assemble final query and query it
	query := fmt.Sprintf("INSERT INTO `%s` (%s) VALUES (%s)", table, propsQuery.String(), valuesQuery.String())
	return query, args, nil
}

/**
 *	Takes an http request and returns an SQL query with values sepperate
 *	The SQL query PUTS(updates) entries in the given table.
 *
 *	@param r - a pointer to the http request
 *	@param joinData - a map where the key is a string representing the table name and any foreign key references, and the value is a slice containing:
 *		- the name of the table
 *		- the name of the primary key for that table
 *		- any data values requested from that table
 *	@param keys - a slice of strings representing the keys in the joinData map
 *	@param kwargs - Additional arguments presented as strings:
 *		"alterForeignTables" - Alters foreign table values rather than the main table values whenever possible
 *
 *	@returns - an SQL string with placeholders
 *	@returns - a list of values to fit the SQL query
 *  @returns - any potential errors thrown
 */
func ConvertPutURLToSQL(r *http.Request, joinData map[string][]string, keys []string, kwargs ...string) (string, []interface{}, error) {
	table := joinData["main"][0] // The table which the request is aimed at

	// Additional arguments
	alterForeignTables := false
	for _, kwarg := range kwargs {
		if kwarg == "alterForeignTables" {
			alterForeignTables = true
		}
	}

	// Decode body
	var data []map[string]interface{}

	err := json.NewDecoder(r.Body).Decode(&data)
	if err != nil {
		println("error decode")
		return "", nil, err
	}

	// Get props
	props_values := flattenMapSlice(data)
	props := make([]string, 0, len(props_values))
	for k := range props_values {
		props = append(props, k)
	}
	sort.Strings(props)

	// TODO: Add exception for when NO values are given
	// OR primary_key's value isnt found in the JSON data

	var queryPref strings.Builder
	queryPref.WriteString(fmt.Sprintf("UPDATE %s ", table))

	// Add inner joins
	for _, key := range keys {
		if key == "main" {
			continue
		}

		// Extract values from data
		data := joinData[key]
		tableName := data[0]
		primaryKey := data[1]

		// Extract target table name from key
		targetTableName := strings.Split(key, ":")[0]

		// Construct SQL JOIN statement
		queryPref.WriteString(fmt.Sprintf("LEFT JOIN %s ON %s.%s = %s.%s ", targetTableName, tableName, primaryKey, targetTableName, primaryKey))
	}

	queryPref.WriteString(" SET")

	it := 0
	var args []interface{}
	for _, property := range props {
		if property == "primary" {
			continue
		}

		// Find which table the given field belongs to, if allowed to alter foreign tables
		belongsToTable := table
		if alterForeignTables {
			belongsToTable = FindOriginTable(property, joinData)
		}

		//Ensures that if there is only one type of primary key there wont be an empty update field for that value
		delayedEntry := ""
		if it == 0 {
			delayedEntry = fmt.Sprintf(" %s.`%s` = CASE", belongsToTable, property)
		} else {
			delayedEntry = fmt.Sprintf(", %s.`%s` = CASE", belongsToTable, property)
		}
		prevVal := ""
		for index, val := range props_values["primary"] {
			if propVal, ok := val.(string); ok {
				if prevVal == propVal {
					continue
				}
				prevVal = propVal
				if propVal != property {
					it++
					queryPref.WriteString(delayedEntry)
					queryPref.WriteString(fmt.Sprintf(" WHEN %s.`%s` = ? THEN ?", FindOriginTable(propVal, joinData), propVal))

					// If the args are numbers, don't use scientific connotation
					args = append(args, RemoveScientificNotation(props_values[propVal][index]))
					args = append(args, RemoveScientificNotation(props_values[property][index]))

					if index+1 != len(props_values["primary"]) {
						queryPref.WriteString(fmt.Sprintf(" ELSE `%s`", property))
					}
					queryPref.WriteString(" END")
				}
			} else {
				println("error prop value")

				return "", nil, errors.New("error asserting props_values as string")
			}
		}
	}

	for p, property := range props_values["primary"] {
		if propVal, ok := property.(string); ok {
			belongsToTable := FindOriginTable(propVal, joinData)

			if p == 0 {
				queryPref.WriteString(fmt.Sprintf(" WHERE %s.`%s` = '%s'", belongsToTable, propVal, RemoveScientificNotation(props_values[propVal][p])))
			} else {
```

```
                        } else {
                                queryPref.WriteString(fmt.Sprintf(" OR %s.`%s` = '%s'", belongsToTable, propVal, RemoveScientificNotation(props_values[propVal][p])))
                        }
                } else {

