NTNU – Trondheim
Norwegian University of
Science and Technology

01_FFI_android
Situation Awareness And Incident Reporting

Torgrim Bøe Skårsmoen
Chun Fan
Junjun Guo
Morten Opdahl
Simen Teigen Søgård
Liang Zhu

Mai 2015

PROJECT THESIS
Department of Computer and Information Science
Norwegian University of Science and Technology

Supervisor: Alfredo Pérez Fernández

Acknowledgement

This project consumed big amount of work and dedication. Still, implementation would not have been possible if we did not have guidance, help and support from our course coordinator, supervisor and customers. Therefore we would like to extend our sincere gratitude to all of them.

First of all we are thankful to our course coordinator Monica Divitini for organizing and provide the great opportunity for this wonderful project, and the important guidance at the beginning of the Course.

We are also grateful to our supervisor Alfredo Perez Fernandez for taking his time to give us valuable feedback and for giving us guidance when we needed it the most.

We would also extend our most sincere gratitude to Frank T. Johnsen and Trude H. Bloebaum from The Norwegian Defence Research Establishment(FFI) for their cooperation, giving us very useful feedback on the application as well as on our report, and for providing us with the necessary developing tools.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project Introduction

With the rapid advances in smart device technology and the increasing integration of smart devices into our culture, the military has been implementing solutions to adapt this technology for military purposes. A report from the United States Department of Defence(DoD)suggested that, the integration of the smart devices into the military infrastructure, could provide its mobile workforce with greater access to information and improving operational advantages[17]. Specialized applications could then be developed for said smart devices to utilize this new infrastructure.

The aim of this project is to develop an application that can handle operational intelligence gathering through the use of smart devices. The application will be developed by 6 students from the Norwegian University of Science and Technology(NTNU) and will be developed for the Android platform. This report will include all the necessary documentation to ensure the reader gets a comprehensive understanding of how the application performs and the development process used to achieve the end goal.

1.2 Report Structure

The report is structured to ensure that the information is conveyed in a clear and concise manner.

The purpose of chapter 1 is to introduce the reader to problem description and provide some background information on the team working on the application

Chapter 2 goes into detail about the development environment, including the tools used and the justification of said tools.

Chapter 3 is dedicated to the project management portion of the project. This includes information on the chosen development process and time management.

Chapter 4 presents the requirement analysis part of the project. Including use cases and scenarios that were implemented to get a better understanding of the clients requirements.

Chapter 5 incorporates the design aspect of the project. This includes information regarding the architecture of the system, the design of the client, and the design of the database.

Chapter 6 presents the implementation details of the project. This chapter will provide a detailed look at the solutions incorporated to solve the problem description.

Chapter 7 is dedicated to the testing part of the application. This chapter will present test cases and methods that were used to ensure the application was stable and robust.

Chapter 8 will conclude the report with an evaluation of the project as a whole. The development process will be discussed with respect to solutions that were implemented and implications that arose along the way.

1.3 Project Description and Goals

The goal of this project is to develop a situational awareness and incident reporting application for the Norwegian Defence Research Establishment(FFI). The application is meant to apply military functionality to civilian commercial products. In this case it will be developed for Android to support various smart phones and tablets. The main purpose of this application is to aid teams in the field with intelligence gathering and establish a robust network of information sharing between the teams on the ground and central command. This will enable the operational teams and central command to get a clearer overview of the situation in the area of operation.

To accomplish this the application will have to feature a map that tracks the soldiers positions and displays relevant operational reports on the map. The devices will have to communicate via either Wifi provided by a military vehicle, or a mobile network. Reports and team locations will need to be stored on a centralized server from which the application can request the necessary information and update the devices accordingly. The choice of map format and database implementation will have an impact on the performance and functionality of the application and will have to be considered.

Since the application will be used in military context there are a variety of environments the application will be tried in, therefore it must come out as robust and simple to use. There are two variables in particular that need to be considered.

The application will have to function in a DIL(Disconnected Intermittent Limited) environment. A DIL environment is an environment in which the network can be unstable. To get around this issue the application will need to store unsent information on the device until the connection is restored. However this introduces a second complication. If the device is constantly trying to transmit information while there is no connection, this can have a major impact on the battery life. High bandwidth rates will also reduce the lifetime of the battery substantially. This implies that the compression software utilized will have a major impact on the devices up time. Since the operational teams might not have regular access to methods of charging the device this poses a problem.

To get around these issues a few factors will have to be taken into account. The intervals at which locations and reports are sent, and the web service and compression software utilized in the application.

1.4 Customer

The customer is the Norwegian Defence Research Establishment(FFI). Their establishment is the chief adviser on defence-related science and technology to the ministry of defence and the Norwegian Armed Forces. The Establishment focuses particularly on developments in science and military technology, which have an impact on political security and defence planning [5].

Demand for FFI's research increases as the nature of armed conflict evolves. Particularly new communications technologies in new areas of conflict is just one of the many fields in which FFI is currently investing resources. The overall goal for FFI is to enable men and women in uniform to be more effective in the field and ensure safety on the job, as well as their safe return.

Our main contacts at FFI were Frank T. Johnsen and Trude Hafsøe Bloebaum.

1.5 Team Background

The developer team consist of the following members:

Junjun Guo
Previous experience related to this project: Java, Android, MySQL, Git and Github

Chun Fan
Previous experience related to this project: Java, Spring, Hibernate, Tomcat, MySQL, MSSQL, SQLite, Android

Morten Opdahl
Previous experience related to this project: Java,MySQL and Git

Torgrim Bøe Skårsmoen
Previous experience related to this project: Java, MySQL, SQLite, Sharelatex, Git and Github

Simen Teigen Søgård
Previous experience related to this project: Java and MySQL

Liang Zhu
Previous experience related to this project: Java, Github and MySQL

Chapter 2

Development Environment

The goal of this chapter is to provide the reader with a comprehensive view of the tools and processes that were utilized to develop the application. Since the development environment will dictate the functionality of the application, a lot of time was spent researching which tools would be appropriate for the development of the application.

2.1 IDE

An Integrated development environment(IDE) is an application which brings together multiple tools used when developing a software product and can make developing an application more automated. Most common among these tools are: an editor with which you can write and edit text, tools that compile written text into something that is executable for the CPU(Central processing unit), debugger for quickly finding errors, easy library management, and version control. Since Android uses the Java programming language there was a choice between two of the most used IDEs for android development.

2.1.1 Eclipse

Eclipse is the most commonly used IDE when developing Java applications. The standalone android sdk for Eclipse was considered. This alternative was initially examined as there was substantial information available on how to develop android applications in Eclipse. Unfortunately the Android Developer Tool for eclipse was discontinued after Android Studio left beta. For this reason the team decided against it. It was imperative that the IDE being used was up to date as to avoid conflicts later in the development cycle.

