



Norwegian University of
Science and Technology

Manual Follow-up of Sheep on Free Range Pasture

Thomas Alm

Alexander Olav Gård

Master of Science in Informatics

Submission date: May 2018

Supervisor: Svein-Olaf Hvasshovd, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Norwegian farmers are required by law to perform an observation trip at least once a week while their sheep roam on a free-range pasture. During these trips, the farmer has to walk through the pasture and register his findings, which can range from wounded and dead sheep to predators. Today, most farmers write down their findings using pen and paper. When the season ends, farmers aggregate their findings and send a report to the government. In cases where farmers collaborate together, they submit a shared report.

In recent years, the supervisor of the project has experienced first-hand the observation trips performed by the farmers. Through his experience, it became apparent that the current pen and paper method is cumbersome and could be modernized. He envisioned a solution, where the farmers could register their findings through a mobile device. This laid the foundation for this research project, which will aim to ascertain the viability of the solution. The research was conducted through the development of a prototype application named Pecora. Furthermore, the utilization of modern technologies for developing mobile applications was of interest. Thus this project will also investigate what benefits these modern technologies can provide.

Based on the feedback received from interviews and user tests, we believe that the concept of Pecora is a viable solution. Compared to what is used today, it would also be a significant improvement in terms of efficiency and functionality. Through the creation of Pecora, we have observed that there are several benefits to using modern technologies for developing applications. Although there are some disadvantages from utilizing these technologies, it is easier to write better code in the long-term; therefore using them is a worthwhile investment.

Sammendrag

Norske bønder er påkrevd av loven til å utføre en observasjonstur minst en gang i uken så lenge sauene deres er på et frittgående beite. Under disse turene må bonden gå gjennom beitet og registrere sine funn, som kan variere fra døde og skadde sauer til rovdyr. I dag skriver de fleste bønder ned sine observasjoner ved bruk av penn og papir. Når sesongen avsluttes, samler bonden sine observasjoner og sender en rapport til det offentlige. I tilfeller der bønder samarbeider, sender de en felles rapport.

Over de siste årene har veilederen vår fått førstehåndserfaring med slike observasjonsturer. Gjennom hans erfaring ble det tydelig at dagens penn- og papirmetode kunne moderniseres. Han så for seg en løsning der bøndene kunne registrere sine observasjoner gjennom en mobil enhet. Dette la grunnlaget for dette forskningsprosjektet, som hadde som mål å fastslå levedyktigheten for denne løsningen. Forskningen ble utført gjennom utviklingen av en prototype applikasjon, navngitt Pecora. Forøvrig så var bruken av moderne teknologier for utvikling av mobilapplikasjoner av interesse. Prosjektet vil derfor også etterforske hvilke fordeler disse teknologiene kan gi.

Basert på tilbakemeldingene fra intervjuer og brukertester, tror vi at konseptet til Pecora er en levedyktig løsning. Sammenlignet med det som blir brukt i dag, så ville den også vært en stor forbedring med tanke på effektivitet og funksjonalitet. Gjennom utviklingen av Pecora har vi observert at det er flere fordeler med moderne teknologier for utvikling av applikasjoner. Selv om det er noen ulemper ved bruken av disse teknologiene, er det lettere å skrive bedre kode på lang sikt, og er derfor en verdig investering.

Prologue

This master's thesis was developed during the course IT3901 - Informatics Postgraduate Thesis: Software. It was delivered to the Department of Computer Science, which belongs to the Faculty of Information Technology and Electrical Engineering, at the Norwegian University of Technology and Science.

We would like to express our gratitude towards our supervisor Prof. Svein-Olaf Hvasshovd for his continuous guidance, feedback, and support throughout the project.

Alexander Gård,

Thomas Alm

Table of Contents

- List of Figures** **xi**

- List of Tables** **xv**

- Listings** **xix**

- Acronyms** **xxi**

- I Introduction** **1**

- 1 Introduction** **3**
 - 1.1 Problem Definition 3
 - 1.2 Project Context 4
 - 1.3 Concept 4
 - 1.4 Stakeholders 6
 - 1.5 Thesis Structure 7

- 2 Research** **9**
 - 2.1 Research Phases 9
 - 2.2 Research Questions 10
 - 2.3 Research Strategy 10

- 3 Preliminary Study** **11**
 - 3.1 Existing Solutions 11
 - 3.2 Previous Work 15
 - 3.3 Technologies 16

3.4	Software Development Methodologies	24
II	Process & Requirements	27
4	Project Development	29
4.1	Process	29
4.2	Planning	33
5	Requirements	37
5.1	Requirement Elicitation	37
5.2	Functional Requirements	39
5.3	Quality Attributes	53
III	Application	61
6	Walkthrough of Pecora	63
6.1	Log in	63
6.2	Navigation Drawer	64
6.3	Groups	66
6.4	Map Data	70
6.5	Trips	74
6.6	Timeline	84
6.7	Reports	86
6.8	Profile	87
6.9	Settings	88
6.10	Help	90
7	A Brief Introduction To Android	91
7.1	Activity	91
7.2	Fragment	91
7.3	Dialog	92
7.4	Android Lifecycle	93
7.5	Gradle	93
7.6	The Manifest File	94

8	Architecture	95
8.1	System Architecture	95
8.2	Application Architecture	96
9	Implementation	111
9.1	Rxjava 2	111
9.2	Model View View-Model	112
9.3	Libraries	118
9.4	Quality Attribute Tactics	122
10	Testing	127
10.1	Test Strategy	127
10.2	Unit Testing	129
10.3	GUI Testing	131
10.4	System Testing	132
11	Code Documentation	137
IV	Interviews, Future Work & Conclusion	139
12	Interviews	141
12.1	Farmer Interview	141
12.2	County Governor Interview	144
13	Future Work	147
14	Conclusion	153
V	Bibliography & Appendices	157
	Bibliography	159
A	Useful Tools	163
A.1	Android Studio	163
A.2	Android SDK	163

A.3	Git	163
A.4	GitHub	164
A.5	Trello	164
A.6	Google Drive	164
A.7	ShareLaTeX	164
B	Interview	165
B.1	Transcript	165
B.2	Interview Plan	173
C	County Governor Report	181
D	Usability Testing	185
D.1	Personal Questions	185
D.2	Introduction	185
D.3	Tasks	186
E	Sketches	191
F	System Test Cases	197
F.1	Login	197
F.2	Create User	199
F.3	Navigation Drawer	201
F.4	Trips	202
F.5	Trip	205
F.6	Map Data	216
F.7	Timeline	219
F.8	Groups	223
F.9	Profile	228
F.10	Settings	228
F.11	Help	231

List of Figures

1.1	Initial concept	5
1.2	New concept	6
3.1	Screenshot of Norgeskartet	12
3.2	Screenshot of Skandobs	13
3.3	Screenshot of Viltappen - List of observations	14
3.4	Screenshot of Viltappen - Observation details	14
3.5	Sketch of the Pecora application from the supervisor	16
3.6	The MVC architecture	19
3.7	The MVP architecture	20
3.8	The MVVM architecture	20
3.9	The Scrum process	25
3.10	Representation of a Kanban board	25
3.11	Waterfall model	26
4.1	Trello idea column	30
4.2	Trello todo column	30
4.3	Trello doing column	31
4.4	Trello done column	31
4.5	Trello color tag column	32
4.6	Card flow	32
4.7	Gantt diagram	35
5.1	Sketch from supervisor	39
5.2	Sketch of timeline component	39
5.3	Use case diagram of log in	40

5.4	Use case diagram of map data	41
5.5	Use case diagram of group	42
5.6	Use case diagram of trip part 1	43
5.7	Use case diagram of trip part 2	44
5.8	Use case diagram of report	45
5.9	Use case diagram of timeline	46
5.10	Example quality attribute scenario	54
5.11	Usability tactics	56
5.12	Availability tactics	58
5.13	Modifiability tactics	60
6.1	Sign in screen	64
6.2	Create user screen	64
6.3	Navigation drawer	65
6.4	The group list	67
6.5	Creating a new group	67
6.6	Specific group overview	68
6.7	User details	68
6.8	Delete group dialog	69
6.9	Send invitation dialog	69
6.10	Invitation list	70
6.11	Invitation details dialog	70
6.12	List of downloaded map areas	71
6.13	Map data delete mode	71
6.14	The undo pop-up menu	72
6.15	Map area download	73
6.16	Download dialog	73
6.17	Downloaded map area	74
6.18	Edit name dialog	74
6.19	Initial trip component	76
6.20	Group trips	76
6.21	Delete state	77
6.22	Share state	77

6.23	Start trip dialog	79
6.24	General Tab	79
6.25	Scenario Tab 1	80
6.26	Scenario Tab 2	80
6.27	Camera tab	81
6.28	Camera tab in delete state	81
6.29	Observation highlight	82
6.30	Closest observation button	83
6.31	Next observation button	83
6.32	Open observation button	83
6.33	Link observation button	83
6.34	Trip toolbar	84
6.35	Trip details dialog	84
6.36	The timeline component	85
6.37	Reports Component	86
6.38	Reports share mode	86
6.39	Spreadsheet extract	87
6.40	Profile	88
6.41	Change username dialog	88
6.42	Settings	89
6.43	Select distance interval dialog	89
6.44	Help	90
7.1	Relations between activities and fragments	92
7.2	An example of two Android dialogs	93
8.1	Deployment diagram	96
8.2	Application package diagram	98
8.3	GUI test package diagram	99
8.4	Unit test package diagram	99
8.5	Resource structure in Pecora	100
8.6	MVVM structure	103
8.7	EventHandler structure	104

8.8	Observable function structure	105
8.9	Data binding	106
8.10	Overview of Firebase managers	107
8.11	Dependency injection in Pecora	108
8.12	Multiple components listening for location requests	109
9.1	CircularImageView example	119
9.2	Logger example	120
9.3	Swipe to delete button	120
9.4	Number picker	121
9.5	Color picker	121
9.6	Cancel tactic	123
9.7	Undo tactic	123
9.8	Pause tactic	123
9.9	Aggregate tactic	124
10.1	Testing Levels	128
12.1	Observation registration - BeiteSnap	143
13.1	Multiple scenarios sketch	148
13.2	Suggested form - page 1	151
13.3	Suggest form - page 2	152
C.1	County governor form - page 1	182
C.2	County governor form - page 2	183

List of Tables

5.1	The boilerplates used to construct requirements	46
5.2	The functional requirements of Pecora.	47
5.3	U1 - Using the system efficiently	54
5.4	U2 - Learning the system	55
5.5	U3 - Minimize accidental deletions	55
5.6	U4 - Feedback for interactions	55
5.7	A1 - Persist data when no network connection	57
5.8	A2 - Reconnect to server when no network connection	57
5.9	A3 - Synchronize with server when reconnecting to server	57
5.10	A4 - Send report when application crashes	57
5.11	M1 - Create new observation scenario	58
5.12	M2 - Modify observation scenario	59
5.13	M3 - Change report format	59
10.1	Example Test Case	133
10.2	SUS-form	135
D.1	SUS form values	189
D.2	SUS form	189
F.1	STC 1.01	197
F.2	STC 1.02	198
F.3	STC 1.03	198
F.4	STC 1.04	198
F.5	STC 2.01	199
F.6	STC 2.02	199

F.7	STC 2.03	200
F.8	STC 2.04	200
F.9	STC 3.01	201
F.10	STC 3.02	201
F.11	STC 3.03	201
F.12	STC 3.04	202
F.13	STC 4.01	202
F.14	STC 4.02	202
F.15	STC 4.03	203
F.16	STC 4.04	203
F.17	STC 4.05	203
F.18	STC 4.06	204
F.19	STC 4.07	204
F.20	STC 4.08	205
F.21	STC 5.01	205
F.22	STC 5.02	206
F.23	STC 5.03	206
F.24	STC 5.04-5.06	207
F.25	STC 5.07	207
F.26	STC 5.08-5.15	209
F.27	STC 5.16	210
F.28	STC 5.17	210
F.29	STC 5.18	211
F.30	STC 5.19	211
F.31	STC 5.20	212
F.32	STC 5.21-5.22	212
F.33	STC 5.23-5.24	213
F.34	STC 5.25	213
F.35	STC 5.26	214
F.36	STC 5.27	214
F.37	STC 5.28	215
F.38	STC 5.29	215

F.39	STC 6.01	216
F.40	STC 6.02	216
F.41	STC 6.03	217
F.42	STC 6.04	217
F.43	STC 6.05-6.09	218
F.44	STC 6.10	219
F.45	STC 7.01	219
F.46	STC 7.02	220
F.47	STC 7.03	220
F.48	STC 7.04-7.07	221
F.49	STC 7.08	221
F.50	STC 7.09	222
F.51	STC 7.10	222
F.52	STC 7.11	223
F.53	STC 8.01	223
F.54	STC 8.02	224
F.55	STC 8.03	224
F.56	STC 8.04	224
F.57	STC 8.05	225
F.58	STC 8.06	225
F.59	STC 8.07	226
F.60	STC 8.08	226
F.61	STC 8.09	226
F.62	STC 8.10	227
F.63	STC 8.11	227
F.64	STC 9.01	228
F.65	STC 9.02	228
F.66	STC 10.01	229
F.67	STC 10.02	229
F.68	STC 10.03	229
F.69	STC 10.04	230
F.70	STC 10.05	230

F.71 STC 11.01 231

Listings

9.1	Simple RxJava example	112
9.2	Listening for changes to user data	113
9.3	ProfileFragment class definition	113
9.4	ProfileFragment setup functions	113
9.5	ProfileViewModel class definition	116
9.6	ProfileViewEvents	116
9.7	TextView being bound to the username	117
9.8	Username field	117
9.9	Initialization of data binding	118
9.10	Picasso example	119
10.1	Unit test naming convention	129
10.2	Unit test with precondition	129
10.3	Unit test example	130
10.4	GUI test example	132
11.1	Javadoc example	137

Acronyms

API Application Programming Interface

APK Application Package Kit

CVP Corporate Vice President

DAL Data Access Layer

GPS Global Positioning System

GUI Graphical User Interface

HTML HyperText Markup Language

IDE Integrated Development Environment

IT Information Technology

JVM Java Virtual Machine

MVC Model View Controller

MVP Model View Presenter

MVVM Model View View-Model

NL Natural Language

OS Operating System

PC Personal Computer

RQ Research Question

SDK Software Development Kit

STC System Test Case

SUS System Usability Scale

URL Uniform Resource Locator

UWP Universal Windows Platform

XML Extensible Markup Language

Part I

Introduction

Chapter 1

Introduction

This chapter gives a brief overview of the scope of the project. It introduces the project's context and definition, stakeholders, as well as an overview of the thesis' structure.

1.1 Problem Definition

Norwegian sheep farmers are required by law to have their sheep on pastures for at least 16 weeks each year [1]. During these weeks the farmers are also required to follow up on their sheep at least once per week, and more if the sheep are exposed to potential risks, such as predators, dangerous environment or weather conditions [1]. On these weekly follow-ups, the farmers are also required to register their findings [1]. Registering information is crucial if a sheep is wounded or killed, as this is necessary for receiving compensation from the government [1].

Today, most farmers use pen and paper to note down their findings. This approach can be inefficient, as it can take time and resources to write down all observed details. In some cases, these findings may be inaccurate, as time or weather does not always allow for detailed notes. Several farmers have large outdoor areas that they must cover, and it is not unusual for them to collaborate with neighboring farmers and share the responsibility. Sharing information between farmers can prove difficult, as it has to be done manually. This especially becomes a challenge at the end of the season, when they produce their reports, which requires all findings to be aggregated in a time-consuming process.

1.2 Project Context

In recent years our supervisor, Svein-Olaf Hvasshovd, has been involved with sheep farmers and helped them with their observation trips, gaining first-hand experience. During these trips, he realized that the pen and paper method had several disadvantages, and that farmers could greatly benefit from modernizing their current approach. His concept revolved around the usage of a mobile device, running an application tailored for registering observations. This idea was later turned into a master's project.

The project was well received when reviewed by a farmer. However, some shortcomings were uncovered, such as the lack of functionality for taking pictures during the observation trips. There was also room for improvement regarding the GUI design. Svein then decided that the master's project could be further improved, and offered a revised version, which became the basis for this project. It should be noted that the influence of the previous project was limited to the overarching concept; no other artifacts from the previous project has been used.

1.3 Concept

The initial concept for this project was created by Svein-Olaf Hvasshovd, and revolved around two roles: the shepherd, and the farmer. The shepherds would have an application on their smartphones, which they would use to register observations during their trips, and then transfer it to the farmer's PC. The farmer would then be able to inspect the data through a web application, and eventually generate a report when needed. In cases where the farmer does not have a shepherd employed, he would have to perform both roles by himself. The initial concept is illustrated in Figure 1.1.

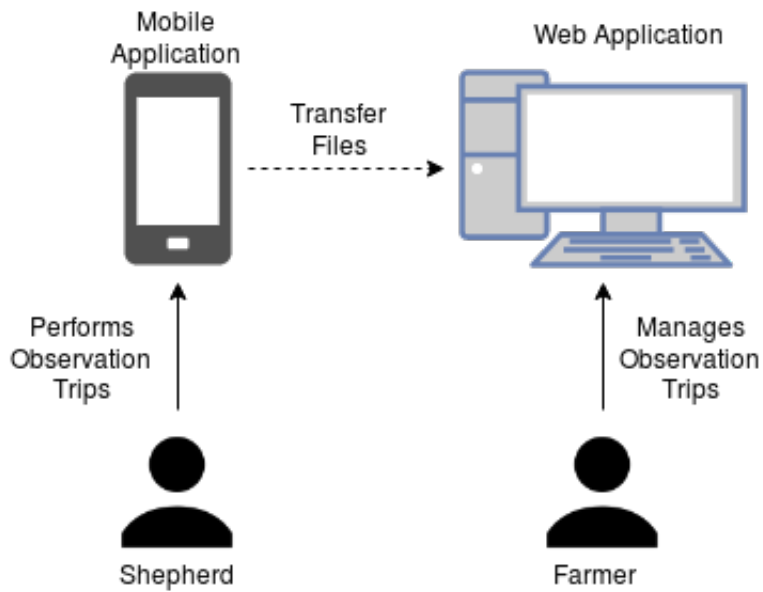


Figure 1.1: Initial concept

The concept would provide farmers with an effective way of managing their trips, but it does not have any functionality for collaborating with other farmers, which is a significant disadvantage. Another disadvantage is that if the farmer does not have a shepherd employed he will have to interact with two applications in order to complete the process. For example, if the farmer is walking a trip and needs to plan out where to go next based on earlier trips, then he would need both the mobile application and the web application.

To solve these problems, we developed a new concept based on the initial one. The new concept merges the functionality from the web app on the farmer's PC with the app on the shepherd's smartphone. The result is an app which can perform all the functionality required by both the shepherds and farmers, which could increase usability and decrease the learning curve for using the system. Merging the functionality could be considered a disadvantage as it would give the shepherd access to functions he might not necessarily need. However, it might be important to have access in some situations, so it should not pose a significant problem.

The new concept revolves around the usage of groups and does not impose the farmer and shepherd roles upon its users, unlike the initial concept. Anyone with the application can join the group if they receive an invitation, but the groups will usually consist of collaborating farmers. Groups allow farmers to share their trips with other group members, which also shares the associated observations. It should be noted that the group aspect of the application is optional, and is not necessary to perform trips. The new concept is illustrated in Figure 1.2. The

new concept formed the foundation of our application, which was named "Pecora", the Italian word for "sheep".

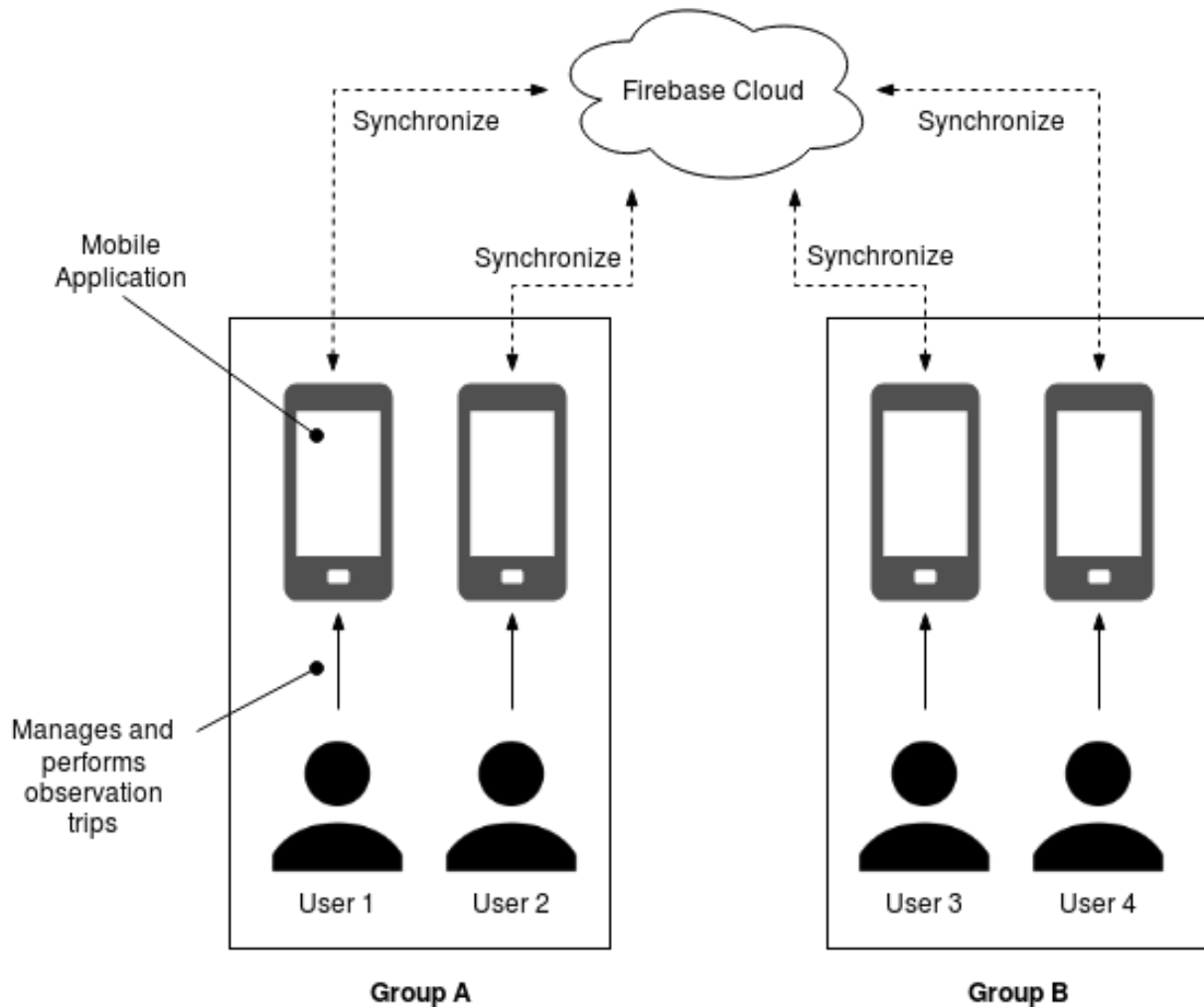


Figure 1.2: New concept

1.4 Stakeholders

The stakeholders in this project are people or entities that have an interest in either the finished product or the project process.

1. **Developers** - The developers will implement Pecora according to the requirements of the project, write tests and documentation for the software, and write the thesis.
2. **Testers** - The testers will test the system during the system tests, and provide feedback to the developers. The testers should not be associated with the development of the software they are testing.
3. **Supervisor** - The supervisor creates the project requirement specification, oversees the

project development, and provides feedback during the development phase.

4. **Farmers** - The farmers use the application for registering and managing observation trips, collaborates with other farmers, and generates and shares seasonal reports.
5. **Government** - The government receives the seasonal reports from the farmers.

1.5 Thesis Structure

This thesis has been separated into five parts, based on the content of the underlying chapters.

The five parts are defined as follows:

1. **Part I** - Introduces the project and the research aspect. It also contains information about the preliminary study.
2. **Part II** - Explains the development process, and presents the functional requirements and quality attributes of Pecora.
3. **Part III** - Focuses on what has been developed during the project, and gives a thorough overview of Pecora and the underlying code base.
4. **Part IV** - Consists of information about the interview, the planned future work, and our conclusion regarding the research questions.
5. **Part V** - Contains the bibliography and the appendices.

Chapter 2

Research

In this chapter, the research aspect of the project will be presented. This includes the formulated research questions and the employed research strategy.

2.1 Research Phases

This project consists of three research phases, but this thesis will primarily concern itself with the second phase. The first phase is performed by our supervisor, where he will research the subject of manual follow-up of sheep on pasture through participation. The purpose of this phase is to gather information about the subject, such that the initial concept of Pecora can be developed. This phase was completed before the start of our project.

The second phase is performed by us and revolves around the development of Pecora. The second phase intends to ascertain the viability of the concept and acquire initial feedback from potential users. The feedback will be used to further refine Pecora before the third and last phase.

The third phase is performed by the supervisor, where he and several farmers will test Pecora thoroughly in a real environment. The farmers will also be interviewed by the supervisor, with the intent of gaining a deeper understanding of which requirements the farmers have. The knowledge gained from this phase will be used to develop Pecora further.

2.2 Research Questions

According to the project's specification, the primary purpose of Pecora, as mentioned earlier, is to ascertain the viability of the concept. However, the utilization of modern architectures, frameworks, and libraries to develop applications were also of interest. Based on this, the following research questions were formulated:

- **RQ1** - How viable is the concept of Pecora for managing and performing observation trips and creating reports for sheep farmers?
- **RQ2** - What benefits do modern architectures, frameworks, and libraries provide when developing mobile applications?

2.3 Research Strategy

Due to the nature of the research questions, we decided to use the Design and Creation research strategy. We also chose to employ interviews and observations to elicit feedback, which are both commonly used in this strategy.

The Design and Creation research strategy focuses on obtaining knowledge through the development of IT products. In this project, the main focus was to develop an application that was to be used in a new domain, to exhibit its technical viability [2]. Design and Creation is well suited for this type of project and was therefore a natural choice.

Chapter 3

Preliminary Study

This chapter describes the preliminary study done in this project. It discusses a set of related existing solutions, the previous work done on related subjects, and the considered technologies and software development methodologies for this project.

3.1 Existing Solutions

The initial research phase of the project looked at different applications that could offer farmers the capabilities that were required by the project to register observations at the necessary level of detail. Three apps with similar functions were found, which are described separately in the following sections.

3.1.1 Norgeskartet

Norgeskartet [3] is an application developed by Asplan Viak Internet AS. The application offers a traditional map view, where the user can navigate the map through touch gestures. As the user navigates, the application dynamically downloads and caches the viewed map tiles. In addition, the user is allowed to download and store map tiles. The application also has a paid and a free version, where the paid version requires a yearly subscription (40 NOK at the time of writing). The paid version expands on some free version features, such as allowing maps to be downloaded for an unlimited time instead of expiring after a certain set of days. The application also offers its users the possibility of recording their tracks, and register basic points of interests.

A screenshot of the application can be found in Figure 3.1.

Norgeskartet provides many features related to location and maps, but it does not have any functionality tailored for registering animals. It is possible to record tracks, but they are not associated with the findings, making them unorganized and difficult to use for our purpose. Also, there are no functionalities for creating reports or collaboration between farmers.

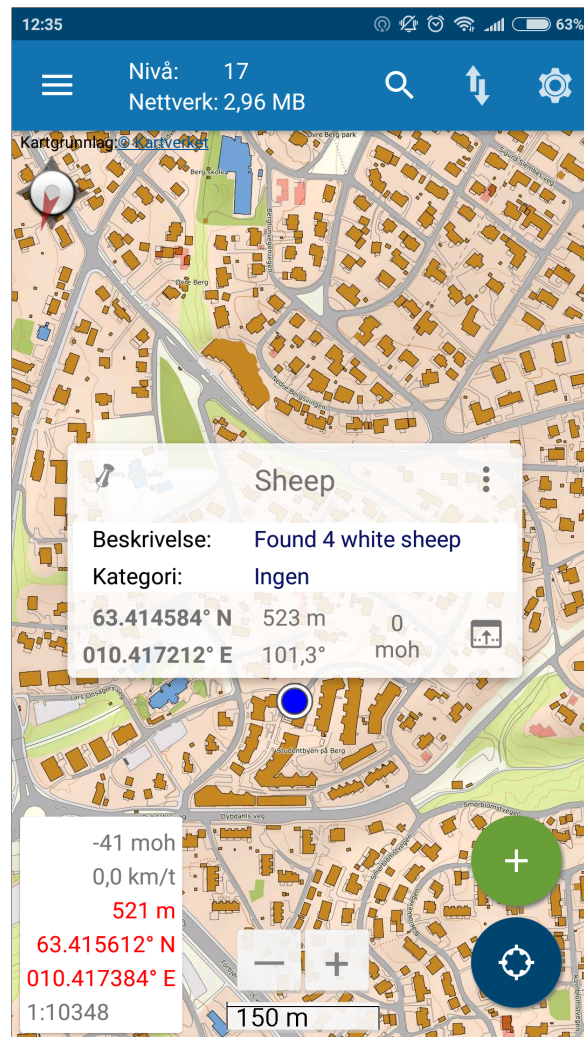


Figure 3.1: Screenshot of Norgeskartet

3.1.2 Skandobs Touch

Skandobs Touch [4] is an application for registering observations of wildlife, such as lynxes, wolverines, brown bears or wolves. The observations are performed by the users of the app and are available for anyone to see. The data from the observations are used by Rovdata in Norway, and Naturvårdsverket in Sweden, to increase the knowledge of how the different species live,

and how many of them there are in Scandinavia. Figure 3.2 depicts the Skandobs app.

Skandobs Touch shares many similar aspects to Pecora, in that both are focused on registering animals. However, Skandobs Touch is centered around dangerous wildlife, so it is not possible to register observations of sheep. All observations are also made public [5], which is problematic if privacy is a concern. Besides, there is no way of downloading an offline map or generate reports of the observations you have done.



Figure 3.2: Screenshot of Skandobs

3.1.3 Viltappen

Viltappen [6] is a Swedish application for giving guidance and knowledge when encountering wildlife. The app is mainly focused on providing information about the different species of animals that can be found in the Scandinavian nature, but it also has a function for saving

observations of animals, as shown in 3.3 and 3.4.

Viltappen could be used for registering observations of sheep, but it is quite generic and requires a lot of typing to precisely record what was observed. There is no map functionality in this app, either online or offline. Besides, functionality for generating a report is not present.

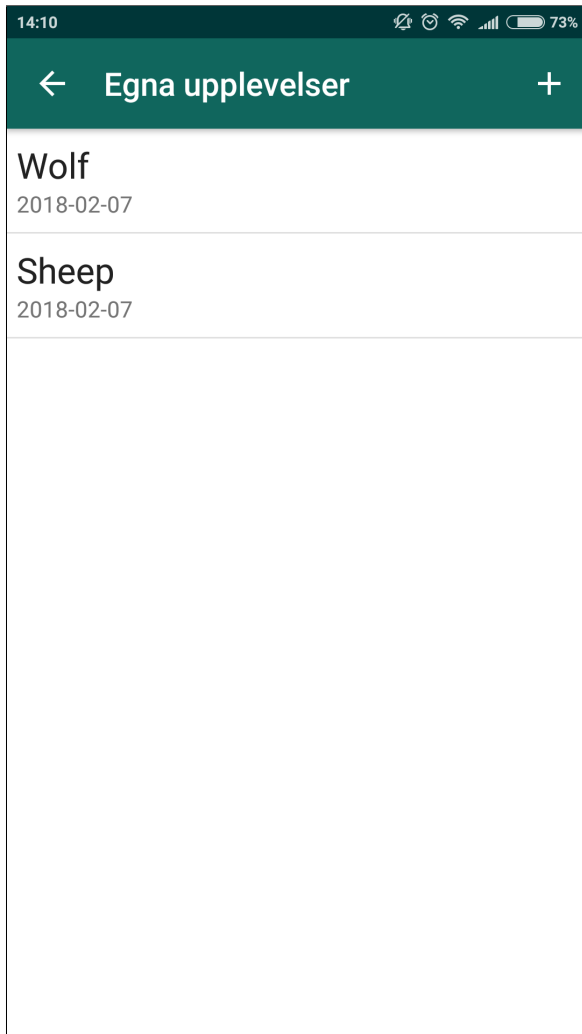


Figure 3.3: Screenshot of Viltappen - List of observations

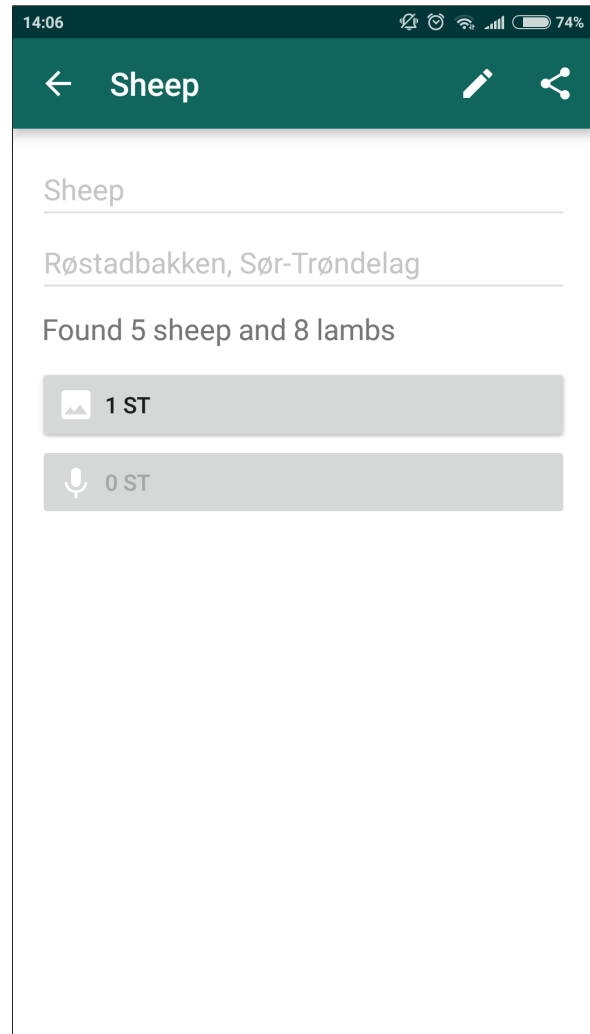


Figure 3.4: Screenshot of Viltappen - Observation details

3.1.4 Conclusion

After examining similar applications, we concluded that the apps did not fully offer the functionalities necessary for registering detailed observation trips. All the applications were missing at least one significant feature. While the existing applications could work to some extent, they would come with considerable limitations. Therefore, the need for an app that can fulfill all the requirements for registering observation trips and generating reports is necessary. We decided

that the application must have the following core functionalities:

- The user must be able to perform and save trips.
- The user must be able to register observations during his trips.
- The user must be able to download an offline map to use during his trips.
- The user must be able to generate reports based on the trips/observations he has done.

3.2 Previous Work

As previously mentioned in the introduction, our concept was based on an earlier master's project. Similar to the current project, the previous project developed an application for registering observation trip findings. However, through an interview with a farmer, it became clear that some features were missing, and some of the implemented functionalities should be further improved. This led to the continuation of the project, with the knowledge gained through the first version used to create a new requirements specification.

It should be noted that the newer version of Pecora is not influenced by or use any of the code or components used in the earlier version. Only the initial requirements specification, which was developed by the supervisor, has been affected by the previous work.

The requirements given to us by the supervisor consisted of several essential elements of what the app should contain, as well as sketches depicting the trip functionality. Figure 3.5 shows one of the sketches. The sketches show the different fields required from specific observations, as well as how to navigate around the application. The sketches were only meant as a rough idea of how the supervisor had imagined the app should function, and was not intended to be followed strictly.

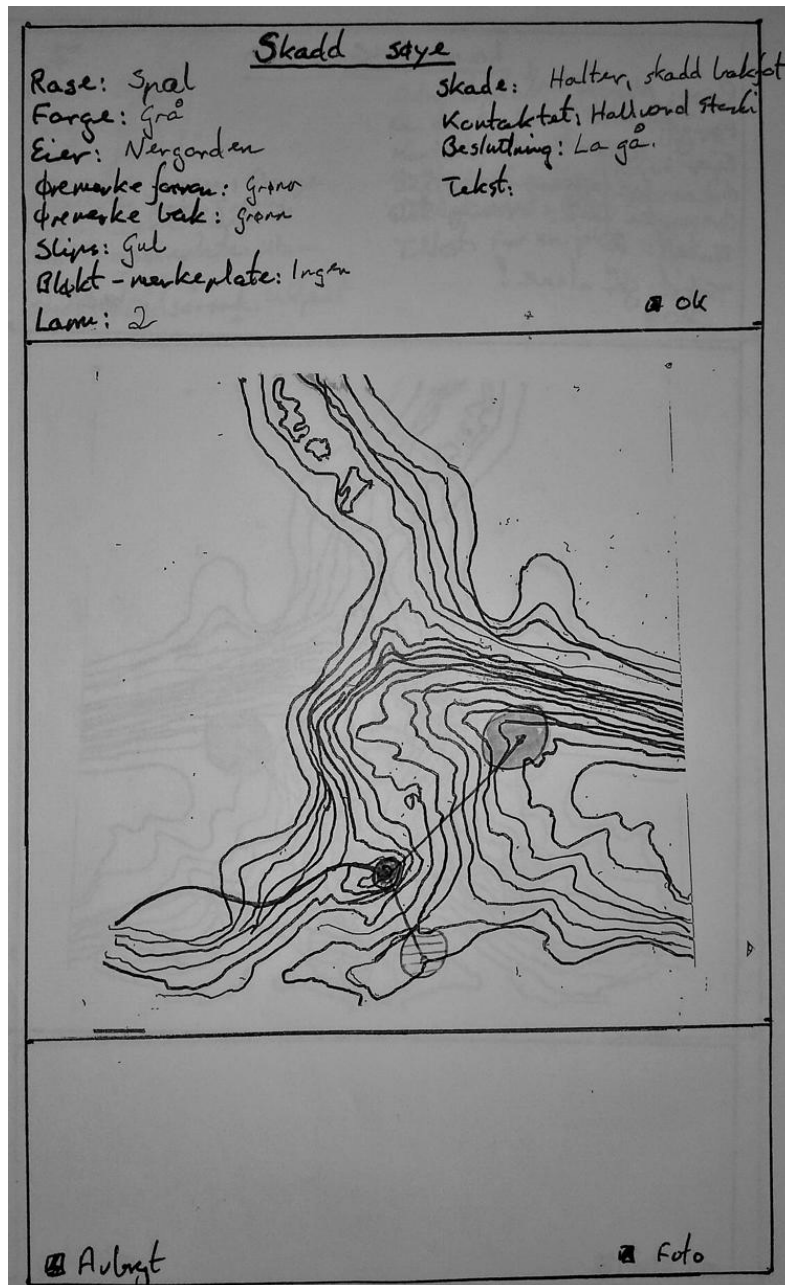


Figure 3.5: Sketch of the Pecora application from the supervisor

3.3 Technologies

There were not any specific technological requirements for Pecora, except for the platform. Pecora had to be available on a smartphone. Because of this, there were quite a few different technologies that could be used to achieve the functionalities needed from Pecora. The following sections will overview the various technologies that were considered and will discuss which technology that was chosen, and which advantages it has over related technologies.

3.3.1 Mobile platform

In today's smartphone market, there are mainly three major operating systems, Android, iOS, and Windows Phone. There are other operating systems as well, but compared to the three aforementioned systems, they can be considered insignificant due to their small userbase.

Pecora was intended to be a proof of concept solution, and it was therefore not a requirement to support multiple platforms. If this was not the case, then it would be necessary to compare different hybrid solutions such as React Native [7] or Cordova [8]. The reasoning for not choosing a hybrid solution for a proof of concept is the amount of overhead that is created by having to support multiple platforms. For us, it was instead more important to focus on rapid development for a single platform and to not be limited by the constraints that hybrid solutions impose.

3.3.1.1 Android

Android [9] is an open-source mobile operating system developed by Google. It is designed primarily for devices with a touchscreen, such as smartphones and tablets. Android devices claim 75.66% of the worldwide market share [10], which makes it the most popular operating system for mobile devices. Android supports programming applications in Java, Kotlin, and C++.

3.3.1.2 iOS

iOS [11] is a mobile operating system created and developed by Apple Inc. iOS powers Apple's mobile devices, such as iPhones, iPads, and iPod Touch. Apple claims 19.23% of the worldwide market share [10], which makes it the second most popular mobile operating system after Android. iOS supports programming applications in Objective-C/C++ and Swift.

3.3.1.3 Windows 10 Mobile

Windows 10 Mobile [12] is a mobile operating system developed by Microsoft. In October 2017, Joe Belfiore CVP at Microsoft, announced on Twitter that Microsoft had stopped actively developing Windows 10 Mobile, due to the lack of market shares and low interest among application developers [13]. Windows has a worldwide market share of 0.53%, which is quite

low compared to its competitors, Android and iOS. Windows supports creating applications through the UWP platform [14], which uses C#, C++, Visual Basic, and Javascript.

3.3.1.4 Chosen Mobile Platform

Firstly, we decided to pick a mobile platform with a large user base, which excluded Windows 10 Mobile. This left us the choice between iOS and Android. Secondly, we decided to go for a mobile platform that was familiar. Both of us had experience with Android from previous projects. Besides, we had Android devices readily available, whereas no iOS devices were in our possession. Lastly, we believed that we would be able to create the best possible product with Android. For these reasons we decided to use the Android mobile platform.

3.3.2 Architecture

There are several architectural patterns to choose from when developing applications. This section will give an overview of the considered patterns, and explain the reasoning behind our choice.

3.3.2.1 Model View Controller

MVC [15] is an architectural pattern that divides an application into three interconnected pieces. These three pieces can be further defined as follows:

- **Model** - The model is the data layer. It is responsible for business logic, and handling network and database events.
- **View** - The view is the visual representation of the model. Fetches data from the model when the controller tells it to update.
- **Controller** - The controller handles the user interactions, notifies when the view needs to update and alters the state of the model as needed.

Figure 3.6 is a visual representation of all the interconnected MVC pieces. It should also be noted that there are many different variants of the MVC pattern, such as whether the model is active or passive, but will not be further discussed here.

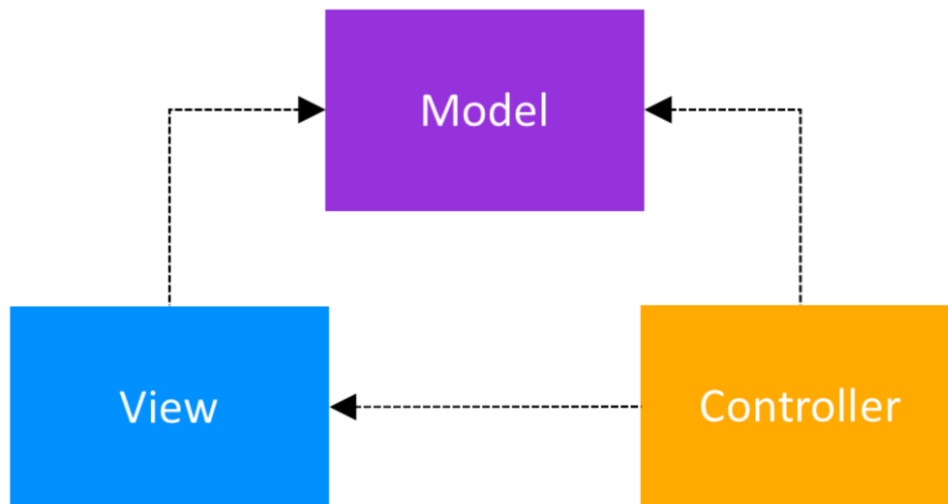


Figure 3.6: The MVC architecture

3.3.2.2 Model View Presenter

MVP [16] is an architectural pattern that is a derivation of MVC. MVP also consists of three interconnected parts, and can be defined as follows:

- **Model** - The model is the data layer. It is responsible for business logic, and handling network and database events.
- **View** - The user interface layer that displays data and notifies the presenter about user actions.
- **Presenter** - Retrieves data from the model. Manages the state of the view, handles the user interface logic, and reacts to user input notifications from the view.

The main difference between MVC and MVP is that MVP breaks the connection between the view and the model, and MVP creates only one class that handles the things related to the presentation of the view, which is the presenter class. Figure 3.7 is a visual representation of the MVP architecture. Since the view and the presenter is closely related, they need to hold a reference to each other, which is done through designated interfaces.

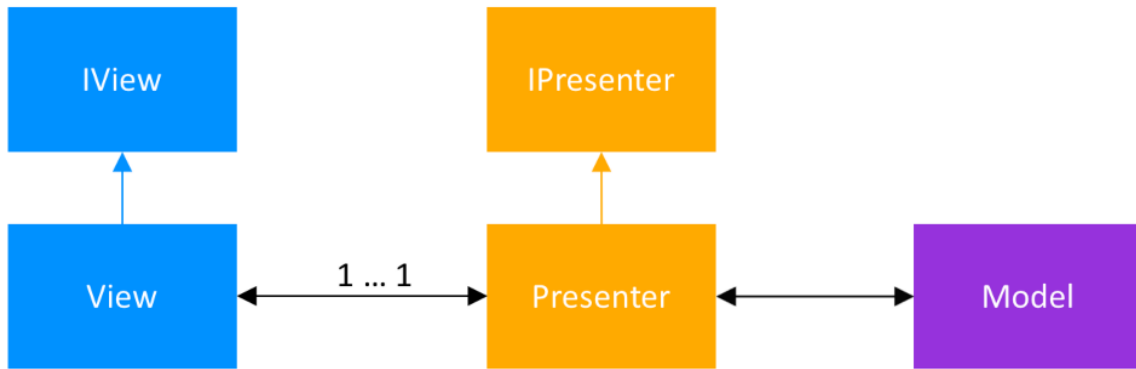


Figure 3.7: The MVP architecture

3.3.2.3 Model View View-Model

MVVM [17] is a software architectural pattern. MVVM main parts are the view, the model, and the view model. The different parts can be defined as follows:

- **View** - The view informs the view model about the user interactions.
- **View Model** - The view model exposes streams of data to the view. The view model can be seen as the state of the data in the model. It presents public properties and functions to the view.
- **Model** - The model is the domain object. It handles the data of the application. It holds the information but does not manipulate it.

The main difference between MVP and MVVM is that MVVM does not need a reference to the view, since the view can bind to streams of data exposed by the view model. In other words, the presenter interface can be avoided. The view can also notify the view model about user interactions, creating a two-way binding between the view and the view model. The view model does not know about the view, i.e., it does not hold a reference to the view. Figure 3.8 is a visual representation of the MVVM pattern.



Figure 3.8: The MVVM architecture

3.3.2.4 Chosen Architecture

We wished to expand our knowledge of different Android architectures, and as we had experience with MVC from previous projects, the decision was mainly between MVP and MVVM. Both architectures are frequently used in application development and looked promising. However, MVVM offered exciting features such as data-binding, which could make development significantly easier. MVVM also provided better decoupling of code than MVP, which makes the code easier to test and port to other platforms. MVVM is also a relatively new architectural approach for Android, which makes it interesting in regards to RQ2. For the aforementioned reasons we decided to use the MVVM architecture for this project.

3.3.3 Data Storage Technologies

There are numerous options when it comes to storing data, each with their advantages and disadvantages. This section will overview some of the different technologies that were considered, and present our conclusion.

3.3.3.1 Realm

Realm [18] is an open-source object database management system. It was initially developed for mobile platforms but has since expanded to other areas. Realm offers a two-way synchronization service between the remote Realm servers and the client side database that belongs to the individual users. Realm also provides functionality to store and query data locally on the devices, unlike traditional server-side databases. In Realm, the data is directly exposed as objects and can be queried by the code.

3.3.3.2 Room

Room [19] is an Android database library that provides an abstraction layer over SQLite [20]. It allows easy access to the database while still utilizing the full power of SQLite. Room makes this possible by generating the SQL-code through provided templates, which specifies how the database should be structured. Room also has functionality for listening to data changes, which makes it easier to keep data in sync. It should be noted that Room is a local database, and does not have server functionality built-in.

3.3.3.3 Firebase

Firestore [21] is a development platform for mobile and web applications offered by Google. Firestore provides a large selection of services, one of them being the Realtime Database. The Realtime Database allows data to be synchronized across multiple clients both online and offline. Firestore also has a service for authentication, which enables the user to choose between several OAuth [22] providers, such as Facebook and Google, for signing in. Other services include machine learning, crash analytics, storage, hosting and more.

3.3.3.4 Chosen Data Storage

Pecora required its data storage solution to work both online and offline. It is crucial that the application will function correctly when the farmer is performing a trip in an area with poor network connectivity, and that it can synchronize the data with cloud when online. While Room does offer all the functionality required for offline usage, it does not work for syncing data with the cloud. This functionality would have to be added to it by using an online database and then manually sync all data transactions.

