

Duggal, Simon  
Josefsen, Johannes Løvold  
Lindgård, Hans Andreas

## Ship Organizer

Inventory management onboard

Bachelor's thesis in Engineering in Computer Science

Supervisor: Mikael Tollefsen

May 2022



Duggal, Simon  
Josefsen, Johannes Løvold  
Lindgård, Hans Andreas

## **Ship Organizer**

Inventory management onboard

Bachelor's thesis in Engineering in Computer Science  
Supervisor: Mikael Tollefsen  
May 2022

Norwegian University of Science and Technology  
Faculty of Information Technology and Electrical Engineering  
Department of ICT and Natural Sciences



---

## Abstract

Giske Kystfiske was pursuing a way to simplify their inventory management system. Having experienced difficulties with their current system, Giske Kystfiske requested the development of a solution tailored to their problem. The team found this problem interesting, and contacted them about wanting to solve their problem.

Onboard a fishing boat it can be hard to keep track of the inventory. Typical solutions today include simple Excel spreadsheets available on stationary computers around the boat. The usage of spreadsheets makes the tracking of details such as when and where an item was used more cumbersome. The current solution is slow and requires the employees to relocate to track changes. This makes it easy to forget the amount used, or forget to register the use at all, making accurately ordering new stock difficult.

During the project, the team followed the agile development methodology. Weekly sprints kept the team organized, and the project on track. The agile methodology made the team adaptable to changes in the problem domain, which proved advantageous for completing the product on time.

The product developed is a complete solution for the inventory tracking problems. The cross-platform mobile application helps track inventory and item usage. It is designed with an industrial environment in mind, with focus on efficiency and ease of use. The application is developed in the Flutter framework and written in Dart.

As of May 1st 2022 the application, named "SeaStorage", is publicly available on Google Play Store for Android users, and through the Apple TestFlight system for Apple users. The backend server handling the business logic is developed using Java's Spring Boot framework, with a MySQL database. It has been deployed using Docker on the cloud-based hosting service Digital Ocean.

Giske Kystfiske is very satisfied with the product they have received. Since May 1st the client has used the application in its entirety to actively manage their inventory onboard. The application has simplified recording what equipment is used, and made it easy to see where, when and who used what.

---

## Sammendrag

Giske Kystfiske var på utkikk etter en måte å forenkle lagerstyringsystemet deres på. Etter å ha opplevd vanskeligheter med sitt nåværende system, ønsket Giske Kystfiske at noen skulle utvikle en løsning tilpasset deres problem. Teamet tenkte at dette problemet var interessant, og kontaktet dem for å hjelpe med å løse problemet.

Ombord en fiskebåt kan det være vanskelig å holde styr på beholdningen. Typiske løsninger i dag inkluderer enkle Excel-regneark tilgjengelig på stasjonære datamaskiner rundt om i båten. Regnearkene registrerer kun nåværende mengde, og mangler for eksempel når og hvor det ble brukt, og hvem det ble brukt av. Dagens løsning er treg og fører lett til feilhåndtering av varelageret. Ansatte redigerer regnearket for feil produkter, og legger inn feil mengde, noe som gjør nøyaktig bestilling av ny beholning vanskelig.

I løpet av prosjektet har teamet jobbet etter den smidige utviklingsmetodikken. Ukentlige sprinter holdt teamet organisert, og arbeidet i rute. Å følge den smidige utviklingsmetodikken gjorde at teamet kunne tilpasse seg endringer i problemdomenet i løpet av prosjektet, noe som viste seg å være fordelaktig for å fullføre produktet i tide.

Løsningen utviklet i dette prosjektet er en komplett løsning for de organisatoriske problemene. Kryssplattform-mobilapplikasjonen hjelper med å holde orden på inventar og varebruk. Den er designet for bruk i et industrielt arbeidsmiljø, med fokus på effektivitet og brukervennlighet. Appen er utviklet ved hjelp av Dart språkets Flutter-rammeverk.

Fra 1. mai 2022 er appen, kalt "SeaStorage", offentlig tilgjengelig på Google Play Store for Android-brukere, og gjennom Apple TestFlight-systemet for Apple-brukere. Backend-serveren som håndterer forretningslogikken er utviklet ved hjelp av Javas Spring Boot-rammeverk. Den har blitt distribuert ved hjelp av Docker på en Digital Ocean Droplet.

Giske Kystfiske, oppdragsgiveren, som hadde bedt om en løsning på deres lagerstyringsproblem, er veldig fornøyd med sluttproduktet. Fra 1. mai har kunden brukt appen i sin helhet for å aktivt administrere beholdningen ombord. Appen har forenklet registrering av hvilket utstyr som brukes, og gjort det enkelt å se hvor, når, og av hvem utstyr har blitt brukt.

---

## **Preface**

### **About**

This project was chosen as developing and publishing an app seemed like a real challenge to the team. It was interesting to get insight into the process of publishing an app and getting it publicly available. The team worked in three main phases, planning, creating the app and server, and finally publishing.

20.05.2022 Ålesund

Simon Duggal

Johannes Løvold Josefsen

Hans Andreas Lindgård

### **Thanks to**

The team would like to express their gratitude towards:

- The client, Kurt Skjong of Giske Kystfiske, for being readily available to answer questions and give feedback on the app.
- Our supervisor, Mikael tollefsen. Thanks for good advice on what to focus on in the app, helping with various problems, and guiding us through writing the report.

---

## **Assignment text**

The assignment text can be found in appendix D. The more detailed and developed requirements which were worked towards are documented in appendix C.



# Contents

<b>Abstract</b>	<b>i</b>
<b>Sammendrag</b>	<b>ii</b>
<b>Preface</b>	<b>iii</b>
About . . . . .	iii
Thanks to . . . . .	iii
<b>Assignment text</b>	<b>iv</b>
<b>1 Introduction and relevance</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Problem . . . . .	1
1.3 Long term effects . . . . .	2
1.3.1 Product vision . . . . .	2
1.4 Limitations . . . . .	2
1.5 Report structure . . . . .	2
1.6 Acronyms and jargon . . . . .	4
1.6.1 Acronyms . . . . .	4
1.6.2 Jargon . . . . .	4
<b>2 Theory, and materials</b>	<b>5</b>
2.1 Theory . . . . .	5
2.1.1 Object-Oriented Programming . . . . .	5
2.1.2 Design patterns . . . . .	5
2.1.3 Quality assurance . . . . .	6
2.1.4 Development . . . . .	7
2.1.5 Relational database . . . . .	8
2.1.6 Queue data structure . . . . .	8
2.1.7 Race conditions . . . . .	8
2.1.8 Containerization . . . . .	8
2.1.9 Domain specific theory . . . . .	9
2.1.10 Security . . . . .	9

<b>3 Method</b>	<b>10</b>
3.1 Planning and Design Process	10
3.1.1 Pre-project plan	10
3.1.2 Meeting with client	10
3.1.3 Research and deciding on technologies	10
3.1.4 Design	11
3.2 Equipment, tools, and technology	11
3.2.1 Flutter	11
3.2.2 Spring Boot	12
3.2.3 Authentication	12
3.2.4 MySQL Database	12
3.2.5 Digital Ocean Spaces	12
3.2.6 Docker	13
3.2.7 Collaboration tools	13
3.2.8 Communication	14
3.3 Agile development	14
3.3.1 Roles and work distribution	14
3.3.2 Sprints	14
3.4 Development Process	15
3.4.1 User Experience First	15
3.4.2 Flutter Application	15
3.4.3 Server and Database	15
3.4.4 Quality Assurance	16
3.5 Testing	17
3.5.1 Process for user testing	17
<b>4 Results</b>	<b>18</b>
4.1 Scientific results	18
4.1.1 Network in a maritime setting	18
4.1.2 Principles of design	20
4.2 Engineering results	20
4.3 Engineering results - Mobile app	21
4.3.1 Logging in and staying logged in	21
4.3.2 Updating inventory	21
4.3.3 Language settings	22
4.3.4 Map integration	23
4.3.5 Scanning barcodes	25
4.3.6 Phone and tablet	26
4.3.7 Registering users	26
4.3.8 Missing inventory	27
4.3.9 Data handling	27
4.3.10 Order confirmation	28
4.3.11 Error handling	28
4.3.12 Deployment and testing	29

## CONTENTS

---

4.4	Engineering results - Server application . . . . .	30
4.4.1	Controllers . . . . .	30
4.4.2	Services . . . . .	30
4.4.3	Models . . . . .	31
4.4.4	Repositories . . . . .	31
4.4.5	Image upload . . . . .	31
4.4.6	Database . . . . .	31
4.4.7	Security . . . . .	32
4.4.8	Future deployment of server . . . . .	33
4.5	Administrative results . . . . .	33
4.5.1	Working in Jira . . . . .	33
4.5.2	Development overview . . . . .	33
<b>5</b>	<b>Discussion</b>	<b>36</b>
5.1	Scientific discussion . . . . .	36
5.1.1	Reducing data traffic . . . . .	36
5.1.2	Ease of use . . . . .	37
5.2	Mobile application . . . . .	37
5.2.1	Solving the clients problem . . . . .	37
5.2.2	Flutter framework . . . . .	38
5.2.3	Deployment to client devices . . . . .	38
5.2.4	Handling offline scenarios . . . . .	40
5.2.5	Handling missing GPS signal . . . . .	40
5.2.6	Requirements . . . . .	41
5.3	Server application . . . . .	41
5.3.1	Spring Boot . . . . .	41
5.3.2	Swapping SQL dialect . . . . .	42
5.3.3	Stored Procedures . . . . .	42
5.3.4	Deployment of server and database . . . . .	42
5.4	Administrative discussion . . . . .	43
5.4.1	Communication . . . . .	43
5.4.2	Development process . . . . .	44
5.4.3	Collaboration with Jira and Confluence . . . . .	45
<b>6</b>	<b>Conclusion and further work</b>	<b>46</b>
6.1	Conclusion . . . . .	46
6.2	Further work . . . . .	47
	<b>Societal impact</b>	<b>48</b>
<b>A</b>	<b>Project Plan</b>	<b>53</b>
<b>B</b>	<b>System Documentation</b>	<b>63</b>
<b>C</b>	<b>Requirements Documentation</b>	<b>73</b>

*CONTENTS*

---

<b>D Project description</b>	<b>98</b>
<b>E Emails</b>	<b>103</b>
<b>F Project Manual</b>	<b>105</b>
<b>G Repository</b>	<b>106</b>

# List of Figures

- 4.1 Inventory view . . . . . 22
- 4.2 My account view . . . . . 22
- 4.3 Map view . . . . . 25
- 4.4 Barcode scanning . . . . . 25
- 4.5 My account view on phone and tablet . . . . . 26
- 4.6 Missing inventory view . . . . . 27
  
- E.1 Feedback from client on final product . . . . . 104
- E.2 Feedback from client on wireframes . . . . . 104

# List of Code examples

4.1	The addtoQueue and startService implementations. ( <i>OfflineEnqueue-Service.dart</i> ) .....	19
4.2	Sorting map markers into a grid ( <i>ReportService.java</i> ) .....	24
4.3	Example of an async call for trying to delete a user ( <i>api_controller.dart</i> ) . . . .	28
4.4	Handling error codes received after trying to register a user. ( <i>api_controller.dart</i> )	29
4.5	Handling request data for editing user. ( <i>UserController.java</i> ) .....	30
4.6	Handling the user editing functionality. ( <i>UserService.java</i> ) .....	30
4.7	Defining the query to call the procedure for editing a user. ( <i>UserRepository.java</i> )	31
4.8	Code taken from the HandleUser Procedure. Displaying the use of Calltime . .	32
4.9	The start of the security configuration method restricting access to endpoints. ( <i>SecurityConfig.java</i> ) .....	32

# **1. Introduction and relevance**

## **1.1 Background**

Giske Kystfiske is searching for a way to simplify their storage and logistics management on board their ship. They experience difficulties keeping control of their inventory between shifts, and see a mobile application as a possible solution to this problem.

Currently, the client is using a desktop computer, outfitted with Excel, located in the office on board the boat. When the employees use equipment they have to walk to the office, find the equipment they have used in their Excel spreadsheets, and then change the amount of that equipment they have on board. The client has said that this system works poorly, employees struggle to find the right equipment in the spreadsheets, or forget to register what they have used. It is also very time consuming. A mobile application could simplify and speed up the administration process of their storage.

The client also states that their employees have a varying degree of digital competence, and that therefore, the application should be easy to use.

## **1.2 Problem**

The ship has several departments, for example bridge, factory, office, etc. Each with their own stock of different equipment and components. When an employee removes something from their department's inventory it needs to be kept track of in a way that everyone else working in this department also can see the changes.

When the trip is completed an order for the missing inventory needs to be placed, so that the ship's department inventories can be restocked.

When a shipment arrives at the dock, the workers at the different departments need to check that the received products match with the order bill. The workers then need to notify the office department if the order bill is correct, so that they can complete the payment.

## **1.3 Long term effects**

The solution aims to have a positive impact towards increased efficiency for inventory management. This can decrease the time spent by employees on inventory management to free up time to spend on other tasks. By having a more accurate overview of their inventory the client is able to order equipment more accurately and therefore reduce costs.

### **1.3.1 Product vision**

The product vision is that the application will increase efficiency and streamline the inventory management process on board. By simplifying the sharing of documents between departments and office, invoices can be confirmed faster. The product will also enable quickly sending an overview of the missing inventory to the office so they can order the parts necessary for the next trip.

## **1.4 Limitations**

There was a high degree of freedom when working on the project, considering the design and technical solution of the project. There were however still some limitations that needed to be taken into consideration when starting the planning and development process. The application needed to work on iOS and Android. The application also needed to function as uninhibited as possible even when offline.

## **1.5 Report structure**

This report is split into 6 overarching parts.

### **Chapter 1 Introduction**

The introduction gives a brief overview of the problem domain and the solution envisioned at the start of the project.

### **Chapter 2 Theory and Materials**

Theory and materials describes the theoretical background for the project, it includes definitions of technologies and other explanations necessary to understand the report.

### **Chapter 3 Method**

This chapter goes through the varying choices made during the project and justifications for choosing them.



#### **Chapter 4 Results**

The fourth chapter goes through the results of the project. It is split into three smaller parts: scientific results, engineering results, and administrative results.

#### **Chapter 5 Discussion**

This chapter discusses why the results became the what they did, and how they relate to the initial problem.

#### **Chapter 6 Conclusion and further work**

The conclusions which can be drawn from the issues presented in the introduction in relation to the results is shown in this chapter. It also discusses future work, and recommendations for what should be prioritized if someone was to continue development.

## 1.6 Acronyms and jargon

### 1.6.1 Acronyms

**API** Application Programming Interface

**FIFO** First-in-first-out

**GPS** Global Positioning System

**GUI** Graphical User Interface

**iOS** iPhone Operating System

**JPA** Java Persistence API

**JSON** JavaScript Object Notation

**JWT** JSON Web Token

**MVP** Minimum Viable Product

**REST** Representational State Transfer

**SQL** Structured Query Language

**TDD** Test-Driven Development

**UX** User Experience

### 1.6.2 Jargon

**App** Mobile application, for both tablets and phones

**Backend** Refers to the software stack handling business logic and data storage. Not directly accessible by the user.

**Frontend** Refers to the software stack which the user interacts with. The GUI.

## **2. Theory, and materials**

### **2.1 Theory**

#### **2.1.1 Object-Oriented Programming**

Object-oriented programming is a programming paradigm in which the base concepts revolve around creating code modules which represent objects. Objects in code contain data and methods to give the object its desired behavior. [1]

##### **Coupling and cohesion**

Coupling and Cohesion are two central aspects for writing high-quality code in an object-oriented programming language. Writing high-quality code entails writing code where changing aspects of the program doesn't require unnecessary work. This can be achieved by writing code which has high cohesion and low coupling. [2]

##### **Coupling**

Coupling is defined by the degree of which different components are dependant on one another. It describes relationships between components.

##### **Cohesion**

Cohesion is defined by the degree of which elements of a component are functionally related. It describes relationships within components. [3]

#### **2.1.2 Design patterns**

Design patterns are formalized best practices that can be used to solve common problems when creating a system.[4]

### **Observer and observable pattern**

In the observer pattern an object, known as the observable, contains a collection of objects, known as observers. The observable notifies all of its observers whenever there is an event the observers are subscribed to. This is done by calling a specified notification method of the observers. [5]

### **Singleton pattern**

Singleton is a design pattern which ensures that only one object of its kind can exist. Furthermore it provides a single point of access to it for any other point in the code. [6]

## **2.1.3 Quality assurance**

### **Unit tests**

Unit tests is a software development process which each of the smallest testable parts of software are individually and independently scrutinized to determine if they are fit for use. Doing unit testing ensures that the code, at a base level, functions as intended. [7]

### **Code review**

Code review is the process of consciously and systematically checking a fellow team members code. It is done to check that the code written contains no mistakes, and that it is high quality. Code review has been repeatedly shown to accelerate and streamline the development process. [8]

### **Universal design**

Universal design is the practice of designing in a way that the design is accessible for everyone, regardless of age, ability, and disability. [9]

### **Principles of design**

The principles of design are rules that a designer should follow if they want to create a design that can deliver a message in the most organized and functional way. There are multiple recognized principles for design, but the main ones include; balance, contrast, hierarchy, proportion, and white space[10]. Good GUI design does not necessarily implement all the principles of design, often multiple are ignored, but using the principles of design as guidelines can help create good GUI and UX.

### **DRY-principle**

DRY stands for Don't Repeat Yourself, which is used as a software development principle in which the developer should avoid writing duplicate pieces of code. Duplication of code can lead to a lot of work when a change is made, as it can lead to having to implement that same change in multiple places. [11]

### **Localization**

Localization is the process of modifying a product to better fit another country or region. Translating text is one of the key components for localization. Outside of simply translating text the localization process can also include adapting design to correctly display translated text, and changing date-time format. [12]

## **2.1.4 Development**

### **Agile development**

Agile development is an iterative approach to large projects such as software development. It is an umbrella term for frameworks and practices based on the principles presented in the Manifesto for Agile software development. Agile development means delivering work in small incremental pieces. Working in agile development makes the team more adaptable to changes in the product requirements and other eventual issues with the final product. [13]

### **Git**

Git is an open-source software used to track changes to any set of files. It is used to coordinate work between developers, making collaboration on the same source-code easier. [14]

### **RESTful web services**

A RESTful web service is a web service which follows the principles of the REST architecture. [15]

- **Resource identification through URI** Resources are identifiable through URIs, normally links on the Web.
- **Uniform interface** Manipulation of resources through a set of operations, GET, PUT, POST, and DELETE HTTP-methods.
- **Self-descriptive messages** The received message contains all information needed to understand its contents.
- **Stateful interactions through hyperlinks** As every interaction with the API is stateless, the required information about the current state must be included in the request.