2.1.2 Android Studio

After researching both options the team decided on Android Studio. It is specifically designed by Google to develop software for the android platform and just recently released its first stable build. Some specific reasons for picking Android Studio include:

- The platform is updated regularly
- It supports increased functionality for developing android applications in comparison with Eclipse, including android code completion
- Improved user interface support
- It uses the gradle build environment which ensures the entire team can work with a consistent environment
- Most of the modern tutorials on the Internet, including Google's own getting started guide, focus on Andriod Studio as the main IDE

2.2 Version Control and Code management

When developing a relatively large product it is preferable to use tools for handling the codebase. The size of the code can quickly become considerable and changes to the code needs to be managed. One of the tools used is concerned with version control of the code. Another tool that is needed, at least when working together with a team of developers, is something that makes it easy to share code with others.

When it comes to version control there are various options to choose from, where som are Bazzar, svn and git. Git was chosen on the basis that it is a reliable tool, and everybody on the team had experience with Git from other projects.

As a team working on a product it is often not satisfactory with using Git as a standalone. Git alone does not provide sharing of source code in an easy manner. To alleviate this problem there is a separate web client that is often used together with Git to make code sharing easy. One of these websites is Github, and as the name implies, it works as a hub for Git repositories.

2.3 Report documentation

Since this was first and foremost a class project, the documentation of the working process had to receive as much attention as the program itself. To ensure that the editing and sharing of the documentation was as seamless as possible the team settled on a few tools to provide the necessary assistance.

2.3.1 Google Docs

To ensure it is easy to produce the required documentation for the project, the team had some requirements regarding the writing tools:

- The tools would have to make sharing of documents as easy and pain-free as possible.
- There should be support for editing the documents concurrently.
- It should be easy to convert the document to its final form i.e. pdf
- The tools have to be stable so that work is not lost or at risk of being lost.

With this in mind there were some options available to choose from. Dropbox is a cloud solution that lets you store files on remote servers. Because of its lack of concurrent editing it was decided against using dropbox. The team researched some others office applications like Thinkfree and Zoho, but eventually came to the conclusion that Google drive would be sufficient for composing the preliminary and midterm report.

2.3.2 ShareLatex

For the final delivery a more flexible tool was needed. Sharelatex supports the same features as google docs such as, document sharing, concurrent editing, and cloud storage. Although it can be more cumbersome to write a report in Sharelatex, the end result is better formatted and structured due to its built in functionality.

2.4 Communication

In this section the tools utilized to improve communication will be discussed.

2.4.1 Skype & Facebook

Skype and Facebook were the team's main tools used for communication throughout the project. Skype is an application that can be used to communicate via voice chat, video chat or instant messaging. As a result of the customer being located at Kjeller near Oslo, Skype was used as the tool for customer communication. Within the group the social networking service Facebook was used to communicate when not in the same room. The reason for choosing Facebook over Skype for this task was that the likelihood of getting quicker responses on messages were decided to be higher with Facebook.

2.5 Web services

A web service is a group of software functions. The functions are provided at a network address over the Web with the service always on. The functions together construct a software system in order to support interoperable machine-to-machine interaction over a network. Two of the most popular standards for web service are SOAP and REST. Using public standards would make the functions much easier to be used.

2.5.1 SOAP

SOAP(Simple Object Access Protocol) is type of messaging protocol that provides an envelope for sending information via a network [14]. It allows programs that are run on different operating systems to communicate with each other. It uses XML as its message format but relies on other application layer protocols such as HTTP or SMTP for message negotiations and transmissions. SOAP was the first web service implemented during the course of development. REST would later be implemented due to reasons stated in the next section.

2.5.2 REST

REST(Representational State Transfer) is a software architecture style that incorporates a set of principles that determine how networked resources should be defined and addressed[12]. It is one of the alternatives to SOAP for implementing a web service. A RESTful architecture uses HTTP commands such as GET,POST,PUT, or DELETE to exchange information via the network. REST was considered a good alternative to SOAP as it works especially well with java and android applications.REST also permits many different data formats unlike SOAP which is XML only. Since the battery life has to be considered for this application, it was necessary to implement different web services to test the alternatives.

2.6 Web Frameworks

A web framework is a software framework that is designed to support the development of dynamic websites, web applications, web services and web resources. The framework aims to alleviate the overhead associated with common activities performed in web development. Using web framework libraries of third-party and open source would make implementation easy, stable, extensible.

2.6.1 Spring Framework

The Spring Framework is an open source application framework and inversion control that provides a comprehensive programming and configuration model for modern Java-based applications. The dependency injection(Springs' inversion control) helps to glue java classes together while still keeping them separated allowing for easy reuse of code. Since the application needs to be developed according

to some very specific requirements, several solutions have to be implemented and analysed. The Spring Framework will make it easy for several implementations.

2.6.2 Apache CFX Framework

Apache CXF is an open-source web services framework [2]. It can be used for implementing web services Java. These services can speak a variety of protocols including SOAP and REST. The CXF framework can be embedded in web services such as Spring, which was one of the main reasons why it was chosen, since spring will be used for both the SOAP and the RESTful version of the web service. Although CXF can technically be used with the RESTful architecture, it was concluded that the Spring MVC framework would be easier to implement.

2.6.3 Spring MVC Framework

As mentioned in section 2.7.2 the Spring MVC framework would be implemented for the RESTful architecture. Spring MVC is the web component of the Spring framework. It offers a model-view-controller architecture that includes the necessary components to develop flexible and loosely coupled web applications. Since it uses the MVC architecture, it separates the different aspects of the application such as the view and logic, allowing for higher levels of modifiability and scalability. This is an important factor with regards to this application, as the client might want to make modifications to certain parts of the application after it is handed over.

2.6.4 Hibernate Framework

Hibernate ORM is a Java library for mapping an object oriented domain model to a traditional database. The Hibernate Framework was chosen for this project due to the following reasons:

- Only has to be configured once and it can be used forever
- Maps models and tables easily
- Executes SQL in an easy way
- Manages transactions with one annotation
- Can change one database to another in one second.

2.6.5 KSoap2

KSoap2 is a lightweight and easy to use SOAP client library developed for the Android platform. [8] To connect to the SOAP server the KSoap2 library is needed. One drawback of using KSoap2 library is that it does not support GZIP(see 2.10) when sending a request to the server. To solve this, the source code of KSoap2 library is modified by integrating the GZIP encoding.

2.7 Web container

Apache Tomcat is a fast and stable open-source web server and servlet container. It provides a pure Java web server environment for running Java code [3]. It can be configured directly by editing XML configuration files. This ensures ease of configuration during application development. It is also a cross-platform software, which allows it to run on different operating systems. This was necessary for this project as team members were running different OS's such as Mac,Linux, and Windows.