On the other hand, Realm supports online and offline usage out of the box and provides synchronization between the online and offline databases. Firestore supports the same functionality, but it also comes with other services such as storage and authentication, which is also required by Pecora. Due to this, we believed that Firestore was the best choice of data storage for Pecora.

3.3.4 Map Technologies

It is necessary for Pecora to be able to show a map regardless of internet connection, creating the need for the user to be able to download map tiles for offline usage. This section will present the three map technologies considered for Pecora, and present the reasoning behind our choice.

3.3.5 Mapbox

Mapbox [23] is an open-source platform that provides the building blocks needed to create location-based features such as maps, search, and navigation. It has functionality for downloading tiles out of the box, however, to surpass the imposed limit of 6000 tiles, a payment plan is required. The payment plan is also necessary if a certain amount of users uses the application.

3.3.5.1 Google Maps

Google Maps [24] is a web-based mapping service provided by Google. Google Maps offers street maps, panoramic street views, satellite images, route planning, and more. In addition, it offers an API that allows developers to show maps inside their applications, as well as accessing other features of Google Maps. Unfortunately, it does not support downloading of tiles through the SDK, but it does have support for displaying tiles that have been downloaded from other sources. Like Mapbox, Google Maps requires a payment plan if a certain amount of users are reached.

3.3.5.2 Osmdroid

Osmdroid [25] is an open-source library that is an almost full replacement of Android's MapView. It offers a modular tile provider system that supports a wide range of offline and online tile sources, which can be used to download tiles. It also provides functionality for plotting icons, tracking location, and drawing shapes. Osmdroid is free to use, and due to it being open-source, it is highly flexible and customizable.

3.3.5.3 Chosen Map Technology

Google Maps integrates quite well with Android, but it comes with a few limitations. It costs money depending on the number of users and does not come with built-in functionality for downloading maps. While it is possible to implement this, it would require a lot of time and resources that could be better spent elsewhere.

Mapbox and Osmdroid both have functionality for downloading offline areas out of the box, but Mapbox puts a limit to how many tiles the user can download, which in some cases might not be enough. It is possible to increase this tile limit with a paid plan, but this would result in extra costs. Osmdroid is entirely free, highly customizable and can easily be extended to provide further functionality. While it might not be as polished as Mapbox or Google Maps, it was still chosen as the most suitable map library for Pecora.

3.3.6 Programming Languages

Android officially supports the programming languages Java, Kotlin, and C++. Java is the default language for Android applications. Due to our previous experience with the Java language in conjunction with Android application development, it was a natural choice for this project. We did not want to invest time into learning C++ or Kotlin. However, Kotlin was strongly considered as it was recently added as an officially supported language for Android and would be an exciting learning experience. Kotlin is also closely tied to Java, and would most likely require less time to learn than C++. In the end, we decided to use Java instead, as it has been supported by Android for many years, and has a considerable amount of documentation available online.

3.4 Software Development Methodologies

When developing software, it is important to choose an appropriate software methodology. Choosing the correct methodology will provide an effective, structural guideline for the development team. The upcoming sections will list the considered methodologies, and explain the reasoning behind our choice.

3.4.1 Scrum

Scrum [26] is an iterative and incremental agile framework that is used for managing work. When using Scrum, the project is divided into a fixed-length set of iterations called sprints. Each sprint starts with a planning meeting that defines the work and goals of the sprint and ends with a review. Scrum emphasizes a working product at the end of each sprint, that is fully functional, tested, documented, and potentially shippable. Scrums define different roles, such as the product owner, the scrum master, and the scrum team, and each role comes with its responsibilities. A product backlog is utilized to keep track of features, requirements, bug-fixes, and more. Each sprint has a backlog, which is a list of work that must be addressed during the sprint. The scrum process is depicted in Figure 3.9

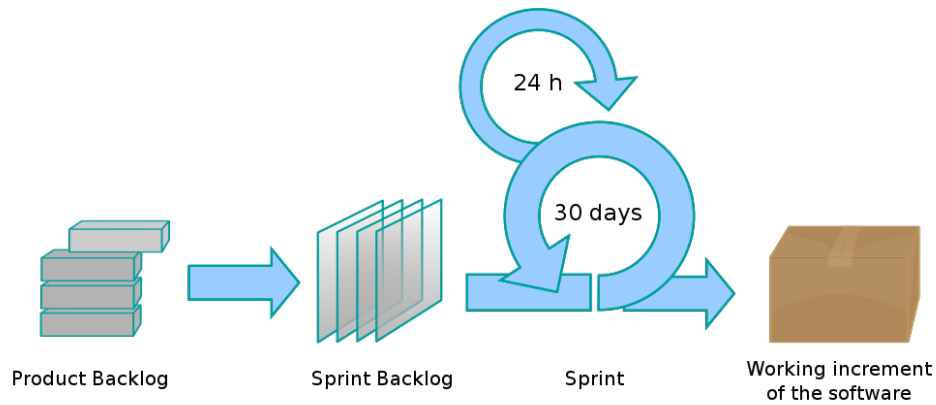


Figure 3.9: The Scrum process

3.4.2 Kanban

Kanban [27] is a lean method used to manage and improve the workflow by providing a visual representation of the individual work items. Kanban is used as a software development methodology but originated from manufacturing. The most important aspect of Kanban is the Kanban board, which visualizes the work items. The board is divided into several columns where each of the items are placed and flows from left to right based on their progression. The items are prioritized based on their placement in the column, where the items placed higher have a higher priority. Figure 3.10 shows an example of a fictional Kanban board.

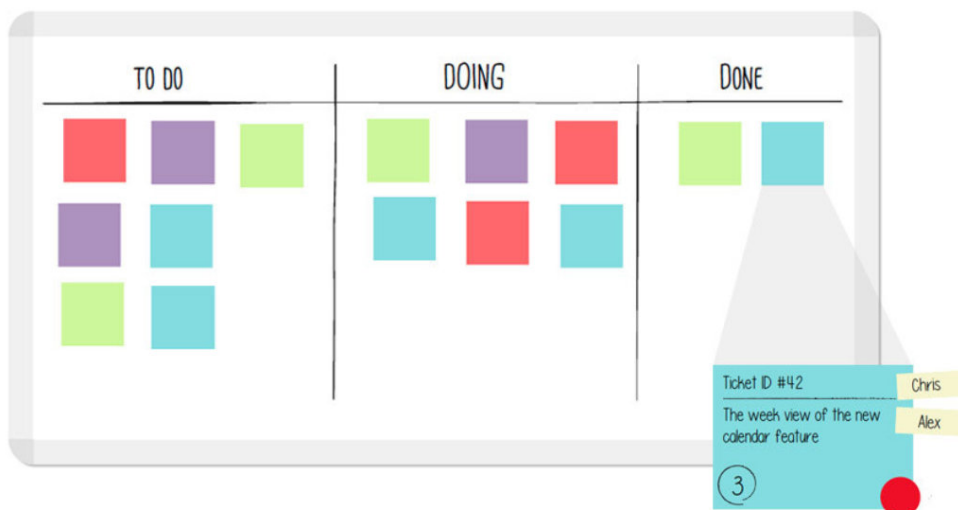


Figure 3.10: Representation of a Kanban board

3.4.3 Waterfall

The waterfall model [28] is a mostly linear methodology in which each step is completed before the next is started, as shown in Figure 3.11. This makes the waterfall method an excellent choice for projects where the requirements are well known beforehand, and where significant changes during the development do not occur.

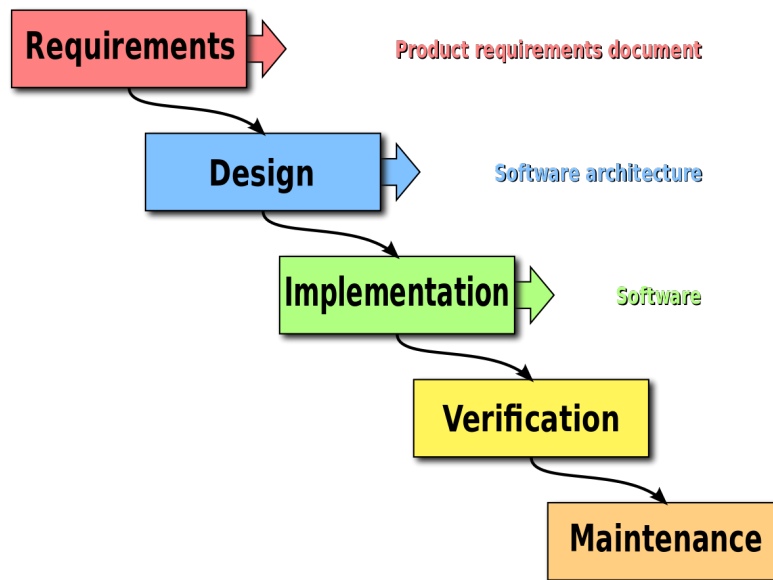


Figure 3.11: Waterfall model

3.4.4 Chosen Methodology

The waterfall methodology was considered in the early stages, as the sketches of the application and the earlier master's project had laid a solid foundation for the requirements of the system. However, we believed that changes would occur to the requirements as the project progressed; thus the waterfall method was discarded.

Scrum was considered as it would be agile enough to properly manage changes that might occur. We believe that the overhead using Scrum (backlogs, Scrum master, and sprints) would be quite significant, especially for a small team of two people. For this reason, Scrum was also discarded.

Kanban does not impose overhead to the same degree as Scrum, and we believed that Kanban's way of structuring the workflow would prove efficient and useful for our project. Besides, Kanban provides many of the benefits that Scrum does, such as allowing rapid change from user feedback. Thus Kanban was the natural choice for us.

Part II

Process & Requirements

Chapter 4

Project Development

This chapter explains the project development process used in this project, and how the project was planned.

4.1 Process

Kanban was the methodology used for this project, as explained in Section 3.4.4, and was realized through the usage of a Trello workboard. The workboard was divided into five columns that each had a specific purpose, and each column held cards that depicted either a concept or a task.

The "ideas" column was designed for ideas that were not thoroughly thought through or were otherwise not ready to be committed to the "todo" column. The "todo" column was designed for tasks that were to be done but was not actively being worked on. The ideas-column and the "todo" columns can be seen in Figure 4.1 and Figure 4.2 respectively.

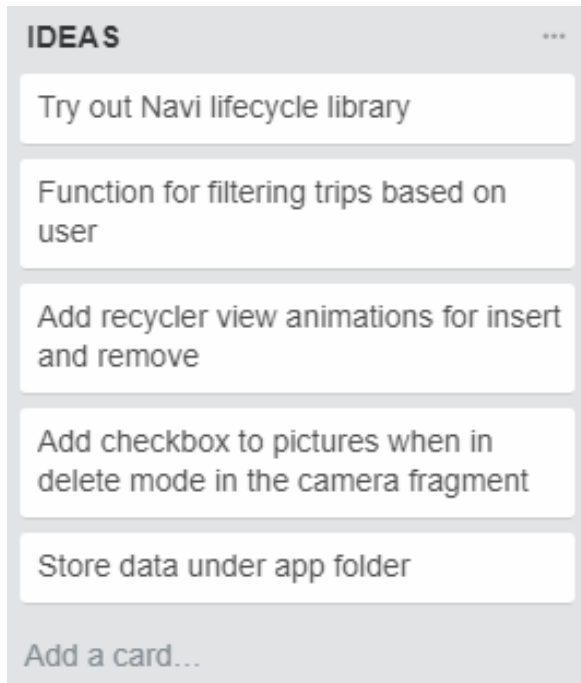


Figure 4.1: Trello idea column

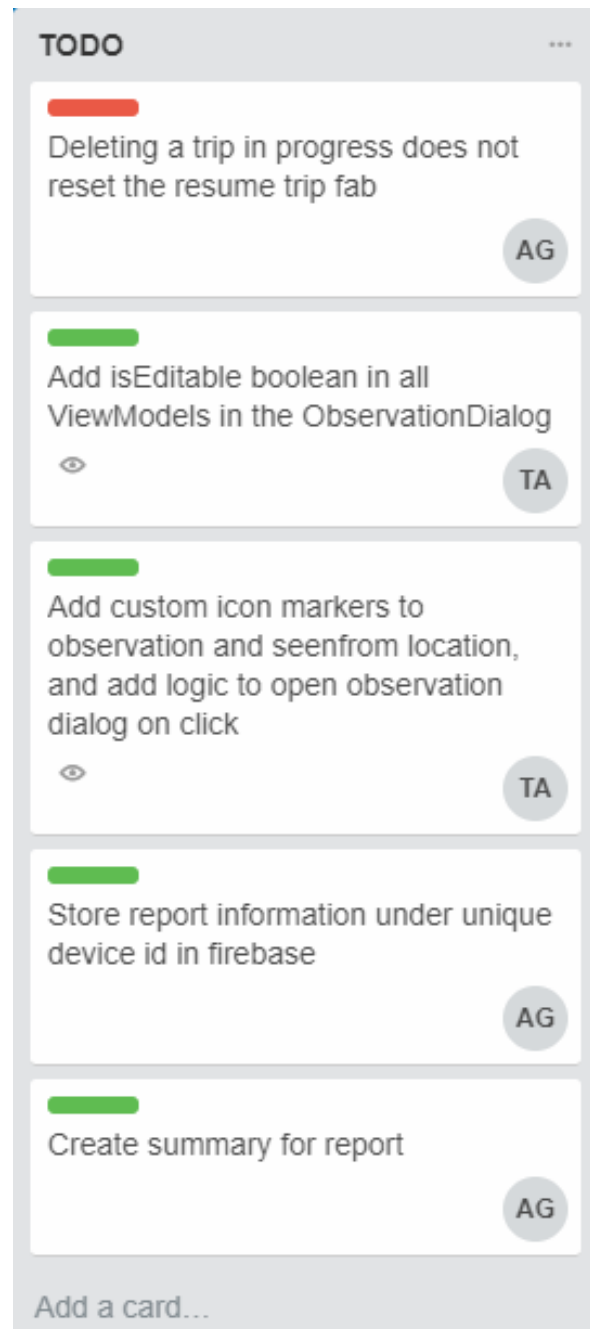


Figure 4.2: Trello todo column

The "doing" column was designed for tasks that were actively being worked on. The column also had a number assigned to it, which indicated how many cards could be placed in the column at any given time. We chose to allow four cards at the same time, two for each person. The purpose of the restriction was to avoid having one person with too much concurrent work ongoing. The "doing" column can be seen in Figure 4.3. The "done" column was designed for tasks that were completed and can be seen in Figure 4.4.

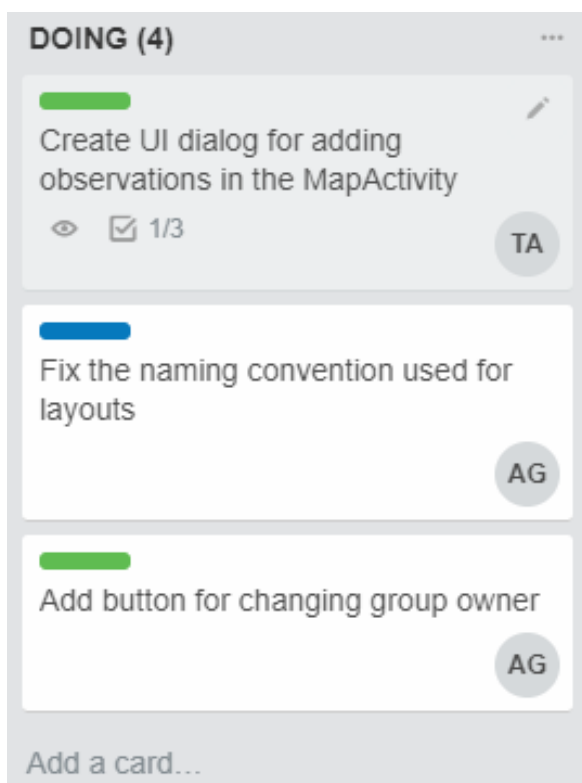


Figure 4.3: Trello doing column

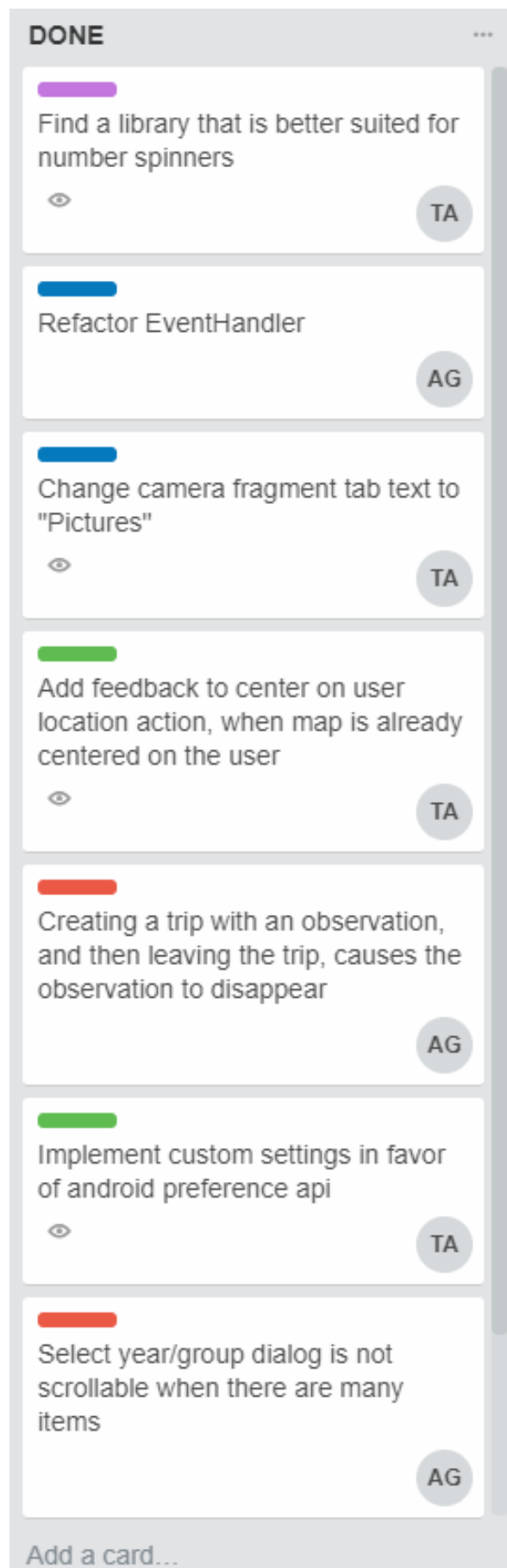


Figure 4.4: Trello done column

The "Color Tag Explanations" column was designed to hold the cards that explained the color tags. Each card in the other columns was given a color tag to signify what type of task it was. The purpose of the tags was to organize the tasks depending on what type of work is involved. The different tags can be seen in Figure 4.5.

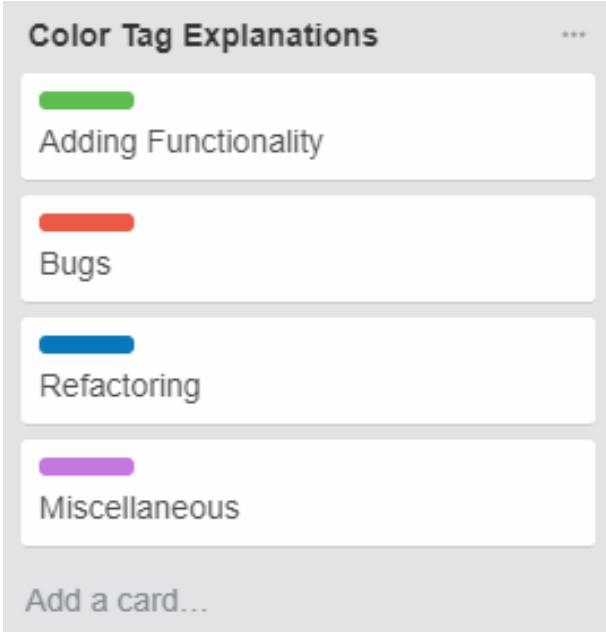


Figure 4.5: Trello color tag column

The cards of the columns usually floated from left to right, i.e., a card that was in the "todo" column would move on to the "doing" column. Sometimes it was necessary for cards to move back, e.g., when a task that was in the "done" column had to be refactored or go through bug-fixing, it would be moved to the "doing" column. One card could not move more than one column at the time. The card flow is showcased in Figure 4.6, where the thicker, filled arrows indicate the primary direction flow.



Figure 4.6: Card flow

Each column had the cards prioritized, where the higher the card was in the stack, the more important it was to progress. Each card would either have one or two people assigned to it, meaning that the person or people connected to the card were responsible for its progression.

Throughout the project, we met several times each week. Each meeting began with a stand-up

session where we took turns to explain and showcase what had been done since the last meeting. This usually spurred a discussion, which resulted in new cards being added or already existing cards being modified, moved, or removed. The days that had no meetings was usually spent working separately on the project, either working on the application or the thesis. Besides our meetings, we also met with the supervisor on a weekly basis to discuss and showcase the progress. The supervisor meetings were also a source for changes, additions, and removals to the Trello workboard.

4.2 Planning

In the beginning, an overarching plan was created for the project. To keep track of the plan, a Gantt-diagram was constructed. The Gantt-diagram can be found in Figure 4.7. The diagram shows three overarching categories: preliminary study, application, and report. The project was divided into two semesters, which is reflected in the diagram. During the first semester, the focus was the preliminary study, application development, and testing. The goal of the first semester was to have a functional and tested beta version. The second semester focused on refining the application, testing it further, working on minor additions, and writing this thesis. The goal of the second semester was to finish both the application and the thesis. The diagram workflow goes from left to right and from top to bottom.

The preliminary study category contains the research and planning phase of the project. During this phase, the focus was on obtaining knowledge about which technologies to use, examining existing solutions, and planning the course of the project.

The application category focuses on developing and testing features and fixing bugs. The features were prioritized in order of importance, and only one or two features were worked on at the same time. Some features were also scheduled with more time to spare than others, as some would require significantly more time and work to complete. From the figure, it is evident that the testing was delayed compared to the development of the first features. This was done due to our lack of experience with the architecture. We were uncertain as to how the architecture would work at the beginning of the project and had to familiarize ourselves with it. The duration of each testing item reflects how much work is required to reach a satisfying level of completion, i.e., unit testing should take more investment in terms of work hours than system testing, and so

forth. The testing cannot be completed before the refactoring, bug fixing, and minor additions are all done.

The report category contains three iterations and an item that is dedicated to the final adjustments of the thesis. The first iteration was planned to yield the first full draft of the thesis, and iteration two would refine the first draft, whereas the third and last iteration would consist of final adjustments. Each iteration was planned to require less time as the thesis approached completion.

It is also important to notice that whenever we made changes or changes were requested by the supervisor, the diagram was adjusted accordingly. This was the case when the "Group" and "Timeline" features were added. The new additions had to be inserted into the project plans, pushing other items away and taking their place instead.

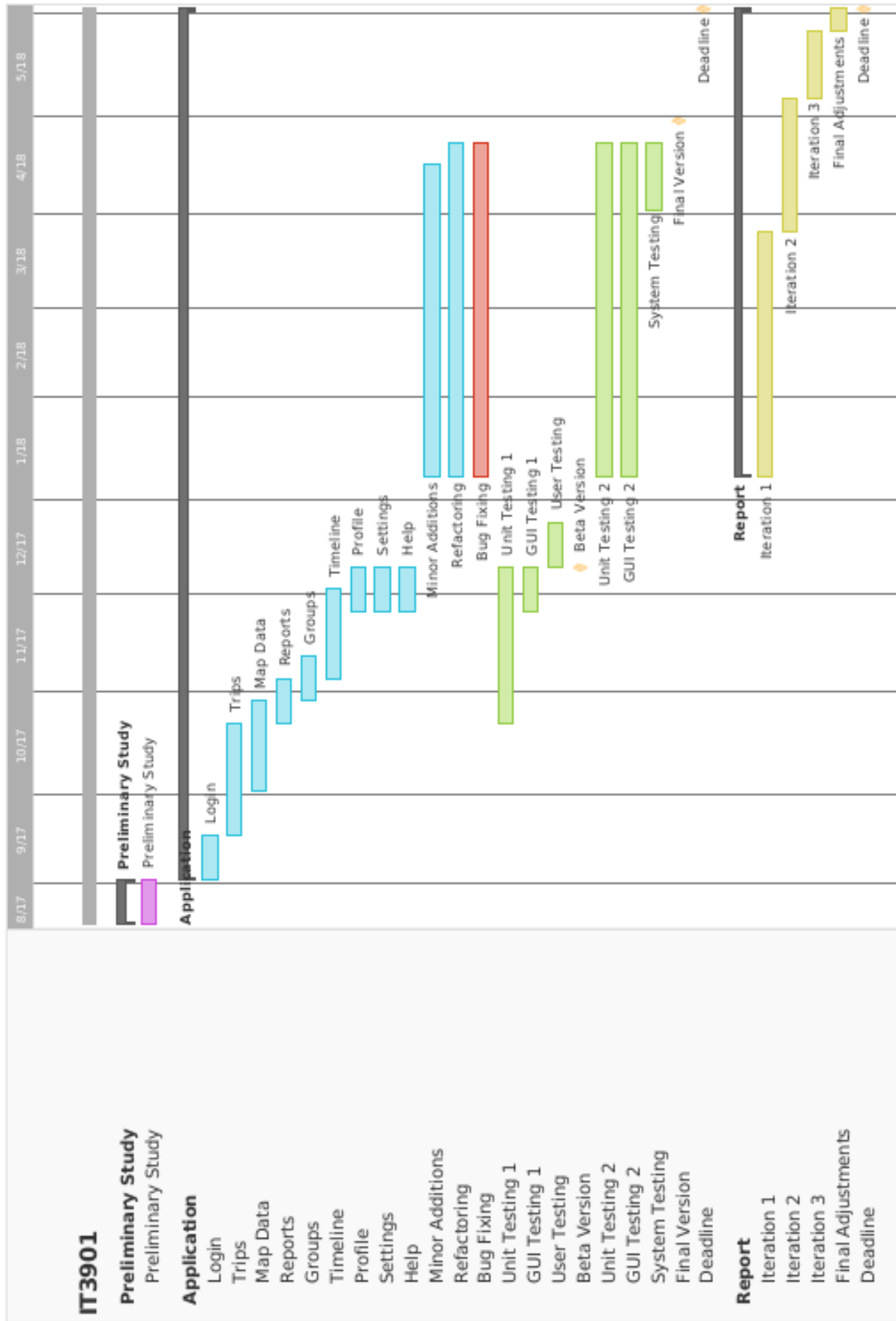


Figure 4.7: Gantt diagram

Chapter 5

Requirements

This chapter focuses on the functional requirements and quality attributes of Pecora, and how they were elicited.

5.1 Requirement Elicitation

It is often difficult to collect good requirements by asking the stakeholders what the system should or should not do, as one can never be certain if all requirements have been uncovered. To deal with this several different requirement elicitation approaches were employed: brainstorming, paper sketches, prototyping, and user observations.

The initial meeting with the supervisor set the stage for the elicitation of the requirements. The supervisor presented a document to us that contained a rough idea of how the product should look and work. Figure 5.1 showcases one of the pages from the document. The rest of the pages can be found in Appendix E. These pages, together with the dialogue with the supervisor, were used to devise a set of requirements which are presented later in this chapter.

Throughout the project, we met with the supervisor on a weekly basis. It was not uncommon for these meetings to contain a brainstorming session, where new requirements were uncovered, and already existing requirements were discussed and reviewed. The meetings also served as a place to showcase when significant progress had been made on Pecora, where new components or features were displayed to the supervisor as either paper sketches or a working prototype.

We usually sketched the components to be added to the Pecora application on paper before developing them. This enabled us to have a brainstorm session before writing any code on the component, and agree upon how the components should be designed, both visually and technically. Figure 5.2 shows one of the sketches that was drawn for the timeline component. The paper sketches were also showcased to the supervisor during the weekly meetings. The prototype was used to show a concrete implementation of the agreed upon requirements. Both the prototype and the paper sketches were used as a tool to elicit requirements, and often sparked a brainstorming session between the supervisor and us. It also presented something tangible, which is easier to form a discussion around, rather than discussing just concepts and ideas.

User observations were performed in conjunction with the usability testing, see Section 10.4.2.2 for more information. This gave us a chance to listen and observe as external users used Pecora, which provided valuable feedback that would be difficult to replicate from other sources.

Utilizing the methods above helped refine the initial requirements and uncover new requirements. All requirements associated to timeline, group, linking, login, and navigation between observations, mentioned in Table 5.2 were all devised from one of the methods mentioned above. In addition, a multitude of small changes, such as icon changes, text placements, etc. was uncovered but did not bare enough significance to create a new requirement.

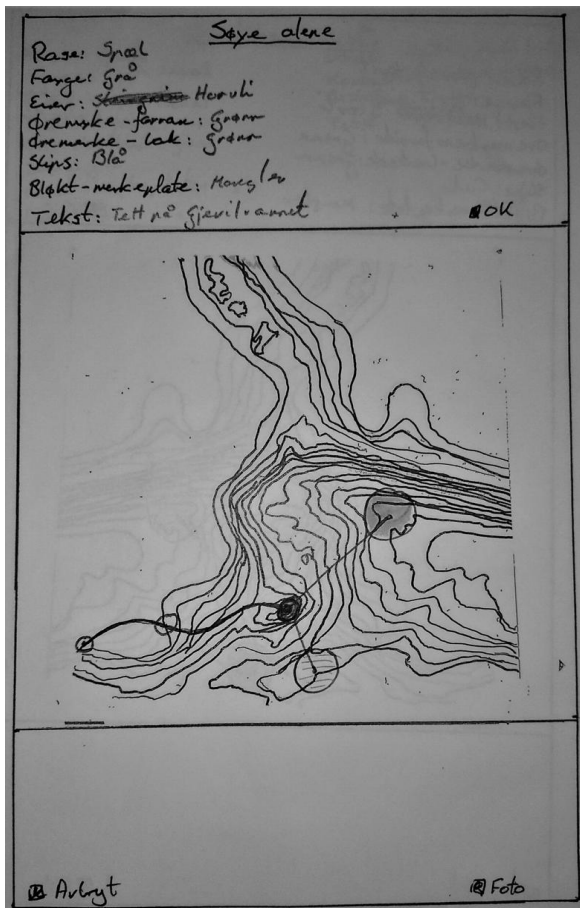


Figure 5.1: Sketch from supervisor

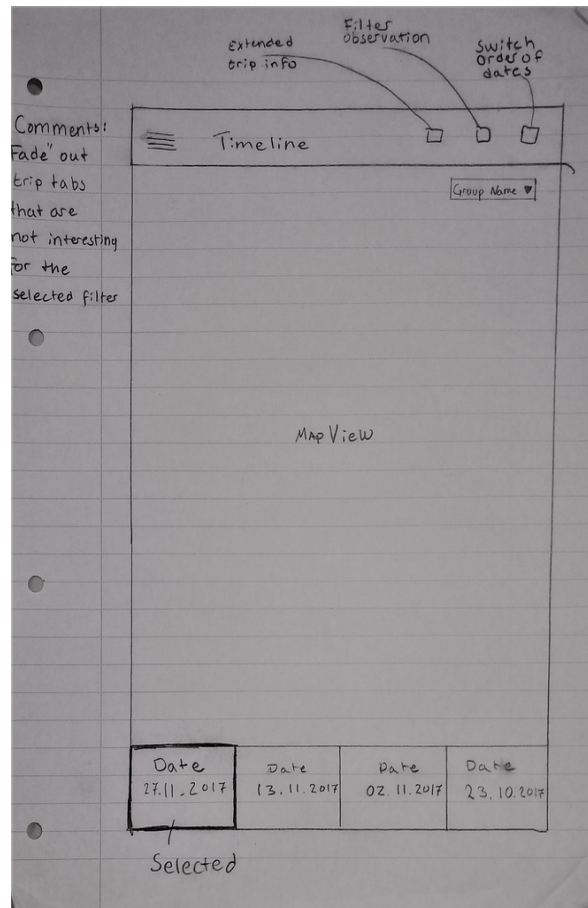


Figure 5.2: Sketch of timeline component

5.2 Functional Requirements

This section contains the use cases created before formalizing the functional requirements and the formalized functional requirements that were devised by utilizing said use cases. It also introduces the boilerplate method used to construct the requirements.

5.2.1 Use Cases

Initially, the functional requirements were constructed as use case diagrams and used to devise the actual functional requirements that can be found in table 5.2. The use cases provided in this section will give an overview of the core functionality of the Pecora application.

5.2.1.1 Log In

Figure 5.3 shows the use case diagram related to the log in interactions, which are also described below.

- **Log In** - Allows the user to log in to the application.
- **Create User** - Allows the user to create a new user for the application. This also automatically logs in the user after completion.
- **User Authentication** - Validates the user credentials when the user logs in.

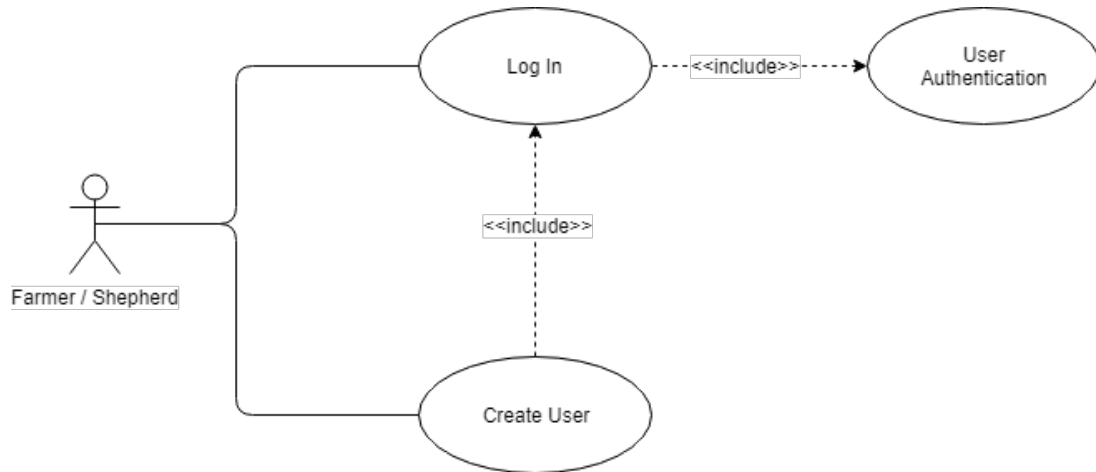


Figure 5.3: Use case diagram of log in

5.2.1.2 Map Data

Figure 5.4 shows the use case diagram related to the map data interactions, which are also described below.

- **View Downloaded Maps** - Shows a list of previously downloaded maps.
- **View Map Area** - When clicking on a downloaded map, show the area that has been downloaded.
- **Download Map Data** - Allows the user to select an area, and download it.
- **Delete Map Data** - Allows the user to remove areas that he has downloaded.

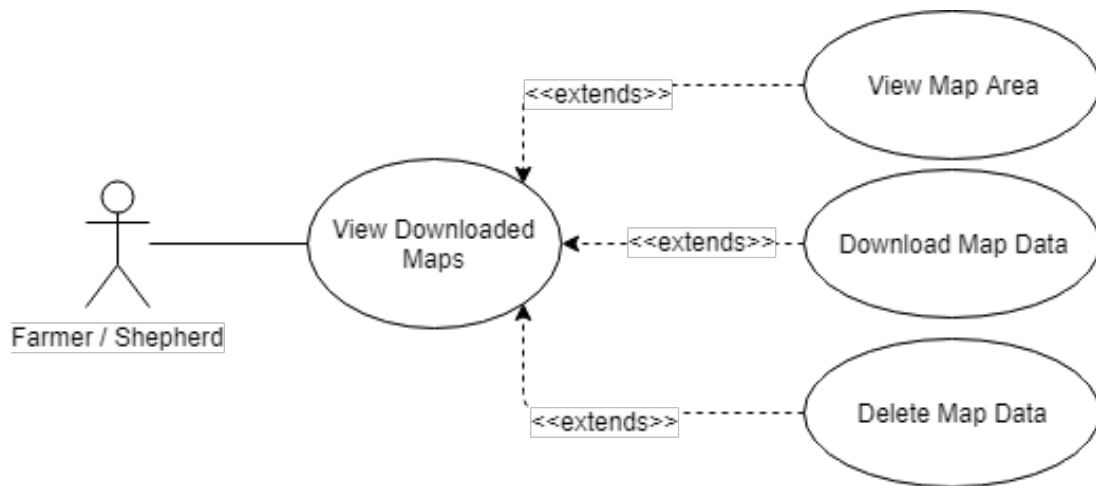


Figure 5.4: Use case diagram of map data

5.2.1.3 Group

Figure 5.5 shows the use case diagram related to the group interactions, which are also described below.

- **Create Group** - Create a new group.
- **View Group** - View list of groups that the user is a member of.
 - **View Group Members** - View list of members that belongs to a group.
 - **Add Member** - Invite a member to the group.
 - **Remove Member** - Remove a member from the group.
- **View Group Invitations** - View group invitations.
 - **Accept Group Invitation** - Accept group invitation. User will become member of group.
 - **Reject Group Invitation** - Reject group invitation. Invitation will be removed.

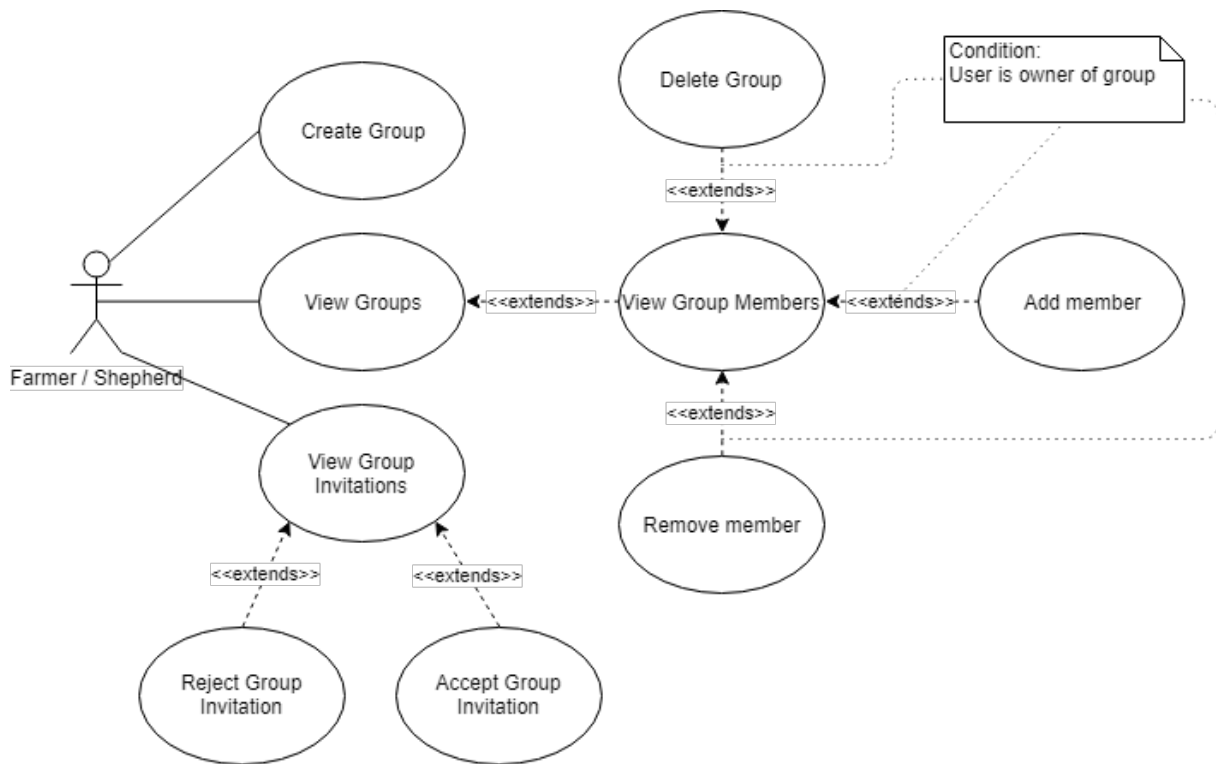


Figure 5.5: Use case diagram of group

5.2.1.4 Trip - part 1

Figure 5.6 shows the use case diagram related to the trip interactions (part 1), which are also described below.

- **Create Trip** - Create a new trip.
- **View Trips** - See a list of trips.
- **Delete Trip** - Delete a trip.
- **Move Trip** - Move a trip to a different group.
- **Filter By Year** - Filter trips by the year they were completed.

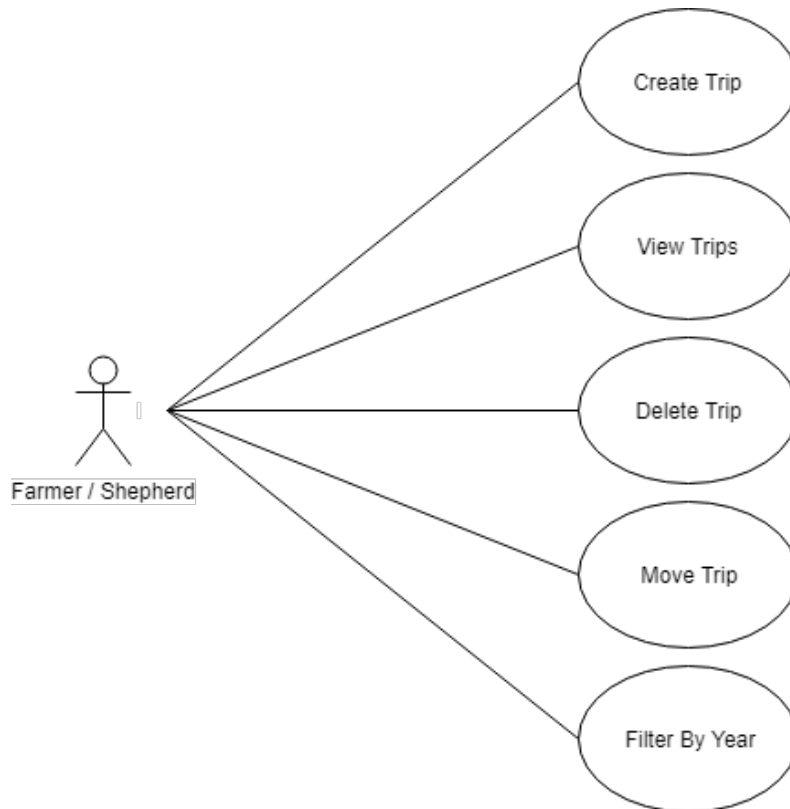


Figure 5.6: Use case diagram of trip part 1

5.2.1.5 Trip - part 2

Figure 5.7 shows the use case diagram related to the trip interactions (part 2), which are also described below. This use case diagram is a continuation of the use case diagram for trip part 1, i.e., the "Create Trip" use case. Trip part 1 also explains the "Create Trip" use case, which is why it is not repeated below.

- **Link To Observation** - Link a new "observed from position" to an already existing observation.
- **Show Trip Information** - Shows information about the trip.
 - **Edit Trip Information** - Edit the information about the trip.
- **Create Observation** - Create an observation.
- **Show Observation** - Shows information regarding a specific observation.
 - **Delete Observation** - Delete an observation.
- **Complete Trip** - Complete the current trip. The trip cannot be changed afterwards.

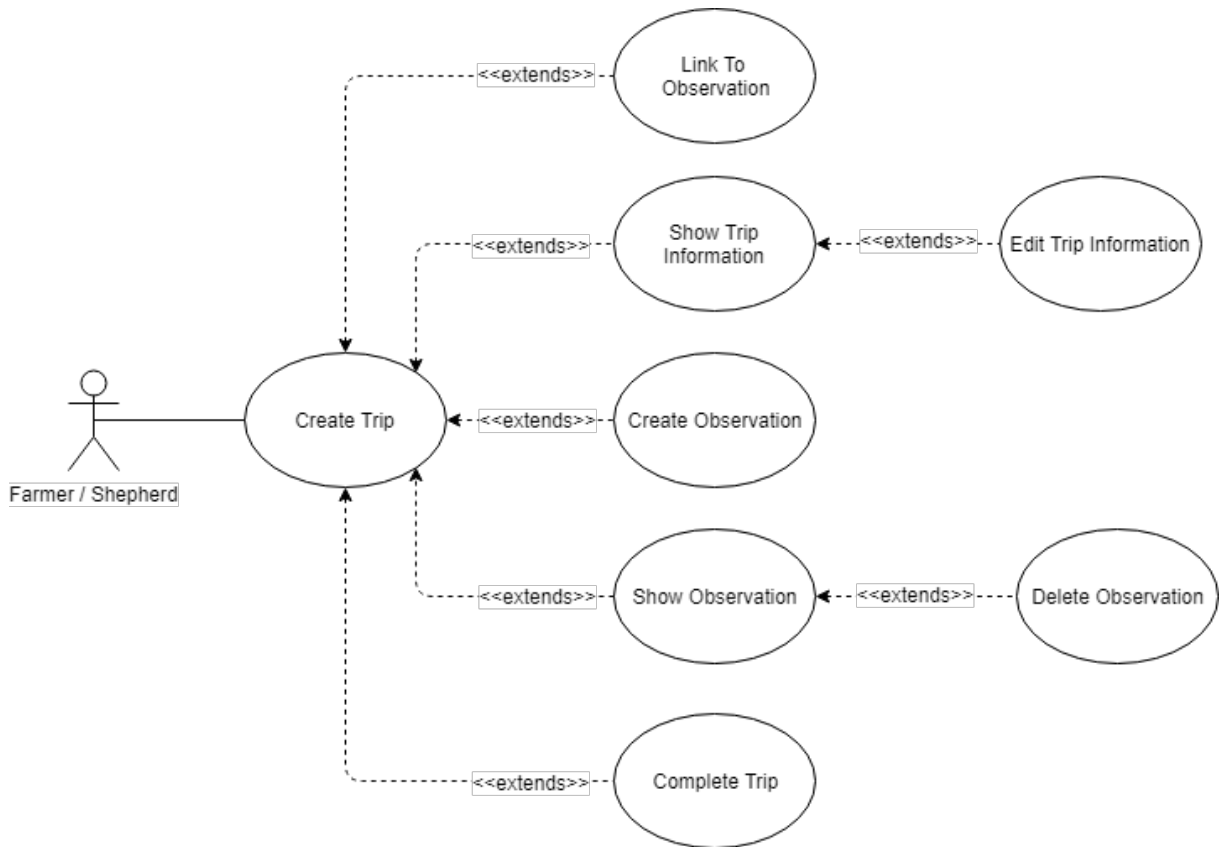


Figure 5.7: Use case diagram of trip part 2

5.2.1.6 Report

Figure 5.8 shows the use case diagram related to the report interactions, which are also described below.

- **View Reports** - Shows a list of previously created reports to the user.
- **Create Report** - Generate a new report.
- **Delete Report** - Delete selected reports.
- **Share Report** - Share reports, for example through email.

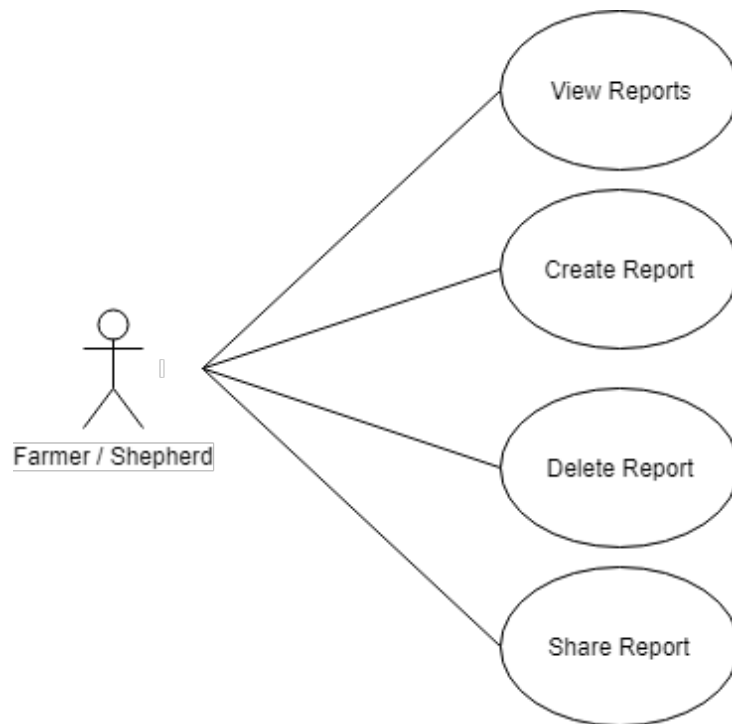


Figure 5.8: Use case diagram of report

5.2.1.7 Timeline

Figure 5.9 shows the use case diagram related to the timeline interactions, which are also described below.

- **Reverse Trip Order** - Reverse the chronological order of the trips.
- **View Completed Trips** - Shows the user a list of completed trips that can be selected and viewed.
- **Show Observation** - Show the observation from a completed trip.
- **View Trip Information** - Show trip information for the selected trip.