## **Cloud services**

Cloud services are infrastructure and services hosted by third-party providers which are made accessible through the internet. Cloud services are designed to provide easy and affordable access to applications and resources, without an internal requirement for infrastructure and hardware. There are multiple types of cloud services, Infrastructure-as-a-service, Platforms-as-a-service, Software-as-a-service and Function-as-a-service. For example, hosting a server for a mobile application is Infrastructure-as-a-service. Using cloud services enables running an application without needing to set up and maintain a physical server for the client. [16]

### **2.1.5 Relational database**

A relational database is a database that stores and provides access to data that are connected to one another. In a relational database, each row in a table has a unique ID, the primary key. The columns of a table hold the attributes of the data, each record usually has a value for each of the attributes. This makes it easy to establish relationships between data points. In a relational database an attribute of one data-point could link to another data-point entirely. For example one data-point could represent a car and one of its attributes could be driver which links to another table with a data-point representing a driver, that also has a set of attributes. [17]

### **2.1.6 Queue data structure**

The queue data structure is an ordered list following the FIFO principle. When items are en-queued, they are inserted at the rear of the queue. When the queue is accessed, it is only possible to access and remove the item at the front of the queue. [18]

### **2.1.7 Race conditions**

A race condition is a condition that occurs when the system is dependent on the sequence of timed, or uncontrollable events. This means that the result of a process can vary depending on when other, often concurrent, processes complete. [19]

### **2.1.8 Containerization**

Containerization is a way of packing software code along with only the operating system and dependencies needed for it to run. This creates a portable lightweight software package which can be easily deployed on a large amount of systems. [20]

## 2.1.9 Domain specific theory

### Barcodes

Barcodes are a method for representing data visually, in a machine-readable form. The barcodes used in this project represent data using parallel lines with varying widths and spacing. Barcodes in this form are commonly used as a means for quickly identifying products. [21]

### Latitude and longitude

Latitude and longitude are a geographic coordinate system which is used to accurately determine any position on the earth's surface. Latitude ranges from 0 to 90 degrees north or south, whilst longitude ranges from 0 to 180 degrees east or west. Each decimal point after the initial degrees indicate an increase in accuracy of the position. For common use, 5 decimals, accuracy within  $1.11m$ , is enough. [22]

## 2.1.10 Security

### Authentication and authorization

Authentication and authorization are security measures used to control and secure access. Authentication is used so that it is known who is accessing the resource, and authorization is used to determine who has access to the specified resource. [23]

### Hashing

Hashing is a process in which a specified key is converted into another value, called a hash, using a hashing algorithm. Given the same key, the algorithm will produce the same value. A good hashing algorithm is one where the value cannot be converted back into the original key. [24]

### Token

A token is a string or object containing information or credentials, often for authentication and authorization of a user. [25]

## **3. Method**

### **3.1 Planning and Design Process**

The team focused on thorough planning, and preparation during the start of the project.

#### **3.1.1 Pre-project plan**

The team prepared a pre-project plan to have some planned paths, guidelines, and deadlines, to follow throughout the development process. The team created a road map containing important milestones and deadlines. It was decided that there should be early deadlines for some of the milestones to push the team to work hard from the start. The details of the pre-project plan can be found in attachment A.

#### **3.1.2 Meeting with client**

Meetings were held with the client to clarify important decisions surrounding the requirements and design of the solution. Uncertainties from the team about how and where the client wanted to use the solution, were discussed in these meetings.

#### **3.1.3 Research and deciding on technologies**

A part of the planning process was trying to foresee which technologies would work the best for the app, server, and database. Initial team meetings were used to research and plan what technologies best suited the team for development, within the given time frame. The strict time frame influenced the choice of technologies. With little time to work with, the team had programming languages and frameworks that the members were already familiar with as forerunners for the project. The main deciding factor was what technologies best suited solving the problem presented by the client. As the application was to be used exclusively on iOS and Android, the team decided on using Flutter for developing the frontend.



### 3.1.4 Design

Making wireframes was the initial step in the design process. The team collaborated on the wireframes using Mockplus. Mockplus is an online wireframe editor for phones and tablets [26]. The team chose to use Mockplus as it was free, online and in the browser, which made it easy to collaborate on the wireframes. The focus was on creating a design which was easy to use. The principles of design<sup>1</sup> were useful guidelines which aided in creating a clean and intuitive GUI.

The app's area of use is on a fishing boat operating in varying conditions, therefore the design focused on having components which were large and contrasted with other components<sup>2</sup>. The color palette was inspired by the ocean [27].

#### Target audience

In the design process considerations towards the target audience was made. The client had said that the users had generally low digital competence. The team therefore decided the app needed to be relatively simple in design.

#### iOS and Android

A difference between iOS and Android is that Android devices have a built in back button, whilst iOS does not. Therefore, in the design process the team had to make design considerations around this. This meant that each view in the app would need a back button in the top-bar so that users on both iOS and Android could navigate the app with ease.

## 3.2 Equipment, tools, and technology

### 3.2.1 Flutter

One of the requirements for the application was that it had to work on both iOS and Android. Flutter would let the team write a single code base for both platforms. There was a strict time limit on the project and the client wanted the application to be usable as quickly as possible, therefore writing a single code base for both platforms would make it possible to complete the project on time. In Flutter each GUI view is constructed using widgets. Each widget has varying constructors and features which help customise the view and make it function in whatever way is necessary. A view is constructed starting with one parent widget, this widget is commonly a Scaffold. A parent widget can have one or multiple children, each child being another widget.

---

<sup>1</sup>See subsection 2.1.3 in chapter 2

<sup>2</sup>See the completed wireframes in appendix C.

In Flutter views are constructed of multiple small widgets instead of fewer large components. Therefore, if there is one huge widget encompassing everything in the view, it should probably be split into smaller faster widgets. Another advantage with Flutter was the ability to “Hot Reload”. This is a feature in Flutter which allows the developer to quickly reload the application during development to see the most recent changes made in the code. This made the GUI simpler to construct and style according to the planned design, and necessary adjustments could be made continuously.

### **3.2.2 Spring Boot**

Spring Boot was chosen for developing the application backend server. Spring Boot was chosen because of the amount of different functionality that is included with it, and its simplicity when creating production-ready RESTful<sup>3</sup> web services [28]. This allowed an MVP of the application server to be developed within the schedule, which could then be developed further.

### **3.2.3 Authentication**

#### **Spring Security**

The Spring Security module, included in Spring Boot, is frequently tested and updated to maintain a high degree of security, which was an important aspect of the project. [29]

#### **BCrypt hashing algorithm**

The BCrypt hashing algorithm was chosen for this project because of its adaptable slowness. It gives the possibility to adjust the complexity of the hashing depending on the computers speed. This helps future-proofing the password hashing process as the hashing complexity, and respectively its security, can be increased following the increase of computational power. [30]

### **3.2.4 MySQL Database**

A MySQL database is an open source relational database management system, this makes MySQL highly compatible with a variety of operating systems, different database models and programming languages [17]. MySQL uses an SSL encryption and offers multiple authentication plugins to secure the integrity of the database.

### **3.2.5 Digital Ocean Spaces**

For online image storage, the team opted to use Digital Ocean Spaces. This offered a quick, and reliable method for storing and retrieving images used for the application’s invoice confirmation functionality. [31]

---

<sup>3</sup>See subsection 2.1.4 in chapter 2

### **3.2.6 Docker**

Docker was used to containerize the backend. Docker reduced server configuration time and complexity. It also made the backend more mobile, as it could then be deployed on different hosting platforms with minimal changes to the configuration. [32]

### **3.2.7 Collaboration tools**

#### **GitHub**

GitHub is the code collaboration tool chosen to collaborate on the code-base. Two separate Git repositories were created on GitHub, one for the frontend mobile application and one for the backend server application. GitHub streamlined collaboration on the same code-base. Whenever a team member wants to merge their branch into the main development branch a pull-request is made. To approve a pull request another member of the team must review the code, and approve or disapprove the merge. Working this way ensures that the written code is high quality and documented properly.

#### **Jira**

Jira was chosen for process administration, to help the team keep track of what had, and needed, to be done. In Jira it is possible to create issue boards, track project progression, and log working hours. Issue boards on Jira are helpful to get an overview of tasks, and makes it easier to ensure that no one is working on the same task.

#### **Confluence**

Confluence was used during the entire duration of the project. The purpose of confluence was to write and manage different documents. It was primarily used to record meeting notes, sprint meetings, and sprint retrospectives. Using Confluence was very natural as it is developed by the same company that has developed Jira, Atlassian. Using both of Jira and Confluence at the same time made coordinating and documenting the work process simple.

### **3.2.8 Communication**

There were several means of communication, both internally and externally. Internally, the team used Facebook Messenger and Microsoft Teams as the primary means of communication. The Messenger group was used to inform members of the team of absence. Microsoft Teams was primarily for sharing simple files, such as images, Word documents and brainstorming documents.

Email was the primary mean of communication externally. It was used to set meetings and send meeting agendas. Furthermore it was also used whenever the team had questions for the client or supervisor regarding the project which needed answers before the next meeting.

## **3.3 Agile development**

The team decided at the start of the project that following the Agile methodology would be best for the development process, so that the team could adapt to possible unforeseen circumstances.

### **3.3.1 Roles and work distribution**

All the members would help equally in every aspect of the project, especially during the development process, but the main team roles needed to be delegated. Simon Duggal was appointed team leader, Johannes Josefsen quality assurance, and Hans Lindgård document manager. Simon was responsible for contact with the client and supervisor, Johannes' main responsibility was handling GitHub to ensure high code and product quality. Hans had the main responsibility for all different documents, mainly meeting notes, and sprint documentation.

### **3.3.2 Sprints**

The project was split into 17 sprints, with each sprint spanning one work week. A typical sprint during the project would consist of approximately 30 hours per team member. Each new sprint was started with a meeting Monday morning. During these meetings it was decided what work would be completed during the sprint. Every Friday, there was a sprint retrospective towards the end of the work day. Said retrospectives were used to summarize what each member had done during the sprint and review major changes to the code-base. Instead of working on the project as a whole, using sprints enabled splitting the project into smaller, bite-sized, pieces. Once split into bite-sized pieces it was easier to focus on getting each piece to a functioning stage.

## 3.4 Development Process

### 3.4.1 User Experience First

The team started development by creating an MVP of the frontend Flutter application with some mocked data so that the client got the ability to test and see the team's vision for the application. This got the team feedback on the user experience early, and clarified how both the client and the development team envisioned the application.

### 3.4.2 Flutter Application

#### Class structure

For the Flutter app the team organized the classes in a way that made it easy to find files. Under the lib files there are multiple classification directories. There are separate directories for: api handling, configuration, entities, translation files, offline service, views, and widgets. Each of these directories has one or multiple sets of files which fit under the general description of the directories.

#### Reducing network load

As the application was to be used on board a ship where the internet is connected via satellite, optimizing network calls and decreasing data usage was an important part of the development process. Caching data in the mobile phone's secure storage became an important part of this. This was used to store important data that needed to be kept secure, such as the user's token.

#### Handling offline scenarios

Through a dialog with the client the team found out early in the planning process that the application would have to handle offline scenarios as the user's were not always connected to a network aboard the ship. Since the user's would update the inventory stock while not having a connection to the server, a solution to keep the online database as updated as possible, while still offering the option to use the application offline was critical.

### 3.4.3 Server and Database

#### Spring Boot Server

The development of the Spring Boot server was started after the MVP version of the Flutter mobile application was completed.

## **Class structure**

As the project was setup in Maven there is a standard organizing of files. In the `src/main/java/package_name` are all the classes in the project. Under here there is a set structure to easily find any file. There is a directory for controller, service, repository, model, authorization, configuration, and user wrapper classes<sup>4</sup>.

## **Spring Security**

Since the application would handle functionality that could be critical to the users, it needed to be secure. The Spring Security framework would offer a high degree of customizability for the security configuration, and is also a well known industry standard, which points to its high degree of security, when used correctly, and its versatility.

For this application a JWT based implementation was used to handle authentication and authorization. This allowed for a stateless authentication and authorization approach, which simplified the implementation of the offline functionality.

## **Database, SQL, and procedures**

To be able to store different types of data, the application is linked to a database. Since the database is relational, the tables are linked on primary key and foreign key. An example of this is found on the user table (LoginTable) and UserDepartment table. Here the link is using the login tables primary key against a foreign key in the UserDepartment table. UserDepartment table also has a foreign key that links to the department table. This makes it possible to links a user to multiple departments without making multiple records into login table and department table.

The use of procedures makes the SQL execution simpler and more efficient. The Stored Procedures reduces the network traffic by executing a block of SQL query rather than executing a multiple queries one after another.

### **3.4.4 Quality Assurance**

Pull requests were mandatory for every merge into the main development branch for both the frontend and backend projects. This would assure that at least two of the three members of the team approve of the submitted code, i.e. the submitter and the approver.

---

<sup>4</sup>See backend source code

## 3.5 Testing

Flutter has a feature made for developing called hot-refresh, this quickly refreshes the app with the updated changes without having to recompile the whole app. This makes the testing on both Android and iOS more efficient. Furthermore, the team used JUNIT to test some of the parts of the backend server, to ensure they worked on the most basic level.

### 3.5.1 Process for user testing

User testing was done from the start of the project. Initial user tests consisted of some predetermined actions which were essential to the app, such as logging in, adding new inventory, and checking the map. During development the client was unavailable for testing at regular intervals. The client would be offshore for 5 weeks at a time on work missions. Due to this most of the initial testing was done by peers, and internally in the team. Each team member would manually test features created by another member to ensure that the feature was robust and worked as intended.

When the client got on-shore again it was possible to perform testing with the expected user-base.

# 4. Results

## 4.1 Scientific results

### 4.1.1 Network in a maritime setting

#### **Problem**

The network connectivity on a boat often varies depending on its location while offshore. Using satellite connections the crew can experience low bandwidths and poor reliability. Using these solutions also result in data usage being expensive, and traffic should be kept to a minimum.

#### **Process**

Handling poor reliability would infer that the application needs to handle important network-calls at times where the device is not connected to the internet. It needs to ensure that the user either cannot proceed with these tasks while not connected, or a process for handling the network-calls later needs to be implemented.

Reducing the data amount needed for usage offshore will reduce the cost of using the application, and better the reception of the product when used in production. To accomplish this, the current solutions need to be reviewed and improved. Methods of decreasing the amount of data needed for each network call, as well as the amount of network calls needed to be researched.

#### **Possibilities**

As the most important functionality of the application while offshore is the tracking of inventory usage, a solution was suggested where the device would store the information needed for the network-call whenever the device did not have network connectivity. The stored data would then be used to make the network-call as soon as the device reconnected to the internet.



Decreasing the data usage can be done by looking at the network-call which updates the inventory. A possible solution would be to only update the items in the inventory which have changed since the last update. This would decrease the amount of data needed for the inventory update after the initial fetch.

## Solutions

The solution for handling the offline scenarios, was a queue stored locally on the mobile device. Whenever the user needed to add or remove an item from the inventory the app would check if the device had an internet connection so that the call could proceed normally. If the device was not connected to the internet the data needed to perform the network call would be stored locally in a queue on the device to be carried out later, using the "addToQueue"-method seen in code example 4.1. A listener would then start the service, using the "startService"-method seen in code example 4.1, processing the queue when the device reconnected to the internet. The service handles the calls in their original order.

**Code example 4.1:** *The addToQueue and startService implementations. (OfflineEnqueueService.dart)*

```
1 ...
2 addToQueue(Map<String, dynamic> model) async {
3   _queue.add(model);
4   _storage.write(key: "OFFLINE_QUEUE", value: _queueToString(_queue));
5 }
6
7 startService() async {
8   if (await _isOfflineOrServiceAlreadyRunning()) {
9     return;
10  }
11  _serviceRunning = true;
12  await _updateQueueFromStorage();
13  List<Map<String, dynamic>> pendingItems = _getPendingItems();
14  if (pendingItems.isEmpty){
15    _serviceRunning = false;
16    return;
17  }
18  await _processItems(pendingItems);
19  _storeQueueAndStopService();
20 }
21 ...
```

---

To implement a solution for only fetching the items updated after last fetch, the product table needed to keep the time for when each item was last changed, and the mobile device would store the last time it fetched or updated the inventory. When the mobile device requests an update it includes the time for when it did it the last time. The backend can then fetch the items which have been updated after this, to include only the necessary items. When the mobile device receives the updated items, it can update its locally stored inventory with the new data.

### 4.1.2 Principles of design

Design was a focal point of the project. The client wanted an app that was intuitive and easy to use. By considering the principles of design the team was able to deliver an app according to these needs. Below is a list of design principles that influenced how the app was designed.

- **Contrast** To easily identify different components the design had high contrast in colors used in each view. Done by selecting a color palette and using colors from it.
- **Balance** The team tried to design with balance in mind, this meant having even spacing, same size icon, and symmetry were possible.
- **Emphasis** Changing colors depending on selection and state was done to make parts stand out.
- **Hierarchy** Functionality has been sorted with most important features higher up in views.
- **Repetition** Using the same colors on similar components made the app more intuitive.
- **Pattern** Each view is constructed in a similar way, which is done to make the app intuitive.
- **White space** White space is used to separate components and make them more distinguishable.
- **Unity** A common aesthetic theme is used throughout the app to give the features an appearance which makes them all feel like they belong in the app.

## 4.2 Engineering results

Appendix C details the requirements for the developed application. The engineering results goes through the product that has been created from the specified requirements. A detailed description of the system, including the various libraries used, is contained in appendix B.

## 4.3 Engineering results - Mobile app

The product was developed to simplify and streamline handling the inventory on board a fishing boat. The developed app is in-line with the product described by the client. Over the course of the project some aspects of the product description changed. The mobile app had a series of requirements, both technical and non-technical. The app needed to support both English and Norwegian as it would be used by international workers. It needed to integrate GPS and use maps. A user had to be able to manually edit the inventory. Lastly it also needed to be able to scan barcodes.

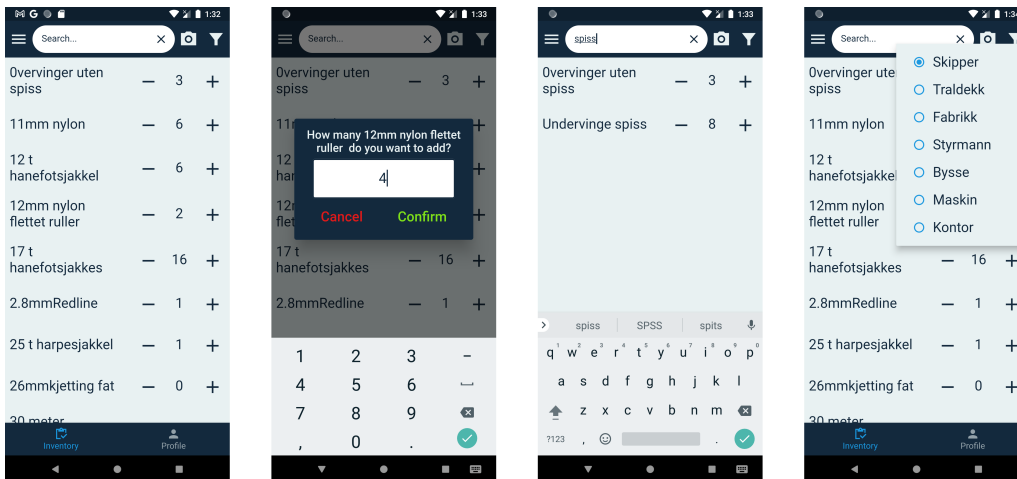
There were a set of technical requirements supplied to the team at the start of the project. This section gives an overview of how these have been implemented as well as an overview of key features.

### 4.3.1 Logging in and staying logged in

The server application uses token based authentication. Keeping the users logged in on their respective devices is a matter of storing the token locally on their device, and checking its validity when executing requests. Using the token, the backend server is able to both authenticate, and authorize the user. This removes the need for multiple logins within a short time period. It also removes the need for storing the username and password locally with the user, and sending it with every request requiring authentication or authorization.

### 4.3.2 Updating inventory

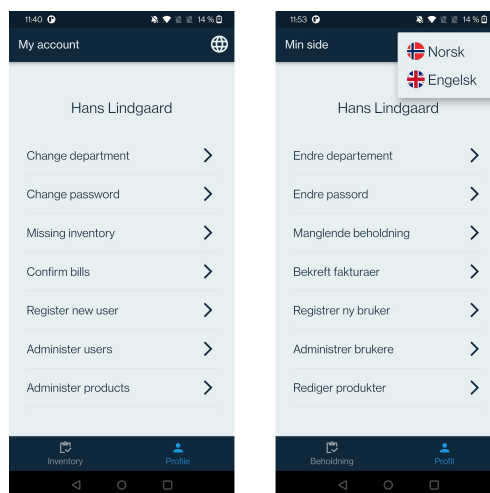
The main purpose of the app was to have a better overview of the equipment present on the boat. As mentioned in the introduction the crew were using Excel to keep track of equipment on board. In the app the users have a full overview of the inventory for each specific department. The equipment is listed in alphabetical order with the option to either add or remove to the amount of the equipment. In the top bar the user can easily search for equipment either by product name, number, or by scanning its barcode. It is also possible to swap the displayed inventory, to another department the user has access to. Adding or removing from the inventory creates a report in the database. The report contains information such as when the equipment was used, who used it, and at what latitude and longitude. The reports signifying used equipment is shown in the map feature to quickly see where what equipment has been used. Figure 4.1 shows how the inventory view looks, and what happens after pressing various buttons.