2.8 Databases

2.8.1 MySQL

MySQL is an open source relational database management system(RDBMS), which is widely used for web applications.[9] MySQL was the database used during the initial iterations of the application, and was used mostly for testing the functions of the system. It was later replaced by H2 for reasons stated in the next section.

2.8.2 H2

H2 is another open source RDBMS. It is more similar to SQLite than MySQL in terms of the way it operates. It can be implemented using the standard server/client side model, however it can also use a single file server.[7] The backend was altered to use H2 for the final implementation as it has better compatibility with SQLite, and since it can be implemented as a serverless database, it doesn't require a machine to constantly run the database.

2.8.3 SQLite

SQLite is an alternative open source RDBMS. In contrast to a database management system like MySQL, it is not implemented as a separate process that a client program running in another process accesses.[16] Rather, it is part of the using program. This makes SQLite, as the name implies, lightweight and easy to work with. Since SQLite is embedded into every android API level, it makes it easy to implement in an android application. It does not require a setup procedure or any form of administration of the database.

2.9 Encoding

2.9.1 GZIP

GZIP is a software application that can be used for compressing and decompressing files.[6] GZIP is widely used in HTTP compression and most web servers and web clients support GZIP. Particularly, in this project all the framework above has GZIP support. Though kSOAP2 library does not support

compressing request by GZIP, it supports decompressing responses from a server. After modifying the source code of kSOAP2 library, the problem got solved perfectly. Therefore GZIP is selected in this project.

2.9.2 Base64

Base64 encoding schemes are commonly used when there is a need to encode binary data that needs to be stored and transferred over media that is designed to deal with textual data.[4] Here Base64 is used for convert the images to a normal string so that the SOAP or REST service could contain the information in the XML format or JSON format, and then be compressed via GZIP encoding. Therefore the application would use less battery power according to the DIL-environment (Disconnected, Intermittent and Limited environment).

2.10 Map

2.10.1 OpenStreetMap

OpenStreetMap is an open source collaborative project to create a free and editable map of the world, and it is inspired from sites like Wikipedia. [10] The maps are created from data collected from portable GPS devices, aerial photography, and most importantly all kind of sources and local knowledge provided by users. OSM was created because other map services, etc. google maps, have legal and/or technical restrictions on their use, restricting users from utilizing all the features in a manner they see fit.

This can have both positive and negative sides. Since everyone can edit the information, there is possibility of wrong information appearing, but also it can be updated continuously, and mistakes can be fixed whenever they are identified. In our experience, the level of details on the maps were as good, if not better than some of the google maps, but it lacks the satellite and hybrid views. Luckily only the standard map was planned for our app, so this was not a problem for us.

2.10.2 Osmdroid

The OSM api used for developing android apps is called OSMdroid , and it is a replacement for the Google maps MapView class.[11] This means that it is very similar to the google maps API, which makes it quite easy to implement as there is a lot of available information and tutorials available online. It has a lot of easy to implement features already built in, and there is no further restrictions on what you can extend it with.

Chapter 3

Project Management

This chapter will present the planning and organization part of the project. The goal of this chapter is to show the steps that were taken in order to deliver a functional application as well as quality documentation. The chosen methodology will be introduced, followed by time management and risk evaluations.

3.1 SCRUM

There are several uncertain elements that the group will have to tackle during the project: The customer might change their minds during the course of development, the team is new to Android development and therefore needs to learn it. Also the project development time is short so time management is crucial. Due to these implications it was clear an iterative, incremental and agile development framework was needed to ensure it was easy to make the necessary adjustments as quickly as possible along the way.

3.1.1 Introduction

It was decided early on that Scrum would be the management tool used throughout the development of the application. The team was already familiar with Scrum from previous projects, and the iterative nature of the project fit the Scrum framework. The team briefly discussed some advantages of other development frameworks, like the waterfall model. However this was rejected due to its rigid structure that does not lend itself well to changes in client specifications.

Scrum is an iterative and incremental development framework that allows for substantial flexibility during the course of the development cycle. Constant communication with the customer is a key aspects of Scrum. Since the workload is iterative and incremental it is easy to adjust to the feedback of the client, and make the necessary changes along the way. Sprints, as the iterations are called in Scrum, are usually set to a reasonable timeframe based on the time estimations made early on in the planning phase. The allotted time for each iteration combined with frequent communication with the client, can help ensure that the direction of the product will not deviate too far from the cus-

tomers expectations. On this basis meetings were arranged with the client once per week. During these meetings the team would inform the client of the current progress and show iterations of the system whenever applicable.The team would then receive feedback on what parts of the application was in accordance with their expectations, and where improvements might be necessary. Detailed accounts of the client meetings can be found in Appendix B

3.1.2 Responsibilities

Each of the team members had specific areas they were supervising during the project. This was done to ensure that all areas of the project would be given sufficient care and effort.

Database: [CHUN FAN]
As a result of his previous experience with a variety of database technologies and database framework Chun Fan was assigned the task of supervising the database part of the product

Server side: [CHUN FAN & TORGRIM BØE SKÅRSMOEN]
Because of his knowledge concerning server development, server frameworks and core Java features Chun Fan was also chosen to be one of the supervisors of the back-end together with Torgrim who also had previous knowledge of core Java features and showed an interest in being part of building the server side technology.

Client side: [JUNJUN GUO & SIMEN TEIGEN SØGÅRD]
Junjun had some previous knowledge with Android development and was very comfortable with Java, so he was chosen together with Simen to supervise the Client side of the project.

Graphical User Interface: [LIANG ZHU]
No team member had any real experience with GUI design coming into this project, therefore there was no clear choice on who would supervise this area. In the end Liang volunteered to supervise the GUI design concerning this project.

Report: [MORTEN OPDAHL]
As with GUI there was no clear choice of who would supervise the documentation part of the project, but luckily Morten volunteered for this task.

3.1.3 Group Meetings

Group meetings were scheduled for three times per week. Every meeting would start with a Scrum meeting. During the Scrum meeting each team member would take turns to state the progress made on a specific task since the last meeting, and what would be the agenda until the next meeting. After the Scrum meeting the team would proceed to work on the designated tasks. This allowed everyone to maintain a clear overview of the tasks being worked on and the current progress. By following this format, confusion that can arise from bad communication, could be avoided and solved at an early stage.

3.1.4 Product Backlog

The product backlog is a list of requirements for the application. It contains features, bugfixes and other requirements. The content of the backlog is the basis of what needs to be done in order to deliver a satisfying product. The items are usually ordered in considerations of importance, risk, dependencies and so on. The full product backlog for our project is provided in chapter 4.3.4.