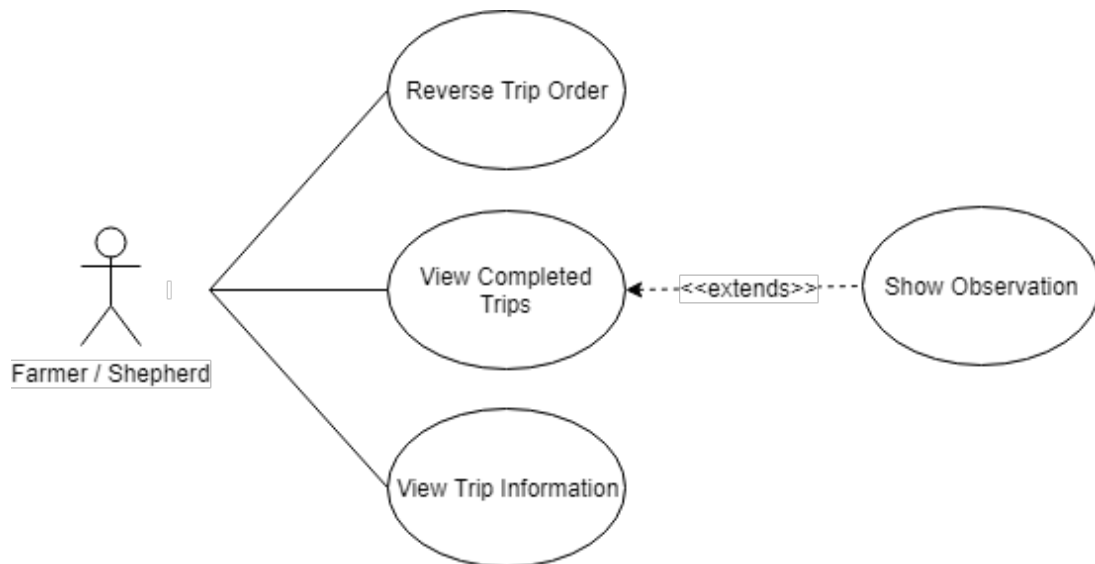


Figure 5.9: Use case diagram of timeline

5.2.2 Requirement Boilerplates

A boilerplate is a Natural Language (NL) pattern that provides grammatical structures for requirements by restricting the syntax of requirements to a predefined set of linguistic structures. When applied correctly the boilerplates provide a simple, yet effective method to improve the quality of requirements, by avoiding inconsistency, ambiguity, and complex structures [29]. Table 5.1 presents the boilerplates used to construct the requirements in Table 5.2. The generic variables of the boilerplates are replaced with real properties of the system in question, e.g., the <System name> variable could be replaced with *Pecora* in this specific case.

Table 5.1: The boilerplates used to construct requirements

ID	Boilerplate
B1	<System name> shall be able to <function> if <operational condition>
B2	<System name> shall be able to <function>
B3	<System name> shall be able to <function> with <operational condition>
B4	<System name> shall <function>
B5	<System name> shall <function> if <operational condition>

5.2.3 Requirements

Table 5.2 presents the functional requirements of Pecora. The requirements were prioritized in terms of how necessary they were to complete. Requirements with important features or requirements that were tightly coupled with other requirements and needed to be completed prior to others were prioritized higher. The priority levels are low, medium, and high. Prioritizing

the requirements made it easier to ensure that effort was focused on the necessary parts of the development of Pecora. All the higher prioritized requirements should be either be completed or be well in progress before starting on lower priority requirements. In addition, each requirement has a reference to which boilerplate that was used to construct the requirement.

Table 5.2: The functional requirements of Pecora.

ID	Functional Requirement	Priority	Boilerplate
Login			
1.01	Pecora shall be able to log in a user with his email and password	High	B2
1.02	Pecora shall be able to log in a user with his Google account	Medium	B2
1.03	Pecora shall be able to log in a user with his Facebook account	Medium	B2
1.04	Pecora shall log in the user automatically if the user has already logged in once before	High	B5
Create User			
2.01	Pecora shall be able to create a new user account with a user defined email and password	High	B3
2.02	Pecora shall be able to create a new user with the user's Google account	Medium	B3
2.03	Pecora shall be able to create a new user with the user's Facebook account	Medium	B3
2.04	Pecora shall automatically log in the user if his account has been created	Low	B5
Navigation drawer			
3.01	Pecora shall be able to log out the user	High	B2
3.02	Pecora shall show the user's avatar icon	Low	B4
3.03	Pecora shall show the name of user	Low	B4
3.04	Pecora shall show the email of the user	Medium	B4

Continuation of Table 5.2

Trips			
4.01	Pecora shall list the completed trips associated with the logged in user and his groups	High	B4
4.02	Pecora shall show the trip in progress if a trip is currently in progress	High	B5
4.03	Pecora shall separate shown trips into their associated groups	High	B5
4.04	Pecora shall be able to show a trip's observations and track on a map	High	B1
4.05	Pecora shall be able to create a new trip	High	B2
4.06	Pecora shall be able to delete a trip if the trip is owned by the logged in user	High	B1
4.07	Pecora shall be able to move trips to other groups if the user owns the trips	Medium	B1
4.08	Pecora shall be able to filter trips by year	Medium	B2
Trip			
5.01	Pecora shall be able to start a new trip	High	B2
5.02	Pecora shall be able to save a completed trip	High	B2
5.03	Pecora shall show the downloaded map tiles on a map	High	B4
5.04	Pecora shall show the center of the map through a crosshair	Medium	B4
5.05	Pecora shall be able to show the GPS track of a user during a trip	High	B2
5.06	Pecora shall be able to show the start GPS location of a trip	High	B2
5.07	Pecora shall be able to show the end GPS location of a trip	High	B2
5.08	Pecora shall be able to create a new observation	High	B2
5.09	Pecora shall be able to let the user set the observation radius	High	B2

Continuation of Table 5.2

5.10	Pecora shall be able to let the user set the observation direction	High	B2
5.11	Pecora shall be able to let the user set the observation scenario to sheep, lamb, wool, predator, reindeer, dog, and other	High	B2
5.12	Pecora shall be able to add a new picture from the user's gallery	Medium	B2
5.13	Pecora shall be able to capture and add a new picture by using the device's camera	High	B2
5.14	Pecora shall show pictures related to the selected observation	High	B4
5.15	Pecora shall be able to let the user delete pictures associated with an observation	High	B2
5.16	Pecora shall be able to let the user display a picture in fullscreen	Medium	B2
5.17	Pecora shall be able to let the user delete an observation	High	B2
5.18	Pecora shall be able to let the user edit observations if the trip is not completed	High	B1
5.19	Pecora shall be able to let the user view the trip's observations and track	High	B2
5.20	Pecora shall be able to center the map on the user's location	Medium	B2
5.21	Pecora shall be able to link observations from different locations	High	B2
5.22	Pecora shall be able to let the user delete links between an observation and a location	High	B2
5.23	Pecora shall be able to find and set the start location name based on the user's location	Low	B2

Continuation of Table 5.2

5.24	Pecora shall be able to find and set the end location name based on the user's location	Low	B2
5.25	Pecora shall be able pause a trip	High	B2
5.26	Pecora shall be able to highlight an observation and its associated links if the user centers the map on the observation	Medium	B1
5.27	Pecora shall be able to navigate between observations if the user clicks the 'Next Observation' button	Medium	B1
5.28	Pecora shall be able to navigate to the closest observation if the user clicks the 'Closest Observation' button	Medium	B1
5.29	Pecora shall be able to open an observation and show its details	High	B2

Map Data

6.01	Pecora shall show a list of downloaded areas	High	B4
6.02	Pecora shall be able to delete a downloaded area	Medium	B2
6.03	Pecora shall be able to show the downloaded area if the user selects it	Medium	B1
6.04	Pecora shall be able to rename a downloaded area	Low	B2
6.05	Pecora shall be able to download a map area	High	B2
6.06	Pecora shall be able to adjust the size of the map download area	High	B2
6.07	Pecora shall be able to place the map download area with hand gestures on a map	High	B3
6.08	Pecora shall be able to let the user set a name for the downloaded area	Low	B2
6.09	Pecora shall be able to let the user set the minimum and maximum zoom for the downloaded area	High	B2

Continuation of Table 5.2

6.10	Pecora shall be able to find and set the downloaded area's name based on its location	Low	B2
Timeline			
7.01	Pecora shall be able to show a timeline for trips completed by a specific group for a given year	High	B2
7.02	Pecora shall be able to show the trip details dialog if a trip is available in the timeline	Medium	B1
7.03	Pecora shall be able to reverse the order of the timeline trips	Low	B2
7.04	Pecora shall be able to show a trip on a map if the user selects it	High	B1
7.05	Pecora shall be able to show the GPS track of a user for a trip	High	B2
7.06	Pecora shall be able to show the start GPS location of a trip	High	B2
7.07	Pecora shall be able to show the end GPS location of a trip	High	B2
7.08	Pecora shall be able to highlight an observation and its associated links if the user centers the map on the observation	Medium	B1
7.09	Pecora shall be able to navigate between observations if the user clicks the 'Next Observation' button	Medium	B1
7.10	Pecora shall be able to navigate to the closest observation if the user clicks the 'Closest Observation' button	Medium	B2
7.11	Pecora shall be able to show the selected observation's registered information	High	B2

Continuation of Table 5.2

Groups			
8.01	Pecora shall show a list of all the groups that the user is a member of	High	B4
8.02	Pecora shall be able to create new groups	High	B2
8.03	Pecora shall be able to show all the members of a group if the user selects the group	Medium	B1
8.04	Pecora shall be able to let users delete a group if they are the owner of the group	Medium	B1
8.05	Pecora shall be able to let users change group name if they are the owner of the group	Low	B1
8.06	Pecora shall be able to let users transfer ownership of the group if they are the owner	Low	B1
8.07	Pecora shall be able to let users leave a group	Medium	B2
8.08	Pecora shall show a list of all group invitations	High	B4
8.09	Pecora shall be able to let users send a group invitation if they are the owner of the group	High	B1
8.10	Pecora shall be able to accept a group invitation	High	B2
8.11	Pecora shall be able to decline a group invitation	Medium	B2
Profile			
9.01	Pecora shall be able to change the user's avatar icon	Low	B2
9.02	Pecora shall be able to change the user's username	Low	B2
Settings			
10.01	Pecora shall be able to change application language	Low	B2
10.02	Pecora shall be able to clear out cached map data	Low	B2
10.03	Pecora shall be able to toggle on/off mobile network usage for downloading map data	Low	B2
10.04	Pecora shall be able to toggle on/off map rotation through hand gestures	Low	B2

Continuation of Table 5.2

10.05	Pecora shall be able change location distance interval for tracking user movement	Low	B2
-------	---	-----	----

Help

11.01	Pecora shall provide information about how to use the application in the help section	Low	B2
-------	---	-----	----

End of Table 5.2

5.3 Quality Attributes

This section deals with the quality attributes of Pecora. Quality attributes are also known as the non-functional requirements of the system. A quality attribute is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders. The quality attribute is characterized using quality attribute scenarios [30]. A quality attribute scenario is a quality-attribute-specific requirement and consists of six parts. Figure 5.10 showcases an example of a quality attribute scenario. The six parts of quality attribute scenarios are as follows [30]:

1. **Source of stimulus** - This is some entity (a human, a computer system, or any other actuator) that generates the stimulus.
2. **Stimulus** - The stimulus is a condition that needs to be interpreted when it arrives at a system.
3. **Environment** - The environment that the stimulus occurred in.
4. **Artifact** - The artifact that is stimulated. This may be the whole system or some piece of it.
5. **Response** - The response is the action undertaken after the arrival of the stimulus.
6. **Response measure** - When the response occurs, it should be measurable in some fashion so that the requirement can be tested.

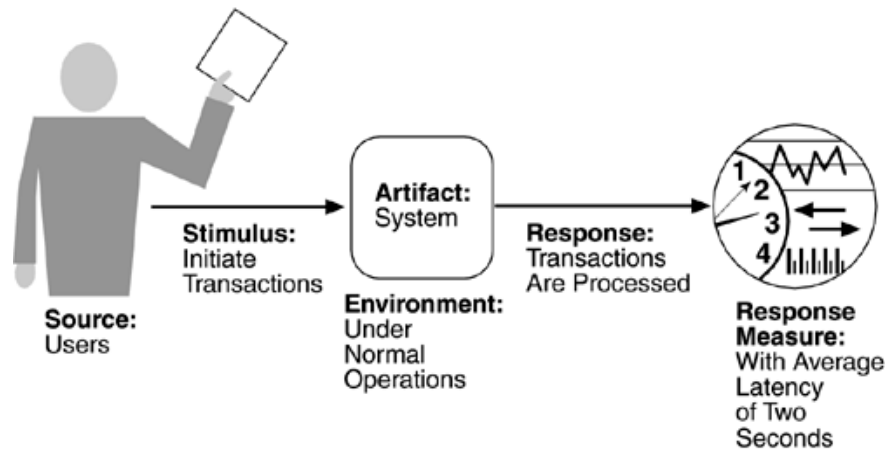


Figure 5.10: Example quality attribute scenario

All quality attributes have a set of tactics which describes how they can be achieved [30]. The tactics are defined in the following sections. The implementation of the tactics, however, will be discussed in Chapter 9.

5.3.1 Usability

Usability is defined as how well the system meets the requirements of the user. This includes properties such as how easy the system is to use and learn, and how well the user experience is received. Usability is an essential aspect of all systems that are used by end users in a non-professional setting. To improve the user experience and learning curve when using Pecora, usability was chosen as one of the quality attributes to focus on.

Source	End User
Stimulus	The end user interacts with the application
Environment	Runtime
Artifact	Application
Response	The end user uses the application efficiently
Response Measure	The end user should interact correctly with the application in 9 out of 10 operations

Table 5.3: U1 - Using the system efficiently

Source	End User
Stimulus	User starts the application for the first time
Environment	Runtime
Artifact	Application
Response	User feels comfortable using the application
Response Measure	User feels comfortable using the application within 1 hour of usage

Table 5.4: U2 - Learning the system

Source	End User
Stimulus	User accidentally deletes an item
Environment	Runtime
Artifact	Application
Response	Provide the user with functionality for undoing the deletion
Response Measure	Accidentally deleted items are restored 95% time

Table 5.5: U3 - Minimize accidental deletions

Source	End User
Stimulus	User perform an interaction
Environment	Runtime
Artifact	Application
Response	Provide the user with appropriate feedback
Response Measure	The end user is informed about the state of the application through feedback in 9 out of 10 interactions

Table 5.6: U4 - Feedback for interactions

5.3.1.1 Usability Tactics

Usability tactics can be divided into two categories, "Support User Initiative" and "Support System Initiative". The categories are concerned with what entity takes the initiative, which can either be the system or the user. The goal of the usability tactics is to provide the user with appropriate feedback and assistance [30], as seen in Figure 5.11.

The following "Support User Initiative" tactics were employed.

- **Cancel** - The user shall be able to cancel an action, and the system must respond appropriately. The system must:
 - Not block the action from being canceled.
 - Free up system resources used by the canceled action.
 - The action being canceled must be terminated.

- Any components that utilize the canceled action must be informed and take appropriate actions.
- **Undo** - The user shall be able to undo an action, and the system must respond appropriately. The system must maintain the earlier state and be able to restore it.
- **Pause/Resume** - The user shall be able to pause/resume an action, and the system must respond appropriately. Pausing a long-running operation will require the system to free up resources.
- **Aggregate** - Actions that are repetitive and affect a large number of system objects in the same manner, the system shall provide the user with the ability to aggregate said actions.

The following "Support System Initiative" tactics were employed.

- **Maintain Task Model** - Information about the given task being performed. Used to determine the context so the system has an idea of what the user is trying to accomplish.
- **Maintain User Model** - Used to represent information associated to the specific user using the system. The user can customize the user model to alter the behaviour of the system according to his preference.

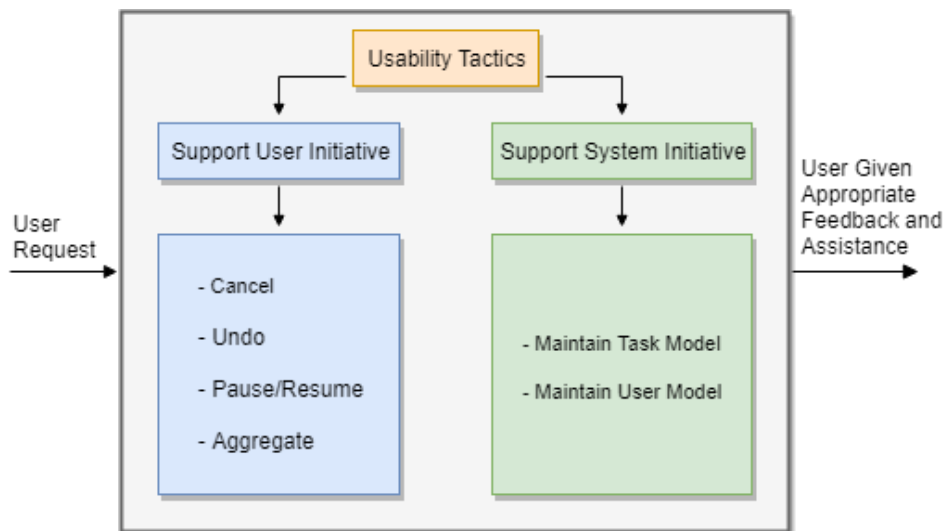


Figure 5.11: Usability tactics

5.3.2 Availability

Availability is defined as the software's ability to remain in a functional condition, such that the cumulative downtime does not exceed the required amount of time over a defined interval [30].

Pecora is required to run in an environment where connecting to the internet might not be possible. In such situations, Pecora should still be able to perform normally, without any further hindrances. For this reason, availability was one of the chosen quality attributes.

Source	Application
Stimulus	No network connection available
Environment	Normal operation
Artifact	Network connection
Response	Persist data to disk
Response Measure	100% of all data is persisted to disk

Table 5.7: A1 - Persist data when no network connection

Source	Application
Stimulus	No network connection available
Environment	Normal operation
Artifact	Network connection
Response	Reconnect to server
Response Measure	System should try to reconnect to the server within 5 seconds

Table 5.8: A2 - Reconnect to server when no network connection

Source	Application
Stimulus	Application reconnects to the server
Environment	Normal operation
Artifact	Network Connection
Response	Synchronize persisted data with server
Response Measure	100% of the persisted data should be synchronized with the server

Table 5.9: A3 - Synchronize with server when reconnecting to server

Source	Application
Stimulus	Application crashes
Environment	Normal operation
Artifact	Process
Response	Crash report is sent to remote server
Response Measure	All crashes should be reported to the server

Table 5.10: A4 - Send report when application crashes

5.3.2.1 Availability Tactics

The goal of the availability tactics is to enable the system to endure faults, such that the availability of its services is not compromised. Availability tactics can be divided into several

categories, but Pecora only uses tactics from "Preparation and Repair". Figure 5.12 is a visual representation of the availability tactics employed.

- **Exception Handling** - Whenever an exception occurs, the system must handle it in some fashion.
- **Retry** - When performing an action, the system should try to retry the action until it completes successfully.
- **Degradation** - When the system is in a degraded state, it should gracefully reduce the system’s functionality, while the modules not affected should operate normally.

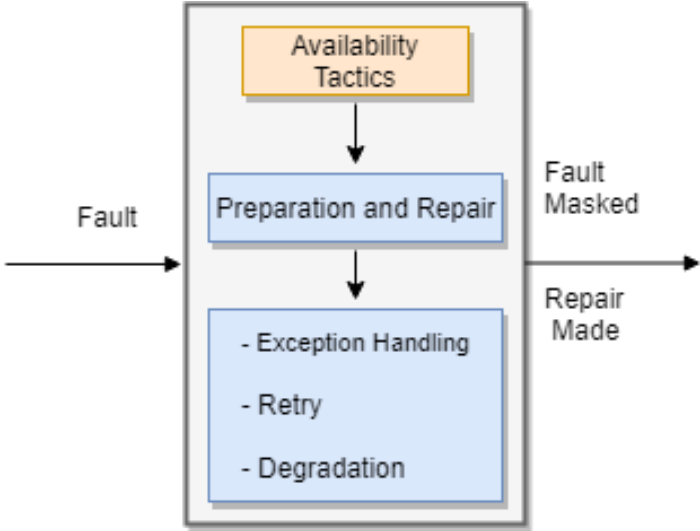


Figure 5.12: Availability tactics

5.3.3 Modifiability

Modifiability is defined as the system’s ability to handle changes and emphasizes reducing the cost of change in terms of time and money [30]. There is a possibility for changes in the functional requirements in the future, either from end users, or external stakeholders. To allow Pecora to handle such changes with minimal cost, modifiability is of great importance.

Source	Developer
Stimulus	The developer wants to add a new observation scenario
Environment	Design time
Artifact	Code
Response	New observation scenario is created and unit tested
Response Measure	Observation scenario is created within 1 day

Table 5.11: M1 - Create new observation scenario

Source	Developer
Stimulus	The developer wants to modify an observation scenario
Environment	Design time
Artifact	Code
Response	The observation scenario is modified and tested
Response Measure	The observation scenario is modified and tested within 6 hours

Table 5.12: M2 - Modify observation scenario

Source	Developer
Stimulus	The developer wants to change the report format
Environment	Design time
Artifact	Code
Response	The developer changes the report format and update unit tests
Response Measure	The report format is changed and tested within 6 hours

Table 5.13: M3 - Change report format

5.3.3.1 Modifiability Tactics

The goal of the modifiability tactics is to control the complexity, time, and cost of making changes. Modifiability can be divided into several categories: "Reduce Size of a Module", "Increase Cohesion", "Reduce Coupling", and "Prevention of Ripple Effects" [30]. The categories are shown in Figure 5.13.

The following tactic from the "Reduce Size of Module" category was employed:

- **Split Module** - Refining one large module into several smaller modules.

The following tactic from the "Increase Cohesion" category was employed:

- **Increase Semantic Coherence** - If a module is responsible for A and B, but the responsibilities A and B do not serve the same purpose, then the responsibilities should be placed into different modules.

The following tactics from the "Reduce Coupling" category were employed:

- **Encapsulate** - Introduces an interface to a module. The interface exposes an API for other modules to use.
- **Intermediary** - Breaks dependencies between modules. If responsibility A and B depend on each other, the dependency can be broken by using an intermediary. The type of inter-

mediary depends on the dependency, but one example is a publish-subscribe intermediary.

The follow tactics from the "Prevention of Ripple Effects" category were employed:

- **Hide Information** - Modules should hide information that are not required by others. Thereby exposing only a minimal amount of information to other modules.
- **Restrict Communication Paths** - Modules should only be allowed to communicate through specific communication paths.

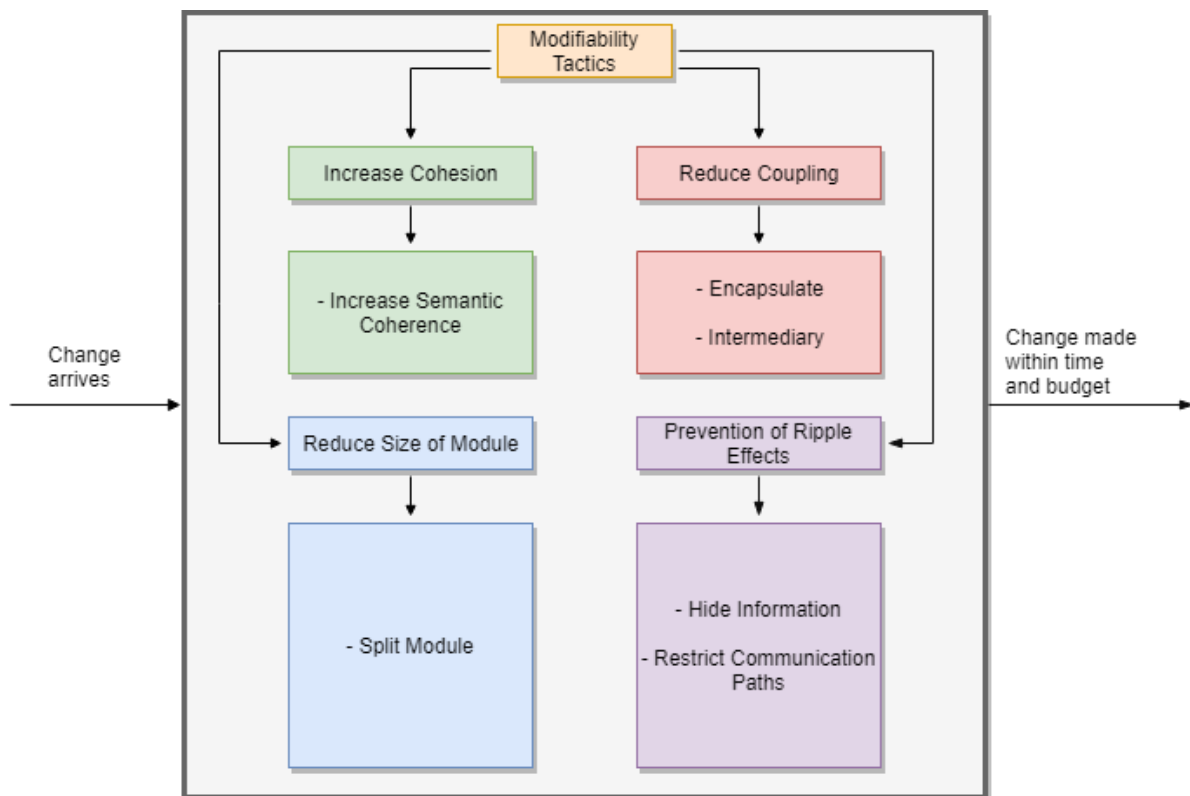


Figure 5.13: Modifiability tactics

Part III

Application

Chapter 6

Walkthrough of Pecora

This chapter will give an extensive walkthrough of the Pecora application, and each section will examine one of the core components. The walkthrough will focus on the functionality provided in each component, as well as the underlying design choices.

6.1 Log in

When starting Pecora for the first time, the user will be shown the "Log In" screen as seen in Figure 6.1. Here the user can log in to the application using his Facebook, Google, or custom application-specific credentials. If the user does not have an account, he can click the "Create User" button, which will take them to the "Create User" screen seen in Figure 6.2. This must be done regardless of the preferred login option, as an account must be registered with the application before the user can be logged in. This means that first-time users must always complete this step before logging in.

In the "Create User" screen the user can either click the "Join with Facebook" button or the "Join with Google" button, which will make it possible to use the "Sign in with Facebook" and "Sign in with Google" buttons respectively. It is also possible to create an account with a custom username, email, and password. Using Google or Facebook is the recommended way, as it is much faster than creating an account with email and password. It also helps that the user will have one less password to remember. When the account is created, regardless of which method was chosen, the user will be logged in automatically. When the user starts the application the

next time, he will also be logged in automatically.

Since Pecora is based around the concept of users and groups, it is necessary that the user can log in with an account. Different users have different ways that they would like to log in. To support the needs of as many users as possible, Pecora uses three different login methods. The usage of single sign-on means that the users only have to log in once. This saves them a lot of time whenever they start the application again in the future.

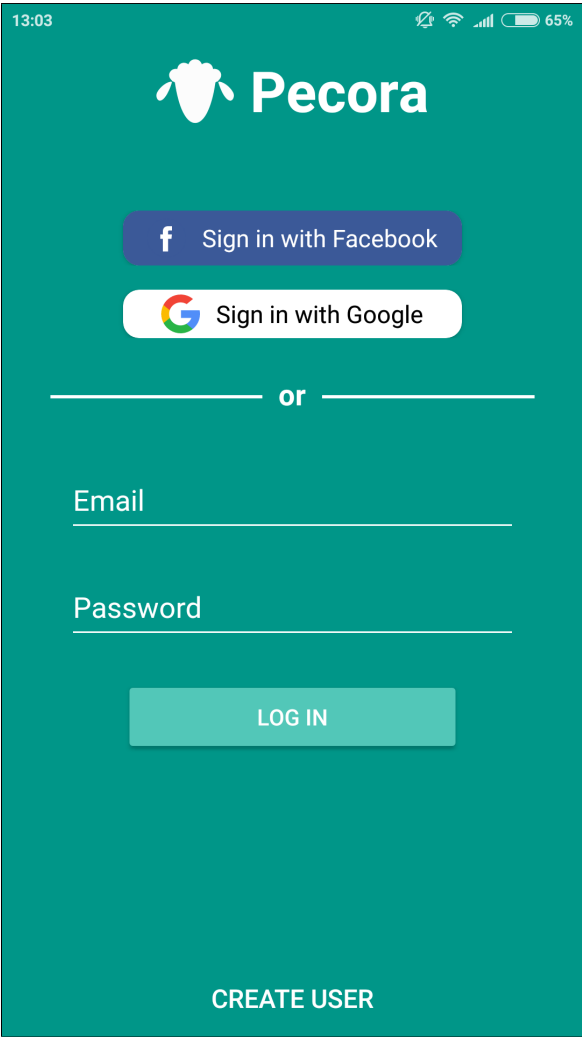


Figure 6.1: Sign in screen

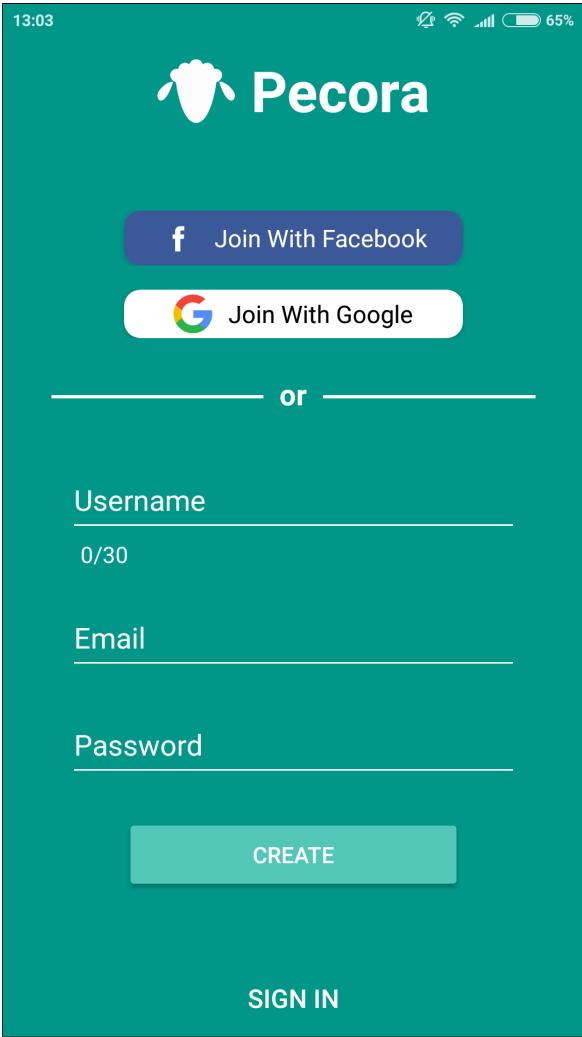


Figure 6.2: Create user screen

6.2 Navigation Drawer

The navigation drawer offers a flat hierarchy navigation option that is accessible from the majority of the application through the "Hamburger" icon. It presents the core components of the Pecora application as a list of items, where the more commonly used components are placed towards the top of the list. If the user clicks on one of the components in the navigation drawer list, the

initial screen for that specific component is shown. In addition, the navigation drawer displays the name, email, and profile picture of the signed in user. This allows the user to quickly identify the logged in account, which is helpful in cases where multiple accounts are utilizing the device. Quick access to the email of the account is also useful in regards to group invitations, which will be further discussed in Section 6.3. Figure 6.3 shows the navigation drawer in the Pecora application.

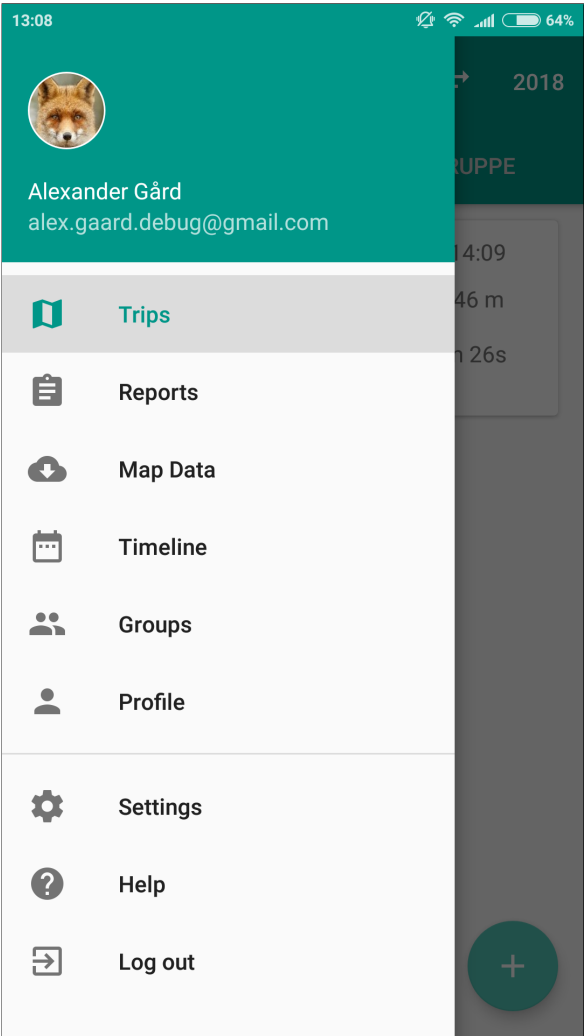


Figure 6.3: Navigation drawer

The navigation drawer is frequently used in other Android applications. By using a navigation drawer in Pecora, the user will have an effective and familiar way of navigating around. Another option would be to use an overflow menu in the upper right corner. While this would work, it is harder to access, as it requires clicking the upper right corner, instead of swiping in from the

left. It also lacks the option of displaying the user's account information, which is an important feature of the navigation drawer.

6.3 Groups

Groups are an integral part of the Pecora application, as it allows users to collaborate through a shared entity. The purpose of the group concept is to enable users to distribute observation trips among themselves, where one user takes the responsibility of covering the ground of all the group members during an observation trip. The responsibility could then shift to another group member for the next observation trip. The group component also allows a user to view other group member's observation trips, which can be beneficial if the user wants to be certain that the trips were completed as agreed upon.

The group component functions as a management tool for the user's groups. Initially, the user is presented with an image and text message informing them that they are not a member of any groups yet. If the user is a member of one or more groups, then they will be presented in a list format as shown in Figure 6.4. Each group has their name and its member's profile picture displayed in the list. The first picture is always the leader of the group, and there can only be one leader per group.

To create new groups the user can click the "create" button in the lower-right corner. This will open a dialog where the user can specify the name of the new group, as shown in Figure 6.5. When a new group is created, the user that created it is automatically set as the leader.

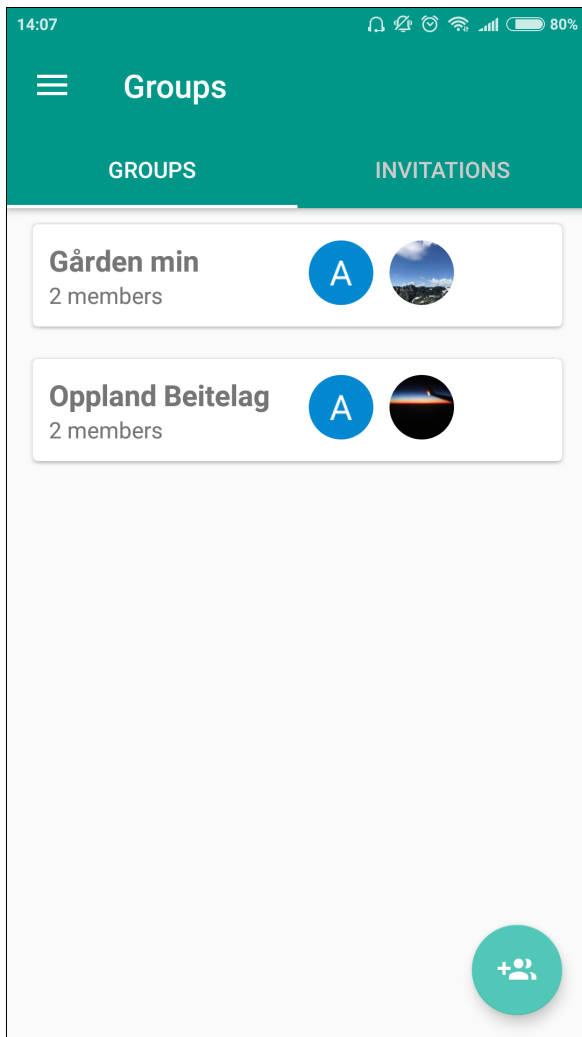


Figure 6.4: The group list

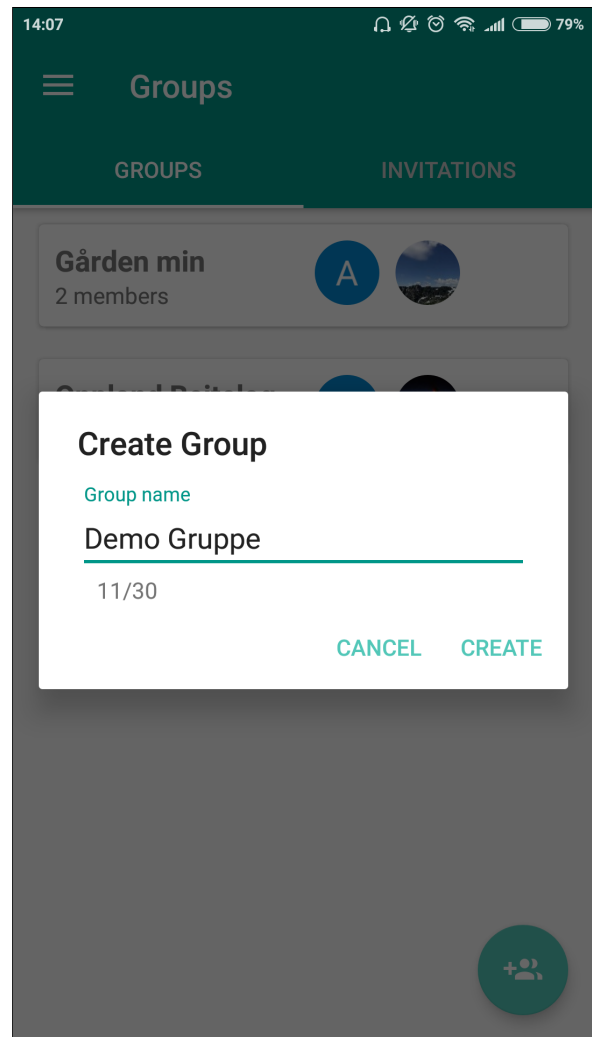


Figure 6.5: Creating a new group

Clicking a group will open a new screen that shows all its members, as shown in Figure 6.6. The members of the group can also be clicked, which will open a dialog with content that depends on the role of the user that clicked it. If the user is the leader of the group, two buttons are displayed. The "remove" button is to remove the member from the group, and the "make owner" button is to transfer the ownership of the group. If the user clicks himself, he is presented with a button that allows him to leave the group. In all cases, a larger profile picture of the clicked member is shown. Figure 6.7 shows a dialog that has been opened by the leader of a group.

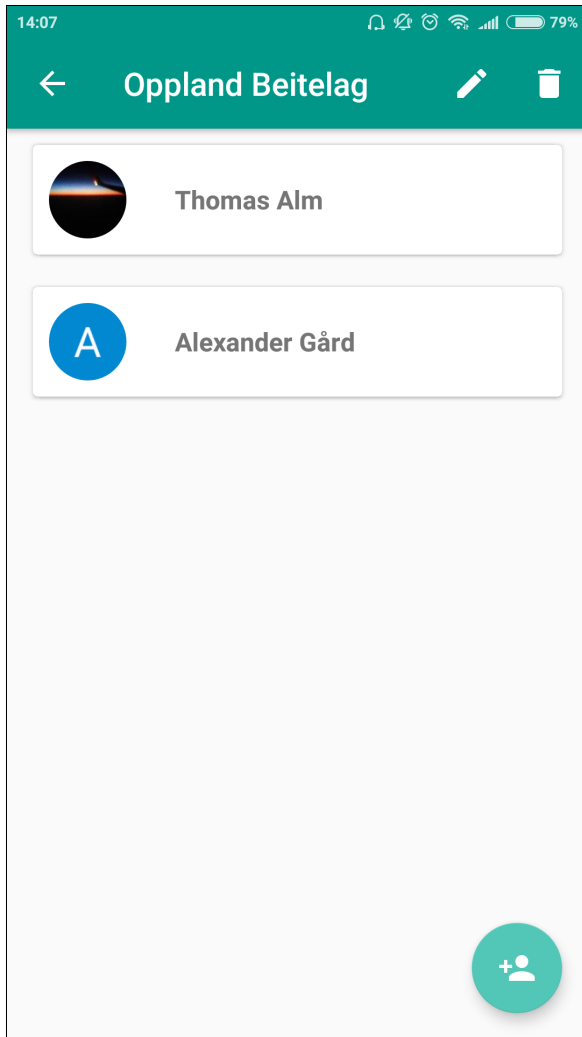


Figure 6.6: Specific group overview

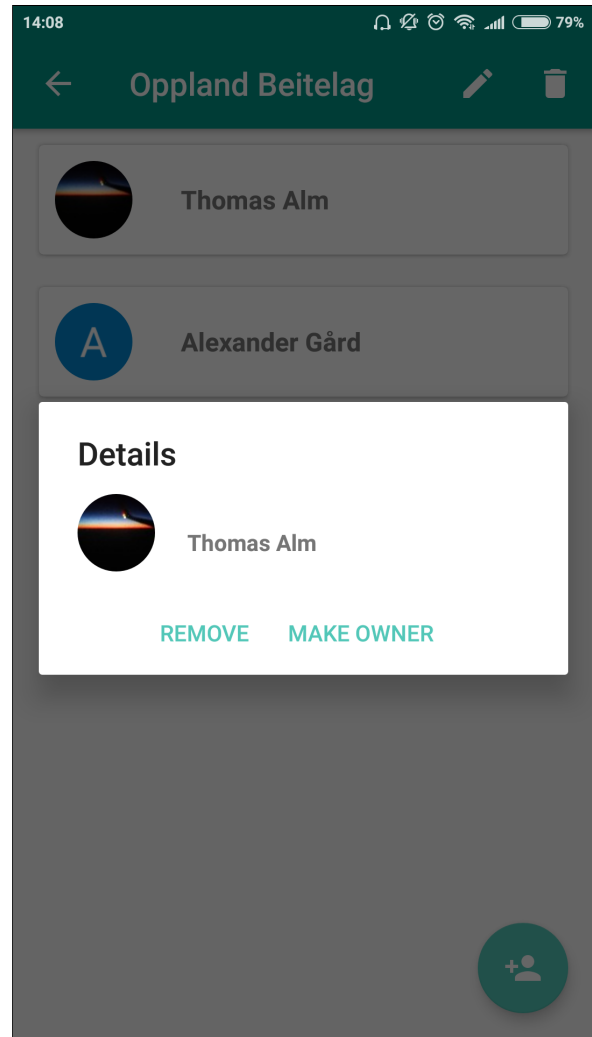


Figure 6.7: User details

The leader can also rename or delete the group from this screen. When the leader clicks the delete icon, a confirmation dialog will first be shown. The user will be presented with a confirmation message, which explains that the group will be permanently lost if he chooses to delete it. If the user wishes to delete the group, then he must swipe the "delete" button to confirm, as shown in Figure 6.8. Since deletion of groups is permanent, it is crucial that there is a confirmation dialog before the user can delete the group. Using a typical button for confirming the deletion would be possible, but it might be clicked by accident. The swipe button is harder to activate by mistake, which is why it was used instead.

Initially, groups only have one member, which is the leader of the group. The leader has the option to invite new members by clicking the button in the lower-right corner. Clicking the button will open a dialog where the user can type in the email of the user he wishes to invite. The "send invitation dialog" is shown in Figure 6.9.

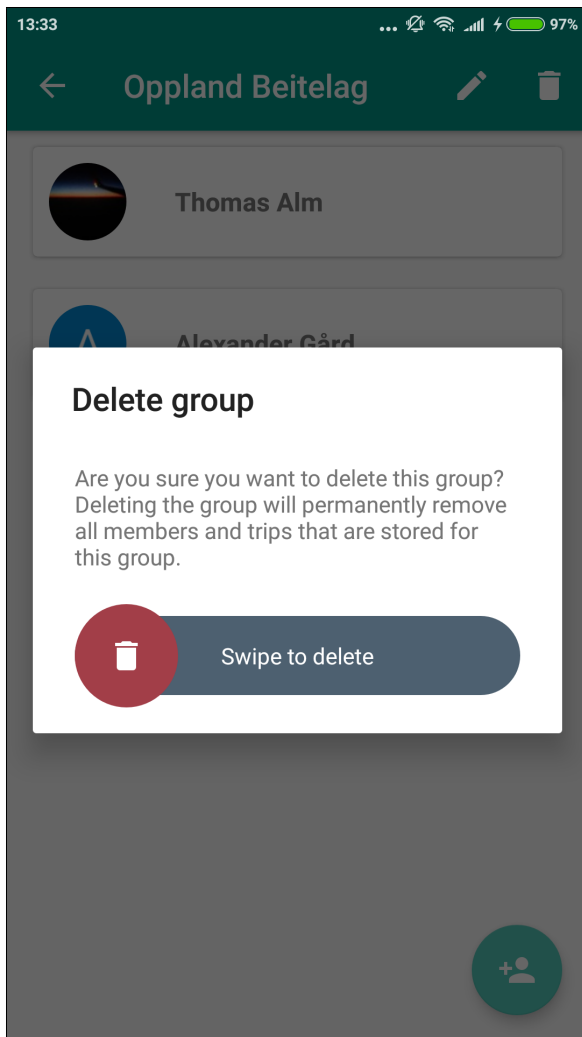


Figure 6.8: Delete group dialog

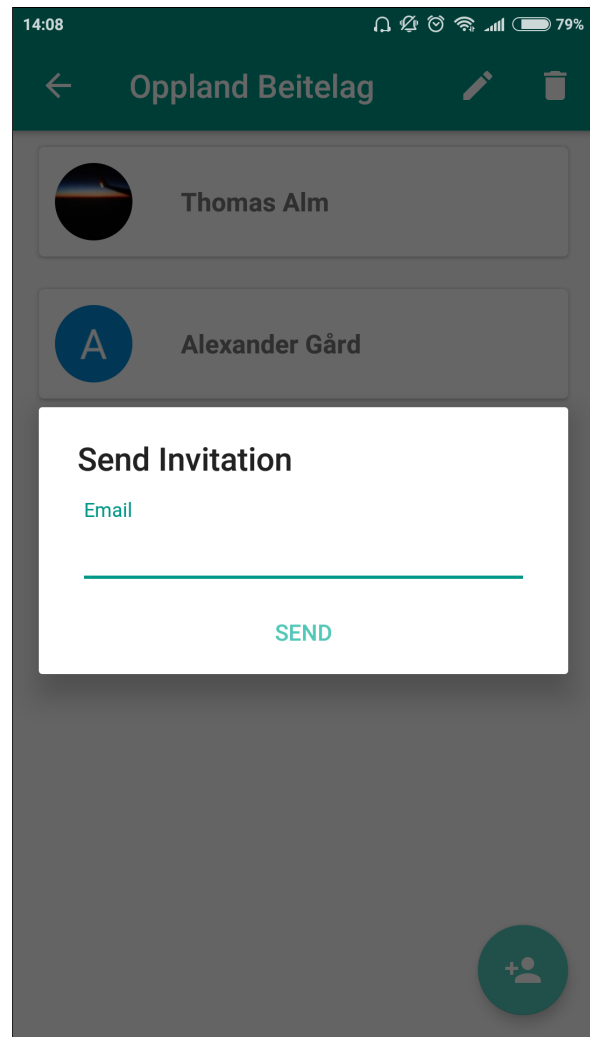


Figure 6.9: Send invitation dialog

The invitations tab lists all the group invitations that the user has received. Figure 6.10 shows an example where the user has one invitation he has not responded to. To respond to an invitation, the user can click on the individual invitations. This will open a dialog where the user can either accept or decline the invitation. If the user accepts the invitation, then the invitation will be removed, and the user will become a member of the group. If the invitation is rejected, then it will be removed, and the user will not become a member of the group. Figure 6.11 shows the "invitation details dialog".