**Figure 4.1:** Inventory view. Second image shows the popup shown when adding stock. Third image shows the inventory after a search has been made. List image shows the dropdown for selecting department inventory to view

### 4.3.3 Language settings

As the whole user base did not know both Norwegian and English the app needed to have an option for changing language so that it could be used by anyone. Every mobile device has a list of preferred languages. When the user first launches the app on their device the language chosen is the first language in the list of preferred languages that is a dialect of either Norwegian or English. If the user has neither a dialect of Norwegian or English the app defaults to Norwegian. After the user has logged in they can swap between the two language choices as shown in figure 4.2. Swapping the language also stores the selected language so that when the app is closed the app retains the chosen language.



**Figure 4.2:** My account view. Left is language not pressed, right is pressed

The app can use two or more languages by using the `flutter_localizations` package. The package allows a developer to create multiple translation files, one for each language. These files contain key-value pairs, there is a list of keys with their associated translation, e.g. `helloWorld : "Hello World!"` and `helloWorld : "Hallo Verden!"`. When displaying text in the app one of the language files are chosen depending on the selected or default language of the device, and for each key the associated word or phrase is displayed. In the code this enables simply writing the key for the phrase and the correct version is displayed. In short changing language changes the translation file that is used to select text.

#### 4.3.4 Map integration

A key feature within the app was to be able to see what equipment had been used where. Google Maps API was used to show a detailed map and allowed adding markers to it. Google Maps has a library for Flutter applications which has multiple built in functions used in this view. Each marker on the map shows that at least one piece of equipment has been used within  $505m$  of it<sup>1</sup>. Each time some equipment is used a report is made on the latitude and longitude the user is currently at. Getting the latitude and longitude is done using the Geolocator library. This library uses the mobile device's GPS to find the latitude and longitude. If the locator is not able to find the current location it uses the last known location. Markers are placed based on these reports.

Code example 4.2 shows how each report has its latitude and longitude rounded to 2 decimals and how any reports with equal rounded coordinates are grouped together. In doing this the team was able to reduce visual clutter in the view. The grid of sorted markers also made it easier to differentiate markers, if they were closer together it would be hard to see and click them. Each marker is also color coded according to how many items have been used there. By using the max and min amount of equipment used at different locations a hue is calculated. A simple normalization function is used to make the hue of the marker with the most used equipment be purple, and the least used equipment red. This is done so that a user can quickly see where they have used the most and least equipment.

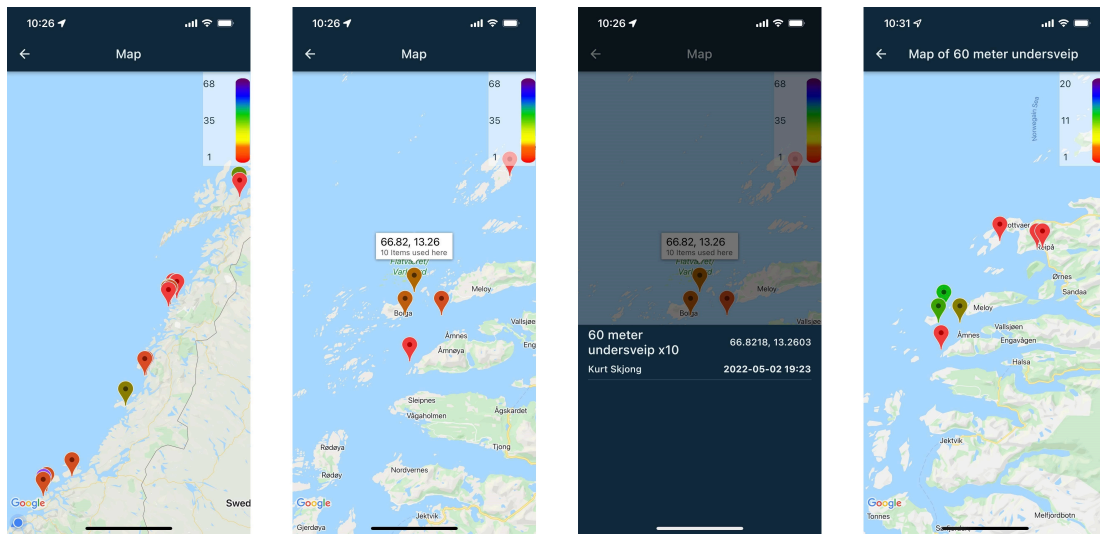
---

<sup>1</sup>This is naive sorting of markers, the markers are on a 3D globe not a 2D square so the rounding becomes slightly inaccurate, but as the real coordinates are available to users the team felt doing naive rounding was fine.

**Code example 4.2: Sorting map markers into a grid (ReportService.java)**

```
1 private Map<String, List<Report>> sortReportsByIntoGrids(List<Report> markers) {
2
3     Map<String, List<Report>> mapSortedByLatLng = new HashMap<>();
4     markers.forEach(report -> {
5         float lat = report.getLatitude();
6         float lng = report.getLongitude();
7
8         lat = getRoundedFloat(lat, 2);
9         lng = getRoundedFloat(lng, 2);
10
11         String latLng = "" + lat + ", " + lng;
12
13         // If the Map is empty or it doesn't contain the new key
14         // create a new key with that latLng and add the report which
15         // produced the key to the list in the map
16         if(mapSortedByLatLng.isEmpty() || !mapSortedByLatLng.containsKey(latLng)) {
17             mapSortedByLatLng.put(latLng, new ArrayList<>());
18             mapSortedByLatLng.get(latLng).add(report);
19         } else if(mapSortedByLatLng.containsKey(latLng)) {
20             // If the map already contains the key just add the report
21             // To the list present on that key
22             mapSortedByLatLng.get(latLng).add(report);
23         }
24     });
25
26     return mapSortedByLatLng;
27 }
28 ...
29 private float getRoundedFloat(float floatToRound, int howManyDecimals) {
30     float roundedFloat;
31
32     roundedFloat = (float) (Math.round(floatToRound * Math.pow(10, howManyDecimals)) / Math.
33     pow(10, howManyDecimals));
34
35     return roundedFloat;
36 }
```

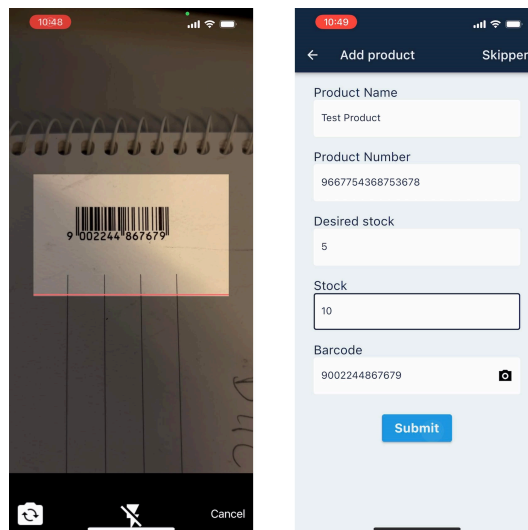
Figure 4.3 shows the view in its different forms. When pressing a marker a bottom drawer pops up. Within the drawer there are more details about the different equipment used there. Who used the equipment, when and more accurate coordinates are all shown when the drawer opens. The map can either be opened up to display all the equipment used by a users active department, or a user can double tap a piece of equipment in the inventory view and see a map of only that piece of equipment.



**Figure 4.3:** Map view. Left is the normal map. Second is when a marker is pressed once. Third is the box pressed. Right is an equipment specific map.

### 4.3.5 Scanning barcodes

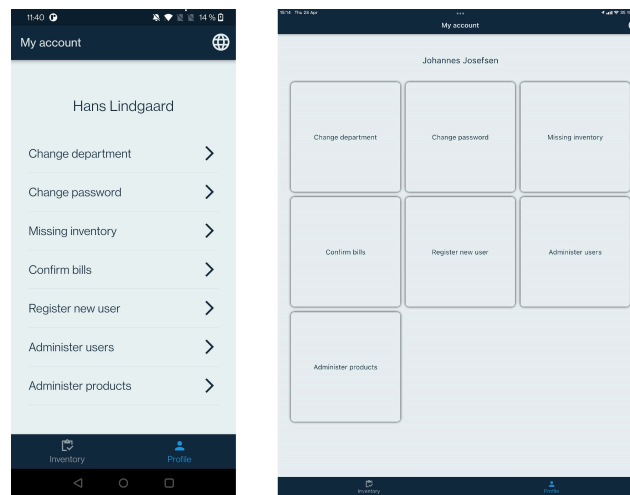
Most of the pieces of equipment used on the fishing boat have barcodes. The flutter\_barcode\_scanner library is used to scan barcodes. Scanning barcodes makes finding and identifying equipment much faster. A user can scan the barcode of a piece of equipment and the code appears in the search-bar and is used to search for equipment. If the equipment has been registered correctly it will show up in the search and the user can register that they have used it. Figure 4.4 shows a user scanning a barcode. After being scanned the numbers are entered into the text field associated with this specific scan.



**Figure 4.4:** Scanning a barcode whilst registering a product

### 4.3.6 Phone and tablet

The app automatically detects and sets the GUI-layout depending on if the user is using a phone or a tablet. The Flutter framework uses the `MediaQueryData` [33] class to help detect the size of the users screen, this was used to accurately modify and adjust all the views in the app. During development most views were first completed for mobile before being modified to better fit tablets. Different views such as my account view 4.5 where specifically designed for a tablet to accommodate the extra space given.



**Figure 4.5:** *My account view on both phone and tablet. On the left is on an Android phone, the right is an iPad.*

### 4.3.7 Registering users

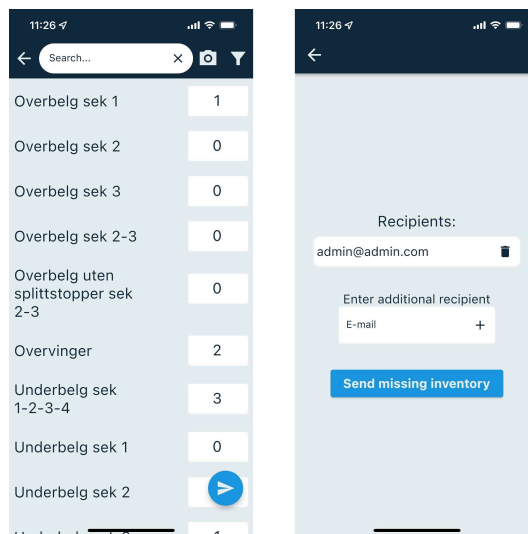
There needed to be a way to restrict who could register for the app. The way this was solved was by allowing the app's administrators to register the email-addresses of their employees. When they are registered they receive an email telling them that they have been registered for the app, and need to set themselves a password in the app before they can log in.

The process of setting a new password is the same as when the user has forgotten their password. They enter the "set password" view from the app's login screen, here they enter their email-address to receive a verification code, and when they have entered the correct verification code they can set a new password.



### 4.3.8 Missing inventory

Each product has a desired stock which was set when it was created. The client wanted the office to receive a report containing the number of each piece of equipment that was needed before the next trip. The missing inventory view displays text boxes with the amount missing of each equipment displayed in figure 4.6. This number is calculated by taking desired stock and subtracting the current stock. The user can change the number that should be ordered by entering another number. After the user has checked what the inventory is missing they can send the report to the office. Only the equipment which has a missing amount is sent. The intended recipients receives a PDF on their email, and an employee can easily order the equipment that is needed.



**Figure 4.6:** *Missing inventory and send missing inventory.*

### 4.3.9 Data handling

To delegate the responsibility of handling the requests and data, a singleton class was created. This allowed for less complicated implementations in the rest of the app, and gave a single point of access for request handling, increasing the code's cohesion. An example of how the methods in this class are structured can be seen in code example 4.3, where the Dio code library is used for creating the requests.

**Code example 4.3:** *Example of an async call for trying to delete a user (api\_controller.dart)*

---

```
1 Future<bool> deleteUser(String email) async {
2   bool success;
3   try {
4     await _setBearerForAuthHeader();
5     var data = {"username": email};
6     await dio.delete(baseUrl + "api/user/delete-user", data: data);
7     success = true;
8   } on DioError catch (e) {
9     if (e.response!.statusCode == 403) {
10      _showErrorToast(
11        AppLocalizations.of(buildContext)!.notAuthorizedToDeleteUser);
12      forceLogout();
13    } else {
14      _showErrorToast(AppLocalizations.of(buildContext)!.deleteFailed);
15    }
16    success = false;
17  }
18  return success;
19 }
```

---

#### 4.3.10 Order confirmation

To simplify the client's process of checking that the received invoice match the received shipment, the app includes functionality for uploading a picture of the invoice and send it to the receiving department for confirmation before the invoice is payed. The user's in the respective departments can then see the invoice, check that the received invoice agrees with what has actually been received, and either confirm or reject the uploaded invoice.

#### 4.3.11 Error handling

Doing calls to the server may sometimes result in an error being received back. These errors can occur due to different reasons. To handle the errors which can be produced by the network calls every call is surrounded by a try-catch block. Depending on the status or error code received, different error messages are shown. Code example 4.4 shows an example of how different HTTP error codes are handled. The switch case selects the appropriate error message depending on the possible error codes sent from the server.

**Code example 4.4:** *Handle error codes received after trying to register a user. (api\_controller.dart)*

```
1 void _handleRegistrationDioError(DioError e) {
2   switch (e.response!.statusCode) {
3     case 403:
4       _showErrorToast(
5         AppLocalizations.of(buildContext)!.notAllowedToCreateUser);
6       forceLogout();
7       break;
8     case 409:
9       _showErrorToast(AppLocalizations.of(buildContext)!.userAlreadyExists);
10      break;
11     case 400:
12       _showErrorToast(AppLocalizations.of(buildContext)!.badRequest);
13       break;
14     case 422:
15       _showErrorToast(AppLocalizations.of(buildContext)!.invalidEmail);
16       break;
17   }
18 }
```

---

### 4.3.12 Deployment and testing

The app has been deployed to both Android and iOS devices. Due to some policies at Apple it is not available directly in the iOS app store. Apple has policies restricting public release of apps which are only for use internally in a company. As the app is only usable by the client and their employees the app could not be released on the iOS App Store. It is therefore only available through the TestFlight system, which is Apple's testing service. However the app has been made publicly available on Google Play store. [34]

The client went on a trip on May 1st, before the start of the trip some of the employees downloaded the app and tested keeping track of the inventory. In this period small bugs were found by the users. Sometimes removing equipment would not work, the team figured out this was due to location not always being available which stopped the method before the stock could be updated. Using last known location, and adding a default location, the team fixed the issue and allowed stock to be updated consistently.

Another bug relating to updating stock was that when the clock was 00:xx the stock would not update. The date-time format the team had used in the frontend used 1-24 hours instead of 0-23 which was used in the backend. The team had not tested the app after midnight and had therefore never encountered the bug. Changing the date-time format in the frontend to use 0-23 hours fixed this bug. Despite of these bugs the client has said they are impressed and very pleased with the final product. They said it was easy to use and intuitive [E.1].

## 4.4 Engineering results - Server application

The backend is a REST API server developed with Java's Spring Boot framework. It is structured and developed following Spring Boot's best practices [35]. Separating the layers in their own package, to make the code more organized, easier to maneuver, and more cohesive.

### 4.4.1 Controllers

The controllers handle the endpoints of the server. The controllers responsibility is handling the request data sent to the server, and giving the response. The controllers handle formatting of i.e. the JSON from the request body as seen in code example 4.5, or the request parameters, to arguments usable for the service layer, as well as calling the correct methods in the service layer to create the correct response to the request.

**Code example 4.5:** *Handling request data for editing user. (UserController.java)*

---

```
1  @PostMapping("/edit-user")
2  public ResponseEntity<String> editUser(HttpEntity<String> entity) {
3      try {
4          JSONObject json = new JSONObject(entity.getBody());
5          String oldEmail = json.getString("oldEmail");
6          String newEmail = json.getString("newEmail");
7          String name = json.getString("name");
8          List<Department> departments = getDepartmentsFromJson(json);
9          User user = new User(name, newEmail);
10         userService.editUser(user, oldEmail, departments);
11         return ResponseEntity.ok().build();
12     } catch (JSONException e) {
13         return ResponseEntity.badRequest().build();
14     }
15 }
```

---

### 4.4.2 Services

The service layer handles the main business logic. When it is called from the controller, the services handle the parameters and execute the logic, often using data retrieved from the repository layer, or by storing or updating the data using the repositories as seen in code example 4.6.

**Code example 4.6:** *Handling the user editing functionality. (UserService.java)*

---

```
1  public void editUser(User editedUser, String oldEmail, List<Department> departments) {
2      userRepository.editUser(oldEmail, editedUser.getEmail(), editedUser.getFullname());
3      updateUserDepartments(editedUser, departments);
4  }
```

---

### 4.4.3 Models

The models are used by the controllers, services, and repositories, to represent data and the entities in an easily manipulated way within the Spring Boot application. They also represent the structure of the tables which these are modeled after.

### 4.4.4 Repositories

The repositories represent the data access layer of the Spring Boot application, and are interfaces for interactions with the persistence layer. They create and handle the queries between the Spring Boot application and database. The repositories offers the service layer an easy way of retrieving data from the database. This project has taken advantage of Spring Data JPA which makes it possible to define the Java interface, with possibilities for custom queries and configuration as seen in 4.7, to simplify development of the persistence interface.

**Code example 4.7:** *Defining the query to call the procedure for editing a user. (UserRepository.java)*

---

```
1  @Modifying
2  @Transactional
3  @Query(value = "Call HandleUser('Update',:newEmail,',',:fullname,',',:username);", nativeQuery =
   true)
4  void editUser(@Param(value = "username") String email,
5               @Param(value = "newEmail") String newEmail,
6               @Param(value = "fullname") String fullname);
```

---

### 4.4.5 Image upload

For handling the images of the invoice, the images received in the request are uploaded to a Digital Ocean Space, which allows retrieval of the stored images using URLs. The location of the image of the invoice is stored in the respective order's entry in the database.

Uploading and storing the images using this method also allows use of the images in other scenarios, such as further development and extensions of the system.

### 4.4.6 Database

The database is structured into 7 tables and 8 procedures. The main relation in the database is a Many-to-One relation between most of the tables, linking the tables on primary and foreign key. By linking the tables it is possible to fetch data from both tables using only one query.

The procedures take in different parameters to collect and change data in the database. The procedures uses a parameter called "Calltime" to distinguish the wanted block of SQL code to executed. Procedure HandleUser using calltime to differentiate between a Insert and Update action as displayed in Figure 4.8

**Code example 4.8:** Code taken from the `HandleUser` Procedure. Displaying the use of `Calltime`

```
1  if(Calltime = 'Insert')
2  then
3      Insert into LoginTable (Username,Password,FullName) values (UsernameString,
4      PasswordString,Fullname);
5  end if;
6  if(Calltime = 'Update')
7  then
8      Update LoginTable set Username= UsernameString, Fullname=Fullname where Username =
9      OldEmail;
10 end if;
```

---

### 4.4.7 Security

By using Spring Security the application could be secured in a way that required authorization and authentication for all requests except for the log in, and set password functionality. This ensures a high degree of security, requiring a token with the correct authorization to execute calls to the API.

Keeping the log in, and set password functionality open was necessary to let users log in from any new device, and also to let them set a new password if they had forgotten their current. Setting a new password still requires authentication in the form that the user receives an email to their registered address, which they need access to, if they are to complete their password change.

Using Spring Security the team was able to configure the endpoints in a way which uses the tokens for securing the endpoints from unauthorized users, as seen in code example 4.9. This also makes sure that anyone using an external method for accessing the API can only do so using a token with the correct authorization.