                        println("error prop value as string")
                        return "", nil, errors.New("error asserting props_values as string")
                }
        }

        // Return
        queryPref.WriteString(";")
        return queryPref.String(), args, nil
}

/**
 *        Takes an http request and returns an SQL query with values sepperate
 *        The SQL query DELETES entries in the given table.
 *
 *        @param r - a pointer to the http request
 *        @param joinData - a map where the key is a string representing the table name and any foreign key references, and the value is a slice containing:
 *                - the name of the table
 *                - the name of the primary key for that table
 *                - any data values requested from that table
 *        @param keys - a slice of strings representing the keys in the joinData map
 *
 *        @returns - an SQL string with placeholders
 *        @returns - a list of values to fit the SQL query
 *  @returns - any potential errors thrown
 */
func ConvertDeleteURLToSQL(r *http.Request, joinData map[string][]string, keys []string) (string, []interface{}, error) {

        table := joinData["main"][0] // The table which the request is aimed at

        // DELETE statement
        query := fmt.Sprintf("DELETE P FROM %s P", table)
        var args []interface{}

        // JOIN statement
        for _, key := range keys {
                if key == "main" {
                        continue
                }

                // Extract values from data
                data := joinData[key]
                tableName := data[0]
                primaryKey := data[1]

                // Use P rather than the main table name
                if tableName == table {
                        tableName = "P"
                }

                // Extract target table name from key
                targetTableName := strings.Split(key, ":")[0]

                // Add left join
                query += fmt.Sprintf(
                        "\nLEFT JOIN %s ON %s.%s = %s.%s",
                        targetTableName,
                        tableName,
                        primaryKey,
                        targetTableName,
                        primaryKey,
                )
        }

        // Set up start- and end date variables for later
        var (
                startDates []string
                endDates   []string
        )

        // Iterate query keys- and values
        urlQuery := r.URL.Query()
        urlQueryKeys := make([]string, 0, len(urlQuery))
        for k := range urlQuery {
                urlQueryKeys = append(urlQueryKeys, k)
        }
        sort.Strings(urlQueryKeys)

        var queryWhere strings.Builder
        for _, k := range urlQueryKeys {
                v := urlQuery[k]
                // Save and skip over start- and end date fields
                if k == "start_date" {
                        startDates = v
                        continue
                } else if k == "end_date" {
                        endDates = v
                        continue
                }

                // Find which table the given field belongs to
                belongsToTable := FindOriginTable(k, joinData)
                if belongsToTable == table {
                        belongsToTable = "P"
                }

                // If found, add field and table to query string
                if belongsToTable != "" {
                        for _, vd := range v {
                                if queryWhere.String() != "" {
                                        queryWhere.WriteString(" OR")
                                }
                                queryWhere.WriteString(fmt.Sprintf("\n\t%s.%s = ?", belongsToTable, k))
                                args = append(args, vd)
                        }
                }
        }

        // Add WHERE statement
        if queryWhere.String() != "" {
                query += "\nWHERE" + queryWhere.String()
        }

        // Append start- and end date to SQL query
        if startDates != nil && endDates != nil {
                // Ensure the "WHERE" part has been added...
                firstWhere := queryWhere.String() == ""

                // ...and add the ranges
                for i, startDate := range startDates {
                        // (Stop if the current startDate doesn't have a corresponding endDate)
                        if i >= len(endDates) {
                                break
                        }

                        endDate := endDates[i]
                        if firstWhere {
                                query += "\nWHERE"
                                firstWhere = false
                        } else {
                                query += " OR"
                        }

                        query += "\n\tP.date BETWEEN '" + startDate + "' AND '" + endDate + "'"
                }
        }