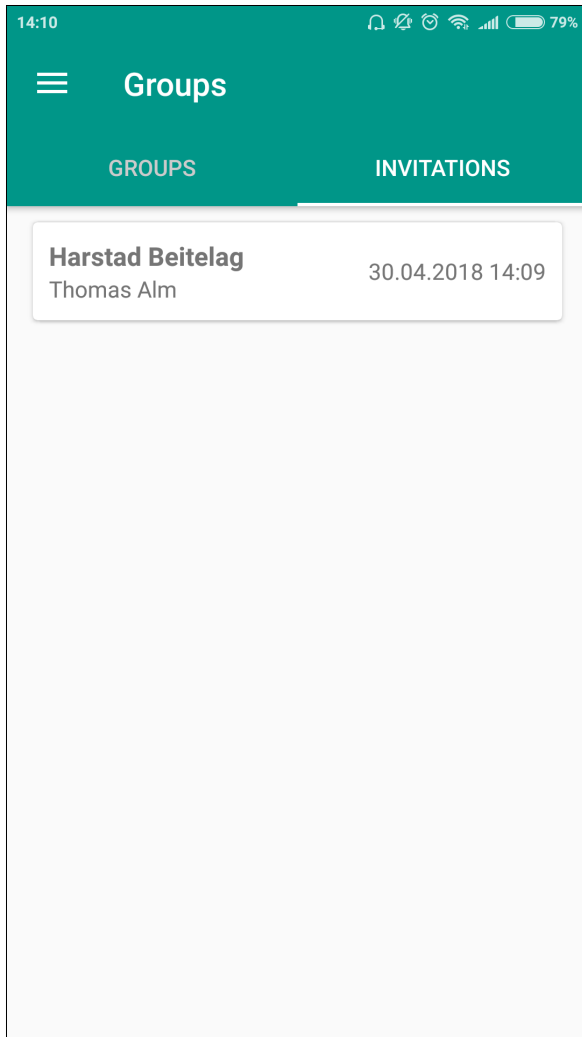


Figure 6.10: Invitation list

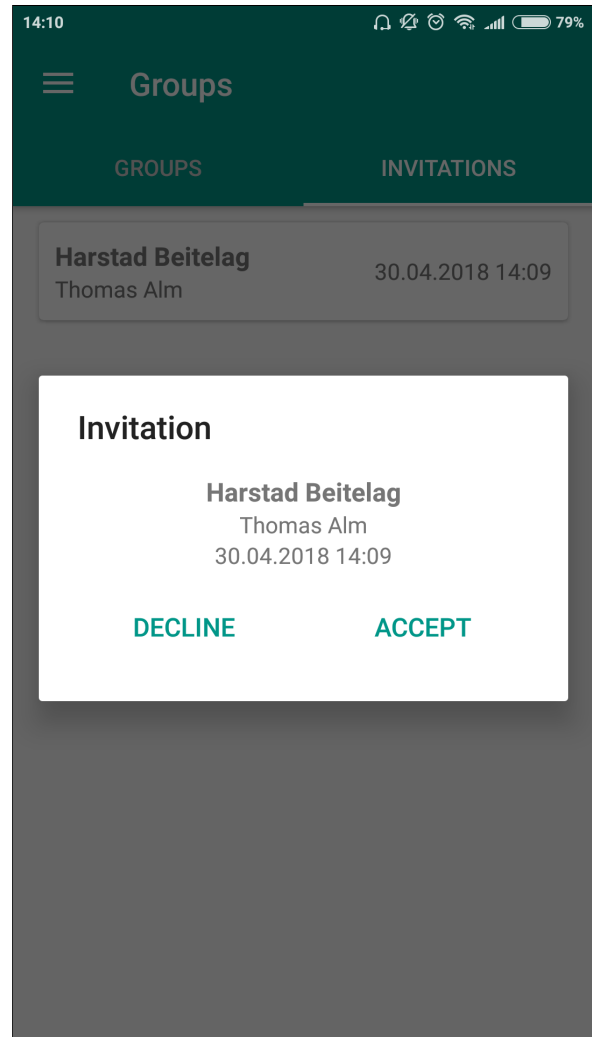


Figure 6.11: Invitation details dialog

6.4 Map Data

The map data component allows the user to manage and download maps. Initially, the user is informed that he has not downloaded any map areas yet. Once the user has downloaded a map, it will show up as a list item, as seen in Figure 6.12. The listed maps items have their own name, as well as additional information that can be useful to the user such as zoom level and size.

The purpose of the map component is to allow users to download maps that can be used when there is no internet connection. Since the application's primary environment is outdoors, where a reliable or fast internet connection cannot be expected, the user may rely on this functionality to a great extent. An internet connection is always required when downloading a map. This means that the maps should be download prior to the observation trips. Once a map is downloaded, it will be stored on the device and can be used regardless of internet connectivity.

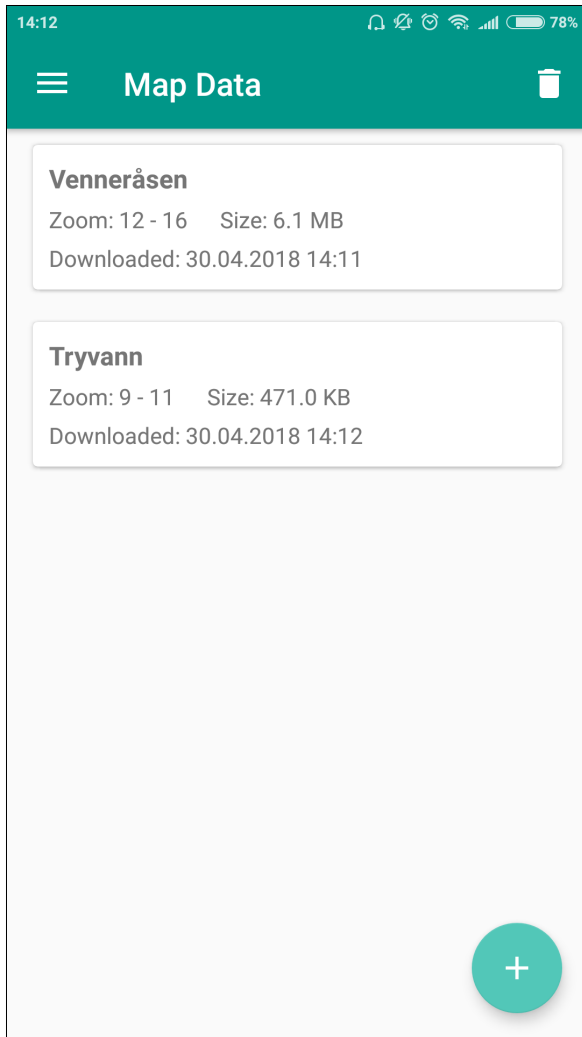


Figure 6.12: List of downloaded map areas

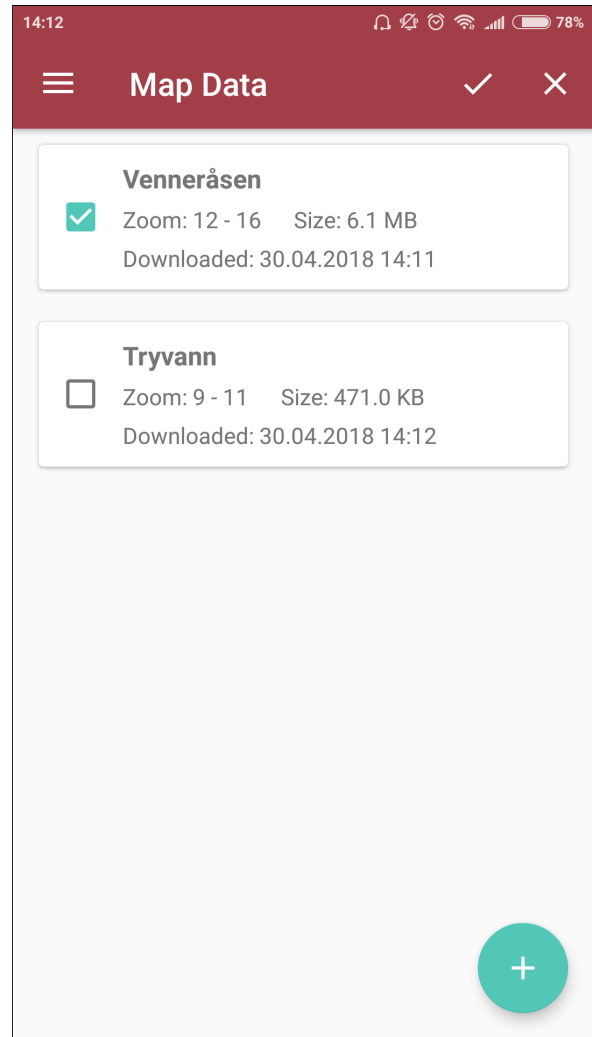


Figure 6.13: Map data delete mode

To delete a map, the user has to click the "delete" icon in the toolbar, which will put the component in a "delete" state. In this state, the map items can be individually selected and marked for deletion. The user can cancel or accept the deletion, by clicking the "cancel" icon or the "check" icon in the toolbar respectively. If the "check" icon is clicked, with one or more maps selected, the maps will be permanently deleted from the device. When a map is deleted, a pop-up menu will appear in the lower part of the screen for a short duration, as seen in Figure 6.14. This pop-up menu allows the user to undo the deletion of maps done by mistake.

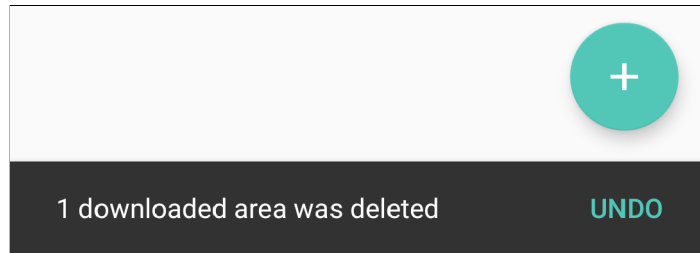


Figure 6.14: The undo pop-up menu

To download a new map area, the user can click the "add" button in the lower-right corner. This will take the user to another screen where he will specify the area that he wishes to download. The "Download Map" screen can be seen in Figure 6.15. The slider in the lower part of the screen allows the user to adjust the area that will be downloaded, from 1 km^2 to 15 km^2 . The blue square indicates what part of the map will be downloaded. The user can click on the map to change the downloaded map location. This will center the blue square on the clicked position.

The toolbar contains three icons. The "arrow" icon allows the user to navigate back to the initial screen. The "my location" icon centers the map on the user's current location. The "check mark" icon opens a dialog that allows the user to name the area to be downloaded, and adjust its zoom levels. The dialog is shown in Figure 6.16. Once the "download" button in the dialog is clicked, the user is taken back to the initial screen, and the download process will start. The map is stored on the device's internal or external storage and can be accessed freely. In the initial screen, the map being downloaded will be listed with a progress bar. Here the user can pause and resume downloads. Once the download is finished, the dialog will disappear and the map will be displayed in the list.

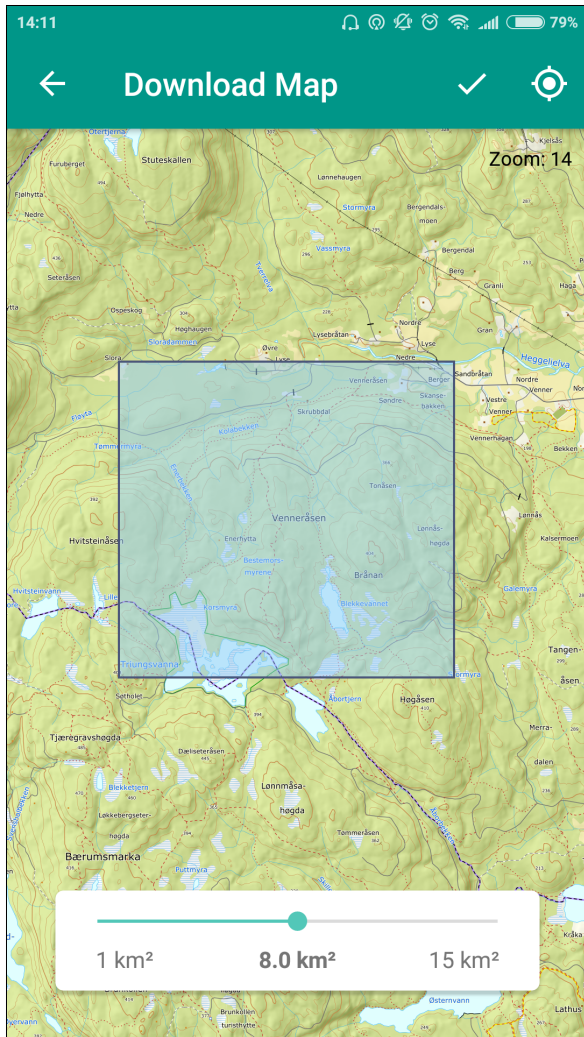


Figure 6.15: Map area download

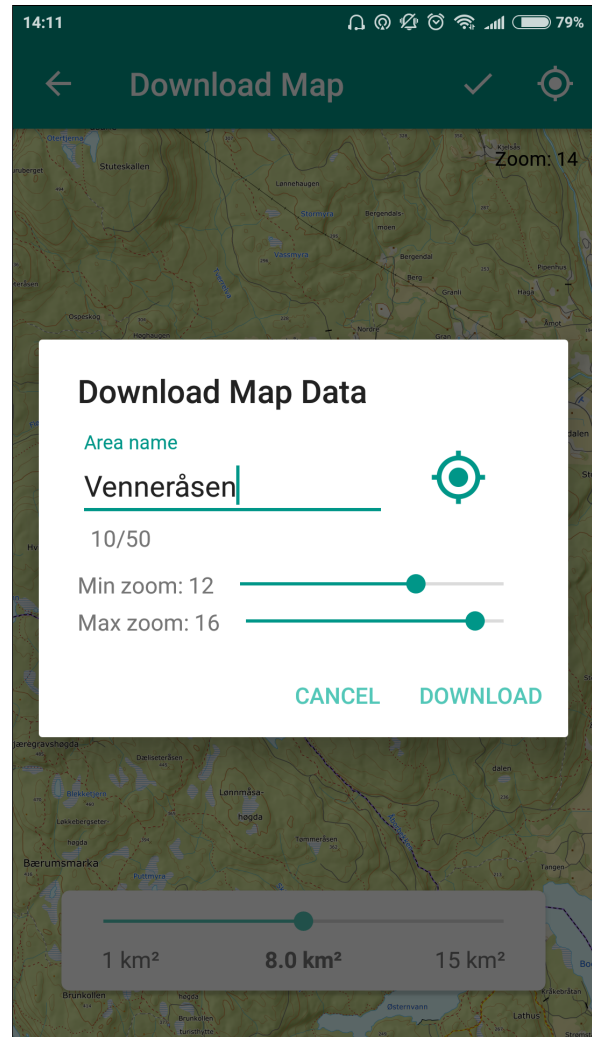


Figure 6.16: Download dialog

In the initial screen, it is also possible to click the listed map items. The user will be taken to a new screen that shows the map that is associated with the clicked item. Figure 6.17 shows an example of this screen. In the toolbar, there are two icons. The "arrow" icon allows the user to navigate back to the initial screen. The "pen" icon allows the user to edit the name of the downloaded map area through a dialog, as shown in Figure 6.18.

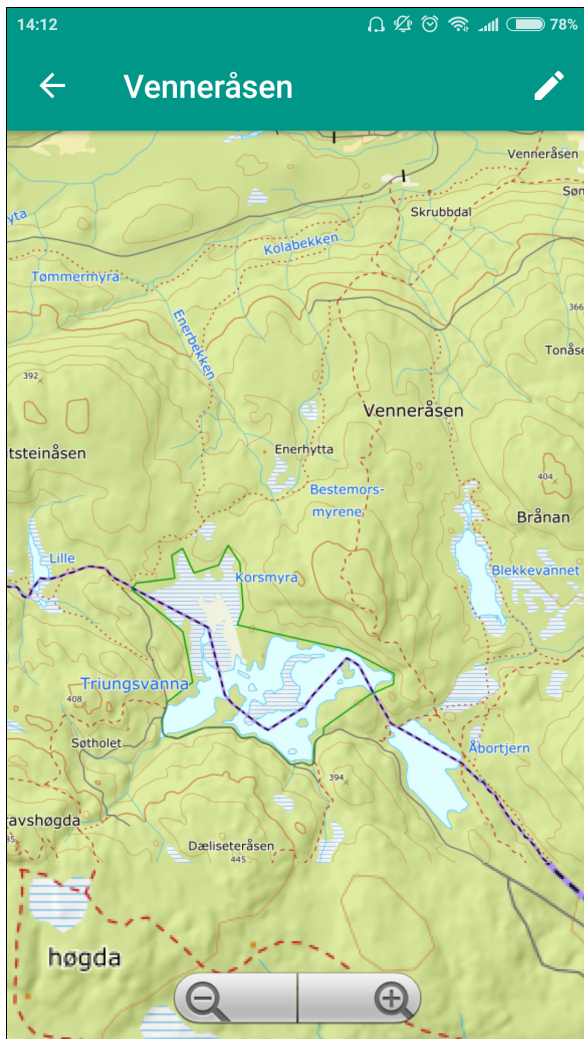


Figure 6.17: Downloaded map area

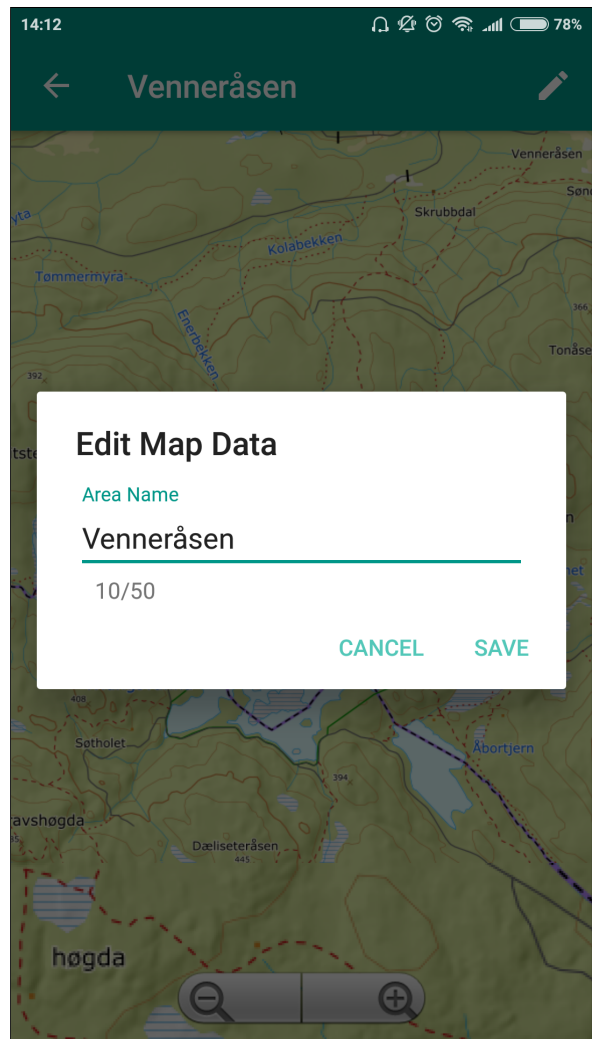


Figure 6.18: Edit name dialog

6.5 Trips

The trip component enables the user to perform and manage observation trips. In its initial state, the user is greeted with an image and a message that explains that no trips have been completed, as seen in Figure 6.19. However, trips that have been completed will be listed as clickable items. The same goes if the user has a trip in progress.

The component will distribute the trips into their associated group, and the user can easily navigate between the groups he is associated with. Figure 6.20 shows the trips that have been completed by the group "Oppland Beitelag", where two trips have been completed, and one trip is in progress.

Each listed trip item shows the name of the user that conducted the trip, the date, the number

of observations, the distance traveled, the trip duration, and three optional color boxes. The color boxes indicate whether a particular event occurred during that observation trip. If nothing unusual happened, the boxes would not be shown. The boxes are always in the same position, and can take on the following colors:

- **Orange** - A predator was observed.
- **Black** - A dead sheep or lamb was observed.
- **Red** - A wounded sheep or lamb was observed.
- **White** - This special event did not occur.

The component differentiates between trips that have been completed and the trip that is in progress. This is also indicated in the GUI, as the listed trip in progress item has a slightly yellow background, while the completed trip items have a white background. In addition, the "add/resume" button in the lower-right corner changes whether a trip is in progress or not. If a trip is in progress, it will be yellow and resume the in-progress trip once clicked. If no trip is in progress, the "add/resume" button will start a new trip instead. This ensures that each user can only have one trip in progress at the time. The reason behind this choice is that users should never have to walk two different trips at the same time, and should either complete or discard the current trip, before starting another. Having several trips in progress at the same time might also be confusing, as it can be hard to tell which trip is which.

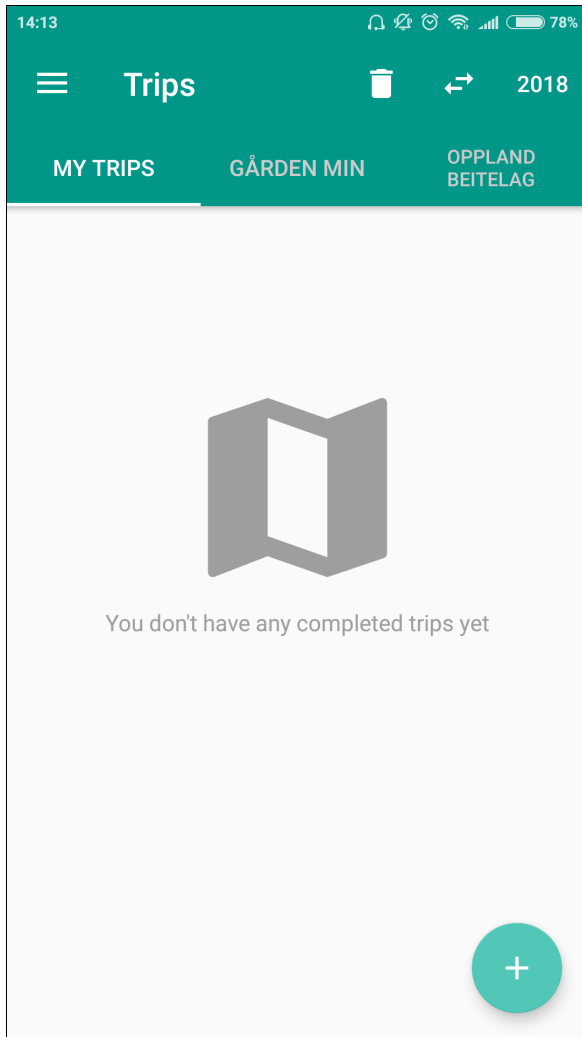


Figure 6.19: Initial trip component

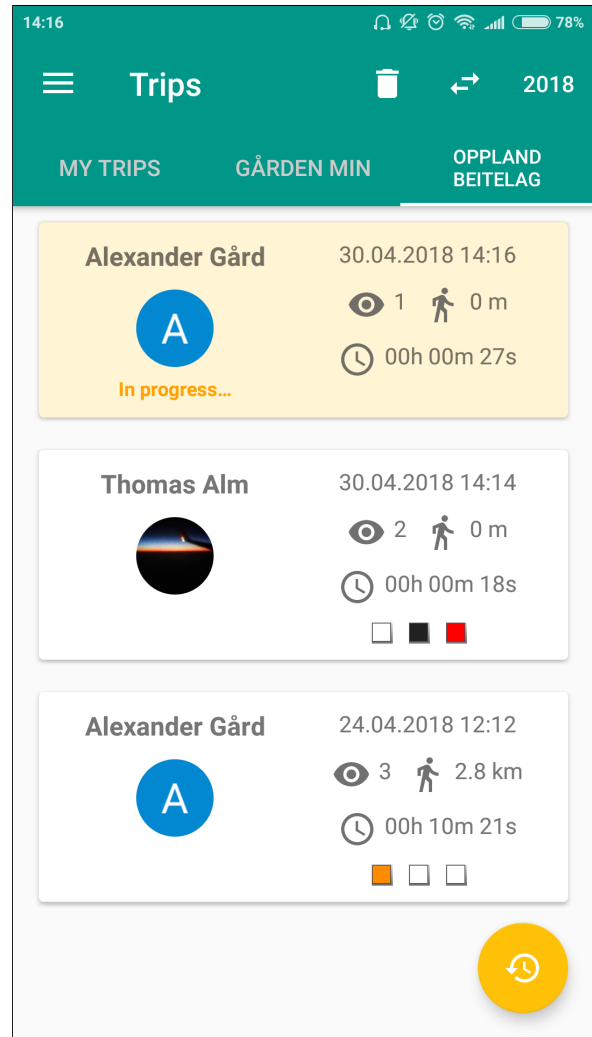


Figure 6.20: Group trips

To delete a trip, the user can click the "delete" icon in the toolbar. When this icon is clicked, the component is put into a "delete" state, as shown in Figure 6.21. In this state, the user can select a set of trips and click the confirmation "check mark" icon to confirm the deletion. However, users can only delete trips that have been started/completed by themselves. When a deletion is confirmed, a pop-up menu will appear for a short duration that allows the user to undo the action. This lets the user recover from deletions done by mistake. If the user clicks the "cancel" icon, the component will return to its "normal" state, and no trips will be deleted. The purpose of the delete action is to allow users to delete unnecessary trips or trips created by mistake.

To move a trip between groups, the user can click the "move" icon in the toolbar. This will put the component in a "move" state, as seen in Figure 6.22. The "move" state is similar to the "delete" state in some regards, as the same icons, restrictions, and undo pop-up menus will appear. The purpose of the "move" action is to let the users reorganize their trips, which could

be useful in cases where a user joins a group during a season. In this case, the user may have a set of completed trips that he now wants to share with the newly joined group. Another use case for the move functionality is if the user accidentally creates a trip in the wrong group, and wishes to move it to the correct group.

The component will automatically filter the listed trips, initially showing the trips of the most recent year. To see trips completed earlier, the user can click the "year" icon in the toolbar. This will open a dialog, where the user can select a specific year. The component will then list the trips completed in the specified year. The idea behind the filter function is to limit the number of listed trip items the user sees at once, since showing them all could quickly overwhelm the user with old and perhaps irrelevant information.

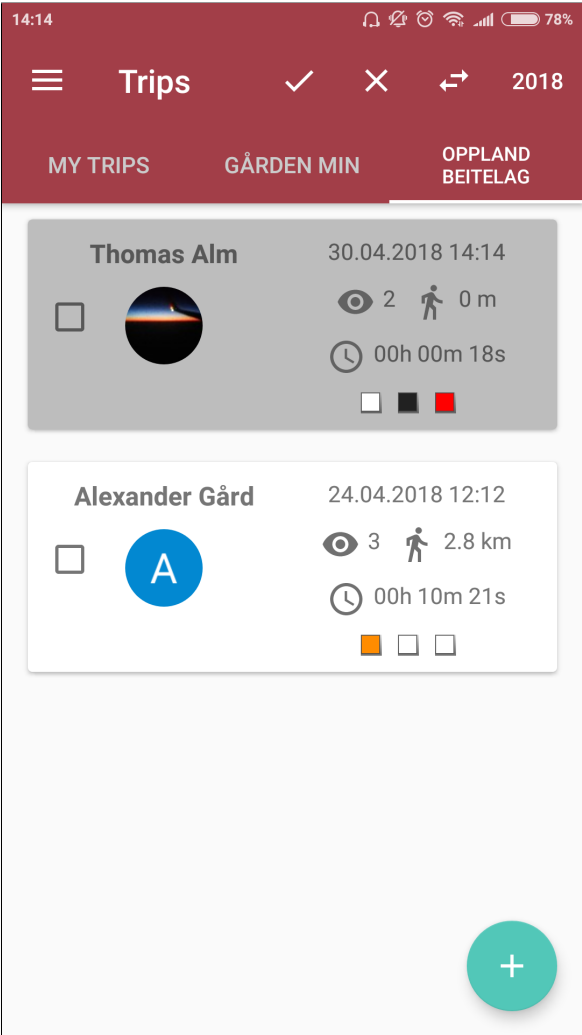


Figure 6.21: Delete state

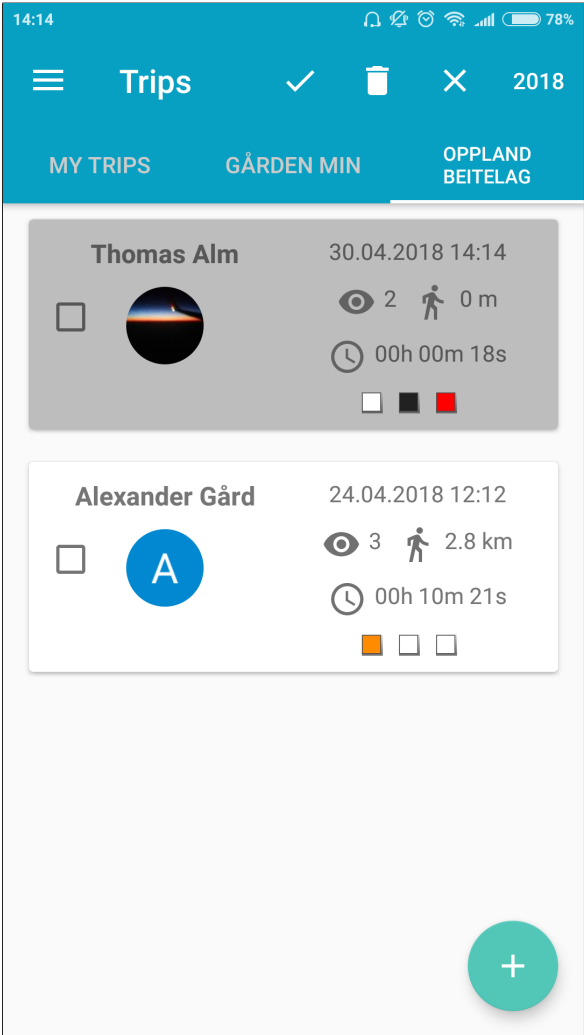


Figure 6.22: Share state

If the user does not have a trip in progress, he can create a new trip by clicking the "add/resume" button in the bottom-right corner. Creating a new trip will take the user to a screen that contains

a map. In this screen, the movement of the user is tracked, and through the map the user can register observations. This screen will also be shown when the user either clicks a trip in progress or a completed trip, from the initial screen. However, if the trip is completed, the user will not be able to add to or edit the trip, its observations and track. This is done to prevent accidental changes after a trip has been completed.

To start the observation trip, the user has to click the "start" button. This will open a dialog, where the user can select which group the trip should be associated with. Figure 6.23 shows the dialog, with the group "My Trips" selected. Once a group has been selected, the application will start tracking the location of the user. On the map, a start flag will be drawn on the first location registered.

To register a new observation the "new observation" button in the bottom-right corner must be clicked. This will bring up a dialog where information specific to the new observation can be registered. The dialog can be seen in Figure 6.24. The dialog consists of three tabs: general, scenario, and picture. Each tab will be explained later in this section.

The crosshair on the map depicts the center of the map. When registering a new observation, the center of the new observation will be set equal to the center of the map. This means that crosshair can be used to place accurate observations. A line will also be created between the observation and the current location of the user.

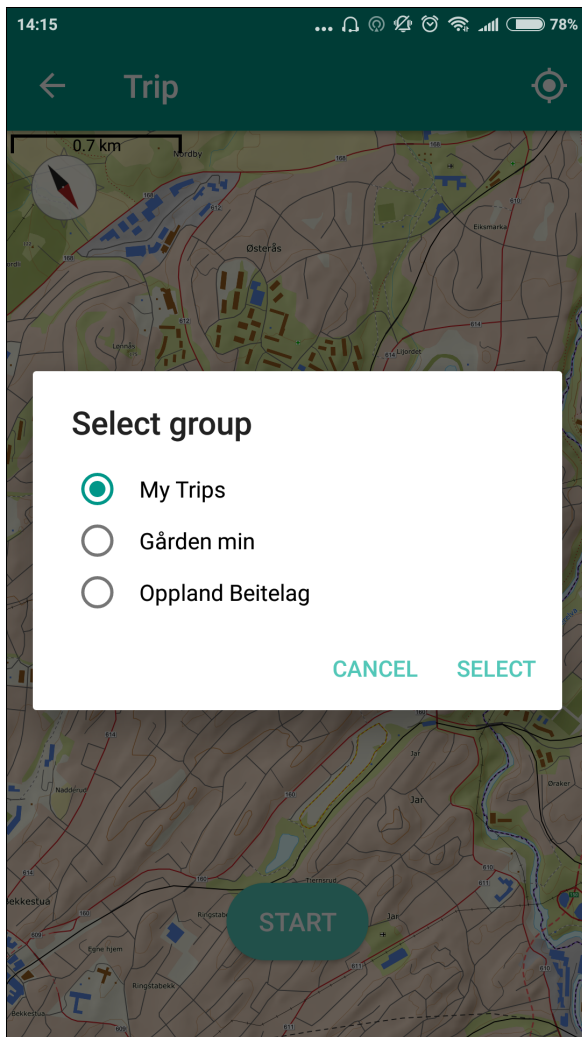


Figure 6.23: Start trip dialog

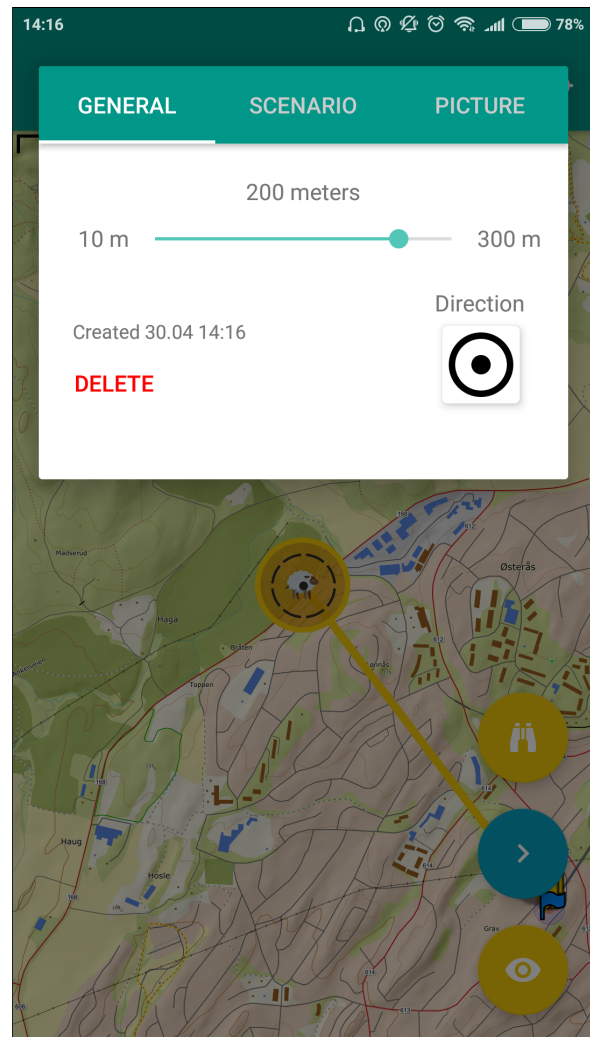


Figure 6.24: General Tab

In the observation dialog, the "general" tab is selected by default. In this tab, the user can use a slider to set the area for the observation. When the slider is incremented or decremented, the circle area of the observation is adjusted accordingly. The tab also allows the user to set the movement direction of observed scenario, with the default value being stationary. Clicking the direction button will open an additional dialog, where a direction can be selected. It is also possible to delete the observation through the "delete" button.

In the "scenario" tab, the user can specify the observed scenario. The user can select between the following main scenarios: sheep, lamb, predator, wool, reindeer, dog, and other. In each scenario, the user can fill out the optional predefined fields to add further details to the observation. In addition, some scenarios have sub-scenarios that are used to specify the main scenario further. These sub-scenarios are used to decrease the number of items needed in the main scenario drop-down menu. Too many items could overwhelm the user, and make it more

difficult to navigate the dialog. Figure 6.25 shows the default scenario "sheep", its default sub-scenario "sheep herd", and some of the optional fields the user can fill out. Figure 6.26 shows the "predator" scenario selected, and its selectable sub-scenarios.

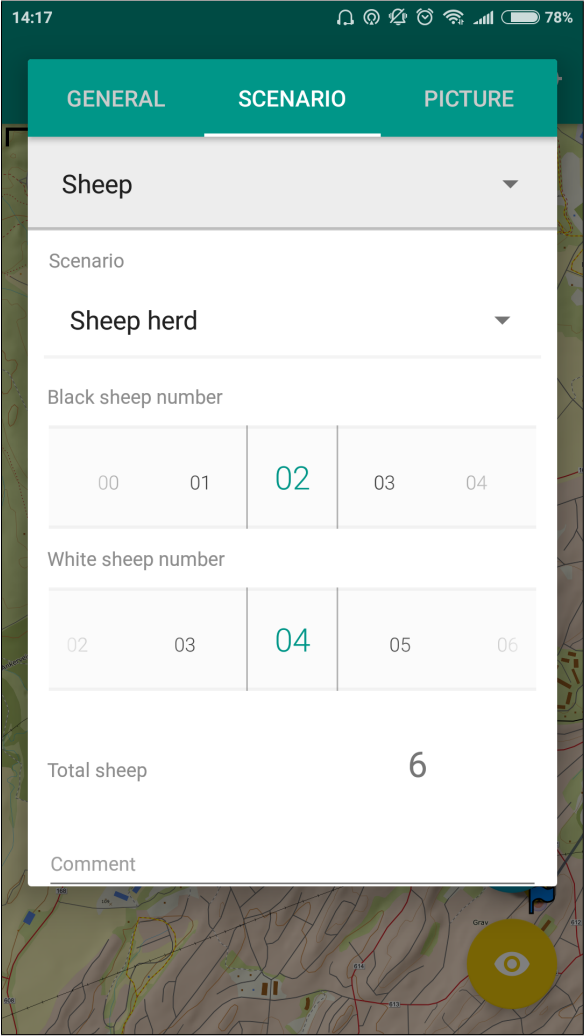


Figure 6.25: Scenario Tab 1

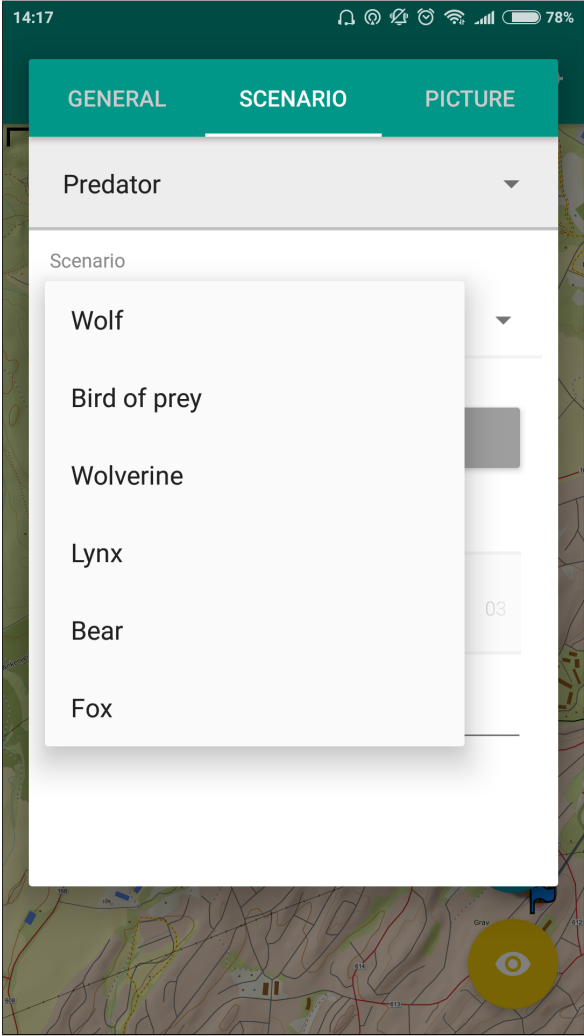


Figure 6.26: Scenario Tab 2

In the "camera" tab, the user can attach pictures to a specific observation. The user can either capture a new picture or load a picture from the device storage, using the "take picture" and "load picture" buttons respectively. These actions will open third-party applications, which will be different from device to device. Figure 6.27 shows the "camera" tab with three pictures. To delete pictures the "delete" icon must be clicked. The tab will be put in a "delete" state, where pictures can be selected for deletion. Clicking the "check mark" icon will confirm the deletion. Figure 6.28 shows the "camera" tab where one of the three pictures have been selected for deletion. To get a better view of the pictures, the user can click on the individual pictures when the tab is in the normal state. This will open a third-party application that shows the

picture in greater detail.

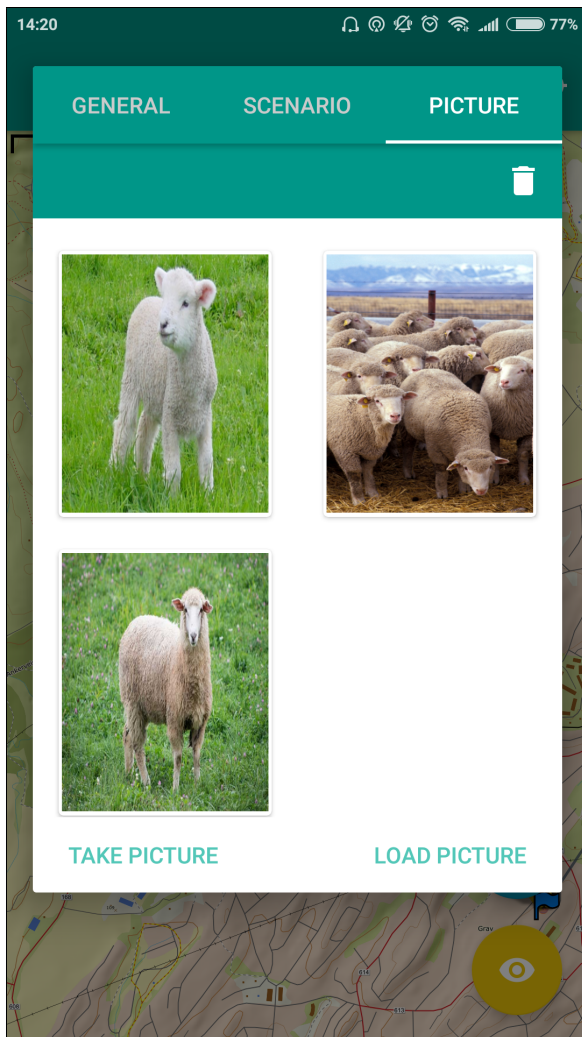


Figure 6.27: Camera tab

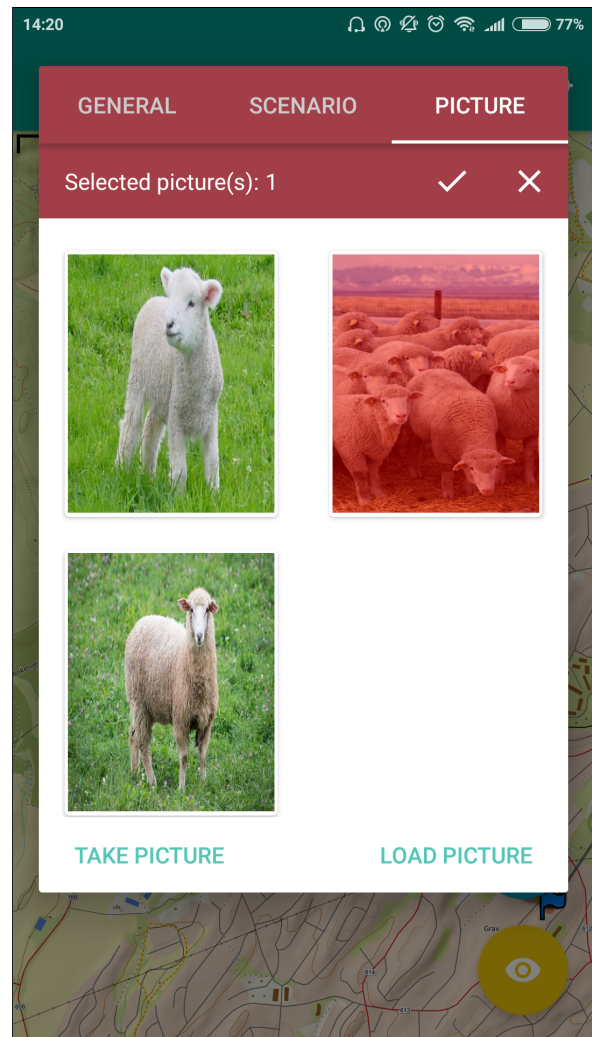


Figure 6.28: Camera tab in delete state

The trip component has a set of buttons that is designed to make it easier to interact with the observations on the map. The "closest observation" button seen in Figure 6.30 allows the user to navigate to the closest observation, based on the current map center. Whenever an observation is in the center of the map crosshair, it will be highlighted as shown in Figure 6.29. Once an observation is highlighted, the "new observation" and "closest observation" buttons are replaced by three other buttons.

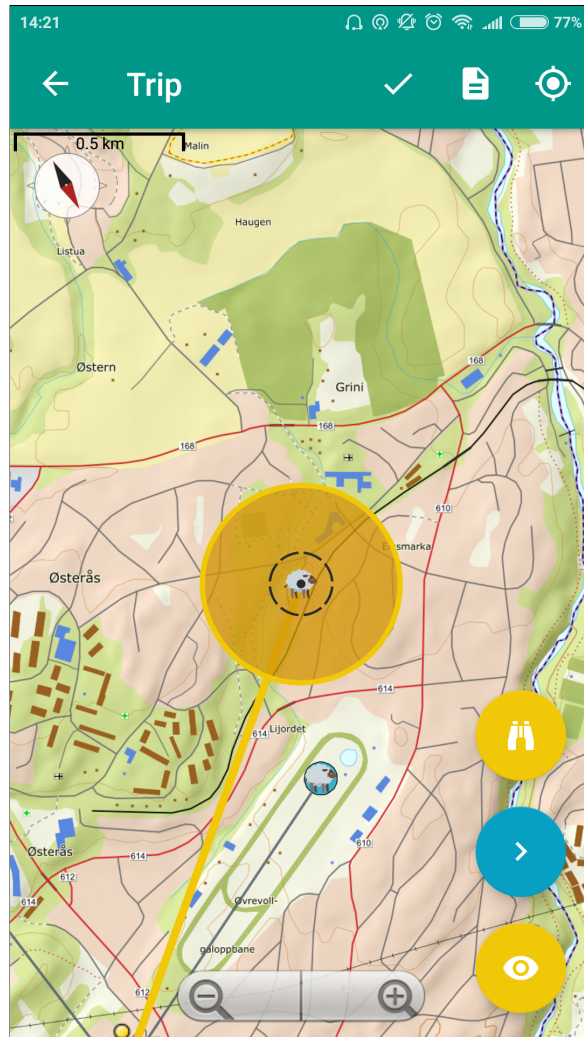


Figure 6.29: Observation highlight

The "open observation" button seen in Figure 6.32 opens the observation dialog of the highlighted observation. The "next observation" button seen in Figure 6.31 navigates to the next observation, in the order they were registered. The "link observation" button seen in Figure 6.33 "links" the highlighted observation with the current user location, allowing observations to be observed from multiple locations. Figure 6.29 shows a highlighted observation with two lines, which means that the observation has been observed from two locations. To delete the "link" lines, the user can click on the lines to bring up a "delete" dialog. Here the user can either confirm or cancel the deletion. However, an observation must have at least one line connected to it at all times; thus the user cannot delete the line of an observation that has only been "linked" to one location.



Figure 6.30: Closest observation button



Figure 6.31: Next observation button



Figure 6.32: Open observation button



Figure 6.33: Link observation button

The toolbar of the component contains four icons, which are labeled in Figure 6.34. The "arrow" icon takes the user back to the initial screen of the component. This will not complete the trip, but will instead keep the trip in progress. The "check mark" icon will complete the trip, which will also take the user back to the initial screen. A stop flag will also be drawn at the last location of the trip. In either case, the trip will be listed on the initial screen, as either an in-progress or completed trip. The "description" icon will open the "trip details" dialog, which can be seen in Figure 6.35. In this dialog, the user can submit additional information specific to the in-progress trip, i.e., start location, end location, weather, participants, and a comment. The "my location" icon will center the map on the current location of the user.



Figure 6.34: Trip toolbar

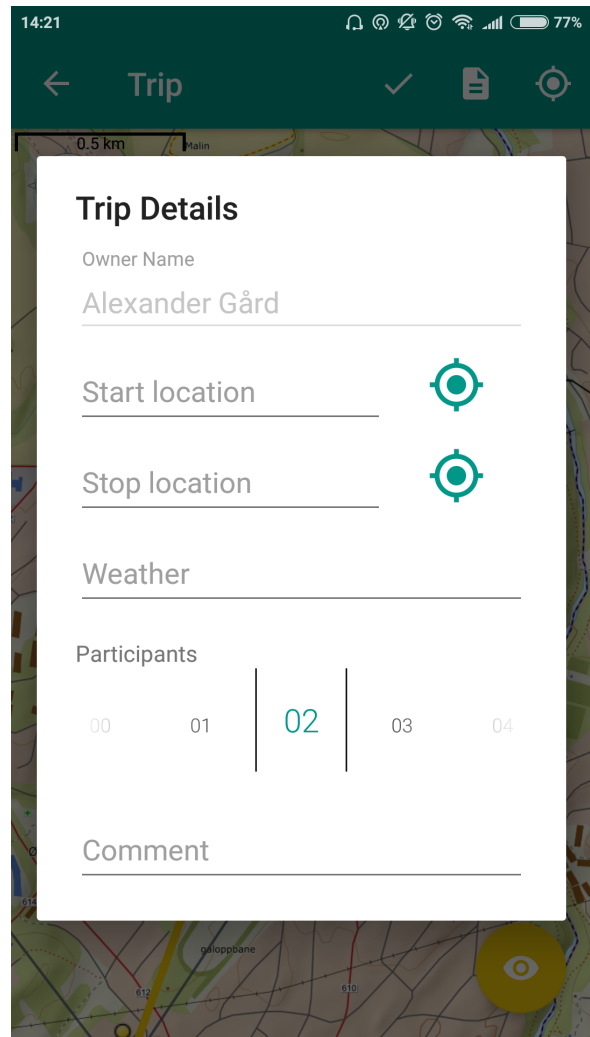


Figure 6.35: Trip details dialog

6.6 Timeline

The timeline component was designed to give a quick overview of all the trips completed by a group for a specific year. The user can traverse through the trips, which will show the individual trips and their observations on the map. The timeline component shares much of the same functionality as the trips component. However, new trips cannot be started or edited from the timeline. This section will focus on the features that are specific to the timeline. Figure 6.36 shows the timeline component.