3.1.5 Sprint Backlog

The sprint backlog shows which tasks that should be completed during a specific time period. Due to the timeframe of the project, the team settled on 2 week long sprints. The team felt that this was a reasonable estimate to ensure that steady progress was made, without over committing to the workload and trying to attain unrealistic goals. The sprint backlogs can be found in Appendix C.

3.1.6 Sprint Burndown chart

A burndown chart is often used together with Scrum. This chart shows the relationship between work effort estimated and time. There are different types of burndown concerning the period they are depicting and the unit they use for measurement. The group chose to use a sprint burndown chart that represents the period between each sprint which in this project was set to be two weeks. We used the burndown chart during the sprints to see how our progress was going, compared to what we had estimated at the beginning of the sprint. Then we could see if it was necessary to do some extra work the rest of the sprint, or if we would have to postpone it to the next. Also if we were Far ahead of the estimates, we might want to add additional tasks, or accelerate the end of the sprint. Also at the beginning of the next sprint, we could then use the last sprint's burndown chart to easier estimate the workload. One example will be given here, the remaining charts, with some more detail on what the individual task were, can be found in appendix D.

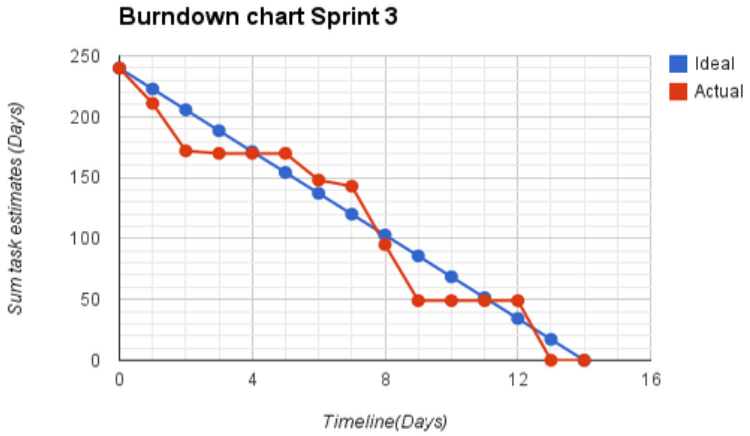


Figure 3.1: *burndown chart sprint 3*

Figure 3.1: Burndown chart sprint 3

Since the team consist of six members and it is expected that each member work approximately 20 hours a week, the total estimated effort to be spent during this sprint is 240 hours over the course of 14 days.

In figure 3.1 there is an example of the burndown chart from Sprint 3. The red line shows the actual effort that the team spent on finishing tasks compared to the blue line which shows our estimated time needed. During this sprint there was a few tasks that too a bit longer than expected, and some tasks that were completed faster than planned, but no larger deviations. This meant that by the end of the Sprint, all tasks were completed 1 day earlier than estimated.

3.2 Time and workload estimation

Below is the group's plans on how the time and workload throughout the different phases of the project would be distributed. We estimated that each group member would work 20 hours a week in average throughout the project, except during holiday. In the beginning of the project, the group were a bit too eager on starting the development, and therefore did not plan this part very well. After the first meeting with the supervisor, the group understood that this was a very important part, and that it would help later in the project by easier knowing what resources were available, and what might be needed to finish the various components. This way, the group would not end up spending disproportionally amounts of resources working on one package, while not leaving enough for the others. The work breakdown structure (WBS) is arranged as a hierarchy where each package is made up of several smaller packages underneath it.

3.2.1 Work Breakdown Structure

The team's WBS went through many iterations before it was of any real use. These iterations often happened after getting feedback from the supervisor. The most important changes were the separation of report writing and the rest, and the estimated allocation of time resources. This gave the group a clearer overview of the tasks that had to be done and if the team was spending too much time on one part of the project.

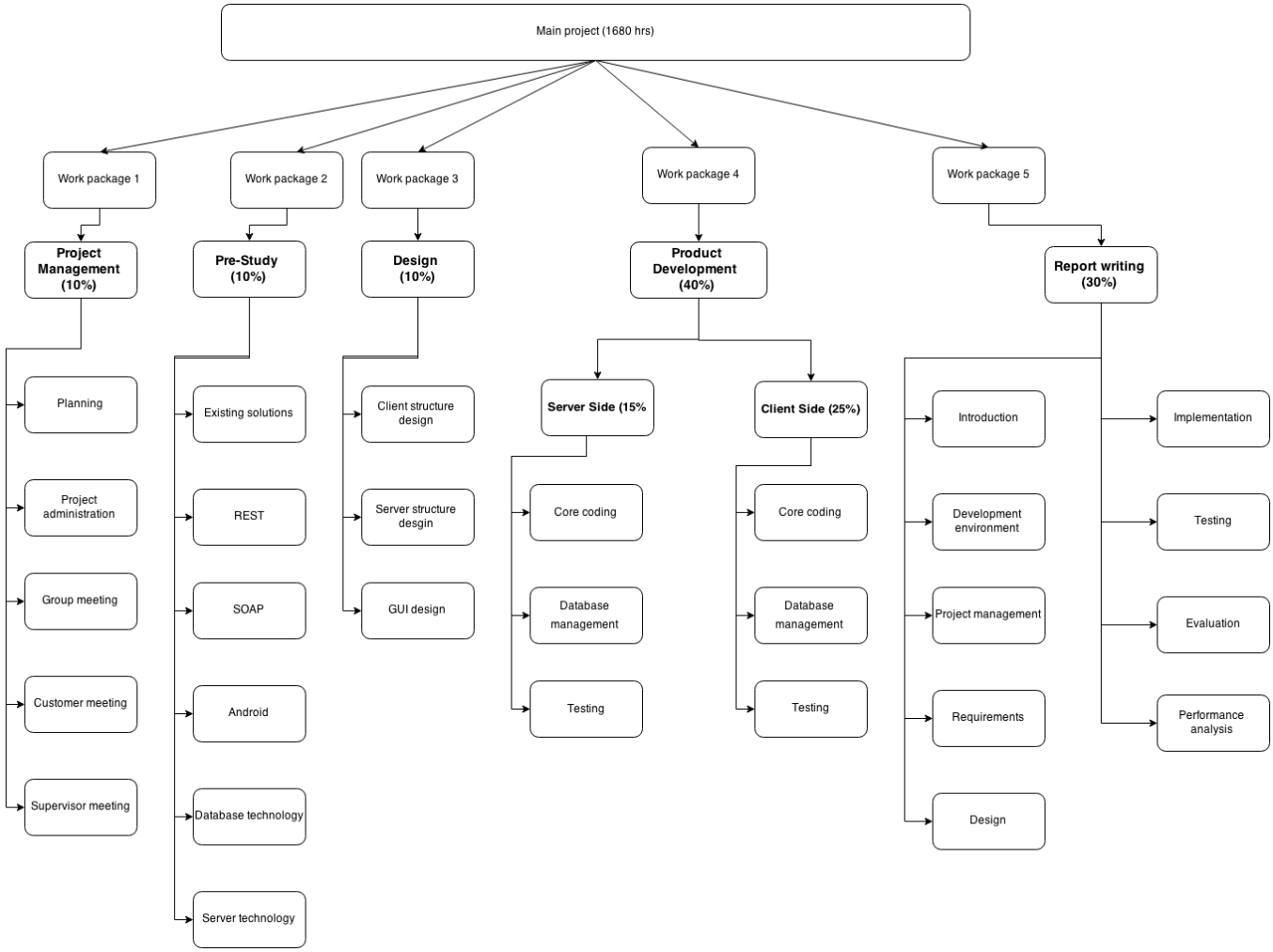


Figure 3.2: Work Breakdown Structure

3.2.2 WBS Pie Chart

Below is a pie chart showing the sizes of each of the main work packages outlines in the WBS