        // Return
        return query, args, nil
}
```

Appendix L

File tree backend

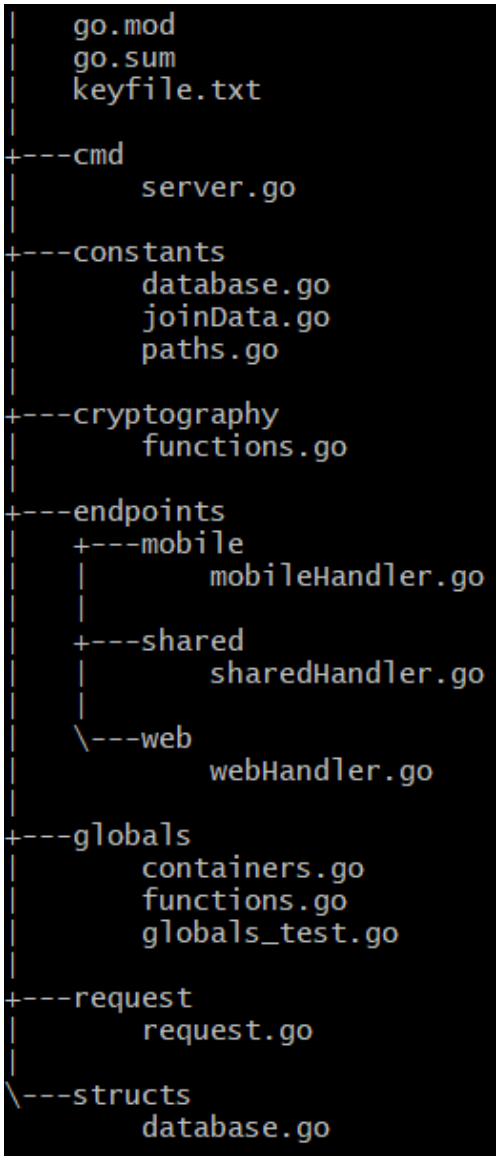


Figure L.1: File tree backend

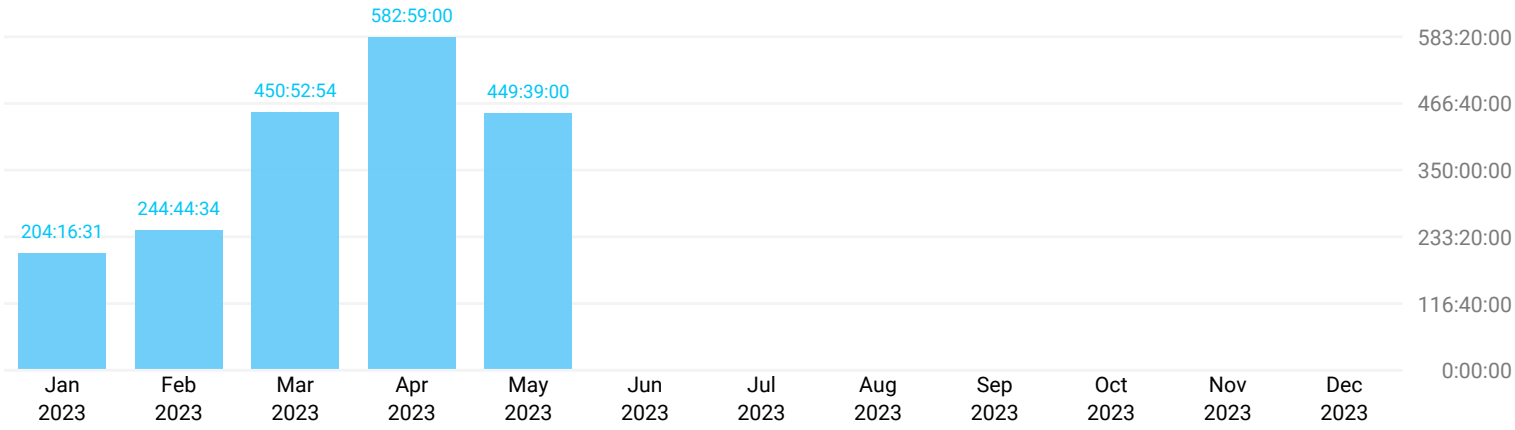
Appendix M

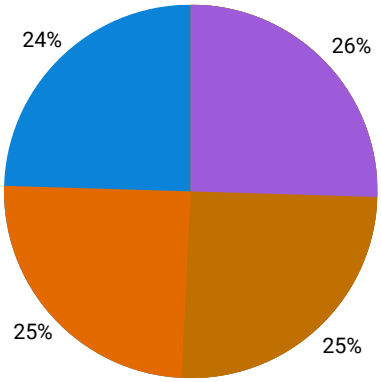
Time tracking

Summary Report

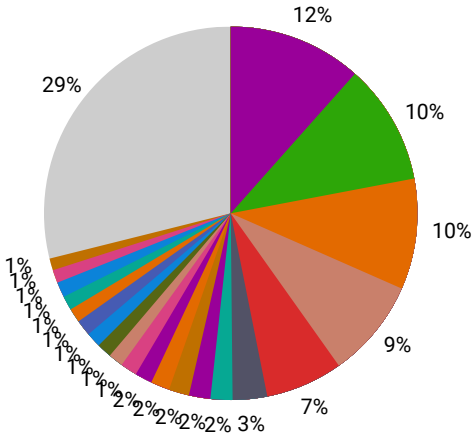
01/01/2023 – 12/31/2023

TOTAL HOURS: 1932:31:59

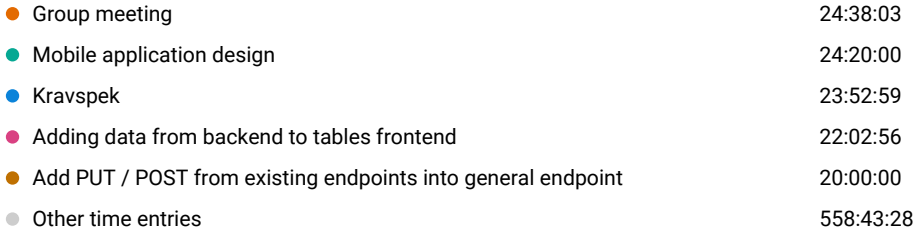




USER	DURATION
Havard_ski	493:22:24
Mats Greeven	489:45:38
Axel Jacobsen	476:31:46
Lars L R	472:52:11



TIME ENTRY	DURATION
Thesis	224:15:00
Android application development	200:24:55
Writing thesis	186:35:00
thesis work	166:00:00
backend	129:18:46
mobile application	55:30:00
Last design revison	38:03:44
Icons	34:22:46
Backend	34:12:00
GET calls from Backend to Web	32:26:52
meeting	29:35:07
Website HTTP calls	26:30:00
Project Plan	25:55:14
Web draft	25:21:19
Meeting	25:13:00
Low fidelity design website	25:10:50



USER - TIME ENTRY	DURATION	PERCENTAGE
AJ Axel Jacobsen	476:31:46	24.66%
API requests	10:35:00	0.55%
Backend	34:12:00	1.77%
Backend dummy	2:37:00	0.14%
Backend dummy data	8:00:00	0.41%
Backend planning	6:12:32	0.32%
Backend SQL	6:00:00	0.31%
First meeting	2:00:00	0.1%
Fixed CORS errors	2:15:00	0.12%
Icons	34:22:46	1.78%
Inventory fragment	10:15:00	0.53%
Kravspek	21:18:13	1.1%
Lecture	1:30:00	0.08%
Logo & Meeting	5:43:00	0.3%

USER - TIME ENTRY	DURATION	PERCENTAGE
Low fidelity	3:30:00	0.18%
Lynkurs	1:45:00	0.09%
Meeting	24:13:00	1.25%
Meeting with advisor	5:30:00	0.28%
Meeting with client	0:30:00	0.03%
Meeting with Cryogenetics	5:00:00	0.26%
Mobile bugs	14:00:00	0.72%
Project Plan	10:42:15	0.55%
PUT request on backend	19:05:00	0.99%
Questionaire	1:31:00	0.08%
Status report	6:05:00	0.31%
Thesis	224:15:00	11.6%
Url To sql convertion	15:25:00	0.8%
Havard_ski	493:22:24	25.53%
Android app Sw Dev Plan	4:45:00	0.25%
Android application development	200:24:55	10.37%
Application design	11:30:15	0.6%

USER - TIME ENTRY	DURATION	PERCENTAGE
Container menu	12:15:00	0.63%
Designing on figma	2:00:00	0.1%
