

## Bachelor's project

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

Henrik Ranvik Bjørnland, Marius Blokkum Nakstad,  
Theodor Giskegjerde Urtegård.

## Distributed and Mobile Systems

Application to optimize ploughing and  
communication.

Bachelor's project in Bachelors in Computer Engineering  
Supervisor: Kjell Inge Tomren

May 2020



TITLE:

**Mobile application for snow ploughing**

CANDIDATE(S):

**Henrik Ranvik Bjørnland, Marius Blokkum Nakstad, Theodor Giskegjerde Urtegård**

DATE:	COURSE CODE:	COURSE TITLE:	RESTRICTION:
20/5/2020	IE303612	Bacheloroppgave (data)	

STUDY PROGRAM:	PAGES / APPENDIX:	LIBRARY NO.:
BACHELOR I INGENIØRFAG - DATA	98 / 4	

CLIENT(S)/SUPERVISOR(S):

CLIENT(S): AVENTO AS  
Supervisor(s): Kjell Inge Tomren, Girts Strazdins

SUMMARY:

This report contains the results, reflection and conclusion for the bachelor project done by Henrik Ranvik Bjørnland, Marius Blokkum Nakstad and Theodor Giskegjerde Urtegård at NTNU Ålesund, as well as the theoretical basis, materials and methods used to achieve those results.

The task was given by Avento, with Anders Beite as contact person. The purpose of the task was for Avento to get a system which could be used to optimize and administer snow ploughing in cabin areas for both the cabin owner, the area owner and snow plower.

The solution was to make a service consisting of a back end and a front end. The back-end is with a Spring-Boot application and a MySQL database, the front-end was designed for mobile through Flutter and the server configurations were created on google cloud platform using Terraform and Ansible.

Result is a functional system which consists of a working back-end running a database and server application and a working front-end mobile application, which fulfils most of the requirement specification.

*This task is an exam report done by students at NTNU in Ålesund.*



# CONTENTS

<b>SUMMARY</b>	<b>6</b>
<b>TERMINOLOGY</b>	<b>7</b>
TERMS	7
KEYWORDS	8
<b>1 INTRODUCTION</b>	<b>11</b>
<b>2 THEORETICAL BASIS</b>	<b>12</b>
2.1 LAWS AND REGULATIONS	12
2.1.1 <i>Personal Data Act</i>	12
2.1.2 <i>GDPR</i>	12
2.1.3 <i>Consent</i>	12
2.2 SECURITY	12
2.2.1 <i>Hashing</i>	12
2.2.2 <i>Software Tokens</i>	12
2.2.3 <i>Cryptography</i>	13
2.2.4 <i>HTTPS</i>	13
2.2.5 <i>Common attacks</i>	13
2.3 GROUP MANAGEMENT	14
2.3.1 <i>Agile</i>	14
2.3.2 <i>Scrum</i>	15
2.3.3 <i>Communication Tools</i>	16
2.3.4 <i>Jira</i>	16
2.3.5 <i>Confluence</i>	17
2.4 STANDARDS & CONCEPTS	17
2.4.1 <i>Object Oriented Programming</i>	17
2.4.2 <i>Coupling &amp; Cohesion</i>	17
2.4.3 <i>Infrastructure as Code</i>	17
2.4.4 <i>Version control</i>	17
2.4.5 <i>Continuous Integration</i>	18
2.4.6 <i>Continuous Deployment</i>	18
2.5 PROGRAMMING	19
2.5.1 <i>IDE</i>	19
2.5.2 <i>Flutter</i>	19
2.5.3 <i>Dart</i>	19
2.5.4 <i>SQL</i>	19
2.5.5 <i>Java</i>	19
2.5.6 <i>Spring Framework</i>	20
2.5.7 <i>Maven</i>	20
2.5.8 <i>Ansible</i>	20
2.5.9 <i>Terraform</i>	20
2.6 DESIGN	20
2.6.1 <i>Wireframes</i>	20
2.6.2 <i>Model-View-Controller</i>	20
2.6.3 <i>Back-end</i>	21
2.6.4 <i>Relational Database Design</i>	25
2.6.5 <i>Front-end</i>	26
2.7 SERVICES & RESOURCES	28
2.7.1 <i>Cloud Services</i>	28
2.8 TESTING	29
2.8.1 <i>Unit testing</i>	29
2.8.2 <i>User testing</i>	29
<b>3 MATERIALS AND METHODS</b>	<b>30</b>
3.1 PROJECT ORGANIZATION	30

3.1.1	<i>Project Team</i>	30
3.1.2	<i>Workflow</i>	30
3.1.3	<i>Pre-project</i>	30
3.1.4	<i>Project planning</i>	30
3.1.5	<i>Group communication</i>	31
3.2	PROGRAMMING	31
3.2.1	<i>Java</i>	31
3.2.2	<i>Flutter</i>	31
3.3	THIRD-PARTY SERVICES	32
3.3.1	<i>RouteXL</i>	32
3.3.2	<i>Frost API</i>	32
3.3.3	<i>Google Maps Services</i>	32
3.3.4	<i>Kartverket</i>	32
3.3.5	<i>Google cloud platform</i>	32
3.4	CODING CONVENTION	32
3.4.1	<i>Class names</i>	33
3.4.2	<i>Method names</i>	33
3.4.3	<i>Field names</i>	33
3.5	FRAMEWORKS & DEPENDENCIES	33
3.5.1	<i>Spring Framework</i>	33
3.5.2	<i>Back-End Dependencies</i>	34
3.6	ENVIRONMENTS	34
3.6.1	<i>MySQL</i>	34
3.6.2	<i>Postman</i>	34
3.6.3	<i>Gitlab</i>	34
3.6.4	<i>Draw.io</i>	34
3.6.5	<i>Eclipse</i>	35
3.6.6	<i>Android Studio</i>	35
3.6.7	<i>Visual Studio Code</i>	35
<b>4</b>	<b>RESULTS</b>	<b>36</b>
4.1	GENERAL SYSTEM DESIGN	36
4.2	SERVER	36
4.2.1	<i>Infrastructure</i>	36
4.2.2	<i>Configuration</i>	40
4.3	API	42
4.4	BACK-END	43
4.4.1	<i>Database</i>	43
4.4.2	<i>Back-end application</i>	48
4.5	FRONT-END	54
4.5.1	<i>Architecture</i>	54
4.5.2	<i>Programming language</i>	54
4.5.3	<i>Front-end Code</i>	55
4.5.4	<i>Interaction flow</i>	60
4.5.5	<i>Graphical user interface</i>	61
<b>5</b>	<b>REFLECTION</b>	<b>80</b>
5.1	GRAPHIC USER INTERFACE	80
5.2	SERVER INFRASTRUCTURE AND CONFIGURATION.	80
5.3	SCRUM WORKFLOW	80
5.4	FRONT-END	81
5.4.1	<i>General User Requirements</i>	81
5.4.2	<i>Cabin Owner Requirements</i>	81
5.4.3	<i>Ploughmen Requirements</i>	82
5.4.4	<i>Admin Requirements</i>	83
5.4.5	<i>Non-functional Requirements</i>	83
5.5	BACK-END	85
5.5.1	<i>Database</i>	85
5.5.2	<i>Service Layered Design</i>	88
5.5.3	<i>DTO – Object Mapping</i>	88

5.5.4	<i>Business logic</i>	89
5.5.5	<i>HTTP's GET</i>	89
5.5.6	<i>Proxy Server</i>	89
5.5.7	<i>Product specification</i>	89
5.6	FURTHER DEVELOPMENT	89
5.7	LEARNED	90
<b>6</b>	<b>CONCLUSION</b>	<b>91</b>
<b>7</b>	<b>REFERENCES</b>	<b>92</b>
	<b>APPENDIX</b>	<b>98</b>

## SUMMARY

This report contains the results, reflection and conclusion for the bachelor project done by Henrik Ranvik Bjørnland, Marius Blokkum Nakstad and Theodor Giskegjerd Urtegård at NTNU Ålesund, as well as the theoretical basis, materials and methods used to achieve those results.

The task was given by Avento, with Anders Beite as contact person. The purpose of the task was for Avento to get a system which could be used to optimize and administer snow ploughing in cabin areas for both the cabin owner, the area owner and snow plower.

The solution was to make a service consisting of a back end and a front end. The back-end is with a Spring-Boot application and a MySQL database, the front-end was designed for mobile through Flutter and the server configurations were created on google cloud platform using Terraform and Ansible.

Result is a functional system which consists of a working back-end running a database and server application and a working front-end mobile application, which fulfils most of the requirement specification.

## TERMINOLOGY

### *Terms*

#### **Ahead of time (AOT) Compilation**

Compiling higher level code into native machine code before it runs.

#### **Atomicity**

Meaning a single object which cannot be divided into smaller sub objects.

#### **JSON**

JavaScript Object Notation, JSON, is a standard format for storing and transporting information.

#### **XML**

Markup language mainly used to store and transport data.

#### **C/C++**

A general-purpose programming language often considered the "base" for a lot of other programming languages. C++ is an extension to C which adds functionality such as object orientation.

#### **C#**

A general-purpose, object-oriented programming language.

#### **JavaScript**

High level programming language based on the ECMAScript specification. Often used in web programming.

#### **Jar**

Java Archive, is a compiled, packaged and runnable Java application.

#### **Back-end**

The part of an application the user does not see. Usually handles logic, communication and data.

#### **Front-end**

The part of an application the user sees. Handles user interaction and forwards it to the back-end.

#### **Cache**

Data stored temporarily where it is quickly accessible, usually from the data being used a lot or recently.

#### **Google**

A company specialized in information technologies, services and products.

#### **Pipeline**

A sequence of action where the output from the previous action is used as input for the next.

#### **Ubuntu**

A free operating system based on the Linux kernel.

### **Keywords**

UML - Unified Modelling Language  
JSON - JavaScript Object Notation  
JWT - JSON Web Token  
DAO - Data Access Object  
POJO - Plain Old Java Object  
EER - Enhanced Entity-Relationship  
UI - User Interface  
GUI - Graphical User Interface  
GCP - Google Cloud Platform  
MVC - Model View Controller  
GDPR - General Data Protection Regulation  
DTO - Data Transfer Object  
REST - Representational State Transfer  
API - Application Programmable Interface  
CRUD - Create, Read, Update and Delete.  
1NF - First normal form  
2NF - Second normal form  
3NF - Third normal form  
BCNF - Boyce-Codd normal form  
UX - User Experience  
SSH - Secure Shell  
IoC - Inversion of Control  
CI - Continuous Integration  
CD - Continuous Deployment  
XML - eXtensible Markup Language  
HTTP - Hypertext Transfer Protocol  
URI - Uniform Resource Identifier

## Figures

Figure 1: Encryption of messages [10] .....	13
Figure 2: Agile workflow [15] .....	14
Figure 3: The scrum board [18].....	15
Figure 4: Scrum workflow [19].....	16
Figure 5: Version control [26] .....	18
Figure 6: Spring logo [42] .....	20
Figure 7: Model View Controller design [47].....	21
Figure 8: HTTP request [54] .....	22
Figure 9: Service without layered design [57] .....	23
Figure 10: Service with layered design [58] .....	24
Figure 11: Proxy server [64].....	25
Figure 12: Coding convention classes.....	33
Figure 13: Coding convention methods .....	33
Figure 14: Coding convention fields .....	33
Figure 15: Deployment Diagram of the system.....	36
Figure 16: Syntax validation .....	36
Figure 17: Terraform Test.....	37
Figure 18: Successful deployment .....	37
Figure 19: Destroy resources .....	37
Figure 20: Successful destruction .....	37
Figure 21: Variable.tf .....	38
Figure 22: Provider.tf.....	38
Figure 23: Outputs.tf .....	38
Figure 24: Actual output.....	39
Figure 25: Gitlab pipeline .....	39
Figure 26: Google Cloud Platform .....	39
Figure 27: Servers in description file.....	40
Figure 28: Inventory.gcp.yaml structure and content .....	40
Figure 29: Running playbook .....	41
Figure 30: Playbook.yaml .....	41
Figure 31: Postman interface .....	42
Figure 32: Snippet from generated API documentation.....	43
Figure 33: EER diagram.....	43
Figure 34: Security filter.....	48
Figure 35: Controller layer.....	49
Figure 36: Service layer .....	49
Figure 37: Repository layer.....	50
Figure 38: Layered Service Design.....	50
Figure 39: Kartverket request .....	51
Figure 40: RouteXL request .....	52
Figure 41: Google Directions request .....	53
Figure 42: Frost request .....	53
Figure 43: Cabin adapter.....	54
Figure 44: Flutter pubspec file snippet .....	55
Figure 45: Flutter private field.....	55
Figure 46: Flutter private method .....	55
Figure 47: Flutter stateful widget snippet .....	56
Figure 48: Flutter setState example .....	56
Figure 49: Flutter stateless widget example .....	57
Figure 50: Flutter build method .....	58
Figure 51: Route name for widget .....	58
Figure 52: Example of pushNamed navigation .....	58
Figure 53: "pop" of the current widget .....	58
Figure 54: Example of pushReplacementNamed navigation.....	59

Figure 55: Method from a service class .....	59
Figure 56: Interaction Flow chart.....	60
Figure 57: Example of error reporting with the ability to recover .....	62
Figure 58: Pre-project diagram of application.....	63
Figure 59: Login diagram .....	64
Figure 60: Login view.....	64
Figure 61: Message diagram .....	65
Figure 62: Register diagram .....	65
Figure 63: Register view.....	65
Figure 64: OTP view .....	66
Figure 65: Navigation diagram .....	67
Figure 66: Navigation view .....	67
Figure 67: Profile diagram .....	68
Figure 68: Profile view .....	68
Figure 69: Edit profile diagram.....	69
Figure 70: Edit profile view .....	69
Figure 71: Cabins diagram.....	70
Figure 72: Cabins view.....	70
Figure 73: Cabin diagram .....	71
Figure 74: Cabin view .....	71
Figure 75: Ploughmen cabin area diagram .....	72
Figure 76: Ploughmen cabin area view .....	72
Figure 77: Assign cabin area diagram .....	73
Figure 78: Assign cabin area view.....	73
Figure 79: Ploughing requests diagram .....	74
Figure 80: Ploughing requests view.....	74
Figure 81: Map diagram .....	75
Figure 82: Map view .....	75
Figure 83: Inbox diagram .....	75
Figure 84: Ploughmen application diagram .....	76
Figure 85: Ploughmen application view.....	76
Figure 86: Ploughman application diagram .....	77
Figure 87: Ploughman application view.....	77
Figure 88: Admin's cabin areas diagram .....	77
Figure 89: Admin's cabin area diagram .....	78
Figure 90: Admin's users diagram .....	78
Figure 91: Admin's user diagram .....	78
Figure 92: Admin's messages diagram .....	79
Figure 93: First EER diagram .....	85
Figure 94: Last EER diagram.....	86
Figure 95: Package structure .....	88



## 1 INTRODUCTION

This bachelor is carried out by Henrik Ranvik Bjørnland, Marius Blokkum Nakstad and Theodor Giskegjerde Urtegård as students of computer engineering at NTNU Ålesund. The task was given to us based on a list of tasks we could choose from. After creating a list of the top 3 tasks we would like to work with, we got chosen to work with the task of creating a system to help optimize and administer snow ploughing. The task had previously been worked on by other students, but we wanted the task if we could start the project from scratch. The task was our number one choice.

The task for the bachelor, as mentioned earlier, was to develop a system to help optimize snow ploughing and administration in cabin areas. This would be done by making communication between plower and cabin owner easier, and by providing a snow plower with the information about which cabins within a cabin area need ploughing and which ones don't to reduce the amount of unnecessary ploughing and help save time, resources, emission and road deterioration. The general concept for the service was that a cabin owner could register their own cabin to their account, and then add a reservation to that cabin whenever they were going to use it. When a reservation was made it would be added to a list of reservations for a specific area and a snow plower could then get a list of all the requests for ploughing in their area so they would know exactly which cabins needed ploughing and which ones they could skip.

The task was given by Avento AS, a consulting firm based in Ålesund and Ørsta, with Anders Beite being our person of contact from Avento AS. Through several meetings at Avento AS' offices a requirement specification was created which would detail the functionality of the system, which components should be made, which user roles should exist within the system as well as what the goals of the entire system would be. The requirement specification can be found as Appendix 1.

The background for the task was that someone at Avento AS was tired of trying to keep track of all the numbers and people they would need to contact in order to get their cabin ploughed in time, so they wished for a centralized system which could streamline the communication.

This report includes the theoretical background, knowledge and concepts used to solve the task; the tools, methods and materials used to solve the task as well as the reasoning to why they were chosen; and the achieved result of the project with a reflection and discussion regarding aforementioned results.

## **2 THEORETICAL BASIS**

### **2.1 Laws and regulations**

#### **2.1.1 Personal Data Act**

The Personal Data Act is a Norwegian law regarding how to store and use data from Norwegian citizens. Its purpose is to protect individuals and their right to privacy, as well as ensuring that their personal data is stored and processed in a manner where it is safe and will not be misused. [1] [2]

#### **2.1.2 GDPR**

The General Data Protection Regulation, GDPR, is a regulation introduced by the European parliament in 2018. The goal of the regulation is to protect the privacy and personal information of EU citizens and citizens of the European Economic Area. Although Norway is not a part of the EU, legislators have decided to make GDPR a part of the Personal Data Act. [3] [4]

#### **2.1.3 Consent**

In relation to regulations regarding data storage, consent is when a person has given explicit approval of their data to be stored or processed. Consent is only valid if it is presented in an optional, specific, informative and explicit way, requiring the user to consent through an action, and ensuring that the act of giving consent can be documented. The person giving consent should also be able to withdraw the consent in an equally easy manner. An example of giving consent in relation to allowing data storage would be to click a button, filling out a form or accepting the terms of conditions through a checkbox which is unchecked by default. [5]

### **2.2 Security**

#### **2.2.1 Hashing**

A hash is a function that takes an input and returns an output with different values. It is commonly used in cryptography, compression and checksum generation. Hashing is commonly used in cryptography as it hides the original value and can easily be converted back to its original value with a hash table. [6]

#### **2.2.2 Software Tokens**

A security token is often used as a method of authorization for single sign-on services. Tokens usually contain a user's authentication values, allowing a user to create a more secure password. Tokens are usually hashed, so if a token is leaked, it is not compromised. [7]

##### **2.2.2.1 JSON Web Token**

JSON web token (JWT) is a standard for creating JSON based access tokens. JWT works across different programming languages and they are self-contained. A token consists of a header that declares the type and the hashing algorithm to use. A payload that contains claims, the claims are different values that a client or server uses to authorize users without being stateful nor accessing persistent data. A signature that allows a server to sign, distribute and verify tokens. [8]

### 2.2.3 Cryptography

Cryptography – “The art of writing or solving codes” [9]

In software engineering cryptography is widely used as a means of security. As information is passed between clients, it can be looked at by an intermediary. If you as a Facebook user would send a private message to a friend without cryptography. Any intermediary would be able to read its content. By encrypting the message content before sending it, any intermediary would have to decrypt its content to understand it. When the other Facebook user retrieves the message, they can decrypt the message and read it. How a message is encrypted and decrypted depends on the encryption method. They usually use large pseudo prime numbers to make it difficult for any intermediary to decrypt the message. The larger a prime number is the longer time it would take to decrypt the message; this is used to discourage hackers. [11]

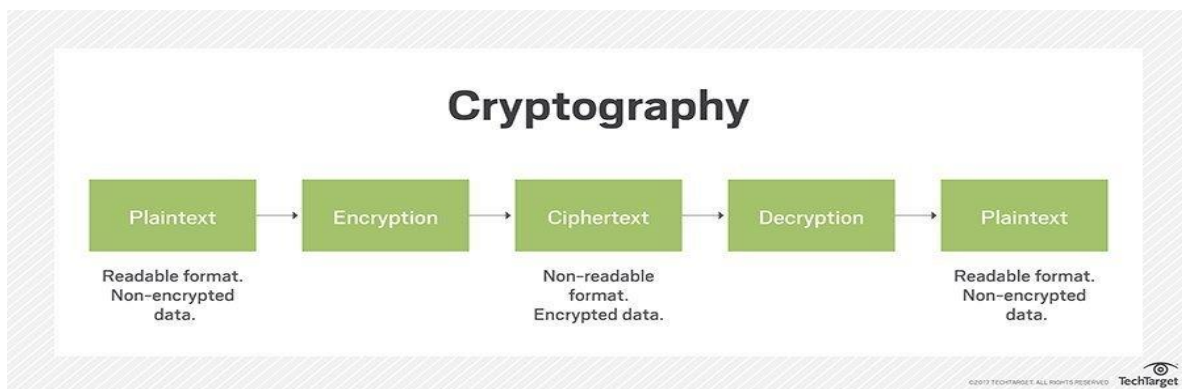


Figure 1: Encryption of messages [10]

### 2.2.4 HTTPS

HTTPS or Hypertext Transfer Protocol Secure is an extension to the Hypertext Transfer Protocol. It lays in the application layer, and encrypts communication using Transport Layer Security (TLS) or its predecessor Secure Sockets Layer (SSL). HTTPS protects against the man in the middle attacks, eavesdropping and so on. [12]

### 2.2.5 Common attacks

Some common attacks are man in the middle attack, SQL injection, XML injection, denial of service and so on. [13]

**XML injection** is an attack that compromise an XML service or application by injecting unintended XML.

**SQL injection** uses SQL statements injected into services or applications using SQL.

**Man in the middle** attack is a method where an attacker secretly modifies and/or passes on communication between two end points.

**Denial of service** attack is a method were an user attacks a service, leaving the service unusable by other clients.

## 2.3 Group management

### 2.3.1 Agile

Software development process that focuses on some key aspects.

Agile manifesto states that they focus on working with the customer, iterate and change the plan over time to satisfy the ever-changing customer needs. Agile focuses more on having software that is functional and focuses on the people and how they work together more than the tools used. [14]

Agile teams work in sprints which often last for two weeks. Each sprint involves the process of a 6-phase lifecycle. After a sprint the customers previous requirement should be fulfilled and be able to be further worked on. For the next sprint the customer might have different requirements and with agile it can now be built on the previous iteration making agile capable to respond to changes from the customer very well. [16]

An iteration lifecycle includes:



Figure 2: Agile workflow [15]

### 2.3.2 Scrum

A framework to apply Agile software development. [17]

Scrum focuses on breaking the project into smaller segments of required or wanted features of the project. Each of these features are then put into a list called the backlog.

Each iteration also known as a sprint in scrum is often a two-week process of finishing up some of the features from the backlog.

Each feature is broken down to smaller subtask required to finish the feature. The subtasks are handed out to the group to finish, and during the sprint this feature is only to be prioritized no other features from the backlog can be started on before the sprint is either stopped or completed.

The group will place their subtasks known as stories ("the user should be able to log in") on the scrum board. After each sprint, all the stories should be in the "Done" column.

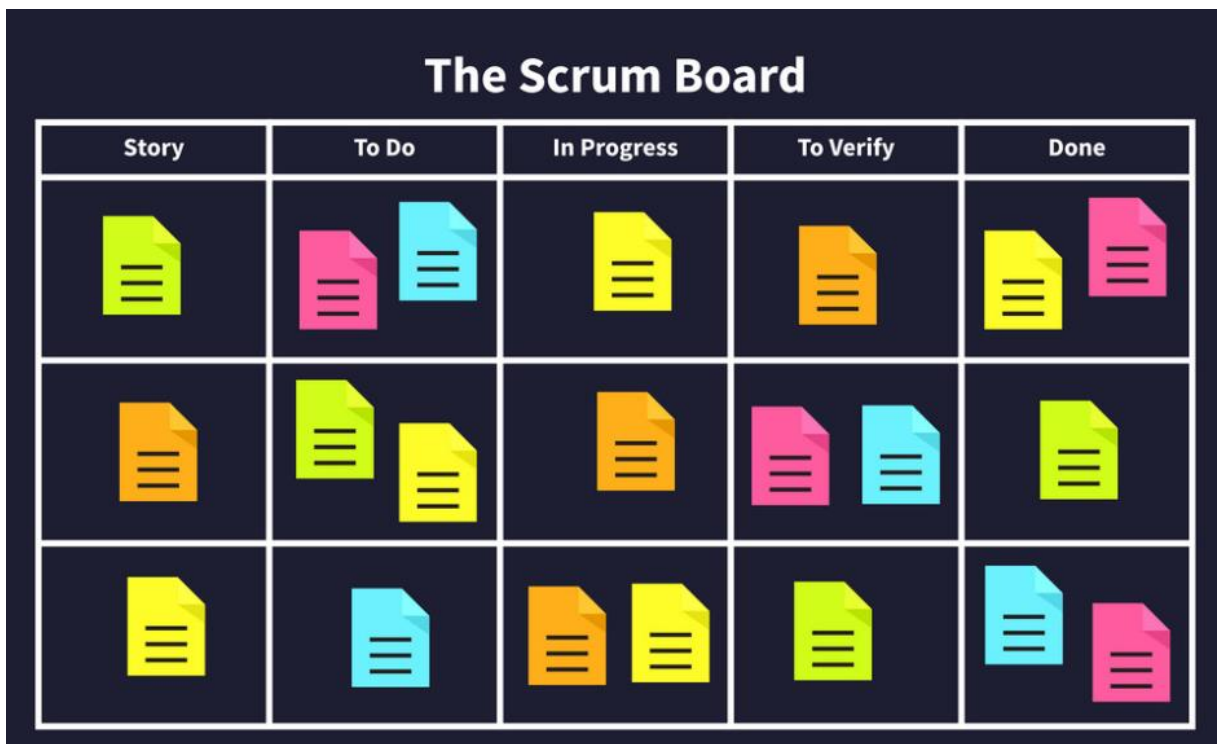


Figure 3: The scrum board [18]

After each sprint, a report is finished and written to document the sprint. Next sprint is then planned from the backlog which is sorted by customer requirements. The feature on top of backlog is then planned, and a new sprint is started.

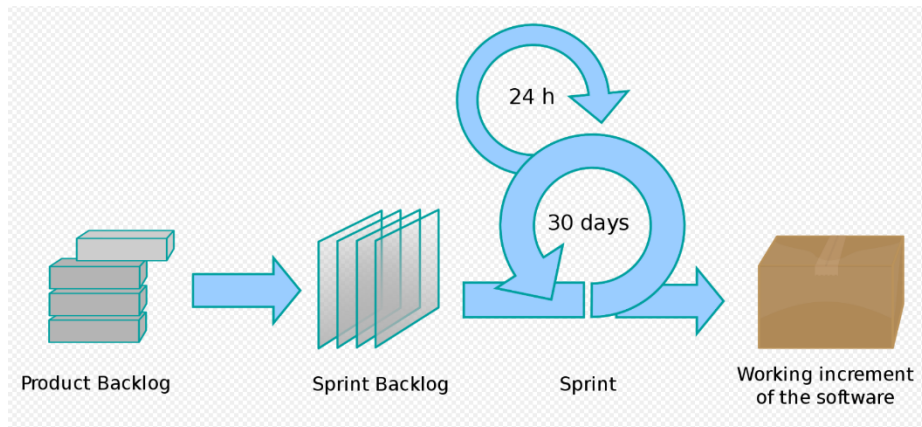


Figure 4: Scrum workflow [19]

## 2.3.3 Communication Tools

### 2.3.3.1 Discord

Discord is a platform for creating communities and groups for communication usually used within gaming environments. It builds on the IRC concept and offer real time VOIP and the ability to share screen, camera and files in a private group chat. [20]

### 2.3.3.2 Slack

Slack is an IRC based communication platform mostly used by corporations to keep work groups connected to the project and updated on the communications that is sent to the members of that group. Slack is often used to replace mail, it is one place where the members can communicate privately. [21]

## 2.3.4 Jira

Software tool used to keep track of progress and plan sprints to optimize scrum and agile development.

Jira is used by teams to plan, track, release sprints and generate report documentation after a sprint. Jira tracks issues and tasks completed or in progress by the team members. Jira can be used to create scrum board and backlog and keep it updated.

Jira was designed as a bug and issue tracker, today Jira has changed to a management tool for most use cases. Today Jira is used to practice agile software methodologies, with scrum and Kanban boards to the user.

Jira does more than just generate a Kanban/scrums board for us to use. It also generates scrum burn up/down charts that can be used to plan next sprints to easier determine the scope of the sprint.

Jira is one of the tools from the Atlassian toolkit. [22]

### **2.3.5 Confluence**

Confluences motto is "There's a template for that" meaning that using confluence you can generate reports from templates. Confluence is mostly used for documentation, procedures and sharing knowledge. [116]

## **2.4 Standards & Concepts**

### **2.4.1 Object Oriented Programming**

Object oriented programming is a way of programming which bases itself on modelling everything around "objects" containing data with a specific representation or structure which these objects have. Objects are created from classes which are abstract ways of describing and modelling how an object should look like and behave. Objects have procedures and functions which allow them to interact with themselves or other objects. [23]

### **2.4.2 Coupling & Cohesion**

Coupling and cohesion are two important principles in object-oriented programming when it comes to how to design good and reusable components. [24] [25]

#### **Coupling**

Coupling is a measurement of how reliant two or more software components are on each other. The more coupling that exist between two components the more dependent they are on each other and the "tighter" they are tied together, making them harder to keep independent. This is something which one would wish to avoid when designing and implementing software as a higher degree of coupling makes it harder to change the software component later on.

#### **Cohesion**

Cohesion is a measurement of how much a component can do, or its potential, in relation to other components. A component or module with high cohesion is very focused on the functionality and the job which it is supposed to be doing and should strive to excel at one task.

Low cohesion on the other hand would be a component which tries to do many things and might not excel at any of them.

Good software design should strive aim to have low coupling and high cohesion to ensure that components are easily reusable and easy to change in if needed, while also keeping each component as focused on its own task as possible. Coupling and Cohesion are inversely proportional, meaning that if you increase coupling, you reduce cohesion and vice versa.

### **2.4.3 Infrastructure as Code**

Server(s) generated with required configuration and setup by code, is defined as Infrastructure as Code. No matter how many times we run the code the output shall always be the same server with the same configuration and setup.

### **2.4.4 Version control**

Version control as a concept is a system which keeps track of changes made to a file or project over time, keeping a record over what was changed and who changed it. Version control also allows a developer to check out and swap between previous versions of a file or a project. Version control can be done locally or on a server shared between developers, a Centralized Version Control System, allowing them to work on the same code while also keeping track of all the changes while ensuring that any conflicts within a file or project can be resolved by a developer.