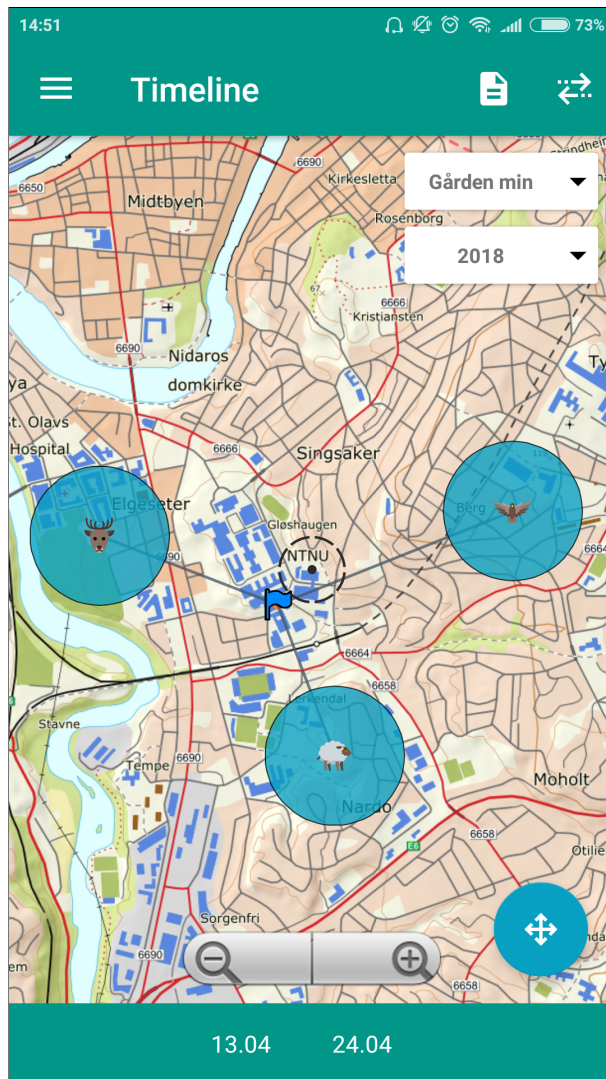


Figure 6.36: The timeline component

The component has two drop-down buttons in the top-right corner that allows for a specific group and year to be set. The trips that are associated with the selected values will then be loaded into the component. Beneath the map, there is a selection of dates that mirrors the completion of the associated trips. Once a date is selected, the component loads the trip into the map, showing its observations, tracks, etc. If multiple trips are completed on the same date, then they will be listed based on their exact time of completion. If no trips are found for the specified year and group, the map will be replaced by a text message and image that explains that no trips were found.

The toolbar at the top of the screen contains three icons. The "hamburger" icon brings up the navigation drawer. The "description" icon brings up the "Trip Details" dialog, which contains information about the trip being displayed. The "arrows" icon sorts the trips in either ascending

or descending order, in terms of their completion date/time.

6.7 Reports

The reports component generates and displays reports based on the user's associated observation trips. In its initial state, the component will show an image and text that explains that no reports have been generated. The "create" button in the lower-right corner can be clicked to generate a new report. This will bring up a dialog where the year and group can be set. The selected values are used to determine which trips will be a part of the generated report. When the report is generated, it will be listed under the tab of the specified group. Each listed report contains a summary of the most important information, to give the user a quick overview of the report's contents. Figure 6.37 shows the report component with one report in the list.

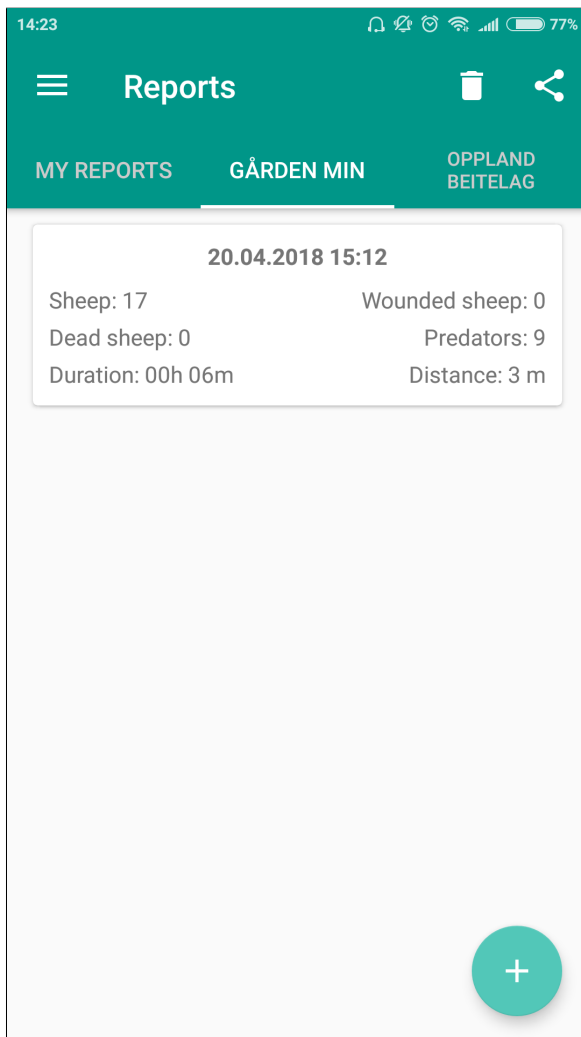


Figure 6.37: Reports Component

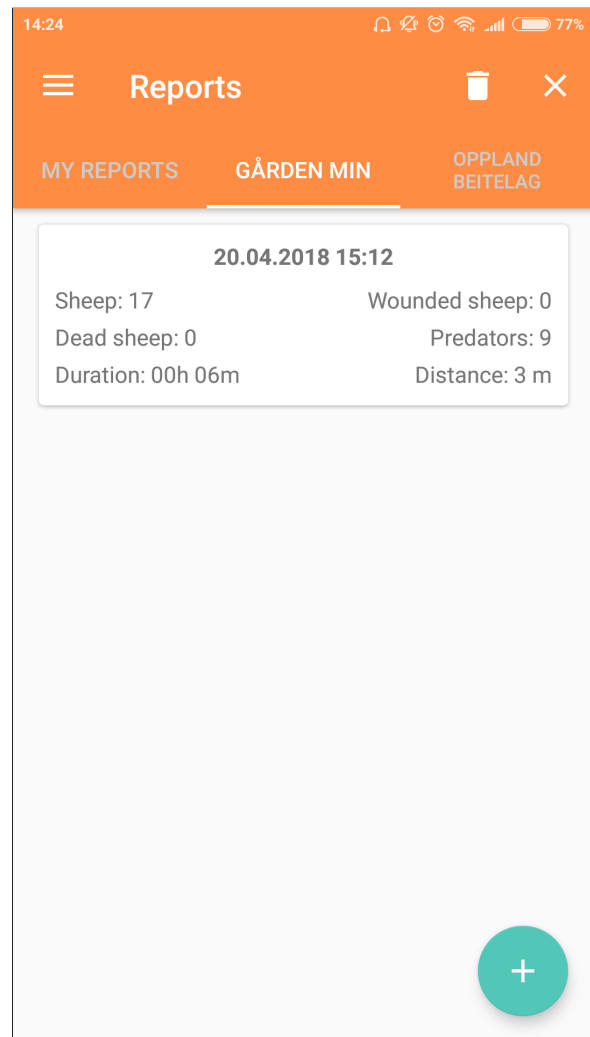


Figure 6.38: Reports share mode

It is also possible to share and delete the reports through the toolbar icons. If a report is deleted

through the "delete" icon, a pop-up menu with an undo button will appear for a few seconds. If the "share" icon is clicked, the component will go into the "share" state. The "share" state is visualized in Figure 6.38. If one of the listed reports are clicked when the component is in this state, a dialog will appear. The dialog offers several methods for sharing the clicked report, such as email, Bluetooth, and Google drive, depending on which apps the user has installed on his device. The purpose of the sharing functionality is to make it easier for farmers to share reports with the government or between themselves.

If a listed report is clicked when the component is the "normal" mode, then the report will be opened in a spreadsheet app, chosen by the user. The report consists of two sheets. The first sheet lists all trips and information from each of them. The second sheet contains a summary of the report. Figure 6.39 shows an extract of the report spreadsheet with one trip per row, which totals to three trips.

Sheep	Wounded sheep	Dead sheep	Predators
5	1	2	2
2	0	0	0
8	1	8	1

Figure 6.39: Spreadsheet extract

6.8 Profile

The profile component displays information related to the user's account and provides functionality for changing avatar image and username. If the user has created his account through Google or Facebook, the user's default avatar image will be provided by Google or Facebook respectively. If the user has created his account with an email and password, the avatar image will be a placeholder image until it is changed to something else. The date when the account was created is also displayed. Figure 6.40 shows an example profile.

The avatar image of the user can be changed by clicking the "change" button. Clicking the button will bring up a third-party app where an image stored on the user's device can be picked. After selecting the image, it will be rescaled and uploaded to the Firebase Storage.

The username can be changed by clicking the other "change" button, next to the displayed username. This will open a dialog, where the user can type in his new name. Figure 6.41 shows

the "change username" dialog.

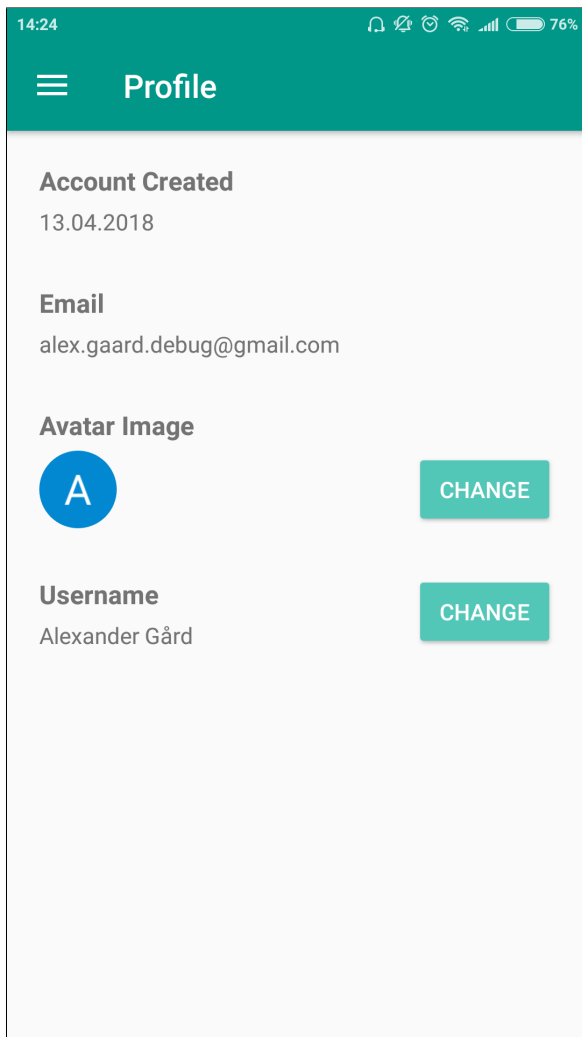


Figure 6.40: Profile

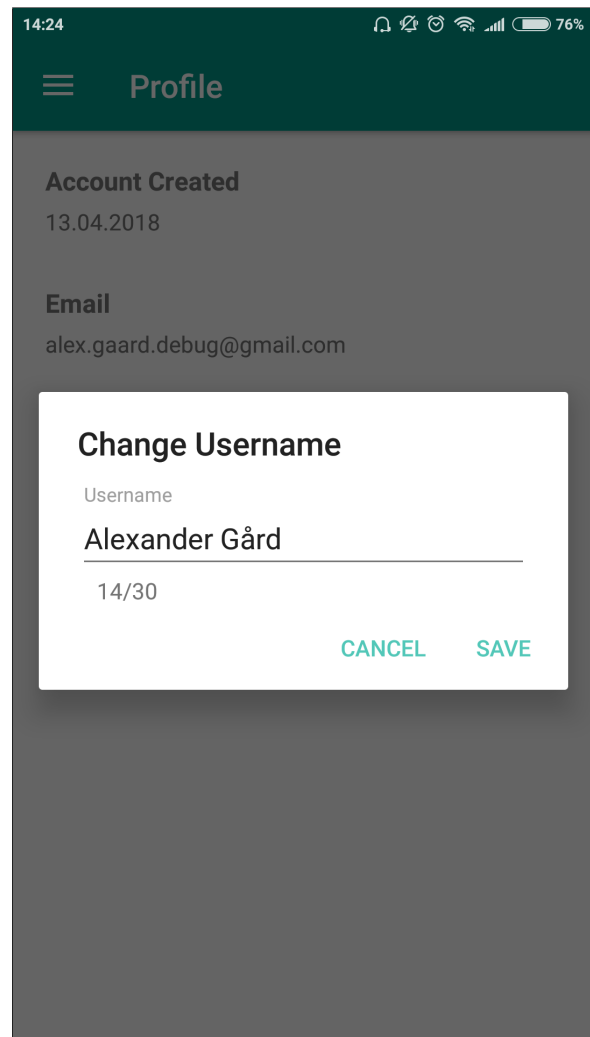


Figure 6.41: Change username dialog

6.9 Settings

The settings component contains customizable settings that the user can tweak according to his preferences. The component is shown in Figure 6.42, and has the following settings:

- **Language** - Changes the application language. The supported languages are Norwegian and English. Clicking this setting opens up a dialog where the language can be selected.
- **Map Data** - Clears out cached map data from Osmdroid. This will not affect the downloaded map areas. Clicking this setting opens up a dialog where further details are given to the user. From here the user can either delete the cache or cancel the operation.
- **Mobile Network** - Enables or disables the usage of mobile network for downloading map data. This setting is turned off by default to prevent unexpected costs.

- **Map Rotation** - Enables or disables the user's ability to rotate the map. This setting is turned off by default as it is easy to accidentally rotate when trying to zoom or pan.
- **Distance Interval** - Allows the user to set how often Pecora will track his location when performing trips. Having a low interval will give a detailed track of where the user has moved during his trip, but it might cause the track to become less accurate when standing still, as the location can shift around. Setting too large an interval might also cause problems, as the track might lack the details that the user desires. This setting is set to 25 meters by default, which gives a sufficient balance between accuracy and details. When the user clicks the setting, a dialog with different interval options is shown, as seen in Figure 6.43.

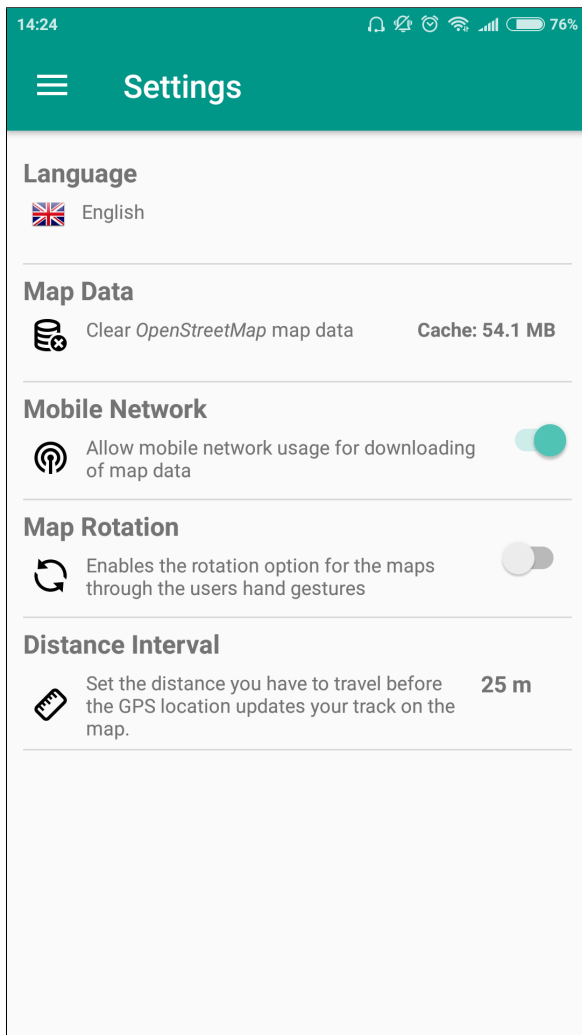


Figure 6.42: Settings

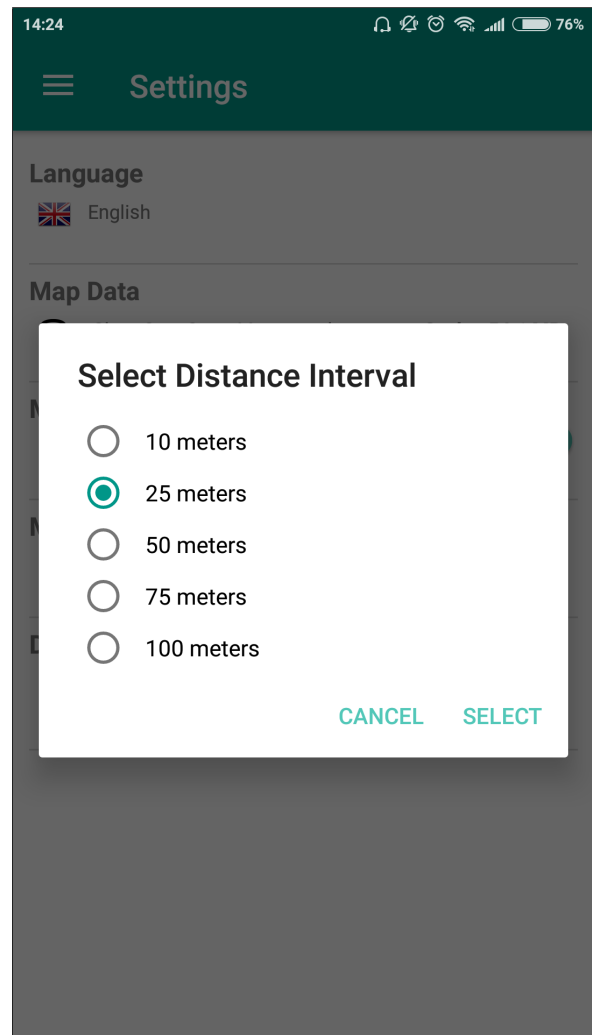


Figure 6.43: Select distance interval dialog

6.10 Help

The help component is relatively simple and contains the user manual for Pecora. The user manual comes in the form of a PDF document and is designed with the purpose of showcasing what the application is capable of and how it can be utilized. The component is shown in Figure 6.44.



Figure 6.44: Help

Chapter 7

A Brief Introduction To Android

This chapter will give a brief introduction to Android activities, fragments, dialogs, and the activity lifecycle. The introduction is provided to the reader with the intent of making it easier to understand the architecture and the implementation chapters of the thesis.

7.1 Activity

Activities provide a window where the application can draw the user interface. Activities are a single, focused window that allows the user to interact with the application. Typically, the windows are full-screen, but they can also be smaller and float on top of other windows. It is common practice to have each screen implement one activity. Applications usually contain several screens, thus containing several activities. Each activity can launch other activities to perform different actions. For example, an activity with a to-do list can launch another activity that is responsible for adding or editing new items for the to-do list. While the activities are working together to form a cohesive user experience, they are loosely bound to each other, and there are minimal dependencies between them [31].

7.2 Fragment

A fragment represents either a behavior or a piece of the user interface in an activity. Multiple fragments can be used in one activity, and fragments can be reused in multiple activities.

Fragments can be thought of as a modular part of the activity’s user interface, which has its own lifecycle and receives its own inputs. In other words, it can be considered a sub-activity of sorts. Fragments are always embedded in an activity, and the fragment’s lifecycle is directly connected to its host activity’s lifecycle. For example, when an activity is paused the connected fragments are also paused. However, it is possible to manipulate fragments, such as adding or removing them, when the host activity is active. It is also possible to have a fragment without a user interface that functions as a worker for the host activity [32]. Figure 7.1 shows the relations between activities and fragments, and how the same activities and fragments can be utilized differently on different devices.

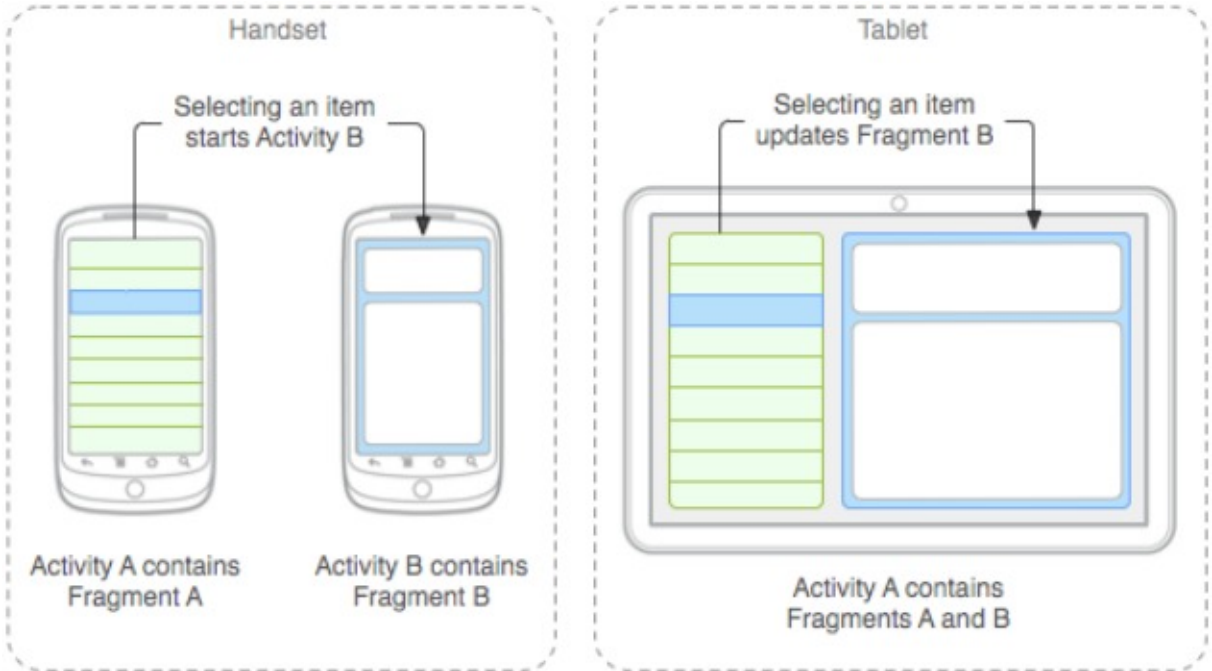


Figure 7.1: Relations between activities and fragments

7.3 Dialog

A dialog is a window that floats over an activity or fragment and does not take up the entire screen. The dialog prompts the user with the request for additional user input or displays a message. The user usually has to comply with the dialog input request or close the dialog to proceed [33]. Figure 7.2 shows an example of two Android dialogs.

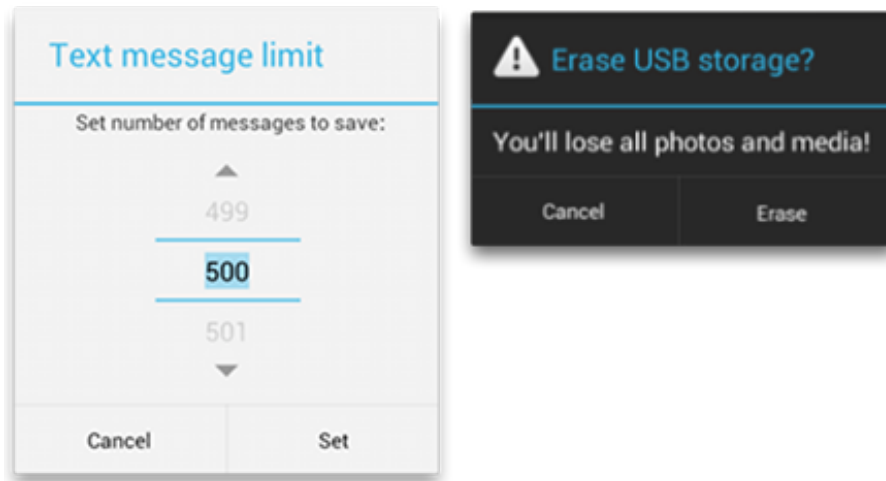


Figure 7.2: An example of two Android dialogs

7.4 Android Lifecycle

When a user navigates through an application, it transitions through a different set of lifecycle states. The activities, fragments, and dialogs of the application provide several callback functions that allow the application to capture state changes, e.g., when an activity is created, stopped, resumed, destroyed. It is possible to define custom behavior for the activity, fragments, or dialogs inside the lifecycle callback functions. When the user interacts with the application, these custom behaviors will be executed. For example, an application that is designed to stream videos might want to pause the video if the user switches to a different application. It might also want to resume the video from the same spot once the user returns to the video streaming application. Being aware of lifecycle events and handling them properly will make your application more robust and performant [34] [32].

7.5 Gradle

Gradle is a common build system used when developing Android applications. Gradle compiles the app resources, source code, and packages them into APKs that can be tested, deployed, signed, and distributed. It is normal for an Android project to have at least two *build.gradle* files. One *build.gradle* file acts as a parent for all the sub-modules of the project and contains configuration options that are shared among them. Each sub-module also has its own *build.gradle* file that contains information specific to the given module. The *build.gradle* file for the specific module is among other things, used to declare dependencies or define configurations [35].

7.6 The Manifest File

All Android applications are required to have an *AndroidManifest.xml* file. The manifest declares critical information about the application, the Android OS, Google Play, and the Android build tools. The manifest declares the application's package name, activities, and application permissions, among other things [36].

Chapter 8

Architecture

This chapter focuses on the architecture of Pecora, and consists of the architecture for both the system and the application. The system architecture captures the different parts of the system, and the application architecture focuses on the project structure and software architecture of the Android application.

8.1 System Architecture

The system architecture in Pecora is simple, as it only consists of two different parts. The first part is the Pecora application which runs on an Android device. The second part is the Firebase services used by the application.

Some Firebase services require an internet connection, such as crash reporting, authentication, and the Firebase Storage. However, The Firebase Realtime Database can be used regardless of internet access. When the user is not connected to the internet, the local database will be used until Pecora can reconnect to the cloud. After reconnecting, all changes that have been done locally will be sent to the cloud, and all changes done by other users will be retrieved. This ensures that the local version is up-to-date when the user goes offline again.

Figure 8.1 shows a deployment diagram of Pecora, which describes the physical deployment of Pecora. In the diagram, the APK-file is created by packaging the resources used by Pecora such as icons, layouts, and localized text together with the compiled code. After being built,

the APK-file is then installed in an Android environment, on the user’s device. The APK can either be installed directly on the device’s internal storage, or on an external storage, such as an SD-card.

The diagram also shows the physical separation of the different Firebase services. The Firebase Cloud Services includes all Firebase services that are used by Pecora. Since the Firebase Realtime Database is also used locally on the device, it is shown in the diagram as two separate entities.

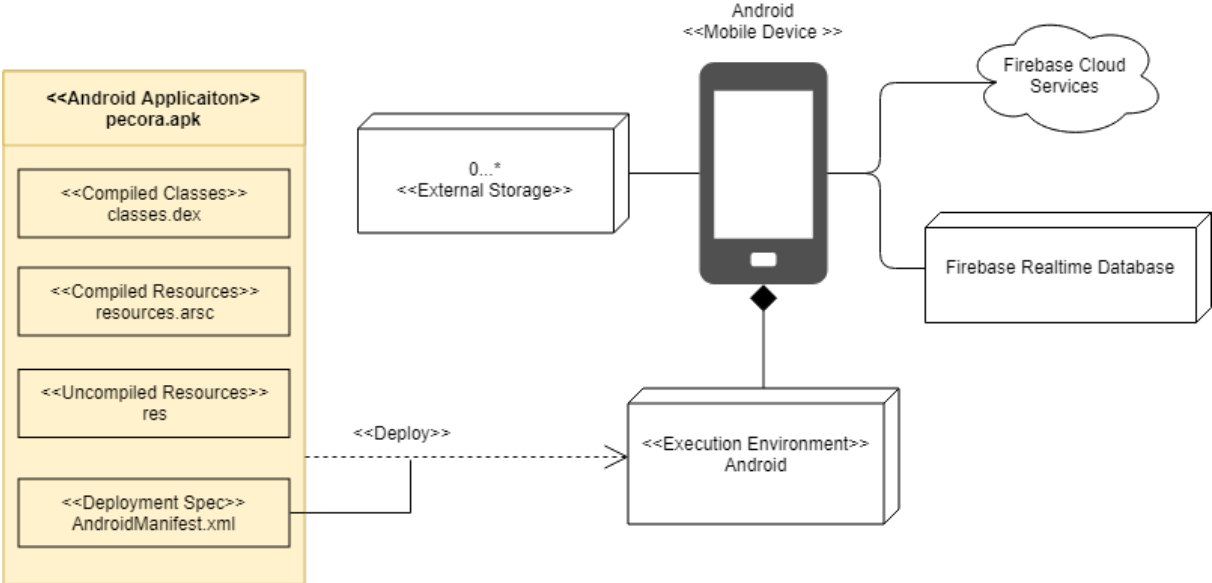


Figure 8.1: Deployment diagram

8.2 Application Architecture

This section presents the architecture used in the Pecora Android application. This includes both the structure of the project, as well as the architecture for core software components.

8.2.1 Project Structure

This section focuses on how the project is structured. Pecora is structurally divided into three overarching parts: source code, resources, and configuration files.

8.2.1.1 Source Code

The source code of Pecora consists of over 33400 lines of code, divided across 493 files. The code is divided into three parts: the code used by the application during runtime, code for unit tests, and code for GUI tests. Figures 8.2, 8.3, and 8.4 consists of package diagrams which depicts the relationship between the packages in the source code for each part respectively. The relationship to the resource package is also included in Figure 8.2, but will be further discussed in Section 8.2.1.2. Unit tests and GUI tests are further explained in Section 10.2 and 10.3 respectively.

The following list briefly describes the packages seen in Figure 8.2, which visualizes the dependencies in the source code used by the application during runtime.

- **application** - Contains code used throughout the entire application. This includes global constants and code for initializing the application on startup.
- **components** - Contains the different components used in the application, such as profile, timeline, help, etc.
- **data** - Data abstraction layer responsible for storing and retrieving data. It also contains the data models used by the application.
- **handlers** - A handler is defined in this project as a class that is responsible for a specific area, and is usually implemented as a singleton instance. For example, the *LocationHandler* handles retrieving the user's location, and configuring location settings.
- **utils** - Contains different types of classes such as helper classes or useful abstractions, such as the *BaseViewModel* class.
- **dagger** - Responsible for dependency injection.
- **res** - Contains resources such as layouts, animations, images, and localized text.

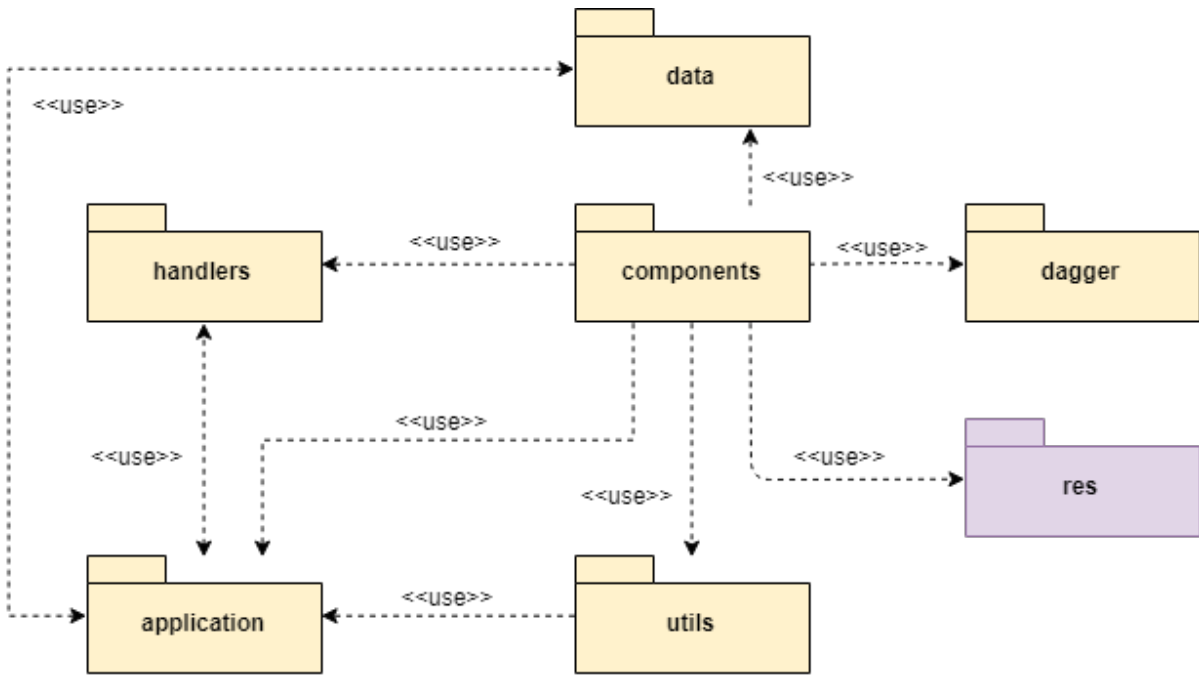


Figure 8.2: Application package diagram

The following list briefly describes the packages seen in Figure 8.3, which visualizes the dependencies in the source code used by GUI tests.

- **ui** - GUI tests.
- **utils** - Utility classes for performing tests. This includes helper classes and classes for providing test data.
- **components** - As previously mentioned.
- **dagger** - As previously mentioned.
- **data** - As previously mentioned.

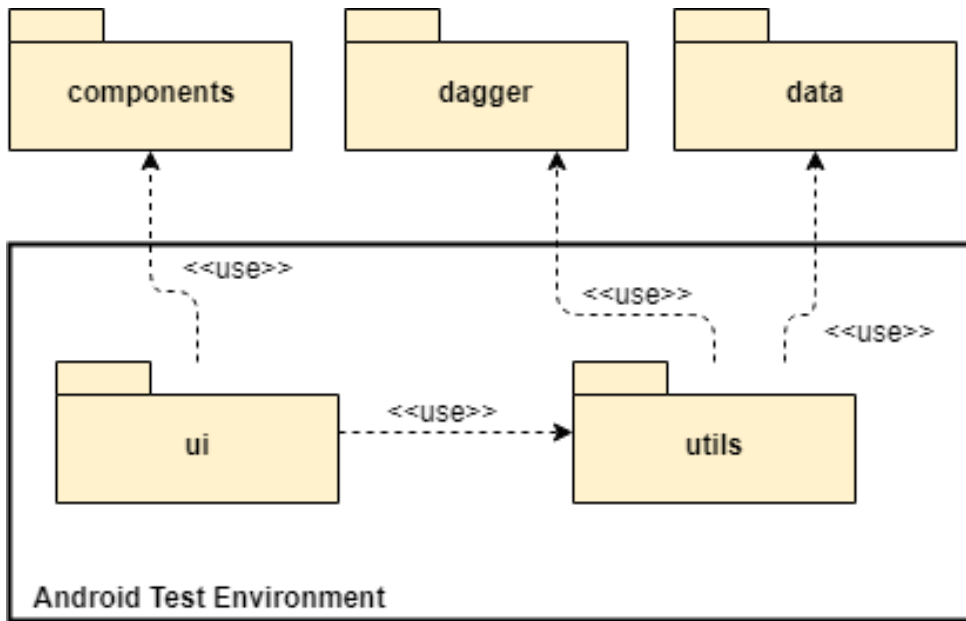


Figure 8.3: GUI test package diagram

The following list briefly describes the packages seen in figure 8.4, which visualizes the dependencies in the source code used by unit tests.

- **unit** - Unit tests.
- **utils** - Utility classes for performing tests. This includes helper classes and classes for providing test data.
- **component** - As previously mentioned.
- **data** - As previously mentioned.
- **dagger** - As previously mentioned.

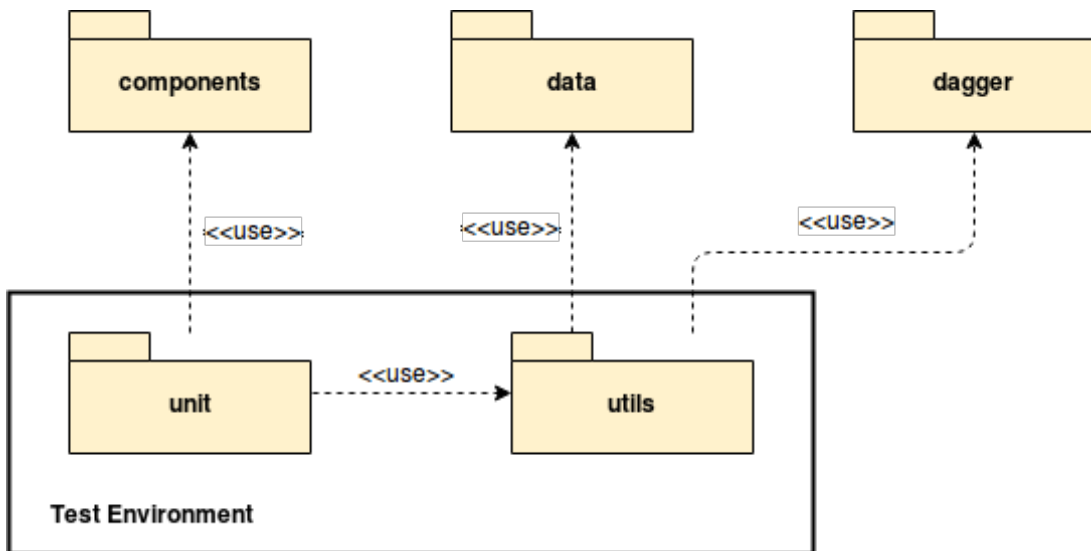


Figure 8.4: Unit test package diagram

8.2.1.2 Resources

The resources in Pecora is divided into different folders depending on what they contain. This structure is provided by default when the project is first created with Android Studio. Figure 8.5 shows an overview of where the different resources are located. The following list briefly explains each resource folder.

- **anim** - Contains animations and transition effects.
- **drawable** - Contains icons and images, as well as custom shapes.
- **layout** - Contains layout files for activities, fragments, dialogs, dialog fragments, and custom views.
- **menu** - Contains layout files for menus.
- **mipmap** - Contains app launcher icons.
- **values** - Contains mappings between variables and values. This includes the following items.
 - **colors** - Contains the color palette.
 - **dimens** - Contains view dimensions.
 - **strings** - Contains all the text strings that are displayed in the application. Each supported language has its own strings file.
 - **styles** - Contains styles and themes used by views.
- **xml** - Contains miscellaneous XML-files such as network security configuration or file paths.

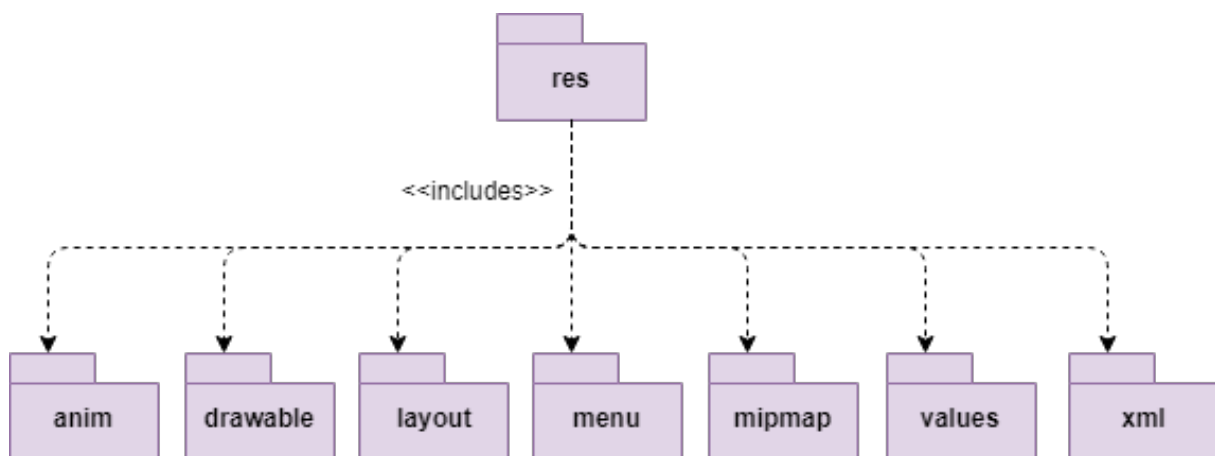


Figure 8.5: Resource structure in Pecora

8.2.1.3 Configuration Files

There are two different configuration files used in Pecora. The first one being the Android manifest. The manifest describes among other things, which permissions the application requires to function properly, and which activities, services, and styles are being used. The manifest can be found inside *app/src/main/AndroidManifest.xml*.

The other configuration file is the *build.gradle* file. This file can be located under the *app/* folder and contains configuration parameters used when building, testing, and debugging the application. The file also includes dependencies to other libraries, build/version number, minimum and maximum SDK-version needed for installing the application, among other things.

8.2.2 Software Architecture

The focus in this section will be on the software architecture and it will give a high-level overview of the several core software components used in Pecora. For further details regarding the implementation of the different components, see Chapter 9.

8.2.2.1 Model View View-Model

As mentioned in Chapter 3 - Preliminary Study, Pecora uses MVVM as it's software architecture pattern. This has heavily influenced how the project was structured, and the majority of the software components were written with the MVVM pattern as their foundation.

The model is represented by the *FirestoreManager*, which takes care of synchronizing data between the local and remote storage. It also sends events to the view model whenever data is changed. The view model is initialized with an instance of the *FirestoreManager* which can be used to subscribe to data events. The view model can either extend *ViewEventsViewModel* which extends *BaseViewModel*, or it can extend *BaseViewModel* directly.

Which class should be extended depends on whether the view model is using view events to communicate with the view or not. These view events could include things such as notifying state change or retrieving data from the view. The reason for using view events, and not accessing the view directly from the view model, lies within the core principles of MVVM. One of the fundamental ideas behind MVVM is that the view model should never hold a reference to the

view. This is done to make sure that the view model is properly decoupled.

Another way of communicating with the view from the view model is through data binding. Data binding is defined in the layout file of the view and allows the view model to bind observable fields to properties of the view. For more complex actions, such as changing toolbar icons or navigation between screens, view events are required.

View events are implemented by using observable functions. These functions can be called by the view model to make the view perform a specific action. The view is responsible for binding to all the functions that the view model provides. How they are implemented is determined by the view which binds to them. This makes it possible to have several views share the same view model, where each provides their own implementations of the bounded functions.

While view events can effectively communicate between the view model and the view, it does not provide a solution for sending events between view models. This is done by using the *EventHandler* class instead. The *EventHandler* is an implementation of the publish-subscribe pattern and functions as an event bus where all view models can send and listen for events. The events are identified by tags, which allows for the view model to determine which event is relevant.

Figure 8.6 shows how the different components are related to each other and how they realize the MVVM pattern. The *EventHandler* has been omitted from the diagram, as it is not directly involved in the MVVM pattern. The three highlighted components are generic representations which would be implemented when creating dialogs, fragments or activities. For example, the profile component is made up of three classes: the *ProfileViewModel*, *ProfileViewEvents* and *ProfileFragment*.

The *ProfileFragment* is what the user sees on his screen. It receives data from the view model such as the user's name and email. When the user clicks a button in the *ProfileFragment*, an event is sent to the *ProfileViewModel* through the usage of data binding. Since the *ProfileViewModel* does not contain a reference to the *ProfileFragment*, it uses view events from *ProfileViewEvents* to communicate with it. This pattern is used for all the different views in Pecora, such as dialogs, fragments, and activities.

For common actions such as showing a pop-up message or a confirmation dialog, helper view events are used instead. Helper view events function the same as normal view events, except that they are implemented through the base view classes such as *BaseFragment*. This enables view models to access these common actions without having to implement them on a per-use basis.

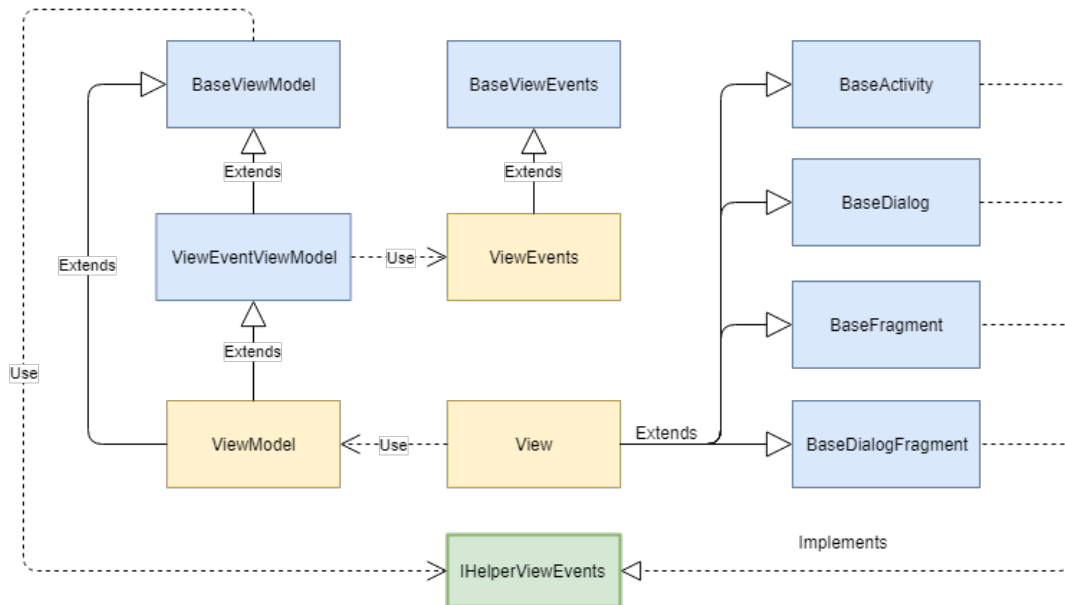


Figure 8.6: MVVM structure

8.2.2.2 EventHandler

The *EventHandler* is as previously mentioned used to send events between view models. All events contain a tag which is used by classes who listen for events to determine if the event was meant for them. There are three different kinds of events:

- **Event** - A normal event which only consists of an event tag.
- **DataEvent** - An event which sends data to the receiver.
- **RequestDataEvent** - An event which requests data to be sent back from the receiver.

Each of the events are used for different purposes. The normal *Event* is used for sending events that do not require additional data by the receiver. The *DataEvent* and *RequestDataEvent* is used for sending and receiving data respectively. Together, the events can be used to resolve all situations that require view models to communicate with each other. The Figure 8.7 shows how the *EventHandler* is structured.

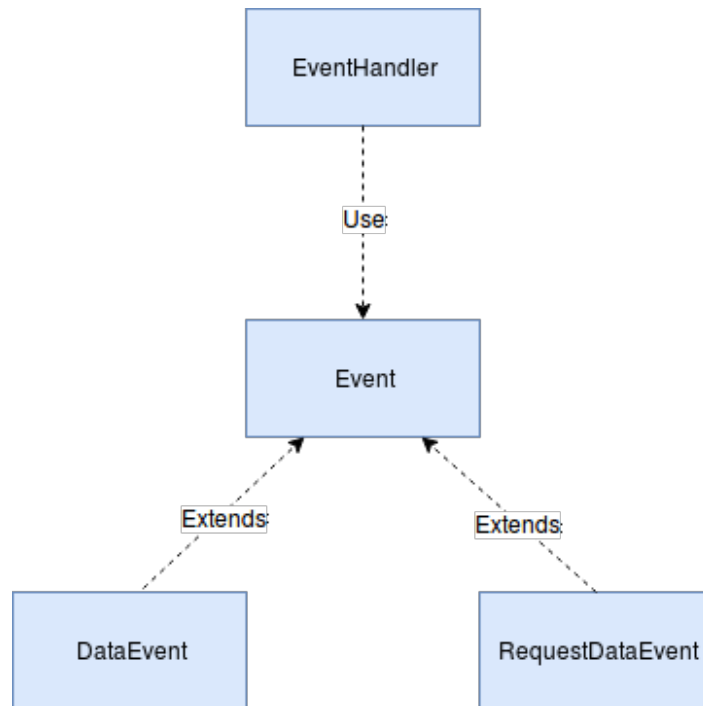


Figure 8.7: EventHandler structure

8.2.2.3 Observable Functions

Observable functions are used by Pecora to implement view events. View events are events that are sent from the view model to the view, to trigger an action or receive data.