**Code example 4.9:** The start of the security configuration method restricting access to endpoints. (`SecurityConfig.java`)

```
1  @Override
2  protected void configure(HttpSecurity http) throws Exception {
3      http
4          .csrf().disable()
5          .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
6          .and()
7          .addFilter(new JwtAuthenticationFilter(authenticationManager(), userService,
8          jwtProperties))
9          .authorizeRequests()
10         // Permit login and registration
11         .antMatchers(HttpMethod.POST, "/auth/login").permitAll()
12         .antMatchers(HttpMethod.POST, "/auth/register").hasRole("ADMIN")
13         ...
```

---

### **4.4.8 Future deployment of server**

Using Docker gives any new set of users the possibility to build the server using the Docker file. The Docker file can be built in a new Digital Ocean droplet, locally on a computer or any other hosting platform that supports Docker. The file compiles and creates both the Spring Boot server and the database.

## **4.5 Administrative results**

The team was able to use Jira, Confluence and GitHub to manage and complete the project within the allotted time. Appendix A shows the plan for the project. It contains internal and external deadlines.

### **4.5.1 Working in Jira**

Jira was used as the issue tracking tool throughout the whole project. Each member of the team had the option of creating, editing and assigning issues.

### **4.5.2 Development overview**

This section goes over the development process as a whole documenting what has been done and when.

#### **Sprint**

The team had retrospectives at the end of each sprint to summarise what had been done by each member. In the meetings the team discussed what had worked well and what had not worked well. These things were areas which the team tried to improve or maintain the next sprint.

#### **Planning**

The initial step of the project started January 10. Before the team had decided on what development process would be used, the team went through the project specifications from the client to gather an overview of the project at hand. On January 13, the team had their first meeting with the client and most of the product requirements were cemented. During the course of this week the team decided to work in the agile method, use Jira and Confluence to aid the project, and that each sprint should be one week long. The first sprint started on the subsequent Monday, January 17. The first sprint was used to decide what technologies the team wanted to use for the different parts of the project.

### **Preliminary work**

After the planning process had been completed the team started working on the work needed in advance of development. The first goal was to create wireframes for all the views for both tablet and phone. The team created issues in Jira for each view, and a team member could then select a view in the issue board, move it to "in progress", complete it and then move the issue to "Done". Parallel to creating the wireframes the team also worked on creating the pre-project plan which was one of the obligatory work requirements for the Bachelor thesis. The team sent the wireframes to the client and received feedback that the views looked good and they were pleased.

### **MVP**

The team had decided that the completing an MVP for the frontend should be the first step in the development process. Creating an MVP early would allow the team to test the app with end users early on. Testing early would also follow the agile principle of failing fast. An MVP was completed by the end of the 3rd sprint. While the MVP was completed early it would take time to test it physically with the end users as the client were on a 5 week trip/mission from the end of February.

### **Backend**

The team was able to complete the MVP before the end of sprint 3 so work on the backend had started before the start of sprint 4. The team expected the backend development to take more time than that required to develop the frontend. Initially the code was written in Java 8 as the team had misunderstood and thought Docker could only run the code if it was written in 8. At the beginning of March the team realized this was wrong so the project was moved to Java 17. At the end of sprint 6 most of the functionality in the backend had been completed, therefore the team started connecting the frontend MVP with the backend.

### **Time lost**

At the start of March, the team had to dedicate the majority of work hours to another class, INGA2300. This class had a project that had a deadline in the middle of March as well as an exam towards the end of the month. The class took most of the teams time so little progress was made in the time period from the end of sprint 7 until the start of sprint 11.



### **Integrating the two parts**

When the MVPs for both the frontend and backend were completed the team started connecting them to each other. The team had accounted for the integration when writing the backend and frontend. When writing the backend considerations towards how the data will be handled in the frontend were made. Similarly in the frontend there were considerations towards handling data in the backend. Doing this meant the integration process was mostly problem free, and the team could move on to improving and extending the applications functionality.

### **Publishing**

The internal deadline for finishing the app and having it published was before Easter, April 8. By then the app was close to done, only missing the publishing before the full-scale test could start. Docker was chosen to containerize the server and database, which could then be published to a Digital Ocean droplet. The server was published by April 23rd. With an internal test version of the app the same day. The client was therefore able to test the app with some of their employees before their 5 week trip, from May 1. The feedback received from the client up until May 1 allowed the team to make small changes and fix bugs. Due to this it was possible to get a public version for the whole crew available to download before May 1st.

Following this release the team kept in close contact with the client to resolve any issues they discovered, and provide rapid bug fix updates to the app.

# 5. Discussion

This chapter focuses on discussing and reflecting on the work that has been completed in this project. It will also discuss any challenges and difficulties met during the process.

## 5.1 Scientific discussion

### 5.1.1 Reducing data traffic

As the boat has a varying degree of bandwidth and internet connection quality, implementations which reduce the data usage and request amount needed to be taken into consideration.

Keeping data usage to a minimum involves not including unnecessary data in the requests and responses. When updating the inventory list for example, it would not be necessary to receive the items which are already up to date, but only include the changed items.

Implementing this method for the inventory updates, where only the changed items are fetched from the server to decrease data usage, could be used in other areas of the app to further decrease data usage. Due to time constraints, and other functionality not being used frequently when offshore, this has not been prioritized during the time of the project.

Another method to reduce data usage could be to optimize the requests to the API further. This could be done by only requesting the data that will be displayed in the app, and not include data which is only necessary after a view change. When changing view, the app can then do a new call where it again only gets the necessary data.

### Image quality restrictions

The team has ensured that the traffic over the satellite network is minimal. To do this it was necessary to restrict the image quality, as some mobile devices are equipped with high quality cameras, which can produce large, high resolution images, which require a larger amount of data.

In the public test the team encountered a possibility that the image size exceeded the limitations made for the backend server. The decrease of the image quality to 50% in the application, makes the image size decrease. This, and making a small increase of the upload size limit, made the image possible to upload. Taking into consideration that this functionality is mostly used by the on-shore office department, and that the crew will use this functionality while on land, this problem does not increase the offshore data usage excessively. The team acknowledges that the solution is not the optimal solution, but necessary to deliver the product with the specified requirements on time.

A method to further reduce data usage could be that when the backend receives an image, it prepares several versions of the image in different qualities. This way when the app requests an image, it has the possibility of choosing the quality depending on the quality of the network connection.

For similar inventory management systems, including an image with each item in the inventory list could prove beneficial. It could make it easier to find products. In this solution however, due to network limitations, and user familiarity with the equipment, this would be excessive.

### **5.1.2 Ease of use**

A key requirement of the app was ease of use. Having the app be easy to use was a key consideration during the entire design process. The app will be used in an industrial setting, so when designing the app there was a focus on large icons and text to make it easy to see what buttons did what. Furthermore, aesthetics was not a focal point as there is no need to attract new users. When the client used the app full-scale, with users who have never seen the app before, the team received feedback that all users found the app easy to use E.1.

One of the principles of universal design is to have a tolerance for mistakes[36]. The app has a relatively low tolerance for mistakes. If a user changes something there is no "undo" button. For example if a user deletes a product there is no way to roll back the change. A possible improvement would therefore be to implement a feature were the user would be able to restore mistakes.

## **5.2 Mobile application**

This section discusses the app created in its entirety, the different aspects which are satisfactory, and aspects that could be improved.

### **5.2.1 Solving the clients problem**

Collaborating with the client to define their problems, the team has been able to develop and publish a solution which manages and tracks the inventory on board the client's boat.

The inventory functionality lets the users easily see, and change the amount of any available product in their department's stock. Displaying the inventory as a list with search functionality is a simple way to let the user see what they have, and with the buttons for adding and removing items directly on the product's tile in the list makes the usage of it intuitive.

A web based interface could be a useful expansion on the existing system. The employees working in the office mainly use stationary computers or laptops, so having a web based interface for some of the administrative features could be useful. Making registering a user, seeing the map with markers, administering users and products, available in the browser could simplify the work of the office department.

### **5.2.2 Flutter framework**

Using the Flutter framework gave the team the ability to have an MVP of the frontend app on both platforms early in the process. Because of the agile work method, the team was able to use the sprints to focus on creating better iterations of the app. Being able to create an MVP of the app quickly also allowed some bugs, and errors to enter the code and manifest if they were not discovered for a while, making the bug fixing and clean up process at the end of the project a little more extensive. The speed of development using a cross-platform development framework, did however greatly make up for these problems.

### **5.2.3 Deployment to client devices**

The internal deadline for completing and publishing the app set by the team during planning was right before Easter<sup>1</sup>. Unforeseen difficulties with deployment caused this deadline to be pushed back. By Easter the app was close to done, only having minor bug fixing, and full scale testing left. After the app was fully completed it took a lot of effort getting the app published. The deployment process was slightly delayed due to missing access to necessary accounts and services, as the team got access from the client to the different accounts needed to deploy the app right after Easter.

Giske Kystfiske wanted a finished version of the app to be public by May 1st so they could use it when they were offshore for 5 weeks. The team was able to release the app so that Giske Kystfiske could try the app for this trip, and the team could receive feedback from a full-scale test of the solution, while the users familiarize themselves with the solution.

#### **Apple:**

The deployment process to deploy an app to Apple's App Store is strict, and the team used some days to complete the structure Apple demanded to deploy the app for testers using TestFlight.

---

<sup>1</sup>Detailed plan available in appendix A

All deployed apps on App Store have to be eligible to all consumers and as the app initially is made for a specific company, the App Store does not recognize that the app can be distributed to a wider audience. After some research, the team found that Apple has a distribution platform designated for company-specific apps, called Apple Business Manager. However, Apple Business Manager needs an extra account, which the team was not provided. With internal and external discussions, the solution was to deploy the app to testers using TestFlight.

### **Google Play Store:**

The app is released to the Google Play Store and is currently in use by the client. When starting the Android deployment process the app was first released for internal testing allowing a few invited testers to download so that the team could receive user feedback and quickly update the test versions. When the app was deemed production ready it was released for production after an approved review from the Google Play Store.

### **Testing**

The team conducted testing in three different phases. The client has been positive and happy with the changes made in every phase, while also giving constructive feedback.

The first phase included having a walk-through with the client on the wireframes. The client was sent the wireframes and gave a small feedback before the meeting shown in E.2. In the meeting the client was able to interact with the wireframes and give feedback on the different design elements and the user experience. The client tested the wireframes without direct instructions on how to use it, and from their experience gave verbal feedback to the team. The team wrote down the user's feedback from the meeting and made the appropriate changes.

Phase two of testing involved a more complete test, as it involved giving the client the MVP to interact with. MVP was compiled on a phone so the user got a more realistic experience of the app. The users got a set of instruction to test and give feedback on. The feedback received gave the team a more complete overview of the remaining work. The overall feedback was that the MVP was in line with what they wanted, and that no design changes needed to be made.

The last testing done before the app was in full production was the full scale test from the client before their trip May 1st. Only the captain and one staff member had tested the application in the previous tests. For this test the whole crew got to try the app. The users were told to use the app in a working environment, and to send an email detailing any bugs or errors encountered. The client reported some bugs, all of which the team was able to fix. Despite the bugs the users found, the client said they are very pleased with the final product as expressed in appendix E.

Overall, the team is satisfied with how the test process was conducted and thankful that the client was able to participate in all phases of the process. In retrospect the team acknowledges that instead of doing the feedback verbally the team could have created a form to better note the client's thoughts.

### 5.2.4 Handling offline scenarios

The team has developed the app while taking into an account that parts of the ship lack network access. The most critical and main functionality of the app, which is the inventory, needed to handle this in a way that the user could still update the stock while being offline. Using the queue implementation would allow for several changes to be made from a user before the device regained its connection to the internet. Implementing this functionality means that the inventory is cached in the app, so that the user has access to the items even when offline.

The queue implementation for the app ended up being a success. When using the inventory functionality while offline the user is notified through a red bar, indicating that there might be some mismatches between the displayed list and the actual inventory<sup>2</sup>. This gives no further hindrance in using the app as intended even when offline, as the stock is updated with the changes when the device reconnects to the internet.

During the full-scale test the users have tested the app both with, and without an internet connection. Using it as they normally would they experienced no problems relating to the offline functionality.

The current solution gives feedback when the app is offline and unusable functionality displays relevant messages on errors. Further improvement of the offline functionality could include locking functionality which is not available when offline, and finding a better way to give this feedback to the user. Using the offline queue service to handle more functionalities where possible has also been accounted for by the current implementation.

### 5.2.5 Handling missing GPS signal

While offshore, GPS signals can be of various quality, and sometimes might not exist at all. It is not possible to determine exactly where the user is when they do not have any GPS signal on their device. To handle these scenarios the team needed to determine how the location for the usage reports should be gathered. The possible solutions the team came up with were these:

- Using the last known location.
- Using the next known location.
- Using both the last, and next known location and determining a middle point.

Using the last known location is the fastest solution to implement, as the location library used, includes a method for fetching the last known location. The downside is that depending on the time the last location was logged, the accuracy of the report location can vary greatly.

---

<sup>2</sup>This happens when another user has changed the stock while the first user is offline.

Using the next known location would involve another form of call queue, like the offline queue used for when the device does not have a network connection, this would entail some more work needed than implementing the last known location solution while not gaining any more accuracy.

The last alternative was using both the last known and next known location would likely be the solution which gives the most accurate location for the report. This solution would still involve another queue implementation as it would have to wait for the result of the next location, before the calculation of a middle point could start. It would also involve research to develop a method to calculate a point which is likely to be closer to the actual usage than the previous two methods. Depending on how straight the boat has moved it can also give locations further away than just using the previous or next known location.

Taking these problems and solutions into account, the team decided using the last known location would be the best solution to implement in the app. As the amount of work needed to implement the other solutions were substantially larger, the probable accuracy gained by calculating a middle point not big enough, and the accuracy not important enough for the client to justify the amount of work estimated for those solutions.

### **5.2.6 Requirements**

The client had given the team a list of requirements at the start of the project, as can be seen in appendix D. These were used to develop the detailed system requirements, which can be found in appendix C. After discussions between the team and the client, as well as internally, some requirements were found to be nonessential features rather than requirements from the client.

One of the features the client wanted was to be able to scan order bills they received from a supplier and have it entered into the stock automatically. A problem with this was that there was no standard format for the order bills. Differences between each order include that some might be hand written, some had product numbers only and some only product names. Creating a scanning functionality to read and interpret all of these was outside the scope of the project. So the team, with a recommendation from the supervisor, told the client that this was not achievable. The team did however implement functionality to simplify the re-stocking process, by implementing the order confirmation view.

## **5.3 Server application**

### **5.3.1 Spring Boot**

The solution to the clients problem included creating a server for the app to communicate with, to make it simpler to handle requests and persist data. For this, a REST API is a good solution. Creating a REST API meant that it would be simple to use the API for frontend extensions, i.e. a web interface.

Being robust, highly configurable, and including a scalable, production-grade library for handling security, Spring Boot was used for creating the REST API. It allowed the team to quickly create an MVP of the server which was testable against the MVP of the frontend app. Combined with the agile work method this allowed for fast iterations of the product as a whole, while letting the team and client try the various versions of the solution. This gave the opportunity to “fail fast”, and implement changes “on the go”.

### **5.3.2 Swapping SQL dialect**

During the majority of the development process, the backend server application used MS SQL for database communication. MS SQL made it easy to create procedures and set up the necessary tables in the database. One of the members of the team had extensive experience with it, so it was a natural choice when starting the project. However, MS SQL is expensive to get a license for, so the team decided that the project should swap to MySQL as this would be cheaper for the client. Swapping dialects required quite a lot of additional work, which could have been avoided had MySQL been used from the start. However, by using MS SQL initially, the team was able to start the backend development earlier, as the team had a development database with default data to work towards.

### **5.3.3 Stored Procedures**

The use of stored procedures made the team less dependent on the parameters to use in the execution of queries to the database. An overview of all the procedures as well as the database structure can be found in appendix B. Within the procedures there is a possibility to fetch data to use before an insert or update is executed. This allows the backend server to make the call to the procedures to handle the data and not directly run a query into the database.

As this was new to the team, it took some time to be able to create all the procedures. The result of the procedures has been that its easy modify a SQL query. The team can update the procedures to fix a potential bug without having to re-deploy the entire backend server, saving a new update and downtime of the server when updating.

### **5.3.4 Deployment of server and database**

The server and database are deployed to production using a Digital Ocean Droplet. Using a docker-compose file, Docker builds the MySQL database together with the Spring server in two separate containers under the same local network. This creates the possibility to deploy the server and database on any deployment service supporting docker.

The team did experience some difficulties setting up the Docker container, mainly due to the MySQL and Spring Boot server not communicating with each other. After some research and work, the team was able to setup the docker-compose file in the correct manner, so the server and database were able to communicate with each other.



## **Docker**

The use of Docker makes it possible to containerize the developed solution. Allowing for high mobility of the solution as the Docker container takes responsibility for configuring and running the server, the deployment service only needs to run the container. Initially the deployment service was not familiar to the team, but rigorous research meant the team was able to utilize it effectively. The team is satisfied with using Docker which made the deployment process easier after the deployment service, Digital Ocean, was chosen.

## **5.4 Administrative discussion**

### **5.4.1 Communication**

This section goes over the communication during the project, both internally in the team and also externally, from the team to other parties.

#### **Client**

Communication with the client has been excellent. The client works on a fishing boat and is therefore offshore for five weeks at a time. Communication with the client has therefore mainly been via email. The client has received the wireframes by email, and the team has received positive feedback on all the design choices [E.2]. The client has been able to provide regular constructive feedback on the progress of the project as updates were sent to them whenever major project milestones were hit.

When the client was on-shore the team was able to have in-person meetings. The first two meetings of the project were held physically at campus NTNU. The first meeting was used to clarify any questions the team had regarding what the client wanted. During the second meeting the team was able to show and demonstrate some wireframe sketches and receive early feedback on the design.

Through the project the team has experienced that communication through email has been sufficient, but that physical meetings tended to speed-up the communication drastically.

#### **Internally**

The team worked in the agile method so it was decided that each morning the team would have short 15-minute stand-up meetings at 9 o'clock. Having stand-up meetings every day helped structure the workday and helped keep the team up-to-date on what the other team members were doing. Having everyone working on campus was also very important for the daily structure. Everyone meeting on campus helped keep morale up. Furthermore, it trivialised helping the other team members, with everyone situated in the same room it made it easy to simply ask the peer next to you for help.

## 5.4.2 Development process

### The plan

The effort put into the plan was reflected by the fact that it was easy to follow. Being optimistic about the schedule during planning to put pressure on the team, and not accounting for some challenges and time sinks, the team had to deviate from one of the milestones on the plan. The plan to complete and publish the app before Easter was slightly delayed.

Delays were mostly caused by the whole team, at different points, catching COVID-19. The team managed to make the app available the first week back after Easter. Working agile, and setting optimistic internal deadlines, helped let these deviations have little to no impact on the final result of the project.

### Agile methodology

The team, having had some prior experience working in the agile methodology and since it is the industry standard for development projects, meant it was natural to work agile. One of the main advantages the team experienced working in agile was being able to work with what was most important at the moment, and not what was established in a rigid plan. Working in agile also kept the client involved during the whole process. As mentioned in 5.4.1 collaboration with the client was done by email. Having the client continuously involved meant the team got feedback on progress and how the app looked continuously and changes could be made quickly to better fit the clients wants and needs.

### Test-driven development

A development process that the team considered early in the planning process is test-driven development. Not developing using a test-driven development process has had both up-sides and downsides for the project. Considering that none of the team members had any experience working test-driven it had the possibility of doing more harm than good.

TDD would have helped develop an easily refactored code base, where implementation improvements and functionality extensions would be safer to develop. This does however require a lot of experience and knowledge around good test design, and the general TDD process, to execute in a proper fashion, something none of the team members have.

Using TDD does extend the time used for implementing new functionality quite a bit, something this project would involve a lot as it is a brand new system. Given the amount of time the team was given for the project it is likely that the deadline would be exceeded if the team had decided to work using a TDD process.

### 5.4.3 Collaboration with Jira and Confluence

The team collaborated exceptionally throughout the project. Having collaborated on several projects previously the team was already familiar with each others working habits. Each work day was also set in the same room, with all team members present most days. Working next to each other meant that Jira sometimes ended up being a hindrance more than an aid. If someone discovered a small bug it was easier to simply ask the person next to you if they could fix it, rather than creating an issue, assigning it to the person, moving it to "doing", and moving it to done. Having never used Jira before the team was inexperienced with this at the start. As the project progressed Jira was used more actively, and proved a helpful tool to keep track of bugs which no-one could fix straight away.