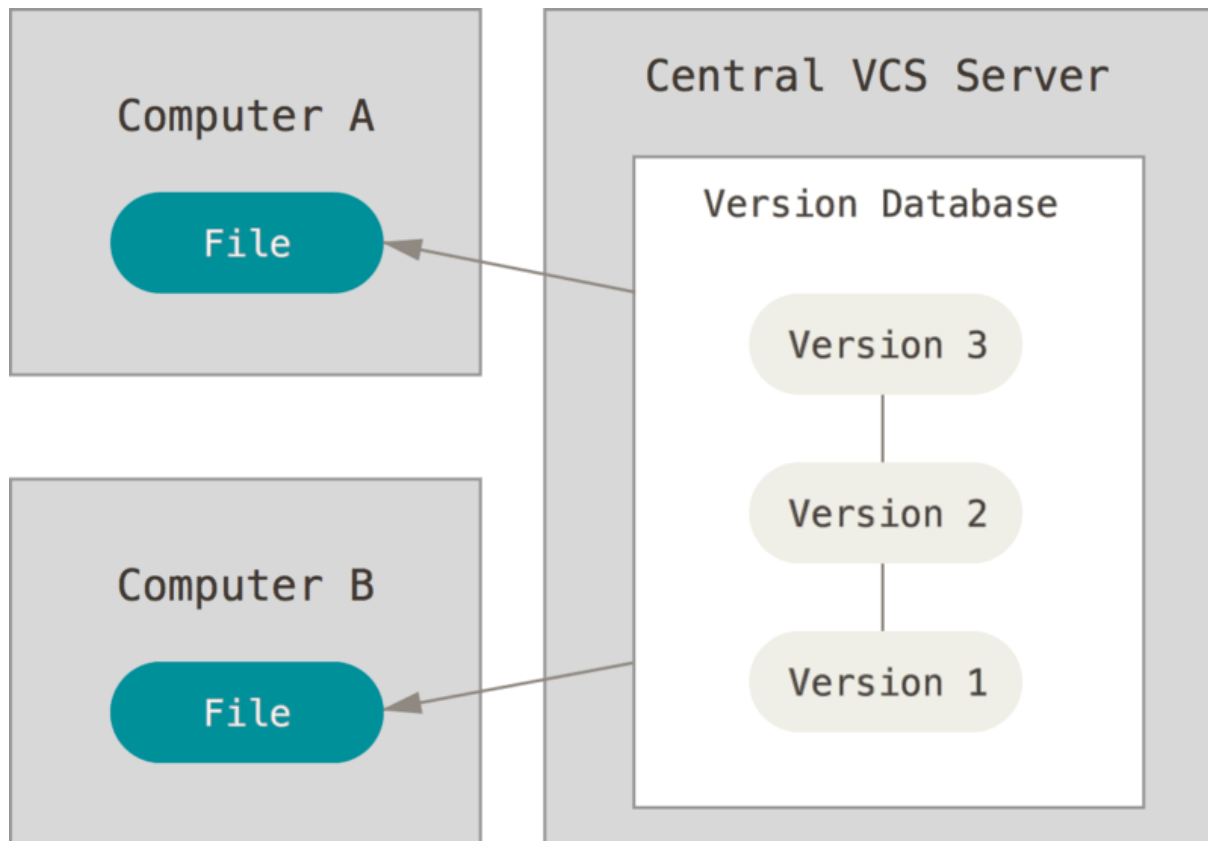


Figure 5: Version control [26]

The above image shows an example of a Centralized Version Control System. A common version is stored on a server, and individual computers have their own local copies which are either pulled straight from the server or modified and then later pushed onto the server with the changes. One of the most common version control system is Git. [27]

### 2.4.5 Continuous Integration

Continuous integration is a methodology where an implementation, change or update to a repository are automatically and immediately tested and built. This automation of testing and building allows for a fast and automatic detection of errors or problems with the repository in the event problematic code has been added. Since testing and building is done automatically it gives developers more time to focus on developing instead of spending time to test and build manually. [28] [29]

### 2.4.6 Continuous Deployment

Continuous deployment is a practice which focuses on automatically deploying a piece of software as soon as it has been built. This way it is testable in a test environment as soon as possible allowing for the discovery of issues which might not become apparent during testing or building. [30]



## **2.5 Programming**

### **2.5.1 IDE**

Integrated Development Environment, IDE, is a program which allows a programmer to effectively edit, test and run code by providing the developer with a lot of useful tools and features. Common features to find in an IDE are: Syntax highlighting, debugging tools and autocomplete. Often IDEs will allow a developer to install plugins to further enhance the functionality of their IDE. [31]

### **2.5.2 Flutter**

Flutter is an open-source UI toolkit created by Google which aims to help create natively compiled applications to run on mobile, in browser or as desktop applications. The Flutter engine is written in C/C++ with Skia, Google's 2D rendering engine, to provide low level rendering support. The Flutter SDK and all developer code is written in a language known as Dart. Flutter does not rely on OEM or default system widgets, but instead provide its own set of default widgets provided by the Flutter framework and engine. [32] [33]

### **2.5.3 Dart**

Dart is an object oriented, class-based, statically typed programming language introduced by Google in 2011. Dart has a syntax like C based programming languages and was designed to be easy to learn for programmers familiar with Java, C# and JavaScript. Dart can be compiled as JavaScript which allows it to run in browser, made to run in its own stand-alone virtual machine or be compiled ahead of time to be run natively on a machine. [34] [35] [36]

### **2.5.4 SQL**

Structured Query Language, SQL, is the standard, domain-specific language based upon the concepts of relational algebra, used for communicating with a relational database. SQL statements can be used to create or delete databases or tables within a database; add, update, retrieve or remove data. SQL was recognized and defined as an ISO standard in 1987 and had its latest major revision in 2016. SQL queries mainly consist of 3 commands: "Select", "From" and "Where". The "select" command defines what should be returned from the query, "from" defines which tables in the database should be queried, and "where" defines the criteria if the specific data from a row in the table should be returned or not. [37] [38] [39] [40]

### **2.5.5 Java**

Java is a programming language that adopts the concept of object-oriented programming where everything inherits from a common Object class. Software written in Java can be run on any device which supports the Java virtual machine (JVM). JVM offers garbage collector to remove old objects who no longer are used by the program to free up memory. Java has been one of the most popular programming languages the last twenty years and are still one of the most used programming languages. [41]

## 2.5.6 Spring Framework



*Figure 6: Spring logo [42]*

The Spring framework contains many APIs that is used in most modern Java applications. These tools allow developers to focus more on the business logic of their application. They contain useful features such as dependency injection, DAO support, scheduling and so on. Some larger components as Spring Security, Spring Web and Spring BOOT provides almost every tool one needs to make a web service. [117]

## 2.5.7 Maven

Maven is a powerful tool for project management. It provides a definition of what a project consists of and a way to share JARs across several projects. It makes the build process of an application very easy as it fetches all the JARs that are specified in a pom.xml file. [43]

## 2.5.8 Ansible

Ansible is used as a server configuration tool. Ansible is used to push script/configuration setups from a host computer to nodes/target computers. Ansible does not require the installation of any agent service to run on the target, but instead communicates through SSH. Ansible can store node/target configuration in what it calls playbooks. [44]

## 2.5.9 Terraform

Terraform is used as a server provision and management tool. Terraform adapts the strategy of "Infrastructure as Code" which is to describe the servers as code instead of a service. Terraform is used to push script/configuration setups to a server cloud service for that service to create the servers as described. Since servers are now created from code it is now very easy to do small changes and request a reboot with that change, many servers can be booted by one code. Through Terraform we can create, manage, destroy as servers as needed. [45]

## 2.6 Design

### 2.6.1 Wireframes

Wireframes are a way of creating simple UI concepts which gives a designer and a client a general idea of what the application's layout will look like. Wireframes are often simplistic and do not usually contain choice of colour or fonts, but rather focuses on the layout and relation of UI elements. [46]

### 2.6.2 Model-View-Controller

Model-view-controller (MVC) is a design pattern where the service is mainly split into three components. A view which the user interacts with, a controller that sends commands to the view and model components and a model that contains data. This

design pattern was traditionally used for graphical user interfaces but has become popular for designing web applications. [48] [49]

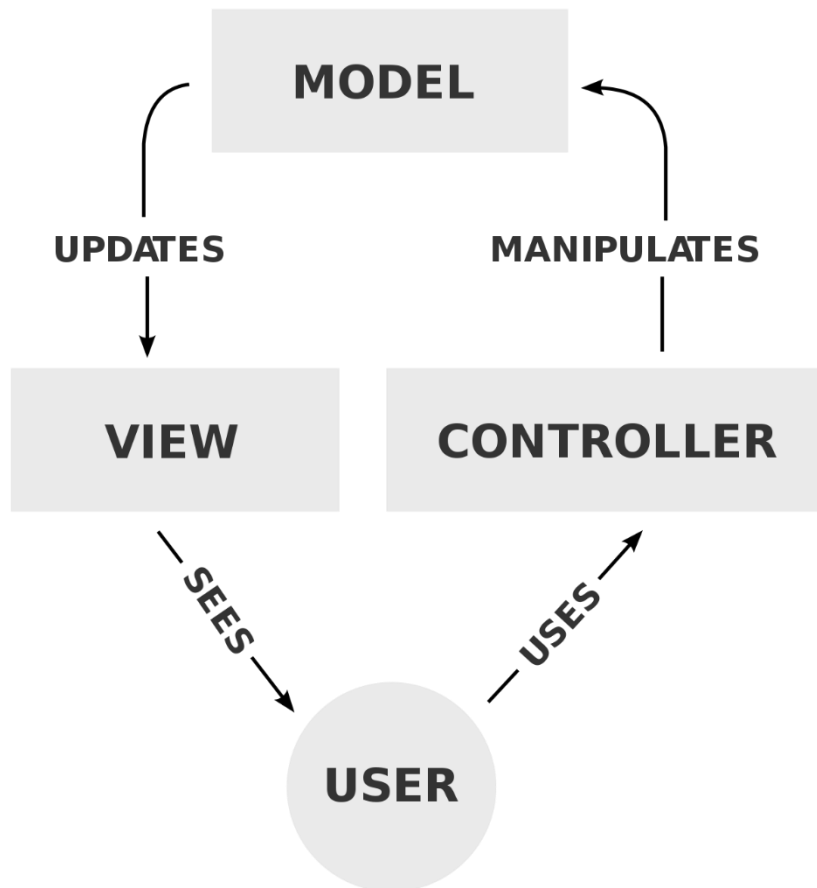


Figure 7: Model View Controller design [47]

### 2.6.3 Back-end

Back-end is a data access component usually used for separation of concerns between the user interface and the data access layer. In a client-server architecture, the server is considered the back-end and the client is considered the front-end. [50]

#### 2.6.3.1 Representational state transfer

Representational state transfer (REST) is a software architectural style for creating web services. REST also called restful web services provide a service for systems to access and manipulate web resources. Web resources are an abstract term for an entity or an action which can be identified, named, addressed, handled, or performed in any way whatsoever, on the web. Requests are made to the restful web services, which results in a response. The responses can be formatted in a large variety of ways, I.e. HTML, XML, JSON and others. Responses are application specific, but often contains references to other resources, state changes or the current state of a resource. The most common transfer protocol used with this system architecture is HTTP. The REST-system consists of six guiding constraints.

Client-server architecture, by separating client and data access systems, the scalability increases. It is possible to have multiple clients on different platforms which are almost independent of each other and can communicate with the same data access object.

Statelessness, no client state is stored on the server, only on the application side. Every request from the client contains all information necessary for the server to execute it accordingly.

Cache-ability, clients or intermediaries can cache responses, which can prevent excess calls to a server, thus improving performance.

Layered system, a client does not know if he is connected to the end server or an intermediary. This provides the possibility of scaling the service as demand rises. It is also possible to separate security logic from the business logic, allowing focus on security policies.

Code on demand, the server can extend a client's functionality by transferring executable code to the client.

Uniform interface, simplifies and decouples the architecture. There are four constraints for a uniform interface. Resource identification in request provides individual identification of a resource in the request, for example as an URI. Resource manipulation through representations, a client has enough information to manipulate the state of the resource. Self-descriptive messages as each response contains enough information about how to process the content. Hypermedia as the engine of application state (HATEOAS), the client should be able to access other resources without much knowledge of its content. [51] [52] [53]

### 2.6.3.2 Hypertext Transfer Protocol

"Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information system." [55]

An HTTP request is made up of a request line, request header fields, an empty line and an optional message body. There are many request methods, some are GET, POST, PUT and DELETE. GET method is used to request resources. POST method is used to add a new resource instance to the server. PUT method is used to update an existing resource instance on the server. DELETE method is used to delete resources.

HTTP responses consist of a status line, a header, an empty line and an optional body. The status code consists of three digits describing the response. 1xx is an informational response, I.e. request received, continuing process. 2xx is a success response meaning the request was received, understood and accepted. 3xx is the redirection response, further action is required to complete the request. 4xx is a client error response, meaning the request contains error or cannot be fulfilled. 5xx is a server error response, meaning the server failed to fulfil the request. [55] [56]

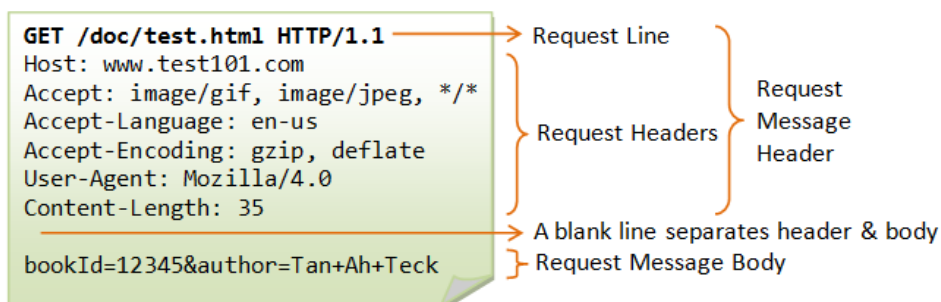


Figure 8: HTTP request [54]

### 2.6.3.3 Service Layer Pattern

Service layer pattern is a design pattern used in software development to increase understanding and readability of the software. By separating components that share logic into their own respective layer, providing low coupling. The typical server architecture designed with service layer pattern consists of a data access layer, the service layer and the controller layer. Each layer should only communicate with adjacent layers. [59]

Service without the service layer pattern

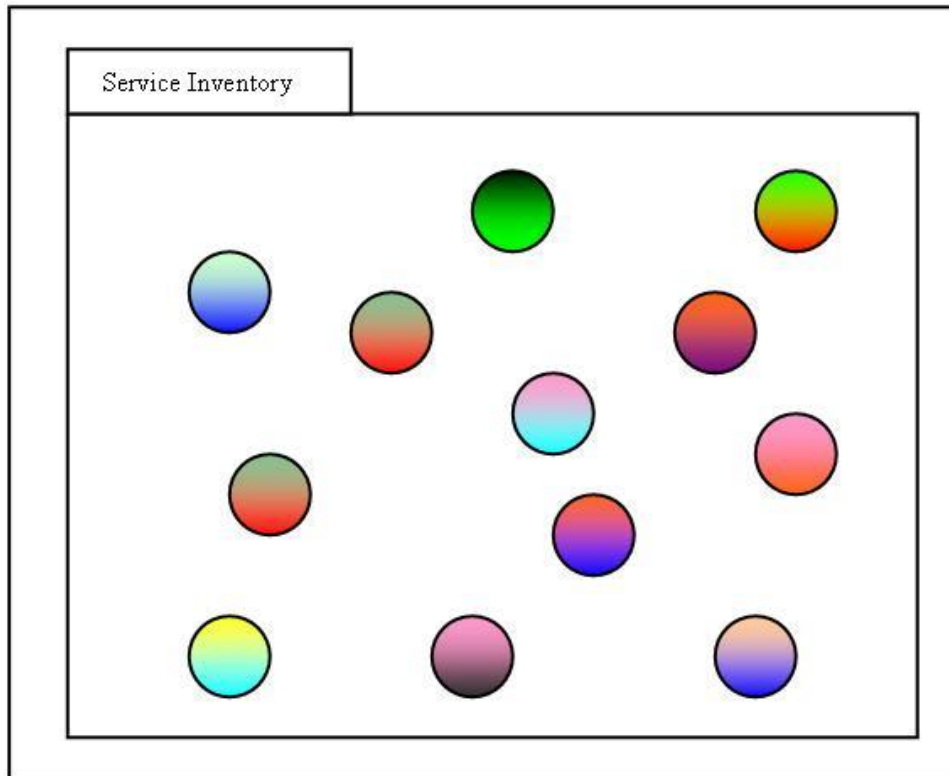


Figure 9: Service without layered design [57]

Service with the service layer pattern

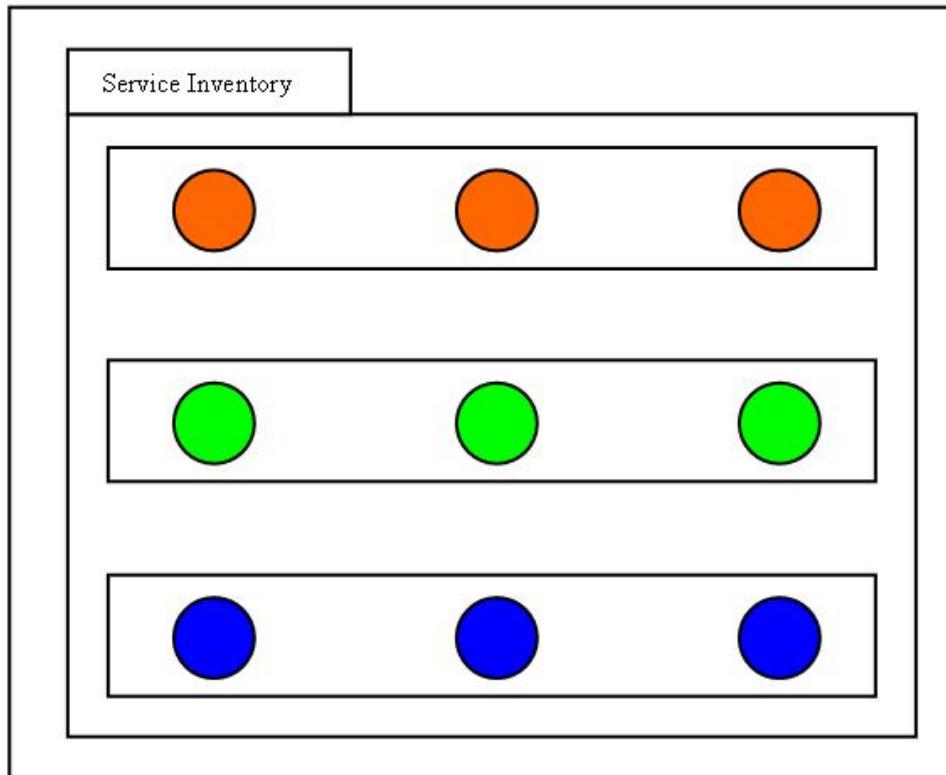


Figure 10: Service with layered design [58]

#### 2.6.3.4 Data Transfer Object Pattern

A Data Transfer Object (DTO) is an object which carries data between two processes. The DTO pattern is often used to increase performance in web services. Remote calls to a server are often a costly process which slows the client application. By using data aggregation, the necessary calls to the server decreases. The pattern also provides a layer of control for data accessed by a client application. [60]

#### 2.6.3.5 Application programming interface

"An application programming interface (API) is a computing interface which defines interactions between multiple software intermediaries." [61] With the rise of web services, it is also often used in context of REST APIs. Though an API is also referred to as an intermediary between software components, service layers and could encompass the entirety of a system.

#### 2.6.3.6 Create, Read, Update and Delete

Create, read, update and delete are often referred to as the CRUD methods. These methods cover the basis of operations in a persistent storage and REST services. The CRUD methods could also be mapped to four HTTP methods, POST, GET, PUT and DELETE in the order listed. [62]

#### 2.6.3.7 One Time Password

One-time password (OTP) is an authentication method often used by two-factor authentication. It can also be used as the only authentication method, but this is vulnerable to social engineering and man in the middle attacks. [63]

### 2.6.3.8 Proxy Server

A proxy server is an intermediary between a client and an end point. A proxy can provide security and performance for both the client and server. A client can use a proxy service to conceal sensitive information. A proxy can cache information that is frequently accessed by clients, providing another end point for data to be accessed. Thus, decreasing server load and providing clients faster communication. A proxy server is also often used to steer traffic to third party APIs, when other services are needed. [65]

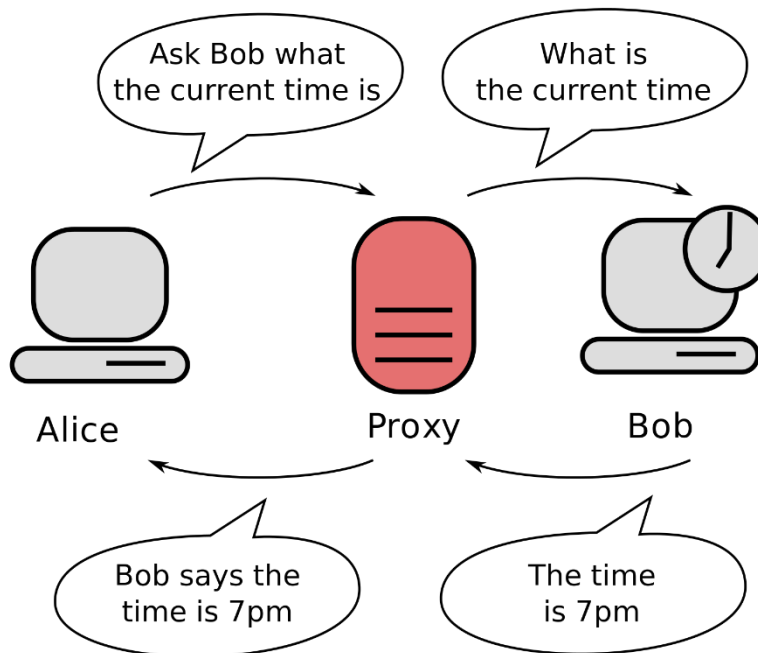


Figure 11: Proxy server [64]

### 2.6.3.9 JSON Infinite Recursion Problem

JSON infinite recursion problem is when a client accesses a linked list of entities, in which two or more entities have bi-directional associations and encoded into JSON. This results in an infinitely large response to the client. [118]

## 2.6.4 Relational Database Design

Relational Model for database management is a structure where data is represented as tuples grouped into relations. Tuples also identified as rows and relations also identified as a table. The purpose of this model is to provide a declarative approach for the CRUD methods. The process of creating a relational database design can be divided into 4 steps.

Each table should have a column or columns defined to be the primary key. A primary key is a unique identifier for a tuple. If a primary key is made up of one column, it is called a simple key. If it is made up of multiple columns it is called a composite key. Primary keys can be any data type, but integers are preferred for efficiency.

A relational database consists of different relationships. One-to-many relationship where one row in a parent table can be linked to multiple rows in a child table. This is achieved using a foreign key, referencing the primary key from a parent table in a child table. Many-to-many relationship is when two or more tables have a one-to-many relationship with one table. This table, often known as a junction table, is then created with foreign keys of the related entities. One-to-one relationship is when two tables are linked by exactly one row from each table. This relationship is often used for optional data belonging to a table with only the required data. [67] [68]

Normalization comes in multiple layers from first normal form (1NF) to higher normal form. First normal form states that every attribute should consist of a value considered atomic. "An atomic value is a value that cannot be divided." [66] Second normal form (2NF) is if 1NF is present and every non-key column is fully dependent on the primary key. Third normal form (3NF) is if 2NF is present and every non-key value is independent of each other. Higher normal form is used because 3NF is lacking in principles. Some higher normal forms are 4NF, 5NF and Boyce/Codd normal form.

Boyce-Codd normal form (BCNF) is a stricter form of 3NF as 3NF has its inadequacies. BCNF is present if the database is 3NF compliant and for every functional dependency  $A \rightarrow B$ , A should be the super key of the table.

Some integrity rules for designing a relational database is:

- Entity Integrity rule – Primary key cannot be null.
- Referential integrity rule – Each foreign key should match a primary key in referenced table. Cascading rules should be set to handle changes in a parent row.
- Business logic integrity – There should be validation and restrictions set in the business logic of an application persisting to the database. For example, a phone number must be 8-digits and not start with a zero.

#### **2.6.4.1 Database Auditing**

As government regulations grow in the software field, organizations need a way to track data that is accessed, if it is edited and by whom. A method for this is database auditing. Where an organization stores information about which data was accessed, is there a change and who performed the action. Auditing may also help data integrity by detection of security breaches, by storing old values, new values and who accessed the values. [69]

### **2.6.5 Front-end**

#### **2.6.5.1 Don Norman's Principles**

Don Norman's Principles for interaction design covers the main principles for designing everyday objects and products with the user's interaction in mind. These design principles are also used in creating designs for user interfaces for software applications. These six principles are as follows [70]:

##### **Visibility**

The principle of visibility is to make elements in the user interface easy for the user to see it and know about it. This can sometimes be difficult as there needs to be a balance between showing enough to the user so they know what they can interact with, but not overwhelming them with too many elements. This also goes the other way around where there needs to be a balance between keeping the interface tidy but avoid hiding away the elements where users cannot find them.

##### **Feedback**

The principle of feedback is to clearly communicate back to the user that their action has been registered. In software development this can be anything from on-screen confirmation, a sound or even tactile feedback in form of a vibration on mobile phones.



### **Constraints**

The principle of constraint is about limiting the interactions the user can do or to guide their actions by simplifying their possibilities. For software design this makes it clearer to the user what action they can take with certain interface elements. Here there also needs to be a balance between restricting the user but also making sure to not restrict them too much.

### **Mapping**

The principle of mapping defines how there should be a clear relationship between the controls and how their affected components. For user interfaces this means to put buttons, input fields and controllers in such a manner that a user can easily identify which component in the interface these controllers are tied to. An example would be to clearly indicate which text input field relates to which text label when the user is presented with a form which require multiple text inputs.

### **Consistency**

The principle of consistence explains how similar elements should perform similar tasks, as well as having elements be placed and look similar throughout. For applications this would mean to have a similar looking theme throughout the entire application and having buttons and navigation be placed on at the same place everywhere so a user would know where to find the elements they are looking for. Examples of consistency here would be to have a navigation menu always appear in the same spot, or to have a confirmation button always appear green.

### **Affordance**

The principle of affordance defines how objects and elements appear in such a way or gives a clue that a user "knows" how to use it. For applications this would be buttons indication that they can be clicked, arrows indicating to navigate backwards or forwards, hamburger menus to indicate there is a submenu which appears when clicked, icons indicating which action something may be related to and sliders to indicate a value of something can be adjusted up and down just to mention a few common examples of affordance commonly found in modern user interfaces.

## **2.6.5.2 Eight Golden Rules of Interface Design**

Computer scientist, Ben Shneiderman, has developed a set of eight golden rules for how to achieve good interface design within computer applications. Some of these overlap with Don Norman's Principles but are more focused on software specific design. These eight golden rules are [71]:

### **Strive for consistency**

Aim to have the same actions for similar situations, identical labels for similar inputs, menus and prompts, and a consistent colour for the entire application and elements which perform similar action.

### **Seek universal usability**

Aim to make an interface which can be used by everyone who might use the application, regardless of prior experience, age, disabilities or technological specifications of their devices. Implementing features which guide new users and empower experienced users should also be considered.

### **Offer informative feedback**

The user should get informative feedback in the interface which indicates that their action has been registered. Common and frequent actions require little feedback, while major important actions should have more noticeable feedback which in some cases might require user interaction.

### **Design dialogs to yield enclosure**

Give the user informative feedback when a group of actions has been completed to indicate to the user that their actions/inputs were handled correctly.

### **Prevent errors**

Design the application to be as fool proof as possible to prevent a user from performing an action incorrectly, or even preventing them from attempting the action in the first place if not appropriate. If the user performs an erroneous action, they should be given appropriate feedback with the possibility for recovery.

### **Permit easy reversal of actions**

Allow a user to reverse or undo an action if possible, as it allows a user to correct themselves if they end up doing a mistake on their own.

### **Keep users in control**

Make the user feel like they are "in the pilot's seat" by making the tools available to the user powerful and avoid tedious, repetitive actions or messages. Make the user be the one who controls the application and not the other way around.

### **Reduce short-term memory load**

Avoid having the user remember information between different screens within the application. The human attention span and short time memory is limited, and having a user remember information or having to navigate back and forth through different application screen makes the application tedious to use and should be avoided.

## **2.6.5.3 Material Design**

Material Design is a design language developed by Google which works as a framework/guideline on how to achieve a certain, consistent visual scheme or style whilst also aiming to unite design principles with technology. The framework focuses on a minimalistic design with inspiration from physical materials, how they appear in the real world and how to best translate it into a digital format. Examples of this is adding shadows behind elements to make them appear as if on top of each other and make motions and animations provide satisfying visual feedback to the user indication that their actions have been registered. [72] [73] [74]

## **2.6.5.4 Inversion of control**

Inversion of control (IoC) is the principle of giving away control to for example a user interface, achieving looser coupling. For example, if you have designed a REST API, accessing this through the command line tool on your machine. Inversion of control would then be to design an application which makes the calls for you, using buttons and other input methods. [75]

## **2.7 Services & Resources**

### **2.7.1 Cloud Services**

Cloud services are dynamic and scalable services available through the internet, provided by companies. Services can easily be created, removed, updated, changed and scaled based on the requirement for the customers who use the cloud service. Customers and users also only pay for the resources they use, often making cloud services a more economically viable option instead of buying their own hardware and hosting the same services on their own.

Cloud services can be divided into three main categories depending on the level of service they provide: Infrastructure as a service, platform as a service and software as a service. [76] [119]

### **2.7.1.1 Infrastructure as a service**

Infrastructure as a service provides lowest level of basic infrastructure such as networking, storage and servers. This gives high flexibility and is used when an IT department want to manage and control their own infrastructure as much as possible.

### **2.7.1.2 Platform as a service**

Platform as a service provides a platform for developers to deploy, manage and test their applications on without having to manage the underlying infrastructure, hardware or operating system.

### **2.7.1.3 Software as a service**

Software as a service delivers fully ready software solutions as a ready to use product, often in the form of ready to use web applications. Maintenance and infrastructure are managed by the provider and the end user only uses the provided software.

## **2.8 Testing**

Software testing is a process in which a person or code evaluates the functionality of software. Used to ensure that services and applications do not fail when published. [77]

### **2.8.1 Unit testing**

Unit testing is a method of ensuring that units of a software is not faulty. It is white box testing, as the software engineer writing unit tests has knowledge of the code. Often testing units consist of sending an input to a unit and ensuring a valid output. [78]

### **2.8.2 User testing**

User testing usually is black box testing of a service or an application where peers are used to test its functionality and design then give feedback. They also review user interaction and design, giving helpful feedback to developers in UX and UI. [79]

## 3 MATERIALS AND METHODS

### 3.1 *Project organization*

This project was organized through Jira and Confluence. With these tools we could manage agile workflow and share files such as diagrams and other descriptive information about the project.

#### 3.1.1 Project Team

Our team consisted of three students doing a bachelor's in computer engineering. Two supervisors, primarily Kjell Inge Tomren and secondary Girts Strazdins. And the project owner, Avento AS with Anders Beite as contact person.

Development team structure rotated where each sprint we defined one leader, one secretary and a filler role. This method was chosen so that everyone could experience each role. The leader took the role of scrum master, if the leader were not present the filler would step in.

Technical responsibilities were decided by interest. There was one person mainly focused on the REST service and database management, the second focused on documentation and front-end development and the third focused in both front-end and back-end.

#### 3.1.2 Workflow

The workflow was based upon the agile method.

Each sprint was set to last 2 weeks, starting with a customer meeting. In the customer meetings the project team presented issues completed and the result, discussed it with customer and got feedback. At the end of a meeting the participants discussed which issues to focus on the next sprint. After each customer meeting the development team would have a meeting, decide on story points and who is responsible for which issue. The project team had a daily scrum meeting about what they were working on and what they have worked on.

Jira was used to keep track of all user stories, issues and programming tasks at hand, as well as creating and keeping track of current and previous sprints.

Meeting reports can be found as Appendix 2, and sprint reports can be found as Appendix 4.

#### 3.1.3 Pre-project

At the beginning of the project a product specification was created in agreement with the customer. It contains information about how the team would work, what the goals and requirements were and an analysis of the project. The pre-project report can be found as Appendix 1.

#### 3.1.4 Project planning

Before work on the project began, plans were made to help guide ourselves as well serving as documentation and reference for the project. A requirement specification was created in conjunction with the client to ensure that we had an idea on what the functionality of the system was going to be. Furthermore, a developer agreement, a testing plan, UML diagrams and wireframes were made.

These documents can be found as Appendix 3.

Confluence was used heavily in the planning stage to easily create, keep track off and collect all plans and documentation in the same place.

### **3.1.5 Group communication**

The developer team mainly used Discord for communications and meetings as the team were already familiar with Discord through personal usage. Discord is a widely used service, which has grown in use for people and companies as it provides such a customizable container for content and people.

Discord provides free calls and screen sharing; it became an important communication platform for the group at the later stages of the project.

As most professional projects do, we also used email to contact the product owner and supervisors, as well as Slack for faster communication or smaller messages.

## **3.2 Programming**

### **3.2.1 Java**

Java is the language the Spring framework builds upon. This project used Java 11 as it is the only free long-term support release after Java 8. Java 8 became deprecated 2019. [80]

#### **3.2.1.1 Java Servlets**

Java servlet is a software component often used for implementing web containers that host web servers or applications. Servlets can communicate over any client-server protocol. Though they are most often used with HTTP. The content it responds with is often HTML, but also XML and JSON. A web container is used to deploy a servlet, this component essentially interacts with the servlet. Responsible for managing lifecycles and URL mapping. [81]

#### **3.2.1.2 Timertask**

Java timer tasks is used to schedule blocks of code to run after a certain time and/or every time interval. These tasks run on a separate thread than the main thread, providing better performance overall to the application. [82]

### **3.2.2 Flutter**

#### **3.2.2.1 Widgets**

The way Flutter builds the UI is through something they call "Widgets". Widgets are essentially UI components which can be combined and stacked to compose a fully functioning UI. Everything from visible structural elements such as text and buttons, stylistic elements which provide colours or a theme, and layout widgets such as padding, and alignment is considered widgets in flutter. All widgets in Flutter must return itself via the "build" method. There exist two main type of widgets in Flutter: stateful widgets and stateless widgets. [83]

#### **3.2.2.2 Stateless widgets**

Stateless widgets in Flutter are widgets which do not have a mutable state, and which does not have any properties which will change after the widget has been built. When the widget is created it will always remain with its initial configuration. Stateless widgets are useful when creating static UI elements. [84]

### **3.2.2.3 Stateful widgets**

Stateful widgets are widgets which provide a mutable state which might change over time, the state being the information the widget carries. Every time the state of a stateful widget is updated it must be done through a "setState" method, which updates the state and notifies that it has been updated so the widget can be re-rendered with the changes. Stateful widgets are useful when building dynamically changing UI elements. [85]

## **3.3 Third-party services**

### **3.3.1 RouteXL**

RouteXL is a third-party service that provides tools for navigation. Most useful is their optimized route API call. Google Maps Services does not provide any tools for quickest route between multiple locations. Which is why we chose RouteXL to calculate the quickest route between multiple locations. [86]

### **3.3.2 Frost API**

The Frost API provides free access to MET Norway's weather and climate data. Which is quite useful when providing a service for snow ploughing. As we can aggregate data from the Frost API and include it in our business logic. [87]

### **3.3.3 Google Maps Services**

Google maps services is probably the biggest service within navigation by road. As they have a large amount of geo data about roads, services and driving patterns. Which makes it the most reliable navigation service for our application. [88]

### **3.3.4 Kartverket**

Kartverket is a Norwegian data service containing information about properties, positional data and more. They are the national geodata coordinator in Norway. Meaning they are partly responsible for the geographical infrastructure in Norway. [89]

### **3.3.5 Google cloud platform**

Google Cloud Platform is one of the biggest cloud providers on the market, and provides infrastructure, platform and software as a service. [90]

We primarily chose Google cloud platform for our previous experience with it, and that it provided students with 3000 free credits for use on the platform. The credits were enough for our use on this project.

## **3.4 Coding Convention**

We chose to try to follow Google's coding convention for Java for the back-end application, and mostly the default coding and naming conventions for Flutter and Dart.

Google's Java Style guide can be found in [91].

Flutter naming and formatting conventions can be found in [92].

### 3.4.1 Class names

Class names are set to use camel casing starting with an upper case.

```
public class CodingConvention {  
  
}
```

Figure 12: Coding convention classes

### 3.4.2 Method names

Method names are set to use camel casing starting with a lower case.

```
public void methodName() {  
  
}
```

Figure 13: Coding convention methods

### 3.4.3 Field names

Constants such as the FINAL\_STRING is to use CONSTANT\_CASE, which means all characters are upper case and to separate words with an underscore.

Variables are to use camel casing starting with lower case, such as the localVariable presented below.

```
private final String FINAL_STRING = "string";  
  
private int localVariable = 1;
```

Figure 14: Coding convention fields

## 3.5 Frameworks & Dependencies

### 3.5.1 Spring Framework

Spring framework provided the project with some inversion of control as Spring is known for its dependency injection. Spring also provides the developers with some out of the box configuration. Allowing any developer to start working on business logic quite early in a project.

Spring Boot is a tool that allows a project team to create an application that you can just run. It also comes with embedded Servlets, such as Apache Tomcat. [93]

Spring Data can provide you with tools to interact with data. It contains custom object-mapping and repository abstractions. Provides auto generated methods for repository classes, service classes and REST classes. Which also are customizable. [94]

Spring Security provides support for authentication and authorization. It is integrated with the servlet API and can also be integrated with the Spring MVC. [95]

Spring Mail made API calls to send emails quite easy to use.

### 3.5.2 Back-End Dependencies

The **JSON** dependency provided JSON encoders and decoders in Java. Which is important when transferring JSON between front-end and back-end.

The **HTTP client** provides an API for making calls to third party services from the back-end. Allowing the service to also act as a proxy server.