All observable functions extends the *BaseObservableFunction* class, which abstracts the binding to the function. There are several observable functions, each one taking a different number of arguments. For example *ObservableFunction2* takes two arguments while *ObservableFunction3* takes three arguments. All observable functions have an optional return value which is sent through the *FunctionCallback*.

Every observable function uses a corresponding function class, which is passed along to the view that has bound to the function. For example, the *ObservableFunction2* uses the *Function2* class which takes two arguments. Currently, there are observable functions that support up to four arguments, but support for more can easily be added due to the architecture of the observable functions. Figure 8.8 shows an overview of how the observable functions are structured.

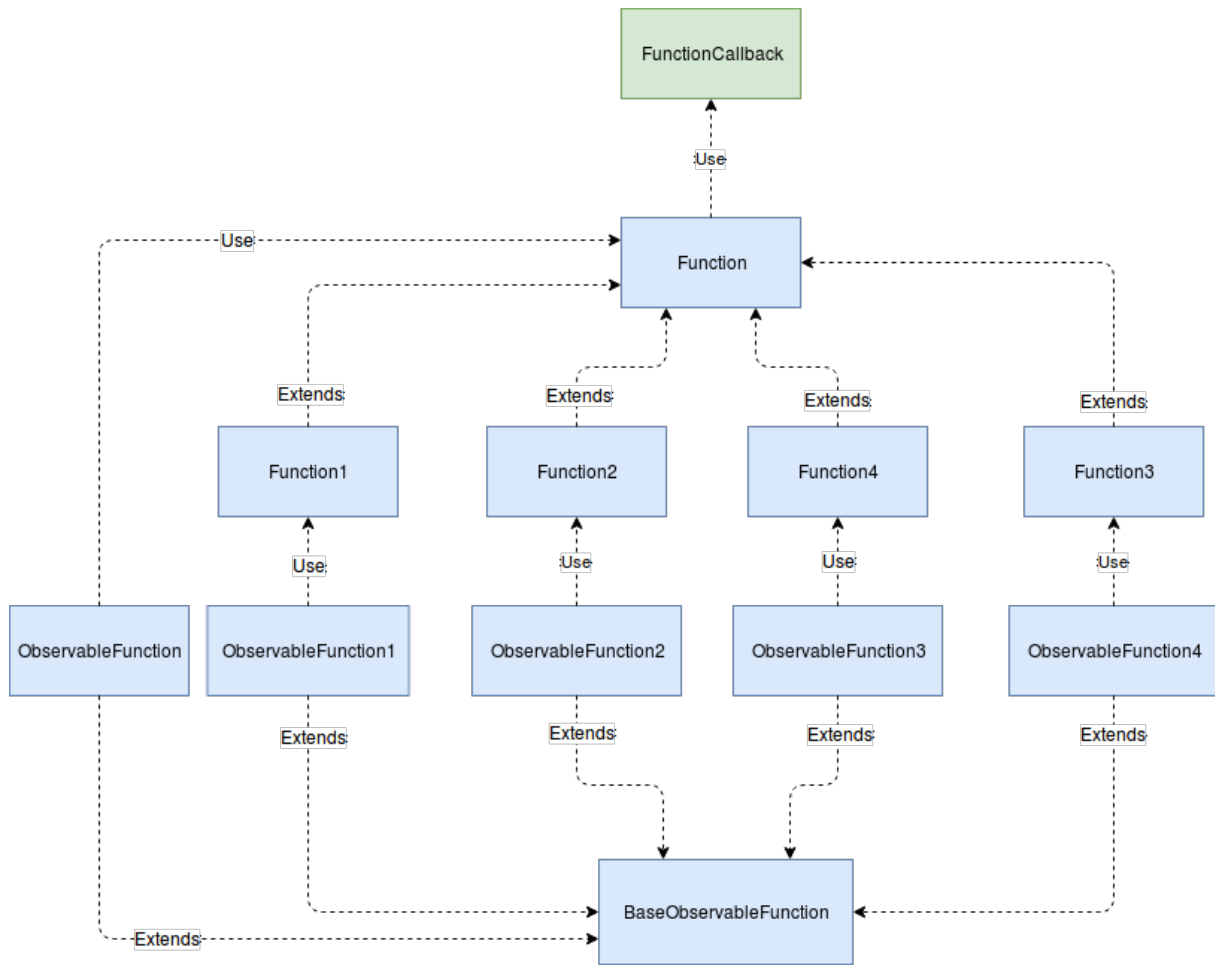


Figure 8.8: Observable function structure

8.2.2.4 Data Binding

Data binding is utilized to minimize the glue code between the application logic and the view. The views of the Pecora application, i.e., the activities, fragments, etc., can all instantiate their own binding class. The binding class is responsible for handling the data streams between the view and the view model and is generated automatically based on the associated view. The view models can define observable fields that the view can subscribe to. These subscriptions create the streams of data that the binding class is responsible for.

There are two possible streams of data exchange. One is from the view to the view model; the other is from the view model to the view. When the user interacts with the application and updates the view, e.g., clicking a button, the binding is notified and executes the necessary methods to update the view model. When the view model is updated, e.g., when the database model changes, the binding class is notified and executes the necessary methods to update the

view. It is possible to utilize both streams at the same time, which is called a two-way data binding. Figure 8.9 is a visualized representation of a two-way data binding.

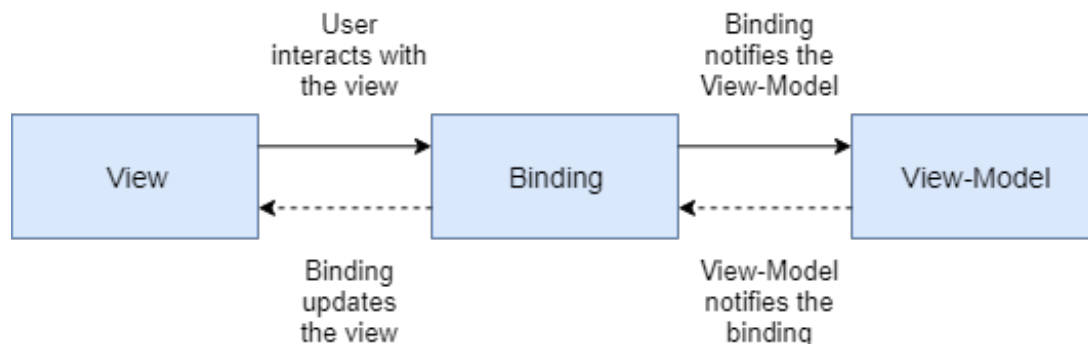


Figure 8.9: Data binding

8.2.2.5 Firebase

Pecora relies heavily on interacting with Firebase for storing and retrieving user data. There are a significant amount of data types to manage, such as map data, trips, observations, reports, account information, etc. Due to this, the code for managing Firebase has been separated into several manager classes, each responsible for their distinct type of data. For example, the *ReportManager* contains code for managing reports, while the *TripManager* is responsible for managing trips.

This modular approach allows for changes to be done in one manager, without the risk of side effects appearing in other managers. It also makes the code more manageable if the project were to grow in size, where more managers might be needed.

- **FirestoreManager** - Responsible for initializing the other managers, and providing them to the view models when needed.
- **AuthenticationManager** - Manages the authentication process of Pecora.
- **GroupInvitationManager** - Manages sending and receiving group invitations.
- **GroupManager** - Manages group related interactions.
- **ImagePathManager** - Manages the paths of images taken during trips.
- **MapDataManager** - Manages the areas of map data that the user has downloaded. This manager is not responsible for downloading the map, but storing the metadata for the area.
- **ObservationManager** - Manages observations by the user during trips.
- **ReportManager** - Manages generated reports. This manager does not generate the report,

but stores and retrieves metadata from previously generated reports.

- **TrackManager** - Manages user positions which are stored as tracks.
- **TripManager** - Manages user trips.
- **UserDataManager** - Manages account information such as email, username and avatar icon.
- **StorageManager** - Manages interactions with *Firebase Storage*. This includes operations such as uploading a new avatar icon, or downloading the user manual.

Figure 8.10 shows the different managers connected to the *FirebaseManager* class, which aggregates the managers into one place, to make them more accessible.

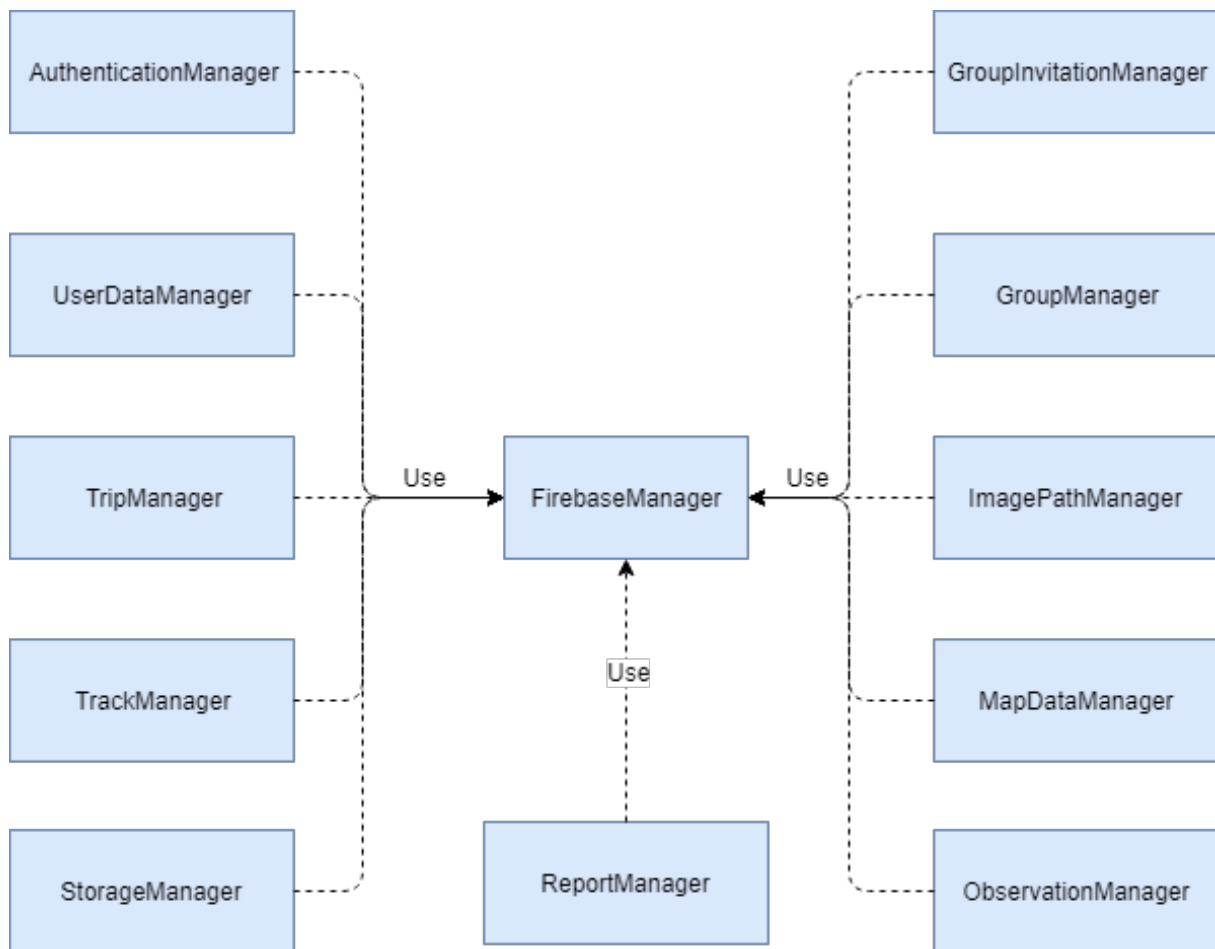


Figure 8.10: Overview of Firebase managers

8.2.2.6 Dagger 2

Dagger 2 is a dependency injection framework that makes it easier to manage dependencies between classes. Dagger 2 is used in Pecora to manage the dependencies that the view models have to other classes. Classes such as the *EventHandler* and *FirebaseManager* are initialized once when the app starts, and then injected into every view model when they are first used. Using Dagger to manage dependencies, makes it easier to introduce new dependencies without breaking existing code, as well as injecting mocks when testing.

The *ViewModelComponent* interface defines the template for which dependencies that can be injected. Dagger 2 will automatically generate the class *DaggerViewModelComponent*, which implements the previously mentioned interface and injects dependencies by using data provided through the *ViewModelModule* class.

When the application is first started, the *Injector* class will be initialized. The *Injector* is responsible for injecting the dependencies into the view models, by using *DaggerViewModelComponent* together with the *ViewModelModule*. This is illustrated in Figure 8.11.

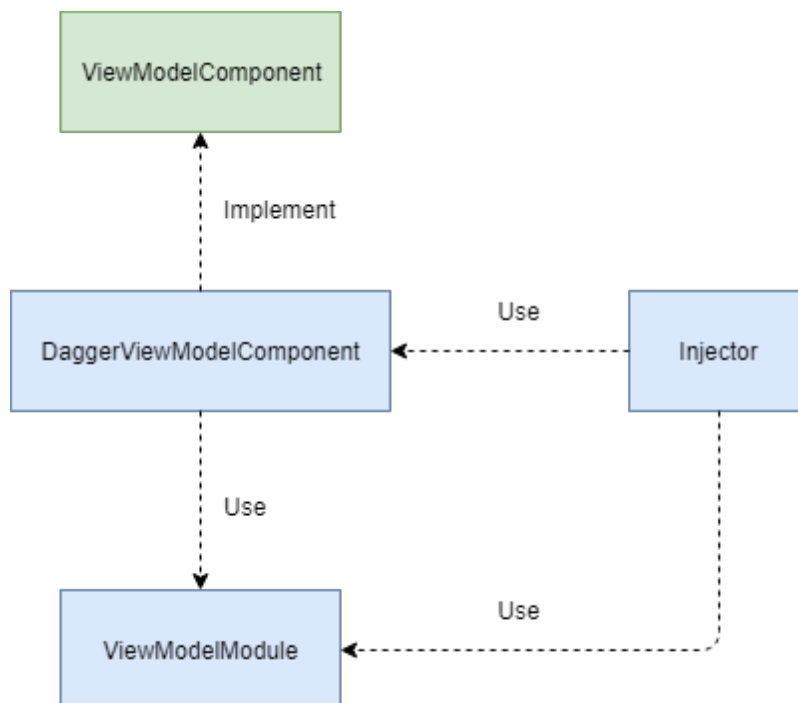


Figure 8.11: Dependency injection in Pecora

8.2.2.7 LocationHandler

There are several components in Pecora which need the user's location for actions such as reverse geocoding or recording the user's position during a trip. In Pecora, all location related operations are done through the *LocationHandler*. When a component is in need of the user's location, it can register a listener through the *LocationHandler* and will be notified whenever the location is updated. By using the publish-subscribe pattern, multiple components can listen for location updates at the same time. It also makes it easier to cache to user's last location, so that new components that are in need of a location can get one immediately, instead of waiting for the next location update. The concept behind the *LocationHandler* is illustrated in Figure 8.12.

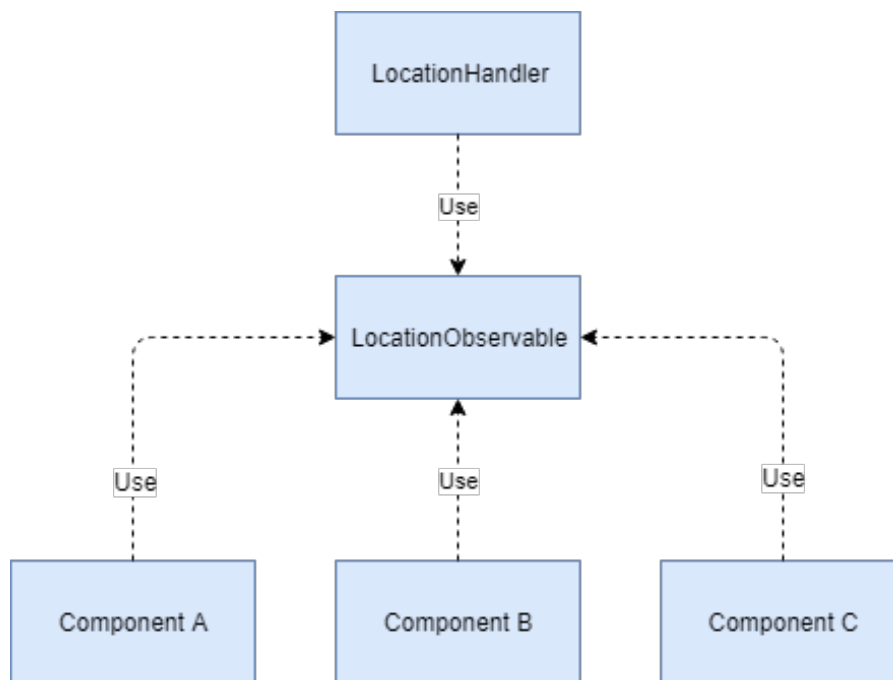


Figure 8.12: Multiple components listening for location requests

Chapter 9

Implementation

This chapter will go into details on how the different core software components of Pecora were implemented.

9.1 Rxjava 2

A considerable amount of the implementation chapter will revolve around the usage of RxJava 2. It is therefore important to have a fundamental understanding of what RxJava 2 is before continue reading this chapter.

RxJava is a Java VM implementation of Reactive Extensions: a library for composing asynchronous and event-based programs by using observable sequences. It extends the observer pattern to support sequences of data/events and adds operators that allow you to compose sequences together declaratively while abstracting away concerns about things like low-level threading, synchronization, thread-safety and concurrent data structures [37].

Listing 9.1 shows a simple example of how to use RxJava. In the example, we have an observable, *messageObservable*, which is initialized with two messages. When this observable is subscribed to, it will send the two messages to the subscriber. When subscribing, we can give the function a lambda expression which will be called for each message. In this example, the code would print the output "Message from observable: Message1" and "Message from observable: Message2" on two separate lines. RxJava supports a plethora of other operations, such as changing the

thread which the subscriber uses, or mapping the input of an observable to a different type. For more information, please take a look at the RxJava homepage [37].

```
Observable<String> messageObservable = Observable.just("Message1",
    "Message2");

messageObservable.subscribe(message -> {
    System.out.println("Message from observable: " + message);
});
```

Listing 9.1: Simple RxJava example

9.2 Model View View-Model

Pecora's implementation of MVVM is inspired by several other Android MVVM frameworks. We decided to implement our own version of MVVM, as the available frameworks enforced their own particular way of writing code, which felt limiting in some regard. This resulted in some trial and error during the implementation. However, it was an excellent opportunity to gain a better understanding of how MVVM works behind the scenes.

9.2.1 Model

The model acts as a Data Access Layer (DAL), which provides the rest of the application with a simplified way of accessing data. Almost all data operations in Pecora are done through Firebase, which provides listeners for monitoring data changes. The listeners are implemented by using RxJava. For example, the *UserDataManager* returns an RxJava *Observable*, which can be subscribed to for data changes. Listing 9.2 contains an example where the *UserDataManager* is used for listening to changes. When the user data is updated, for example, if the user changes his username, the function *onUserUpdated* will be called on the main thread, with the updated user data as an argument.

```
firebaseManager.userDataManager.get()
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(this::onUserUpdated);
```

Listing 9.2: Listening for changes to user data

In some cases, where the data is simple and unique for each device, *SharedPreferences* is used instead. *SharedPreferences* is provided by the Android platform and can store key-value pairs persistently on a device. This is useful in cases such as when storing user-defined settings, or other small pieces of data. While Firebase could be used for this purpose as well, it is better to use *SharedPreferences*, due to it not being necessary to store the data online.

9.2.2 View

The view is implemented by extending one of the several base classes such as *BaseActivity* or *BaseFragment*. In Listing 9.3, *ProfileFragment* is extending the generic class *BaseFragment*, which requires a view model as a parameter. This allows for the base class to take care of most of the boilerplate code that connects the view to the view model, and the cleanup after the view is destroyed.

```
class ProfileFragment extends BaseFragment<ProfileViewModel> {...}
```

Listing 9.3: ProfileFragment class definition

The view is required by the base class to provide some parts of the setup. This is done by overriding the three setup functions which are used by the base class during the initializing. The functions are *setupView*, *setupBindings* and *setupViewModel*. Listing 9.4, shows an example of how these functions are implemented in *ProfileFragment*.

```
@Override
protected View setupView(@NonNull LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {
```

```

binding = ProfileFragmentBinding.inflate(inflater, container, false);
binding.setViewModel(viewModel);

setHasOptionsMenu(true);

binding.toolbar.inflateMenu(R.menu.empty);

NavControllerHelper.setupNavController(getActivity(), binding.toolbar);

return binding.getRoot();
}

@Override
protected void setupBindings() {

    disposables.add(viewModel.viewEvents.updateUserAvatarImage.function()
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe((function) -> updateUserAvatarImage(function.arg1)));

    disposables.add(viewModel.viewEvents.showChangeUsernameDialog.function()
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe((function) -> showChangeUsernameDialog()));

    disposables.add(viewModel.viewEvents.sendPickImageIntent.function()
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe((function) -> sendPickImageIntent()));
}

```

```
@Override
protected void setupViewModel() {
    viewModel = ActivityUtils.findOrCreateViewModel(getActivity(),
        ProfileViewModel.class);
    Injector.inject(viewModel);
}
```

Listing 9.4: ProfileFragment setup functions

The *setupView* function is responsible for setting up and returning the inflated view, which contains the buttons, lists, text fields, etc. The view should be ready for user interaction after this function has returned. In the case of *ProfileFragment* this means setting up the navigation drawer.

The *setupBindings* function is responsible for subscribing to all view events that are provided by the view model. When the view is initialized, this function will be called before the view model is initialized. If a view event is not bound after the *setupBindings* function has been called the base view class will throw an exception. This makes it easy to debug missing bindings.

Lastly, the *setupViewModel* function initializes the view model that the view will use. In most cases, this will create a view model through the *ActivityUtils* class, which stores the view model inside a fragment that is retained throughout the lifecycle of the activity. This is important, as the view model will lose its state whenever it is recreated. After the view model has been created, it has its dependencies such as the *FirebaseManager* class injected, through the *Injector* class.

9.2.3 View Model

Similar to the base view classes, the view model also has a base class. The *BaseViewModel* contains common view model functionality. However, it does not contain view events to communicate with the view. In cases where this is needed, the generic class *ViewEventViewModel* is used instead. *ViewEventViewModel* extends *BaseViewModel*, and requires the view events of the view model to be specified.

```
class ProfileViewModel extends ViewEventViewModel<ProfileViewEvents>{...}
```

Listing 9.5: ProfileViewModel class definition

The view events used by *ProfileViewModel* is contained inside *ProfileViewEvents*, which is shown in Listing 9.6. When the view model needs to show the *ChangeUsernameDialog*, it will use the *showChangeUsernameDialog* function to send an asynchronous request. The request will then be received by *ProfileFragment*, which bounded to the view event inside its *setupBindings* function. When the fragment receives the notification through the view event, it will show the dialog.

```
public class ProfileViewEvents extends BaseViewEvents {  
  
    public final ObservableFunction<Void> showChangeUsernameDialog = new  
        ObservableFunction<>();  
  
    public final ObservableFunction<Void> sendPickImageIntent = new  
        ObservableFunction<>();  
  
    //Takes 1 argument: avatar image url  
    public final ObservableFunction1<String, Void> updateUserAvatarImage =  
        new ObservableFunction1<>();  
}
```

Listing 9.6: ProfileViewEvents

For the view model to initialize and clean up after the view is destroyed, it needs to be notified what happens with the view. This is done through four different lifecycle functions, which covers the lifespan of the view. The functions are as follows:

- *onViewCreated* - Called when the view is first created.
- *onViewVisible* - Called when the view becomes visible to the user.

- *onViewHidden* - Called when the view becomes hidden from the user.
- *onViewDestroyed* - Called when the view is destroyed.

The lifecycle functions allow the view model to know when to request data from the model and when it can release the data. Depending on a view model's needs, all or none of them might be used.

9.2.4 Data Binding

Data binding requires the binding to attach to a view, such as a *TextView* or an *EditText*. An example of this can be seen in Listing 9.7. In this example, the text that the *TextView* displays, is bound to the observable field *username*, which is defined inside the view model. Other properties can also be bound to, such as the visibility or color of the *TextView*.

```
<TextView
  android:id="@+id/usernameTv"
  <!-- ... -->
  android:text="@{viewModel.username}"
/>
```

Listing 9.7: *TextView* being bound to the username

Since the *TextView* is being bound to the username field, it must be defined in the view model. Listing 9.8 shows how the username field is defined. In this case, the field contains a string value, but it can hold other values such as integers or booleans instead. It is important to note that the field is made "public", or else it would not be accessible to the binding.

```
public final ObservableField<String> username = new ObservableField<>();
```

Listing 9.8: Username field

After the field has been defined in the view model, it is then necessary to initialize the binding. This is done in the *setupView* function seen in Listing 9.9, which is called before the view is

being displayed to the user.

```
@Override
protected View setupView(@NonNull LayoutInflater inflater, ViewGroup
    container, Bundle savedInstanceState) {

    binding = ProfileFragmentBinding.inflate(inflater, container, false);
    binding.setViewModel(viewModel);

    //...

    return binding.getRoot();
}
```

Listing 9.9: Initialization of data binding

After the initialization is done, whenever the value of username in the view model changes, the value being displayed by the *TextView* will automatically change as well. This significantly reduces the amount of boilerplate code needed as well as reducing the chance of bugs being introduced.

9.3 Libraries

There are several libraries that are currently being used in Pecora. This section will briefly describe each of them, and how they were implemented in the application.

9.3.1 Facebook Login

One of the options the user has when logging in is using his existing Facebook account. To accomplish this, the application needs to be registered with Facebook, and use Facebook's login library.

9.3.2 Google Login

As most Android users have a Google account to log in with, having an option to log in with Google is important. The user can either log in to his existing account or create a new one if they do not already have one. To log in with Google, the Google authentication library is needed.

9.3.3 Picasso

Picasso is an image loader library that makes it easy to display an image from an URL, a local file, or through a caching mechanism. Listing 9.10 shows an example using Picasso to load and display an image. All images in Pecora are displayed through Picasso.

```
Picasso.get().load("http://i.imgur.com/DvpvklR.png").into(imageView);
```

Listing 9.10: Picasso example

9.3.4 CircleImageView

CircleImageView extends the default Android ImageView class and makes it possible to display images as a circle. It also comes with other features that make it especially suited for profile pictures. CircleImageView is used in Pecora to display circular avatar images, such as Figure 9.1.



Figure 9.1: CircularImageView example

9.3.5 Logger

While the default logging in Android is sufficient for most uses, it still lacks some features. Logger works similar to the default logging that is used. However, the output from Logger is easier to read and provides a brief stack trace and code location from where the logging was done. This makes it much more efficient to read log messages during debugging. An example of the output from Logger is provided in Figure 9.2.

```

D/PRETTY_LOGGER:
D/PRETTY_LOGGER: Thread: main
D/PRETTY_LOGGER:
D/PRETTY_LOGGER: Activity.performCreate (Activity.java:6679)
D/PRETTY_LOGGER: MainActivity.onCreate (MainActivity.java:29)
D/PRETTY_LOGGER:
D/PRETTY_LOGGER: Thread info, method info and message
D/PRETTY_LOGGER:

```

Thread info
Method info
Message

Figure 9.2: Logger example

9.3.6 Gson

Gson is a Java serialization/deserialization library, which can convert Java objects to JSON and back. Using Gson allows Pecora to store arbitrary data structures into a manageable string format, which can easily be persisted to disk through the usage of *SharedPreferences*.

9.3.7 Swipe Button

The swipe button is frequently used in mobile applications for user confirmation. The swipe button is currently only used for confirmation when deleting groups, but it might be used more frequently in newer versions of Pecora. An example of the swipe button is shown in Figure 9.3.

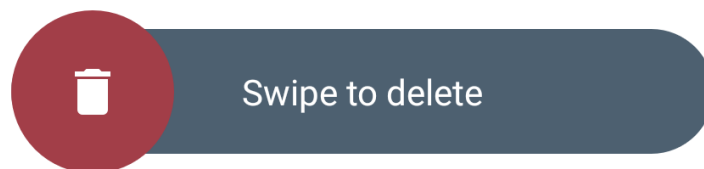


Figure 9.3: Swipe to delete button

9.3.8 Number Picker

The default Android NumberPicker takes up a lot of space, lacks flexibility, and customization options. The utilization of space is important in the observation dialog, where there are a lot of input fields, and space is limited. For this reason, the library Number Picker was used, to replace the default NumberPicker. The number picker from the library can be found in Figure 9.4.

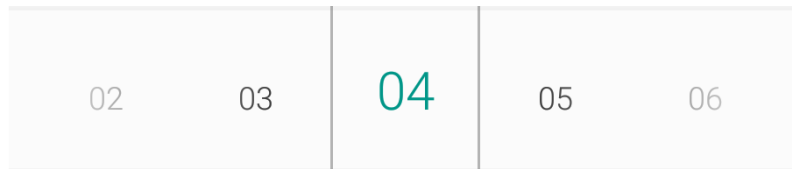


Figure 9.4: Number picker

9.3.9 Color Picker

There are several fields in the observation dialog where there is a need for picking a color. While we could implement this functionality ourselves, using a library saved time and prevented potential bugs. The color picker library is very lightweight, customizable, and supports different styling as well. An example of the color picker can be found in Figure 9.5.

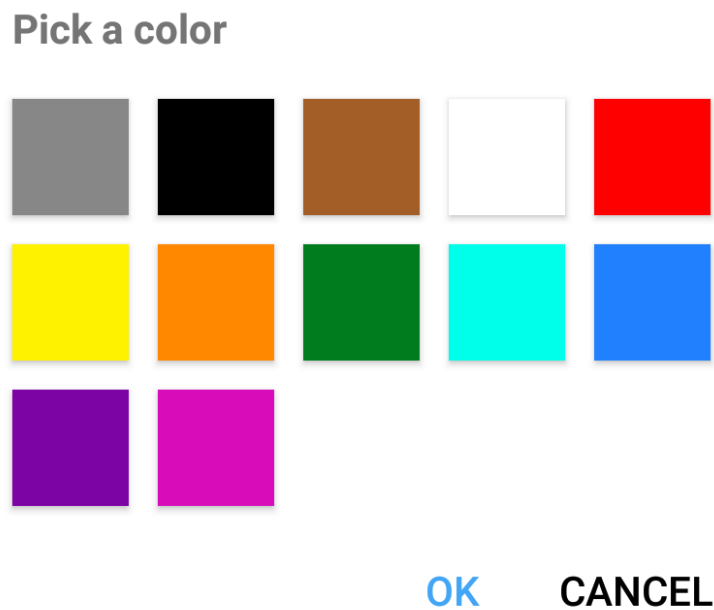


Figure 9.5: Color picker

9.3.10 PDF Viewer

The help-component in Pecora contains a user manual that provides detailed information on how to use the application. The user manual is in a PDF format, which requires a special viewer to present it to the user. For this purpose, the PDF viewer library was used. An example of the PDF viewer being used, can be found in Chapter 6 - Walkthrough of Pecora, in Figure 6.44.

9.3.11 Osmdroid

Osmdroid is an open-source Android library which aims to be a full/free replacement for Android's MapView. Osmdroid is used both for displaying and downloading maps. It is quite flexible and allows for customizing most of its components. It also comes with a modular tile provider system, which supports using offline tiles.

Pecora uses a custom tile provider which supports both online and offline tile sources. When a tile is requested from the *PecoraTileProvider* it will first look through all downloaded areas after the tile. If the tile has not been downloaded, then it will look through previously cached tiles. If the cache does not hold the tile either, then it will try to download it from Kartverket's servers, as long as the user is connected to the internet. Using this approach allows Pecora to display a map correctly regardless of internet connection. Also, the usage of offline tiles first, ensures that the tiles are loaded quickly.

9.4 Quality Attribute Tactics

This section will discuss how the quality attribute tactics defined in Chapter 5 have been implemented in Pecora.

9.4.1 Usability Tactics

This section will present the implementations of the usability tactics.

Cancel

All operations and confirmations in Pecora have a cancel functionality. For example, when creating a new group, the user can cancel the operation if he has reconsidered. This is shown in Figure 9.6.

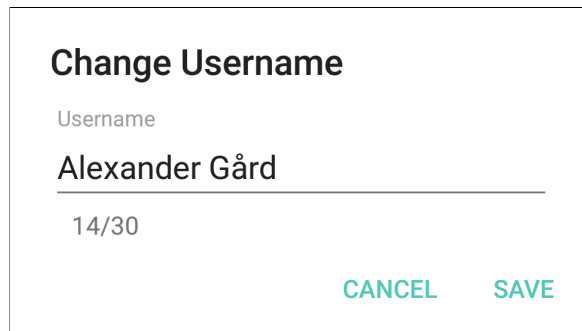


Figure 9.6: Cancel tactic

Undo

Most reversible operations in Pecora have an undo functionality. For example, when deleting a downloaded area, the user is presented with an undo option for a short duration. This is shown in Figure 9.7.

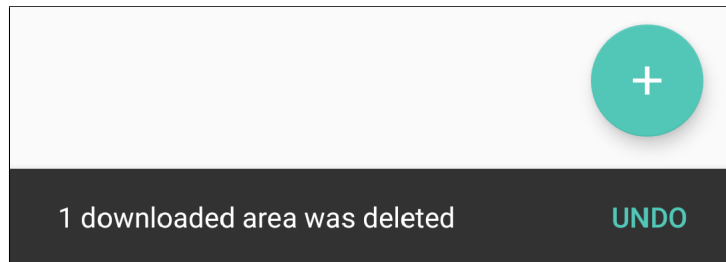


Figure 9.7: Undo tactic

Pause/Resume

All long running operations should have the functionality for pausing/resuming. There is currently only one component that needs this feature, which is the "Download Tile" component. This can be seen in Figure 9.8.

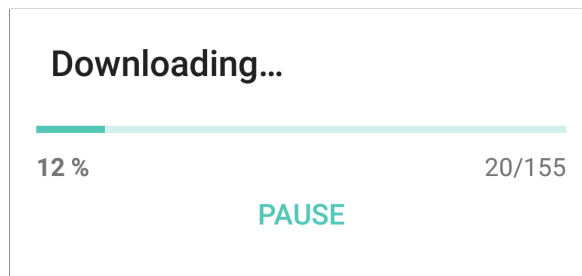


Figure 9.8: Pause tactic

Aggregate

Most repetitive actions that can affect a large number of entities in the application can be aggregated into one action. For example, if a set of trips is to be deleted, the user can select

multiple trips before confirming the deletion, as shown in Figure 9.9. If the actions were not aggregated, the user would have to delete the trips one by one.

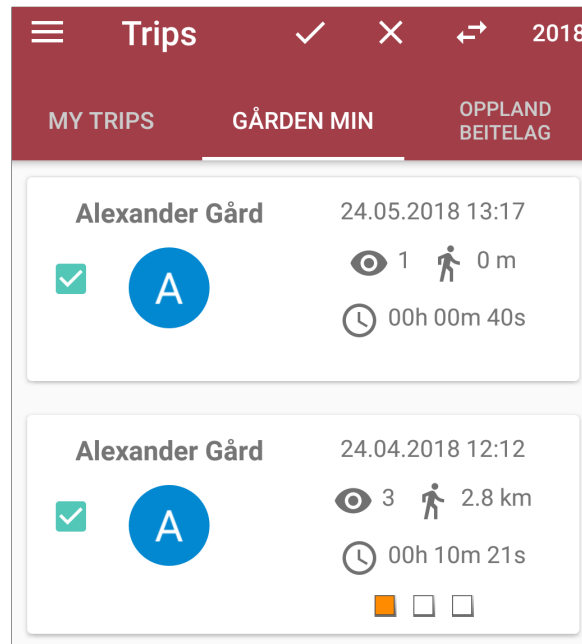


Figure 9.9: Aggregate tactic

Maintain Task Model

The task model is responsible for containing information about the given task being performed. The task model has been implemented in Pecora through the usage of view models. The view model contains the state, information, and logic that the view can use to help the user accomplish his tasks.

Maintain User Model

The user model is responsible for containing information about specific preferences that the user might have while using the application. This is implemented through the settings and profile components, where the user can customize the application.

9.4.2 Availability Tactics

This section will present the implementations of the availability tactics.

Exception Handling

Exception handling in Pecora is implemented in separate ways depending on what part of the code was responsible for the exception. All data operations and view events use RxJava for

running the code on different threads. If an exception occurs, then an "error consumer" will be used to catch the exception and handle it appropriately. For code that does not use RxJava, normal try-catches are used instead.

Retry

Since Pecora runs in a demanding environment, where the internet connection might fail at any moment, operations that require an internet connection must be retried until they succeed. For Firebase Realtime Database operations, this feature is implemented by default. For operations such as uploading images from observations, this must be done manually. This is completed through the usage of the *StorageUploadHandler* class. If the upload handler fails to upload to the storage, then it will wait until the application is connected to the internet before retrying. If the application is closed before the upload handler has finished uploading, then it will persist the pending operations to disk. When the application is started at a later time, the handler will retrieve the stored operations and try to upload them as normal.

Degradation

As previously mentioned, Pecora is often used in an environment with poor or non-existing network connectivity. Therefore, it is important that the system will function properly in a degraded state. This is implemented through configuring the Firebase Realtime Database for offline operations. This will ensure that the application can perform data operations as normal. For operations that require an internet connection such as login or downloading map data, the user will be displayed with a message if the operation was not successful.

9.4.3 Modifiability Tactics

This section will present the implementations of the modifiability tactics.

Split Module

The source code of the Pecora components has been structured so that they only contain the classes that are required. For example, the "change_username" component only consists of the three classes *ChangeUsernameDialog*, *ChangeUsernameViewModel*, and *ChangeUsernameViewEvents*. Code that is used in several places is extracted into static helper classes or handlers, depending on what sort of tasks it performs.

Increase Semantic Coherence

All helper classes and handlers in Pecora are split into classes depending on their responsibilities. For example, the *AnimationHandler* only handles animations, while the *FileHelper* is responsible for file related operations. The model, view, view model and view events are also divided depending on their responsibilities.

Encapsulate

All components are encapsulated from each other, and only exposes a minimal set of functions that can be used by other classes. The components are also encapsulated internally as well. For example, the view can only access the view events that the view model has exposed to it.

Intermediary

Many of the dependencies that exist between components are resolved through an intermediary observable, which uses RxJava. This allows the components to interact without tight coupling. An example of this is the *EventHandler*, which is used as an intermediary between view models, to allow for asynchronous communication. The *LocationHandler* is also implemented similarly.

Hide Information

The components in Pecora only exposes the minimal amount of information that is required. For example, the view model only exposes view events and observable fields to the view. Fields that are not required are made private and are exposed through getter/setter functions if necessary.

Restrict Communication Paths

Communication between view models is restricted to using the *EventHandler*. For communication between the view and the view model, data binding and view events are used instead. In addition to restricting the communication paths, the way of communicating through these paths are also restricted. For example, the data binding can only get/set simple values.

Chapter 10

Testing

This chapter explores the testing aspect of Pecora. It contains the overarching test strategy and the individual testing levels used for testing. The testing strategy and the different levels are presented in their own sections.

10.1 Test Strategy

This section details the overall strategy used for testing Pecora. It is approached in a general sense, i.e., it does not go into detail when it comes to the different levels of testing, as this is mentioned in the individual test level sections. The strategy is approached in a static manner, meaning that changes should not occur. The purpose of the test strategy is to set a frame of reference for other test activities to draw their content from.

10.1.1 Scope

The scope of the test strategy is limited by the nature of the project, i.e., its size and the number of testers involved. In this case, two students are responsible for testing the application. It is therefore not necessary to construct a test strategy with the level of detail one would expect to find in a more professional environment, where it is possible to have dedicated teams strictly focused on testing. The test strategy concerns itself with the testing approach, the different roles and responsibilities of the participants, and how defects uncovered from testing were handled.

10.1.2 Testing Approach

Pecora underwent three consecutive testing levels: unit testing, GUI testing, and system testing. Each test level was completed in the order they were mentioned, as visualized in Figure 10.1. However, whenever alterations, additions, or deductions were made to the codebase, the test levels had to be retested, starting from the unit test level.

The majority of the unit tests were written as the components of Pecora were developed. However, the initial unit tests were delayed due to inexperience with the MVVM architecture. At the beginning of the development, we were uncertain as to how the architecture would function in its entirety. We believed that a fundamental understanding of the inner workings of MVVM was necessary, to prevent incorrect unit test code and thus unnecessary work.

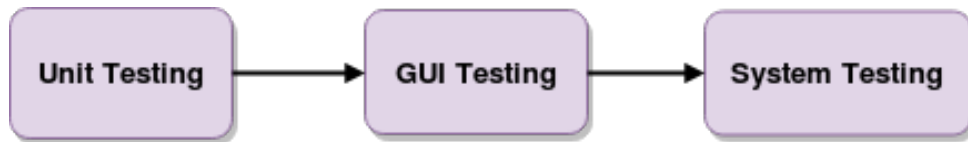


Figure 10.1: Testing Levels

10.1.3 Roles and Responsibilities

It is not uncommon to have well-defined roles and responsibilities among the projects members. However, due to the size of this project, we shared the responsibility for writing, executing, and documenting the tests. This meant that both of us were involved in all testing levels.

10.1.4 Defects Tactic

Whenever defects were uncovered during one of the test levels, the defects were documented using the Trello work board. The defects were prioritized, assigned to one person, and labeled for bug fixing, in compliance with the explanation of how we used Trello in Section 4.1. Before the defect could be marked as resolved, the affected unit, GUI, and system tests had to be passed.

10.1.5 Test Naming Convention

All code tests follow a predefined naming convention. This makes the tests more structured and easier to read and write. The convention follows the form of "When_StateUnderTest

`_Expect_ExpectedBehavior"`. The state under test is defined as the current state that is being tested, and the expected behavior is the behavior expected after the test has been executed. An example is shown in Listing 10.1.

```
public void When_CancelBtnClicked_Expect_DismissDialog(){...}
```

Listing 10.1: Unit test naming convention

In cases where there is a precondition, the precondition will be appended to the state under test. If the test from Listing 10.1 had a precondition that it would only dismiss the dialog if the user is authenticated, then the test would be written as in Listing 10.2.

```
public void  
    When_CancelBtnClicked_And_UserIsAuthenticated_Expect_DismissDialog(){...}
```

Listing 10.2: Unit test with precondition

10.2 Unit Testing

Unit tests are responsible for testing individual units of source code. The unit tests are performed to decide whether a unit is behaving according to the expected behavior by identifying correct and erroneous logic. Unit testing will help prevent further additions to the source code from breaking the system.

10.2.1 Test Plan

The test plan describes the environment, the entry and exit criteria, and the tools used for the unit tests.

Environment

Unit testing requires a device that can compile and run the unit tests.

Entry Criteria

Unit tests should be performed as soon as individual units of source code provide functionality to the application.

Exit Criteria

The unit test level is completed when all unit tests assert that the expected and given outputs are equal.

Tools

The following tools were used for the unit testing level.

- **JUnit** - JUnit is a Java unit testing framework, which allows for writing clean and concise tests.
- **Roboelectric** - Roboelectric is an Android unit testing framework that provides a mock of the Android SDK. This allows for testing parts of the code that relies on features that are native to Android.
- **Mockito** - Mockito is a Java mocking framework which is frequently used during unit tests.
- **Powermock** - Powermock is a Java mocking framework. Unlike Mockito, Powermock focuses on mocking code that would otherwise be difficult to mock, such as final fields and static functions.

10.2.2 Testing Execution

The unit tests were written to cover all view models and helper classes where the main logic of Pecora is found. Each class that is tested is separated into its own test class. Some of the code that relies on Android is mocked, as it cannot be tested through unit testing, which runs on the JVM. The tests that use code that cannot be mocked will instead be tested through the GUI tests. An example of a unit test is shown in Listing 10.3.

```
@Test
public void When_CreateCopy_And_ListIsNull_Expect_ReturnEmpty(){
    List<String> test = null;
```

```
List<String> copy = CollectionHelper.createCopy(test);
assertNotNull(copy);
assertEquals(copy.size(), 0);
}
```

Listing 10.3: Unit test example

10.3 GUI Testing

GUI tests are responsible for testing interactions with the user interface. This includes actions such as clicking buttons, navigation and asserting that views are correctly displayed. These tests are essential for ascertaining the functional state of the user interface.

10.3.1 Test Plan

The test plan describes the environment, the entry and exit criteria, and the tools used for the GUI tests.

Environment

GUI testing requires a device that can compile, run and validate the GUI tests.

Entry Criteria

The GUI test is carried out after all the unit tests are completed.

Exit Criteria

The GUI tests are completed when the user interface of the application has been covered thoroughly, and all possible interactions have been tested.

Tools

The following tools were used for the system testing level.

- **JUnit** - JUnit is a Java unit testing framework, which allows for writing clean and concise tests.
- **Hamcrest** - Matchers that can be used to express intent.
- **Espresso** - Framework for writing Android GUI tests.

10.3.2 Testing Execution

The GUI tests were written to cover all possible GUI interactions. Each component is tested separately in its own test class. For each test that is run, the test runner will start up a new instance of Pecora, to prevent side effects from earlier tests. When testing, Espresso will emulate user actions and assert that the appropriate response is executed. The GUI tests use a special database instance for testing, to prevent interference with the production environment. An example of a GUI test is shown in Listing 10.4.

```
@Test
public void When_LogInBtnClicked_And_EmailIsEmpty_Expect_ShowToast() {

    onView(withId(R.id.logInBtn)).perform(click());

    onView(withText(R.string.sign_in_invalid_email))
        .inRoot(not(isFocusable()))
        .check(matches(isDisplayed()));
}
```

Listing 10.4: GUI test example

10.4 System Testing

System testing is a method to check the behavior of a unit and GUI tested application, and does not require any knowledge of the inner workings of the source code. This is the first time the application is tested in its entirety. The purpose is to find application defects, and verify whether the functional requirements have been met.

10.4.1 Test Plan

The test plan describes the test cases, test types, environment, the entry and exit criteria, and the tools used for the system tests.

Test Cases

Test cases were used to track the status of the functional requirements for Pecora. Table 10.1 shows an example test case. Initially, the fields of the test cases were empty but were filled out with relevant data as the tests progressed. Each field is briefly explained in the following list:

- **Test ID** - A unique identifier for the test case.
- **Requirements** - The associated functional requirements for the test case.
- **Preconditions** - Lists the preconditions of the test case.
- **Input** - The input for the test case. Button clicks, textual data, etc.
- **Expected Result** - The expected result for executing the test case.
- **Result** - The actual result from executing the test case.
- **Comment** - (*Optional*) A comment if additional information is needed based on the previous fields.
- **Passed** - Whether the test passed or failed. Options are *Yes* or *No*.

Test ID	STC 1.03
Requirements	1.03 - Pecora shall be able to log in a user with his Facebook account
Preconditions	The user is not logged in to the application with an existing user
Input	Click the 'Sign in with Facebook' button
Expected result	The user is logged in to the application with the given Facebook credentials
Result	The user was logged in to the application with the given Facebook credentials
Comment	-
Passed	Yes

Table 10.1: Example Test Case

Test Types

In addition to doing all the test cases, the system test utilized the following test types:

- **Regression Testing** - A technique that ensures that the previously developed version of the software still performs as intended and has not introduced new faults after it has changed.
- **Usability Testing** - A technique used to evaluate software by testing it on end-users. Normally, the users are asked to complete a predefined set of tasks while observers watch, listen, and take notes. The purpose is to identify usability problems with the software and determine the user's satisfaction with the software.

Environment

System testing requires a device that can run the unit and GUI tested application.

Entry Criteria

The system test is carried out after all the unit and GUI tests are completed.

Exit Criteria

The system test is complete when all the functional requirements have been met, the application is behaving as expected, and no defects have been found. In this case, all the test cases have passed.

Tools

No tools were used in the system test.

10.4.2 Testing Execution

The system test began with us constructing a set of test cases. All the fields of the test cases were filled out as they were being constructed, except the *Result* and *Comment* fields. They were initially left empty, as they required the application to be tested. When all the test cases had been constructed, we could begin the actual system testing. We went through all the test cases, and each test case was tested with the Pecora application. The remaining fields of each test case, which were *Result* and *Comment*, were filled out as the test cases were completed. The completed test cases can be found in Appendix F. By verifying all the functional requirements through the test cases, we ensured that all our functional requirements were satisfied. We also performed regression testing and usability testing.

10.4.2.1 Regression Testing

Whenever major changes were made to the application code base, the application went through a regression test. In the regression test, the test cases were evaluated in terms of their connection with the specific change at hand. The test cases with the strongest connection were retested. If defects were detected, it was handled as mentioned in Section 10.1.4.