Discussion	0:30:00	0.03%
Driving to Cryogenetics	2:10:00	0.11%
Final moments	4:12:00	0.22%
Group meeting, work session	1:30:00	0.08%
Inventory design	2:00:00	0.1%
Kravspek - Operational Requirements	1:30:00	0.08%
Last desgin revison	3:54:00	0.2%
Last design revison	38:03:44	1.97%
Lynkurs bachlor oppgave	1:45:00	0.09%
Meeting	1:00:00	0.05%
Meeting about design	1:00:00	0.05%
Meeting notes	0:40:00	0.03%
Meeting with counselor Frode	6:30:00	0.34%
Meeting with Cryogenetics	5:30:00	0.28%
Meeting with steffen from Cryo	1:30:00	0.08%

USER - TIME ENTRY	DURATION	PERCENTAGE
Meeting, work session	1:15:00	0.06%
Mobile application design	24:20:00	1.26%
Project design	1:00:00	0.05%
Project plan	0:30:00	0.03%
Project plan revision	4:00:00	0.21%
Project plan revision meeting	1:20:00	0.07%
Project plan, capter 2 - objectives	2:00:00	0.1%
Scrumban meeting	16:45:00	0.87%
Sequence diagram	6:52:30	0.36%
Tank filling menu	6:00:00	0.31%
Transaction log design	2:30:00	0.13%
Travel from Cryogenetics	1:10:00	0.06%
Travel to Cryogenetics	1:10:00	0.06%
User friendliness	0:30:00	0.03%
User testing at Cryogenetics	4:30:00	0.23%
Work session	0:30:00	0.03%
Working with high fidelity design	11:00:00	0.57%

USER - TIME ENTRY	DURATION	PERCENTAGE
working with high fidelity design	10:00:00	0.52%
Working with low fidelity design	10:15:00	0.53%
Writing Kravspek	5:15:00	0.27%
Writing meeting notes	4:15:00	0.22%
Writing Statusrapport	3:00:00	0.16%
Writing thesis	70:35:00	3.65%
XML designs	2:00:00	0.1%
LL Lars L R	472:52:11	24.47%
add POST to backend	3:00:00	0.16%
add safe POST to backend and fix a few things	1:53:22	0.1%
add safe PUT to backend	2:00:00	0.1%
backend	129:18:46	6.69%
backend dummy	1:00:00	0.05%
conceptual model	8:00:00	0.41%
domain model	7:15:00	0.38%
finish group contract & chapter 3	2:30:00	0.13%
gantt	2:00:00	0.1%

USER - TIME ENTRY	DURATION	PERCENTAGE
group contract	1:00:00	0.05%
kravspek planning	7:00:00	0.36%
maintain database	4:00:00	0.21%
meeting	29:35:07	1.53%
meeting with cryogenetics	5:15:00	0.27%