**Google maps services** was used to handle all calls to the maps service. Through a quite simple API, allowing more inversion of control.

**Json Web Token** dependency provided an easy to use tool to encode and decode tokens.

The **MySQL connector** allows any project member to easily build the project, as less setup is required for communication between the back-end and the database.

**Junit** is a testing framework for java.

## 3.6 Environments

### 3.6.1 MySQL

MySQL is a relational database management system that is open-source.

MySQL Workbench is a graphical user interface that provides tools to manage, edit and create relational databases visually.

InnoDB is a database engine that the database management system uses to create, delete, update and read data. InnoDB can recover from a crash by replaying its logs.

InnoDB stores rows in primary key order, which can be fast for common operations. [96] [97]

### 3.6.2 Postman

Postman is a platform which allows developer to design and document their API, as well as creating a mock server to simulate what their API could behave like, even when the actual back-end is in development.

This useful as it allows the back-end and front-end to be developed in parallel as long as they follow the common API which has been designed. [98]

### 3.6.3 Gitlab

Gitlab is an online version control tool. Gitlab can be used by a team to tracks changes and additions to their code base. Gitlab allows for the creation of private repositories, both for single users and for teams.

One of the other main features of Gitlab is the built in DevOps functionalities which enables CI/CD to be added to the project with very little setup. [99] [100]

### 3.6.4 Draw.io

Draw.io is a free to use website which provides powerful and easy to use tool to create sketches, diagrams and wireframes. The main features we used Draw.io for was to create wireframes and UML diagrams.

Draw.io can be added to the Atlassian toolbox together with Jira and Confluence, allowing Draw.io sketches to appear within Confluence where it will be saved together with the rest of the documentation.



### **3.6.5 Eclipse**

Eclipse is an IDE developed by the eclipse foundation and is commonly used to develop Java applications. Eclipse was the primary IDE for the development of the back-end application for this project. [101]

### **3.6.6 Android Studio**

Android Studio is an IDE mainly focused on development of Android applications, built on JetBrains' IntelliJ IDE.

In the beginning we decided to use Android Studio to develop the front-end application, as it was heavily focused on Android development as well as supporting the Flutter SDK and additional libraries we meant to use. Another feature which made Android Studio appealing was the possibility of installing and running an Android emulator. However, we ended up dropping Android Studio as memory consumption became an issue, and 8 GB of ram ended up not being enough to reliably run Android Studio together with other supporting software, thus resulting in a lot of crashed, which made us turn over to Visual Studio Code. [102]

### **3.6.7 Visual Studio Code**

Visual Studio Code is a lightweight code editor with a lot of focus on extension, allowing developers to install only the extension they need, keeping the editor light weight and the resource usage down.

After having run into memory issues with Android Studio, Visual Studio Code was more light weight and a lot easier to run on a laptop with only 8 GB of ram. Visual Studio Code also supported all the features, extensions and libraries we needed, and thus ended up becoming our editor of choice for the front-end development of the project. [103]

## 4 RESULTS

### 4.1 General System Design

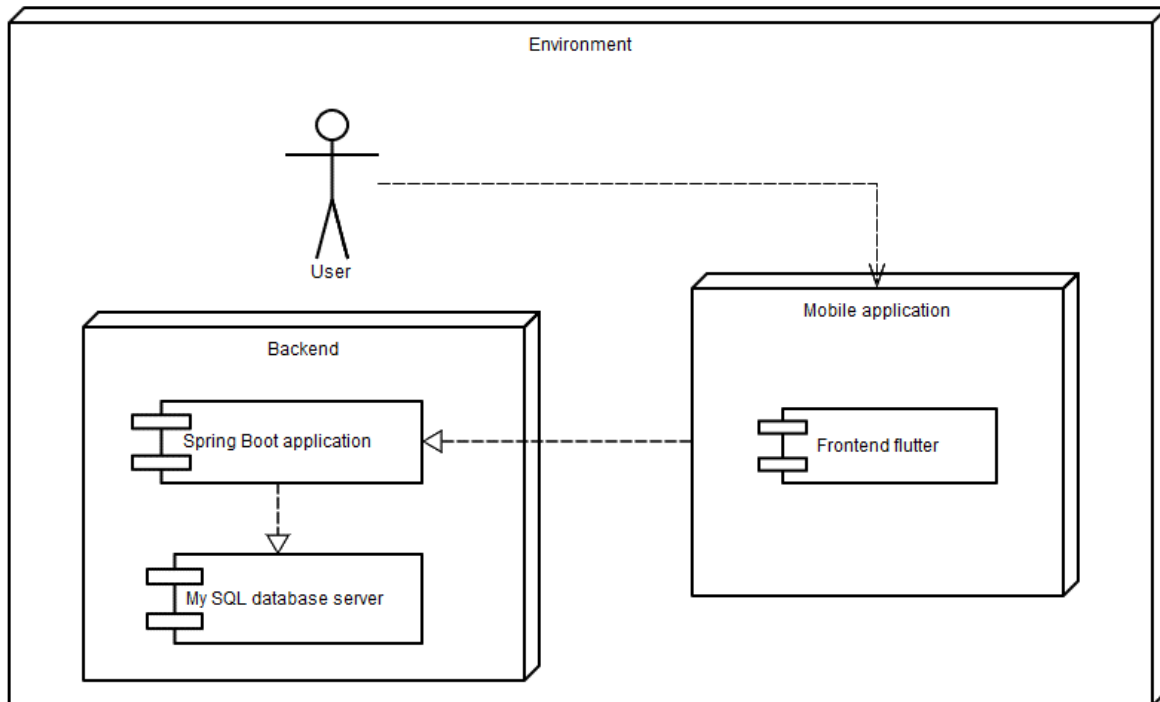


Figure 15: Deployment Diagram of the system

The general design of the system consists of two main components: The front-end and the back-end. The front-end is a mobile application created using Flutter, which communicates asynchronously with the back-end.

The back-end consists of two main components as well: The back-end Spring Boot application and the MySQL database server which the Spring Boot application communicates with.

### 4.2 Server

#### 4.2.1 Infrastructure

Google Cloud Console were used to generate the servers we needed. Terraform were used as Infrastructure as Code to create servers in GCP.

Gitlab CI/CD was used to generate and update the server when changes to the git repository master branch were made. Any updates on the master branch would run the pipeline and update the server.

The Pipeline would automatically understand that the server was created or not and then create or update the server infrastructure.

#### Tasks performed in Terraform:

##### Validate:

checks if correct syntax. And check if valid Terraform code. [104]

```
63 $ terraform validate
64 Success! The configuration is valid.
```

Figure 16: Syntax validation

**Plan:**

Test the Terraform code to see what can be created, does not push to GCP. [105]

```
Plan: 3 to add, 0 to change, 0 to destroy.
```

*Figure 17: Terraform Test*

**Apply:**

Push the plan file to GCP and creates the resources. [106]

```
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

*Figure 18: Successful deployment*

**Plan\_destroy:**

This is created after Apply, it checks the resources on GCP and creates a plan file which we can run later to remove those resources it checked for. [107]

```
Plan: 0 to add, 0 to change, 7 to destroy.
```

*Figure 19: Destroy resources*

**Destroy:**

Similar as Apply, just for destroy, it runs the plan\_destroy and removes the resources from GCP. [108]

```
Apply complete! Resources: 0 added, 0 changed, 7 destroyed.
```

*Figure 20: Successful destruction*

**Terraform files [109]:**

- **Main.tf**

The main configuration file, where everything is declared.

Declare boot\_disk with what Image to boot the VM in, for instance Ubuntu 16.04 LTS, if it should have an external IP, SSH keys attached to them, which resource to use (GCP), labels and descriptions, what network they should be part of and so on.

- **Variable.tf**

This is where all variables are stored, it is how Terraform deal with variables.

Using variables in other terraform files are easy when this file is in the same folder. Using these variables, we call the variable by var."name" in other files.

```

variable.tf 550 Bytes
1  #Variables used for files sharing same parent folder.variable
2
3  variable "image" {default = "ubuntu-os-cloud/ubuntu-1804-lts"}
4
5  variable "machine_count" {default = "1"}
6
7  variable "gce_ssh_pub_key_file" {default = "~/.ssh/id_rsa.pub"}
8  variable "gce_ssh_user" {default = "henrik2890"}
9
10 variable "service_account" {default = "~/server-infrastructure/credentials/service_account.json"}
11
12 variable "name" {default = "database"}
13
14 variable "machine_type" {
15     type = map
16     default = {
17         standard-1 = "n1-standard-1"
18         prod = "f1-micro"
19     }
20 }

```

Figure 21: Variable.tf

- **Provider.tf**

Tell us which provider to use, for this instance we use Google Cloud Platform and we also need to specify which user to authenticate as and what project the user is part of.

```

provider.tf 169 Bytes
1  provider "google" {
2      project = "terraform-gcp-256910"
3      region = "europe-west-2-a"
4      zone     = "europe-west4-a"
5      credentials = file(var.service_account)
6  }

```

Figure 22: Provider.tf

- **Outputs.tf**

When running the commands "terraform apply", "terraform destroy" or any other commands the outputs will be posted as prints after the command is completed. These outputs are primarily used for debugging.

```

outputs.tf 305 Bytes
1  output "name" {value = "${google_compute_instance.default.*.name}"}
2  output "machine_type" {value = "${google_compute_instance.default.*.machine_type}"}
3  output "zone" {value = "${google_compute_instance.default.*.zone}"}
4  output "instance_id" {value = "${join(" ", google_compute_instance.default.*.id)}"}

```

Figure 23: Outputs.tf

```

82  Outputs:
83  instance_id = projects/terraform-gcp-256910/zones/europe-west4-a/instances/database
84  machine_type = [
85    "n1-standard-1",
86  ]
87  name = [
88    "database",
89  ]
90  zone = [
91    "europe-west4-a",
92  ]

```

Figure 24: Actual output

### The Gitlab pipeline:

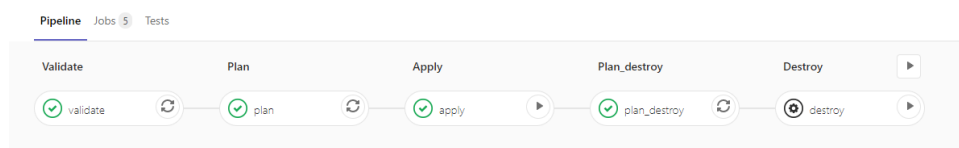


Figure 25: Gitlab pipeline

This pipeline is run when a change is detected on the master branch of the GitLab infrastructure repository. The purpose of the pipeline is automation.

There are numbers of features that Terraform can contribute to the Infrastructure it can add Network, IP, SSH, Firewall, OS and more.

### Google Cloud Platform:

GCP is where we create our server, it is just done through Terraform which uses the API from which GCP provides to manage our servers as "Infrastructure as Code". With the GUI of GCP it is totally possible to do all of this manually. [110] [111]

Name	Zone	Recommendation	In use by	Internal IP	External IP	Connect
<input checked="" type="checkbox"/> database	europe-west4-a			10.164.0.17 (nic0)	34.91.44.48	SSH

Figure 26: Google Cloud Platform

Name	Image	Size (GB)	Device name	Type	Encryption	Mode	When deleting instance
database	ubuntu-1804-bionic-v20200414	10	persistent-disk-0	Standard persistent disk	Google managed	Boot, read/write	Delete disk

## 4.2.2 Configuration

Ansible were used to push configuration to the server.

Inventories hold data of nodes. inventories are used to target servers to which ansible should push configuration to.

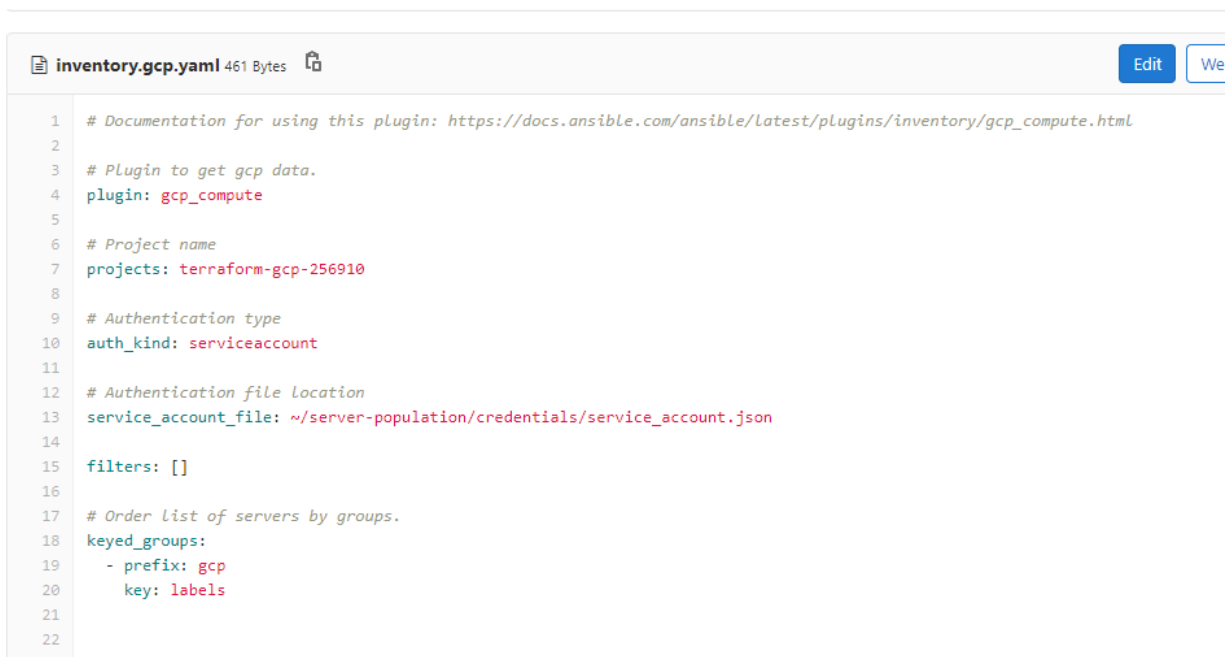
inventories can get the servers from gcp or we can target specific ones by filtering by the tag, name or group it belongs to. Here we were using the inventory.gcp.yaml file which is a dynamic inventory module for getting the servers from gcp and will get the ip addresses for us. [112]

The command in green, will show a graph over which servers are in this inventory file.

```
$ ansible-inventory -i ansible/inventory/inventory.gcp.yaml --graph
@all:
  |--@gcp_machine_type_n1_standard_1:
  |   |--34.91.44.48
  |--@gcp_name_database:
  |   |--34.91.44.48
```

Figure 27: Servers in description file

The inventory.gcp.yaml file structure and content.



```
inventory.gcp.yaml 461 Bytes
1 # Documentation for using this plugin: https://docs.ansible.com/ansible/latest/plugins/inventory/gcp_compute.html
2
3 # Plugin to get gcp data.
4 plugin: gcp_compute
5
6 # Project name
7 projects: terraform-gcp-256910
8
9 # Authentication type
10 auth_kind: serviceaccount
11
12 # Authentication file location
13 service_account_file: ~/server-population/credentials/service_account.json
14
15 filters: []
16
17 # Order list of servers by groups.
18 keyed_groups:
19   - prefix: gcp
20     key: labels
21
22
```

Figure 28: Inventory.gcp.yaml structure and content

### Playbook:

Playbooks are scripts that can be reused on different inventory. To be able to target a specific inventory we include it in the command.

Playbooks are a way of configuring a server. If you know what you want on the server, you can define a playbook of commands that will install those dependencies and

packages, running the playbook again will not reinstall everything it will only get the missing packages and update the outdated ones.

Command to run playbook:

```
ansible-playbook ansible/playbook/mysql.yaml -i ansible/inventory/inventory.gcp.yaml
```

```
$ ansible-playbook ansible/playbook/mysql.yaml -i ansible/inventory/inventory.gcp.yaml
```

Figure 29: Running playbook



```
mysql.yaml 802 Bytes
1 -
2   name: MySQL playbook
3   hosts: all
4
5
6   tasks:
7     - name: Install dependencies
8       apt:
9         pkg:
10          - python
11          - python-setuptools
12          - python-dev
13          - build-essential
14          - python-pip
15          - python-mysqldb
16          state: latest
17
18     - name: Install MySQL database
19       apt:
20         pkg:
21          - mysql-server
22          - mysql-client
23          state: present
24
25     - name: Start Mysql Service
26       service:
27         name: mysql
28         state: started
29         enabled: yes
30
31     - name: Create Application Database
32       mysql_db:
33         name: employee_db
34         state: present
35
36     - name: Create Application DB User
37       mysql_user:
38         name: db_user
39         password: qwerty123
40         priv: '*.*:ALL'
41         host: '%'
42         state: 'present'
```

Figure 30: Playbook.yaml

The image I were using can be found in [113].

We created two servers one for running as host and the other as node where we installed the server back-end on with MYSQL and more.

Ansible pushes changes from a host to a node to check if it needs to be created, updated or do nothing on the node. The apt, pgk: will do this for us, it will check to see if it is there, what version it is running then run the state: which I have set to latest to then update to latest version. If no package is found it will install the desired version.

### 4.3 API

When designing the API, we used a service called Postman. Postman allowed us to create documentation and a mock server with example requests for our API, allowing us to figure out what our API was going to look like early on, and allow us to create a front-end using API data, while the back-end is being worked on. This allowed us to work on the front and back-end of the project in parallel.

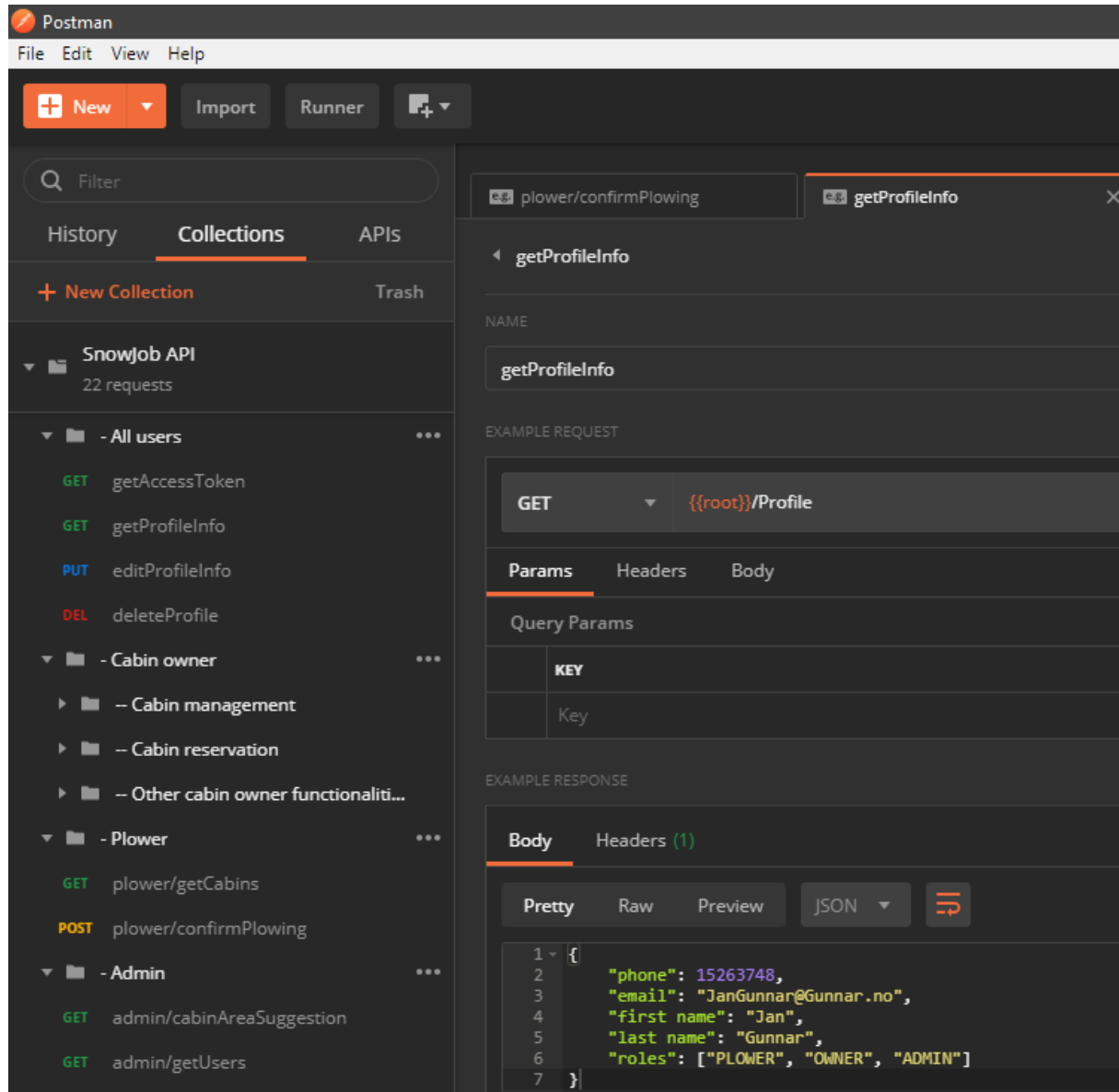


Figure 31: Postman interface

As mentioned, Postman can generate an API documentation based on the created API collection and the examples, which served us as a fast reference on how to implement the API on the front and back-end.



**POST** owner/addCabin Comments 0

```

{{root}}/Owner/Cabins/add
            
```

Request to add a new cabin under the user

Headers

Content-Type	application/json
--------------	------------------

**PUT** owner/updateCabin Comments 0

```

{{root}}/Owner/Cabins/Edit
            
```

Request to update cabin information for a single cabin

Example Request owner/addCabin

```

curl --location --request POST '{{root}}/Owner/Cabins/add' \
--header 'Content-Type: application/json' \
--data-raw '{
  "cabin_name": "Hytta :)",
  "cabin_field": "Oppdal",
  "gardsnr": 43,
  "bruksnr": 2,
  "festenr": 0,
  "seksjonsnr": 0,
  "næstnr": 7450
}'
            
```

Example Request owner/updateCabin

```

curl --location --request PUT '{{root}}/Owner/Cabins/Edit' \
--header 'Content-Type: application/json' \
--data-raw '{
  "cabin_id": 17347236569696966,
  "cabin_name": "Hytta :D",
  "festenr": 2,
  "seksjonsnr": 1
}'
            
```

Figure 32: Snippet from generated API documentation

## 4.4 Back-end

### 4.4.1 Database

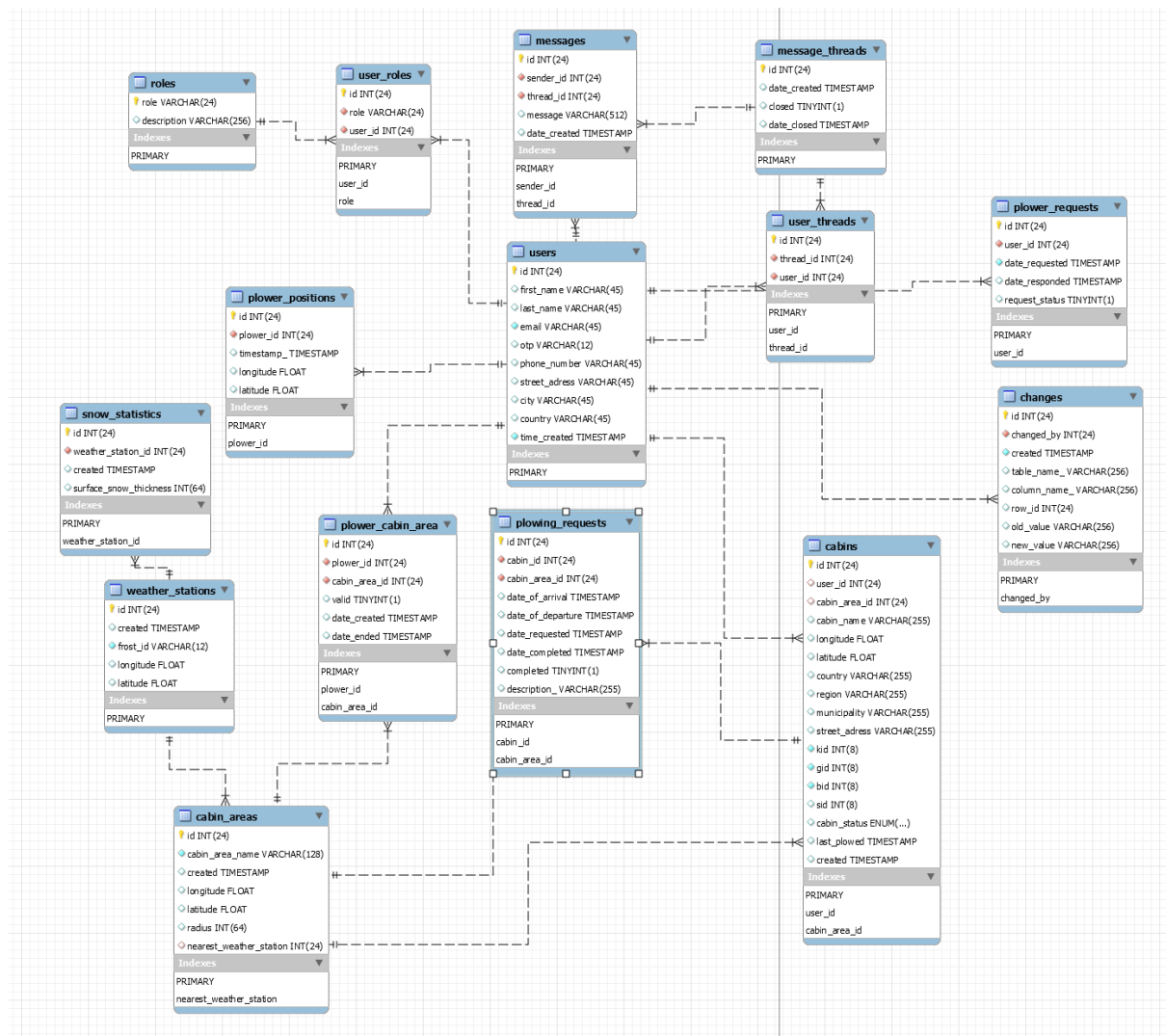


Figure 33: EER diagram

The database is designed with object-oriented programming in mind. All entities created are mapped to a POJO. Most entities are linked to a simple object, with some joined tables for user and role mapping, also for ploughman and cabin area mapping. The base objects for this application are the user and cabin. All names are written as descriptive as possible, to ensure correct maintenance and further development. All columns are designed to represent a value as simple as possible.

#### 4.4.1.1 Users table

The users table is created with an excess of fields, as which data were necessary was unclear at the beginning of this project. Address information about the user was unused but included if needed.

- Id – Primary key with a size of 24 bits.
- First\_name – Varchar of 45 characters
- Last\_name – Varchar of 45 characters
- Email – Needed for authentication and receiving an OTP. Varchar of 45 characters.
- Otp – Varchar of 12 characters, was not saved as integer because an integer would not preserve unnecessary information as leading zeros.
- Phone\_number – Varchar of 45 characters, could have limited size more, but this restricts international clients. Included because OTP over text messages would be easier for users. Not implemented as authentication method in development process because of costs.
- Street\_adress – Varchar of 45 characters
- City – Varchar of 45 characters
- Country – Varchar of 45 characters
- Time\_created – timestamp set to hold information about when a user is created.

#### 4.4.1.2 Cabins Table

The cabins table is designed to hold cabin information and location information.

- Id – Primary key with a size of 24 bits.
- User\_id – Foreign key to the id attribute in the users table.
- Cabin\_area\_id – Foreign key to the id attribute in the cabin\_areas table.
- Longitude – float to hold positional data.
- Latitude – float to hold positional data.
- Country – Varchar of 45 characters.
- Region – Varchar of 45 characters.
- Municipality – Varchar of 45 characters.
- Street\_adress – Varchar of 45 characters.
- Kid - ("Kommunenummer") integer of 8 bits.
- Gid - ("Gardsnummer") integer of 8 bits.
- Bid - ("Bruksnummer") integer of 8 bits.
- Sid - ("Seksjonsnummer") integer of 8 bits.
- Cabin\_status – ENUM which can be a number of colors written in strings (red, yellow, green, gray, black). Should be set based on business logic using snow\_statistics table values and last\_plowed value.
- Last\_plowed – timestamp set to hold information of when last plowed.
- Created – timestamp set to hold information of when a cabin is created.

(Some values are written in Norwegian, as they do not have any appropriate translation.)

#### 4.4.1.3 Roles table

Roles table is created with further development in mind.

- Role – Primary key, varchar of 24 characters. Holds information about what the role is.
- Description – Varchar of 256 characters. Holds descriptive information about a role. Short strings might not describe a role enough, this will help maintenance and security.

#### 4.4.1.4 User\_roles table

This is a many-to-many junction table which holds the mapping between a user and a role.

- Id – Primary key, integer of 24 bits.
- Role – Foreign key, holds a role attribute from the roles table.
- User\_id – Foreign key, holds an id attribute from the users table.

#### 4.4.1.5 Plower\_positions table

This table was intended to hold ploughmen positions through their daily routine. This could provide managers and developers with details about routes, time used and other patterns. Using those details could help develop a custom route planning API and provide managers with details of how to increase efficiency.

- Id – Primary key, integer of 24 bits.
- Plower\_id – Foreign key, integer of 24 bits.
- Timestamp – timestamp, time the positional data is saved.
- Longitude – float, positional data.
- Latitude – float, positional data.

#### 4.4.1.6 Messages table

Table made to store messages as specified in the product specification. Two or more users should be able to communicate through the application. Not implemented due to time limitations and a decrease in efficiency.

- Id – Primary key, integer of 24 bits.
- Sender\_id – Foreign key, integer of 24 bits, references id in the users table.
- Thread\_id – Foreign key, integer of 24 bits, references id in the message\_threads table.
- Message – Varchar of 512 characters, contains the message.
- Date\_created – Timestamp of when the message was created.

#### 4.4.1.7 Message\_threads table

The message threads table links multiple messages and users to a conversation.

- Id – Primary key, integer of 24 bits.
- Date\_created – Timestamp of when the thread was created.
- Closed – Tinyint, a Boolean value to inform if a thread is open. If closed all new transactions related to the thread should stop.

#### 4.4.1.8 User\_threads table

This is a many-to-many junction table, holding information about who belongs to a thread.

- Id – Primary key, integer of 24 bits.
- Thread\_id – Foreign key, integer of 24 bits, references id in the message\_threads table.
- User\_id – Foreign key, integer of 24 bits, references id in the users table.

#### 4.4.1.9 Plower\_requests table

This table should hold all requests of users wanting access to ploughman functionality.

- Id – Primary key, integer of 24 bits.
- User\_id – Foreign key, integer of 24 bits, references id in the users table.
- Date\_requested – Timestamp of when an instance is created.
- Date\_responded – Timestamp of when a response is created by an admin.
- Reques\_status – Tinyint, Boolean value if the request is denied or approved. Could be a more descriptive name.

#### 4.4.1.10 Changes table

This is an auditing table used to keep data integrity and security, mentioned in 2.6.2.1. Business logic for clearing the table after a time period is not completed, as it is not secure to store large amounts of sensitive data for a prolonged period.

- Id – Primary key, integer of 24 bits.
- Changed\_by – Foreign key, references id in the users table.
- Created – Timestamp of when the change went through.
- Table\_name\_ – Varchar of 256 characters, contains information of the table edited.
- Column\_name\_ – Varchar of 256 characters, contains information of the attribute edited.
- Row\_id – Integer of 24 bits, contains information about the tuple edited.
- Old\_value – Varchar of 256 characters, contains the previous value of a tuple.
- New\_value – Varchar of 256 characters, contains the new value of a tuple.

#### 4.4.1.11 Plowing\_requests table

This table holds information about a request sent by a cabin owner to plough his driveway.

- Id – Primary key, integer of 24 bits.
- Cabin\_id – Foreign key, references id in the cabins table.
- Cabin\_area\_id – Foreign key, references id in the cabin\_areas table.
- Date\_of\_arrival – Timestamp of when a cabin owner arrives at a cabin.
- Date\_of\_departure – Timestamp of when a cabin owner departs.
- Date\_requested – Timestamp of when the request was created.
- Date\_completed – Timestamp of when a request is completed.
- Description\_ – Varchar of 255 characters. Optional field for a cabin owner to include in the request.

#### 4.4.1.12 Cabin\_areas table

This table holds information about cabin areas. Longitude and latitude should be the centre of a cabin area. The radius should describe its area.

- Id – Primary key, integer of 24 bits.
- Cabin\_area\_name – Varchar of 128 characters, holds the name if a cabin area.

- Created – Timestamp of when a cabin area instance is created.
- Longitude – Float, positional data.
- Latitude – Float, positional data.
- Radius – Integer of 64 bits.
- Nearest\_weather\_station – Foreign key, integer of 24 bits, references id in the weather\_stations table. Value is set by business logic after a cabin area is added.

#### 4.4.1.13 Plower\_cabin\_area table

This is a many-to-many junction table. It holds information about which cabin areas ploughmen are working in.

- Id – Primary key, integer of 24 bits.
- Plower\_id – Foreign key, references id in the users table.
- Cabin\_area\_id – Foreign key, references id in the cabin\_areas table.
- Valid – Tinyint, Boolean deciding if a ploughman is still related to a cabin area.
- Date\_created – Timestamp of when an instance is created.
- Date\_ended – Timestamp of when the valid attribute is set to false.

#### 4.4.1.14 Weather\_stations table

This table stores information of weather stations retrieved from the Frost API. This is populated by business logic on the back-end.

- Id – Primary key, integer of 24 bits.
- Created – Timestamp of when a weather station instance is created.
- Frost\_id – Varchar of 12 characters, is the id the Frost API uses.
- Longitude – Float, positional data about the weather station.
- Latitude – Float, positional data about the weather station.

#### 4.4.1.15 Snow\_statistics

This table stores information about snow thickness day to day. Business logic on the back-end populates this table.

- Id – Primary key, integer of 24 bits.
- Weather\_station\_id – Foreign key, references id in weather\_stations table.
- Created – Timestamp of when an instance is created.
- Surface\_snow\_thickness – Integer of 64 bits, stores information about the snow thickness retrieved from the Frost API.

## 4.4.2 Back-end application

The back-end application is designed after a service layered approach. Where business logic is put into their appropriate container. This approach provides good readability, high cohesion and low coupling.

### 4.4.2.1 Service layered design

```

@Component
public class JwtRequestFilter extends OncePerRequestFilter{

    @Autowired
    private MyUserDetailsService myUserDetailsService;

    @Autowired
    private JwtUtil jwtUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {

        final String authorizationHeader = request.getHeader("Authorization");

        String username = null;
        String jwt = null;

        if(authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
            // retrieves jwt where token begins in auth header
            jwt = authorizationHeader.substring(7);
            username = jwtUtil.extractUsername(jwt);
        }

        if(username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = this.myUserDetailsService.loadUserByUsername(username);

            if(jwtUtil.validateToken(jwt, userDetails)) {

                UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
                usernamePasswordAuthenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
            }
        }

        filterChain.doFilter(request, response);
    }
}

```

Figure 34: Security filter

Security filter is the topmost layer of the back-end application. It filters every request based on the HTTP configuration file. Every path that does not permit all, such as login requests and create user requests are filtered. Requests goes through this filter for authorization. Resulting in a HTTP status code of 200 if authorized or a status code of 401 unauthorized as stated in the HTTP standard. The filter checks for an Authorization header starting with "Bearer ". This token is a hashed string, containing username (email), roles and expiration time for a token.