10.4.2.2 Usability Testing

The first part of the usability test consisted of creating a document that detailed how the test was to be conducted. The document can be found in Appendix D, and contained the questions, information, and tasks for the test subject. The usability test first started with a brief introduction, where we made sure that the participating parties understood that we were testing the application, and not their ability to solve the tasks at hand. Then the app, its concept, and purpose were explained in more detail.

The test subjects were then asked to complete a set of predefined tasks, and explain the thought process behind each step. We informed the test subjects that we would not interrupt them, and that they had to complete the tasks to their best ability on their own. However, if the subjects came to a complete halt, we would point them in the right direction, but not directly give them the solution. While the subjects completed the individual tasks, we observed their gestures, listened to their thought process, and wrote down all things of significance.

Lastly, the test subjects were asked to fill out a SUS-form that contained ten statements. The subjects had to rate to which degree they agreed with each statement, where one was the lowest rating and five the highest. In total five people were tested. The average usability score and its related statement can be found in Table 10.2.

Nr	Statement	1	2	3	4	5
1	I think that I would like to use this system					X
2	I found the system unnecessarily complex	X				
3	I thought the system was easy to use				X	
4	I think that I would need the support of a technical person to be able to use this system		X			
5	I found the various functions in the system were well integrated					X
6	I thought there was too much inconsistency in this system	X				
7	I would imagine that most people would learn to use this system very quickly					X
8	I found the system very cumbersome to use	X				
9	I felt very confident using this system					X
10	I needed to learn a lot of things before I could get going with this system		X			

Table 10.2: SUS-form

The observations, notes, and results from the SUS-form gave us a clear indication that the application was on the right track. However, some issues made it clear that the user interface

could still use some fine-tuning. Almost all the test subjects tried to perform a long click on items that were to be deleted, instead of using the "delete" button. However, once nothing happened with a long click, the subjects usually tried the "delete" button as a second option and performed the deletion as expected. Another issue was that some dialogs did not have an OK or cancel button, which confused some of the test subjects that were not familiar with how the dialogs of Pecora saved its state when being closed. In addition, some textual descriptions were not descriptive enough and were difficult to understand. Overall the application's usability was considered to be good. In the end, we reviewed the results and made some improvements based on the feedback, such as writing more detailed descriptions and adding OK/Cancel buttons to most dialogs.

Chapter 11

Code Documentation

To ensure that the code base of Pecora is readable and maintainable, documentation is of utmost importance. Documenting the code allows developers to gain a better insight into how the code is structured and how it operates.

The documentation is written according to the Javadoc [38] standard. Javadoc is a tool which converts code documentation into HTML-pages, which gives a clear and concise overview of the source code. To create these pages, Javadoc requires the documentation to be written in a particular way, which is the Javadoc standard. An example of a documented method using the Javadoc standard can be found in Listing 11.1. The Javadoc for Pecora is available here: <https://it3901-55185.firebaseio.com/>.

```
/**
 * Create or update map data
 * @param mapData the map data being created or updated
 * @return empty task that represents the operation being completed
 */
public Task<Void> createOrUpdate(MapData mapData){...}
```

Listing 11.1: Javadoc example

Part IV

Interviews, Future Work & Conclusion

Chapter 12

Interviews

This chapter presents the two interviews that were conducted during the development of Pecora.

12.1 Farmer Interview

We aimed at having the application in a functional state where it could be presented to a farmer during an interview. The purpose behind the interview was to obtain feedback to further improve the application, thus it had to be in a presentable state. The app reached this state on the 15th of January, and the supervisor was notified. It was then agreed that the supervisor would schedule an interview with Steingrim, and let us know when it would take place.

The interview was scheduled on the 15th of February. However, due to extreme weather conditions, the interview was canceled. The supervisor agreed to reschedule the interview as soon as possible, but due to the supervisor's busy schedule, combined with the time constraints imposed by Steingrim, it took a long time before the next interview date was set.

We were informed that a meeting with Steingrim would take place on the 9th of April, but since several groups wanted to conduct their interviews, there was not enough time for every group to participate at this time. The supervisor informed us that another meeting would take place where we were to be included. On the 16th of April, we were informed that Steingrim would be busy with lambing the upcoming months, and would not have time to be interviewed. It was therefore decided that there would be no further interviews. The supervisor asked us to

contact the other groups that had already conducted their interview, to receive a transcript. The transcript has been included in Appendix B.1.

In hindsight, we believe that it would have been better for us to agree to a meeting with Steingrim without the supervisor, as their conflicting schedules made it difficult for us to find a suitable date. The development phase could also have benefited from a closer dialog with Steingrim, with continuous feedback from a potential user of the application.

12.1.1 Transcript Review

In this section, information from the transcript that was not already previously known is presented. From reviewing the transcript, several interesting pieces of information were uncovered. The most important being the existence of an app named BeiteSnap. According to Steingrim, this app was published recently, and he had used it last year to record his observations instead of relying on pen and paper. From our preliminary study, we were not able to find any mention of this app, so we were surprised to find a solution so similar to Pecora. We assumed that we did not find any information on BeiteSnap due to its small user base and recent publishing.

After examining BeiteSnap, it seems that it shares some of the same core features as Pecora. Both apps are used for registering observations, which can be used to generate a report at the end of the semester. However, BeiteSnap allows hikers to send their findings to farmers. This could help in situations where a sheep is either wounded or dead, and an immediate response is needed. An extensive userbase of hikers is required for this to be considered effective, which BeiteSnap currently does not have. Registering observations in BeiteSnap is quite simple and is lacking much of the details that can be registered through Pecora. Figure 12.1 shows a screenshot of the observation registration screen in BeiteSnap.

13:05

← Observasjon SEND

Velg dyreslag:

SAU GEIT STORFE

Velg type observasjon:

SETT SKADE DØD

Evt. individnr:

123456

Evt. melding (f.eks farge på øremerke, klave eller bjelle og annen relevant informasjon):

Figure 12.1: Observation registration - BeiteSnap

From the transcript, we also learned that each ewe that has birthed one or more lambs carry a tie with a specific color. The color denotes how many lambs were birthed by that particular ewe, and indicates how many lambs should be in the ewe's proximity. If there are fewer lambs than the color indicates, then it might be a sign that there are predators in the vicinity.

We also learned more about the process of creating and sending reports to the government. The farmer sends an electronic document on behalf of all the farmers in the area (if they are working together), to the county governor. However, the format of the document was still unknown at this point.

12.1.2 Interview Plan

Prior to the interview scheduled 15th of February, we created a detailed plan for the interview, which can be found in Appendix B.2. This plan would act as a guideline for the interviewers, providing them with a strict order of steps to be followed.

Firstly, the interview would start with us presenting ourselves, and the motivation behind the interview. Secondly, we would introduce the concept behind Pecora, and provide a detailed walkthrough of the application. Thirdly, the user would be presented with a set of predefined tasks to complete. Fourthly, the interviewers would start an open dialogue about the various core components of the application. Lastly, the interviewers would ask several questions regarding the application and associated matters.

12.2 County Governor Interview

After the interview with Steingrim, the supervisor arranged an interview with representatives of the county governor, who are responsible for agriculture in Trøndelag. The interview took place 24th of April. The purpose of the interview was to answer the questions that arose during the interview with Steingrim. Most of the questions were centered around the report the farmer delivers, such as how the report is structured, and who receives it.

During the interview, we learned that the report is sent to both the county governor, as well as Mattilsynet. The reason the reports are sent to Mattilsynet is that they are responsible for the sheep's well being. It is therefore important for them to receive reports, especially if a sheep has been killed or wounded.

From the interview, we also received the form that is used for sending reports, which can be found in Appendix C. The form is intended for an individual farmer's report, but when farmers collaborate, each report is merged and sent as a single report. The form is fairly simple and consists of two different parts. The first is a list of tables where each table is filled out with the details from a single trip. The fields have been translated to English, and are listed as follows:

- **Date** - The date of the trip.
- **Route/area** - The route/area where the trip was performed.
- **Observations** - The observations registered during the trip.

The second part is a summary, where the farmer summarizes information from all the trips that were performed during the season. The fields have been translated to English, and are listed as follows:

- **Animals released on pasture** - Number of animals released of type: sheep, lamb, sheep and lamb (sum), cattle, and goat.
- **Animals lost on pasture** - Number of animals lost of type: sheep, lamb, sheep and lamb (sum), cattle, and goat.
- **Percentage of animals lost** - Percentage of animals lost of type: sheep, lamb, sheep and lamb (sum), cattle, and goat.
- **Individual effort** - Number of days the farmer has spent of type: observation, collecting, other work, and total. Kilometers driven through the season is also registered here.

It was expressed during the interview, that the application needed support for collaboration between farmers. Since many farmers are working together and aggregate their findings into a shared report, it is crucial to have functionality for sharing trips. This has already been implemented in Pecora, in the form of groups, which solves this problem in a flexible way.

Chapter 13

Future Work

This chapter explains the work planned for Pecora in the future. The future work rely on the feedback from the county governor's representatives, and the feedback that we would have gotten from the interview with Steingrim. As the interview never took place, the plans are instead based on the transcript of the interview. In this section, we will present envisioned short-term changes that could be beneficial for the application but was not implemented due to time constraints.

We discussed adding support for multiple scenarios for each observation. This means that one observation can contain different types of scenarios, e.g., a *Wounded sheep* and *Wolf* scenario. This will be helpful when two or more scenarios are observed in proximity to each other. When a scenario is clicked, the details for that scenario will be displayed. Figure 13.1 shows a sketch of how the envisioned change could look.

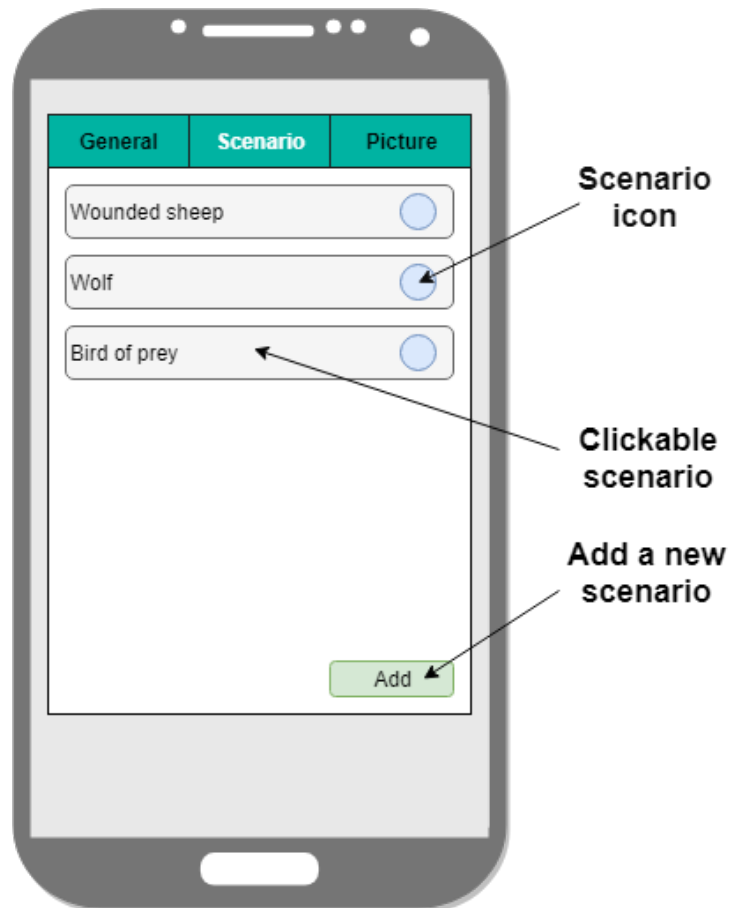


Figure 13.1: Multiple scenarios sketch

We have also discussed improving the design in the profile and reports component. Neither of them are missing functionality, but the user interface looks a bit unpolished compared to other parts of the application. There are also some minor changes that could be done other places in the application to improve the design even further. The design changes are not highly prioritized, as they are mostly aesthetic issues, and do not hinder the user from performing his tasks.

From the transcript, we learned that ewes carry a specific colored tie, indicating how many lambs they have birthed this year. This seems to be an important aspect when registering observations, and it might be necessary to alter the already implemented *Sheep with lamb* scenario to better handle this strict relationship between tie color and the number of lambs. One possible solution is to have the user select a tie color from a palette, which automatically updates the number of lambs.

We found several companies that sell surveillance equipment for livestock, such as Telespor, Shiip, FindMy, and Smartbjella. Through their products, farmers can track the location of

their livestock through GPS. It would be interesting to integrate a communication protocol between Pecora and the surveillance equipment, that would allow Pecora to show the real-time location of livestock. This could potentially be beneficial for both Pecora and the companies, as it provides Pecora with additional functionality and draws customers to the companies' surveillance equipment. It is important to note that purchasing such tools would be entirely optional, as Pecora would work without them. However, it would provide some additional benefits for farmers that are already using GPS trackers.

Another important subject that should be addressed is the report creation process. From the interview with the county governor's representatives, we received the form that is used as a baseline for filling out reports. The form can be found in Appendix C. We believe that this form is not detailed enough, and could be further improved. Therefore, we created a suggestion for how this form could potentially look like. The form is shown in Figure 13.2 and Figure 13.3. The suggested form is based on the insight we have gained on this subject through the course of the project, and the form given to us by the county governor.

In our case, the form would be used by Pecora to generate a complete report. This would eliminate potential errors that could occur if it was filled manually and would also make the creation process significantly more effective. For example, it is time-consuming to accurately estimate time and distance, when performing trips. When using Pecora, this information is included in the report automatically.

Each trip in the suggested form is presented in a more descriptive manner, as it contains more fields compared to the original form. In addition, each observation listed under the trip gives a more precise overview of what happened. For example, the "Lokasjon" field contains a URL which links to a web-based map that makes it possible to see the exact location of the observation.

This report is designed for a single farmer. To accommodate for collaborations another similar form will be used instead. This form will contain a summary for all the farmers that are collaborating, and the information for each individual farmer. The information for the individual farmers will be presented in the same manner as in the depicted form. Using a different form for several farmers, will automate the process of having to merge separate reports, which is currently how it is done.

We still do not have enough information about the current reporting process to be sure that our suggestion is good enough, but we believe that it is a step in the right direction. Therefore, it would be necessary to conduct further dialog with the county governor and ask for feedback.

The last aspect of the future work is the testing already mentioned in Chapter 2 - Research. The supervisor will test Pecora with farmers in a real environment with the purpose of receiving feedback on Pecora, and use it to refine the application further. The farmers would view the application from a point of view that we could not, and provide unique and meaningful feedback about its shortcomings and strengths. This test would provide more thorough feedback than the previously mentioned interview, where Pecora would have been shown to the farmer Steingrim.

Rapportskjema - Tilsyn på utmarksbeite

Denne rapporten er generert av Pecora, og benytter seg av de observasjonene du har utført i løpet av sesongen. Vennligst sjekk at den genererte informasjonen som er oppgitt i rapportskjemaet er korrekt før innsending.

Tilsynsperson: Ola Nordmann
Ar: 2018

Sammendrag av dyr på beite

	Sau	Lam	Totalt
Dyr sluppet på utmarksbeite	50	30	80
Dyr tapt på utmarksbeite	2	0	2
Dyr tapt (%)	4	0	2.5

Sammendrag av egeninnsats

	Tilsyn	Sanking	Annet arbeid	Totalt
Tidsbruk (timer)	9	10	3	28,5

Sammendrag av turer

Antall turer: 2
Total distanse gått: 16 km
Total distanse kjørt: 2,3 km

	Sau	Skadet sau/lam	Død sau/lam	Rovdyr
Antall observert	96	0	2	2

Dato generert: 23.05.2018

Figure 13.2: Suggested form - page 1

Opplisting av Turer

Tur #1		
Dato: 20.04.18		Vær: Overskyet
Startet: 10:30	Avsluttet: 14:40	Tidsbruk: 4 timer og 10 minutter
Startlokasjon: Furuveien		Stoplokasjon: Granveien
Distanse gått: 10,4 km		Distanse kjørt: 2,3 km
Observasjon #1		
Tid observert: 11:05	Lokasjon: https://www.openstreetmap.org/#map=16/63.3928/10.3818	
Beskrivelse: - Flokk med sauer, 30 hvite og 20 svarte - 2 døde sauer		
Kommentar: De døde sauene ble dekket til		
Observasjon #2		
Tid observert: 11:35	Lokasjon: https://www.openstreetmap.org/#map=16/63.3668/10.3556	
Beskrivelse: 2 ulver		
Kommentar: Antakelig ansvarlig for å ha drept sau		

Tur #2		
Dato: 28.04.18		Vær: Regn
Startet: 13:15	Avsluttet: 18:00	Tidsbruk: 4 timer og 45 minutter
Startlokasjon: Eikelia		Stoplokasjon: Furuveien
Distanse gått: 5,6 km		Distanse kjørt: 0 km
Observasjon #1		
Tid observert: 15:45	Lokasjon: https://www.openstreetmap.org/#map=17/63.36485/10.38947	
Beskrivelse: Flokk med sauer, 27 hvite og 19 svarte		
Kommentar:		

Dato generert: 23.05.2018

Figure 13.3: Suggest form - page 2

Chapter 14

Conclusion

In this chapter, we present our conclusion regarding the research questions that were defined in Chapter 2.

RQ1 - How viable is the concept of Pecora for managing and performing observation trips and creating reports for sheep farmers?

As mentioned in Section 12.1, we were unable to join the interview and had to base our evaluation on the provided interview transcript. The transcript states that the farmer expressed the need for functionalities that were missing, such as associating the tie color of ewes with the number of lambs born by that specific ewe. This is not present in our concept, but could easily be implemented in the future. However, we believe that this does not impact the viability of the concept significantly.

Based on the feedback the other groups got from their interview, it would seem like the farmer was satisfied with the presented concept. As our concept is similar to the presented concept, we assume that our solution would be well received, as it contains many of the same features.

From the transcript, it was also revealed that an already existing app was in use by farmers, called BeiteSnap. BeiteSnap aims to solve the same problem as Pecora, but from a slightly different angle. Based on what we learned about BeiteSnap, we believe that Pecora is a better solution, that offers the users more functionality and better management. The existence of BeiteSnap

speaks to the viability of the overarching concept, as it is an already working solution. However, it is apparent that BeiteSnap has room for improvements.

During the interview with the county governor's representatives, the need for the application to support collaboration and sharing of information was expressed. This functionality is already implemented in Pecora as a core feature, in the form of groups.

Based on the interview transcript, the interview with the county governor's representatives, and our examination of BeiteSnap, we believe that the concept of Pecora is viable and could be used in a real setting. We also believe that Pecora is a significant improvement compared to today's solution, in terms of efficiency and functionality.

RQ2 - What benefits do modern architectures, frameworks, and libraries provide when developing mobile applications?

In this project, we used MVVM, RxJava 2, Dagger 2, data binding, and Firebase as our leading technologies. All of these technologies provide a modern approach for developing applications and we were interested in finding out what benefits these technologies provided.

From our experience, the MVVM architecture provided us with a scalable, modifiable and clean code base. One of the significant benefits of MVVM was the separation of concern, which made it easy to organize the logic of the application. Using MVVM also made the application easier to test, due to the natural decoupling of the code base. The testing was also made easier by using Dagger 2 for dependency injection. MVVM also had excellent support for data binding, which greatly reduced the amount of boilerplate code in the application. However, there were some cases where data binding did not work well. For example, the bindings were not easy to debug, if they were not set up properly. There were also some Android views that data binding could not support, such as toolbar menus, which had to be implemented in through view events.

The communication that was done through RxJava 2 made it easy to schedule jobs on different threads and made it very easy to implement the publish-subscribe pattern where needed. The combination of RxJava 2 and Firebase, made it easy to listen for data changes, and display them in the GUI.

The benefits of using modern technologies for developing mobile applications are many, such

as speeding up development significantly while also making it easier to write clean code that is maintainable and follows best practices. However, using these technologies increase the complexity of the code base, and can in some cases be considered over-engineering depending on the issue at hand. Besides, these technologies introduce a steep learning curve for the developers, which could impair the progress of the development process. In the long-term, we believe that the benefits of modern technologies outweigh the disadvantages and is therefore a worthwhile investment. On 8th of May 2018 at the Google I/O conference, Google released Android Jetpack [39]. Android Jetpack is a continuation of many of the technologies that we used, such as MVVM, data binding and more. The fact that Google has decided to support these technologies means that they also believe that these technologies introduce great benefits for developing applications.

Part V

Bibliography & Appendices

Bibliography

- [1] Lovdata. (2013). Forskrift om velferd for småfe, [Online]. Available: <https://lovdata.no/dokument/SF/forskrift/2005-02-18-160> (visited on 05/08/2018).
- [2] B. J. Oates, *Researching Information Systems and Computing*. SAGE Publications Ltd, 2006.
- [3] G. C. Ltd. (2018). Norgeskart, [Online]. Available: <https://play.google.com/store/apps/details?id=no.avinet.norgeskart&hl=nb> (visited on 05/08/2018).
- [4] —, (2018). Skandobs touch, [Online]. Available: <https://play.google.com/store/apps/details?id=com.gw.rovdata&hl=sv> (visited on 05/08/2018).
- [5] N. institutt for naturforskning (NINA). (2018). Skandobs, [Online]. Available: <http://www.skandobs.no> (visited on 05/08/2018).
- [6] G. C. Ltd. (2018). Viltappen, [Online]. Available: <https://play.google.com/store/apps/details?id=se.jagareforbundet.viltappen> (visited on 05/08/2018).
- [7] F. Inc. (2018). React native · a framework for building native apps using react, [Online]. Available: <https://facebook.github.io/react-native/> (visited on 05/18/2018).
- [8] T. A. S. Foundation. (2018). Apache cordova, [Online]. Available: <https://cordova.apache.org/> (visited on 05/18/2018).
- [9] Google. (2018). Android, [Online]. Available: <https://www.android.com/> (visited on 05/08/2018).
- [10] StatCounter. (2018). Mobile operating system market share worldwide, [Online]. Available: <http://gs.statcounter.com/os-market-share/mobile> (visited on 05/16/2018).

- [11] A. Inc. (2018). Ios 11, [Online]. Available: <https://www.apple.com/ios/ios-11/> (visited on 05/08/2018).
- [12] M. Inc. (2018). Windows phones, [Online]. Available: <https://www.microsoft.com/en-us/windows/phones> (visited on 05/08/2018).
- [13] W. Central. (2017). Microsoft's joe belfiore says windows 10 mobile features and hardware are no longer a focus, [Online]. Available: <https://www.windowscentral.com/microsoft-windows-10-mobile-features-and-hardware-are-not-focus-anymore> (visited on 05/16/2018).
- [14] Microsoft. (2018). What's a universal windows platform (uwp) app? [Online]. Available: <https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide> (visited on 05/16/2018).
- [15] M. Badurowicz, "Mvc architectural pattern in mobile web applications," *Actual Problems of Economics*, vol. 6, no. 120, p. 305, 2011.
- [16] Y. Zhang and Y. Luo, "An architecture and implement model for model-view-presenter pattern," in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, IEEE, vol. 8, 2010, pp. 532–536.
- [17] R. Francese, M. Risi, G. Tortora, and G. Scanniello, "Supporting the development of multi-platform mobile applications," in *Web Systems Evolution (WSE), 2013 15th IEEE International Symposium on*, IEEE, 2013, pp. 87–90.
- [18] Realm. (2018). Realm, [Online]. Available: <https://realm.io/> (visited on 05/10/2018).
- [19] Google. (2018). Room, [Online]. Available: <https://developer.android.com/topic/libraries/architecture/room> (visited on 05/10/2018).
- [20] SQLite. (2018). Sqlite home page, [Online]. Available: <https://www.sqlite.org/index.html> (visited on 05/10/2018).
- [21] Google. (2018). Firebase, [Online]. Available: <https://firebase.google.com/> (visited on 05/10/2018).
- [22] A. Parecki. (2018). Oauth community site, [Online]. Available: <https://oauth.net/> (visited on 05/10/2018).
- [23] Mapbox. (2018). Mapbox, [Online]. Available: <https://www.mapbox.com/> (visited on 05/10/2018).

- [24] Google. (2018). Google maps platform, [Online]. Available: <https://cloud.google.com/maps-platform/> (visited on 05/10/2018).
- [25] G. Inc. (2018). Osmdroid, [Online]. Available: <http://osmdroid.github.io/osmdroid/> (visited on 05/10/2018).
- [26] K. Schwaber and M. Beedle, *Agile Project Management with Scrum*. Microsoft Press, 2004.
- [27] C. Ladas, *Scrumban: Essays on Kanban Systems for Lean Software Development*. Modus Cooperandi Press, 2009.
- [28] N. M. A. Munassar and G. Aliseri, "A comparison between five models of software engineering," 2010.
- [29] I. Omoronyia. (2018). Requirements specification and testing part 1, [Online]. Available: <http://www.idi.ntnu.no/emner/tdt4242/materials/TDT4242-lecture-2.pdf> (visited on 05/16/2018).
- [30] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice (3rd Edition)*. Addison-Wesley Longman Publishing Co, Sep. 2012.
- [31] Google. (2018). Activity | android developers, [Online]. Available: <https://developer.android.com/reference/android/app/Activity> (visited on 05/22/2018).
- [32] —, (2018). Fragment | android developers, [Online]. Available: <https://developer.android.com/guide/components/fragments> (visited on 05/22/2018).
- [33] —, (2018). Dialogs | android developers, [Online]. Available: <https://developer.android.com/guide/topics/ui/dialogs> (visited on 05/22/2018).
- [34] —, (2018). Understand the lifecycle activity | android developers, [Online]. Available: <https://developer.android.com/guide/components/activities/activity-lifecycle> (visited on 05/22/2018).
- [35] —, (2018). Configure your build | android developers, [Online]. Available: <https://developer.android.com/studio/build/> (visited on 05/22/2018).
- [36] —, (2018). App manifest overview | android developers, [Online]. Available: <https://developer.android.com/guide/topics/manifest/manifest-intro> (visited on 05/22/2018).

- [37] G. Inc. (2018). Reactivex/rxjava, [Online]. Available: <https://github.com/ReactiveX/RxJava> (visited on 05/15/2018).
- [38] Oracle. (2018). How to write doc comments for the javadoc tool, [Online]. Available: <http://www.oracle.com/technetwork/java/javase/tech/index-137868.html> (visited on 05/16/2018).
- [39] Google. (2018). Android jetpack | android developers, [Online]. Available: <https://developer.android.com/jetpack/> (visited on 05/31/2018).
- [40] F. AS. (2018). Beitesnap, [Online]. Available: <http://www.beitesnap.no/> (visited on 05/16/2018).
- [41] Statista. (2017). App stores: Number of apps in leading app stores 2017, [Online]. Available: <https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/> (visited on 05/16/2018).

Appendix A

Useful Tools

This appendix offers a brief explanation of the tools used throughout the project. It should be emphasized that the tools were largely picked due to us having previous experience with them, thus not requiring time and effort to learn.

A.1 Android Studio

Android Studio is the official Integrated Development Environment (IDE) for the Android operating system. Android Studio was used for writing, debugging, and compiling code written for the Pecora application.

A.2 Android SDK

Android SDK provides the developer tools and the Application Programming Interface libraries that are required to test, build, and debug applications for Android

A.3 Git

Git is a distributed version control system, where your local files is a complete version control repository. These fully-fledged local repositories make it easy to work both offline and remotely. Git allows its users to commit their files locally, and then synchronize their local repository with a repository on a server.

A.4 GitHub

GitHub is an online hosting service for version control using Git. It offers the distributed version control and source code management that Git has, and comes with additional features.

A.5 Trello

Trello is an online project management service. Trello offers a working board that is divided into a user-defined set of columns. In each column, users can place cards, and each card can have a title, a description, and more. Cards can also have a set of users assigned to it. A more definitive explanation of how Trello was used can be found in Section 4.1.

A.6 Google Drive

Google drive is an online file storage and synchronization service. Google Drive offers online real-time collaboration among its users. We used Google Drive to store files and collaborate on text documents.

A.7 ShareLaTeX

ShareLaTeX is an online LaTeX editor that offers real-time collaboration and compiling of projects to PDF. We had previous experience with ShareLaTeX, and it offered a familiar setting for collaboration on larger projects. ShareLaTeX was used to write this thesis.

Appendix B

Interview

This chapter contains the transcript of the interview with Steingrim and a document detailing the plans that we had for our own interview. This transcript was not created by the us, but was rather passed on from the other groups by a request from the supervisor.

B.1 Transcript

A = Andreas Kjerstad

S = Steingrim Horvli

SO = Svein-Olaf Hvasshovd

F = Frida Schmidt-Hanssen

A: Hvilke krav er det til oppfølging av sau, og hvordan foregår dette?

S: For utmarksbeite, er kravet tilsyn minimum én gang pr uke. Når det skjer et eller annet, så skal det økes.

A: Hva kan eksempelvis skje?

S: Et eller annet angrep, eller at du ser det er noe som ikke stemmer i beiteområdet.

SO: Så Steingrim, du sier en gang i uka?

S: Det er minimum én gang pr uke.

SO: Men er det hver eneste sau du skal følge opp én gang i uka?

S: Det har du ikke sjans til. Du skal prøve å få sjekka mesteparten.

SO: Mesteparten, akkurat. Så hvis du har sau som er spredt over fire områder, så er det klart at

du har ikke sjans til å nå over dem én gang i uka.

S: Nei, men så fungerer det slik at vi har organisert tilsyn. Det er utplukket en del personer, mer som en vaktuke. De skal ha minimum fire dager pr uke. De utfyller de områdene vi ikke klarer å sjekke ut. Det kommer i tillegg til den ene dagen vi er pålagt å være ute for å sjekke dyra.

SO: Så det er for å dekke opp et stort område?

S: Det er for å dekke opp et stort område.

SO: Den siste biten her var jeg ikke klar over. Det er vel ikke alle som har det, er det?

S: Stort sett alle områder som er utsatt for rovdyr har det.

SO: Okay. Nei, jeg kan ikke huske at Hallvard hadde det.

S: Jo, dette har vi hatt i alle fall i 15 år.

A: Hvilke utfordringer eksisterer i dag i forbindelse med tilsyn av sau?

S: Utfordringa er å finne dem.

SO: Har du noen triks for å finne dem utover å gå, og prøve på de stedene der du normalt forventer å finne dem?

S: Vi har den radiobjellen vi bruker, for å finne ut hvor de er i terrenget. Så bruker vi dette som utgangspunkt.

A: Hva er hovedårsaken til tap av sau på beite?

S: Det er rovdyr her i området. For noen år siden, da det ikke var rovdyr... mange år fikk vi inn igjen alle dyra.

A: Hvor mange blir typisk tatt i løp av en sesong?

S: Forrige sesong så mistet jeg 90 dyr av 600.

F: Hva er de typiske rovdyrene?

S: I Oppdal nå, så er det jerv, gaupe og ørn.

SO: Har du noen formening om fordeling mellom dem?

S: Nei, det er vanskelig, de har så forskjellige måter å gjøre det på. Kongeørna er typisk tidlig på våren. Da kommer de bort. Gaupen gjemmer det godt, så da er det vanskelig å finne igjen. Jerven er kanskje det enkleste å finne igjen. Han er stort sett i høyfjellet og du klarer å se det på avstand.

SO: Av de 90 som du tapte, er det mest jerv?

S: Problemet er at vi ikke fant igjen så mange, men vi mistenker gaupa i fjor. Vi fikk et ganske stort innrykk av gaupe.

A: Er det noe samarbeid mellom bønder når det gjelder registrering av sau?

S: Jada, det er organisert alt det der.

A: Hvordan foregår det?

S: Det er, som jeg nevnte, den vaktplanen og dyrene holder til i samme område, så når du går en tilsynstur så sjekker du ikke bar dine, du sjekker alle. De er kodemerket, slik at vi vet hvor mange lam hvert dyr skal ha. Hvis jeg treffer nabeons sauer så ser jeg den kodemerkinga, og vet at det dyret skal ha så og så mange lam.

A: Hva slags kodemerking er det?

S: Vi har fargekoding på bjelle.

SO: Du har det på bjella, ikke på slipset?

S: Begge deler, men det henger i bjella alt det her. Da har vi en farge for ett lam, en farge for to lam, og en farge for tre lam. Det er felles for hele Oppdal. De gjør dette andre steder i Norge også, men de bruker ikke samkjørte farger på kodene. Så andre plasser kan de ha andre farger for ett, to og tre lam da.

SO: men innenfor Oppdal så er det én (farge)?

S: Ja, innenfor Oppdal er det én. Men hvis du for eksempel kommer til Rennebu, så kan det være de har noe annet.

A: Finnes det noen utfordringer med deling av informasjon mellom bønder i dag, når det gjelder samarbeid?

S: Nei, egentlig ikke. Det fungerer.

A: Hvordan foregår informasjonsinnsamlingen når du gjennomfører en slik tur? Hva slags teknologi som blir brukt, osv.

S: Tidligere har det bare blitt notert i ei bok, hva vi har gjort om dagene, penn og papir. Men her i fjor tok vi i bruk BeiteSnap, som det kalles. Den fungerte til en viss grad.

SO: Kan du si litt om hvordan den fungerer? Er det en tekstlig beskrivelse, eller noe du fyller

ut?

S: Den fungerer slik at dersom jeg går ut på en tilsynstur så startet jeg logging på den, den logger hele turen. Hvis man ser et eller annet, så tar man et bilde. Dersom det er min, så legger jeg den bare inn i logging. Dersom det er andre sine dyr så legger jeg den ut, og alle som bruker BeiteSnap får den opp på telefonen sin. Det finnes to utgaver av den. En for oss brukere, og en for den vanlige farenfant, og det er en gratisversjon som man kan laste ned. Det er sånn av da jeg la inn den så markerte jeg området dyra mine går på, så alt som skjer utenfor det området det får ikke jeg opp på telefonen min. Men alt som skjer innenfor det kommer inn på telefonen min.

A: Så dersom sauene dine går utenfor dette området så får du ikke informasjon om dem?

S: Nei, men derfor har jeg tegna et ganske stort område. Den skal egentlig ikke gå utenfor.

A: Blir det her brukt av mange turgåere?

S: Det var første året i fjor, det var i bruk. Det var noen, men ikke så mange. Det har kanskje litt med informasjonen ut til publikum her da.

SO: Jeg var ikke klar over den.

S: Nei, den kom ut i fjor.

A: Hva slags informasjon er det som blir registrert i dag?

S: Nei, det er egentlig alt som vi ser. Dersom alt er normalt, så blir det også registrert.

A: Er det noe informasjon som er påbudt av myndighetene å registrere?

S: De har påbudt alt, tror jeg. De skal ha mest mulig informasjon. De skal vite når alt skjer, lokasjon osv. De skal vite hva vi har på utmarksbeite.

A: Er det noen formelle retningslinjer for hva som skal registreres?

S: Nei.

SO: Hvem er det som vil ha denne informasjonen?

S: Mattilsynet og fylkesmannen.

SO: Hvem er det dere sender inn til?

S: Vi sender inn til fylkesmannen.

A: På hvilket format sendes informasjonen inn?

S: Det er ferdige utfylte skjema. Elektroniske skjema som fylkesmannen har. Det er der all informasjonen går ut. All informasjonen som går til fylkesmannen er samla. Vi er et lag som operer i et område, så blir all informasjonen samla inn i det laget og sendt samla. Vi har en samlingsrunde før den datoen, der alle kommer med opplysningene sine, og så blir det sendt videre.

SO: Hvis vi skal prate med fylkesmannen om dette her, har du noe navn?

S: Nei, akkurat no har jeg ikke det, for etter sammenslåinga av fylkene så er det nye folk, så jeg aner ikke.

SO: Har du noe navn fra tidligere?

S: Vi har en... hva er det han heter da? Jeg kommer ikke på navnet hans.

SO: Da er det bare for oss å kontakte fylkesmannen og spørre om dette her.

S: Jeg aner ikke hvem som styrer dette her nå.

A: Det skjemaet, er det noe du laster ned og fyller ut? Eller fylles det ut på nettet?

S: Det er et standard for de lagene.

A: Er dette noe vi kan få tilgang til?

S: Kanskje.

SO: Da er det naturlig at vi kontakter fylkesmannen og spør pent om å få tilgang til det.

S: Det er jo mer som en årsrapport for området dette. Problemet er at det ligger en del personlige opplysninger, siden det er ferdigutfylt av oss, med personopplysninger og alt.

SO: Hvor mange gårder er det som er sammen om dette her?

S: Det er både sau og storfe.

SO: Er det hele søndre Trondheimen?

S: Ja.

A: Så alle dere sender én felles rapport?

S: Ja. Når vi kommer til årets slutt, og skal søke om erstatning, og den biten der, så må du inn med mer utfyllende opplysninger. Da er det hver enkelt.

A: Når du er ute og registrer, hvordan er vanligvis mobildekningen i området?

S: Stort sett er det mobildekning.

A: Er det viktig å registrere hvor du går, når du gjennomfører en slik runde?

S: Det er viktig for oss selv hvor vi har gått. Det er vel ikke så interessant for andre foreløpig. Jeg ser ikke bort fra at det kravet kommer.

A: Hvor mange timer i uka vil du anslå at du bruker på å gå tilsynsturer?

S: Det er varierende. Det avhenger av vært, i tillegg har du oppgaver hjemme du skal utføre også. I snitt tror jeg at jeg bruker én og en halv dag. En dag skal vi regne 7,5 timer på, så 10-12 timer i uka bruker jeg nok på tilsyn.

SO: Bruker du mer tid enn det andre gjør, eller ligger de fleste på omtrent det samme?

S: Jeg tror det er jevnt over ganske bra med tilsyn. Men det varierer litt, noen leier jo hjelp for å gjøre tilsyn for seg og noen gjør det selv.

SO: Jeg spør fordi jeg har forstått at Hallvard brukte mye mindre tid på det. Men du må gå?

S: Jeg har ikke noe vei i områdene, så jeg må gå.

A: Vet du hvordan informasjonen som sendes inn til myndighetene blir brukt?

S: Nei. Det har jeg ikke noen anelse om.

A: Blir det noen reaksjoner dersom sauene ikke blir fulgt opp tilstrekkelig?

S: Da blir det reaksjoner, ja.

A: Vet du hvilke reaksjoner?

S: Nei, det har jeg ikke prøvd ut enda, men Mattilsynet er fort inne i bildet.

SO: Har du hørt om noen som har fått slike reaksjoner?

S: Nei, jeg har ikke det.

A: Du nevnte at du har tatt i bruk BeiteSnap, har du testet noen andre løsninger?

S: Jeg vet ikke om det er noe annet på markedet.

A: Det finner løsninger med GPS-tracking, er dette noe som blir brukt?

S: Ja, jeg bruker det mye. Det er den radiobjella som er. Da får vi sporet hvor de er hele tiden, hvor de er i terrenget. Jeg lurer på om jeg har en 40 slike. Har det på 40 dyr. Problemet er prisen på dem. Det er en forholdsvis høy innkjøpspris, og så er det en forholdsvis høy årspris.

A: Vet du hva den prisen ligger på?

S: Bjella ligger på 1700-1800, og årsabonnementet pr bjelle ligger på rundt 200 kroner.

A: Hvilke fordeler ser du med å bruke slike hjelpemidler som radiobjelle og BeiteSnap? **S:** Radiobjella er en nødvendighet, etter at man har begynt å bruke det. Man har en helt annen kontroll. Den gir beskjed så fort det skjer noe med dyrene. Dersom det er sykdom så får man beskjed.

A: Hvordan detekteres dette?

S: Det er bevegelse. Når dyrene blir dårlige så beveger de seg lite, og det kommer en beskjed om at de bruker så og så lite område at da må vi være oppmerksomme på at det kan være noe galt.

A: Når det gjelder BeiteSnap da?

S: Den må jeg si at jeg ikke har så mye erfaring med enda, men den fungerer, så jeg tror det kan bli et godt hjelpemiddel.

A: Er det noen mangler du ser med BeiteSnap?

S: Problemet i fjor var at det datt ut for oss, så vi fikk ikke fullført loggen. Kanskje den stoppa etter ei periode og da var den dagen egentlig vekkasta. Når det fungerer er det kjempegreier.

SO: Har du noen formening om årsaken til at det datt ut?

S: Nei, det var et problem de hadde, hva som forårsaket det vet jeg ikke.

SO: Er det et privat firma som driver dette?

S: Ja, jeg tror det er et privat firma.

F: Krevde den appen at man hadde mobilnett?

S: Nei. Du kunne gjøre alt med den når du var utenfor dekning, og så fort du hadde dekning, så gikk det ut. Kartet var nedlastet.

SO: Var det mange av dere som brukte dette?

S: I laget vårt så var vi vel en fire-fem, tror jeg.

A: Da tror jeg vi har kommet gjennom spørsmålene jeg har her.

SO: Er det noe du vil fortelle oss, Steingrim, som vi burde ha spurt om?

S: Nei, jeg tror egentlig vi har fått det med oss. Det er jo klart det er store utfordringer med dette her da.

F: Var det slik at BeiteSnap hadde en kombinasjon av at man kan registrere ting man så og den GPS-sporingen med radiobjella?

S: Ikke radiobjella, de er to forskjellige ting, så det går ikke på radiobjella, nei.

SO: Det du kunne registrere var bilder?

S: Ja, og tekst.

F: Er det ofte slik at du observerer sauene fra avstand?

S: Ja det er mye av det vi gjør. Kikkerter noe av det viktigste vi har med oss. Det er et av de viktigste redskapene man har.

F: For eksempel når du tar bilder, er det når du finner noe som er nærmere?

S: Det er stort sett hvis det er et eller annet som har skjedd. Døde dyr, syke dyr, eller dyr som mangler lam. Da prøver jeg å få tatt bildet av nummeret på det dyret.

SO: Du ser på det på fargen på slipset? Det skulle vært tre lam, men det er bare to, så er det åpenbart at her mangler det noe.

S: Ja, så da må vi prøve å få notert det, og finne ut hvorfor det ikke er der.

SO: Er manglende lam noe av det mest vanlige du ser?

S: Ja. Det er da vi begynner å få mistanke om at noe har skjedd og begynner å sette inn ekstra ressurser i det området.

SO: Det er jo klart at lammet kan ha byttet mamma.

S: Det kan det godt. Det kan være sammen med andre dyr. Det må registreres dyr som har med seg for mange lam også.

SO: Jeg har vært en del ute og fulgt opp sau, og da må jeg si at erfaringsmessig er det vanskelig å skille lam og voksne dyr. I alle fall utpå året.

A: Er det noe merking av lam, som gjør at de skiller seg fra voksne?

S: Ikke noe annet enn det første nummeret i øreklypa, den indikerer fødselsår.

SO: Men å få sett det tallet er ikke noe du får gjort.

S: Det er ikke noe lett å se det, nei.

B.2 Interview Plan

This section contains the plan for how the interview with Steingrim would proceed. The plan was intended to be followed as a guideline during the interview.

B.2.1 Introduction

Introduce ourselves briefly.

B.2.2 Pecora

Pecora is an Android application to help register information during observation trips for farmers. The purpose of Pecora is to replace the pen and paper used today, and make it a more streamlined process. Hopefully, it will make it easier to register data, work together with other farmers, and easily generate a report for the whole season.

B.2.3 Presentation of the Application

We will only show the functions that we believe are the most important. The application is a ‘work-in-progress’ and could use some more polishing at the time of this interview.

B.2.3.1 Prerequisites

1. Download demo map data before the presentation.
2. Create a group “Demo gruppe”, where both Alexander and Thomas are members .
3. Ensure that we have at least one trip that shows a real trip before the presentation (track, start, stop, 3 observation, 1 linking), for the group “Demo gruppe”.

B.2.3.2 Login

1. Explain that you can create an account with Google, Facebook or email and password.
2. Explain that all users have an online profile. This allows you to access your data on several devices. For example if you lose your device, or want to use multiple devices.
3. Explain that you do not have to log in more than once.
4. Log in with a Google account.

5. Show the profile that you are logged in with in the navigation drawer.

B.2.3.3 Map Data

1. Explain why you need map data.
 - (a) If you want to take a trip in an area with poor reception, then it is necessary to download the area beforehand.
 - (b) Permanent access to the maps you download.
2. Click on the + icon.
3. Explain that the center of the square is where the user clicks on the map.
4. Explain that the blue square shows the area which will be downloaded.
5. Explain that you can move the square.
6. Change the size with the slider.
7. Click on the check icon.
8. Explain the download dialog.
9. Set area name to “Demo map” (zoom: 12-13) and click download.
10. Explain the list of downloaded map areas.

B.2.3.4 Groups

1. Navigate to the groups component.
2. Explain what groups are, and why they are useful.
 - (a) Explain that the farms form groups together.
 - (b) Explain that the group members can see each other trips.
3. Show create group dialog, and group details / member details.
4. Navigate to the trips component.
5. Go to the “Demo Group” tab.
6. Explain that the trips are separated by the groups in which they belong to.
 - (a) Explain that My Trips are trips that belong to you, and are not in a group.
7. Create a new test trip for the group “demo group”.

B.2.3.5 Trip 1

1. Create a new trip (My Trips).

2. Explain how to navigate around the map (zoom in/out, move around).
3. Create a new observation.
 - (a) Explain the Scenario first tab.
 - i. Create Sheep Herd observation.
 - A. Explain that white and grey sheep are registered as white sheep.
 - B. Explain that you pick the most dominant color when deciding whether the sheep is black or white.
 - ii. Explain the other observation types.
 - (b) Explain the General tab.
 - (c) Explain the Pictures tab.
 - i. Take picture of something to show how it works.
 - ii. Explain that it is also possible to load from gallery.
4. Navigate back, without completing the trip.
5. Show that the trip is in progress, since we didn't complete it.
6. Click on the trip in progress and complete it.

B.2.3.6 Trip 2

1. Click on the finished trip in "Demo Group".
2. Explain concepts.
 - (a) Start / Stop.
 - (b) Track.
 - (c) Observations.
 - (d) "Linking".
3. Explain the fab buttons.
4. Explain the toolbar buttons.

B.2.3.7 Reports

1. Navigate to the reports component.
2. Create a new report.
 - (a) Select group "Demo group".
 - (b) Select year 2018.
3. Open the report.

- (a) Explain the Report tab.
 - (b) Explain the Summary tab.
4. Show that you can easily send the report through mail.

B.2.3.8 Timeline

1. Navigate to the timeline component.
2. Change group to Demo Group.
3. Explain that you can change year.
4. Show that you can easily change between trips.
5. Show that you can open observations.

B.2.4 User Assignments

The following assignments will be given to the interview subject, and he will be asked to complete them. The assignments are open-ended, meaning that the subject will complete them without any further guidance.

1. Download a map area.
2. Complete a trip with two observations.
3. Generate report for the group which you completed the trip for.

B.2.5 Open Dialogue

Start a open dialogue with the interview subject regarding the following areas:

- Trips.
- Map Data.
- Groups.
- Reports.
- Timeline.

B.2.6 Interview

This is a semi-structured interview. The interview subject is asked whether he agrees to having the interview recorded or not. If not, the interview is not recorded and the only record will be what was typed during the interview by the interviewees. The following questions were asked by the interviewees and answered by the interview subject. The answers were written down and can be found below the individual questions.

NOTE: Since the meeting never came in to fruition, the questions were not asked nor answered.

How streamlined does the application feel, e.g. easy to navigate around?

-

How intuitive is the application?

-

How do you perceive the aesthetics of the application?

-

Does the interview subject prefer one-by-one view of the trips, or see a collection of trips together on the same map in the timeline component?

-

Which methods do you currently use when registering sheep observations?

-

What are the positive aspects of how the registration is done today?

-

What are the negative aspects of how the registration is done today?

-

Who do you report your findings to?

-

What requirements are in place for the information being reported?

-

What could be improved about today's solution?

-

How often do you perform observation trips?

-

What is your opinion in regards to what other farmers are doing?

-

How long does an observation trip usually last?

-

How far do you usually walk on an observation trip?

-

Would it be beneficial to see the farmer or shepherd performing the trip in real time, in the application?

-

What if the shepherd or farmer is a member of one of your groups

-

Do you collaborate with other any farmers in regards to performing observation trips?

-

How does the collaboration function? How do you share information, etc.?

-

What is your opinion about the presented solution?

-

What worked well with the application presented to you?

-

What could have worked better with the application presented to you?

-

Should there be a web application also, where you can view the observation trips?

-

Do you know if any other entities would be interested in an application like the one presented here? E.g. NINA (Norsk Institutt for Naturforskning) or Nortura, if the application is commercialized.

-

Do you think the application has any other usages besides the one presented here? E.g. used as a way to broadcast predator sightings.

-

Have you or do you currently use any other technological approaches in conjunction with the observation trips?

-

Appendix C

County Governor Report

This chapter presents report given to us by the county governor. It had two pages in total.