If the team had put more effort into familiarizing themselves with Jira from the start the project could have been better organized. To the team, it seems that Jira has more use in larger teams where direct communication between members is harder.

The product specifications given by the client were quite detailed so the team simply developed from meeting notes and the product specifications. Furthermore, the team did create user stories to help get an overview of the requirements and what each feature needed to function. Upon reflection it became apparent that the team should have created a detailed product specification in Confluence from the start. This would have made it easier to get a complete overview of the project scope, and made it easier to make issues to help track what remained.

#### **Pull Requests**

Using pull requests was important for making sure the entire team was satisfied with the code written and the product developed. The process of using pull requests would sometimes lead to slower development as someone else had to review the code before it was accepted into the main development branch. It did however also help the team members keep insight into every part of the project's code base. The team is happy with the decision to make pull requests mandatory for merging with the main development branch, as it has kept the code quality higher, and helped create a product the team members are proud of.

## **6. Conclusion and further work**

### **6.1 Conclusion**

The combination of Flutter and Spring Boot helped the team develop a complete solution to the client's problem.

As of May 1st the app is deployed to TestFlight for Apple devices and on Google Play Store for Android devices. The server is a REST API, with a MySQL database, hosted on a Digital Ocean Droplet. Docker is used to set up the server and database so that it can be quickly deployed and updated. The product in its entirety solves the clients problem of struggling to manage the boats inventory. Using the app the client is able to move away from the solution of using Excel on a stationary computer. The client was able to use the app for a trip, and has said they are impressed and pleased with the final product.

The app displays a list of equipment with the option of adding or removing stock. Allowing the client to easily keep track of the inventory. Another list displays all the missing equipment, which can be sent by email to the office to make restocking easier and more accurate. By sending images of the order bills received to the respective departments the employees are able to double check that the shipment they have received aligns with the bill the office has to pay.

The team is satisfied with having followed the agile methodology. It has made the team adaptable to changes and problems faced during development. Furthermore, the team is content having used Jira and Confluence. It is useful to gain experience with tools which are used commonly in the industry. However, for this team specifically the tools were sometimes more time consuming than helpful.

## 6.2 Further work

To simplify making future improvements, both the frontend and backend have been developed with a focus on high quality, maintainable code. This includes writing descriptive variable and method names, good documentation and following the DRY-principles.

While regularly using the app the client could discover new features that could prove useful. Hence, for further development, it would be appropriate to maintain a dialog with the client.

Some improvements to the current product have been mentioned in chapter 5. The team recommends that improving ease of use with better error messages, and further improving offline functionality should be prioritised.

# Societal impact

## Ethical aspects

As apps become a normal part of everyday life, it becomes clear that the ethical aspects need to be considered more. This is something large corporations know the average user normally does not consider, and take advantage of this using misleading statements and hidden meanings. From an article in the New York Times we see an example where a woman's location is tracked by an app on her phone, and then sold the data without her knowledge[37]. This case stands as an example of what many apps on the market are trying to do to earn money.

The intent of the app developed during this project is clear, and never tries to mislead the user. The data gathered is only used within the app and is used for purely functional purposes.

The app's device permissions include access to the device's location, which is only used for generating map reports of item usage, and access to the camera is only used for capturing and uploading images of the order bills which are to be sent for confirmation. The app never uses any of these permissions for a hidden feature, or tracks, and gathers the data for sharing.

## Economic effects

For the client, the use of the app will help streamline the process of tracking what, when, where, and by whom, each item from the boat's inventory is used. This allows the crew to spend less time logging this using their old system, and gives them more time to use on productive value-gaining work. This can result in a better result for the client for each trip.

Using the app will also make it easier to keep an accurate log of the stock, letting them order just the items they need when restocking. This will save the client money normally spent on ordering too much of different items.

The map reports gives the possibility for the client to see where and when different equipment is used, and can help them discover vulnerabilities and inefficiencies within their working process. This can allow for less usage of various equipment, which can help the client save money.

## **Relation to UN's sustainable development goals**

### **Goal 8: Promote inclusive and sustainable economic growth, employment and decent work for all**

The app's goal is to increase the productivity of the crew on-board the boat. This lines up with UN's 8th sustainability goal; Decent work and economic growth. Looking at one of the targets for this goal; 8.2 Achieve higher levels of economic productivity through diversification, technological upgrading and innovation, including through a focus on high-value added and labour-intensive sectors [38], the app focuses on using technological upgrading and innovation to increase the productivity of the company and thereby help increasing the economic growth within the company.

### **Goal 12: Ensure sustainable consumption and production patterns**

As the app's map functionality can help the client find overuse of equipment, it can help contribute to UN's 12th sustainability goal; Ensure sustainable consumption and production patterns, and more specifically its target 12.5; Substantially reduce waste generation through prevention, reduction, recycling and reuse. When optimizing the use of the equipment, the usage will be reduced, which will reduce the waste generated by needing to order and dispose of as much equipment.

# Bibliography

- [1] *Object-oriented programming*. Mar. 23, 2022. URL: [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented\\_programming](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_programming) (visited on Apr. 7, 2022).
- [2] *Coupling and Cohesion*. URL: [https://home.adelphi.edu/sbloch/class/adages/coupling\\_cohesion.html#:~:text=%22%20Coupling%20%22%20describes%20the%20relationships%20between,via%20a%20reduction%20in%20coupling](https://home.adelphi.edu/sbloch/class/adages/coupling_cohesion.html#:~:text=%22%20Coupling%20%22%20describes%20the%20relationships%20between,via%20a%20reduction%20in%20coupling). (visited on Apr. 7, 2022).
- [3] Robert C. Martin. *Clean Code*. Pearson, 2009, pp. 140–141. ISBN: 9780132350884.
- [4] *Design patterns*. wikipedia.org. Feb. 25, 2012. URL: [https://en.wikipedia.org/wiki/Software\\_design\\_pattern](https://en.wikipedia.org/wiki/Software_design_pattern) (visited on Mar. 9, 2022).
- [5] *Observer and observable design pattern*. URL: <https://refactoring.guru/design-patterns/observer> (visited on Apr. 7, 2022).
- [6] *Singleton design pattern*. URL: <https://www.javatpoint.com/singleton-design-pattern-in-java> (visited on Apr. 7, 2022).
- [7] *Unit testing*. wikipedia.com. Feb. 26, 2022. URL: [https://en.wikipedia.org/wiki/Unit\\_testing](https://en.wikipedia.org/wiki/Unit_testing) (visited on Apr. 7, 2022).
- [8] *What is Code Review?* URL: <https://smartbear.com/learn/code-review/what-is-code-review/#:~:text=Code%20Review%2C%20also%20known%20as,like%20few%20other%20practices%20can>. (visited on Apr. 7, 2022).
- [9] *What is Universal Design*. universaldesign.ie. 2020. URL: <https://universaldesign.ie/what-is-universal-design/> (visited on May 10, 2022).
- [10] *Breaking Down the Principles of Design*. toptal.com. 2019. URL: <https://www.toptal.com/designers/gui/principles-of-design-infographic#:~:text=There%20are%20twelve%20basic%20principles,that%20make%20sense%20to%20users>. (visited on Apr. 8, 2022).
- [11] *Don't Repeat Yourself*. Apr. 11, 2022. URL: [https://en.wikipedia.org/wiki/Don%27t\\_repeat\\_yourself](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself) (visited on Apr. 22, 2022).
- [12] *Language Services*. 2020. URL: <https://www.gala-global.org/knowledge-center/about-the-industry/language-services> (visited on Apr. 25, 2022).
- [13] *What is Agile?* URL: <https://www.agilealliance.org/agile101/> (visited on Apr. 7, 2022).



## BIBLIOGRAPHY

---

- [14] *What is git?* URL: <https://www.atlassian.com/git/tutorials/what-is-git> (visited on Apr. 27, 2022).
- [15] *What are RESTful web services?* URL: <https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html> (visited on Apr. 8, 2022).
- [16] *What are cloud services?* redhat.com. URL: <https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services#:~:text=Cloud%20services%20are%20infrastructure%2C%20platforms,to%20users%20through%20the%20internet>. (visited on Apr. 7, 2022).
- [17] *What is a Relational Database (RDBMS)?* oracle.com. URL: <https://www.oracle.com/database/what-is-a-relational-database/> (visited on Apr. 7, 2022).
- [18] *Queue data structure.* URL: [https://en.wikipedia.org/wiki/Queue\\_\(abstract\\_data\\_type\)](https://en.wikipedia.org/wiki/Queue_(abstract_data_type)) (visited on Apr. 8, 2022).
- [19] *Race condition.* Mar. 2, 2022. URL: [https://en.wikipedia.org/wiki/Race\\_condition](https://en.wikipedia.org/wiki/Race_condition) (visited on Apr. 8, 2022).
- [20] *Containerization.* URL: <https://www.ibm.com/cloud/learn/containerization> (visited on Apr. 7, 2022).
- [21] *Barcode.* shopify.com. URL: <https://www.shopify.com/encyclopedia/barcode#:~:text=Barcodes%20are%20applied%20to%20products,accounting%2C%20among%20many%20other%20uses>. (visited on Apr. 7, 2022).
- [22] *Latitude/Longitude Distance Calculator.* URL: <https://www.nhc.noaa.gov/gccalc.shtml> (visited on Apr. 22, 2022).
- [23] *Authentication and authorization.* URL: <https://www.bu.edu/tech/about/security-resources/bestpractice/auth/> (visited on Apr. 7, 2022).
- [24] *What is hashing?* URL: <https://www.educative.io/edpresso/what-is-hashing> (visited on Apr. 7, 2022).
- [25] *What Are Access Tokens?* Oct. 8, 2009. URL: [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc759267\(v=ws.10\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc759267(v=ws.10)) (visited on May 10, 2022).
- [26] *Mockplus.* URL: <https://www.mockplus.com/> (visited on May 9, 2022).
- [27] *Color palette.* URL: <https://colors.co/13293d-006494-247ba0-1b98e0-e8f1f> (visited on Apr. 8, 2022).
- [28] *Spring Boot.* URL: <https://spring.io/projects/spring-boot> (visited on Apr. 7, 2022).
- [29] *Why Spring Security.* Apr. 15, 2021. URL: <https://auth0.com/blog/spring-security-overview/#Why-Spring-Security-> (visited on Apr. 7, 2022).
- [30] *Why You Should Use Bcrypt to Hash Stored Passwords.* Sept. 9, 2011. URL: <https://www.sitepoint.com/why-you-should-use-bcrypt-to-hash-stored-passwords/> (visited on Apr. 7, 2022).
- [31] *Digital Ocean Spaces.* URL: <https://www.digitalocean.com/products/spaces> (visited on Apr. 22, 2022).

## BIBLIOGRAPHY

---

- [32] *Why Docker?* URL: <https://www.docker.com/why-docker/> (visited on Apr. 25, 2022).
- [33] *MediaQueryData class*. URL: <https://api.flutter.dev/flutter/widgets/MediaQueryData-class.html> (visited on May 11, 2022).
- [34] *SeaStorage on Google Play*. URL: [https://play.google.com/store/apps/details?id=no.ntnu.idata.shiporganizer.ship\\_organizer\\_app&gl=NO](https://play.google.com/store/apps/details?id=no.ntnu.idata.shiporganizer.ship_organizer_app&gl=NO) (visited on May 5, 2022).
- [35] *Spring Boot - Best Practices*. URL: <https://www.javaguides.net/2019/03/spring-boot-best-practices.html> (visited on May 3, 2022).
- [36] *Universell utforming*. wikipedia.com. Mar. 14, 2019. URL: [https://no.wikipedia.org/wiki/Universell\\_utforming](https://no.wikipedia.org/wiki/Universell_utforming) (visited on May 5, 2022).
- [37] Michael H. Keller Jennifer Valentino-DeVries Natasha Singer and Aaron Krolik. *Your Apps Know Where You Were Last Night, and They're Not Keeping It Secret*. Dec. 10, 2018. URL: <https://www.nytimes.com/interactive/2018/12/10/business/location-data-privacy-apps.html> (visited on May 6, 2022).
- [38] *Economic Growth*. URL: <https://www.un.org/sustainabledevelopment/economic-growth/> (visited on May 6, 2022).

## **A. Project Plan**

---

## **Gruppe 4, Oppgave 8**

### **Ship Organizer Forprosjektplan**

**Versjon <1.0>**

---

# Gruppe 4, Oppgave 8

## Table of Contents

<b>1. Mål og rammer .....</b>	<b>3</b>
1.1 Orientering .....	3
1.2 Problemstilling / prosjektbeskrivelse og resultatmål .....	3
1.3 Effektmål .....	3
1.4 Rammer .....	3
<b>2. Organisering .....</b>	<b>4</b>
<b>3. Gjennomføring .....</b>	<b>4</b>
3.1. Hovedaktiviteter .....	4
3.2. Milepæler .....	5
<b>4. Oppfølging og kvalitetssikring .....</b>	<b>6</b>
4.1 Kvalitetssikring .....	6
4.2 Rapportering .....	6
<b>5. Risikovurdering .....</b>	<b>7</b>
<b>6. Vedlegg .....</b>	<b>8</b>
6.1 Tidsplan .....	8
6.2 Adresseliste .....	9
6.3 Avtaledokumenter .....	9
6.3.1 Arbeidskontrakt for bachelor-gruppen .....	9
6.3.2 3-partsavtale .....	9

---

# Gruppe 4, Oppgave 8

## 1. Mål og rammer

### 1.1 Orientering

Giske Kystfiske sendte en oppgave beskrivelse til NTNU og den ble publisert på Blackboard. Fra Blackboard valgte vi oppgaven ettersom vi syntes den virket interessant samt utfordrende.

### 1.2 Problemstilling / prosjektbeskrivelse og resultatmål

Giske Kystfiske ønsker en løsning som skal forenkle håndteringen av lager og logistikk ombord i båtene i form av en mobil applikasjon.

Virksomheten opplever at det er lett å miste kontroll og oversikt over beholdningen for ulike komponenter om bord i båten og mellom skiftene. De ser da for seg en mobil applikasjon, som hver ansatt kan bruke for å registrere bruk og tilordning av komponenter hos sitt departement, som en ønskelig løsning på dette problemet.

Virksomheten forteller også om varierende teknisk kompetanse blant de ansatte og ønsker derfor også at applikasjonen skal være enkel å forstå og bruke.

### 1.3 Effektmål

Virksomheten ønsker en mer effektiv og oversiktlig logistikksituasjon, med hensyn til beholdning og bruk av deler om bord i sine båter. Virksomheten ønsker å forenkle hverdagen til seg selv og sine ansatte ved å innføre et enkelt grensesnitt for å holde orden på beholdningen. Det er et mål for virksomheten at de ansatte skal bruke mindre tid på det administrative knyttet til logistikk, sånn at de har mer tid til andre oppgaver, og derfor kan være mer effektive på jobb.

### 1.4 Rammer

Kunden vil antagelig trenge en ekstra server dedikert til appen på hver båt. Dette må til for at appen skal kunne synkronisere mellom enheter uavhengig om den er koblet til internett/den sentrale databasen.

---

## Gruppe 4, Oppgave 8

### 2. Organisering

#### **Utviklingsteam:**

Simon Duggal (Teamleder)

Johannes Løvold Josefsen (Kvalitetssikring)

Hans Andreas Lindgård (Dokumentansvarlig)

#### **Veileder:**

Mikael Tollefsen

#### **Kunde:**

Giske Kystfiske, kontaktperson: Kurt Skjong

### 3. Gjennomføring

#### 3.1. Hovedaktiviteter

Appen skal utvikles av hele gruppen. Vi vil fordele de ulike app-aktivitetene jevnt mellom medlemmer i gruppa. Hvert medlem vil i hovedsak utvikle sine app-aktiviteter, men blir de ferdig tidlig vil han kunne hoppe inn for å hjelpe ett annet medlem, eller bli satt til en ny oppgave. Alle app-aktivitetene blir utviklet i språket Dart med Flutter-rammeverket. Før vi kan begynne å utvikle appen må alle wireframesene være ferdige og akseptert av arbeidsgiver. Dert vil også bli opprettet GitHub-repositories for å samarbeide på koden.

Backend til appen skal bli utviklet av hele gruppa. Vi trenger en server som kan håndtere data og være koblingen mellom appen og databasen. Backend blir skrevet i Java og skal være en Spring-Boot-applikasjon. For at vi skal kunne begynne på backend skal det opprettes en GitHub-repository for å samarbeide på koden, samt at strukturen for databasen skal være planlagt. Når dette målet er fullført skal vi ha en fullstendig funksjonell backend som håndterer alle requests som appen sender.

Rapporten skrives av hele gruppen. Rapporten skrives for å dokumentere alt arbeidet som har blitt gjort i bachelor oppgaven. Rapporten skal skrives i Latex i samarbeidsverktøyet OverLeaf. Vi valgte dette fordi vi føler det er lett å samarbeide i samt gjør det enkelt å lage en rapport som ser profesjonell ut.

---

## Gruppe 4, Oppgave 8

### 3.2. Milepæler

- 13.01.2022: Oppstartsmøte med veileder og arbeidsgiver
- 26.01.2022: Statusmøte med veileder og arbeidsgiver
- 28.01.2022: Forprosjektplan leveres på Blackboard
- 08.02.2022: Demo-versjon av Frontend/App
- 09.02.2022: Statusmøte med veileder og arbeidsgiver
- 23.02.2022: Statusmøte med veileder og arbeidsgiver
- 09.03.2022: Statusmøte med veileder og arbeidsgiver
- 22.03.2022: MVP for backend
- 23.03.2022: Statusmøte med veileder og arbeidsgiver
- 06.04.2022: Statusmøte med veileder og arbeidsgiver
- 19.04.2022: App-utvikling ferdig, inkluderer backend, frontend, og deployment (PlayStore/AppStore)
- 20.04.2022: Statusmøte med veileder og arbeidsgiver
- 22.04.2022: Muntlig presentasjon på engelsk
- 04.05.2022: Statusmøte med veileder og arbeidsgiver
- 06.05.2022: Levere siste utkast av rapport til veileder for tilbakemelding
- 19.05.2022: Oppsummeringsmøte med veileder og arbeidsgiver
- 20.05.2022: Levere endelig rapport i Inspira
- ~20.05.2022: Presentasjon av prosjektet plenum



---

## Gruppe 4, Oppgave 8

### 4. Oppfølging og kvalitetssikring

#### 4.1 Kvalitetssikring

Teamet har en dedikert kvalitetssikringsansvarlig, men hvert enkelt teammedlem har fremdeles ansvar for å levere fra seg arbeid av høyest mulig kvalitet. Pull-requests vil bli brukt ved pushing til kodebasen i GitHub, dette fører til at minst to medlem har sett og godkjent all kode.

Med hensyn til andre viktige dokumenter og arbeider har hvert medlem ansvar for å inkludere resten av teamet for å komme med flere innspill og sikre best mulig kvalitet, og et produkt som hele gruppen er fornøyd med.

#### 4.2 Rapportering

Teamet vil arrangere møte annenhver uke mellom alle parter (team, veileder, og kunde), for å oppdatere om status, plan, og eventuelt diskutere oppdagede problemstillinger for prosjektet.