```

@RestController
@RequestMapping("/cabin")
public class CabinRestController {

    @Autowired
    private CabinService cabinService;

    @Autowired
    private JwtUtil jwtUtil;

    @Autowired
    private JpaDtoMapper jpaDtoMapper;

    @Autowired
    private MyUserService myUserService;

    * @param request[]
    public ResponseEntity<> deleteCabin(HttpServletRequest request, @PathVariable(value="id") int id) {}

    * @param request[]
    public ResponseEntity<> saveCabin(HttpServletRequest request, @RequestBody Cabin cabin) {}

    * @param request[]
    public ResponseEntity<> updateCabin(HttpServletRequest request, @RequestBody CabinDTO cabinDTO) {}

    * @param request[]
    public ResponseEntity<> getCabin(HttpServletRequest request, @PathVariable(value="id") int id) {}

    * @param request[]
    public ResponseEntity<> getCabins(HttpServletRequest request) {}

    * @param request[]
    public ResponseEntity<> getCabinsByMyUser(HttpServletRequest request, @PathVariable(value="id") int id){}

    * @param request[]
    public ResponseEntity<> getSortedCabins(HttpServletRequest request, @RequestBody SortedQueryRequest sortedQueryRequest) {}

    * @param request[]
    public ResponseEntity<> getCabinsByCabinNameSortedByCabinName(HttpServletRequest request, @PathVariable(value="name") String name) {}

```

Figure 35: Controller layer

The REST controller is one layer in the application. This layer contains path mapping of resources which is accessed through the service layer. The controller also accesses an object mapper named JpaDtoMapper. The object mapper maps all entities to a data transfer object. Providing more control of the information flow and solves the infinite recursion problem when converting entities that have bi-directional associations to JSON.

```

public interface CabinService {

    public List<CabinDTO> findAll();

    public CabinDTO getCabin(int id);

    public void saveCabin(Cabin cabin);

    public void updateCabin(CabinDTO cabinDTO);

    public void addLocation(Cabin cabin);

    public void deleteCabin(CabinDTO cabinDTO);

    public List<CabinDTO> getCabinsByMyUser(int id);

    public List<CabinDTO> getCabinsSortedBy(SortedQueryRequest sortedQueryRequest);

    public List<CabinDTO> getCabinsByCabinNameSortedByCabinName(String name);

    public void deleteCabinById(int id);

    public Cabin getCabinById(int id);

```

Figure 36: Service layer

The service layer accesses resources through the repository layer, applying business logic and returns resources to the controller layer.

```
public interface CabinRepository extends JpaRepository<Cabin, Integer>, PagingAndSortingRepository<Cabin, Integer> {  
    public List<Cabin> findByCabinNameOrderByCabinName(String cabinName);  
}
```

Figure 37: Repository layer

The repository layer extends JpaRepository and PagingAndSortingRepository. Extending these repositories allows us to access repository functionality and handles all transactional business logic. This applies the inversion of control principle and provides us with CRUD methods, pagination and sorting of resources. It also lowers redundancy of code, as the data access object classes are included in the API. Reducing our total code by at least 15 classes.

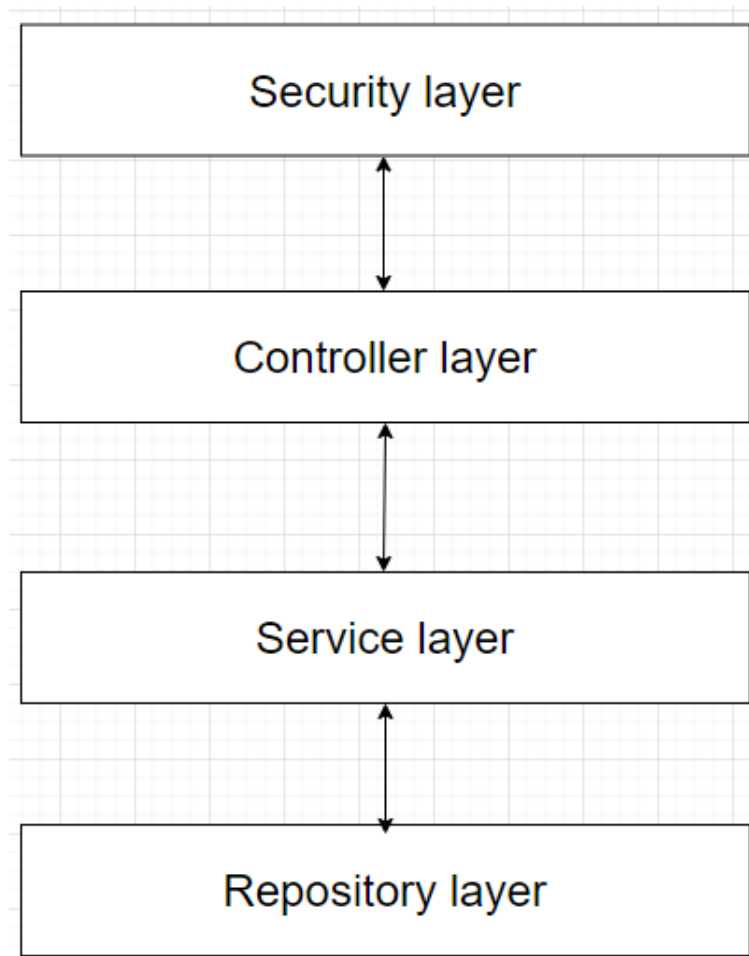


Figure 38: Layered Service Design



#### 4.4.2.2 Business logic

Accessing third party services should happen through a proxy server. This reduces the risk of an API key coming into the hands of a malicious individual or organization through datamining the front-end application. Abusing an API key could result in down time for the service and extra costs.

Upon creation of a new cabin object, a scheduler schedules a call to Kartverket's API. By using a scheduler, the task is handed to a new thread which allows the main thread to keep accepting requests. The service displayed below provides any cabin with location data consisting of longitude and latitude. A ploughman can then use the positional data or our routing service to locate any cabin which should be ploughed. The service also provides cabins with a street address and municipality. This is mostly for any user to identify a specific cabin if they own two or more.

```
@Service
public class CabinPositionService {

    public CabinPositionService() {}

    public Cabin getLongLatForCabin(Cabin cabin) throws ClientProtocolException, IOException {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        // %20 = space
        HttpGet httpGet = new HttpGet(
            "https://ws.geonorge.no/adresser/v1/sok?sokemodus=AND&kommunennummer=" +
            cabin.getKid() +
            "&gardsnummer=" +
            cabin.getGid() +
            "&bruksnummer=" +
            cabin.getBid() +
            "&treffPerSide=10&side=0&asciiKompatibel=true"
        );
        CloseableHttpResponse response = httpClient.execute(httpGet);

        JSONObject jsonObject = null;

        try {
            if(response.getStatusLine().getStatusCode() != 200) {
                return null;
            }

            HttpEntity entity = response.getEntity();

            String responseString = EntityUtils.toString(entity);

            jsonObject = new JSONObject(responseString);
            EntityUtils.consume(entity);
        } catch (ClientProtocolException cpe){
            return null;
        } catch (IOException ioe){
            return null;
        } finally {
            response.close();
        }

        if(jsonObject != null) {
            JSONObject address = jsonObject.getJSONArray("adresser").getJSONObject(0);
            cabin.setLongitude(Double.parseDouble(address.getJSONObject("representasjonspunkt").get("lon").toString()));
            cabin.setLatitude(Double.parseDouble(address.getJSONObject("representasjonspunkt").get("lat").toString()));
            cabin.setCountry("Norge");
            cabin.setStreetAddress(address.get("adresseskjetutenadressesilleggsnavn").toString());
            cabin.setMunicipality(address.get("postnummer") + " " + address.get("poststed"));

            return cabin;
        }
        return null;
    }
}
```

Figure 39: Kartverket request

As mentioned above, a ploughman can use the routing service which provides a sorted array with the optimal route given any unsorted list of locations. This service uses the RouteXL API to process quickest path for the list of locations. RouteXL's service has a

free tier allowing 10 locations per request, but the number of requests is unlimited. The result of the routing service is then processed by Google's Directions service.

```

@Component
public class RoutePlanner {

    public List<RouteXLLocation> getQuickstartRoute(List<? extends BasicNameValuePair> request, List<RouteXLLocation> locations) throws ClientProtocolException, IOException {
        CloseableHttpClient httpClient = HttpClients.createDefault();
        HttpPost httpPost = new HttpPost("https://api.routexl.com/tour");
        httpPost.addHeader(HttpHeaders.AUTHORIZATION, "Basic " + new String(Base64.encodeBase64(("snpl0g2020" + ":" + "gewoftjmdckweiiu").getBytes(StandardCharsets.ISO_8859_1))));
        httpPost.setHeader("Content-type", "application/x-www-form-urlencoded");
        httpPost.setEntity(new UrlEncodedFormEntity(request));

        CloseableHttpResponse response = httpClient.execute(httpPost);
        JSONObject jsonObject = null;

        try {
            if(response.getStatusLine().getStatusCode() != 200) {
                return null;
            }
            HttpEntity entity = response.getEntity();
            String responseString = EntityUtils.toString(entity);
            jsonObject = new JSONObject(responseString);
            EntityUtils.consume(entity);
        } catch (ClientProtocolException cpe){
            return null;
        } catch (IOException ioe){
            return null;
        } finally {
            response.close();
        }

        JSONObject route = jsonObject.getJSONObject("route");
        List<Integer> cabinIds = new ArrayList<>();

        for(int i = 0; i < locations.size(); i++) {
            cabinIds.add(Integer.parseInt(route.getJSONObject(i + "").getString("name")));
        }

        return sortLocations(cabinIds, locations);
    }

    public void getRouteXLStatus() throws ClientProtocolException, IOException {}

    private List<RouteXLLocation> sortLocations(List<Integer> cabinIds, List<RouteXLLocation> locations){
        List<RouteXLLocation> result = new ArrayList<>();

        for(Integer id : cabinIds) {
            for(RouteXLLocation location : locations) {
                if(location.getCabinId() == id) {
                    result.add(location);
                    break;
                }
            }
        }

        return result;
    }
}

```

Figure 40: RouteXL request

As RouteXL returns an ordered array of locations, it is then handed to the DirectionsService. Which in turn uses Google's Directions API fetching decoded polylines which is then in turn rendered by the front-end application. Google's proxy server is incomplete as the front-end application still uses an API key.

```

@Service
@Scope(value = ConfigurableBeanFactory.SCOPE_SINGLETON)
public class DirectionsService {

    private GeoApiContext context;

    private final String API_KEY = "AIzaSyC9odZj0j7J4q5L_UKHrn_Ha2ygmZy1PGM";

    public DirectionsService() {
        context = new GeoApiContext.Builder().apiKey(API_KEY).build();
    };

    public DirectionsResult getDirections(String origin, String destination, List<RouteXLLocation> locations) {
        DirectionsApiRequest req = DirectionsApi.newRequest(context)
            .origin(origin)
            .destination(destination)
            .waypoints(locationsToRequest(locations));

        try {
            DirectionsResult result = req.await();
            return result;
        } catch (Exception e) {
            e.printStackTrace();
        }

        return null;
    }

    // Converts the JSON request from RouteXL's format to Google's format
    private String locationsToRequest(List<RouteXLLocation> locations) {
        String result = "";

        for(int i = 0; i < locations.size(); i++) {
            result += locations.get(i).getLat() + "," + locations.get(i).getLng() + ((i < locations.size() - 1) ? "|" : "");
        }

        return result;
    }
}

```

Figure 41: Google Directions request

The weather\_stations is populated as new cabin areas are added. A scheduler then fetches the closest weather station within 50 km radius, checks if it exists in our weather station table. If it does not exist, it is added to the table. Then it is linked to a cabin area. A request is sent to the Frost API for each weather station registered in the weather\_stations table. This request is sent every 24 hours to retrieve new data of snow thickness. It then populates the snow\_statistics table with the snow thickness in centimetres and linked to a weather station which took the reading. The purpose of this is to update every cabin's status in which a significant snow fall has occurred, then notify ploughmen and cabin owners of useful information.

```

@Service
public class WeatherData {

    private final double MAX_DISTANSE_TO_SENSOR = 50.00;

    // function to get nearest sensor of a location
    public WeatherStation getNearestSensor(double longitude, double latitude) throws ClientProtocolException, IOException {}

    public String getSurfaceSnowThickness(String weatherStationId) throws ClientProtocolException, IOException {}
}

```

Figure 42: Frost request

The WeatherData class handles all calls to the Frost API.

#### 4.4.2.3 Data Transfer Object

As entities became large and many, using @JsonIgnore became difficult. The annotation @JsonIgnore can be set on any field of a class. When translating the POJO to JSON, any field with this annotation is ignored. An easier approach was to create data transfer objects resulting in easier readability of code. A DTO is often enough as there are many libraries providing easy to implement, object mapping. Skimming through the documentation for different libraries, it seemed none or few of them supported mapping

of linked lists with the control we sought for. Using these object mappers would result in infinite recursion if we included references to other objects. Thus, we created adapter classes for all objects, which are tied together in the JpaDtoMapper class. The class checks class type and delegates calls to the adapters. Such as the CabinAdapter in figure 44 below.

```

@Service
public class CabinAdapter {

    /**
     * {@link com.snpllog.backend.adapter.MyUserAdapter} for the {@link com.snpllog.backend.entity.MyUser} class.
     */
    @Autowired
    private MyUserAdapter myUserAdapter;

    /**
     * {@link com.snpllog.backend.adapter.CabinAreaAdapter} for the {@link com.snpllog.backend.entity.CabinArea} class.
     */
    @Autowired
    private CabinAreaAdapter cabinAreaAdapter;

    /**
     * {@link com.snpllog.backend.repository.CabinRepository} for the {@link com.snpllog.backend.entity.Cabin} class.
     */
    @Autowired
    private CabinRepository cabinRepository;

    /**
     * Creates a Data Transfer Object ready to send to an end user
     *
     * @param cabin, to be transformed into DTO Object
     * @return CabinDTO {@link com.snpllog.backend.dto.CabinDTO}
     */
    public CabinDTO toDTO(Cabin cabin) {}

    /**
     * This function is needed to stop the infinite recursion in the JSON message sent to the end user
     *
     * @param cabin, to be transformed into DTO Object
     * @return CabinDTO {@link com.snpllog.backend.dto.CabinDTO}
     */
    public CabinDTO toDTOFinal(Cabin cabin) {}

    /**
     * Function to transform a Data Transfer Object to an Entity.
     * Enabling the application to use userRepository calls (i.e. save, update, delete, etc..)
     *
     * @param cabinDTO, to be transformed into entity
     * @return Cabin {@link com.snpllog.backend.entity.Cabin}
     */
    public Cabin fromDTO(CabinDTO cabinDTO) {}
}

```

Figure 43: Cabin adapter

## 4.5 Front-end

### 4.5.1 Architecture

When talking with Avento, our customer it came clear that the application would be best suited for mobile for this project.

### 4.5.2 Programming language

When choosing what programming language to write our graphic user interface in, we decided with the product owner to make a mobile application. A web-based interface was omitted due to time limitations. We wanted to have the possibility to create an application for Android first and IOS if possible. The choices we had were Flutter, React native and Ionic.

React Native seem to be the best for the moment on raw power and possibilities, with modules created by the community they support cross platform in IOS / Android. React native heavily rely on third party libraries while Flutter is the most complete package and don't rely as much on third party libraries.

Flutter seemed to be the best for us, who have limited JavaScript experience to take full advantage of React Native. Development in React Native is very time demanding compared to flutter where we can use default choices to get something up and running early.

The front-end code in Flutter has been written with low coupling and high cohesion in mind. This enabled us to reuse code between the different views and enabled us to see changes made easier to understand.

### 4.5.3 Front-end Code

The front-end application is written in the Dart language using the Flutter SDK and its widgets as the basis for the mobile application. Flutter projects have a file called "pubspec.yaml" which works as a configuration file. In here the Flutter version, project name and dependencies which helps to synchronize versions and used dependencies between developers.

```
version: 1.12.13+hotfix.8

environment:
  sdk: ">=2.2.2 <3.0.0"

dependencies:
  provider: 3.2.0
  http: ^0.12.0+4
  ansicolor: ^1.0.2
  google_maps_flutter: ^0.5.25+3
  geolocator: ^5.1.4+2
  lombok: ^0.0.5
  datetime_picker_formfield: ^1.0.0
  intl: ^0.16.0
  fluttertoast: ^4.0.0
  flutter_polyline_points: ^0.1.0
```

Figure 44: Flutter pubspec file snippet

#### 4.5.3.1 Private fields and functions

To indicate that a field or function within a class in dart should be private, an underscore is used in front of the field or function name.

```
MyUserDTO _data;
```

Figure 45: Flutter private field

```
void _sendLoginRequest() {
```

Figure 46: Flutter private method

Private fields and methods ensure that other outside classes cannot use these fields or functions. This helps reduce complexity and coupling within the project as other class cannot access these fields and methods it normally has no reason to access. This has been used in most main widgets as other widgets should not be able to access these fields or functions.

### 4.5.3.2 Stateful widgets

Stateful widgets in Flutter allows the widget to change over time and update itself when a change has been made. Stateful widgets consist of two classes: a widget which inherits from the "StatefulWidget" class and the state, inheriting from the "State" class, both found in the Flutter SDK.

```
class ProfilePage extends StatefulWidget {
  static const routeName = "/profile";

  _ProfilePageState createState() => _ProfilePageState();
}

class _ProfilePageState extends State<ProfilePage> {

  //Profile data object
  MyUserDTO _data;

  MyUserService _myUserService = new MyUserService();
  PLowerRequestService _plowerRequestService = new PLowerRequestService();

  void _sendPlowerRequest() {...

  void _fetchMyUser() {...

  //Initialization method
  @override
  void initState() {...

  //Widget build
  @override
  Widget build(BuildContext context) {...
}
```

Figure 47: Flutter stateful widget snippet

The above picture shows an example of a stateful widget. The "createState" method should point to the state which the widget should have. All widgets also need to overwrite the "build" method and return a Widget, which is itself. In order to update a stateful widget, the "setState" method should be called inside the widget. This will properly update the state and re-render the widget.

```
setState(() {
  _data = value;
});
```

Figure 48: Flutter setState example

In the front-end application, stateful widgets are used where the user would interact with a widget or where a widget might change based on loaded data to ensure the application is dynamic.

### 4.5.3.3 Stateless widgets

Stateless widgets are static and once built, will not change. Compared to stateful widgets they do not have a required state.

```
class MainDrawer extends StatelessWidget {  
>   bool checkIfUserIsPlower() { ...  
  
>   bool checkIfUserIsAdmin() { ...  
  
>   Widget buildListTile(String title, IconData icon) { ...  
  
   @override  
>   Widget build(BuildContext context) { ...  
}  
|
```

Figure 49: Flutter stateless widget example

Instead everything exists within the class that extends the default "StatelessWidget" class. Same as the stateless widget, there is a "build" method which returns the Widget itself. Stateless widgets have been used for elements in the UI which does not change after creation.

#### 4.5.3.4 build()

The build method is present in every widget and builds the actual structure of the widget itself. Within the build method the child widgets which makes up the parent widget are placed.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text("Ny Hytte"),
    ), // AppBar
    body: SingleChildScrollView(
      child: Column(
        children: <Widget>[
          Text(
            _createCabinErrorMessage,
            style: Theme.of(context)
              .textTheme
              .subtitle
              .copyWith(color: Theme.of(context).errorColor),
          ), // Text
          InputForm(
            textInputType: TextInputType.text,
            inputController: _cabinNameController,
            label: "Hyttenavn",
          ), // InputForm
        ],
      ),
    ),
  );
}
```

Figure 50: Flutter build method

Since this is the actual structure of the widget the build methods can become quite large if the parent widget contains a lot of child widgets. This is especially present in form widgets which require a lot of input from the user and hits to what each input field should contain.

#### 4.5.3.5 Application navigation

In order to navigate between different widgets within the application, Flutter has a solution which is to navigate using named routes.

Widgets which may occupy the main screen have a route name associated with them

```
static const routeName = "/profile";
```

Figure 51: Route name for widget

Which allows us to use the "Navigator" to swap main widgets using these routes. There are two main ways of navigation we have used in our application: "pushNamed" and "pushReplacementNamed".

"pushNamed" adds a widget on top of the previous one.

```
Navigator.of(context).pushNamed(ProfilePage.routeName);
```

Figure 52: Example of pushNamed navigation

Which gives the widget a backwards arrow on the navigation bar, indication to go back to exit.

```
Navigator.of(context).pop();
```

Figure 53: "pop" of the current widget



Since this type of navigation works as a stack, "pop" can also be used to exit out of this type of widget.

"pushNamed" has mainly been used in widgets which are not considered main views of the application and interactions which are meant to put the user back at the previous widget when done.

```
Navigator.of(context).pushReplacementNamed(ProfilePage.routeName)
```

Figure 54: Example of pushReplacementNamed navigation

"pushReplacementNamed" is mainly used in the nav bar and is mainly used to replace the screen with one of the main widgets as this completely replaces any previous widget.

#### 4.5.3.6 API communication

In order to communicate with the API, the front-end application uses a set of "service" classes which is responsible for creating and receiving http request from the API server. These services classes have been made different depending on which widgets uses them and which part of the API it uses.

```
Future<bool> createMyUser(MyUserDTO myUserDTO) async {
  Map<String, String> headers = {
    HttpHeaders.acceptHeader: "application/json",
    HttpHeaders.contentTypeHeader: "application/json"
  };
  final body = jsonEncode(myUserDTO.toJson());
  final response = await http.post(globals.baseUrl + "/user/create",
    headers: headers, body: body);
  if (response.statusCode == 200) {
    print("sucessful");
    return true;
  } else {
    print("not able to create");
    return false;
  }
}
```

Figure 55: Method from a service class

The methods inside the service classes send a request to the API and depending on the answer it will return true or false which can further handle the result from the API call.

### 4.5.4 Interaction flow

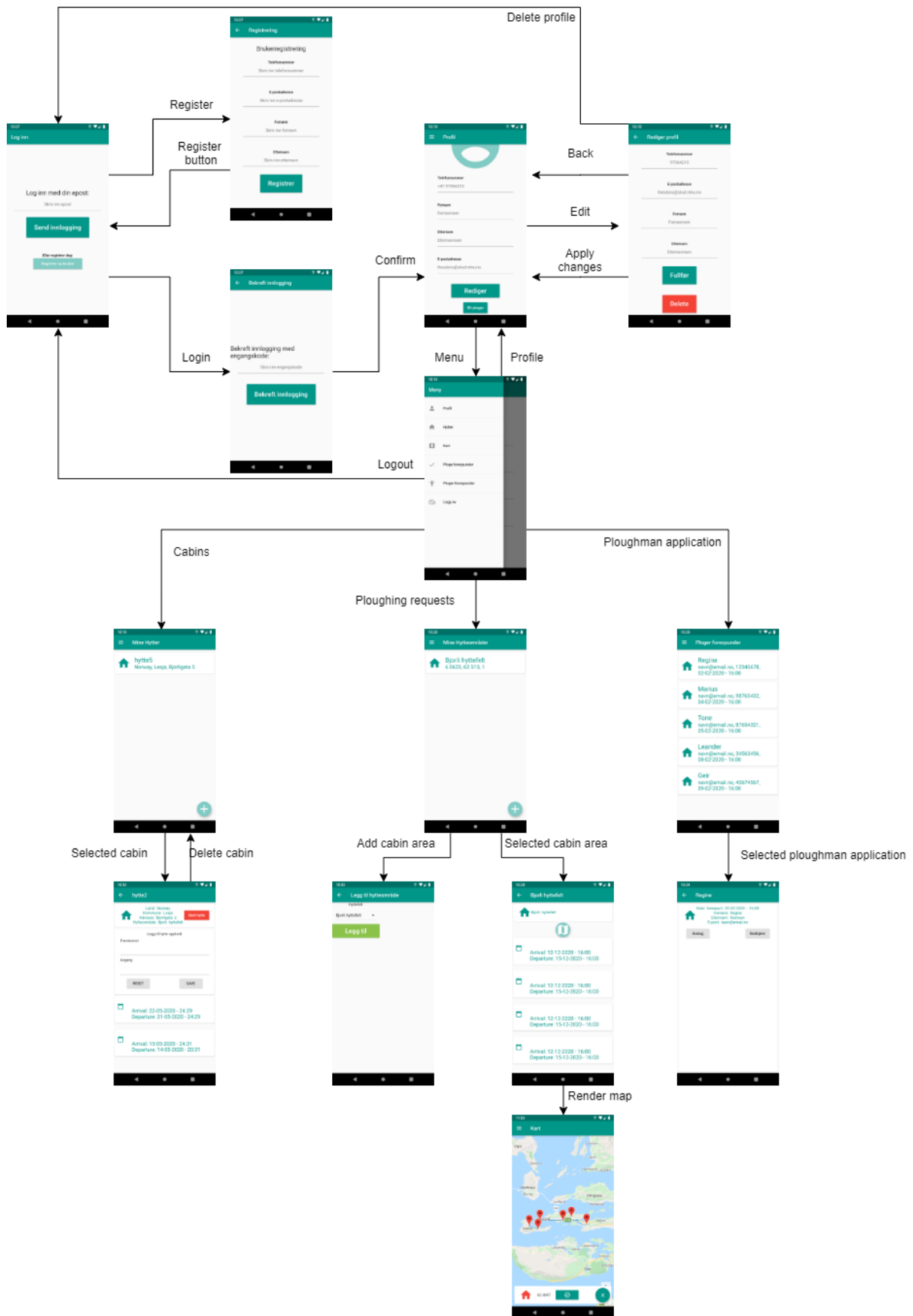


Figure 56: Interaction Flow chart

The model above displays the interaction flow of any user. From the Login view you firstly authenticate or create a new user. After retrieving a one-time password, you receive a token. Keeping a user logged in based on his roles. The first view a user is met with is the profile view. From the user view a user can logout or navigate to edit the profile. From the edit profile view a user can submit changes, go back or delete the user. From here the user has a hamburger menu containing navigation to every other view, this hamburger menu is present if a user is logged in. If a user is in any form view, they have a back button at the top-left corner, navigating them to the previous view.

#### **4.5.5 Graphical user interface**

The user interface was created with simplicity in mind and consistency of colour choices and buttons. All the colours should mean something and be relatable to other parts of life, for instance a green coloured house should be a good thing, as going or have been done almost like traffic lights where the green is good to go, in contrast red means stop and have not been, warning something must be done.

The choices were made so users should be able to recognize the structure and easily navigate and use the application.

We created the wireframes from the use case diagram we had in our pre project plan. The wireframes and GUI were made with the eight golden rules of interface design principles in mind.

Wireframes are very good to show how we saw the application take form and to do changes on wireframes take less time than on paper or by code, providing very good feedback early on.

##### **4.5.5.1 Design philosophy**

When designing the UI there were a few philosophies we kept in mind and tried to follow. For starters we tried to keep UI elements such as text and buttons decently large as it is an application which might be used by older people with poorer eye sight, as well as being used by ploughmen who might use the application in their vehicle and have their phone mounted a bit away from their face.

Secondly, the primary audience for the application were Norwegian cabin owners, so the text and messages in the interface had to be in Norwegian.

Don Norman's principles were also some of the philosophies we tried to keep in mind while creating the interface. Keeping visibility by limiting the amount of elements on the screen at the same time, giving feedback to the user such as error messages when invalid input is given, keeping input fields and their label close to each other, aiming for consistent elements, menus and theme and giving the user hints on how to use the application by hint text or presenting them something similar to what they might have seen before, are some examples on how we strived to apply Don Norman's principles to the interface design.

The image shows a form with four input fields. The first field is labeled 'Hyttenavn' and contains the text 'Hytta :)'. The second field is labeled 'Kommunennummer' and contains the number '7560'. The third field is labeled 'Gardsnummer' and is empty. The fourth field is labeled 'Bruksnummer' and is empty. Below the third and fourth fields, the text 'Bruksnummer mangler' is displayed in red, indicating a missing value. Each field is separated by a horizontal line.

*Figure 57: Example of error reporting with the ability to recover*

Furthermore, the eight golden rules of interface design were also something we tried to follow. Preventing error from happening by pre-selecting which keyboard type the user could use and doing input checks, as well as keeping the information in input fields so the user wouldn't have to re-type everything in the event an error happened, are ways we applied the golden rules of interface design to the application.

### 4.5.5.2 Views

The views in the application is based on the wireframes made in the planning phase of the project.

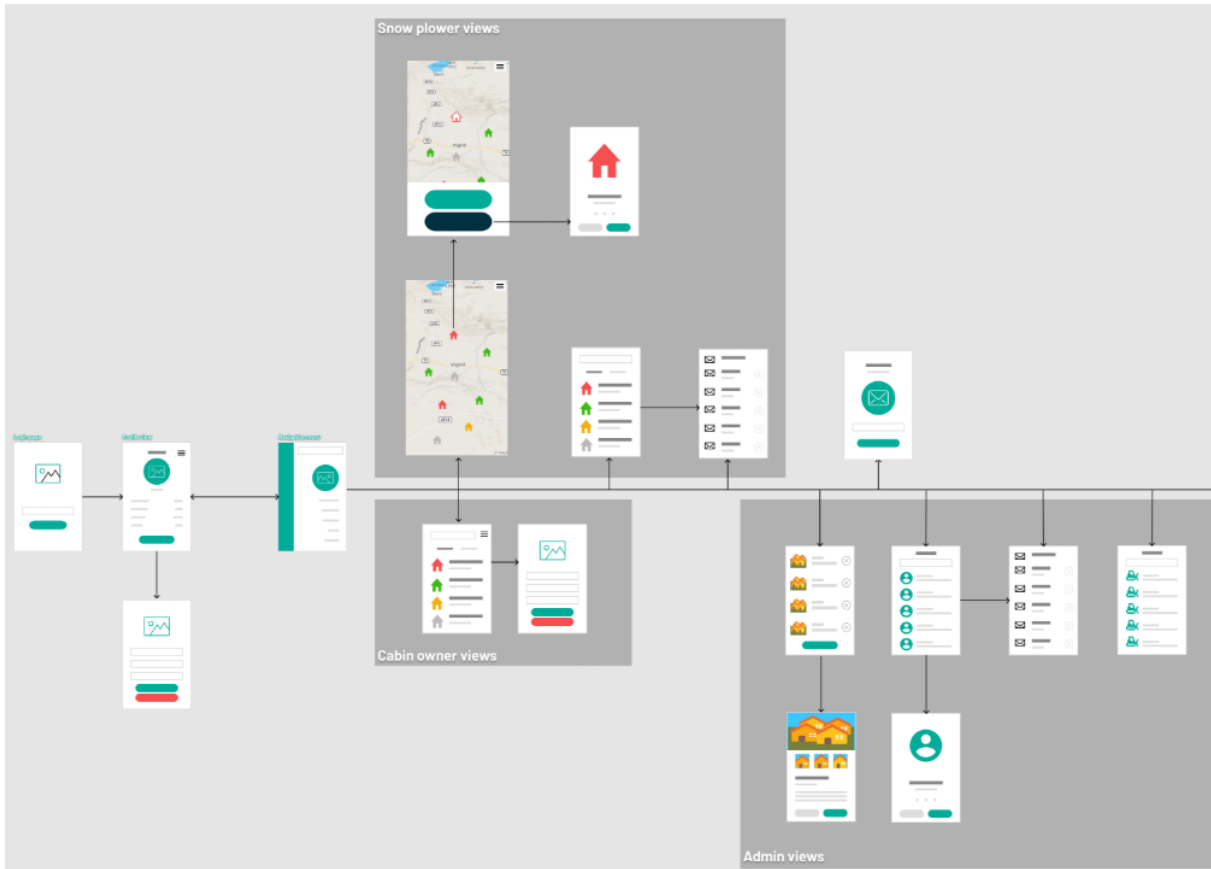
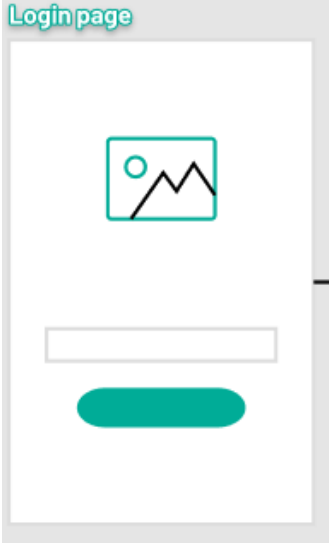
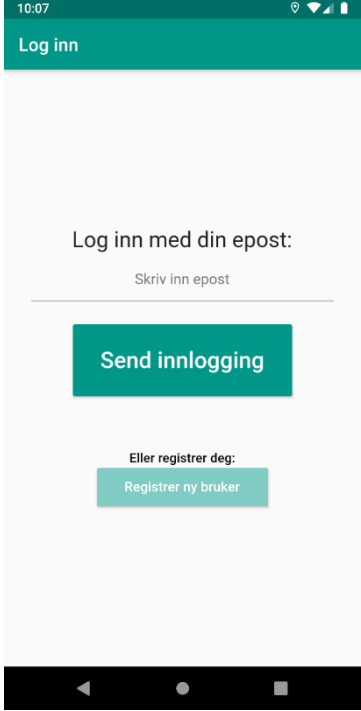

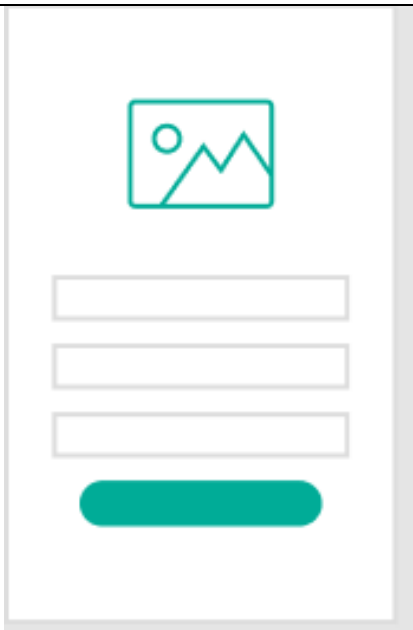

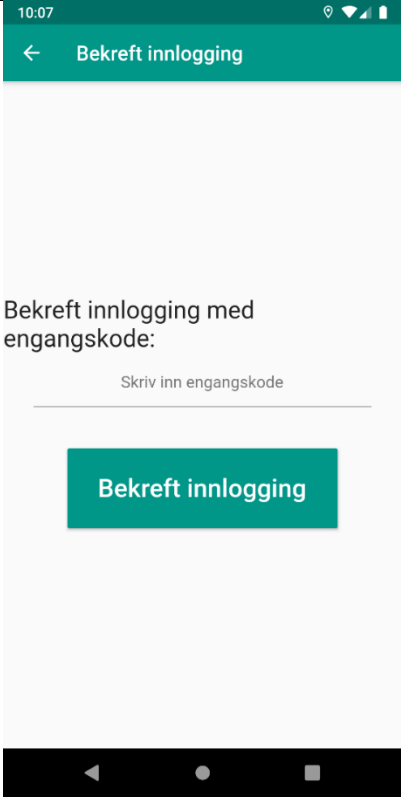


Figure 58: Pre-project diagram of application

<p style="text-align: center;"><b>User views</b></p> <p style="text-align: center;">These views are for any type of user. A user not logged in will only see the login page. A user with no roles will only see the user views until they register themselves and get approved for other roles.</p>		
View	Wireframe	Flutter
<p style="text-align: center;">Login</p>	 <p style="text-align: center;"><i>Figure 59: Login diagram</i></p>	 <p style="text-align: center;"><i>Figure 60: Login view</i></p>
<p>For a user to access the application the user needs to login. The application needs to process owners of cabins, ploughmen and admin(s).</p> <p>This is the only view available to a user not logged in.</p>		

Message	 <p>Figure 61: Message diagram</p>	N/A
Message view for a user logged in to contact Admin.		
Registration	 <p>Figure 62: Register diagram</p>	 <p>Figure 63: Register view</p>
Registration view for a user.		