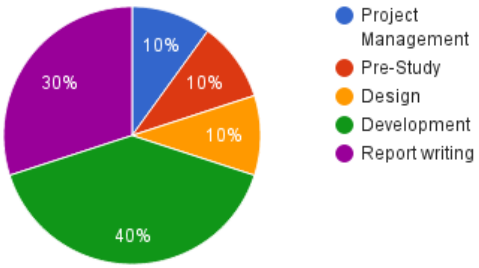
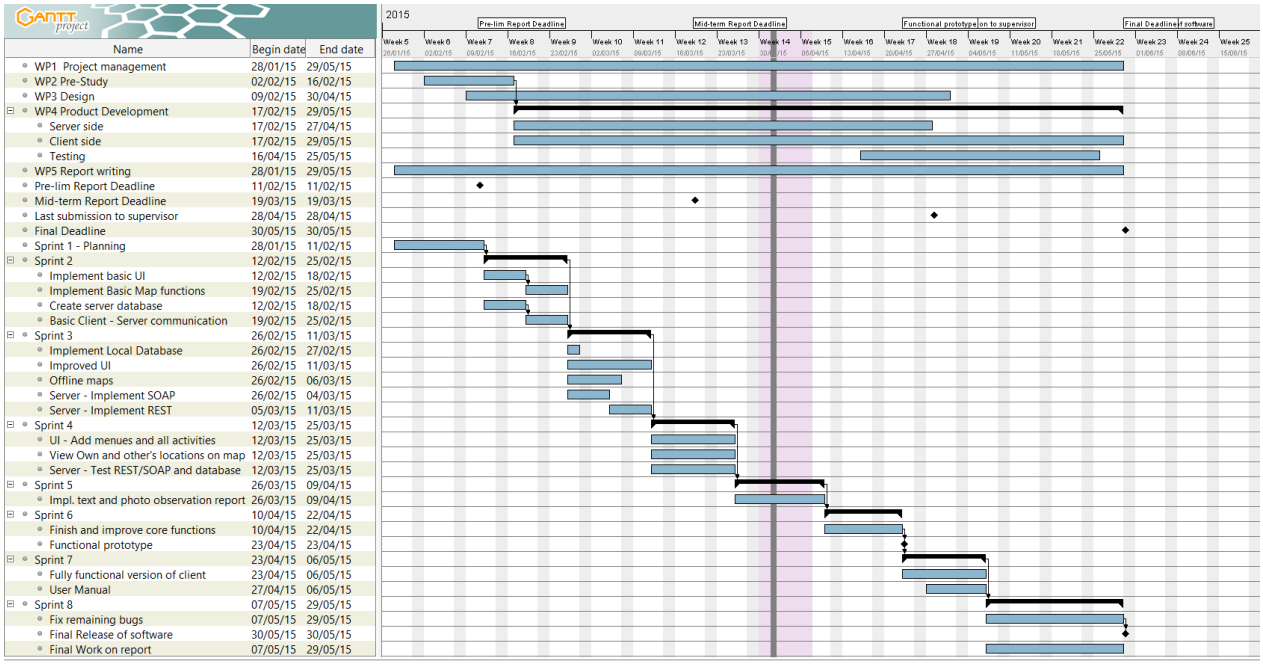


Figure 3.3: WBS pie chart

3.2.3 Gantt Timeline

The Gantt timeline below has worked as an outline of the development of the project. On top it has an outline of the main work packages and their dependencies of each other. Below it has been broken down into our sprints, and shows what has been the main focuses of each of the sprints, and what parts were needed to be completed before it was possible to start on the next parts. Having such a timeline made it easier to track our progress, and see where we might have to put in some more resources in order to not delay future work packages.



Name	Begin date	End date
WP1 Project management	28/01/15	29/05/15
WP2 Pre-Study	02/02/15	16/02/15
WP3 Design	09/02/15	30/04/15
WP4 Product Development	17/02/15	29/05/15
Server side	17/02/15	27/04/15
Client side	17/02/15	29/05/15
Testing	16/04/15	25/05/15
WP5 Report writing	28/01/15	29/05/15
Pre-lim Report Deadline	11/02/15	11/02/15
Mid-term Report Deadline	19/03/15	19/03/15
Last submission to supervisor	28/04/15	28/04/15
Final Deadline	30/05/15	30/05/15
Sprint 1 - Planning	28/01/15	11/02/15
Sprint 2	12/02/15	25/02/15
Implement basic UI	12/02/15	18/02/15
Implement Basic Map functions	19/02/15	25/02/15
Create server database	12/02/15	18/02/15
Basic Client - Server communication	19/02/15	25/02/15
Sprint 3	26/02/15	11/03/15
Implement Local Database	26/02/15	27/02/15
Improved UI	26/02/15	11/03/15
Offline maps	26/02/15	06/03/15
Server - Implement SOAP	26/02/15	04/03/15
Server - Implement REST	05/03/15	11/03/15
Sprint 4	12/03/15	25/03/15
UI - Add menues and all activities	12/03/15	25/03/15
View Own and other's locations on map	12/03/15	25/03/15
Server - Test REST/SOAP and database	12/03/15	25/03/15
Sprint 5	26/03/15	09/04/15
Impl. text and photo observation report	26/03/15	09/04/15
Sprint 6	10/04/15	22/04/15
Finish and improve core functions	10/04/15	22/04/15
Functional prototype	23/04/15	23/04/15
Sprint 7	23/04/15	06/05/15
Fully functional version of client	23/04/15	06/05/15
User Manual	27/04/15	06/05/15
Sprint 8	07/05/15	29/05/15
Fix remaining bugs	07/05/15	29/05/15
Final Release of software	30/05/15	30/05/15
Final Work on report	07/05/15	29/05/15

Figure 3.4: Gantt Diagram

3.3 Risk Evaluation

3.3.1 Description

Risk evaluation is an important aspect of the planning phase. It helps identify potential problems and mitigate uncertainties throughout the development cycle. By identifying these potential risks early on in the planning phase, the team will already have remedial actions in place to ensure that the project can still be finished on time.