## Gruppe 4, Oppgave 8

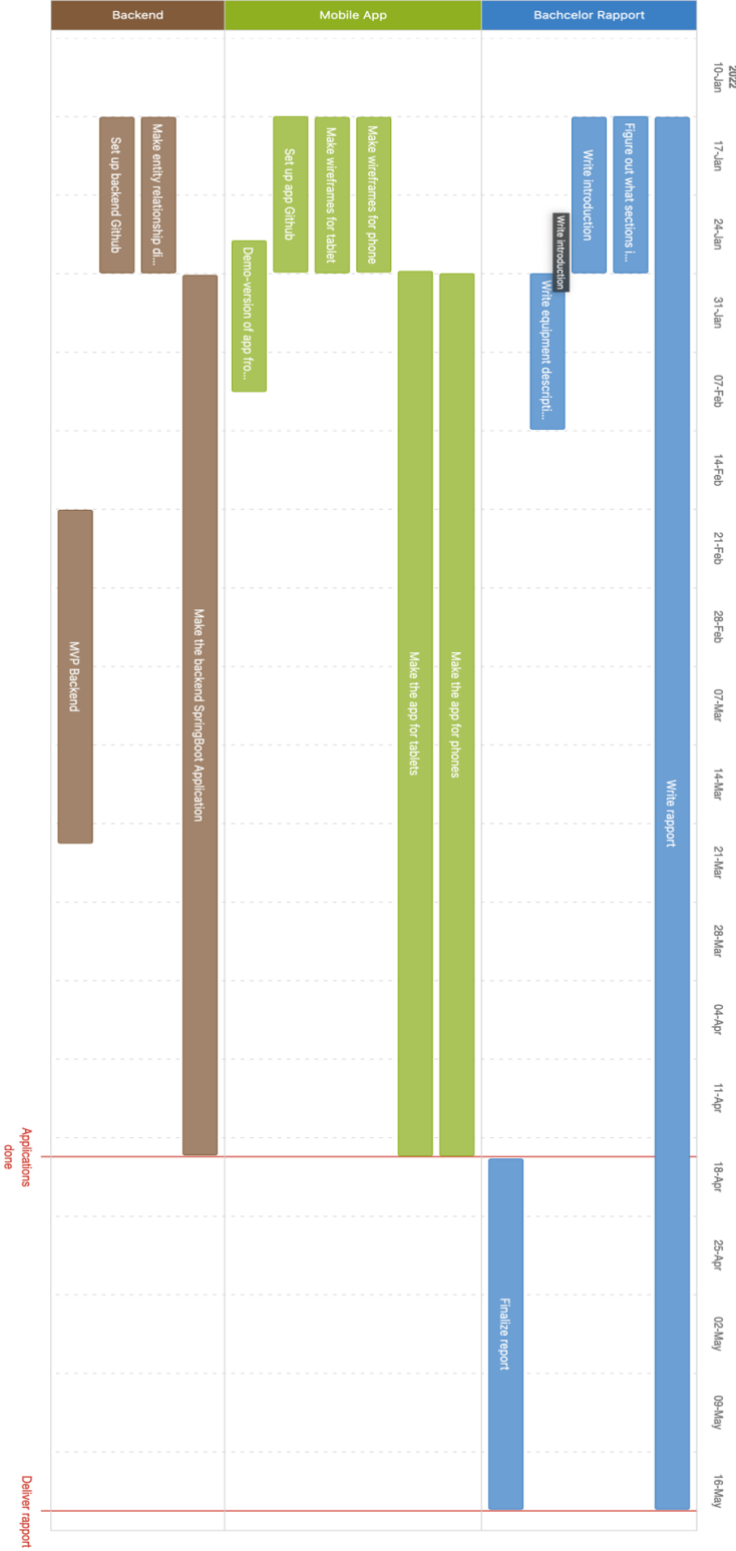
### 5. Risikovurdering

Hendelse	Sannsynlighet (1-5)	Konsekvens (1-5)	Risiko- faktor	Tiltak
Manglende oversikt over tidsplan og nødvendige gjøremål.	2	3	6	Starte tidlig med god planlegging og kartlegging av nødvendige oppgaver.
Dårlig/misledende vurdering av tidsramme for ulike oppgaver.	3	3	9	Være bevisst på hvor lang tid som er tenkt til en oppgave og hvor mye tid som faktisk blir brukt. Være åpen for omprioritering/omfordeling av oppgaver ved store avvik.
Manglende innsikt for optimal prioritering av oppgaver om man ikke rekker alle.	2	4	8	Sørge for å ha god dialog med oppdragsgiver om hvilke funksjoner som burde prioriteres.
Koronavirus fører til vanskeligheter for prosjektarbeid.	3	2	6	Gruppemedlem skal ta hensyn ved tegn til symptom. Gruppen skal legge opp til at medlem i eventuell karantene/isolasjon skal kunne arbeide hjemmefra med digital deltagelse.
Dårlig kommunikasjon innad i gruppen.	1	4	4	Daglige møter hvor medlemmene oppdaterer hverandre om status og plan.
Rekker ikke å bli ferdig med prosjektet grunnet for stor arbeidsmengde.	1	5	5	Sette frister innad i gruppen for deloppgaver som må utføres og sørge for at fristene blir holdt. Være villige til å legge inn noe ekstra arbeid for å komme i mål.
Ikke tilfredsstillende kvalitet på fullført produkt.	2	4	8	Sørge for å ha god dialog med veileder og oppdragsgiver angående produktets kvalitet og hva som forventes av det.

# Gruppe 4, Oppgave 8

## 6. Vedlegg

### 6.1 Tidsplan



---

## Gruppe 4, Oppgave 8

### 6.2 Adresseliste

*Navn, firma, tlf., epost, adresse*

Navn	Firma	Telefon	E-post
Simon Duggal		992 59 192	simondu@ntnu.no
Hans Andreas Lindgård		901 08 618	hansal@ntnu.no
Johannes Løvold Josefsen		412 21 156	johannlj@ntnu.no
Mikael Tollefsen	NTNU, Driw	701 61 340	mikael.tollefsen@ntnu.no
Kurt Skjong	Giske Kystfiske	926 36 079	kurt@maoyi.no

### 6.3 Avtaledokumenter

#### 6.3.1 Arbeidskontrakt for bachelor-gruppen

Se innlevering i Blackboard.

#### 6.3.2 3-partsavtale

Se innlevering i Blackboard.

# B. System Documentation

## Contents

1	Introduction . . . . .	2
2	Architecture . . . . .	2
3	Project structure . . . . .	3
	3.1 Application . . . . .	3
	3.2 Server . . . . .	3
4	Class diagram . . . . .	4
	4.1 Application . . . . .	4
	4.2 Server . . . . .	4
5	Database model . . . . .	5
6	Server services - REST resources . . . . .	5
	6.1 Authentication and authorization resources . . . . .	5
	6.2 User resources . . . . .	6
	6.3 Product resources . . . . .	6
	6.4 Report resources . . . . .	7
	6.5 Order bill resources . . . . .	7
	6.6 Department resources . . . . .	7
7	Security . . . . .	8
	7.1 Token access . . . . .	8
	7.2 Password . . . . .	8
8	Deployment . . . . .	8
	8.1 Application . . . . .	8
	8.2 Server . . . . .	9
9	Documentation of source code . . . . .	9
	9.1 Application . . . . .	9
	9.2 Server . . . . .	9
10	Testing . . . . .	9
11	References . . . . .	10

## 1 Introduction

System documentation describes the code project structure including, the different classes used in both application and server, how the database is setup and the sequence of classes used in a method

## 2 Architecture

The flow chart below shows the structure from mobile device as the start point to the sever being the end point.

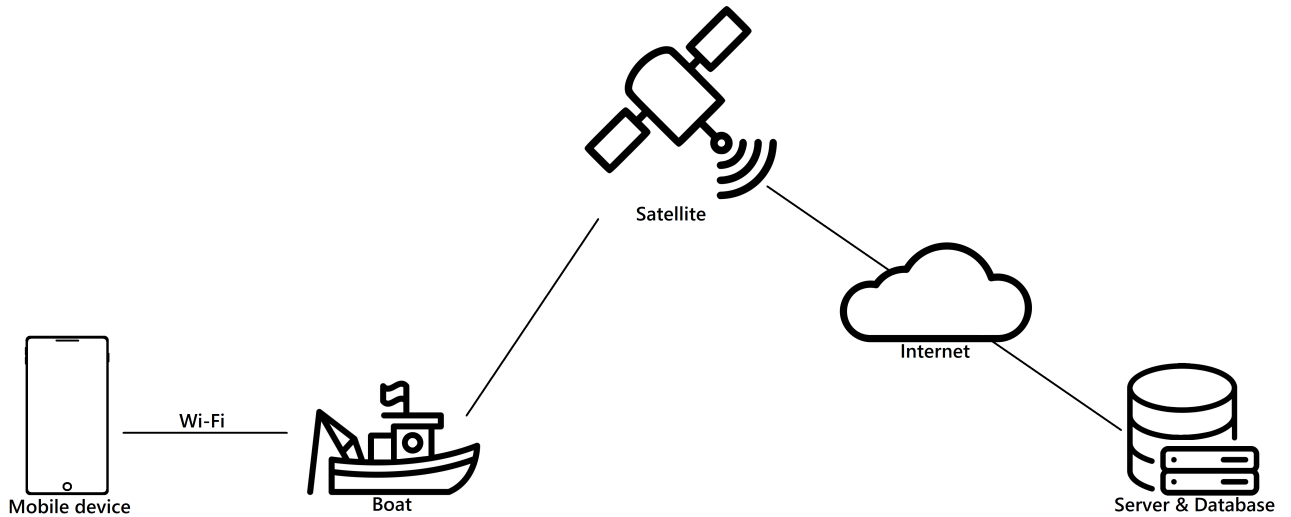


Figure 1: Drawing showing the different components and their communication

## 3 Project structure

### 3.1 Application

The project is structured in different folders on the root level after the initial "lib" folder. These folders represent the content and files used in the application, as shown in the tree structure under

```
lib
├── api handling (Handles all API calls)
├── config (Configuration files)
├── entities (Different entities files)
├── I10n (Language files)
├── offline queue (Handles all offline API calls)
├── views (All view files)
└── widget (Custom widget files)
```

### 3.2 Server

The spring server is divided into three main folders, controller, service and repository. The controller is the endpoint for the applications API call and decodes the JSON request. Then the service receives the decoded JSON object and if necessary modifies the data and make a method call to the repository to fetch, insert or update the data in the database.

```
src
├── java
│   ├── auth (Authentication filter for security config)
│   ├── config (Configuration files)
│   ├── controller (Different controller files for all the possible API calls)
│   ├── model (Entity class files)
│   ├── repository (Handles calls to database )
│   ├── service (Used as buffer between controller and repository to handle ad modify data)
│   └── userprinciple (Provides UserDetails for the user principle)
├── resources
│   └── application properties
```

## 4 Class diagram

### 4.1 Application

The mobile application has 21 view classes and 17 support classes ranging from configuration files and custom widgets.

### 4.2 Server

The spring boot server contains 38 working classes, 1 main class to start the application and 6 test class. The 39 working classes are distributed into controller, service and repository classes. The controller is the first step for the server and the endpoint for the mobile application. Here the request is picked up and the JSON object is decoded, so the call to the service is with the correct parameters. The service modifies the data if necessary and make a call to the repository to handle the call to the procedure directly in the database. The respond from the database is sent back to the service with the creation of entities object when needed, where then the service makes modifications if necessary and then passes the data to the controller to make the responds for the mobile application.

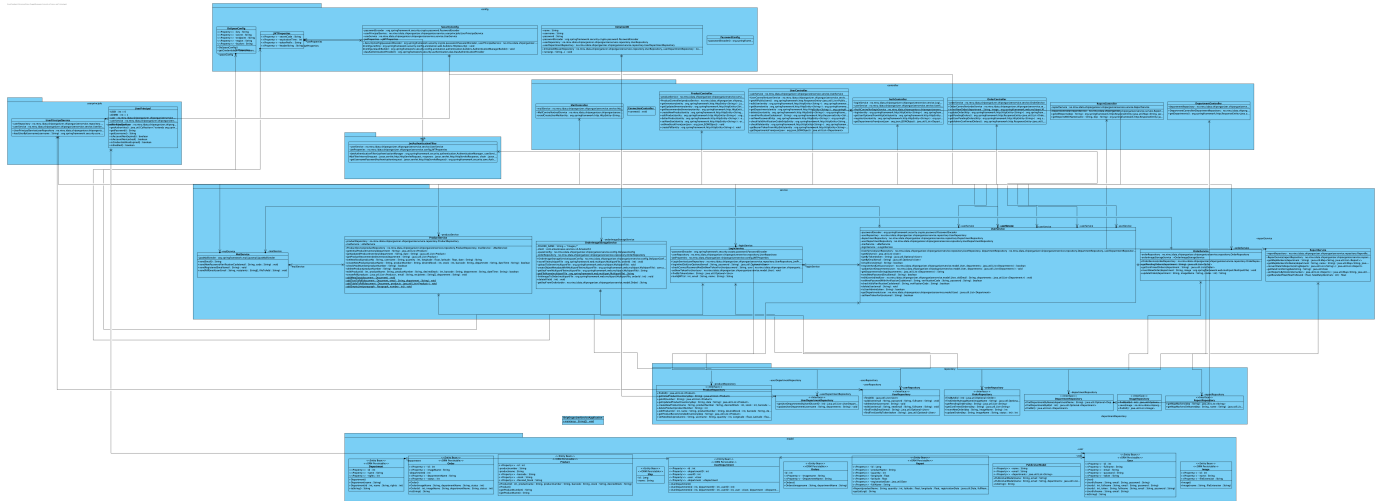


Figure 2: Class diagram for the backend server. See attached image in separate file for larger, more readable class diagram.



## 5 Database model

Figure 3 shows the relation and table structure of the database. The database contains a set of procedures used to collect, insert and update the different tables. Most of the tables has a relation to another, by using an Many-to-One relation.

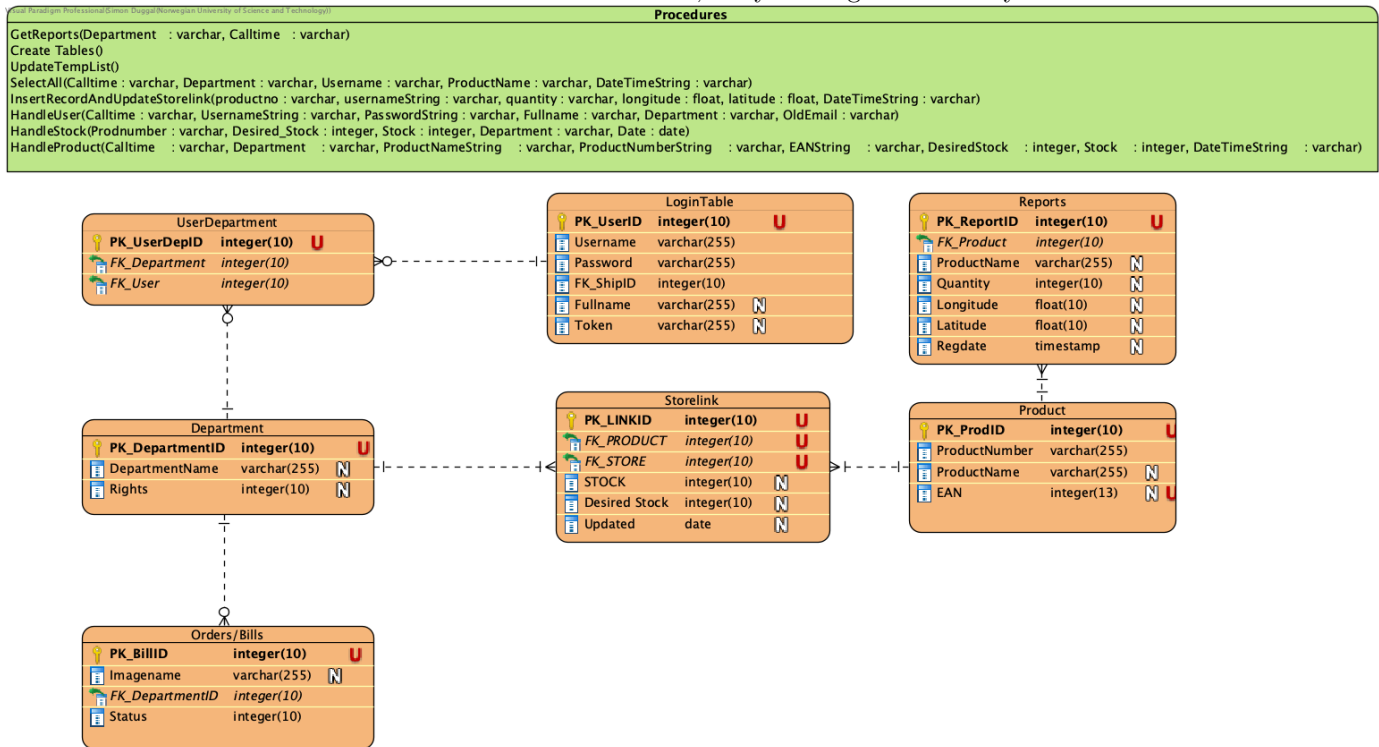


Figure 3: Diagram showing the relation between the tables in the database

## 6 Server services - REST resources

### 6.1 Authentication and authorization resources

#### Registration

*/auth/register* - This endpoint serves the registration resource. To access this, a valid bearer token from an admin user must be included. It allows an admin to register new users to the system, with their specified departments, name and email-address, so that the user can set a password for themselves to start using the system.

#### Login

*/auth/login* - This endpoint serves the login resource. The user can POST a JSON object containing an email-address and password, and receive either a valid token related to the already registered user, or an error response.

## 6.2 User resources

### List of all users

*/api/user/all-users* - Gets a list of all users in the system. Admin rights needed.

### Edit user

*/api/user/edit-user* - Lets an admin edit user information of the users.

### Get name

*/api/user/name* - Allows an authenticated user to get their full name.

### Get departments

*/api/user/departments* - Allows an authenticated user to get a list of the departments they belong to.

### Delete user

*/api/user/delete-user* - Allows an authenticated user to delete themselves from the system, or an admin to delete any user.

### Send verification code

*/api/user/send-verification-code* - Sends a verification code to the specified email, to be used for changing password.

### Set password

*/api/user/set-password* - Allows setting a new password for a user using a valid verification code.

### Check verification code

*/api/user/check-valid-verification-code* - Used for checking the validity of a verification code to an email.

### Check role

*/api/user/check-role* - Lets an authenticated user get their role in the system.

## 6.3 Product resources

### Get inventory

*/api/product/get-inventory* - Gets the inventory for the specified department.

### Get recently updated inventory

*/api/product/recently-updated-inventory* - Gets inventory list of items updated after specified time for specified department.

### Get recommended inventory

*/api/product/get-recommended-inventory* - Gets a list of the recommended amount of each item in the inventory to be ordered, to reach the desired stock for the specified department.

### Create new product

*/api/product/new-product* - Allows a user to create a new product to include in the inventory for the specified department.

### **Edit product**

*/api/product/edit-product* - Lets a user edit information of a product.

### **Delete product**

*/api/product/delete-product* - Lets a user delete a specified product from the inventory.

### **Set new stock**

*/api/product/set-new-stock* - Lets a user update how many of a product to add to, or remove from, the stock. Stores a report of removals to the system.

### **Send PDF of missing items**

*/api/product/create-pdf* - Sends a list in the form of a PDF to the specified email addresses of the items missing to reach the desired stock of the specified department.

## **6.4 Report resources**

### **Get all reports for department**

*/reports/all-reports={department name}* - Gets all usage reports for the specified department.

### **Get reports for product in department**

*/reports/reports-with-name={product name}-dep={department name}* - Gets the reports for specified product in specified department.

## **6.5 Order bill resources**

### **Add new order confirmation**

*/orders/new* - Adds a new order for confirmation with uploaded image to specified department.

### **Update order confirmation**

*/orders/update* - Updates a pending order to the specified status.

### **Get all pending order confirmations**

*/orders/admin/pending* - Lets an admin get all the pending order confirmations.

### **Get pending order confirmations for department**

*/orders/user/pending* - Lets a user get the pending order confirmations for the specified department.

### **Get completed order confirmations**

*/orders/confirmed* - Lets an admin get all completed order confirmations.

## **6.6 Department resources**

### **Get all departments**

*/api/department/get-all* - Gets a list of all departments in the system.

## 7 Security

### 7.1 Token access

Every user gets assigned an token upon creation of the user. When the user is logged in the user gets delivered the token to use as authentication bearer token. This token is only valid in 3 weeks at a time or until the user logs into the application. The token is different from a regular user and the administrator user, and this determines the different API calls the user have access to.