<p>One-time password</p>		 <p><i>Figure 64: OTP view</i></p> <p><b>S</b> snplog2020@gmail.com ty. 12.05.2020 22:08 Theodor Giskejerde Urtegård ✓</p> <p>Your random 6 digit code is : 761246 expires in : 15 minutes</p>
<p>One-time password was used for authentication of users. Containing claims like username, expiration time and a role table.</p>		



Navigation menu



Figure 65: Navigation diagram

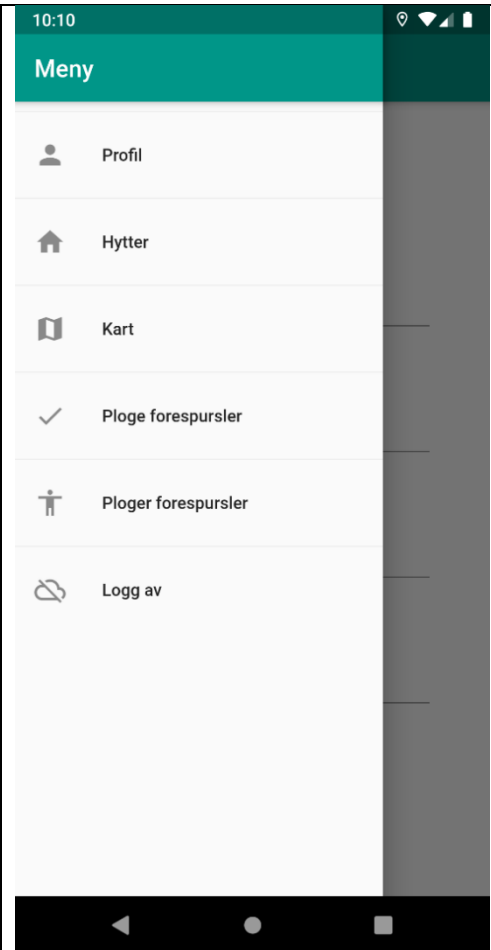


Figure 66: Navigation view

This is the main navigation menu to how to navigate the application between the different views. Role checks are enabled so a user will see less here than an admin.

Profile



Figure 67: Profile diagram

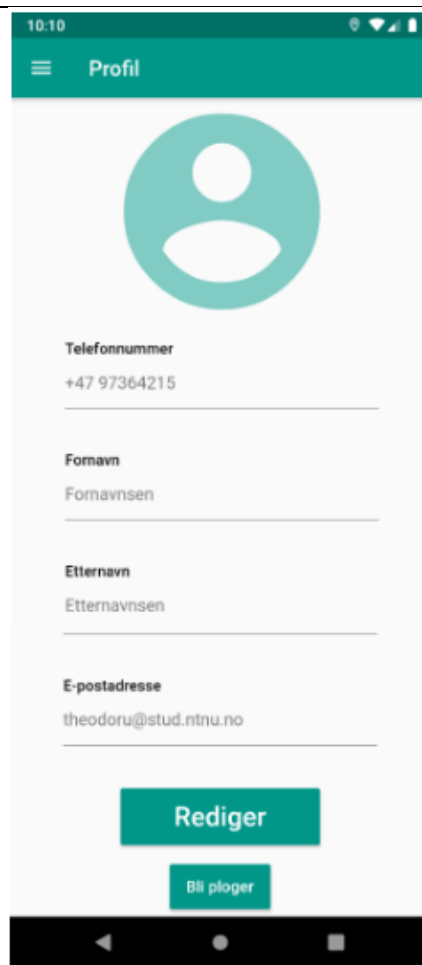
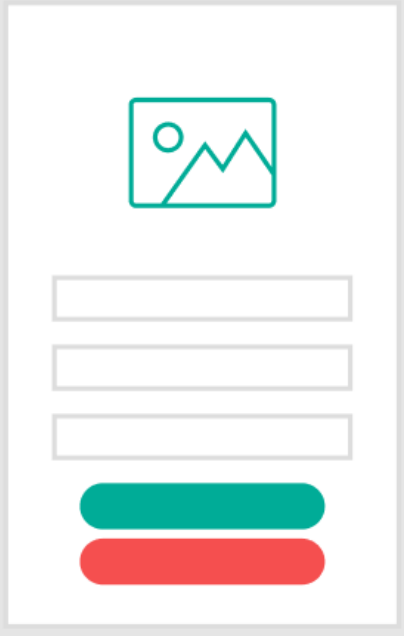
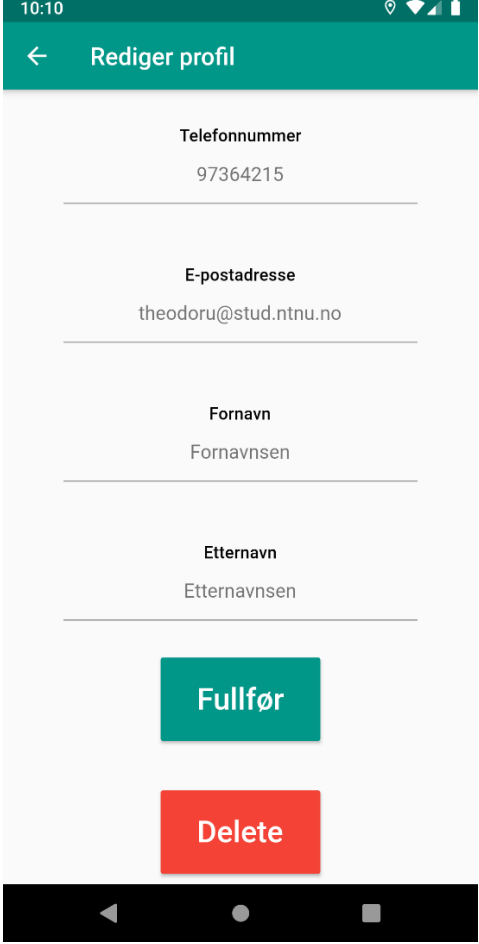


Figure 68: Profile view

As a user, you can see your own user profile view.

<p>Edit profile</p>	 <p><i>Figure 69: Edit profile diagram</i></p>	 <p><i>Figure 70: Edit profile view</i></p>
---------------------	---	--

Every user can edit their profile.

### Cabin Owner Views

These views are for a user who has registered themselves as cabin owner.

<p><b>View</b></p>	<p><b>Wireframe</b></p>	<p><b>Flutter</b></p>
--------------------	-------------------------	-----------------------

Cabins

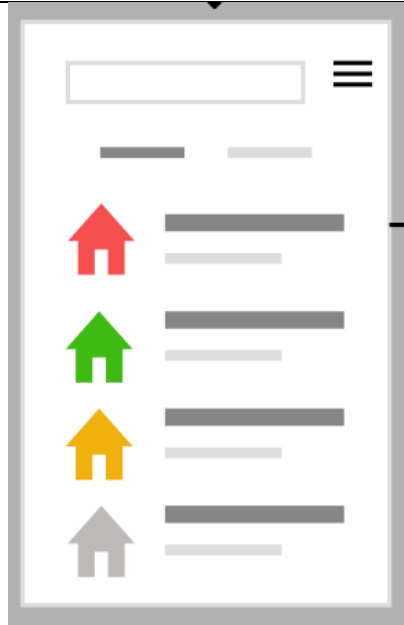


Figure 71: Cabins diagram

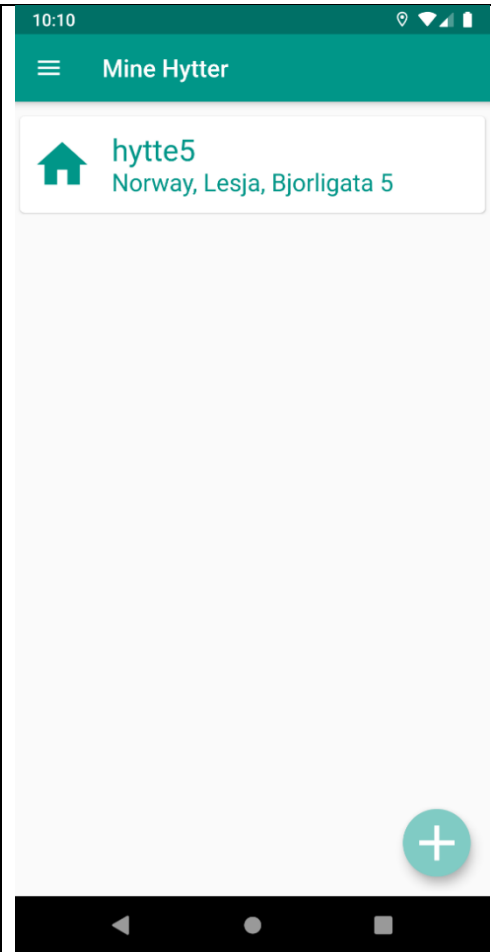
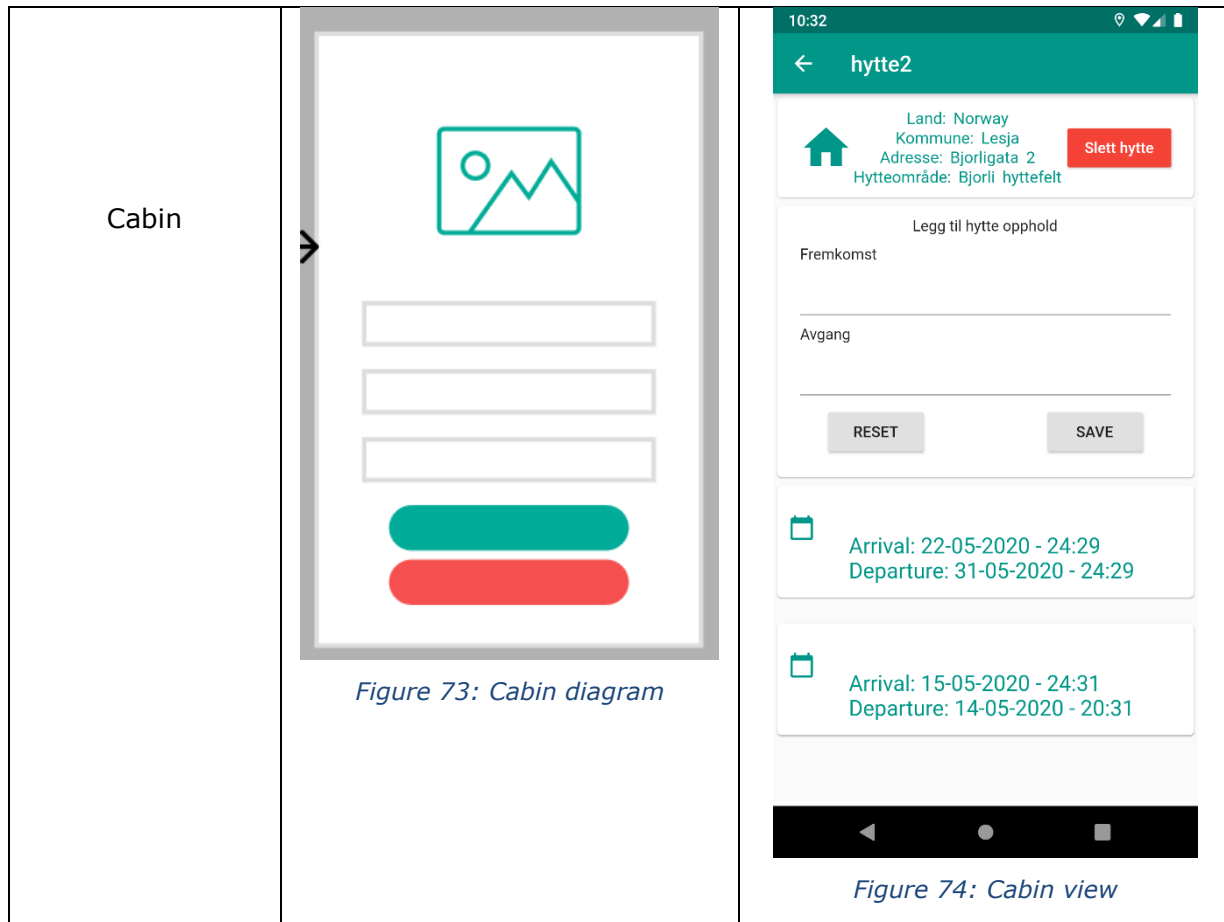


Figure 72: Cabins view

This view will show all the cabins for a cabin owner.



This view is for a user who is viewing or editing their targeted cabin.

### Snow ploughmen views

These views are for a user who has been approved by an Administrator to be a ploughman.

Ploughmen's  
cabin areas



Figure 75: Ploughmen cabin area diagram

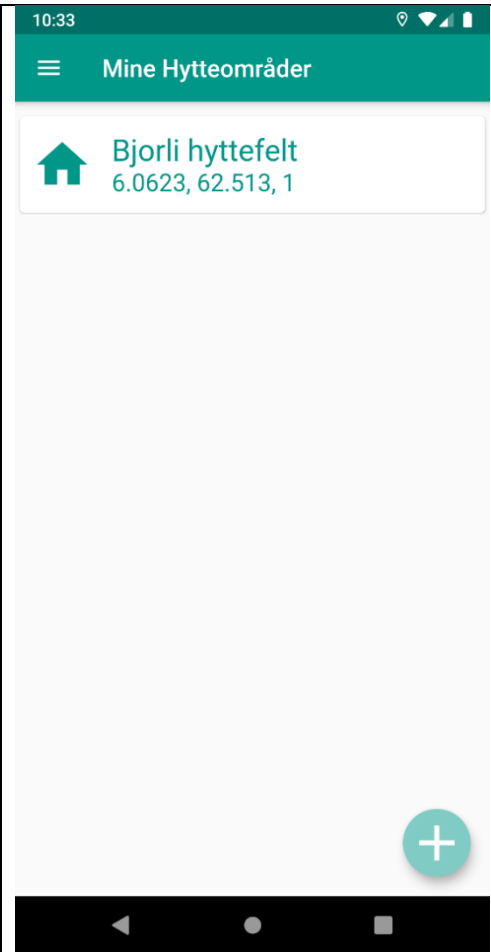


Figure 76: Ploughmen cabin area view

This view will show which cabin areas the ploughmen has been appointed to.

Add cabin area to  
ploughmen



Figure 77: Assign cabin area  
diagram

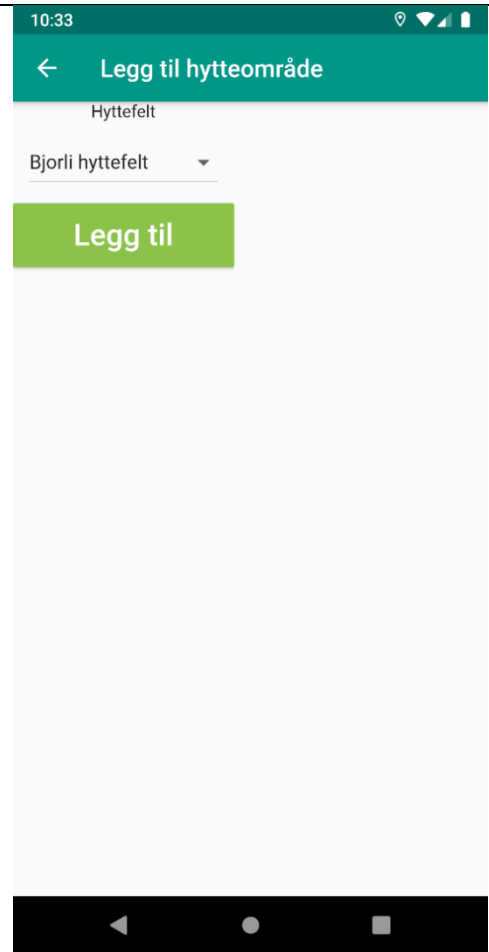


Figure 78: Assign cabin area view

The ploughmen can assign themselves to a cabin area.

Ploughing requests



Figure 79: Ploughing requests diagram

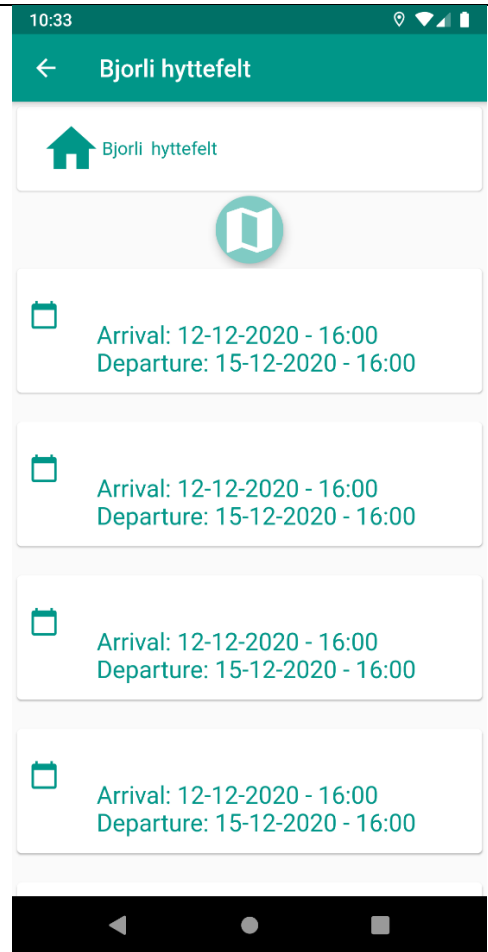

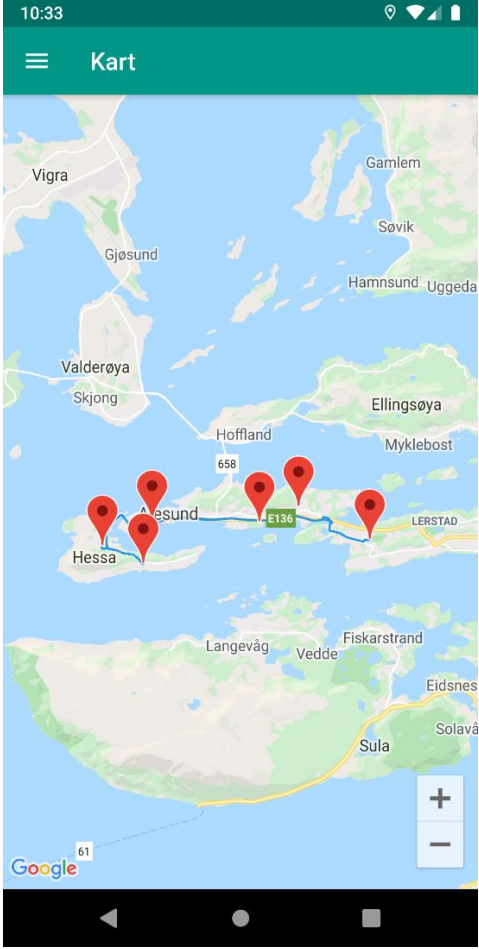



Figure 80: Ploughing requests view

This is the view of current requests made from cabin owners in the cabin area for a ploughman.

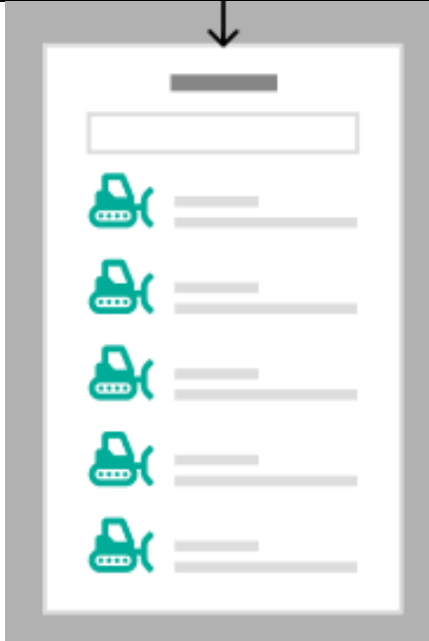
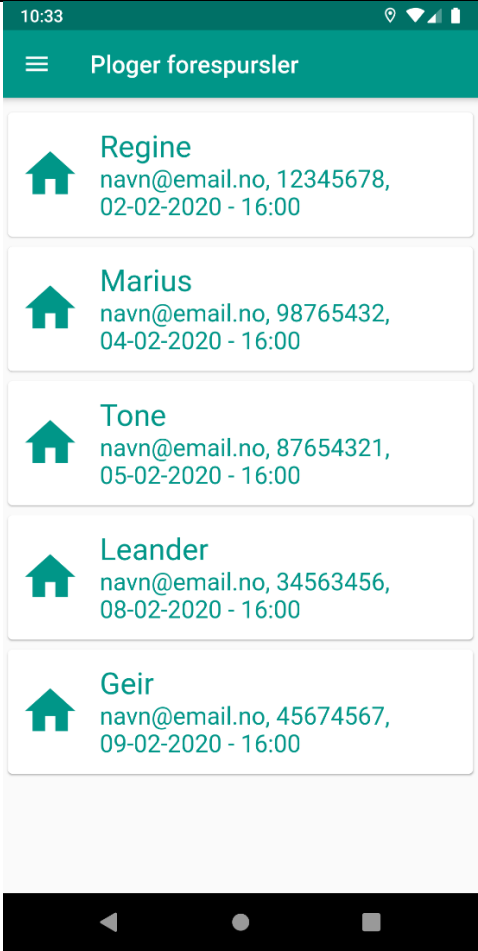


<p>Map</p>	 <p>Figure 81: Map diagram</p>	 <p>Figure 82: Map view</p>
------------	---	--

This is the overview map the ploughman will see of their cabin area and the cabins in that area that has requested ploughing. It will also map a route for ploughman to take.

<p>Message</p>	 <p>Figure 83: Inbox diagram</p>	<p>N/A</p>
----------------	---	------------

This is the view for ploughman to see his messages between him and the users. We did not have time to finish creating this in flutter.

<b>Admin views</b>		
These are the views a user who has been appointed the admin role, can see.		
<b>View</b>	<b>Wireframe</b>	<b>Flutter</b>
Admin ploughman overview	 <p style="text-align: center;"><i>Figure 84: Ploughmen application diagram</i></p>	 <p style="text-align: center;"><i>Figure 85: Ploughmen application view</i></p>
As an admin he can get the overview of all ploughmen in the application and list by requests.		

Admin  
ploughman  
requests



Figure 86: Ploughman application diagram

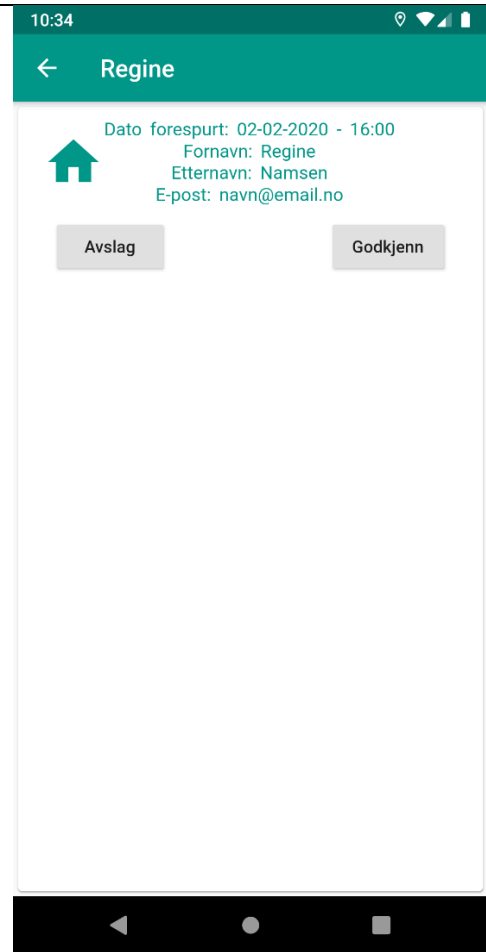


Figure 87: Ploughman application view

As an admin he can deny or approve requests made by a user to be appointed ploughman.

Admin cabin area  
overview

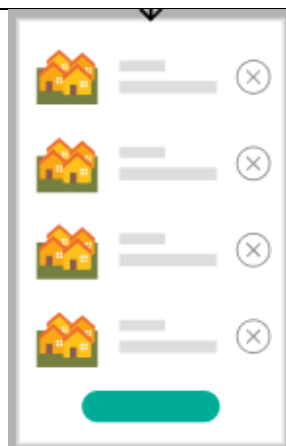
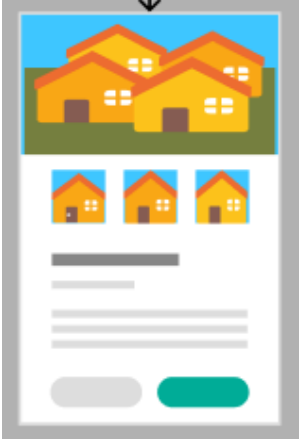
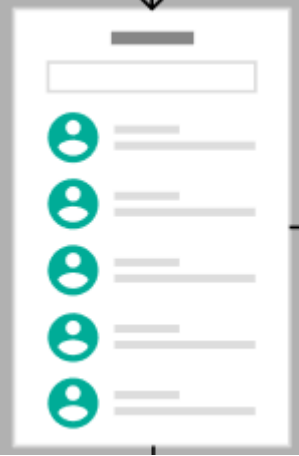




Figure 88: Admin's cabin areas diagram

N/A

The admin can see an overview of all the cabin areas in the application.

<p>Admin cabin area</p>	 <p><i>Figure 89: Admin's cabin area diagram</i></p>	<p>N/A</p>
<p>The admin can see more detailed information about a cabin area of interest.</p>		
<p>Admin overview of user(s)</p>	 <p><i>Figure 90: Admin's users diagram</i></p>	<p>N/A</p>
<p>The admin can see an overview of all the users in the application.</p>		
<p>Admin view of a user</p>	 <p><i>Figure 91: Admin's user diagram</i></p>	<p>N/A</p>
<p>The admin can see a user's profile page in more detail.</p>		

Admin message	 <p><i>Figure 92: Admin's messages diagram</i></p>	N/A
The admin can see all their messages between users in this page.		

## 5 REFLECTION

### 5.1 *Graphic user interface*

We did not have the time to create the views we had planned to create. Corona hit us harder than we first thought, some of us had extra jobs and those jobs required more hours now due to corona. The other negative thing about Corona was that we as a group struggled with working from home, and school was this haven for us where we got out of our homes and to school to work. And if we were at school, we worked on the project fully. But now that we were working from home some of us struggled with just the basic of getting to work on the project and stay focused for some hours a day.

We spent a lot of time reading Flutter documentation, Flutter is relatively new on the market, Flutter 1.17 and only stable version of 2020 was shipped May 6.th, implementing maps to Flutter took a lot of time to do. [114]

### 5.2 *Server infrastructure and configuration.*

We wanted to create a full CI/CD solution with Gitlab to create the back-end Server we needed for this project, but it seemed there were no possible solution to enable Ansible to run playbooks from a runner/docker image in Gitlab and we ended up scrapping the whole automation through pipelines as it took a lot of time, we ended up only with a pipeline to create the server infrastructure.

Two different servers were created by the same Terraform infrastructure code. One of the servers were the one we used as a node to hold the back-end application and the other to push the Ansible configuration files from.

Playbooks ran from the host server proved to be of no problem and instead of spending more time trying to figure out how to be able to run it from the docker/runner images at Gitlab we chose to not spend more time on it.

### 5.3 *Scrum Workflow*

The scrum workflow of sprints lasting 2 weeks began great. Though without any experience as a scrum master and COVID-19 causing lock-down of most public workspaces, the development team used the agile workflow less as the project went on. This resulted in poor follow up on issues to be prioritized and a less strict schedule was held. I.e. integration between front-end and back-end was put off for a substantial amount of time. To combat this, we tried to start meeting in Discord as often as possible. Despite our efforts, motivation faltered. We think the social state of our society made it a difficult task for students and/or employees to motivate themselves during these times. As work either increased or became unsure. Blowing off steam in social events was not possible. Mentioned below is a quote from the World Health Organization. These conditions do not encourage a good work environment.

“In public mental health terms, the main psychological impact to date is elevated rates of stress or anxiety. But as new measures and impacts are introduced – especially quarantine and its effects on many people’s usual activities, routines or livelihoods – levels of loneliness, depression, harmful alcohol and drug use, and self-harm or suicidal behaviour are also expected to rise.” – World Health Organization [115]

Another issue we faced was large user stories not being broken up into appropriate sizes. Resulting in large unspecific tasks leading to a developer losing focus on the user story. This should have been solved by dividing up the user stories earlier in the project, setting as clear as possible goals for the sub issues.

All sprint reports and cumulative flow diagram can be found as Appendix 4.

## **5.4 Front-end**

This section below we will reflect upon which functional requirements were completed for each type of user, functional requirements can be found in Appendix 1. A lot of core functionality was prioritized in this project, with quite asynchronous work between front-end and back-end application. This led our system to fall short on a lot of less essential functionality. The project team should have focused more on developing function as needed for back-end and front-end.

### **5.4.1 General User Requirements**

A user should be able to register themselves. One of the first views developed was the login page along with the security on the back-end. It uses email verification as log in, as this is a free service. Preferably we would use phone numbers instead, but there are costs tied to using such a service.

The user should be able to log in and out. This was implemented using authentication tokens, as the back-end is stateless and does not keep track of users logged in. When a user logs out, the token is cleared by the front-end application.

The user should be able to get an overview of their information. This can be done in the profile view, where stored information about a user is rendered. The information contains a first and last name, email and phone number.

The user should be able to edit their own user information. In the profile view there is an edit button which leads a user to the profile editing page. This allows any user to edit their name, phone number and email. Though if a user edits their email to an email which is not valid. They cannot access their account. There should be implemented a verification to check if the email is correct.

The user should be able to delete his user. There is a button in the edit profile view, where the button press deletes the user. Should be implemented an alert box prompting the user if they are sure.

The user should be able to have multiple roles. This is achieved through the junction table in the back-end. Any user may be as many roles as they are assigned.

### **5.4.2 Cabin Owner Requirements**

The user should be able to register themselves as a cabin owner. This is done by default assigning every user a cabin owner role. The intention was that this role would be assigned after a user adds a cabin. Though this would require a new authentication request, as the authentication token holds roles. Maybe a new authentication token should have been included in the return payload of posting the new cabin.

The user should be able to register a time period, in which the cabin is occupied. This is implemented in the single cabin view. With two time and date inputs, providing any cabin owner to request a ploughed driveway their entire stay.

The user should be able to edit the occupied status. The only change a user can do to a ploughing request is to delete it. Considering the time limit and how easy it is to add a cabin occupation. Implementing a possibility for editing was not prioritized.

The user should be able to add cabin area(s) to their cabin(s). Upon creation of any cabin, a user must select a cabin area from the existing cabin areas to complete the request.

The user should be able to own multiple cabins. This is implemented through the cabins view, in which one accesses a cabin's details and sends ploughing request.

The user should be able to get notified when a ploughing request has been completed. This requirement is not met as no push notifications is implemented.

The user should be able to send individual requests to existing ploughmen in the area of their cabin(s). As any occupied status is set, a request is added to the plowing\_requests table.

The user should be able to remove a cabin from his registered cabins. There is a delete button in every cabin view, which removes the cabin from the database.

The user should be able to contact the administrator. This requirement is not met due to time limitations.

The user should be able to request a new cabin area. This requirement is not met due to time limitations.

### **5.4.3 Ploughmen Requirements**

A user should be able to register as a ploughman. This is implemented through a request system. A request may be sent from a user's profile view. This request must be approved by an admin, then it is added to the user's role.

A user should be able to assign working areas. This is implemented through the cabin areas view in which any registered ploughmen can assign themselves to cabin areas existing in the cabin\_areas table.