Organisert beitebruk

RAPPORTSKJEMA – TILSYN PÅ UTMARKSBEITE

Dette skjemaet kan nyttast for å sikre beitelaget ein dokumentasjon på utført tilsyn med dyra i utmarka gjennom beitesesongen, slik det er sett krav om i *Forskrift om tilskott til organisert beitebruk*. Det er lagt opp til at kvar tur skal noterast fortløpande på dette skjemaet. Av aktuelle opplysningar kan nemnast daude og skadde dyr, årsak, rovvilt, laushundar, generell uro, søyer som manglar lam, beiteforhold og anna. Kartfesting av observasjonar som tillegg til dette skjemaet kan vera aktuelt. Etter endt beitesesong skal oppsummeringsskjemaet på neste side (baksida) fyllast ut og sendast leiaren i laget.

Beitelag: **Beiteår:**

Tilsynsperson:

Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	

Figure C.1: County governor form - page 1

Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	
Dato:	Rute/område:
Observasjonar:	

Oppsummering av beitesesongen - informasjon om dyretal, tap og eigeninnsats, inkludert km køyring

	Sau	Lam	Sau + lam	Storfe	Geit/kje
Dyr sleppt på utmarksbeite					
Dyr tapt på utmarksbeite					
Tap i % (dyr tapt/dyr sleptx100)					
Eigeninnsats, dagar:	Tilsyn	Sanking	Anna arbeid	Samla	Køyring, km

Figure C.2: County governor form - page 2

Appendix D

Usability Testing

This appendix contains the questions, information, and tasks given to the subjects of the usability tests. The following sections are presented as they were presented during the usability test, thus when the text mentions "you" it refers to the subject of the usability test.

D.1 Personal Questions

The following questions are designed to create a clearer identity of the subject that is performing the usability test. The questions are as follows:

- What is your name?
- How old are you?
- What is your education?
- How technically competent would you describe yourself as?
 - How often do you use applications? Daily, weekly, etc.

D.2 Introduction

Throughout the usability test you will perform a variety of tasks that cover the core features of Pecora. After the tasks are complete, you will be asked to answer few open questions regarding the application, and fill out a form in regard to its user experience.

Today, farmers are required by law to regularly record information about their sheep on pasture. To do this, the farmer has to go for an observation trip and note down what was observed, usually with pen and paper. To make this process easier, farmers have the option of forming collaboration groups, where each member takes turns on performing these observation trips. At the end of the season, all farmers must submit a report stating what has been observed. This is especially important in case of a dead or wounded sheep, as it might prove difficult to receive financial compensation from the government without it.

To make this process easier, we have developed an application called Pecora. Pecora is an Android application that gives farmers the possibility to register their observations on a mobile device during an observation trip, replacing the need for pen and paper. In addition, Pecora will also generate a standardized report for all the observation trips performed from the past season. Once the report is generated, it is ready to be handed in to the government.

D.3 Tasks

It is important to stress that the application is being tested, and not your ability to complete the tasks presented here. If the task is not completed, it gives a clear indication that the application is lacking, thus it is not your fault. In addition, we ask you to explain your thought process behind each of the tasks being performed.

Pecora is developed for farmers, thus it is desirable that you try to get into the role of a farmer, and perform the tasks to your best ability.

The following sections list the tasks to be completed.

D.3.1 Log In

- Log in to Pecora.

D.3.2 Trip - 1

- Start a new trip.
- Find and view the trip details.

- Provide your start location.
- Set the number of participants to 2.
- Create a new observation north of your current location.
 - Register 8 white sheep.
 - Register 3 black sheep.
 - Write a comment.
- Create a new observation east of your current location.
 - Register 5 reindeer.
 - Set the radius of the observation to 100 meters.
 - Set the direction to north-west.
- Delete the first observation.
- Complete the trip.

D.3.3 Map Data

- Download a map for your current location (zoom level 13-16).
- Open and view the map data you downloaded.
- Download a map for Trondheim (zoom level 13-14).
- Delete the map data for Trondheim.

D.3.4 Groups 1

- Create a new group called 'Group 1'.
- Invite email X to the group (X is given to the user during the test).

D.3.5 Trip 2

- Start a new trip for 'Group 1'.
- Create a new observation south of your current location.
 - Register 2 foxes.
 - Take a picture of one the "foxes".
 - Load a picture for the gallery.
 - Delete one of the pictures.
- Center the map on your current location.

- Complete the trip.

D.3.6 Report

- Create a new report for the trips performed by the group 'Group 1' in the year 2017.
- Open and view the report.
- Create a new report for the trips performed by the group 'My Reports' in the year 2017.
- Send the 'My Reports' report through email.
- Delete the 'My Reports' report.

D.3.7 Trips

- View the trips completed in 2016.
- View the trips completed in 2017.
- Move a trip from 'My Trips' to 'Group 1'.
- Delete all the trips in 'Group 1'.

D.3.8 Groups 2

- Remove the invited person from 'Group 1'.
- Delete 'Group 1'.

D.3.9 Settings

- Change the language to Norwegian.
- Remove cached map data.
- Allow mobile data to be used for downloading.

D.3.10 Log out

- Log out of the application.

D.3.11 SUS Form

Please fill out Table D.2 using the values shown in Table D.1, placing an 'X' in the appropriate column.

1	Strongly disagree
2	Disagree
3	Neutral
4	Agree
5	Strongly agree

Table D.1: SUS form values

Nr	Question	1	2	3	4	5
1	I think that I would like to use this system.					
2	I found the system unnecessarily complex.					
3	I thought the system was easy to use.					
4	I think that I would need the support of a technical person to be able to use this system.					
5	I found the various functions in the system were well integrated.					
6	I thought there was too much inconsistency in this system.					
7	I would imagine that most people would learn to use this system very quickly.					
8	I found the system very cumbersome to use.					
9	I felt very confident using this system.					
10	I needed to learn a lot of things before I could get going with this system.					

Table D.2: SUS form

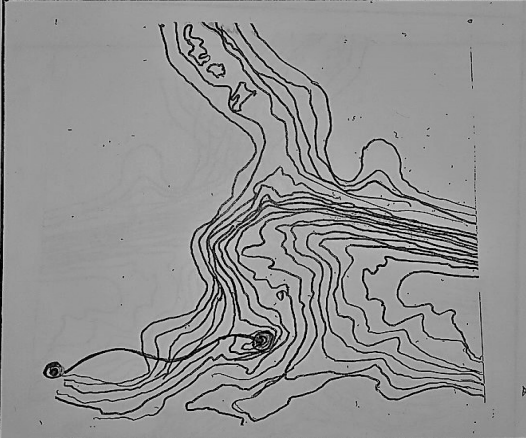
Appendix E

Sketches

This appendix contains the sketches that were given to us by the supervisor at the beginning of the project.

Start gjeteretur

Gjeter: Sørensen
 Antall deltagere: 2
 Start: 07.45 22.07.15
 Fra: Hytta - Høisingveien
 Vær: Småregn
 Tekst: Sjekk rapport skadd lam i Hyttedalen.
 Slur nedl. OK



Steder

- Nytt
- Forrige
- Neste

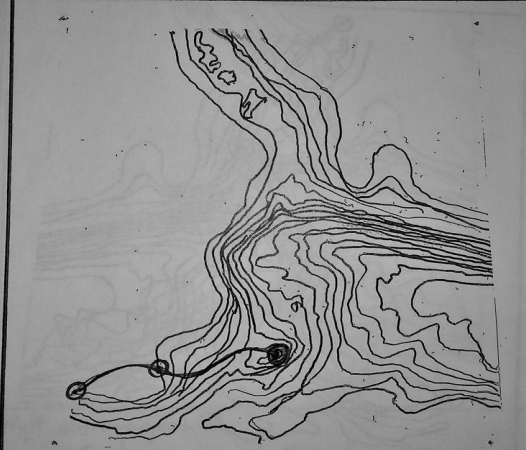
← N →

Observasjoner

- Søye med lam
- Søye alene
- Lam alene
- Skadd søye
- Skadd lam
- Død søye
- Dødt lam

- Hund med eier
- Lyshund alene
- Rovfugl
- Rovdyr
- Reinsdyr
- Annet

← Forrige Neste →

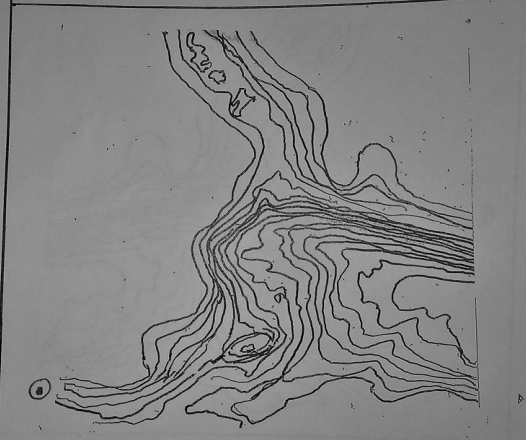


Avlyst Foto

Pecora

- Start gjeteretur
- Areal gjeteretur
- Løst ned til PC
- Restant etter stopp

OK



Avlyst Foto

Observasjoner

- Nytt obs. sted
- Forrige obs. sted
- Neste

- Søye m. lam
- Søye alene
- Lam alene
- Skadd søye
- Skadd lam
- Død søye
- Dødt lam
- uljann

- Hund m. eier
- Hund alene
- Rovfugl
- Rovdyr
- Reinsdyr
- Annet

← →

Nytt observasjonssted

- Sentur
- Forflytning
- OK

← Areal →

← → ↖ ↗ ↘ ↙

Avlyst Foto

Steder

- Nytt obs.sted
- Forrige obs.sted
- Neste

Observasjoner

- Søyre m. lam
- Søyre alene
- Lam alene
- Skadd søyre
- Skadd lam
- Død søyre
- Dødt lam
- Forrige
- Hand m. air
- Hund alene
- Rovfagl
- Rovdyr
- Reinsdyr
- Annet

Neste →

Nytt observasjonssted

- Senter
- Areal ⇔
- Forflytning ⇌ ↗ ↘ ↙ ↚
- OK

Avlyst Foto

Søyre med lam

Rase: Norsk kvit
 Farge: Hvit
 Eier: ~~Haltur~~ Stori
 Øremerke forside: Grønn
 Øremerke bakside: Grønn
 Slips: Gul
 Bløkt-merkeplate: Manglar

Lam: 2
 Tekst: Inne i markens
 Vanskelig obs

OK

Avlyst Foto

Søyre alene

Rase: Spæl
 Farge: Grø
 Eier: ~~Stori~~ Horvli
 Øremerke foran: Grønn
 Øremerke bak: Grønn
 Slips: Blå
 Bløkt-merkeplate: Manglar
 Tekst: Teft på fjærlvannet

OK

Avlyst Foto

Skadd søyre

Rase: Spæl
 Farge: Grø
 Eier: Nergarden
 Øremerke foran: Grønn
 Øremerke bak: Grønn
 Slips: Gul
 Bløkt-merkeplate: Ingen
 Lam: 2

Skade: Haltur, skadd bakst
 Kontakttat: Hallvard Stubi
 Beslutning: La gå
 Tekst:

OK

Avlyst Foto

7.

Lam alene

Rase: Norsk knit
 Farge: Hvit
 Eier: Ukjent
 Øremerke fjerne: Manglende
 Øremerke bak: Manglende
 Antall: 2
 Tekst: Går alene!

■ OK

■ Avlyst

■ Foto

Skadd lam

Rase: Spæl
 Farge: Hvit & grå
 Eier: Hovuli
 Øremerke fjerne: Orange
 Øremerke bak: Grøn
 Skade: Hodeskade
 Kontaktet: Hallvord Storli
 Beslutning: Venter med endelig beslutning
 Tekst: Bliar ikke - sterkt vest.

■ OK

■ Avlyst

■ Foto

Dead soya

Rase: Spæl
 Farge: Hvit
 Eier: Hovuli
 Øremerke fjerne: Orange
 Øremerke bak: Grøn
 Slips: Ikke på plass
 Bildet: mørke plate: uten
 Lam: Ingen
 Skade/Dedersak: Ukjent

Antatt drept av: Jern
 Ca. drept av: 2. Tuke siden
 Kontaktet: Stungum Hovuli
 Beslutning: Bar hjem.
 Bekket over: Ikke relevant
 Tekst: For en jobb

■ OK

■ Avlyst

■ Foto

Ull funn

Rase: Ukjent
 Farge: Mørk grå / svart
 Kontaktet: Hallvord Storli
 Beslutning: Ta med ullprose,
 Tekst: Ull sprett over 100 x 50 m

■ OK

■ Avlyst

■ Foto

Dødt lam

Rase: Norsk kvit
 Farge: Hvit
 Slad: Storh, Nargondan
 Beskrivelse farge: Svart
 Ansett døpt av: Ukjent
 Ca døpt nær: 1/2 km siden
 Odsårsak: Ukjent

Kontaktet: Kunde Stoli
 Beslutning: Bar hjem
 Bekket over: Ja!
 Tekst: For røttel til 2 km hjem

OK

Avlyst Foto

Leshund med eier

Hund:
 Rase: Fuglehund
 Farge: Brun
 Pels: Hår lang
 Kjenne tegn: Hållend
 Tekst: Skremmer
 Svarflakk

Eier:
 Navn: Ukjent
 Alder: ca 30-40
 Bukt: Svart
 Hørfarge: Lys
 Våpen: Rifle
 Andre prøver: Knive
 Tekst: Nombna sakke

Sald: 10-30 km
 Fange: Svart

OK

Avlyst Foto

Hund alene

Rase: Fuglehund
 Farge: Brun
 Pels: Langhåret
 Kjenne tegn: Lyse bein
 Tekst: Forfølger svarflakk

OK

Avlyst Foto

Reinsdyr

Antall: Ca. 40
 Tekst: Sto stille på svarflakk

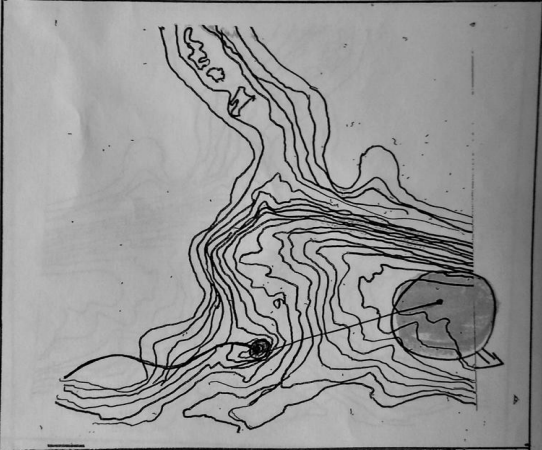
OK

Avlyst Foto

Rovfagl

Type: Kongedrn
 Fange: Bred
 Kjenntegn: Mestet fjer på høye vinge
 Tekst: Sirklet.

OK



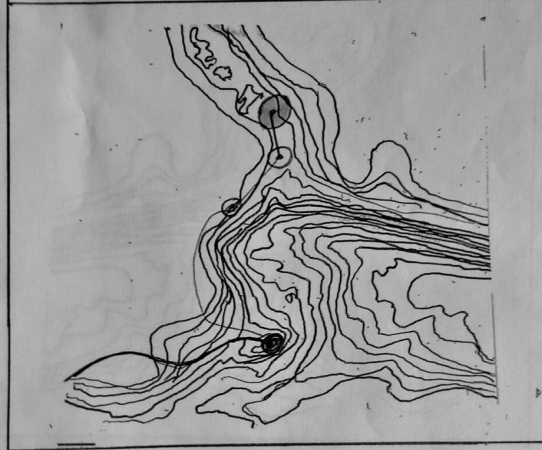
Avlyst

Foto

Rovdyr

Type: Fjerv
 Antall: 1
 Tekst: Hvit stripe på siden langattlar.

OK



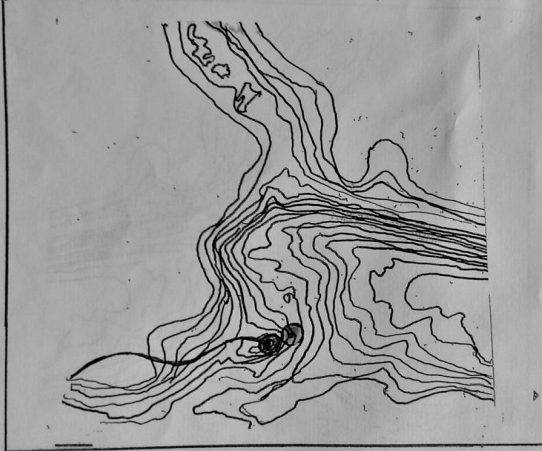
Avlyst

Foto

Annet

Type dyr: Haggorm
 Antall: Ca 40 cm lange haggormer. La og jatte seg.
 Tekst: Trakk seg raskt unna.

OK



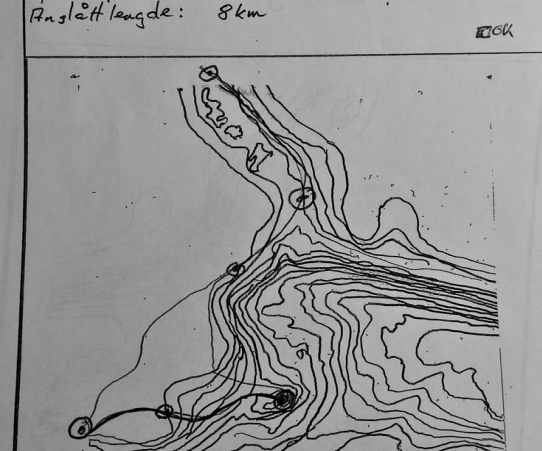
Avlyst

Foto

Avslutt gjestetur

Gjeter: Soghus
 Antall deltagere: 2
 Start: 07.45 22.07.15
 Avslutte: 17.30 22.07.15
 Fra: Hytta - Høisingvassan
 Til: Hytta - Høisingvassan
 Fineste punkt: Hyttedalen
 Avslutt lengde: 8 km

OK



Appendix F

System Test Cases

This appendix contains the system test cases that were performed during the system tests of the Pecora application. Each test case assumes that the application is in a state where the overarching component is active prior to starting the test. This means that when the navigation drawer tests are performed, it is assumed that the navigation drawer is open and active in the application.

F.1 Login

This section contains the system test cases for the login requirements found in Table 5.2.

Test ID	STC 1.01
Requirements	1.01 - Pecora shall be able to log in a user with his email and password
Preconditions	The user is not logged in to the application with an existing user
Input	<ul style="list-style-type: none">• Provide the username of the existing user account• Provide the password of the existing user account• Click the 'Log In' button
Expected result	The user is logged in to the application with the given username and password
Result	The user was logged in to the application with the given username and password
Comment	-
Passed	Yes

Table F.1: STC 1.01

Test ID	STC 1.02
Requirements	1.02 - Pecora shall be able to log in a user with his Google account
Preconditions	The user is not logged in to the application with an existing user
Input	Click the 'Sign in with Google' button
Expected result	The user is logged in to the application with the given Google credentials
Result	The user was logged in to the application with the given Google credentials
Comment	-
Passed	Yes

Table F.2: STC 1.02

Test ID	STC 1.03
Requirements	1.03 - Pecora shall be able to log in a user with his Facebook account
Preconditions	The user is not logged in to the application with an existing user
Input	Click the 'Sign in with Facebook' button
Expected result	The user is logged in to the application with the given Facebook credentials
Result	The user was logged in to the application with the given Facebook credentials
Comment	-
Passed	Yes

Table F.3: STC 1.03

Test ID	STC 1.04
Requirements	1.04 - Pecora shall log in the user automatically if the user has already logged in once before
Preconditions	<ul style="list-style-type: none"> • The user has already logged in to the application with an existing user • The user has not logged out of the application
Input	The user launches a fresh instance of the application on his device
Expected result	The user is automatically logged in to the application
Result	The user was automatically logged in to the application
Comment	-
Passed	Yes

Table F.4: STC 1.04

F.2 Create User

This section contains the system test cases for the create user requirements found in Table 5.2.

Test ID	STC 2.01
Requirements	2.01 - Pecora shall be able to create a new user account with a user defined email and password
Preconditions	The user is not logged in to the application with an existing user
Input	<ul style="list-style-type: none">• Click the 'Create User' button• Provide a username, email, and password• Click the 'Create' button
Expected result	A new user account is created with the given credentials
Result	A new user account was created with the given credentials
Comment	-
Passed	Yes

Table F.5: STC 2.01

Test ID	STC 2.02
Requirements	2.02 - Pecora shall be able to create a new user with the user's Google account
Preconditions	The user is not logged in to the application with an existing user
Input	<ul style="list-style-type: none">• Click the 'Create User' button• Click the 'Join With Google' button• Provide your Google account credentials
Expected result	A new user account is created with the provided Google credentials
Result	A new user account was created with the provided Google credentials
Comment	-
Passed	Yes

Table F.6: STC 2.02

Test ID	STC 2.03
Requirements	2.03 - Pecora shall be able to create a new user with the user's Facebook account
Preconditions	The user is not logged in to the application with an existing user
Input	<ul style="list-style-type: none"> • Click the 'Create User' button • Click the 'Join With Facebook' button • Provide your Facebook account credentials
Expected result	A new user account is created with the provided Facebook credentials
Result	A new user account was created with the provided Facebook credentials
Comment	-
Passed	Yes

Table F.7: STC 2.03

Test ID	STC 2.04
Requirements	2.04 - Pecora shall automatically log in the user if his account has been created
Preconditions	The user is not logged in to the application with an existing user
Input	<ul style="list-style-type: none"> • Click the 'Create User' button • Create a new account using either Google, Facebook, or a custom email and password
Expected result	A new user account is created and the user is logged in to the application with the provided credentials
Result	A new user account was created and the user was logged in to the application with the provided credentials
Comment	-
Passed	Yes

Table F.8: STC 2.04

F.3 Navigation Drawer

This section contains the system test cases for the navigation drawer requirements found in Table 5.2.

Test ID	STC 3.01
Requirements	3.01 - Pecora shall be able to log out the user
Preconditions	The user is logged in to the application with an existing user account
Input	<ul style="list-style-type: none">• Click the "Hamburger" icon to open the navigation drawer• Click the 'Log out' button in the navigation drawer
Expected result	The user is logged out of the application
Result	The user was logged out of the application
Comment	-
Passed	Yes

Table F.9: STC 3.01

Test ID	STC 3.02
Requirements	3.02 - Pecora shall show the user's avatar icon
Preconditions	The user is logged in to the application
Input	-
Expected result	The user's avatar icon is shown
Result	The user's avatar icon was shown
Comment	-
Passed	Yes

Table F.10: STC 3.02

Test ID	STC 3.03
Requirements	3.03 - Pecora shall show the name of user
Preconditions	The user is logged in to the application
Input	-
Expected result	The user's name is shown
Result	The user's name was shown
Comment	-
Passed	Yes

Table F.11: STC 3.03

Test ID	STC 3.04
Requirements	3.04 - Pecora shall show the email of the user
Preconditions	The user is logged in to the application
Input	-
Expected result	The user's email is shown
Result	The user's email was shown
Comment	-
Passed	Yes

Table F.12: STC 3.04

F.4 Trips

This section contains the system test cases for the trips requirements found in Table 5.2.

Test ID	STC 4.01
Requirements	4.01 - Pecora shall list the completed trips associated with the logged in user and his groups
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either completed a trip himself or an associated group has a completed trip
Input	-
Expected result	The completed trips are listed for the user to see
Result	The completed trips were listed for the user to see
Comment	-
Passed	Yes

Table F.13: STC 4.01

Test ID	STC 4.02
Requirements	4.02 - Pecora shall show the trip in progress if a trip is currently in progress
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip in progress
Input	-
Expected result	The trip currently in progress is shown
Result	The trip currently in progress was shown
Comment	-
Passed	Yes

Table F.14: STC 4.02

Test ID	STC 4.03
Requirements	4.03 - Pecora shall separate shown trips into their associated groups
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user is a member of at least two groups • The user has completed a trip for two groups and a personal trip
Input	-
Expected result	The trips are separated in to their respective groups, or shown as a personal trip
Result	The trips were separated in to their respective groups, or shown as a personal trip
Comment	-
Passed	Yes

Table F.15: STC 4.03

Test ID	STC 4.04
Requirements	4.04 - Pecora shall be able to show a trip's observations and track on a map
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either completed a trip or has a trip in progress
Input	Click one of the listed trips
Expected result	The the user is shown a new screen where the trip's observations and track is presented on a digital map
Result	The the user was shown a new screen where the trip's observations and track was presented on a digital map
Comment	-
Passed	Yes

Table F.16: STC 4.04

Test ID	STC 4.05
Requirements	4.05 - Pecora shall be able to create a new trip
Preconditions	The user is logged in
Input	<ul style="list-style-type: none"> • Click on the + fab button • Click 'Start' • Select 'My Trips' • Click 'Select'
Expected result	A new personal trip is started
Result	A new personal trip was started
Comment	-
Passed	Yes

Table F.17: STC 4.05

Test ID	STC 4.06
Requirements	4.06 - Pecora shall be able to delete a trip if the trip is owned by the logged in user
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user is the owner of at least one trip
Input	<ul style="list-style-type: none"> • Click the 'Trash Can icon • Click on the trip owned by the user • Click the 'Check Mark' icon
Expected result	The trip is deleted
Result	The trip was deleted
Comment	-
Passed	Yes

Table F.18: STC 4.06

Test ID	STC 4.07
Requirements	4.07 - Pecora shall be able to move trips to other groups if the user owns the trips
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user is the owner of at least one trip • The user is a member of at least one group
Input	<ul style="list-style-type: none"> • Click the 'Move' icon • Click on the trip owned by the user • Click the 'Check Mark' icon • Select a group • Click the 'Select' button
Expected result	The trip is moved to the specified group
Result	The trip was moved to the specified group
Comment	-
Passed	Yes

Table F.19: STC 4.07

Test ID	STC 4.08
Requirements	4.08 - Pecora shall be able to filter trips by year
Preconditions	The user is logged in
Input	<ul style="list-style-type: none"> • Click the 'Filter Year' button • Select a year • Click the 'Select' button
Expected result	The listed trips are filtered and the only trips shown are from the specified year
Result	The listed trips were filtered and the only trips were are from the specified year
Comment	-
Passed	Yes

Table F.20: STC 4.08

F.5 Trip

This section contains the system test cases for the trip requirements found in Table 5.2.

Test ID	STC 5.01
Requirements	5.01 - Pecora shall be able to start a new trip
Preconditions	The user is logged in
Input	<ul style="list-style-type: none"> • Click the + fab button • Click the 'Start' button • Select a group • Click the 'Select' button
Expected result	A new trip is started for the selected group
Result	A new trip is started for the selected group
Comment	-
Passed	Yes

Table F.21: STC 5.01

Test ID	STC 5.02
Requirements	5.02 - Pecora shall be able to save a completed trip
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip in progress
Input	<ul style="list-style-type: none"> • Open the trip in progress • Click the 'Check Mark' icon • Click the 'OK' button
Expected result	The trip in progress is saved and completed
Result	The trip in progress was saved and completed
Comment	-
Passed	Yes

Table F.22: STC 5.02

Test ID	STC 5.03
Requirements	5.03 - Pecora shall show the downloaded map tiles on a map
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has downloaded a map area • The user does not have an internet connection • The user has cleared the map tile cache
Input	<ul style="list-style-type: none"> • Do either: <ul style="list-style-type: none"> - Start a new trip - Open a trip in progress
Expected result	The downloaded map area is displayed
Result	The downloaded map area was displayed
Comment	-
Passed	Yes

Table F.23: STC 5.03

Test ID	STC 5.04-5.06
Requirements	<ul style="list-style-type: none"> • 5.04 - Pecora shall show the center of the map through a crosshair • 5.05 - Pecora shall be able to show the GPS track of a user during a trip • 5.06 - Pecora shall be able to show the start GPS location of a trip
Preconditions	The user is logged in
Input	<ul style="list-style-type: none"> • Do either: <ul style="list-style-type: none"> – Start a new trip – Open a trip in progress
Expected result	The center of the map view is shown through a marker, and the user's start location and track is shown
Result	The center of the map view was shown through a marker, and the user's start location and track was shown
Comment	-
Passed	Yes

Table F.24: STC 5.04-5.06

Test ID	STC 5.07
Requirements	5.07 - Pecora shall be able to show the end GPS location of a trip
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either <ul style="list-style-type: none"> – Completed a trip himself – Access to completed trip through a group
Input	Open a completed trip
Expected result	The last registered location is shown as the end location
Result	The last registered location was shown as the end location
Comment	-
Passed	Yes

Table F.25: STC 5.07

Test ID	STC 5.08-5.15
Requirements	<ul style="list-style-type: none"> • 5.08 - Pecora shall be able to create a new observation • 5.09 - Pecora shall be able to let the user set the observation radius • 5.10 - Pecora shall be able to let the user set the observation direction • 5.11 - Pecora shall be able to let the user set the observation scenario to sheep, lamb, wool, predator, reindeer, dog, and other • 5.12 - Pecora shall be able to add a new picture from the user's gallery • 5.13 - Pecora shall be able to capture and add a new picture by using the device's camera • 5.14 - Pecora shall show pictures related to the selected observation • 5.15 - Pecora shall be able to let the user delete pictures associated with an observation
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip in progress

Input	<ul style="list-style-type: none"> • Open the trip in progress • Click the + fab icon • Adjust the radius slider • Click the 'Direction' button • Select a direction • Click outside the direction dialog • Click the 'Scenario' tab • Set the scenario to sheep, lamb, wool, etc. • Click the 'Picture' tab • Click the 'Load Picture' button • Select a picture • Click the 'Take Picture' button • Take a picture • Click the 'Trash Can' icon • Select a image • Click the 'Check Mark' icon • Click the 'OK' button
Expected result	A new observation is created with the given radius, direction, scenario, and pictures. The picture taken/loaded is shown in the observation dialog, and the picture selected for deletion is deleted.
Result	A new observation was created with the given radius, direction, scenario, and pictures. The picture taken/loaded was shown in the observation dialog, and the picture selected for deletion was deleted.
Comment	-
Passed	Yes

Table F.26: STC 5.08-5.15

Test ID	STC 5.16
Requirements	5.16 - Pecora shall be able to let the user display a picture in fullscreen
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either <ul style="list-style-type: none"> – A trip in progress – A completed trip available • The trip has an observation • The observation has a picture
Input	<ul style="list-style-type: none"> • Open the trip • Click the observation on the digital map • Click on the 'Picture' tab • Click on the picture
Expected result	The picture is shown in a fullscreen view
Result	The picture was shown in a fullscreen view
Comment	-
Passed	Yes

Table F.27: STC 5.16

Test ID	STC 5.17
Requirements	5.17 - Pecora shall be able to let the user delete an observation
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either <ul style="list-style-type: none"> – A trip in progress – A completed trip available • The trip has an observation
Input	<ul style="list-style-type: none"> • Open the trip • Click the observation on the digital map • Click on the 'Delete' button • Click on the 'OK' button
Expected result	The observation is deleted
Result	The observation was deleted
Comment	-
Passed	Yes

Table F.28: STC 5.17

Test ID	STC 5.18
Requirements	5.18 - Pecora shall be able to let the user edit observations if the trip is not completed
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip in progress • The trip has an observation
Input	<ul style="list-style-type: none"> • Open the trip • Click the observation on the digital map • Edit any observation dialog fields
Expected result	The observation is edited
Result	The observation was edited
Comment	-
Passed	Yes

Table F.29: STC 5.18

Test ID	STC 5.19
Requirements	5.19 - Pecora shall be able to let the user view the trip's observations and track
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip available • The trip has at least one observation • The trip has a user track
Input	Open the trip
Expected result	The trip's observations and track is shown on a digital map
Result	The trip's observations and track was shown on a digital map
Comment	-
Passed	Yes

Table F.30: STC 5.19

Test ID	STC 5.20
Requirements	5.20 - Pecora shall be able to center the map on the user's location
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip available
Input	<ul style="list-style-type: none"> • Open the trip • Click the 'My Location' icon
Expected result	The digital map is centered on the user's location
Result	The digital map was centered on the user's location
Comment	-
Passed	Yes

Table F.31: STC 5.20

Test ID	STC 5.21-5.22
Requirements	<ul style="list-style-type: none"> • 5.21 - Pecora shall be able to link observations from different locations • 5.22 - Pecora shall be able to let the user delete links between an observation and a location
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip available • The trip has at least one observation
Input	<ul style="list-style-type: none"> • Open the trip • Center the map on a observation • Click the 'Link Observation' fab button • Click the linked line from the old location • Click the 'OK' button
Expected result	The observation is linked to a new location, and the link to the old location is deleted
Result	The observation was linked to a new location, and the link to the old location was deleted
Comment	-
Passed	Yes

Table F.32: STC 5.21-5.22

Test ID	STC 5.23-5.24
Requirements	<ul style="list-style-type: none"> • 5.23 - Pecora shall be able to find and set the start location name based on the user's location • 5.24 - Pecora shall be able to find and set the end location name based on the user's location
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip available
Input	<ul style="list-style-type: none"> • Open the trip • Click on the 'Trip Details' icon • Click the 'My Location' button for the start location • Click the 'My Location' button for the end location
Expected result	The start and end location name is found based on the user's location
Result	The start and end location name was found based on the user's location
Comment	-
Passed	Yes

Table F.33: STC 5.23-5.24

Test ID	STC 5.25
Requirements	5.25 - Pecora shall be able pause a trip
Preconditions	The user is logged in
Input	<ul style="list-style-type: none"> • Click the + fab button • Click the 'Start' Button • Select a group • Click the 'Select' button • Click the 'Back' icon
Expected result	The trip is paused
Result	The trip was paused
Comment	-
Passed	Yes

Table F.34: STC 5.25

Test ID	STC 5.26
Requirements	5.26 - Pecora shall be able to highlight an observation and its associated links if the user centers the map on the observation
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip available • The trip has at least one observation
Input	<ul style="list-style-type: none"> • Open the trip • Center the digital map on the observation
Expected result	The observation and its associated links are highlighted
Result	The observation and its associated links were highlighted
Comment	-
Passed	Yes

Table F.35: STC 5.26

Test ID	STC 5.27
Requirements	5.27 - Pecora shall be able to navigate between observations if the user clicks the 'Next Observation' button
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip available • The trip as two or more observations
Input	<ul style="list-style-type: none"> • Open the trip • Click the 'Next Observation' button
Expected result	The digital map is centered on the next observation
Result	The digital map was centered on the next observation
Comment	-
Passed	Yes

Table F.36: STC 5.27

Test ID	STC 5.28
Requirements	5.28 - Pecora shall be able to navigate to the closest observation if the user clicks the 'Closest Observation' button
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip available • The trip has a observation
Input	<ul style="list-style-type: none"> • Open the trip • Click the 'Closest Observation' button
Expected result	The digital map is centered on the closest observation
Result	The digital map was centered on the closest observation
Comment	-
Passed	Yes

Table F.37: STC 5.28

Test ID	STC 5.29
Requirements	5.29 - Pecora shall be able to open an observation and show its details
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a trip available • The trip has a observation
Input	<ul style="list-style-type: none"> • Open the trip • Navigate to the observation • Click the 'Open Observation' fab button
Expected result	The observation is opened the observation details are shown
Result	The observation was opened the observation details were shown
Comment	-
Passed	Yes

Table F.38: STC 5.29

F.6 Map Data

This section contains the system test cases for the map data requirements found in Table 5.2.

Test ID	STC 6.01
Requirements	6.01 - Pecora shall show a list of downloaded areas
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has downloaded at least one area
Input	-
Expected result	All the downloaded areas are displayed as a list to the user
Result	All the downloaded areas were displayed as a list to the user
Comment	-
Passed	Yes

Table F.39: STC 6.01

Test ID	STC 6.02
Requirements	6.02 - Pecora shall be able to delete a downloaded area
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has downloaded at least one area
Input	<ul style="list-style-type: none"> • Click the 'Trash Can' icon • Select a downloaded map area from the list • Click the 'Check Mark' icon
Expected result	The downloaded map area is deleted
Result	The downloaded map area was deleted
Comment	-
Passed	Yes

Table F.40: STC 6.02

Test ID	STC 6.03
Requirements	6.03 - Pecora shall be able to show the downloaded area if the user selects it
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has downloaded at least one area
Input	Click a downloaded map area
Expected result	The downloaded map area is shown
Result	The downloaded map area was shown
Comment	-
Passed	Yes

Table F.41: STC 6.03

Test ID	STC 6.04
Requirements	6.04 - Pecora shall be able to rename a downloaded area
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has downloaded at least one area
Input	<ul style="list-style-type: none"> • Click a downloaded map area • Click the 'Pencil' icon • Edit the name of the area • Click the 'Save' button
Expected result	The name of the downloaded map area is changed
Result	The name of the downloaded map area was changed
Comment	-
Passed	Yes

Table F.42: STC 6.04

Test ID	STC 6.05-6.09
Requirements	<ul style="list-style-type: none"> • 6.05 - Pecora shall be able to download a map area • 6.06 - Pecora shall be able to adjust the size of the map download area • 6.07 - Pecora shall be able to place the map download area with hand gestures on a map • 6.08 - Pecora shall be able to let the user set a name for the downloaded area • 6.09 - Pecora shall be able to let the user set the minimum and maximum zoom for the downloaded area
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The device must have an internet connection
Input	<ul style="list-style-type: none"> • Click the + fab button • Adjust the size slider • Click the map to center the map area to be downloaded • Click the 'Check Mark' icon • Provide a name for the map area • Adjust the maximum and minimum zoom sliders • Click the 'Download' button
Expected result	The map area with the defined size, center, zoom levels, and name is downloaded
Result	The map area with the defined size, center, zoom levels, and name was downloaded
Comment	-
Passed	Yes

Table F.43: STC 6.05-6.09

Test ID	STC 6.10
Requirements	6.10 - Pecora shall be able to find and set the downloaded area's name based on its location
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The device must have an internet connection
Input	<ul style="list-style-type: none"> • Click the + fab button • Click the 'Check Mark' icon • Click the 'My Location' icon
Expected result	The name of the map area is set based on the user's location
Result	The name of the map area was set based on the user's location
Comment	-
Passed	Yes

Table F.44: STC 6.10

F.7 Timeline

This section contains the system test cases for the timeline requirements found in Table 5.2.

Test ID	STC 7.01
Requirements	7.01 - Pecora shall be able to show a timeline for trips completed by a specific group for a given year
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either completed a trip himself or an associated group has a completed trip
Input	<ul style="list-style-type: none"> • Click the 'Group' button • Select a group • Click the 'Select' button • Click the 'Year' button • Select a year • Click the 'Select' Button
Expected result	A timeline for all the trips for the selected group and year is shown
Result	A timeline for all the trips for the selected group and year was shown
Comment	-
Passed	Yes

Table F.45: STC 7.01

Test ID	STC 7.02
Requirements	7.02 - Pecora shall be able to show the trip details dialog if a trip is available in the timeline
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either completed a trip himself or an associated group has a completed trip
Input	Click the 'Trip Details Dialog' icon in the toolbar
Expected result	The trip details dialog for the selected trip is shown
Result	The trip details dialog for the selected trip was shown
Comment	-
Passed	Yes

Table F.46: STC 7.02

Test ID	STC 7.03
Requirements	7.03 - Pecora shall be able to reverse the order of the timeline trips
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either completed two trips himself or an associated group has two completed trips
Input	Click the 'Reverse Order' icon in the toolbar
Expected result	The order of the trips in the timeline is reversed
Result	The order of the trips in the timeline is reversed
Comment	-
Passed	Yes

Table F.47: STC 7.03

Test ID	STC 7.04-7.07
Requirements	<ul style="list-style-type: none"> • 7.04 - Pecora shall be able to show a trip on a map if the user selects it • 7.05 - Pecora shall be able to show the GPS track of a user for a trip • 7.06 - Pecora shall be able to show the start GPS location of a trip • 7.07 - Pecora shall be able to show the end GPS location of a trip
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either completed a trip himself or an associated group has a completed trip • No trip has been selected
Input	<ul style="list-style-type: none"> • Click the 'Group' button • Select the group with the completed trip • Click the 'Select' button • Click the 'Year' button • Select the year with the completed trip • Click the 'Select' Button
Expected result	The trip is shown on a digital map with the GPS track of the user, and the trip's start and stop location
Result	The trip was shown on a digital map with the GPS track of the user, and the trip's start and stop location
Comment	-
Passed	Yes

Table F.48: STC 7.04-7.07

Test ID	STC 7.08
Requirements	7.08 - Pecora shall be able to highlight an observation and its associated links if the user centers the map on the observation
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either completed a trip himself or an associated group has a completed trip • The completed trip has been selected
Input	Navigate the center of the map to a observation
Expected result	The observation and its associated links are highlighted
Result	The observation and its associated links were highlighted
Comment	-
Passed	Yes

Table F.49: STC 7.08

Test ID	STC 7.09
Requirements	7.09 - Pecora shall be able to navigate between observations if the user clicks the 'Next Observation' button
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either completed a trip himself or an associated group has a completed trip • The completed trip has been selected • The completed trip has two or more observations
Input	Click the 'Next Observation' button
Expected result	The digital map is centered on the next observation
Result	The digital map was centered on the next observation
Comment	-
Passed	Yes

Table F.50: STC 7.09

Test ID	STC 7.10
Requirements	7.10 - Pecora shall be able to navigate to the closest observation if the user clicks the 'Closest Observation' button
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either completed a trip himself or an associated group has a completed trip • The completed trip has been selected • The completed trip has one or more observations
Input	Click the 'Closest Observation' button
Expected result	The digital map is centered on the closest observation
Result	The digital map was centered on the closest observation
Comment	-
Passed	Yes

Table F.51: STC 7.10

Test ID	STC 7.11
Requirements	7.11 - Pecora shall be able to show the selected observation's registered information
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has either completed a trip himself or an associated group has a completed trip • The completed trip has been selected • The completed trip has one or more observations
Input	Click on a observation on the digital map
Expected result	The observation dialog is shown with the observation's registered information
Result	The observation dialog was shown with the observation's registered information
Comment	-
Passed	Yes

Table F.52: STC 7.11

F.8 Groups

This section contains the system test cases for the group requirements found in Table 5.2.

Test ID	STC 8.01
Requirements	8.01 - Pecora shall show a list of all the groups that the user is a member of
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user is a member of at least one group
Input	-
Expected result	The groups that the user is a member of is shown
Result	The groups that the user is a member of was shown
Comment	-
Passed	Yes

Table F.53: STC 8.01

Test ID	STC 8.02
Requirements	8.02 - Pecora shall be able to create new groups
Preconditions	The user is logged in
Input	<ul style="list-style-type: none"> • Click the + fab button • Provide a name for the group • Click the 'Create' button
Expected result	The group with the defined name is created with the user as its owner
Result	The group with the defined name was created with the user as its owner
Comment	-
Passed	Yes

Table F.54: STC 8.02

Test ID	STC 8.03
Requirements	8.03 - Pecora shall be able to show all the members of a group if the user selects the group
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user is a member of at least one group
Input	Click one of the listed groups
Expected result	All the members of the group is shown
Result	All the members of the group was shown
Comment	-
Passed	Yes

Table F.55: STC 8.03

Test ID	STC 8.04
Requirements	8.04 - Pecora shall be able to let users delete a group if they are the owner of the group
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user is the owner of a group
Input	<ul style="list-style-type: none"> • Click one of the listed groups • Click the 'Trash Can' icon • Swipe the slider to the right
Expected result	The group is deleted
Result	The group was deleted
Comment	-
Passed	Yes

Table F.56: STC 8.04

Test ID	STC 8.05
Requirements	8.05 - Pecora shall be able to let users change group name if they are the owner of the group
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user is the owner of a group
Input	<ul style="list-style-type: none"> • Click one of the listed groups • Click the 'Pencil' icon • Provide a new name for the group • Click the 'Save' button
Expected result	The name of the group is changed
Result	The name of the group was changed
Comment	-
Passed	Yes

Table F.57: STC 8.05

Test ID	STC 8.06
Requirements	8.06 - Pecora shall be able to let users transfer ownership of the group if they are the owner
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user is the owner of a group • The group has at least two members
Input	<ul style="list-style-type: none"> • Click one of the listed groups • Click one of the other members • Click the 'Make Owner' button • Click the 'OK' button
Expected result	The owner of the group is changed
Result	The owner of the group was changed
Comment	-
Passed	Yes

Table F.58: STC 8.06

Test ID	STC 8.07
Requirements	8.07 - Pecora shall be able to let users leave a group
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user is a member of a group
Input	<ul style="list-style-type: none"> • Click one of the listed groups • Click the listed item that is associated with the user • Click the 'Leave' button • Click the 'OK' button
Expected result	The user is no longer a member of the group
Result	The user was no longer a member of the group
Comment	-
Passed	Yes

Table F.59: STC 8.07

Test ID	STC 8.08
Requirements	8.08 - Pecora shall show a list of all group invitations
Preconditions	The user is logged in
Input	Click the 'Invitations' tab
Expected result	The user's invitations are listed
Result	The user's invitations were listed
Comment	-
Passed	Yes

Table F.60: STC 8.08

Test ID	STC 8.09
Requirements	8.09 - Pecora shall be able to let users send a group invitation if they are the owner of the group
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user is the owner of a group
Input	<ul style="list-style-type: none"> • Click one of the listed groups • Click the 'Invite' fab button • Provide the email of the user being invited • Click the 'Send' button
Expected result	The user that is associated with the email is invited to the group
Result	The user that is associated with the email was invited to the group
Comment	-
Passed	Yes

Table F.61: STC 8.09

Test ID	STC 8.10
Requirements	8.10 - Pecora shall be able to accept a group invitation
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a group invitation
Input	<ul style="list-style-type: none"> • Click the 'Invitations' tab • Click the invitation • Click the 'Accept' button
Expected result	The user accepts the group invitation and joins the group
Result	The user accepted the group invitation and joined the group
Comment	-
Passed	Yes

Table F.62: STC 8.10

Test ID	STC 8.11
Requirements	8.11 - Pecora shall be able to decline a group invitation
Preconditions	<ul style="list-style-type: none"> • The user is logged in • The user has a group invitation
Input	<ul style="list-style-type: none"> • Click the 'Invitations' tab • Click the invitation • Click the 'Decline' button
Expected result	The user decline the group invitation and does not join the group
Result	The user declined the group invitation and did not join the group
Comment	-
Passed	Yes

Table F.63: STC 8.11

F.9 Profile

This section contains the system test cases for the profile requirements found in Table 5.2.

Test ID	STC 9.01
Requirements	9.01 - Pecora shall be able to change the user's avatar icon
Preconditions	The user is logged in
Input	<ul style="list-style-type: none">• Click the 'Change' button• Select a picture
Expected result	The user's avatar icon is changed
Result	The user's avatar icon was changed
Comment	-
Passed	Yes

Table F.64: STC 9.01

Test ID	STC 9.02
Requirements	9.02 - Pecora shall be able to change the user's username
Preconditions	The user is logged in
Input	<ul style="list-style-type: none">• Click the 'Change' button• Edit the username• Click the 'Save' button
Expected result	The user's username icon is changed
Result	The user's username icon was changed
Comment	-
Passed	Yes

Table F.65: STC 9.02

F.10 Settings

This section contains the system test cases for the settings requirements found in Table 5.2.

Test ID	STC 10.01
Requirements	10.01 - Pecora shall be able to change application language
Preconditions	The user is logged in
Input	<ul style="list-style-type: none"> • Click on the 'Language' setting • Click on the preferred language
Expected result	The application language is changed
Result	The application language was changed
Comment	-
Passed	Yes

Table F.66: STC 10.01

Test ID	STC 10.02
Requirements	10.02 - Pecora shall be able to clear out cached map data
Preconditions	The user is logged in
Input	<ul style="list-style-type: none"> • Click on the 'Map Data' setting • Click on the 'OK' button
Expected result	The cached map data is cleared
Result	The cached map data was cleared
Comment	-
Passed	Yes

Table F.67: STC 10.02

Test ID	STC 10.03
Requirements	10.03 - Pecora shall be able to toggle on/off mobile network usage for downloading map data
Preconditions	The user is logged in
Input	Click on the 'Mobile Network' setting
Expected result	Mobile network usage for downloading map data is toggled on/off
Result	Mobile network usage for downloading map data was toggled on/off
Comment	-
Passed	Yes

Table F.68: STC 10.03

Test ID	STC 10.04
Requirements	10.04 - Pecora shall be able to toggle on/off map rotation through hand gestures
Preconditions	The user is logged in
Input	Click on the 'Map Rotation' setting
Expected result	Map rotation is toggled on/off
Result	Map rotation was toggled on/off
Comment	-
Passed	Yes

Table F.69: STC 10.04

Test ID	STC 10.05
Requirements	10.05 - Pecora shall be able change location distance interval for tracking user movement
Preconditions	The user is logged in
Input	<ul style="list-style-type: none"> • Click on the 'Distance Interval' setting • Select the preferred distance • Click the 'Select' button
Expected result	The location distance interval is changed
Result	The location distance interval is changed
Comment	-
Passed	Yes

Table F.70: STC 10.05

F.11 Help

This section contains the system test cases for the help requirements found in Table 5.2.

Test ID	STC 11.01
Requirements	11.01 - Pecora shall provide information about how to use the application in the help section
Preconditions	The user is logged in
Input	-
Expected result	The user is presented with information on how to use the application
Result	The user is presented with information on how to use the application
Comment	-
Passed	Yes

Table F.71: STC 11.01