The risk evaluation table(Figure used follows a fairly standard template. It calculates the importance of a possible issue by multiplying the likelihood of an issue occurring, and the impact it would have on the project if the issue occurs. A high importance value implies the issue should garner special attention. If precautions are not taken, possible delays in the project can be inevitable.

3.3.2 Risk Evaluation Table

Risk ID	Description	Likelihood	Impact	Importance	Preventive Action	Remedial Action
01	Short Term Illness(Two weeks or more)	7	1	7	Work an extra 30 min every day	split missed work hours between other team members
02	Long Term Illness	1	8	8	Inconclusive	Split missed work hours between other team members
03	Unable to meet deadline	4	9	36	Proper planning and good communication within the group by having frequent SCRUM meetings	None
04	Group member abandons project	1	9	9	Good communication so it is possible to prepare for such an event	Work extra hours due to missing team member
05	Group member does not finish an assigned task that is to be completed within a certain time period	8	3	24	Good communication between the person overseeing that part of the project and the team member doing the specific task	Move the task to the next sprint and have other team members provide assistance
06	Hardware Failure	3	4	12	Keep frequent backups	Use an alternative device
07	Faulty Software	2	7	14	Use supported software that is frequently updated	Use alternative software with similar functionality
08	Network failure	1	1	1	Work in a place with stable connection	Go somewhere that has a stable connection
09	Changes to requirements	4	7	28	Frequent meetings, clean and well constructed code that is easy to modify	Adapt the software to meet the new requirements
10	Complicated software	5	2	10	Get familiar with the software before deciding on using it	Read more
11	Conflicts within the group	3	6	18	Good communication and planning, having frequent SCRUM meetings so everyone knows what to do and the progress of the project	Talk it out
12	Software does not meet the requirements of the customer	2	9	18	Frequent meeting with the client to discuss the requirements	Work extra until it satisfy the customer

Table 3.1: Risk Evaluation Table

Chapter 4

Requirements

This chapter presents the requirements analysis portion of the project. The requirements are divided into two main parts, functional and nonfunctional requirements.

The functional requirements specify the functions that a system or a specific method must be able to perform.Use cases can be utilized as a tool for communicating the team's interpretation or understanding of the requirements to the customer in a clear manner.The use cases can be viewed in section 4.3. In the later stages it can also aid the developers in setting up functional test cases.

The nonfunctional requirements are more of a statement of how the system must behave. In other words, it is a restraint upon the system's behaviour. The nonfunctional requirements can be used to judge the operation of a system such as performance,security and usability etc.

Both the functional and nonfunctional requirements were determined based on information from the client. After the first few meetings the requirements that had the highest priority according to their specifications were used as part of the requirements analysis.

A requirement that bears specific mention is that the application must work in a DIL-environment (Disconnected, Intermittent and Limited environment). The concept of a DIL environment will be explained in section 4.1.

4.1 DIL

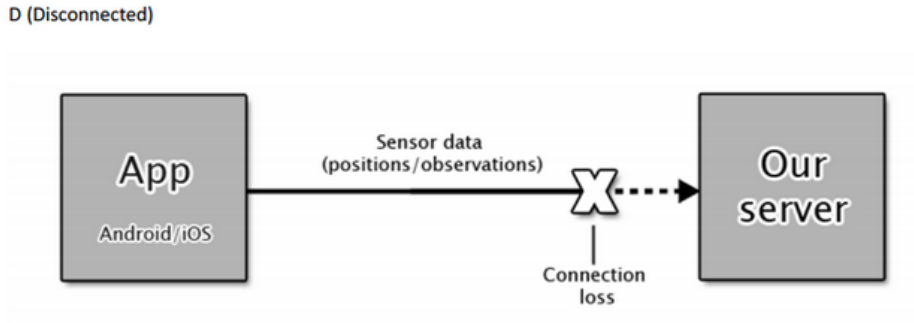


Figure 4.1: Disconnected[1].

The app will be required to run in an environment that might be disconnected. If the app loses connection to the server over an extended period of time, it should still be able to run with the features that do not require a connection. If the app attempts to send data while disconnected, it should be able to identify the problem and try again once it regains network connection.

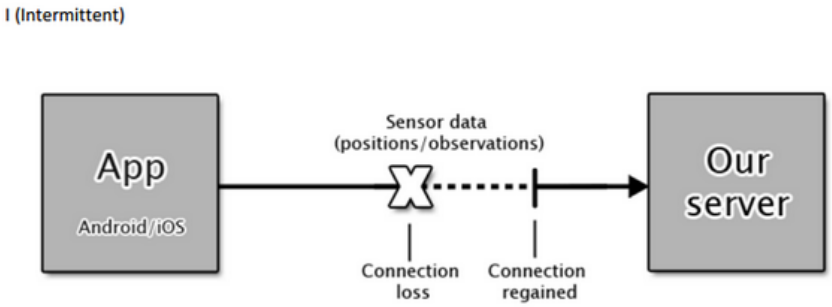


Figure 4.2: Intermittent[2].

It is also a possibility that the app will lose network connection, but then be able to regain it shortly afterwards. An environment where this may be frequent is called intermittent. It is important to implement solutions that ensure no data is lost when trying to communicate with the server. If the application tries to send data to the server but then gets interrupted, it may not be clear to the user that it even lost connection. To make sure that the sent data is actually reaching its target, the app will need to get confirmation from the server that it has successfully received the data in form of an acknowledge . If it does not receive this confirmation, it will have to try again later until it finally goes through.

Since the application is designed to be used during field operations, it might be used far away from current infrastructure providing network connections. Therefore it might have limited bandwidth.

[1] Figure provided by customer
[2] Figure provided by customer

The rates it might have to operate under the different networks is listed in figure 4.3.

Network	Data Rate	Delay	PER	Type of Network
Satellite Communications (SATCOM)	250 kbps	550 ms	0	Transit
Line of Sight (LOS)	2 Mbps	5 ms	0	Transit
Wireless Fidelity (WiFi) 1	2 Mbps	100 ms	1 %	Tactical edge
WiFi 2	2 Mbps	100 ms	20 %	Tactical edge
Combat Net Radio (CNR) with Forward Error Correction (FEC)	9.6 kbps	100 ms	1 %	Transit

Figure 4.3: Different network bandwidth specifications[3].