The user should be able to see plough requests for cabins. This functionality is implemented through cabin areas instead of cabins. As a ploughman only cares about cabins to be ploughed in his registered areas.

The user should be able to confirm ploughing. This is not implemented fully due to time limitations. The button is present in a view, but not integrated with back-end.



The user should be able to see a map of cabins. This is rendered as a ploughman asks for a ploughing route for a cabin area. This navigates the user to a map screen where pins for each cabin is rendered and polylines is drawn between each cabin.

The user should be able to get a notification of a new ploughing request. This function is not implemented due to time constraints.

The user should be able to contact cabin owners. This function is not implemented due to time constraints.

The user should be able to contact administrator. This function is not implemented due to time constraints.

#### **5.4.4 Admin Requirements**

The user should be able to contact cabin owners and ploughmen. This is not implemented due to time constraints.

The user should be able to edit users. This is not implemented due to time constraints.

The user should be able to search for users. This is not implemented due to time constraints.

The user should be able to approve registration of ploughmen. This is implemented through a ploughmen requests view. Though this should also notify a user if they are approved or not, which is not implemented.

The user should be able to add cabin areas. This is not implemented due to time constraints.

#### **5.4.5 Non-functional Requirements**

Temporarily store cookies for authorization. This is implemented through JWT's stored on the front-end application and sent with every request requiring authorization.

Responsive UI, the front-end application has some user feedback, but more could be implemented. Such as loading screens for time consuming requests such as a ploughman's route. Which requires a request to two external services, resulting in an average request time of over 1 second. There could also be more useful user feedback on failed requests, giving them more details of why it failed.

User friendly UI is mostly implemented through the hamburger menu. Leaving most views three layers deep at most, provides an easy application to navigate through. Most buttons are also self-explanatory, leaving not much for the users to interpret.

Secure user data is mostly achieved through the security filter on the back-end, security that comes with the Spring framework and input checks on the front-end application. There also is no password required to log in, hindering any users from being accessed through a data leak.

The app should be available on most android devices. Flutter supports Android Jellybean 4.1.x or newer. As most android phones are running Android Pie 9.0.0, this requirement is met.

## 5.5 Back-end

The development of a back-end application went well as project members were familiar with the concepts of the Spring framework. Though the database ended up quite big, offering its difficulties. Approaching the service layered design pattern with an okay understanding came with difficulties when expanding the application. Then the infinite recursion problem happened, which led to the discovery of data transfer objects. The section below will be dedicated to reflecting upon those problems and their solutions.

### 5.5.1 Database

Through this section we will show you the path from first enhanced entity-relationship (EER) diagram to the last.

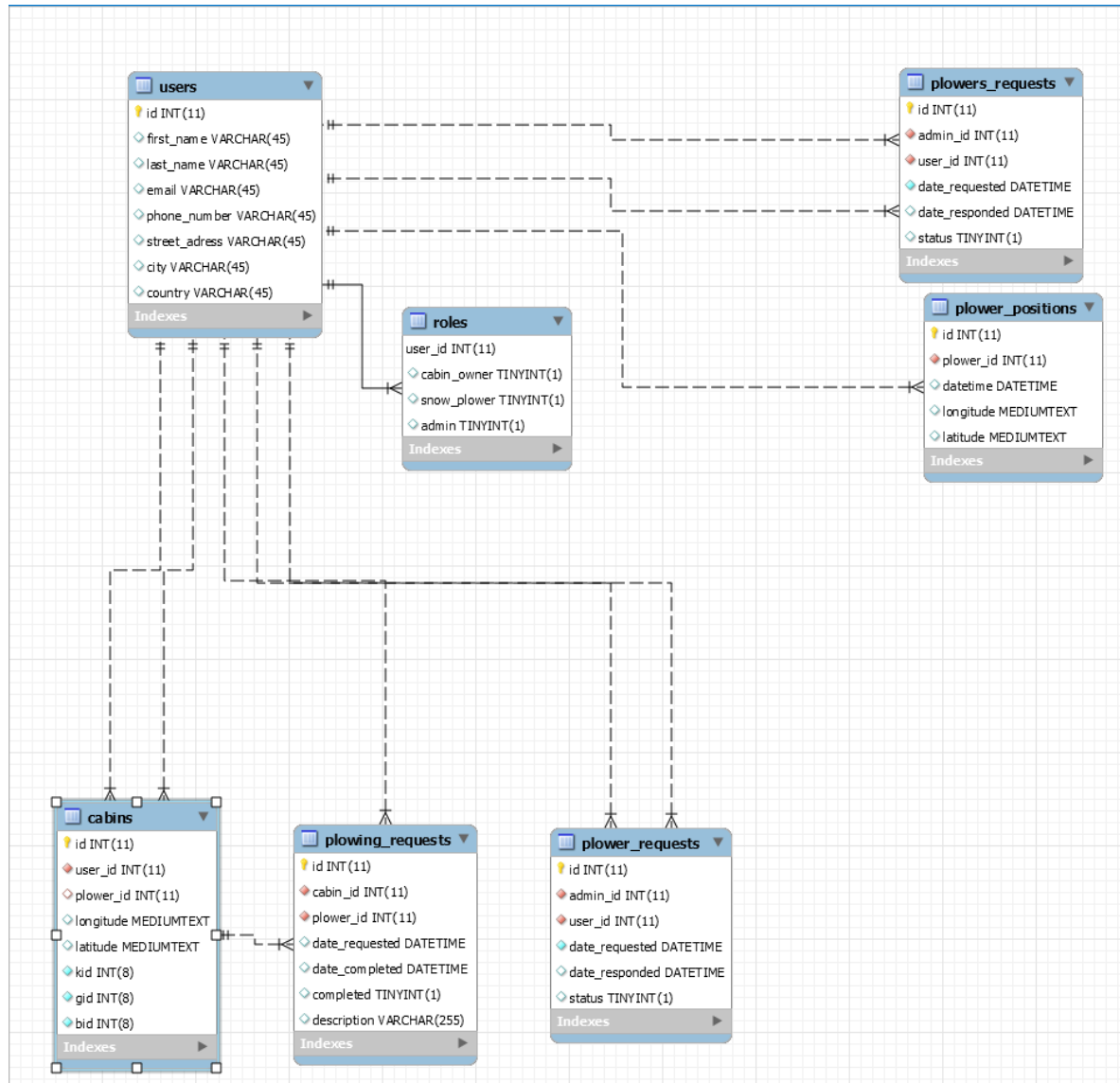


Figure 93: First EER diagram

The first EER diagram created in this project.

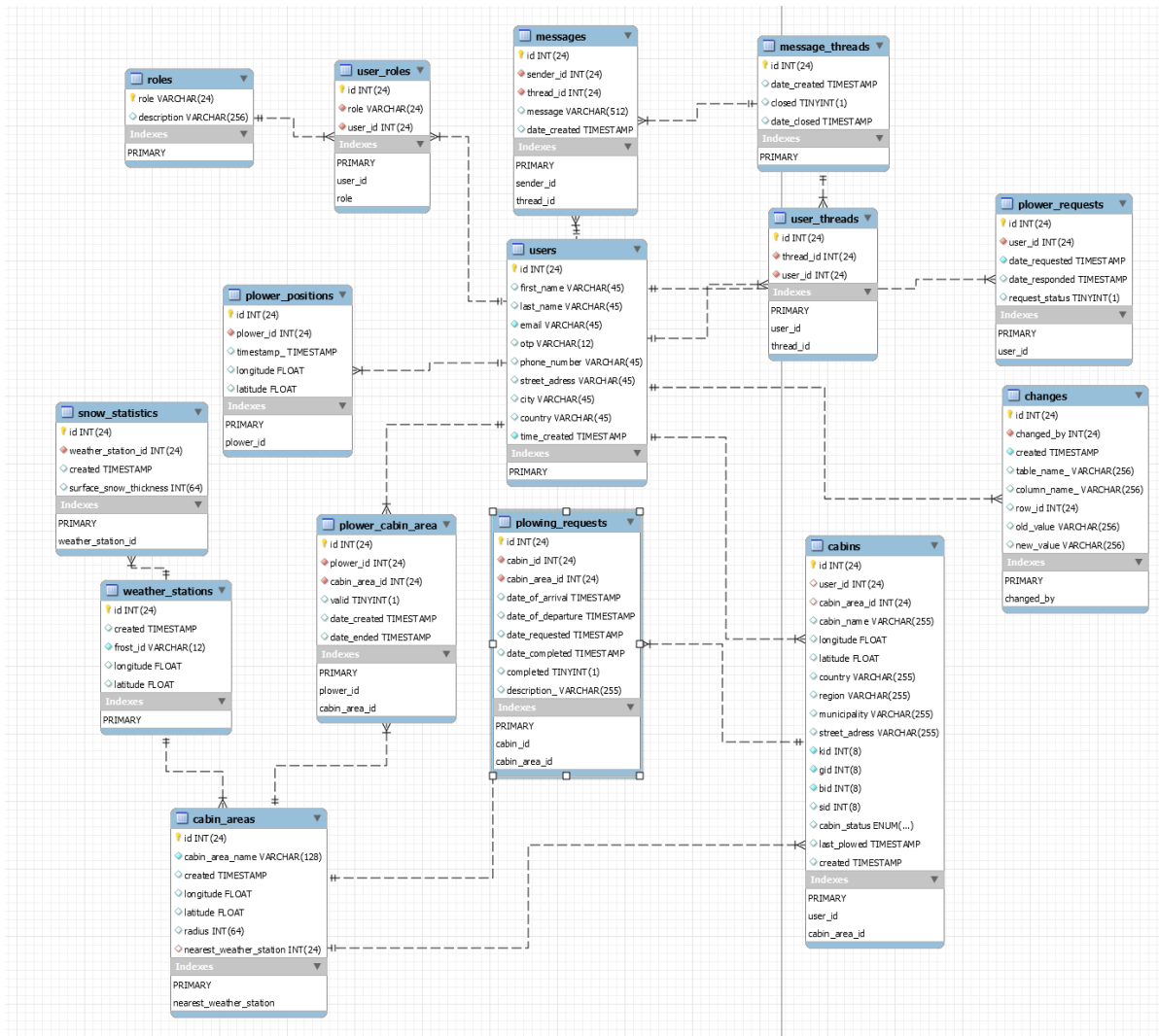


Figure 94: Last EER diagram

The last EER diagram before hand-off to the product owner. The discrepancy between these diagrams displays the growth of this project and the difficulty of defining a complete database in the project planning phase. Defining the database might be easier with more experience within full-stack development.

### 5.5.1.1 Primary Keys

Deciding which column should be the primary key was quite easy. Most tables use integers which cannot be null, and auto-increments. This provides efficient indexing and uses a fact-less key. A fact-less key is a key that holds no information about the tuple's non-key columns. Only exception is the Role table, which uses the roles name as primary key. Though since the table is expected to be small, this should pose little to no problem with efficiency.

### 5.5.1.2 Composite Keys

The relational database uses a few junction tables. All junction tables created in this project uses integer simple keys to index. This seems a bit redundant as composite keys could be made for each table by using the foreign key columns as a primary key. Though this was avoided to save time during the project.

### 5.5.1.3 Junction Table

The Roles table was at first a one-to-one relationship with the users table as you can see in figure 94. This turned out to be a bad design and the product owner asked if we could make it into a junction table. This would increase scalability, as adding another role would be as easy as inserting a new tuple into the Roles table. The final solution can be seen in figure 95.

### 5.5.1.4 One-To-One Relationship

Through the development some tables grew quite large, such as cabins and users. A one-to-one relationship might have been a better design for cabins' and users' information. Users table had some optional data, which would be better placed in its own table. Leaving only the essential data for business logic. Cabins does not contain any optional data, but it could be split into data useful for business logic and visual data. By visual data I mean textual information about a cabin's identity. This could increase readability and efficiency of the back-end application.

### 5.5.1.5 Normalization

First normal form, Atomicity for each attribute was implemented quite early as it can be seen in figure 94, I.e. first\_name and last\_name attributes. Though some values such as street address could be divided even more. Though this would most likely create more string manipulation on the front-end or back-end. As street address fields usually retrieve both address name and number.

Each non-key attribute is dependent on the primary key and atomicity is present, thus fulfilling the second normal form.

To achieve third normal form, we must remove transitive dependence. Some of the tables include address, country, region and municipality. Each attribute could be derived from the address. For 3NF to apply we would have to make a table for the derived values. For example, putting municipality and region in one table. Where municipality is the foreign key for the table containing information about which region it lies in. Because the region of a municipality can be derived by the municipality. I.e. Ålesund municipality lies in Møre og Romsdal region.

Given more time in development, we would have ensured the database is designed after Boyce-Codd normal form. This would fragment the database into tables with less redundancy and more data integrity.

### 5.5.1.6 Auditing

Auditing table entries provides a more secure database and keeps data integrity. As every saved entry can be traced back to a user and every old value is kept. Developing the auditing business logic rendered quite difficult. Because of data transfer objects obscuring what was edited by the front-end application by translating each DTO to an entity. Thinking it then would be easier to create an auditing API our self, it came with lots of bugs and business logic for the front-end to handle. Due to time limitations the auditing table requires one to add changes to a set of changes on the front-end. Then pass it to the back-end. Providing more work on the front-end application than intended. Next time we would rather use an existing API.

## 5.5.2 Service Layered Design

Approaching this application with a service layered design principle was simple for the CRUD methods. Though as the project expanded, implementing calls to third party API's and scheduling calls on new threads. The placement of this business logic was unclear, and the application structure became a bit messy. Given more time we would have refactored the code and consulted our product owner for package structure tips.

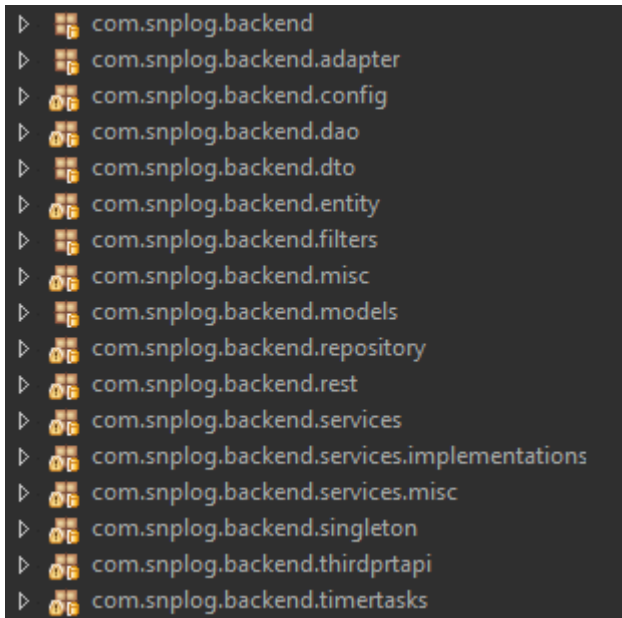


Figure 95: Package structure

## 5.5.3 DTO – Object Mapping

As the DTO solution was considered, the project structure was quite set. Finding solutions to every specific problem in such a large project can be difficult. We found two popular solutions to the infinite recursion problem, data transfer objects and JSON ignore. JSON ignore was the easiest solution, but this omitted linked entities we might want to include in the response body. Because calls to an external server is expensive and slows down the front-end application significantly. Data transfer objects then gave us the possibility to aggregate data, allowing us to rely on few http calls on the front-end application. Deeming DTO to be the best solution we started implementing the business logic, in hindsight this part used too much development time.

After creating the DTO, we needed a function to map entities to their respective DTO. There was some APIs able to handle this mapping, but they did not have the functionality to aggregate data as custom as we needed. Creating another time-sink for our development time, as a custom object mapper was developed.

Updating tuples and auditing then posed a problem, as the conversion from DTO to entity needed to keep the new values. Quickly solved by setting all the editable values in the adapter classes. Though auditing the updates then posed a problem as our back-end application did not know which attributes were updated. The second time-sink for our development process. This was solved adding a set to each DTO which kept track of changes. A quick solution, which created a lot of extra code to be written each time a front-end update entity function was implemented. We probably should have used an existing API for auditing functionality considering our development time was quite short.

### **5.5.4 Business logic**

Using four different third-party APIs we discovered that the format of their requests were quite different. All our requests contained a payload with information needing to be encoded in JSON after the API's documentation. The responses did not always carry enough information back, forcing us to make helper functions to sort and convert responses to entities and DTOs.

When creating a cabin, a scheduler runs on a new thread fetching a cabins location data. This should have run on the same thread, as the response should give an end-user feedback if the address existed.

### **5.5.5 HTTP's GET**

Developing queries for certain result lists using pagination and sorting left us wanting a dynamic as possible GET function. A model for pagination and sort requests was made and included as a body of the GET method. Developing the front-end we discovered that the GET method does not allow any body in their request. Reviewing the documentation of HTTP's standardization, it stated that GET methods should not contain a body. Every GET function used by the front-end got refactored on the back-end to fit this standardization. Though due to time limitations, the refactoring of the unused functions was omitted.

### **5.5.6 Proxy Server**

During development and planning, the project group did not know that a proxy server is required for all external API calls. Nearing end of the project's due date, this came to light. Considering time limitations, this business logic was decided not to be implemented after a conversation with the product owner. Given a longer development period or more efficient work we could have implemented this.

### **5.5.7 Product specification**

Following the product specification, we made priorities based on the purpose of this application. Due to time limitations and time spent on business logic which could have been avoided. The product specification was not met. In hindsight we should have begun front-end development earlier and prioritized some back-end functionality different. This would have allowed us to scale the functionality of the application more equally between front-end and back-end. It would also provide us with user testing and feedback. As these are non-existent.

## **5.6 Further Development**

Some thoughts on what could be developed further.

HTTP compliance could be ensured. As some unused REST API functions reside on the back-end not HTTP compliant. I.e. GET methods requiring a body to process the request.

Kartverket API lookup for new Cabins which gave feedback to any user trying to add a cabin. Given the necessary information, there already is a function that can check if the property exists or not. This should give the user useful feedback about the request.

Exception handling front-end and back-end is poor. There should be implemented exception handling for every expected exception given an input. Those exceptions should give useful feedback if thrown, to the user and to the applications.

Redo the message system developed in the back-end as it does not fit its purpose. Currently it works as a chat system, though we believe it would function better as a ticketing system. Allowing users to ticket a ploughman or an admin, because a direct chat system does not function without a contacts catalogue, nor is it scalable.

A lot of groundwork is ready on the back-end application for missing functionality on the front-end. For example, most admin views are not implemented but the calls are there for front-end to use it. Would also use more time on the design of the application.

Push notifications for cabin owners when their cabin is ploughed. Could also implement push notifications based on the snow statistics data saved for each cabin area. Providing useful information to cabin owners and ploughmen about the snow height of any area.

Cabin status was meant to be updated automatically by a scheduled thread on the back-end. Data is retrieved for every weather station, given that they have the appropriate sensors. Using this data, such a system should be quite easy to implement.

User testing for the front-end application would be useful for any further development. Since the targeted age group for this application is not the quickest to learn new user interfaces.

## **5.7 Learned**

The development process started with a lot of project planning, which was useful to us. Setting milestones and defining product requirements early with the product owner. This taught us how useful a well thought of plan can be early on. As it gave us a good overview over tasks and a clear view of the result. Though keeping an appropriate project scope given a time frame must be learned through experience.

During the development process we got experience discussing specifications with the product owner. Presenting results and discussing the next sprint. Experience being in a development team using scrum methodology. Motivating group member through social interactions.

Most of us were unfamiliar with one or more tools used to develop this software. For example, the Spring framework or Flutter. This gave every group member the ability to grow as a developer. We also expanded on our knowledge of GIT and how to use it in a project group.



## 6 CONCLUSION

Through this project we learned the benefits of planning, the difficulties of time management for each task and communication with product owner. The result is a good base product for further development. With most main functionality present, lacking a bit in administrative functionality.

The server creation was done to try and automate the process of server creation and configuration to ease the workload and remove manual steps, as our back-end service got updated throughout the project. Looking back at the process we learned a lot of server creation and how it weaves the application as a whole, but for the best of the project keeping it simple would have been better as a whole.

The database was created with expansion and further development in mind, using clear naming conventions with first and second normal form present. Leaving only some restructuring and fragmenting to be compliant for higher normal forms.

Back-end application contains a good base structure with CRUD functionality and business logic for most requirements. Though some refactoring of class names and restructuring of packages is needed. The service layered architecture provides low cohesion and high coupling. Considering that this architecture is common for most back-end developers, it should be quite easy to maintain and further develop.

Front-end application provides most of the requirements for cabin owners and ploughmen, lacking a bit of admin functionality. A cabin owner can login, register his cabins and send ploughing requests. A ploughman can apply for access to ploughmen functionality. Through this functionality a ploughman can set himself available in regions called cabin areas. He can view ploughing requests within each region he is assigned to. He can also view the most efficient route. An administrator can accept ploughman requests. Though we wanted him to have more views to control users, cabins and cabin areas. For further development we would have refactored and restructured the code, as it was a learning by doing project.

If the product owner wants to further develop this service, we believe it has some solid base functionality to grow into a complete application. With some design, refactoring and a few views it would be complete. Developing a function or two on the back-end would provide them with statuses of snow fall. Ticketing system could be implemented for an easy communication channel to the administrators. New roles for a super admin to service the app is also possible.

## 7 REFERENCES

- [1] Lovdata, "Lovdata," 10 05 2020. [Online]. Available: [https://lovdata.no/dokument/NL/lov/2018-06-15-38/\\*#\\*](https://lovdata.no/dokument/NL/lov/2018-06-15-38/*#*).
- [2] Universitetet i oslo, "Universitetet i oslo," 10 05 2020. [Online]. Available: <https://app.uio.no/ub/ujur/oversatte-lover/data/lov-20000414-031-eng.pdf>.
- [3] Intersoft consulting, "Intersoft Consulting," 10 05 2020. [Online]. Available: <https://gdpr-info.eu/>.
- [4] Lovdata, "Lovdata," 10 05 2020. [Online]. Available: [https://lovdata.no/dokument/NL/lov/2018-06-15-38/KAPITTEL\\_gdpr-4#KAPITTEL\\_gdpr-4](https://lovdata.no/dokument/NL/lov/2018-06-15-38/KAPITTEL_gdpr-4#KAPITTEL_gdpr-4).
- [5] Datatilsynet, "Datatilsynet," 10 05 2020. [Online]. Available: <https://www.datatilsynet.no/rettigheter-og-plikter/virksomhetenes-plikter/behandlingsgrunnlag/veileder-om-behandlingsgrunnlag/samtykke/>.
- [6] P. Christensson, "techterms," 10 05 2020. [Online]. Available: <https://techterms.com/definition/hash>.
- [7] Wikipedia, "Wikipedia," 10 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Security\\_token#Single\\_sign-on\\_software\\_tokens](https://en.wikipedia.org/wiki/Security_token#Single_sign-on_software_tokens).
- [8] C. o. Code, "Scotch," 10 05 2020. [Online]. Available: <https://scotch.io/tutorials/the-anatomy-of-a-json-web-token>.
- [9] Oxford, "lexico," 10 05 2020. [Online]. Available: <https://www.lexico.com/en/definition/cryptography>.
- [10] TechTarget, "TechTarget," 11 05 2020. [Online]. Available: [https://cdn.ttgtmedia.com/rms/onlineImages/security\\_cissp\\_cryptography.jpg](https://cdn.ttgtmedia.com/rms/onlineImages/security_cissp_cryptography.jpg).
- [11] M. Rouse, "TechTarget," 11 05 2020. [Online]. Available: <https://searchsecurity.techtarget.com/definition/cryptography>.
- [12] Wikipedia, "Wikipedia," 11 05 2020. [Online]. Available: <https://en.wikipedia.org/wiki/HTTPS>.
- [13] K. J. Varsha R.Mouli, "Science Direkt," 11 05 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050916315113>.
- [14] M. B. A. V. B. .. Kent Beck, "Agile Alliance," 11 05 2020. [Online]. Available: <https://www.agilealliance.org/agile101/the-agile-manifesto/>.
- [15] J. v. d. Hoek, "Mendix," 11 05 2020. [Online]. Available: <https://www.mendix.com/blog/pursuing-a-full-agile-software-lifecycle/>.
- [16] Agile Alliance, "Agile Alliance," 11 05 2020. [Online]. Available: <https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/>.
- [17] scrum.org, "Scrum," 11 05 2020. [Online]. Available: <https://www.scrum.org/resources/what-is-scrum>.
- [18] MicroOne, "VectorStock," 11 05 2020. [Online]. Available: <https://www.vectorstock.com/royalty-free-vector/scrum-task-board-with-sticky-notes-for-agile-vector-21042397>.
- [19] Lakeworks, "Wikipedia," 11 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Scrum\\_\(software\\_development\)#/media/File:Scrum\\_process.svg](https://en.wikipedia.org/wiki/Scrum_(software_development)#/media/File:Scrum_process.svg).
- [20] Discord, "Discord," 11 05 2020. [Online]. Available: <https://discord.com/why-discord> .
- [21] Slack Technologies, Inc., "Slack," 11 05 2020. [Online]. Available:

- <https://slack.com/intl/en-no/features>.
- [22] Atlassian, "Atlassian," 11 05 2020. [Online]. Available: <https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for>.
- [23] D. Barnes and M. Kölling, "Objects First with Java - Sixth edition," in *Objects First with Java - Sixth edition*, Essex, Pearson Education Limited, 2017.
- [24] E. Goebelbecker, "Ndepend," 11 05 2020. [Online]. Available: <https://blog.ndepend.com/programming-coupling/>.
- [25] mauris, "Stackoverflow," 11 05 2020. [Online]. Available: <https://stackoverflow.com/questions/3085285/difference-between-cohesion-and-coupling>.
- [26] Git, "Git," 11 05 2020. [Online]. Available: <https://git-scm.com/book/en/v2/images/centralized.png>.
- [27] Git, "Git," 11 05 2020. [Online]. Available: <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>.
- [28] Codeship, "Codeship," 12 05 2020. [Online]. Available: <https://codeship.com/continuous-integration-essentials>.
- [29] M. Fowler, "martinfowler," 12 05 2020. [Online]. Available: <https://martinfowler.com/articles/continuousIntegration.html>.
- [30] Codeship, "Codeship," 12 05 2020. [Online]. Available: <https://codeship.com/continuous-integration-essentials>.
- [31] Codecademy, "Codecademy," 12 05 2020. [Online]. Available: <https://www.codecademy.com/articles/what-is-an-ide>.
- [32] Flutter, "Flutter.dev," 12 05 2020. [Online]. Available: <https://flutter.dev/docs/resources/faq>.
- [33] Wikipedia, "Wikipedia," 12 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Flutter\\_\(software\)](https://en.wikipedia.org/wiki/Flutter_(software)).
- [34] Wikipedia, "Wikipedia," 12 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Dart\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Dart_(programming_language)).
- [35] Google Developers, "Youtube," 12 05 2020. [Online]. Available: <https://www.youtube.com/watch?v=5KInlCq2M5Q>.
- [36] Dart, "Dart," 12 05 2020. [Online]. Available: <https://dart.dev/faq>.
- [37] SQLcourse, "SQLcourse," 12 05 2020. [Online]. Available: <http://www.sqlcourse.com/intro.html>.
- [38] Wikipedia, "Wikipedia," 12 05 2020. [Online]. Available: <https://en.wikipedia.org/wiki/SQL>.
- [39] ISO, "iso," 12 05 2020. [Online]. Available: <https://www.iso.org/standard/16661.html>.
- [40] ISO, "iso," 12 05 2020. [Online]. Available: <https://www.iso.org/standard/63565.html>.
- [41] Wikipedia, "Wikipedia," 12 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Java\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Java_(programming_language)).
- [42] Spring, "Spring," 12 05 2020. [Online]. Available: <https://spring.io/images/spring-logo-9146a4d3298760c2e7e49595184e1975.svg>.
- [43] The Apache Software Foundation, "apache maven project," 13 05 2020. [Online]. Available: <https://maven.apache.org/what-is-maven.html>.
- [44] Red Hat, "Ansible," 13 05 2020. [Online]. Available: <https://www.ansible.com/overview/how-ansible-works>.
- [45] HashiCorp, "Terraform," 13 05 2020. [Online]. Available: <https://www.terraform.io/>.
- [46] experienceux, "Experience ux," 13 05 2020. [Online]. Available: <https://www.experienceux.co.uk/faqs/what-is-wireframing/>.

- [47] Wikipedia, "Wikipedia," 13 05 2020. [Online]. Available: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#/media/File:MVC-Process.svg>.
- [48] rj45, "codeproject," 13 05 2020. [Online]. Available: <https://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design>.
- [49] Wikipedia, "Wikipedia," 13 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Design\\_Patterns#](https://en.wikipedia.org/wiki/Design_Patterns#).
- [50] Wikipedia, "Wikipedia," 13 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Front\\_end\\_and\\_back\\_end](https://en.wikipedia.org/wiki/Front_end_and_back_end).
- [51] Wikipedia, "Wikipedia," 13 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer).
- [52] T. B. Lee, "w3," 13 05 2020. [Online]. Available: <https://www.w3.org/DesignIssues/HTTP-URI.html>.
- [53] R. T. Fielding, "ics uci edu," 13 05 2020. [Online]. Available: [https://www.ics.uci.edu/~fielding/pubs/dissertation/net\\_app\\_arch.html](https://www.ics.uci.edu/~fielding/pubs/dissertation/net_app_arch.html).
- [54] Documentation.help, "Documentation.help," 11 05 2020. [Online]. Available: [https://documentation.help/DogeTool-HTTP-Requests-vt/http\\_requestmessageexample.png](https://documentation.help/DogeTool-HTTP-Requests-vt/http_requestmessageexample.png).
- [55] Network Working Group, "tools.ietf," 13 05 2020. [Online]. Available: <https://tools.ietf.org/html/rfc2616>.
- [56] w3, "w3," 13 05 2020. [Online]. Available: <https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html>.
- [57] Wikipedia, "Wikipedia," 13 05 2020. [Online]. Available: [https://upload.wikimedia.org/wikipedia/commons/8/8f/Service\\_Layers\\_Image\\_A.JPG](https://upload.wikimedia.org/wikipedia/commons/8/8f/Service_Layers_Image_A.JPG).
- [58] Wikipedia, "Wikipedia," 14 05 2020. [Online]. Available: [https://upload.wikimedia.org/wikipedia/commons/a/ac/Service\\_Layers\\_Image\\_B.JPG](https://upload.wikimedia.org/wikipedia/commons/a/ac/Service_Layers_Image_B.JPG).
- [59] Wikipedia, "Wikipedia," 11 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Service\\_layer\\_pattern](https://en.wikipedia.org/wiki/Service_layer_pattern).
- [60] Microsoft, "Microsoft," 14 05 2020. [Online]. Available: [https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649585\(v=pandp.10\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649585(v=pandp.10)?redirectedfrom=MSDN).
- [61] Wikipedia, "Wikipedia," 14 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Application\\_programming\\_interface](https://en.wikipedia.org/wiki/Application_programming_interface).
- [62] M. Heller, "infoworld," 14 05 2020. [Online]. Available: <https://www.infoworld.com/article/2640739/rest-and-crud--the-impedance-mismatch.html>.
- [63] Wikipedia, "Wikipedia," 13 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/One-time\\_password](https://en.wikipedia.org/wiki/One-time_password).
- [64] Wikipedia, "Wikipedia," 14 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Proxy\\_server#/media/File:Proxy\\_concept\\_en.svg](https://en.wikipedia.org/wiki/Proxy_server#/media/File:Proxy_concept_en.svg).
- [65] Wikipedia, "Wikipedia," 14 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Proxy\\_server](https://en.wikipedia.org/wiki/Proxy_server).
- [66] 1keydata, "1Keydata," 14 05 2020. [Online]. Available: <https://www.1keydata.com/database-normalization/first-normal-form-1nf.php>.
- [67] C. Hock-Chuan, "ntu.edu.sg," 14 05 2020. [Online]. Available: [https://www.ntu.edu.sg/home/ehchua/programming/sql/Relational\\_Database\\_Design.html](https://www.ntu.edu.sg/home/ehchua/programming/sql/Relational_Database_Design.html).
- [68] Wikipedia, "Wikipedia," 16 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Relational\\_model](https://en.wikipedia.org/wiki/Relational_model).

- [69] C. Mullins, "Tdan," 13 05 2020. [Online]. Available: <https://tdan.com/database-auditing-capabilities-for-compliance-and-security/8135>.
- [70] S. Rekhi, "medium," 13 05 2020. [Online]. Available: <https://medium.com/@sachinrekhi/don-normans-principles-of-interaction-design-51025a2c0f33>.
- [71] B. Shneiderman, "cs.umd.edu," 16 05 2020. [Online]. Available: [\[http://www.cs.umd.edu/~ben/goldenrules.html](http://www.cs.umd.edu/~ben/goldenrules.html).
- [72] Wikipedia, "Wikipedia," 13 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Material\\_Design](https://en.wikipedia.org/wiki/Material_Design).
- [73] Google, "Material," 11 05 2020. [Online]. Available: <https://material.io/design/introduction>.
- [74] Enginess, "Enginess," 11 05 2020. [Online]. Available: <https://www.enginess.io/insights/design-trend-material-design>.
- [75] M. Fowler, "martinfowler," 09 05 2020. [Online]. Available: <https://www.martinfowler.com/articles/injection.html> .
- [76] Amazon, "Amazon," 10 05 2020. [Online]. Available: <https://aws.amazon.com/types-of-cloud-computing/>.
- [77] Rajkumar, "software testing material," 10 05 2020. [Online]. Available: <https://www.softwaretestingmaterial.com/software-testing/> .
- [78] "software testing fundamentals," 16 05 2020. [Online]. Available: <http://softwaretestingfundamentals.com/unit-testing/>.
- [79] E. Aerts, "agconsult," 12 05 2020. [Online]. Available: <https://www.agconsult.com/en/usability-blog/user-testing-what-why-and-how/>.
- [80] A. Cugh, "Journaldev," 18 05 2020. [Online]. Available: <https://www.journaldev.com/24601/java-11-features>.
- [81] Wikipedia, "Wikipedia," 18 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Java\\_servlet](https://en.wikipedia.org/wiki/Java_servlet).
- [82] Oracle, "Oracle," 18 05 2020. [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/java/util/class-use/TimerTask.html>.
- [83] Flutter, "Flutter," 16 05 2020. [Online]. Available: <https://flutter.dev/docs/development/ui/widgets-intro#basic-widgets>.
- [84] Flutter, "Flutter," 16 05 2020. [Online]. Available: <https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html>.
- [85] Flutter, "Flutter," 16 05 2020. [Online]. Available: <https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html>.
- [86] RouteXL, "RouteXL," 12 05 2020. [Online]. Available: <https://www.routexl.com/>.
- [87] Metrologisk institutt, "Frost," 15 05 2020. [Online]. Available: <https://frost.met.no/index.html>.
- [88] Google, "Developers google," 13 05 2020. [Online]. Available: <https://developers.google.com/maps/web-services/client-library> .
- [89] Kartverket, "Kartverket," 13 05 2020. [Online]. Available: <https://www.kartverket.no/Om-Kartverket/Kartverket/> .
- [90] Wikipedia, "Wikipedia," 13 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Google\\_Cloud\\_Platform](https://en.wikipedia.org/wiki/Google_Cloud_Platform).
- [91] Google, "Google," 15 05 2020. [Online]. Available: <https://google.github.io/styleguide/javaguide.html>.
- [92] Flutter, "Github," 13 05 2020. [Online]. Available: <https://github.com/flutter/flutter/wiki/Style-guide-for-Flutter-repo>.
- [93] Spring, "Spring," 05 05 2020. [Online]. Available: <https://spring.io/projects/spring-boot>.
- [94] Spring, "Spring," 05 05 2020. [Online]. Available: <https://spring.io/projects/spring->

data.