### 7.2 Password

The password is hashed using the bcrypt [**bcrypt**] hashing algorithm. The algorithm gives the possibility to modify the complexity of the password depending on the computers speed. Only the encrypted password is stored in the database, so for an outsider to be able to read an users password, they have to have knowledge of the specific algorithm used.

## 8 Deployment

### 8.1 Application

The application is available to both Android and iOS. The Android version are online on Google Play Store, while the iOS version is only available to the users using TestFlight. TestFlight is Apples test framework and every user needs to be added by the application administrator to be able to access the download site. The following code libraries are used in the app.

- **flutter\_localizations** A library used for localization. Entails generating translation files.[1]
- **multi\_select\_flutter** A widget to select multiple items at once.[2]
- **json\_annotation** Used to create code for JSON serialization and deserialization.[3]
- **google\_maps\_flutter** A library for displaying an interactive map. [4]
- **permission\_handler** A library used to ask for permissions and check their status.[5]
- **flutter\_riverpod** State-management library. [6]
- **dio** Http client for dart. Used to do network calls.[7]
- **flutter\_secure\_storage** Library used to securely store data. Mainly used to persist data between sessions.[8]
- **fluttertoast** Library used to show short messages (toasts) to the user. Usually used to show error messages.[9]
- **flutter\_launcher\_icons** Library used to set the app icons, e.g. the ones the user presses to launch the app.
- **cupertino\_icons** Library containing default icons.[10]
- **image\_picker** A library to either select a picture from image library or take a new one.[11]
- **flutter\_barcode\_scanner** A library to scan barcodes on both Android and iOS.[12]

- **connectivity** Library used to check if the user is online or not.
- **geolocator** A Flutter library used to get specific location.[13]

### 8.2 Server

The server is deployed at Digital Ocean as a droplet using Docker. Docker supports the creation of a compose file where there is possible to create an spring boot container as well as an MySQL database. The Digital Ocean server gets an IP address which the application uses to communicate with the server. To deploy the server in another deployment service the compose file needs to be run, and a set of creation procedures found in a folder sql/procedures.sql in the source code needs to be executed in the database. SpringBoot comes with many features, the list below outlines other code libraries used.

- **mysql-connector-java** Library used to connect to database.
- **spring-boot-starter-mail** The library used to send email in Spring.
- **java-jwt** Used to generate jwt. Essential for security in the app.
- **itextpdf, pdfbox** Library used for creating a PDF. Needed to send a nicely formatted order bill.
- **aws-java-sdk-core, aws-java-sdk-s3, commons-collections4, commons-io** Used for processing images and uploading them to the image server.

## 9 Documentation of source code

### 9.1 Application

The code in the frontend app is documented in-line with how to document in Flutter. Each method is given a descriptive name but also documented using `///` notation. There are also smaller comments with `///` notation inside methods to describe how some more complex lines of code work.

### 9.2 Server

The whole SpringBoot server is documented in detail using Javadoc. It is possible to use a built in IntelliJ feature to generate and show the documentation. This can be done by going to Tools/Generate Javadoc and selecting a valid destination folder. This creates an HTML file containing all the documentation of the server.

## 10 Testing

The server contains classes delegated to testing the main features. The app is tested publicly by having it deployed to clients devices. The server has a configured Docker file, which makes it easy to rerun the server with any updates. The team has created unit tests to test some of the smaller pieces of the code.

## 11 References

- [1] Author: flutterando.com.br <https://pub.dev/packages/localization>, Visited on 09.05.2022
- [2] Author: CHB61, [https://pub.dev/packages/multi\\_select\\_flutter](https://pub.dev/packages/multi_select_flutter), Visited on 09.05.2022
- [3] Author: google.dev, [https://pub.dev/packages/json\\_annotation](https://pub.dev/packages/json_annotation), Visited on 09.05.2022
- [4] Author: flutter.dev, [https://pub.dev/packages/google\\_maps\\_flutter](https://pub.dev/packages/google_maps_flutter), Visited on 09.05.2022
- [5] Author: baseflow.com, [https://pub.dev/packages/permission\\_handler](https://pub.dev/packages/permission_handler), Visited on 09.05.2022
- [6] Author: dash-overflow.net, <https://pub.dev/packages/riverpod>, Visited on 09.05.2022
- [7] Author: flutterchina.club, <https://pub.dev/packages/dio>, Visited on 09.05.2022
- [8] [https://pub.dev/packages/flutter\\_secure\\_storage](https://pub.dev/packages/flutter_secure_storage), Visited on 09.05.2022
- [9] Author: karthikponnam.dev, <https://pub.dev/packages/fluttertoast>, Visited on 09.05.2022
- [10] Author: flutter.dev, [https://pub.dev/packages/cupertino\\_icons](https://pub.dev/packages/cupertino_icons), Visited on 09.05.2022
- [11] Author: flutter.dev, [https://pub.dev/packages/image\\_picker](https://pub.dev/packages/image_picker), Visited on 09.05.2022
- [12] [https://pub.dev/packages/flutter\\_barcode\\_scanner](https://pub.dev/packages/flutter_barcode_scanner), Visited on 09.05.2022
- [13] Author: baseflow.com, <https://pub.dev/packages/geolocator>, Visited on 09.05.2022

# C. Requirements Documentation

## Contents

1	Introduction . . . . .	2
2	Use Case diagram . . . . .	2
3	User stories . . . . .	3
3.1	Story 1 . . . . .	3
3.2	Story 2 . . . . .	3
3.3	Story 3 . . . . .	4
3.4	Story 4 . . . . .	4
3.5	Story 5 . . . . .	4
3.6	Story 6 . . . . .	4
3.7	Story 7 . . . . .	5
3.8	Story 8 . . . . .	5
3.9	Story 9 . . . . .	5
3.10	Story 10 . . . . .	5
3.11	Story 11 . . . . .	6
3.12	Story 12 . . . . .	6
3.13	Story 13 . . . . .	6
3.14	Story 14 . . . . .	6
3.15	Story 15 . . . . .	7
3.16	Story 16 . . . . .	7
4	Domain model . . . . .	7
4.1	Sequence diagram . . . . .	8
5	Wireframe . . . . .	8
5.1	Phone . . . . .	8
5.2	Tablet . . . . .	13

# 1 Introduction

This document contains the different requirements and specifications for the app made in this project.

# 2 Use Case diagram

Use case diagram displayed in 1 show the different possibility the a normal user has compared to the possibility an administrator has shown in 2

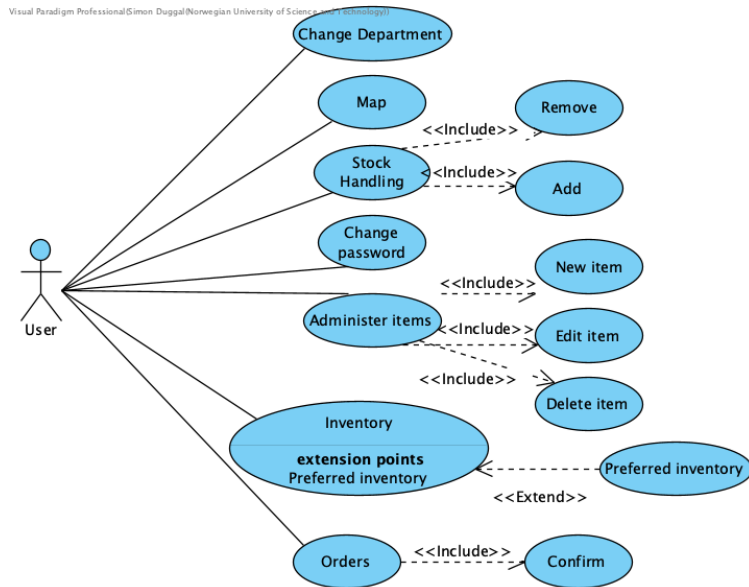


Figure 1: Use case diagram for a regular user



Visual Paradigm Professional(Simon Duggal@Newcastle University of Science and Technology)

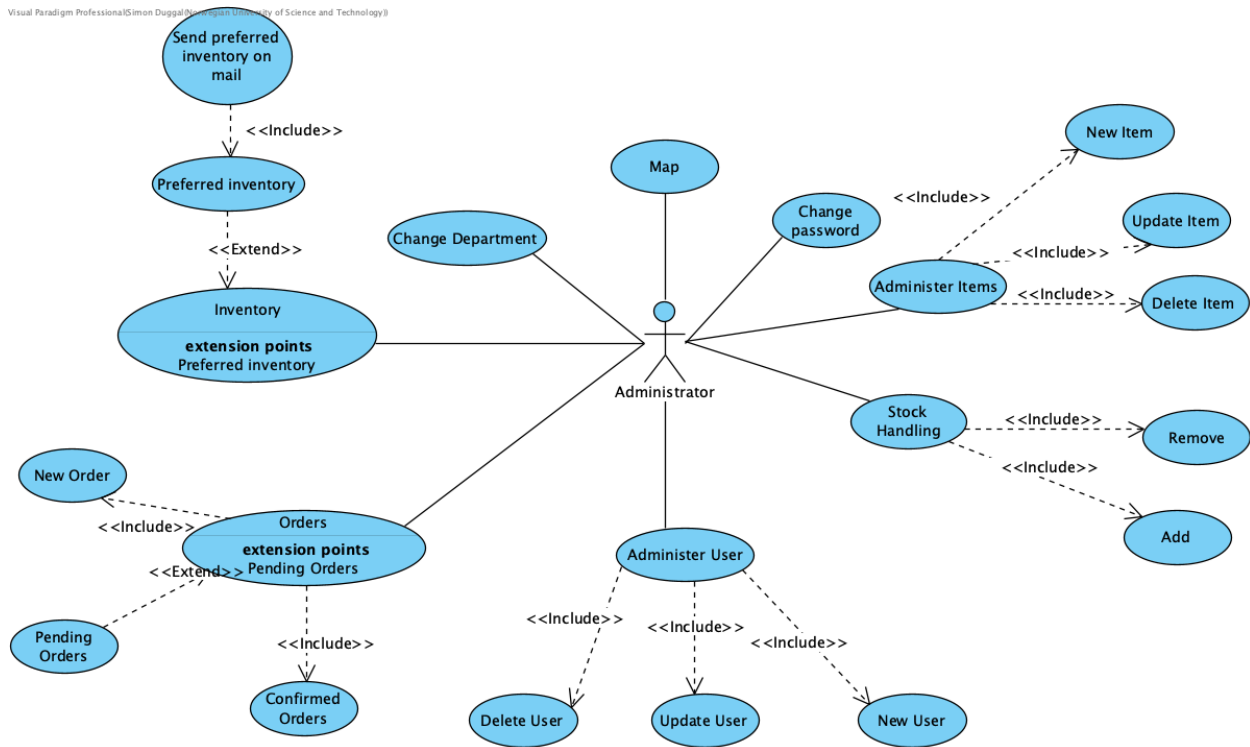


Figure 2: Use case diagram for an administrator user

### 3 User stories

The stories are in no specific order.

#### 3.1 Story 1

As a user, I would like to update only the items in inventory which have been modified after the last time they were fetched, so that the data usage is reduced.

- Send time to database when refreshing inventory.
- Database must store when item was last updated.
- Store items locally so that not all are required to be fetched.

#### 3.2 Story 2

As a user I want to change the language on the app in case the default is different from what i want so that the app is easier to use.

- Have translation files. One for each language to support.
- I MyAccount view have an icon which can be pressed.
- When pressing icon display a list of languages which can be pressed, this changes the language in the app.
- All displayed text needs to be translated.

- All error messages must be translated.

### 3.3 Story 3

As an admin I want the possibility to select department, so that I can see everything from that department.

- Have a button to select department. Not shown if only has one department and not admin.
- When selecting a department show all departments if admin.
- Swapping department swaps the cached inventory.

### 3.4 Story 4

As an admin I want to create new users, so that I can delegate the new user's department.

- A button to access create user.
- A new user needs to have a set of details. Name, email, and departments.
- Need to be able to select multiple departments.
- Need to get feedback if email is invalid format.
- Must get error message if user email already exists.

### 3.5 Story 5

As a user I want to check the map, so that I can see where we have used equipment and where we have used the most equipment.

- Have a map view.
- Have markers on the map to show that equipment has been used.
- Have the markers be clickable to see more details.
- The markers show a drawer with accurate details when pressed.
- Markers are colour coded so that its easy to see where the most items have been used.
- Markers that should be close together are grouped together to reduce visual clutter and increase performance.

### 3.6 Story 6

As a user I would like to log in and stay logged in, so that I don't have to log in each time I open the app.

- A view where the user can log in.
- Feedback if the user cannot log in.
- Use token based logging in.
- Check token when opening the app to keep the user logged in.
- If the token has expired log the user out.

### 3.7 Story 7

As a user I want to be able to change my password, so that if I forget it I can still log in without getting a new account.

- Have a set password view.
- Send verification code to users email.
- User can enter verification code, this is checked.
- Can enter password and confirm password. The passwords must match to confirm the user has entered what they think they have.
- Must be able to change password even if logged in.

### 3.8 Story 8

As a user I want to add or remove from the inventory whenever I use some equipment and I want it to be updated for the rest of the crew, so that the crew can easily track the inventory onboard.

- Have list view showing whole inventory for department.
- Have minus and plus icon buttons on each equipment.
- Pressing either plus or minus shows a popup window where they can enter how much to add/remove.
- The equipment has the amount updated automatically when add or remove is used.
- Add and remove updates the stock in the database so other users also see the changed stock.
- Create a record of when the stock is updated to contribute to low data usage.

### 3.9 Story 9

As an admin I want there to be created records whenever equipment is removed so that it is easy to see when, where and who equipment was used.

- Create record in database with the user who used equipment. Include LatLng, date-time, and user's name.
- Show the details of this in the map.

### 3.10 Story 10

As an admin I want to edit existing users, so that they can be deleted or modified if they quit, or need access to other departments.

- A list view with all users.
- An edit button on each user.
- A view for showing the details of the specific user.
- Need to change and enter text in those text boxes.
- Can select departments the user should have access to.

- An update button to push the changes to the server.
- A back button to cancel the changes.
- A delete button to delete the user.

#### 3.11 Story 11

As a user I want to be able to edit existing products, so that if they are wrong I can change them.

- A view with a list of equipment.
- Each equipment has an edit button.
- A view showing the details of the pressed item.
- The option of editing product number, desired stock and product name.
- Confirming the edit button.
- Back button so changes aren't pushed to server.
- A delete button to delete this piece of equipment.

#### 3.12 Story 12

As an admin I want to send pictures of order bills to specific departments, so that they can confirm they have received what the bill charges for.

- A view for taking picture and selecting department that is receiving the bill.
- A view for seeing pending bills.
- A view for seeing confirmed bills.
- The department user must see pending bills, and have button to confirm them.

#### 3.13 Story 13

As a user I want to see the missing inventory and send it to the office so they can order more of the missing equipment.

- A view displaying each item with the amount that's missing.
- A button to send the missing inventory.
- Only send the items which have a missing amount.
- Enter the emails of the recipients.
- Send an email with a PDF to recipients so they can order missing equipment.

#### 3.14 Story 14

As a user I want to be able to search for equipment, so that its faster to find it.

- A search bar in the inventory views.
- Search the shown equipment only so it is fast.

### 3.15 Story 15

As a user I want to be able to search using barcode, so that the search is fast, easier and I do not have to type.

- A barcode scanner which enters the scanned barcode into the search.
- Search automatically after the barcode is scanned.

### 3.16 Story 16

As a user I want to add new pieces of equipment to the inventory, so that when we receive new equipment the app can give an overlook over the full inventory.

- A view for adding new equipment.
- Need to be able to enter product name, number, current stock, desired stock and scan/type in barcode.
- Able to cancel creation.
- Able to submit new product and have it added to the database and be viewed in the inventory views.

## 4 Domain model

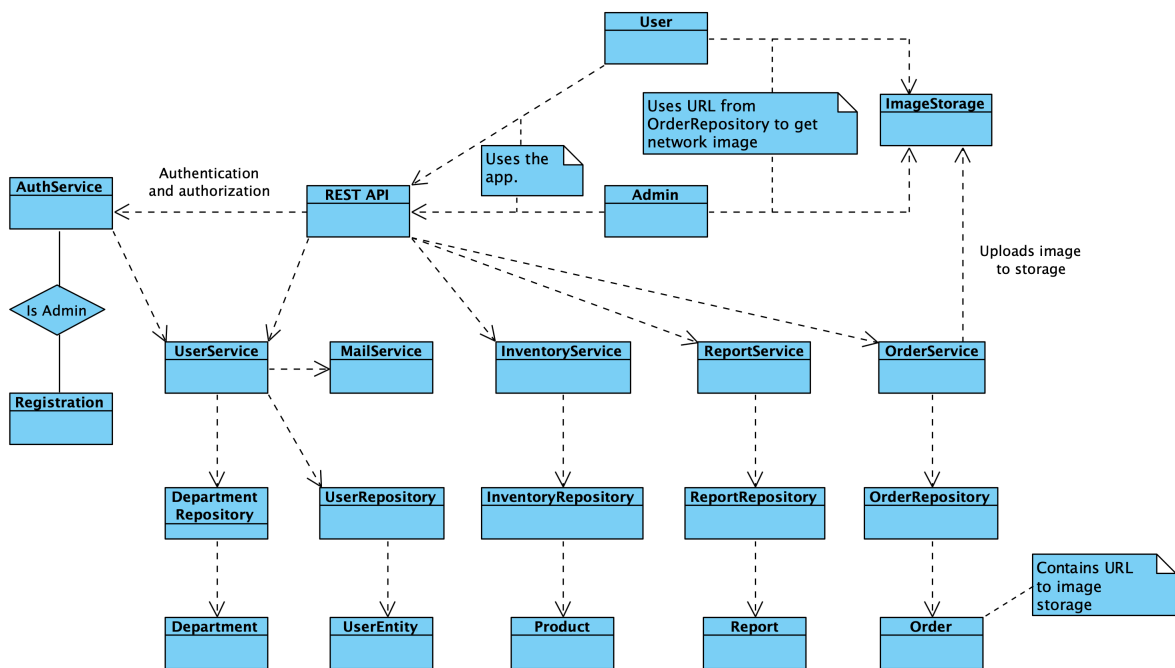


Figure 3: Domain model

### 4.1 Sequence diagram

The sequence diagram show the servers action when a the call to set a new stock for a product is made. The call enters the product controller then proceeds to the product server then to the product repository to make a final call to the procedure in the database.

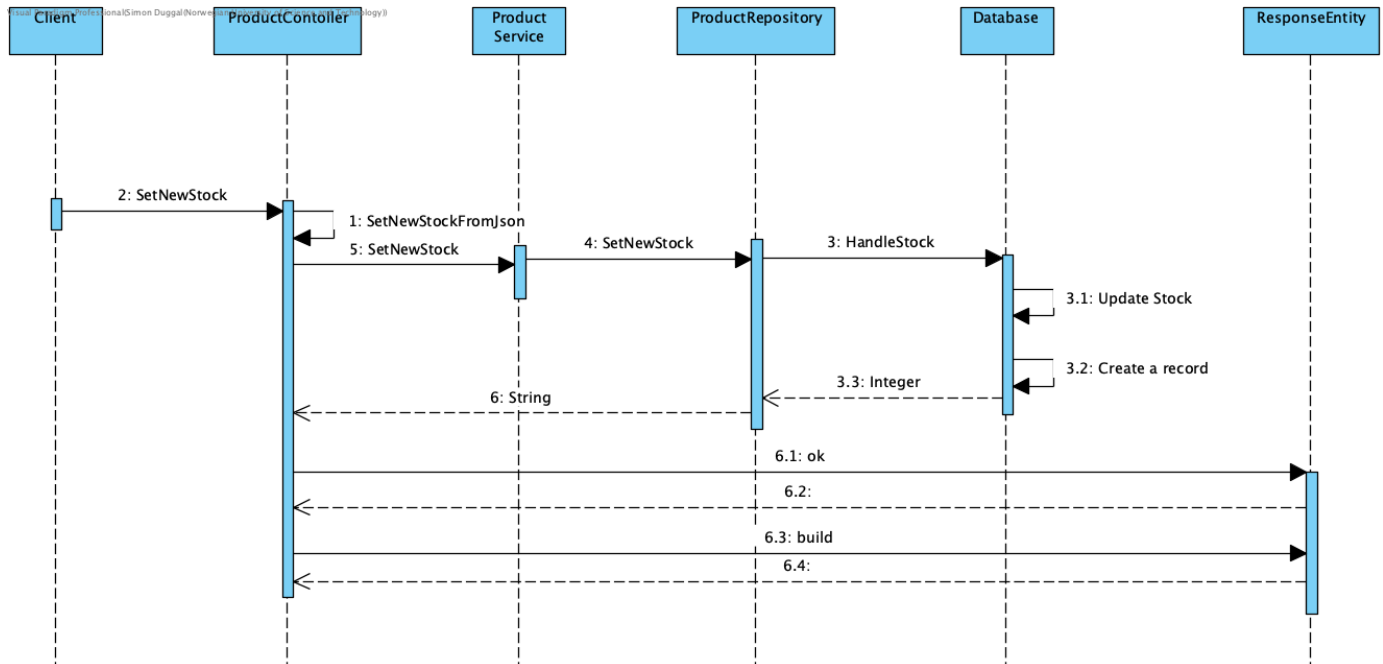


Figure 4: Sequence diagram the method setNewStock

## 5 Wireframe

### 5.1 Phone



Water Inc.