This implies that the application's core functions must still be working when the device does not have a stable network connection or the speed of the connection might be very slow. Therefore it is important that the app knows that all data has reached its destination successfully, and that the data is compressed without losing any valuable data. Another limitation is that the user might be operating under conditions where opportunities for charging the battery are scarce. This means that the battery usage of the application will have to be taken carefully into consideration.

4.2 Functional Requirements

The team created use cases early on in order to help the customer understand the team's interpretation of all the functional requirements. After the initial use cases and functional requirements were set up, it was sent to the customer for feedback. On the next skype meeting with the customer, the group received feedback on the use cases. The customer was mostly satisfied, but had some minor changes to some of the uses cases, and also wanted a few new ones made regarding updating information, and logout function. The use cases were also used as a base for creating the functional tests in chapter 7.2

[3] Figure provided by customer

4.2.1 Use case diagram

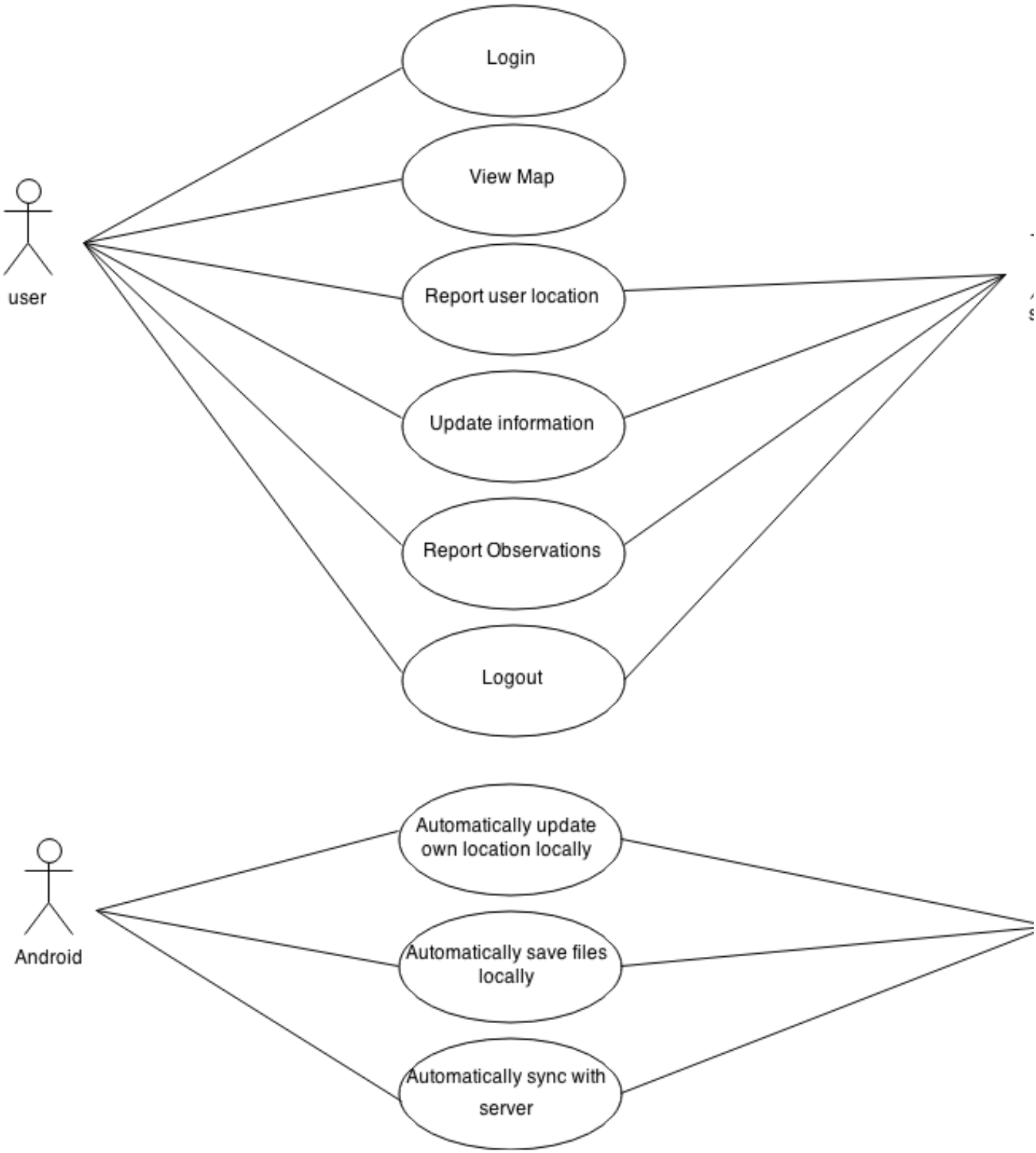


Figure 4.4: Use cases

4.2.2 Textual use case

4.2.2.1 Login

Title	Use Case 1: Login
Scope	Main Application
Primary Actor	User, System
Stakeholders and interests	User: Wants to login to the app System: Should authenticate login attempt
Precondition	Phone or tablet with application installed and enough power. Network connection
Success guarantee	Application starts, logs in with an identified use and enters main screen
Main Success Scenario	1) User Starts Application 2) User type their Username and Password and click login 3) System returns a success message 4) System connects and enters main screen
Alternative Flow	1.1) User has already logged in and authorized his device to the server earlier, and therefore does not need to log in again or have a working connection to start the main activity 2.1) Wrong Username/Password 1. User is asked to review their input 3.1) System can't connect and has not been authorized previously 1. User is asked to connect to a network

Table 4.1: Use Case 1: Login

4.2.2.2 View map

Title	Use Case 2: View Map
Scope	Main Application
Primary Actor	User
Stakeholders and interests	User wants to be able to view and interact with objects on the map including the position of other users and points of interest added by others or by the user himself/herself
Preconditions	1: Login
Success guarantee	User can see and interact with all the relevant positions on the map
Main Success Scenario	A: 1)User clicks on one of the objects on the map 2) Opens an alternative window to display the requested information B: 1) User drags the map around, presses the zoom buttons or uses multi touch to zoom in or out 2) The map responds accordingly
Alternative Flow	A.2.1) No available information at the selected location 1. Returns the coordinates and user id of the selected point B.1.2) Already fully zoomed 1. Zoom button is greyed out

Table 4.2: Use Case 2: View Map

4.2.2.3 Update Information

Title	Use Case 4: Update Information
Scope	Main Application
Primary Actor	User, System
Stakeholders and interests	User: Wants to send current position server. Update positions of other teams and points of interest so that he can view the new information on his screen System: Should update the map automatically at specified intervals
Preconditions	User Is logged in. Application running on the main screen, and connected to the Cloud System
Success guarantee	User position, positions of the other teams, and points of interest gets updated and plotted on the map
Main Success Scenario	1) User clicks the refresh button 2) System retrieves information from the Cloud System and then updates the map so that the information is the newest available
Alternative Flow	2.1) System was not able to retrieve information from the Cloud System 1. User is asked to try again, or ensure he is connected