- [95] Spring, "Spring," 05 05 2020. [Online]. Available: <https://spring.io/projects/spring-security>.
- [96] Wikipedia, "Wikipedia," 16 05 2020. [Online]. Available: <https://en.wikipedia.org/wiki/MySQL>.
- [97] Wikipedia, "Wikipedia," 16 05 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Comparison\\_of\\_MySQL\\_database\\_engines](https://en.wikipedia.org/wiki/Comparison_of_MySQL_database_engines).
- [98] Postman, "Postman," 16' 05 2020. [Online]. Available: <https://www.postman.com/>.
- [99] Gitlab, "Gitlab," [Online]. Available: <https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/>.
- [100] Gitlab, "Gitlab," 16 05 2020. [Online]. Available: <https://about.gitlab.com/>.  
]
- [101] Eclipse, "Eclipse," 17 05 2020. [Online]. Available: <https://www.eclipse.org/eclipseide/>.  
]
- [102] Google, "Android," 16 05 2020. [Online]. Available: <https://developer.android.com/studio>.  
]
- [103] Microsoft, "Visualstudio," 16 05 2020. [Online]. Available: <https://code.visualstudio.com/>.  
]
- [104] HasiCorp, "Terraform," 16 05 2020. [Online]. Available: <https://www.terraform.io/docs/commands/validate.html>.  
]
- [105] HashiCorp, "Terraform," 13 05 2020. [Online]. Available: <https://www.terraform.io/docs/commands/plan.html>.  
]
- [106] HashiCorp, "Terraform," 16 05 2020. [Online]. Available: <https://www.terraform.io/docs/commands/apply.html>.  
]
- [107] HashiCorp, "Terraform," 16 05 2020. [Online]. Available: <https://www.terraform.io/docs/commands/plan.html>.  
]
- [108] HashiCorp, "Terraform," 16 05 2020. [Online]. Available: <https://www.terraform.io/docs/commands/destroy.html>.  
]
- [109] HashiCorp, "Terraform," 16 05 2020. [Online]. Available: <https://www.terraform.io/docs/modules/index.html>.  
]
- [110] Google, "Google Cloud," 16 05 2020. [Online]. Available: <https://cloud.google.com/community/tutorials/getting-started-on-gcp-with-terraform>.  
]
- [111] HashiCorp, "Terraform," 16 05 2020. [Online]. Available: <https://www.terraform.io/docs/providers/google/>.  
]
- [112] RedHat, Inc. , "Ansible," 16 05 2020. [Online]. Available: [https://docs.ansible.com/ansible/latest/plugins/inventory/gcp\\_compute.html](https://docs.ansible.com/ansible/latest/plugins/inventory/gcp_compute.html).  
]
- [113] Google, "Google Cloud," 16 05 2020. [Online]. Available: <https://cloud.google.com/compute/docs/images>.  
]
- [114] C. Sells, "Medium," 17 05 2020. [Online]. Available: <https://medium.com/flutter/announcing-flutter-1-17-4182d8af7f8e>.  
]
- [115] World Health Organization, "World Health Organization," 18 05 2020. [Online]. Available: <http://www.euro.who.int/en/health-topics/health-emergencies/coronavirus-covid-19/novel-coronavirus-2019-ncov-technical-guidance-OLD/coronavirus-disease-covid-19-outbreak-technical-guidance-europe-OLD/mental-health-and-covid-19>.  
]
- [116] Atlassian, "Atlassian," 15 05 2020. [Online]. Available: <https://www.atlassian.com/software/confluence>.  
]
- [117] Spring, "Spring," 15 05 2020. [Online]. Available: <https://www.atlassian.com/software/confluence>.  
]
- [118] F. Spiridonov, "Stackoverflow," 13 05 2020. [Online]. Available:

] <https://stackoverflow.com/questions/3325387/infinite-recursion-with-jackson-json-and-hibernate-jpa-issue>.

[119 V. Beal, "Webopedia," 15 05 2020. [Online]. Available:

] [https://www.webopedia.com/TERM/C/cloud\\_services.html](https://www.webopedia.com/TERM/C/cloud_services.html).

## **APPENDIX**

Appendix 1	Pre-project report including requirement specification
Appendix 2	Meeting reports
Appendix 3	Planning documents
Appendix 4	Sprint reports



## **APPENDIX 1**

# PRE-PROJECT - REPORT

## FOR BACHELOR THESIS

**TITLE:**

Mobile application for snow plowing

**CANDIDATE(S):**

**Henrik Ranvik Bjørnland, Marius Blokkum Nakstad, Theodor Giskegjerde Urtegård**

**DATE:**

30. Jan. 2020

**COURSE CODE:**

IE303612

**COURSE TITLE:**

Bacheloroppgave (Data)

**RESTRICTION:****STUDY PROGRAM:**

BACHELOR I INGENIØRFAG - DATA

**PAGES/APPENDIX:**

17 / 2

**LIBRARY NO.:****CLIENT(S)/SUPERVISOR(S):**

Client(s): Avento AS

Supervisor(s): Kjell Inge Tomren, Girts Strazdins

**SUMMARY:**

Avento AS wishes to develop a system for snow plowers and cabin owners. This system should improve communication between both parties mentioned. Snow plowers should be able to organize themselves by the use of locations. The map should display occupied cabins that the plower is responsible for. This project is given to students by the division of science and engineering, computer engineering. This pre-project report is a description of the bachelor thesis.

The bachelor thesis should develop mentioned system, with focus on efficiency and management. User interface should be easy to use for users that falls into the group of cabin owners, smartphone users and snow plowers. By the end of the project, this should be a working product, with possibility of expansion.

The scope of this project is development of a back-end service, with a front-end user-interface.

Denne oppgaven er en eksamensbesvarelse utført av student(er) ved NTNU i Ålesund.

## TABLE OF CONTENTS

<b>TABLE OF CONTENTS</b> .....	<b>2</b>
<b>1 INTRODUCTION</b> .....	<b>3</b>
<b>2 TERMINOLOGY</b> .....	<b>3</b>
<b>3 PROJECT MANAGEMENT</b> .....	<b>3</b>
3.1 PROJECT GROUP .....	3
3.1.1 <i>Tasks for the project group – organization and tasks</i> .....	3
3.2 SUPERVISORS AND CLIENT .....	4
<b>4 AGREEMENT</b> .....	<b>4</b>
4.1 AGREEMENT WITH CLIENT .....	4
4.2 WORKPLACE AND RESOURCES .....	4
4.3 GROUP NORMS – COOPERATION RULES – ATTITUDES .....	5
4.3.1 <i>Group norms - cooperation rules</i> .....	5
4.3.2 <i>Attitudes</i> .....	5
<b>5 PROJECT DESCRIPTION</b> .....	<b>6</b>
5.1 ISSUE - GOALS - INTENT .....	6
5.1.1 <i>Issue</i> .....	6
5.1.2 <i>Goals and intent</i> .....	6
5.2 CRITERIA FOR THE SOLUTION OR THE PROJECT RESULT – SPECIFICATION .....	6
5.3 PROGRESS PLANNING, COOPERATION – METHOD(S) .....	7
5.4 RESEARCH – KNOWN AND UNKNOWN RELEVANT INFORMATION .....	7
5.5 ASSESSMENT – RISK ANALYSIS .....	8
5.6 MAIN ACTIVITIES IN FURTHER WORK .....	8
5.7 PROGRESS PLAN – PROJECT MANAGEMENT .....	9
5.7.1 <i>Main plan</i> .....	9
5.7.2 <i>Management tools</i> .....	9
5.7.3 <i>Development tools</i> .....	9
5.7.4 <i>Internal regulation – evaluation</i> .....	9
5.8 DECISIONS – THE RULING PROCESS .....	10
<b>6 DOCUMENTATION</b> .....	<b>10</b>
6.1 REPORTS AND TECHNICAL DOCUMENTS .....	10
<b>7 MEETINGS</b> .....	<b>10</b>
7.1 MEETINGS WITH THE CLIENT AND SUPERVISOR(S) .....	10
7.2 PROJECT MEETINGS .....	10
7.3 PERIODICAL REPORTS .....	10
7.3.1 <i>Progress reports</i> .....	10
<b>8 DEVIATION HANDLING</b> .....	<b>11</b>
<b>9 EQUIPMENT REQUIREMENTS/ PREREQUISITES FOR COMPLETION</b> .....	<b>11</b>
<b>10 BIBLIOGRAPHY</b> .....	<b>11</b>
<b>APPENDIX:</b> .....	<b>11</b>
<b>APPENDIX 1 – REQUIREMENT SPECIFICATION</b> .....	<b>12</b>
<b>APPENDIX 2 – ROADMAP</b> .....	<b>17</b>

# 1 INTRODUCTION

We have been tasked by Avento AS, a consulting firm based in Ålesund and Ørsta (Avento AS, 2020), to develop a system for optimizing the process around snow plowing for cabin areas. The system will make the organizing and communication between a snow plower and cabin owners simpler. Furthermore, help reduce the amount of unnecessary plowing, resulting in saved time and money, as well as a reduction of emissions and road deterioration from plowing equipment.

The system will consist of a backend server which is up to date and synchronized with cabin statuses. Cabin owners can check the mobile application for cabin status, and request plowing. Snow plowers can get incoming requests in the mobile application and a map of the cabins which need plowing to help plan their route.

## 2 TERMINOLOGY

## 3 PROJECT MANAGEMENT

### 3.1 Project group

Student number(s)
476066 - Marius Blokkum Nakstad 476083 - Theodor Giskegjerde Urtegård 483869 - Henrik Ranvik Bjørnland

#### 3.1.1 Tasks for the project group – organization and tasks

The group is responsible for working on the project and ensuring the goals are met, all members are responsible for communicating with each other and keeping the group up to date. Each member of the group is responsible for attending meetings or notifying beforehand if they can't attend.

Seen as we are a relatively small group, we have opted to not have a dedicated leader, instead everyone in the group act, work and take decisions together based on our group ethics and group democracy. Decisions within the group will be taken based on an agreement within all the members of the group. If a common agreement cannot be agreed upon, we will come to a common consensus through talking, discussions and explanation of our idea or solution and see if we can reach a middle group. If, however this still does not make the group reach a common consensus, there will be a vote within the group and the solution with the most votes is the solution which will be taken by the group.

Communication with client and supervisor will happen through common communication channels such as a Slack or through e-mail, and it is everyone's responsibility to communicate if they have any questions or need to communicate with the client or supervisor in some way.

Everyone is responsible for ensuring that notes are being taken during a meeting, either by taking notes themselves or verifying that at least one group member is taking notes. Meeting notes should be filed in confluence under meeting and to be written in the meeting template for all group members to view.

Everyone should look out for each other and help with tasks to ensure a fair and pleasant work environment.

## 3.2 Supervisors and client

Client	–	Avento AS
Client contact	–	Anders Hatlehol Beite
Primary supervisor	–	Kjell Inge Tomren
Secondary supervisor	–	Girts Strazdins

## 4 AGREEMENT

### 4.1 Agreement with client

We have an agreement with the client on how often, at what time and where we should have a meeting regarding the project and our progress, which is every second Tuesday from 09<sup>00</sup> to 10<sup>00</sup> at Avento's offices. We also have an agreement with the client regarding what functionality our solution is required to have, defined in the requirement specification. The requirement specification can be found as [appendix 1](#). Lastly, we have an agreement to communicate either through Slack or email.

Otherwise we are free to use the technologies we see fit regarding the project requirement and our own knowledge and competence.

A written agreement with the client has not been discussed, if requested by the client an NDA using NTNU's template will be preferred.

### 4.2 Workplace and resources

We have several workplaces and resources available to us:

Workplace:

- Daily workspace: NTNU / NMK
- Meetings: Avento's office

Resources:

- Personal computers / laptops for development
- Private smartphones / phones loaned from NTNU to test the application
- APIs from Kartverket, Google and more which we can use to get information for our system
- Server space on NTNU or Google to have the back end running on
- Jira and confluence for issue tracking and project documentation
- Slack for communication
- Library in case we need literature resources

Personnel:

- Primary supervisor: Kjell Inge Tomren
- Secondary supervisor: Girts Strazdins
- Client contact: Anders Hatlehol Beite
- Project participants: Marius Blokkum Nakstad, Henrik Ranvik Bjørnland, Theodor Giskegjerde Urtegård.

Reports:

- Daily report between project participants.
- Reports generated in Jira or Confluence
- Report with client and primary supervisor every fortnight Tuesday between 09<sup>00</sup> and 10<sup>00</sup>.

Literature:

- Any literature we might need which we can borrow from the university library
- Helpful documents from online resources
- Previous bachelor reports

### **4.3 Group norms – Cooperation rules – Attitudes**

#### **4.3.1 Group norms - cooperation rules**

Decisions should be made in agreement of all members. If such an agreement cannot be met, a vote must be held. All members must be present. To not neglect any voters, their opinion is to be voiced and heard, before a decision is made. The majority of votes win.

Every member should do an equal amount of work. If someone is absent, they must notify the project lead of reason and time period. If a member is lacking in workdays, he should act to be more present.

We have agreed to a core work schedule between 08:00 and 16:00 from Monday to Thursday, with Friday being flexible in contrast to progress.

Members should treat each other, and their time with respect, as in a professional work environment.

#### **4.3.2 Attitudes**

The product should profit our community on a local and global scope. Data stored on the server must be encrypted and difficult to access for individuals who should not have access. This information is not to be used to gain commercial value. Individuals who have access should act discreetly and treat the data as confidential, the data is not to be misused by a malicious entity.

## 5 PROJECT DESCRIPTION

### 5.1 Issue - Goals - Intent

#### 5.1.1 Issue

Previously the communication between a cabin owner and a snow plower were done by phone and later e-mail or text messages. Now with the widespread usage of smartphones and digitalization there is a market for new solutions which can further help and improve the communication between cabin owner and snow plower, as well as making the snow plowers job easier and faster.

The issues become as follows:

Create a mobile application and a service that will ensure good communication between the cabin owner and the plower.

Create a solution to help optimize the ploughing process.

#### 5.1.2 Goals and intent

This project's goal is to create a competitive product, that can provide benefits for a snow plower and a cabin owner to use.

The product will be an application, provided on Android OS smartphones, with a central backend which synchronizes all requests and actions from the application itself. This application should mainly provide a communication channel between cabin owner and snow plower.

The goals for our system are as follows:

Effect goal:

- Create a service which creates a smarter and more modern communication channel between cabin owners and snow plowers.
- Measurably reduce time spent on planning and unnecessary ploughing by the snow plower.
- Give customers a user-friendly platform in which they can express their needs. The platform should be easier to use than systems existing today.

Result goal:

- Deliver a system that is modular and can easily be expanded.
- Deliver a front-end application for Android OS smartphones.
- Deliver a backend solution with a central server and database.
- Create a system which fulfils the requirement specification ([Appendix 1](#)) and which meets the effect goals.

Process goal:

- Use old and learn new knowledge in the process of creating an application for a customer.
- Work according to our own standards, our own documents, the requirement specification, scrum and our client.
- Meet the effect and result goals within the timeframe (16. January – 22. May.)

### 5.2 Criteria for the solution or the project result – specification

The result of the project should be a working snow plowing service which makes it easier for plowers to know where they need to plow based on demand. The project result should contain a working application with both front and back-end elements.

The project result should cover everything within the approved requirement specification ([appendix 1](#)).

### **5.3 Progress planning, cooperation – method(s)**

To track progress, the group will use the Jira sprint tool. A sprint will last 2 weeks. At the beginning of each sprint, the project group, primary supervisor and client, will have a progress report. At the end of the report, the next sprint will be planned. A sprint consists of different types of issues, most importantly the user stories. User stories will describe the client's needs for the project. Subtasks is to be created and managed by the project group, to track progress. If an issue is not completed by the end of a sprint, a discussion with the necessary parties will be held. The discussion should conclude if the issue is necessary. If the issue is still needed, then it will be a part of a subsequent sprint.

Jira's sprint tool is lightweight and easy to use. The project group consists of three members, making the use of a lightweight project management tool, easier. The tool brings a lot of transparency to the client, as they can see progress on several levels. A drawback of this, is that the project is often of subject to change. Sprint brings, the client and project members, continuous feedback on progress. The continuous feedback makes changes easier to implement and gives the client more power over the result. Progress is measured by weighted issues each individual completes, reflecting every team member's effort. This can be motivating for project members, as they can clearly see their progress. Although, also discouraging, as it might not correctly reflect the effort of a team member.

For everyone to work together on the same codebase, a version control tool must be utilized. Gitlab provides everything which would satisfy the group's needs in terms of allowing everyone to work on the same code at the same time, ensure consistency, allow for rollbacks and keep track of different version. Gitlab also provides useful tools such as CI solutions, pipelines as well as Slack and Jira integration.

To be sure the effort of everyone is not reflected unfairly, the members must report their work hours. Ensuring that other means of tracking the effort of a member is available. To prevent the client of substantially changing the project, good communication is needed.

### **5.4 Research – known and unknown relevant information**

Avento AS already introduced an existing web application, in which our mobile application functions were implemented. There is also an existing solution called *Hytteankomst* (Netsense Software Services AS, 2020), which uses mobile platforms and web platforms. The *Hytteankomst* application looks a bit outdated, at least in terms of colour palette and UX. Further research is planned into existing solutions.

The technologies that will be used is not decided and needs more research. Some are in consideration, as Flutter for front-end, Spring Boot for REST API and MySQL for the database. More research of best practices in chosen technologies will follow.

Further research will involve security, UX design and best practices, useful APIs such as Google's and Kartverket's, etc.



### 5.5 Assessment – risk analysis

Threat	Consequence	Probability	Countermeasure
Loss of data	Need to generate new data, loss of time	MEDIUM	Write code on Gitlab and use google docs or other cloud services to keep backup.
Misunderstand customer need	Project exceeding time boundaries.	LOW	Continuous meetings with customer, working with sprints and good communication channels I.e. Slack.
Unstructured project	Low efficiency.	LOW	Good project structure with agile background and weekly meetings as a group to get on par.
Poor group communication	Misunderstandings	MEDIUM	Good communication channels within the group, I.e. Facebook, Discord, Slack etc...
Sickness	Loss of workhours, information and productivity.	LOW	Good work environment, respect for each other's personal lives and use of communication channels.
Time	Unfinished product according to the product specification.	MEDIUM	Client is aware of the project's small timeframe. Project management tool, such as Jira's sprint tool.

### 5.6 Main activities in further work

Roadmap can be found in [appendix 2](#).

- Database
- Rest API
- User Interface
- POSTMAN
- Kartverket API Integrated
- Application is talking to the database, API and UI
- Bug testing
- User testing
- Unit testing
- Bachelor report
- Poster for bachelor report

## **5.7 Progress plan – project management**

### **5.7.1 Main plan**

The project will be organized through Jira to plan and observe sprints and work objectives. Big user stories will be split down into chunks that can be completed in a timeframe of a sprint. Jira will also be used to track start and stop timestamps for issues in a sprint.

Sprints will be started after the planned meeting with customer; the sprint will end before the meeting. The meetings will be used to discuss changes for the next sprint considering both inputs from customer and the roadmap.

#### Milestones

- Pre-work:  
UML diagrams, UI wireframes, developer agreement, testing plan, database design and class diagrams are finished or in review by the customer is finished.
- Work  
This is the state the group will for the most be in; it's here we use Jira to track and plan activities. This milestone will also be split into smaller secondary milestones as database, API, front-end, user test and more.
- Finished project:  
The project is to be declared completed when 5.6 main activities in further work is completed and reviewed.

### **5.7.2 Management tools**

Different management is required for project functionalities. These are:

- Jira for issue tracking and sprint management
- Confluence for project documentation
- Discord for internal communication
- Slack for internal and external communication with client and supervisor
- Google Drive for internal cloud file sharing

### **5.7.3 Development tools**

Different development tools are required for different project components. These are:

- Jira for project management
- Gitlab for version control, continuous integration and automatic testing
- IntelliJ or Visual Studio Code for IDE
- Flutter for application development
- Spring Boot for backend services
- MySQL for database services

### **5.7.4 Internal regulation – evaluation**

Internal control will be done through Jira, which can track progression throughout the project. Each member should follow all issues being completed, test and/or evaluate the result. For an issue to be completed, the described function should be implemented without any bugs directly caused by the new function. If the new function is being affected indirectly, a new issue should be registered.

## **5.8 Decisions – the ruling process**

As defined in section [3.1.1](#), decisions within the group will be taken based on an agreement within all the members of the group. If a common agreement cannot be agreed upon, we will come to a common consensus through talking, discussions and explanation of our idea or solution and see if we can reach a middle group. If, however this still does not make the group reach a common consensus, there will be a vote within the group and the solution with the most votes is the solution which will be taken by the group.

Major changes or decisions to the project or project definition will happen with the client meeting every second Tuesday if it is deemed necessary to make such changes,

# **6 DOCUMENTATION**

## **6.1 Reports and technical documents**

Most technical documents such as design documents, reports and documentation will be handled within Jira or Confluence.

Jira will contain the documents regarding issue tracking, issue burndown charts and sprint reports.

Confluence will contain our design documentation such as UI mockups, use case diagrams, activity diagrams, requirement specification, deployment diagrams and other UML diagrams we see fit to help document our project.

Confluence will also contain our meeting notes, reports, testing plan, developer agreement and sprint retrospectives.

The bachelor report will contain everything regarding our project and will serve as a complete documentation of our entire project and will be actively worked on during development.

The bachelor poster will be created at the end of the report for summary and presentation.

# **7 MEETINGS**

## **7.1 Meetings with the client and supervisor(s)**

Meeting with the client and supervisor(s) every other Tuesday in at 09:00. Other external meetings are organized if only one part is to be contacted.

## **7.2 Project meetings**

The group will have a sprint meeting after every sprint in order to plan for the next. Otherwise the group will not have set meetings, as all members are sharing office. Internal meetings will be held as necessary and communication within the group is continuous.

## **7.3 Periodical reports**

### **7.3.1 Progress reports**

We will generate sprint report in Jira after each sprint; which is to be shown to the customer at the two-week meetings. We will also plan the next sprint in this meeting.

## 8 DEVIATION HANDLING

If a user story is not completed at the end of a sprint, the client should be informed. We will check if the user still wants this story to be completed in the next sprint or to focus on other tasks.

If issues in a sprint is not completed at the end of a sprint, the group will discuss if the issue should be added to the next sprint.

If changes to the project plan is necessary, the changes will be added to the next sprint. Changes should not occur in an ongoing sprint.

If a main goal is not reached by the set deadline, then that main goal is worked on until satisfactory standards of the product is met.

## 9 EQUIPMENT REQUIREMENTS/ PREREQUISITES FOR COMPLETION

- Kartverket's API has free and premium functionality, it is not yet decided if premium functionality is needed.
- An iPhone need to be borrowed from the University in the event it is decided that development for IOS will happen.

## 10 BIBLIOGRAPHY

(n.d.).

Avento AS. (2020, 01 30). *www.avento.no*. Hentet fra Avento AS corporate website:  
<https://www.avento.no/hvem-vi-er/>

Chandana. (2020, 01 30). *simplilearn: scrum project management article*. Retrieved from simplilearn:  
<https://www.simplilearn.com/scrum-project-management-article>

Malik, S. (2020, 01 30). *Quickstart blog: pros and cons of scrum methodology*. Retrieved from Quickstart blog website: <https://www.quickstart.com/blog/pros-and-cons-of-scrum-methodology>

Netsense Software Services AS. (2020, 01 30). *Hytteankomst Welcome Page*. Hentet fra Hytteankomst Web site: <https://hytteankomst.no/front.php>

Saher. (2020, 01 30). *pcskull: pros and cons scrum methodology*. Retrieved from pcskull:  
<https://www.pcskull.com/pros-and-cons-scrum-methodology/>

## APPENDIX:

## APPENDIX 1 – REQUIREMENT SPECIFICATION

### Requirement specification - “SnøJobb”

This specification defines the requirements, functionality and goals for a cabin snow plowing application.

#### The purpose

The purpose of this application is to allow cabin owners to indicate to snow plowers in an efficient and fast manner that their cabin(s) will require plowing at a specified time, thus giving the snow plower an indication of which cabin roads need to be plowed and which ones can be delayed.

#### The requirements

We have been requested to make a logistic system for optimization of cabin road plowing. The system needs different users, the administrator, the plowers and the cabin owners. The cabin owners want to inform when they are at the cabin and in need of plowing. The plowers need to be able to see the occupations and inform the cabin owners when they have plowed their cabin driveway. The administrator needs to be able to manage users and apply user requests if necessary.

#### Goals

Create a system which provides information for the snow plowers in an efficient manner as well as the communication between cabin owners and snow plowers.

#### Assumptions

Assumption	Description
For the application to function we rely on <b>kartverkets API</b> to be working	Much of the functions will rely on getting information from kartverkets API
It is required for the user to have a <b>smartphone</b>	The application will only run on smartphones
A working <b>network connection</b> is required	The application require access to kartverkets API and to be able to communicate between the different roles network connection is needed.

## Functional requirements

### *All users*

User requirements	Requirement description
The user should be able to register themselves	For the user to access the functionality in the app they must create an account with essential information.
The user should be able to log in and out	For the user to access data and functionality in the app, the user must confirm his identity by login methods. The user should also be able to log out, to prevent others from accessing data and functionalities.
The user should be able to get an overview of their own information	An authenticated user should be able to see their own information and be able to verify that the information is correct.
The user should be able to edit their own user information	A user should be able to change or update their information in case it is incorrect or outdated.
The user should be able to delete his user.	For the user to resign as a member of the app, they should be able to delete their user, if they provide the correct credentials.
The user should be able to have multiple roles.	For a user to be a cabin owner, snow plower and/or administrator, they should be able to have multiple roles as each role is not mutually exclusive.

***Cabin owner requirements***

User requirements	Requirement description
The user should be able to register themselves as a cabin owner	For a user to access cabin owner functionality they must be the owner of at least one cabin.
The user should be able to register a time period, in which the cabin will be in use.	For a user to notify a snow plower that the cabin driveway needs to be plowed, they should be able to set a time period in which the cabin will be in use.
The user should be able to edit the occupied status.	For a user to notify a snow plower that the cabin will not be in use or that they are extending their stay, they should be able to edit their snow plowing request.
The user should be able to add cabin area(s) for their cabin(s).	Registration or editing cabin(s) the user should be able to add/remove a cabin area to the cabin.
The user should be able to own multiple cabins	One person can own multiple cabins and thus a user must be able to register multiple cabins to the same account.
The user should be able to get notified when a request has been completed.	For a user to know that their driveway has been plowed, they should get a notification on their smartphone.
The user should be able to send individual requests to existing plower(s) in the area of their cabin(s).	A cabin owner should be able to send a request to an existing plower(s) for their cabin(s) to be assigned to.
The user should be able to remove a cabin from his registered cabins.	In the event that a user does no longer own a cabin they should be able to remove that cabin from their account.
The user should be able to voluntarily show their contact information to their plower	A cabin owner should be able to voluntarily opt in to show contact information on their cabins' information inside the application in the event that a plower might need to contact them directly.
The user should be able to contact administrator	For a user to solve problems that require an administrator or a super administrator, they must be able to contact the administrator.
The user should be able to request new cabin area	A cabin owner should be able to request new cabin areas, to be approved by administrator.

*Snow plower requirements*

User requirements	Requirement description
The user should be able to register as plower	For a user to access plowing functions, they should be able to register themselves as a snow plower. But need administrator approval before being accepted.
The user should be able to assign working area(s)	The snow plower should be able to pick cabin area(s) where they operate within
The user should be able to see plow request for cabins	For a plower to be able to plow the necessary driveways, they must be able to see which cabins have had their driveways requested to be plowed.
The user should be able to confirm plowing	Once a cabin driveway has been plowed, the plower should be able to mark the job as “complete”.
The user should be able to see a map of cabins	For a user to be productive in their plowing, they should be able to see a map of the cabins which they are responsible for plowing the driveway to, regardless if there is a plowing request or not.
The user should be able to get notification of new plow request	For a user to be more aware of upcoming work, they should be notified of new requests from cabin owners.
The user should be able to contact cabin owners	In the event that a plower needs to contact a cabin owner the plower should be able to do so through the application, or directly if the cabin owner has opted in to show their contact information in the cabin information
The user should be able to contact administrator	For a user to solve problems that require an administrator or a super administrator, they must be able to contact the administrator.



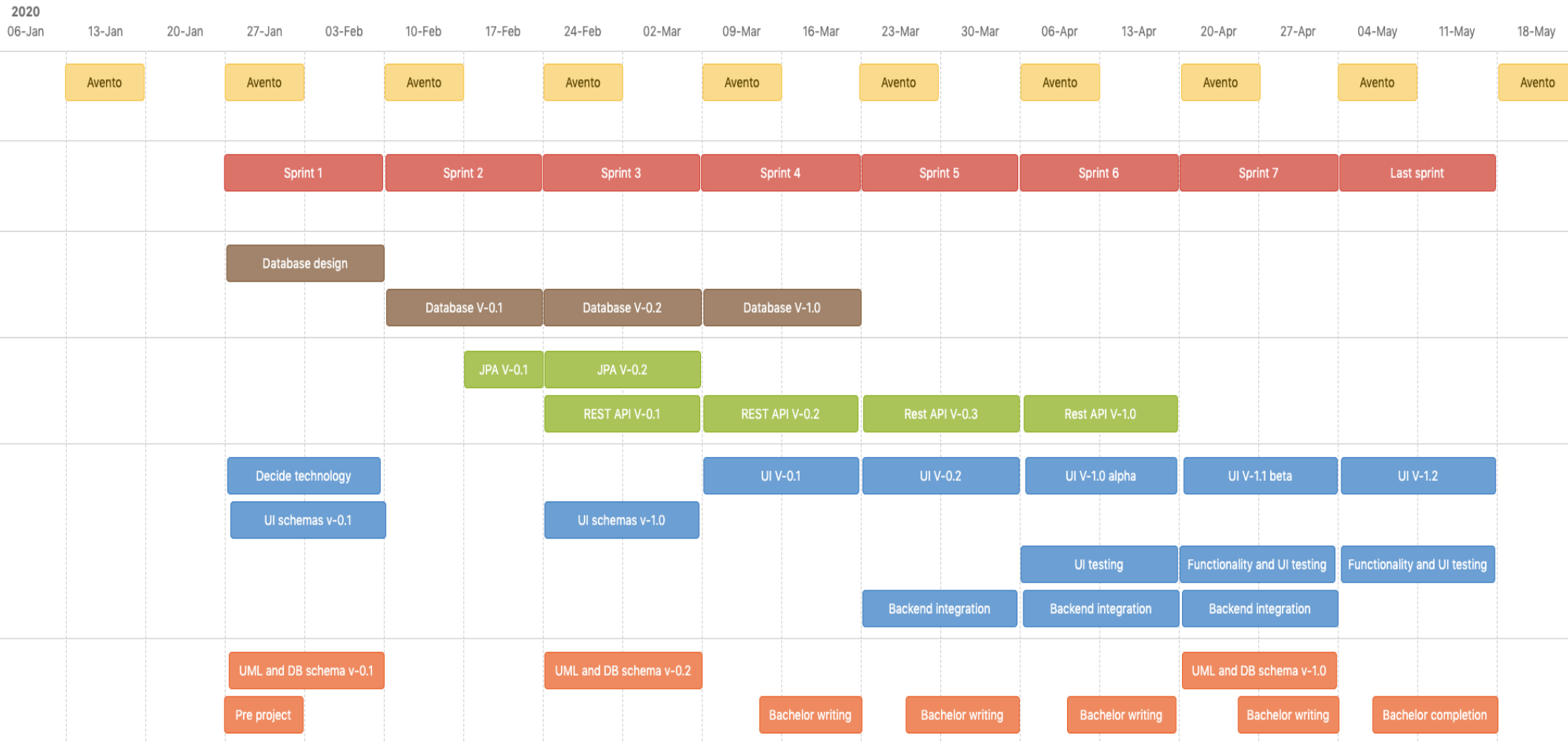
***Administrator requirements***

User requirements	Requirement description
The user should be able to contact cabin owners and plowers	For a user to solve problems and/or other relevant issues, they may contact plowers and cabin owners, to resolve said problems and/or other relevant issues.
The user should be able to edit cabin owner and snow plower users.	An administrator should be able to edit a user in the event it might deem necessary.
The user should be able to look up snow plowers, cabins and cabin owners by certain search parameters.	To be able to edit users and/or properties connected to a user, the administrator should be able to find the relevant user and connected properties through a search field.
The user should be able to approve registration of snow plower	When someone applies to the snow plower role, the administrator should be able to approve/reject the request.
The user should be able to define cabin area(s)	Administrator should be able to form new cabin area(s).