meeting with frode	2:00:00	0.1%
misuse case	2:33:35	0.13%
mobile application	55:30:00	2.87%
physical database draft	15:15:00	0.79%
product backlog & domain model desc	1:30:44	0.08%
redesign endpoints	2:00:00	0.1%
revision after meeting with frode	7:56:28	0.41%
rewrite kravspek to future-tense	0:47:04	0.04%
status report	1:58:41	0.1%
technical requirements	1:48:24	0.09%
thesis work	166:00:00	8.59%
update conceptual and physical model	7:15:00	0.38%

USER - TIME ENTRY	DURATION	PERCENTAGE
usertesting	4:30:00	0.23%
Mats Greeven	489:45:38	25.34%
Add PUT / POST from existing endpoints into general endpoint	20:00:00	1.03%
Adding data from backend to tables frontend	22:02:56	1.14%
Brukertesting	1:20:00	0.07%
Bug fixes	5:15:00	0.27%
Client Meeting	10:15:04	0.53%
Client meeting	0:30:00	0.03%
Code cleanup	0:20:28	0.02%
Councillor meeting	5:33:01	0.29%
DELETE calls from webstite	6:00:00	0.31%
endpoint documentation	4:28:16	0.23%
Endpoints Admin website	2:20:00	0.12%
Final design revision	6:30:00	0.34%
Fix searching on web app	9:48:10	0.51%
Fix sorting on web app	9:54:16	0.51%
Fixing data to send	6:45:00	0.35%

USER - TIME ENTRY	DURATION	PERCENTAGE
Generate Report	12:45:00	0.66%
GET calls from Backend to Web	32:26:52	1.68%
Group meeting	24:38:03	1.27%
Initial meeting for Project Plan setup	1:30:00	0.08%
Kravspek	2:34:46	0.13%
Low fidelity design	1:20:01	0.07%
Low fidelity design App	2:33:07	0.13%
Low fidelity design website	25:10:50	1.3%
Modify SQL translator	11:00:00	0.57%
PROG2900 Seminar Lynkurs prosjektstyring	1:45:00	0.09%
Project Plan	15:12:59	0.79%
QR Code generation	8:00:00	0.41%
rework endpoints	13:00:00	0.67%
Rework website tables with new data	9:30:00	0.49%
Search to frontend	0:16:26	0.01%
Setting up Toggl	0:11:15	0.01%
Use case	10:37:03	0.55%

USER - TIME ENTRY	DURATION	PERCENTAGE
Web draft	25:21:19	1.31%
Web draft (adding Tables)	7:05:46	0.37%
Website HTTP calls	26:30:00	1.37%
Website POST	14:45:00	0.76%
Website PUT	9:00:00	0.47%
Work with Azure to figure out deployment	7:30:00	0.39%
Writing thesis	116:00:00	6.0%