Username

Password

Login

[Forgot Password](#)



## Select Department

You Have Access to Multiple departments please pick the one you want to view

Deck >

Bridge >

Technical >

Factory >



## New Password

**Password**

**Confirm password**

Submit



## Map

Map

LatLng: 10.24, 124.12  
Rope x3 12.03.2022  
Net x1 12.03.2022



## Enter Order


Inventory

Name

Stock

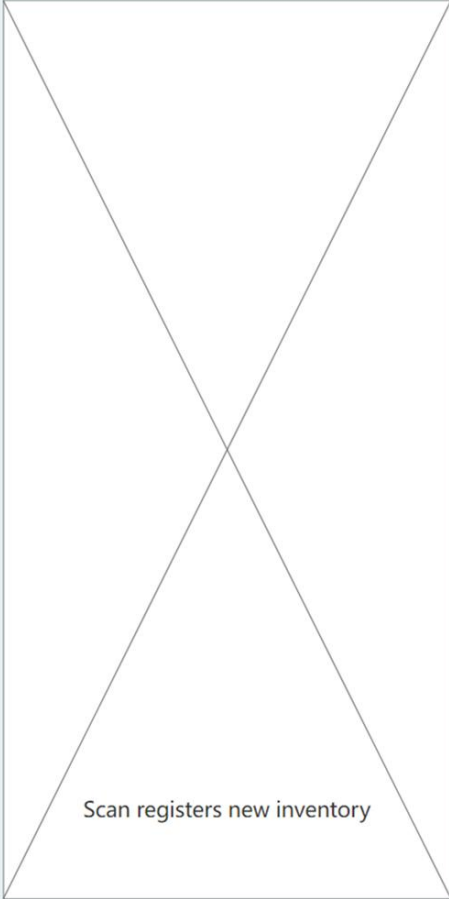
Product name

Product number


Barcode  


**Submit**

← Scan Order





Scan registers new inventory







Search...  

**Recommended Inventory**

Item 1	<input type="text" value="50"/>
Item 2	<input type="text" value="2"/>
Item 3	<input type="text" value="10"/>
Item 4	<input type="text" value="3"/>
Item 5	<input type="text" value="4"/>
Item 6	<input type="text" value="15"/>
Item 7	<input type="text" value="436"/>
Item 8	<input type="text" value="13"/>

- Admin Page
- Full Name
- Register New User >
  - Send Bill >
  - Administer Users >
  - Change department >
  - Change password >
  - Preffered Inventory >
- 


- My Account
- Full Name
- Change department >
  - Change password >
  - Preffered Inventory >
- 




← Register New User

Full Name

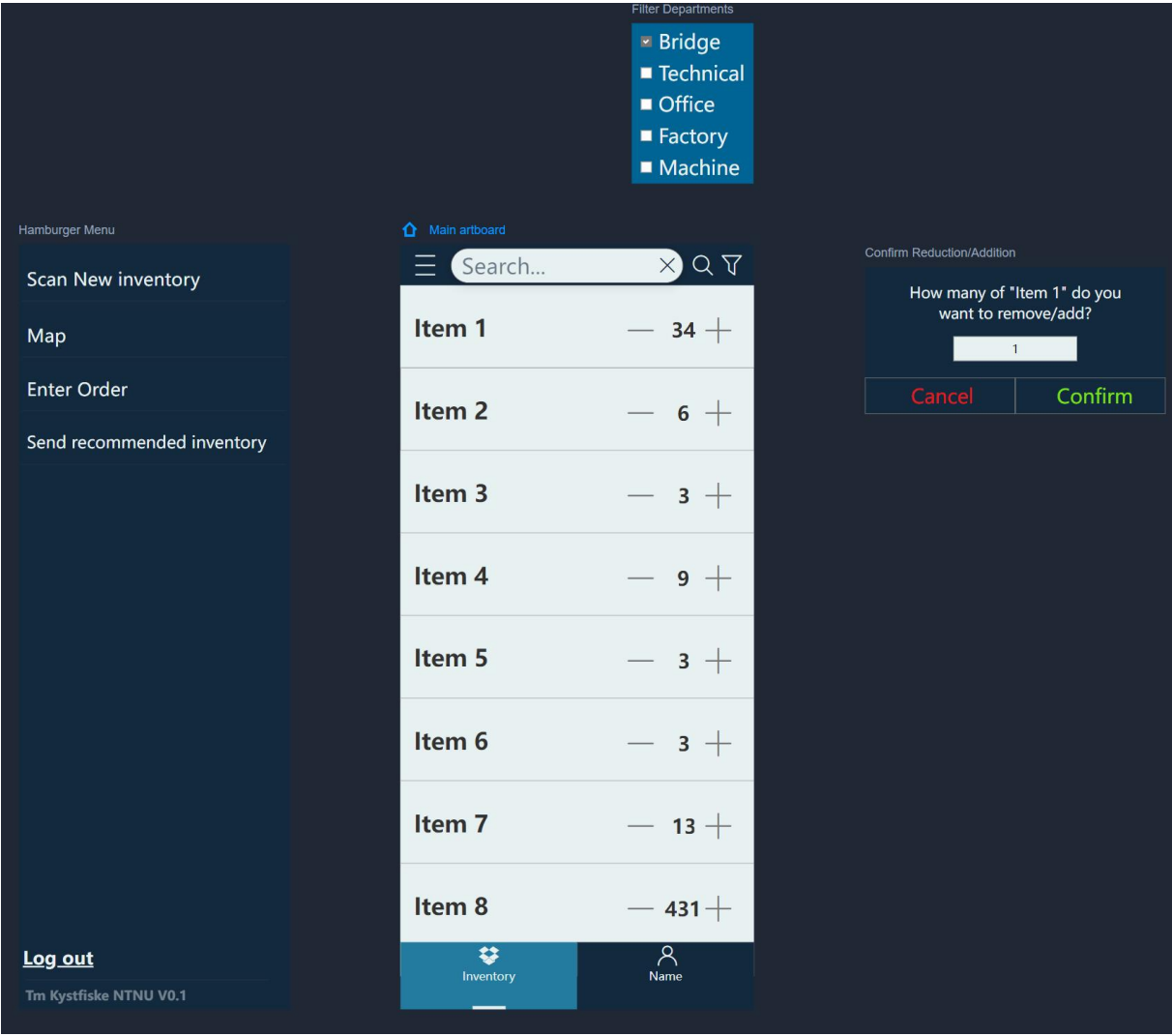
Email

Department

**Send Mail**





Scan New inventory

Map

Enter Order

Send recommended inventory

[Log out](#)

Tm Kystfiske NTNU V0.1



Search...



Item 1 — 34 +

Item 2 — 6 +

Item 3 — 3 +

Item 4 — 9 +

Item 5 — 3 +

Item 6 — 3 +

Item 7 — 13 +

Item 8 — 431 +



Inventory



Name

Confirm Reduction/Addition

How many of "Item 1" do you want to remove/add?

1

Cancel

Confirm

5.2 Tablet



Water Inc.

Username

Password

Login

[Forgot Password](#)



# New Password

## Password

## Confirm password



# Select Department

You Have Access to Multiple departments please pick the one you want to view

Deck

Confirm

- Scan New inventory
- Add User
- Enter Order
- Map
- [Log out](#)

Search...   

Item 1	—	34	+
Item 2	—	34	+
Item 3	—	34	+
Item 4	—	34	+
Item 5	—	34	+
Item 6	—	34	+
Item 7	—	34	+
Item 8	—	34	+
Item 9	—	34	+
Item 10	—	34	+
Item 11	—	34	+
Item 12	—	34	+



# Map

LatLng: 10.24, 124.12

Rope x3 12.03.2022

Net x1 12.03.2022



Scan New inventory

Add User

Enter Order

Map

# My Account

{Full Name}

Change department



Change password



Preffered Inventory



Log out

© Tm Kystfiske NTNU V0.1



Inventory



{Name}



Scan New inventory

Add User

Enter Order

Map

# Admin Page

{Full Name}

Register New User



Send Bill



Administer Users



Change department



Change password



Preffered Inventory



Log out

™ Kystfiske NTNU V0.1



Inventory



Account



# Administer users

Hans Hansen	hans@hansen.no	Delete



# Register New User

Full Name

Email


Department +

Register


Stock


Product name

Product number

Barcode 

**Submit**

 Inventory

 Account

# Enter Order

Scan New inventory

Add User

Enter Order

Map

[Log out](#)



Inventory



Account

- Scan New inventory
- Add User
- Enter Order
- Map
- [Log out](#)

Search...   

Preferred Inventory

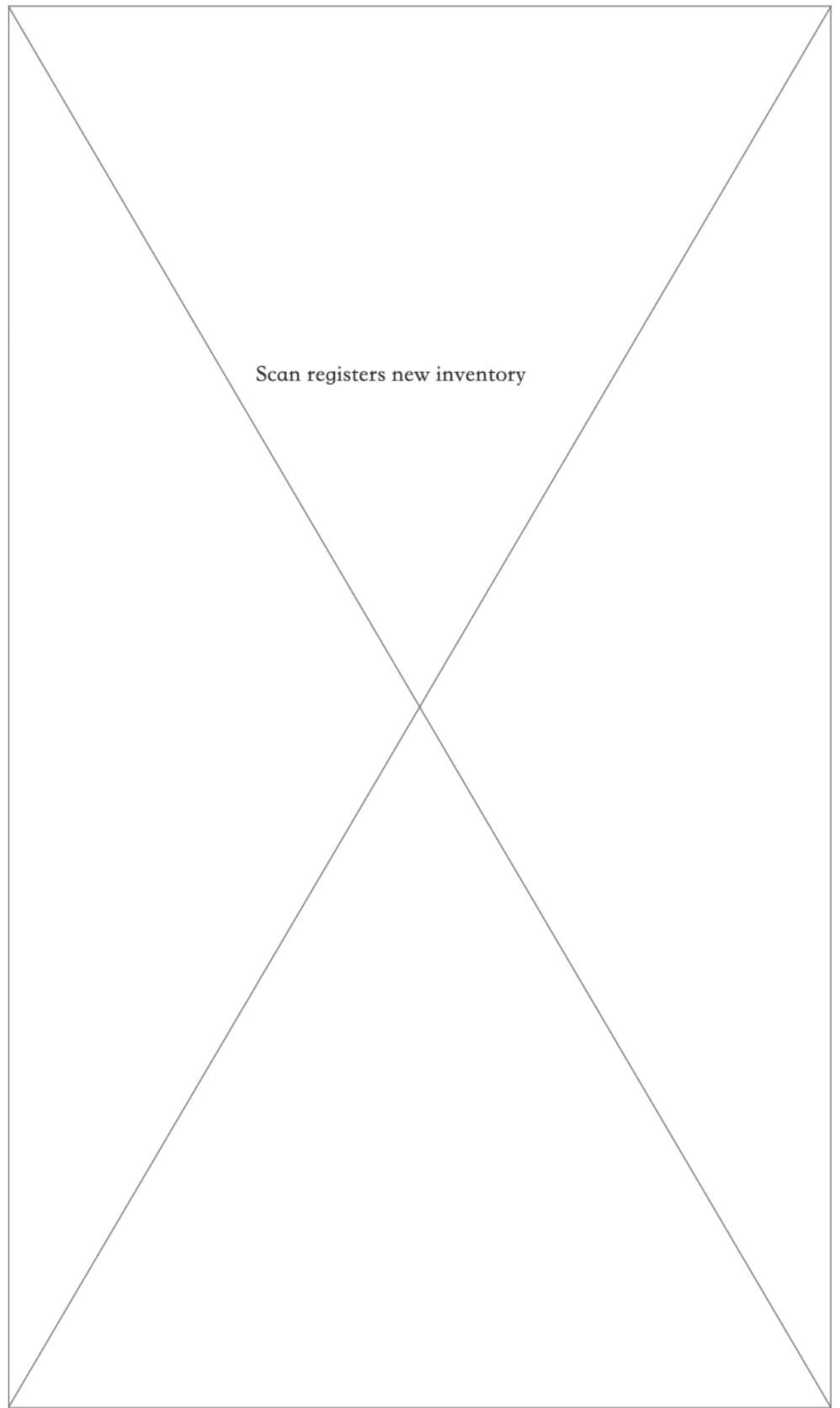
Item 1	— 34 +
Item 2	— 2 +
Item 3	— 4 +
Item 4	— 10 +
Item 5	— 1 +
Item 6	— 3 +
Item 7	— 0 +
Item 8	— 12 +
Item 9	— 3 +
Item 10	— 7 +
Item 11	— 5 +
Item 12	— 65 +

Scan New inventory

Add User

Enter Order

Map



Scan registers new inventory



Upload from phone

Log out



Inventory



Account

## **D. Project description**



# **Kravspesifikasjon.**

## **Tiltenkt navn på applikasjon.**

**Ship organizer.**

## **Målet til applikasjonen.**

Applikasjonen skal være en hjelp til å holde orden på arbeidsplassen ombord i en båt. Vi som arbeider ombord ser ofte at det er lett og miste kontrollen over beholdning ombord. Her er ingen skikkelig rutine for å registrere uttak og påfyll av utstyr og deler, samt sjekke opp mot faktura som kommer til kontoret på dei forskjellige ordrene. Det vil vi med denne applikasjonen gjøre noe med. Da det er varierende kompetanse på bruk av data ombord, vil vi prøve å gjøre det så enkelt som mulig å både registrere forbruk og innkjøp. Slik kan en til enhver tid ha full kontroll over hva som er ombord.

De forskjellige departement ombord er kontor, teknisk avdeling, styrhus, maskin, dekk, fabrikk og bysse.

## **Velkomstbilde.**

⇒ På denne siden har du innlogging.

⇒ Brukernavn og passord bestemmer hvilket departement en får tilgang til. Har en tilgang til flere må man velge.

⇒ En skal kunne huske brukernavn og passere slik en ikke trenger og logge inn hver gang fra samme enhet.

### **Mine sider de forskjellige departement.**

⇒ Her har en valg mellom og legge inn komponenter eller ta ut komponenter.

⇒ En må kunne ta ut beholdningsliste.

⇒ En må kunne legge inn anbefalt beholdning ombord.

⇒ En må kunne hente ut og sende bestillingsliste i henhold til beholdning ombord og ønsket beholdning.

⇒ En må kunne scanne ordresedler som leser og legger de forskjellige komponenter det det skal i beholdningslisten.

⇒ En må kunne legge inn/ta ut komponenter manuelt.

### **Mine sider skipper, tekniske sjef og kontor.**

⇒ Samme muligheter som mine sider forskjellige departement.

⇒ Har admin tilgang og kan legge til brukere.

⇒ Har full oversikt over alle departement.

⇒ Kunne sende faktura ombord til rett departement for godkjenning.

## **Tekniske krav.**

- Applikasjonen skal være på Norsk og Engelsk.
- GPS.
- Må ha kartintegrasjon.
- Må kunne scanne ordresedler.
- Må kunne oppdatere beholdning manuelt.
- Må kunne lese barcoder.

Applikasjonen er i første omgang ment som en demo. Vi tenker å testkjøre den på eget rederi opp mot egne kunde, og om det blir en suksess tenker vi å videreutvikle den.

## **Ulike integrasjoner.**

- Mot rederi modul.
- Alle departement må være integrert med hverandre.
- Må kunne integreres mot eksterne moduler som leverandører.

## **Ikke funksjonelle krav.**

- Plattform som IOS og Android.
- Skal brukes på telefon og nettbrett.
- Høg sikkerhet.
- Starte å virke innen rimelig tid.

## **Referanser til applikasjoner vi liker.**

Ideen kom da vi ombord med oss har hatt store problemer mellom skiftene å få til en felles måte å registrere beholdning ombord. Og utifra det med å betale regninger i bank, der vi kan ta et bilde og sende til banken, så fyller det seg automatisk ut, tenkte vi att dette var noe vi kunne bruke ombord med oss til å gjøre det enklere å legge inn ordresedler i ei beholdning.

## **Tidsramme.**

Det er ikke satt noen tidsramme på prosjektet. Det viktigste er att det ferdige produktet blir best mulig.

## **Kontakt person.**

Kurt Louis Skjong. Tlf: 92636079, [email kurt@maoyi.no](mailto:kurt@maoyi.no)

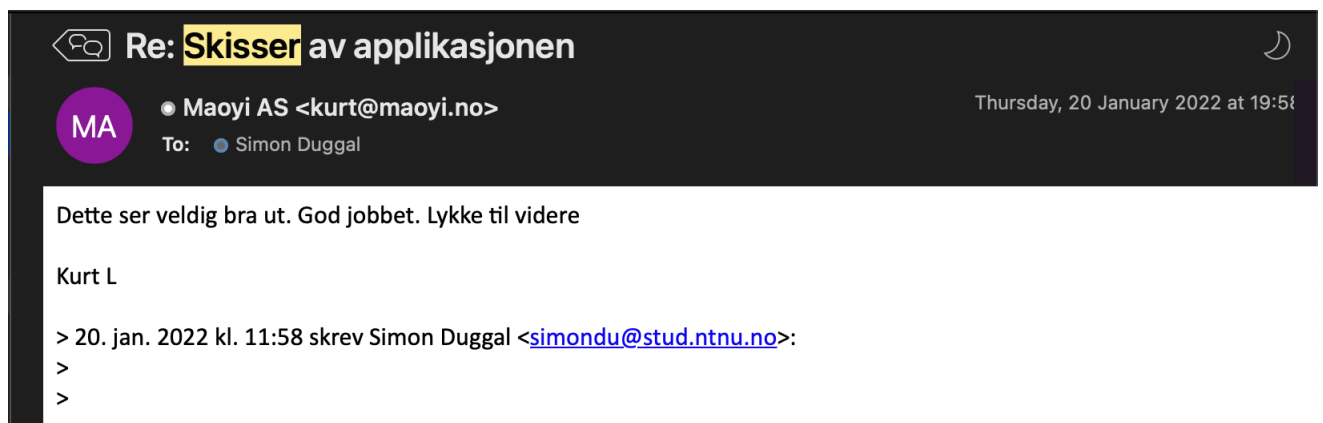
## **E. Emails**

Hei.  
Da har vi fått testet app en stund, og det ser veldig bra ut. Godt jobbet. Dere har levert et godt produkt. Det nærmer seg vel innlevering av bachelor oppgaven, og vi vil bare ønske dere masse lykke til med resultatet der. Blir en sekser fra oss her.  
Takk for samarbeidet.

Mvh  
Kurt L Skjong  
James R Eide.

Sendt fra min iPad

**Figure E.1:** Feedback from client on final product



**Figure E.2:** Feedback from client on wireframes

## **F. Project Manual**

See separate attachment.

## **G. Repository**

See Attachments for source code to both repositories.

### **Application**

GitHub: <https://github.com/johanneslj/ShipOrganizerApp>

### **Server**

GitHub: <https://github.com/johanneslj/ShipOrganizerService>