***Non-functional requirements***

Requirements	Requirement description
Temporarily store cookies for automatic login	For the app to be user friendly and secure, it will store cookies received which time out after a set time period depending on the role of the user.
Responsive UI	For the UI to be user friendly it should respond to a user's input, in such a way that the app feels quick in response.
User friendly UI	Most users should be able to maneuver the app quickly without any steep learning curve.
Secure user data	User data saved should be non-accessible to users or non-users which are not intended to see said data.
The app should be available on most android devices.	The app should be able to use on most recent Android OS versions.  (Flutter supports: Android Jelly Bean, v16, 4.1.x or newer)
The application requires a network connection.	For a mobile to access the apps functionality, it must have an internet connection.
Useful feedback on errors	The app should give useful feedback on errors occurring.

## APPENDIX 2 – ROADMAP



## **APPENDIX 2**

# 2020-01-28 møtenotater

## Dato

28.jan.2020

## Deltakere

- Henrik Bjørnland
- Marius Nakstad
- Theodor Urtegård
- Kjell Inge Tomren

## Mål

- Gjennomgang av forprosjekt rapport

## Debattpunkter

Tidsbruk	Element	Hvem	Notater
10 min	Jira/confluence	Alle	<ul style="list-style-type: none"><li>• Oppdatering Jira / Confluence for støtte til Gitlab og portfolio meny i Jira</li></ul>
10 min	Forprosjekt	Alle	<ul style="list-style-type: none"><li>▪ Gjennomgang av rapport</li><li>▪ Tilbakemelding og oppdatering på hva som behøves innenfor kapitlene</li></ul>
2min	Avtale bedrift	Alle	<ul style="list-style-type: none"><li>▪ Avtale med bedrift</li><li>▪ Ikke nødvendig for vår del, opp til Avento</li></ul>

## Handlingselementer

- [Henrik Bjørnland](#) Secretary

# 2020-02-11 møtenotater

## Dato

11.feb.2020

## Deltakere

- Henrik Bjørnland
- Marius Nakstad
- Theodor Urtegård
- Kjell Inge Tomren
- Anders beite, Avento

## Mål

- Gjennomgang av diagrammer, planlegging av sprint

## Debattpunkter

Tidspunkt	Element	Hvem	Notater
09:00	Wireframe	Alle	Mulige endringer <ul style="list-style-type: none"><li>• Optimere rute basert på hvilke hytter som er aktive</li><li>• Loggføre snøploger gps data</li><li>• Implementere vær data, f.eks: Google, YR</li></ul>
09:10	UML diagram	II	Database <ul style="list-style-type: none"><li>▪ Viste flere valg til kunde, bestemt å bruke valg #2.</li><li>▪ Brukere deles inn i roller med many to many</li></ul> User interface <ul style="list-style-type: none"><li>▪ Enkelt</li><li>▪ Hjem skjerm basert på rolle, f.eks snøploger viser oversiktsbilde.</li></ul>
09:25	Deployment	II	<ul style="list-style-type: none"><li>▪ SMS service, token service (e-mail)</li></ul>
09:30	Kravspesifikasjon	II	<ul style="list-style-type: none"><li>▪ Tilbakemelding, OK</li></ul>
09:35	Forprosjektrapport	II	<ul style="list-style-type: none"><li>▪ Avento, har ikke fått, tilbakemeldinger på Slack.</li></ul>
09:36	Plan videre	II	<ul style="list-style-type: none"><li>▪ Springboot security</li><li>▪ Flutter</li><li>▪ Gitlab</li></ul>

## Handlingselementer

- Send forprosjektrapport til Anders, Avento. Marius Nakstad

# 2020-02-25 møtenotater

## Dato

25.feb.2020

## Deltakere

- Henrik Bjørnland
- Theodor Urtegård
- Marius Nakstad
- Anders Beite, Avento

## Mål

- Oppdatering etter endt sprint med kunde
- Neste sprint, starte kode

## Debattpunkter

Tidsbruk	Element	Hvem	Notater
	Database	Alle	<ul style="list-style-type: none"><li>• Cabin field</li><li>• Plover_id</li><li>• Roles</li><li>• changed by, changed time, m.m</li></ul>
5 min	GDPR		<ul style="list-style-type: none"><li>▪ Anonymisere data</li><li>▪ Beholde plover data, men ikke bruker/person data.</li></ul>
10 min	Postman		<ul style="list-style-type: none"><li>▪ Authentication token i gruppe for modularitet</li></ul>
5 min	Oauth		<ul style="list-style-type: none"><li>▪ Standard i bransjen</li><li>▪ mest tid blir ofte brukt her</li></ul>
5 min	Begynnelse		<ul style="list-style-type: none"><li>▪ Opstart av koding for neste sprint</li></ul>

## Handlingselementer



# 2020-03-13 møtenotater

## Dato

13.mar.2020

## Deltakere

- Henrik
- Theodor
- Marius
- Anders Beite

## Mål

- Covid-19, finne ut hvordan vi skal avholde møter fremover.
- Informere kunde at vi kommer til å fokusere på en eksamen i et annet fag i en ukes tid.
- Planlegge neste møte
- Vise kunde og få tilbakemelding på gjennomgang av hva som er blitt gjort fra sist møte.

## Debattpunkter

Tidsbruk	Element	Hvem	Notater
5 min	Covid-19	Alle	Enighet i å bruke Discord til å avholde møter fremover da de har gratis skjermvisning.
10 min	Eksamen	Alle	Eksamen i fokus en uke fram i tid, lite planlagt å gjøre på applikasjonen i denne perioden.
5 min	Neste møte	Alle	Møte avtalt til 24. mars
10 min	Fremgang	Alle	Viste kunde hva vi hadde gjort, han hadde ingen innvendinger.

## Handlingselementer



# 2020-03-24 møtenotater

## Dato

24.mar.2020

## Deltakere

- Henrik Bjørnland
- Theodor
- Marius
- Kjell Inge
- Anders Beite

## Mål

- Oppdatere kunde på hvor vi er i løpet.
- Spør kunde om tips og endringer på det vi har vist.
- Avtale neste møte, påske er lagt til opprinnelig møtedato denne må sikkert endres.

## Debattpunkter

Tidsbruk	Element	Hvem	Notater
15 min	Oppdatering	Alle	<ul style="list-style-type: none"><li>• Database må gjøres større, litt for begrenset</li><li>• Få opp en server til å teste backend</li></ul>
5 min	Neste møte	Alle	<ul style="list-style-type: none"><li>▪ Møtet flyttet en uke, til 14 april isteden for 7 april som planen tilsa.</li></ul>

## Handlingselementer





# 2020-04-14 møtenotater

## Dato

14.apr.2020

## Deltakere

- Henrik Bjørnland
- Theodor
- Marius
- Kjell Inge
- Anders Beite

## Mål

- Oppdatere kunde på fremgang, få tilbakemelding.

## Debattpunkter

Tidsbruk	Element	Hvem	Notater
20 min	Oppdatering	Alle	<ul style="list-style-type: none"><li>• Viste kunde fremgang.</li><li>• Diskuterte hva vi viste, samt planla neste sprint.</li><li>• Problemer med å få server opp, ssh fra gitlab runner går ikke, må finne alternativ løsning.</li></ul>

## Handlingselementer



# 2020-05-5 møtenotater

## Dato

05.mai.2020

## Deltakere

- Henrik Bjørnland
- Theodor
- Marius
- Kjell Inge
- Anders Beite

## Mål

- Siste møte med kunde før innlevering av bachelor.
- Informere kunde om at vi fokuserer på rapporten fra denne datoen
- Informere kunde om prosjektets ferdighet og hva som står igjen å gjøre.

## Debattpunkter

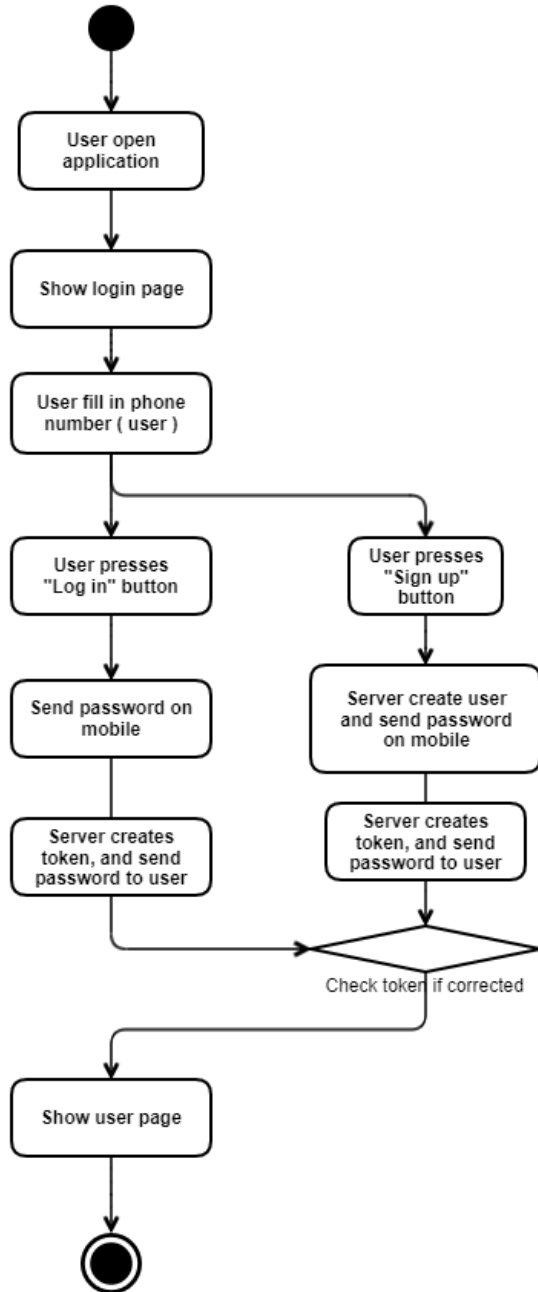
Tidsbruk	Element	Hvem	Notater
20 min	Gjennomgang av prosjekt	Alle	<ul style="list-style-type: none"><li>• Informerte om at dette var siste møte.</li><li>• Viste kunde slutt resultat</li></ul>

## Handlingselementer

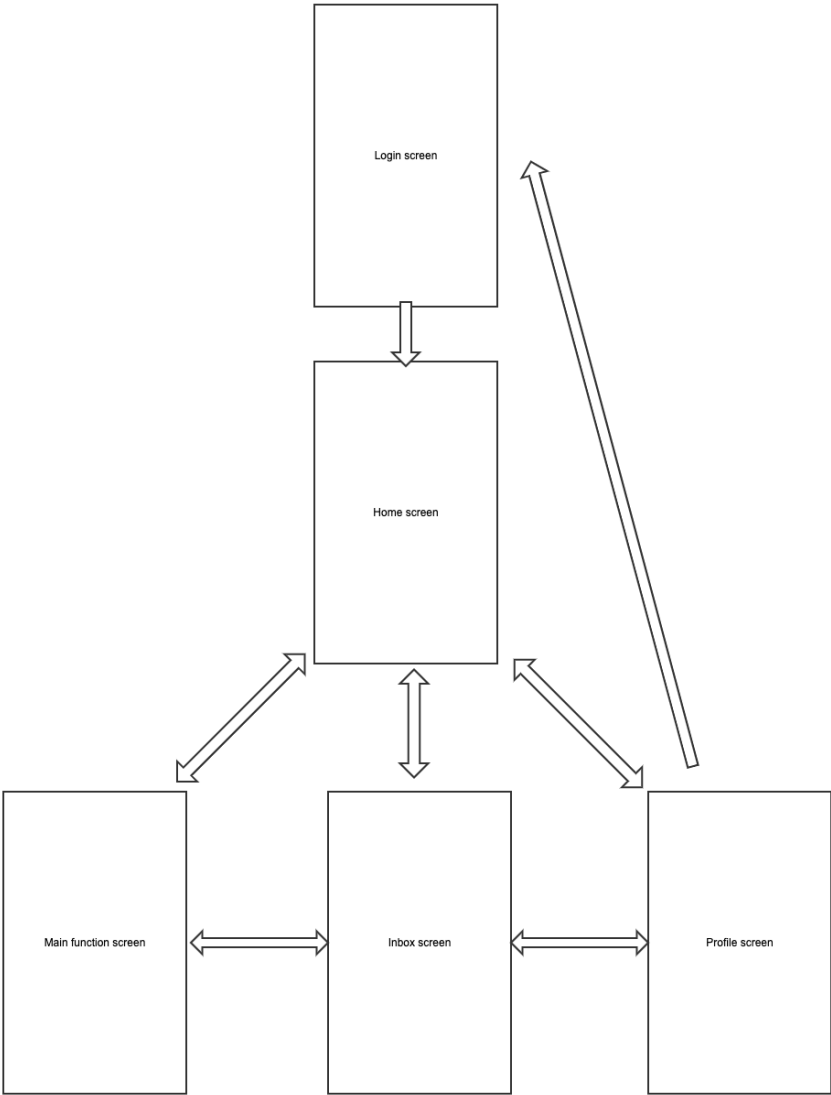


## **APPENDIX 3**

# Activity diagram



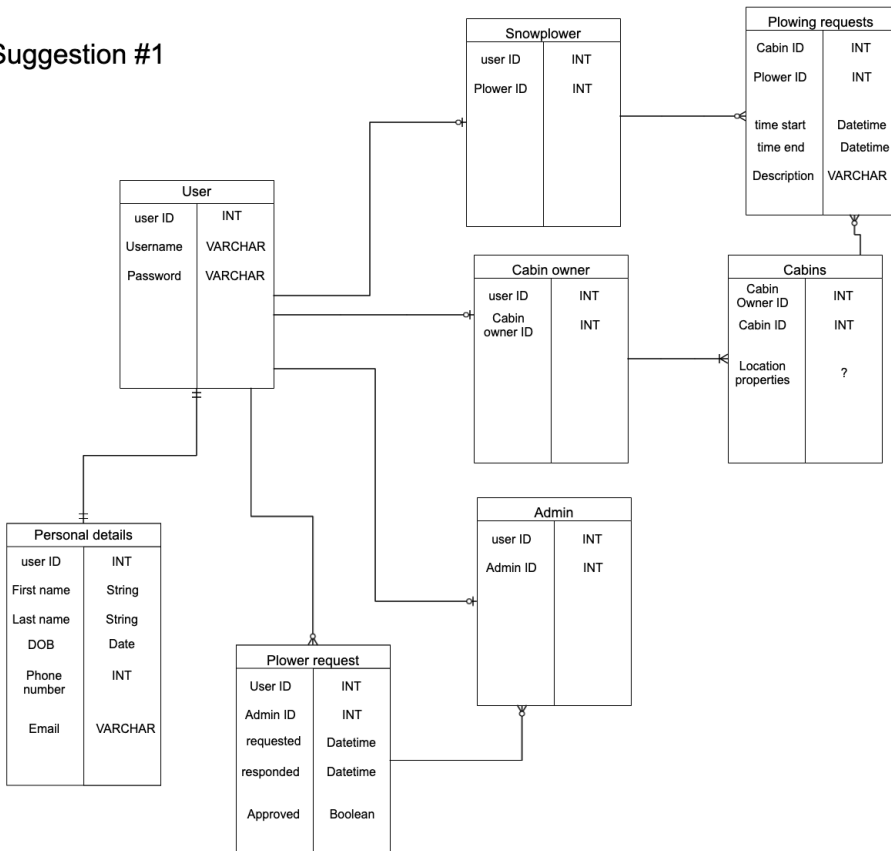
# Application state diagram



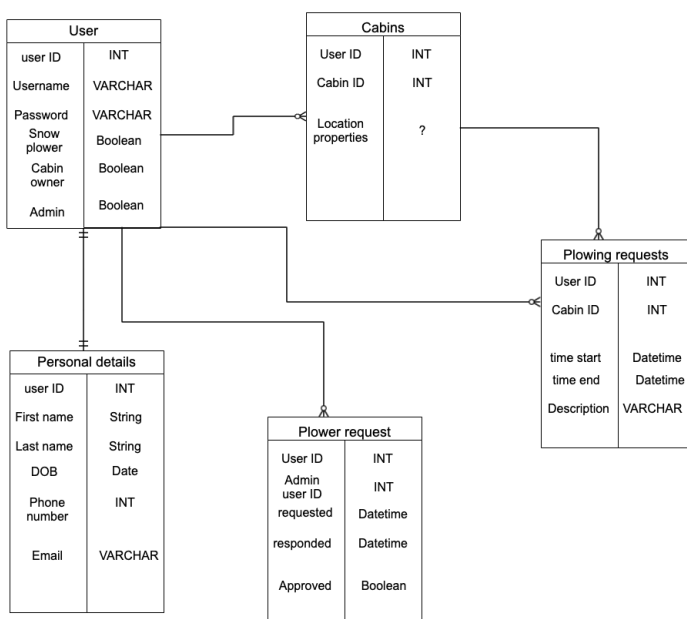
# Database ERD

## Database ERD

### Suggestion #1



### Suggestion #2







# 2020-05-5 møtenotater

## Dato

05.mai.2020

## Deltakere

- Henrik Bjørnland
- Theodor
- Marius
- Kjell Inge
- Anders Beite

## Mål

- Siste møte med kunde før innlevering av bachelor.
- Informere kunde om at vi fokuserer på rapporten fra denne datoen
- Informere kunde om prosjektets ferdighet og hva som står igjen å gjøre.

## Debattpunkter

Tidsbruk	Element	Hvem	Notater
20 min	Gjennomgang av prosjekt	Alle	<ul style="list-style-type: none"><li>• Informerte om at dette var siste møte.</li><li>• Viste kunde slutt resultat</li></ul>

## Handlingselementer



# Testing plan

## Introduction

This document contains the testing plan for our snow plower system. The document will outline the objectives, scope, strategy, environment and which features will and which ones will not be tested.

## Objectives and tasks

### Objectives

The objective with this document is to outline and define the way we want to perform testing of our snow plowing system, which includes a front end mobile application, and a back-end server and database. The reason for testing to begin with is to help prevent, discover and fix issues, bugs or problems as early as possible.

### Tasks

- Write automated tests
- Write Unit tests
- Set up CI and automatic testing environment
- Perform manual tests where needed
- Perform user testing
- Write test reports

## Scope

To help prevent issues and bugs, both the front and the back end of our system needs to be tested. The different components will have different scopes of testing based on feasibility and functionality.

The scope for the testing will be:

- Back end logic: Ensuring the back end server performs correctly.
- Back end database: Ensuring that the database stores and retrieves data correctly, and that the results from the database are as expected.
- Front end GUI: Ensuring that the front end GUI is working correctly and is understandable by the user.
- Front end logic: Ensuring that the front end application logic is working correctly.

## Testing references

The references for testing will be for our application to be able to reach the goals outlined by our requirement specification, UML diagrams and other documentation.

- [Requirement specification](#)
- [UML diagrams](#)

## Testing strategy

### Testing stages

### Testing frequency

## Testing environment

internal

(Own laptops, unit testing, testing before committing)

external

(GitLab CI, other tools)

## Features not to be tested

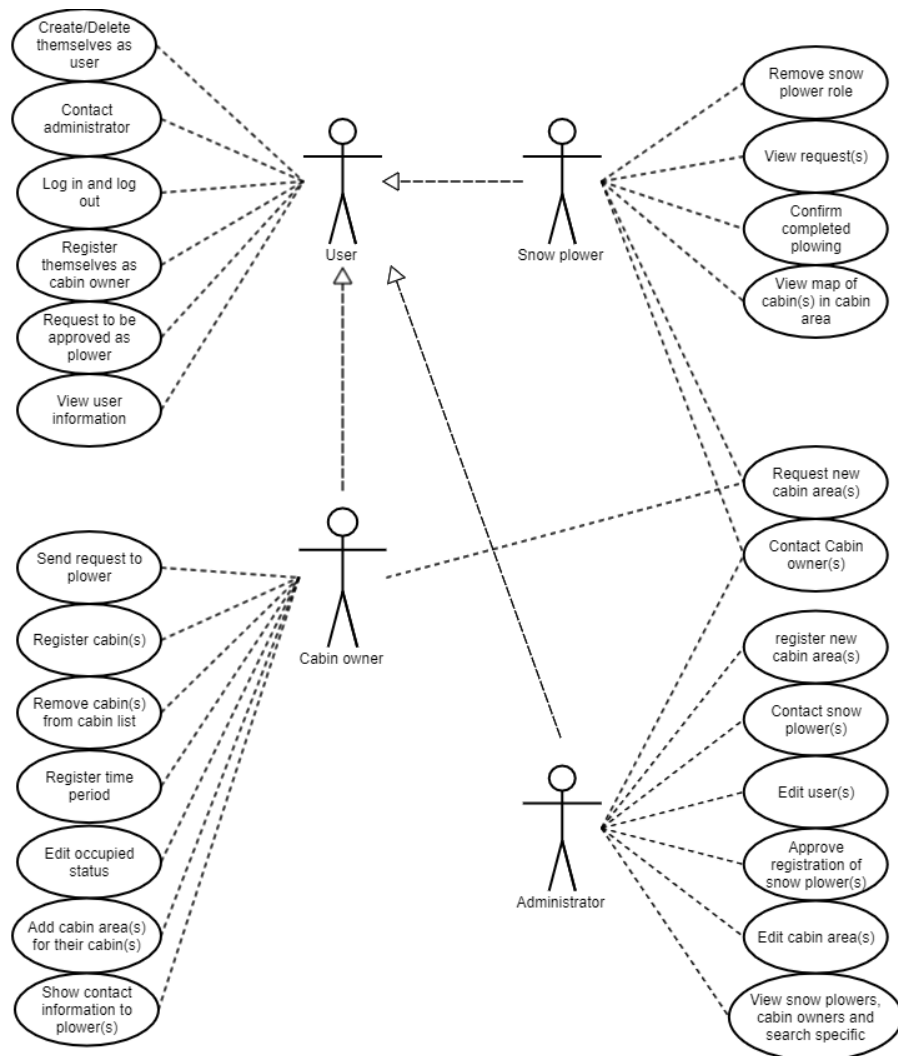
(external libraries or frameworks)

## Test cases

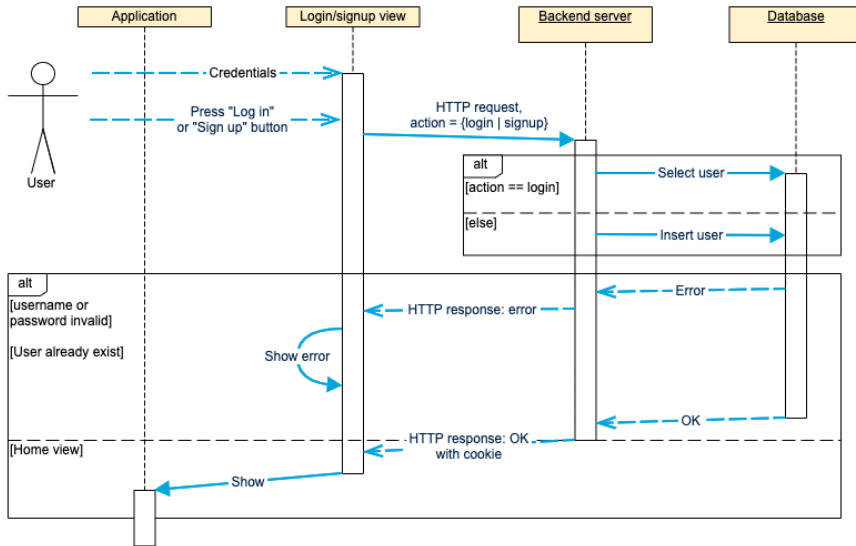
Test ID	Tested feature	Approach	Pass/Fail criteria	Description
1				
2				

# Use case diagram

Role	
User	New accounts are all created as user. User can have multiple roles
Cabin owner	
Snow plower	
Administrator	

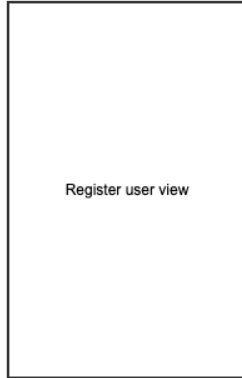
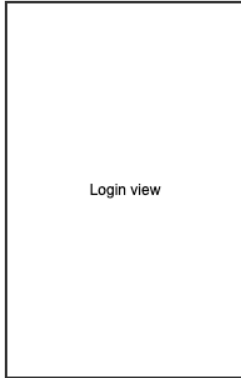


# User login diagram

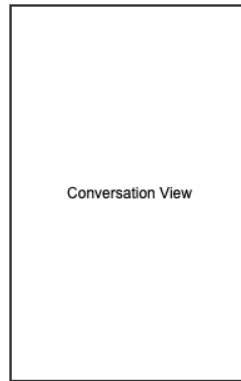
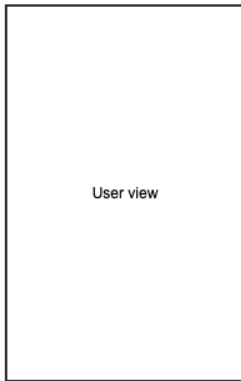


# Views

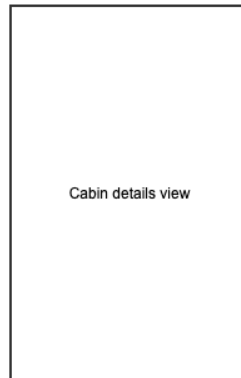
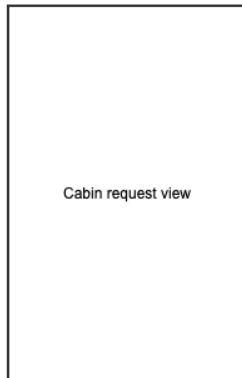
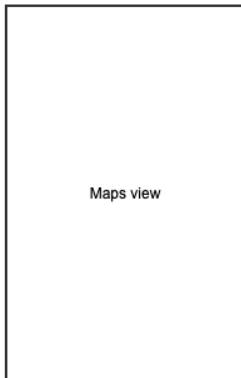
## General views



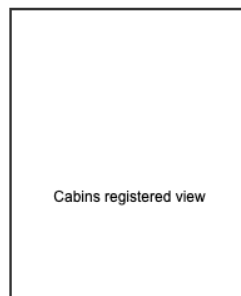
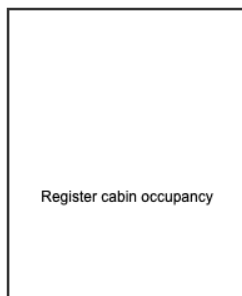
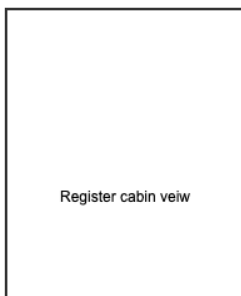
## Logged in views



## Snow plower views

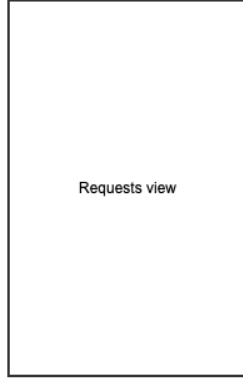
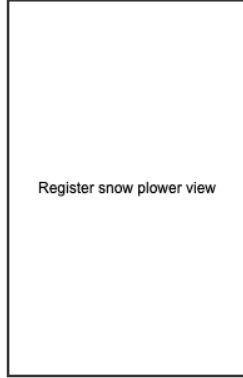
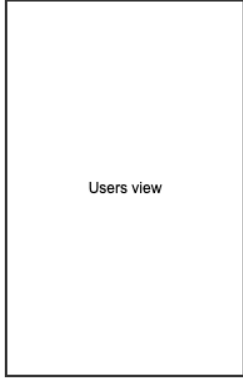


## Cabin owner views





**Admin views**

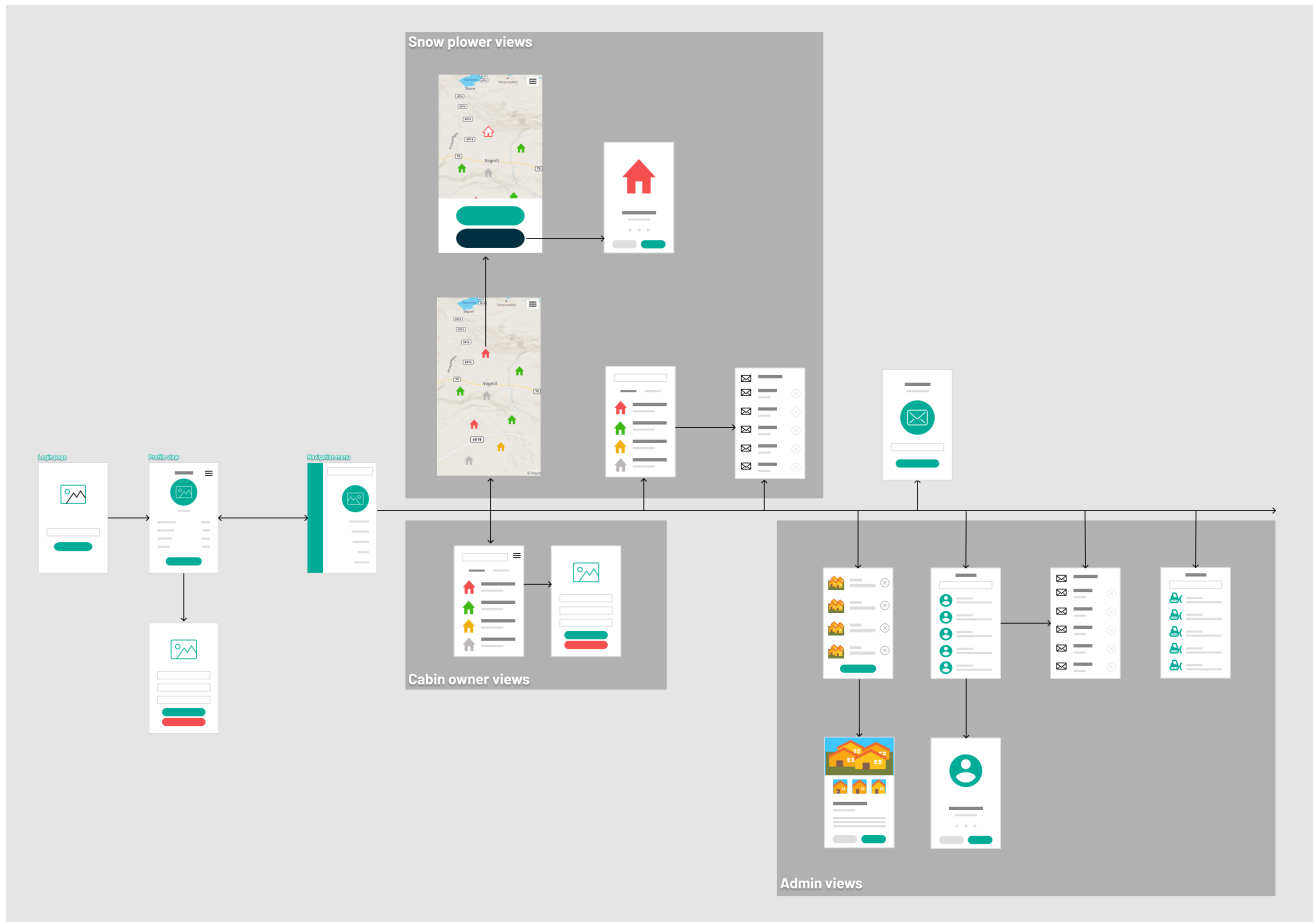




# Wireframes

A file list of the wireframes

<https://www.figma.com/file/pS2logyJszKCGH8sx4gBDZ/Sn%C3%B8jobb-wireframes?node-id=0%3A1>



## **APPENDIX 4**

# Sprint retrospectives:

## 2020-02-11 Retrospective

Created by Marius Nakstad, last modified by Henrik Bjørnland on Feb 11, 2020

<b>Date</b>	📅 11 Feb 2020
<b>Participants</b>	@Marius Nakstad @Henrik Bjørnland @Theodor Urtegård

### Retrospective

#### What did we do well?

- We completed the tasks we set for the sprint
- We planned for the project
- Communication
- Preparation for future work
- Completed pre project report
- Slack up and running

#### What should we have done better?

- Better sleeping schedules

### Actions

- ✓ @Marius Nakstad @Henrik Bjørnland @Theodor Urtegård Focus on the next sprint and keep up the good work
- ✓ @Marius Nakstad @Henrik Bjørnland @Theodor Urtegård Get some good nights sleep and try to get up and be at University at around 8

## 2020-02-25 Retrospective

Created by Henrik Bjørnland, last modified on Feb 25, 2020

<b>Date</b>	📅 25 Feb 2020
<b>Participants</b>	@ Henrik Bjørnland @ Marius Nakstad @ Theodor Urtegård

### Retrospective

#### What did we do well?

- We completed the sprint with all but one issue remaining.
- Estimation of story points was good.
- Completed the ground work to start programming the application.

#### What should we have done better?

- Better turn up, throughout the sprint.
- Should have written more on the report.

### Actions

- All, start programming

## 2020-03-18 Retrospective

Created by Marius Nakstad on Mar 18, 2020

<b>Date</b>	📅 18 Mar 2020
<b>Participants</b>	@ Marius Nakstad @ Theodor Urtegård @ Henrik Bjørnland

### Retrospective

#### What did we do well?

- Worked good
- Impressed customer
- Selected good tasks to work with

#### What should we have done better?

- Could get a little bit better with issue tracking
- Don't be afraid to add add new tasks in case we run out

# 2020-03-27 Retrospective

Created by Marius Nakstad on Mar 27, 2020

<b>Date</b>	📅 27 Mar 2020
<b>Participants</b>	@Marius Nakstad @Henrik Bjørnland @Theodor Urtegård

## Retrospective

### What did we do well?

- Exam
- Got work done even with time spent on preping for exam
- Picked good tasks to work with
- Worked well from home considering the COVID-19 situation

### What should we have done better?

- Completed the sprint on time
- Better sleep schedules

## Actions

- @Marius Nakstad @Theodor Urtegård @Henrik Bjørnland Stay healthy, don't get infected
- @Marius Nakstad @Theodor Urtegård @Henrik Bjørnland Go to bed at correct time

# 2020-04-14 Retrospective

Created by Marius Nakstad 13 minutes ago

<b>Date</b>	📅 14 Apr 2020
<b>Participants</b>	@Marius Nakstad

## Retrospective

### What did we do well?

- Worked well on the project
- Managed to keep meeting despite easter and corona

### What should we have done better?

- Try to keep things more constant even with changed workflow due to corona

## Actions

-

# Cumulative flow diagram:

27/Jan/20 to 20/May/20 (All Time) Refine report [How to read this chart](#)