Table 4.3: Use Case 3: Update Information

4.2.2.4 Report Observations

Title	Use Case 5: Report Observations
Scope	Main Application
Primary Actor	User
Stakeholders and interests	User: Wants to send in information about a location providing text and/or images
Preconditions	User is logged in and is running the application on the main screen
Success guarantee	System sends text/and or images successfully to the Cloud System
Main Success Scenario	1) User clicks the send position button in main menu 2) User then gets taken to a new window(Send Screen) where he can add information 3) User types in the text he wants in the textbox 4) User clicks the Photo button which will take him/her to the android camera function where he/her takes a picture before returning to the Send screen 5) User clicks send 6) System then sends the position + the added information to the Cloud System
Alternative Flow	3.1) User does not want to input any text and can just skip this point 4.1) User does not want to attach an image and can just skip this point 4.2) User wants to use an image already taken on his phone he can then go to the gallery and choose a photo from there 6.1) System does not have a connection to the Cloud System 1. System saves the data and tries to resend it next time it auto updates the user's position

Table 4.4: Use Case 4: Report Observation

4.2.2.5 View Observations

Title	Use Case 6: View Observations
Scope	Main Application
Primary Actor	User
Stakeholders and interests	User: Wants to view observations reported by other users
Preconditions	User is logged in and is running the application on the main screen
Success guarantee	User can see the coordinates, the content of an observation, and the user id associated with the observation
Main Success Scenario	1) User selects the report view tab from the drop-down menu and is presented with a list of observations 2) User clicks an observation and a pop-up displays the location, the content of the observation, and the id of the user that reported the observation
Alternative Flow	1.1) User clicks an observation on the map and a pop-up displays the location, the content of the observation, and the id of the user that reported the observation

Table 4.5: Use Case 5: View Observations

4.2.2.6 Logout

Title	Use Case 7: Logout
Scope	Main Application
Primary Actor	User
Stakeholders and interests	User: Wants to logout of the application
Preconditions	User is logged in and application is running
Success guarantee	The user logs out and is returned to the login screen
Main Success Scenario	1. User clicks the logout button 2. User is redirected to the login window and is no longer connected to the server
Alternative Flow	None

Table 4.6: Use Case 6: Logout

4.2.2.7 Automatically update own location

Title	Use Case 8: Automatically update own location
Scope	Main Application
Primary Actor	User, System
Stakeholders and interests	User: Wants to keep his location updated so he can see where he is at any time System: Wants to have an updated location stored locally to be able report to server or use in other functions such as to show the location on the map
Precondition	Logged in to the app and have either GPS or Network connection to be able to determine current position
Success guarantee	Application acquires the user's current position and updates the map to show the changes
Main Success Scenario	1) Application is up and running 2) Every 20 seconds or after moving a minimum of 5m the application will reacquire its current geographical position. Both the update time and the minimum movement distance is configurable through the settings screen. 3) Map updates it's state to show the newest location of the user
Alternative Flow	2.1) There is no GPS or Network connection. System will use latest known location until connection is reacquired

Table 4.7: Use Case 7: Automatically update own location

4.2.2.8 Automatically store files locally

Title	Use Case 9: Automatically store files locally
Scope	Main Application
Primary Actor	System
Stakeholders and interests	User: Wants be able to use the application and the last updates data if there is no network connection System: Wants to be able to use all its functions even if there is no connection to the server.
Precondition	Logged in to the app and have enough storage space on the device
Success guarantee	The system stores data in a local database where it can either be sent to the server during the auto-sync, or be accessed by the application to show the information to the user.
Main Success Scenario	1) Application is up and running 2) Every time something gets changed (User position updates, user adds an observation), it gets stored in the local database.
Alternative Flow	2.1) Not enough space to store data on the device. To solve this, the user will have to delete some other applications or data on the device.

Table 4.8: Use Case 8: Automatically store files locally

4.2.2.9 Automatically sync with server

Title	Use Case 10: Automatically sync with server
Scope	Main Application
Primary Actor	System, Server
Stakeholders and interests	User: Wants to keep information updated so he can see the most recent events and observations at any time System: Wants to have all the latest data Server: Wants to have the most recent information from the user, so that it can distribute this information on to all the other users as soon as possible
Precondition	Logged in to the app and have a Network connection with the server
Success guarantee	The system sends the data to the server which receives it, and updates it's database. Then the server sends all of it's new data to the system, which then again updates it's local database and shows the new information to the user through the different views
Main Success Scenario	1) Application is up and running 2) Every 1 minute (or whatever period of time the user configures though the settings page) The system sends its data to the server requesting a new update on other users locations and observations 3) System receives the new data, and updates it's views show them to the user
Alternative Flow	2.1) There is no Network connection. System will try again during the next scheduled auto-sync. (by default every 1 minute)

Table 4.9: Use Case 9: Automatically sync with server

4.3 Non-functional requirements

4.3.1 Usability

Usability is an important requirement of the application. The application could be used in stressful situations where information gathering has to be quick and efficient. For this reason the user interface must be intuitive and quick to respond so time is not wasted fumbling around the menus.

Another important aspect is that the application is quick to get accustomed to. If for some reason a user with no previous experience of the application has to learn how to use it during say, an operation, it should be simple to pick it up and get it functioning as quickly as possible.

A user interface test will be conducted with a mock up user interface. This is for the purpose of finding out what is intuitive for the user.

4.3.2 Security

Security is another key requirement of the application. The information stored in the application can be considered sensitive, especially if it would end up in the wrong hands. For this reason the login system was implemented to ensure only valid users can access the data. If the application has not been used after a certain amount of time, the user will be automatically logged out of the system and can not send requests to the server.

4.3.3 Performance

A key requirement of the application is to use as little power as possible. The Application may be used in an environment where the user is unable to charge the device regularly and will be forced to try to conserve energy. Therefore it is vital for the app to only use the minimal amount of power while still being able to perform all of its core functions.

As part of the non-functional requirements, our app should limit its impact on the battery life of its host device. There are several ways implemented in the application, and each of ways might affect the battery consumption differently. For example, the service is implemented in both SOAP and REST ways, and both ways also contains the GZIP compression and none GZIP compression. In order to select proper ways to improve the performance of battery consumption, the test should be conducted by using the app according to those different ways.

The tools used for testing the battery consumption is a third party application which is called PowerTutor. This app can calculate the consumption of the battery for every applications running in the Android system. It can also record the battery consumption used by LED, CPU, Wifi and 3G. Therefore the tool is quite easy to use for the battery consumption testing.

4.3.4 Product Backlog

The following product backlog represents the team's understanding of the application requirements on a high level. They are in prioritized order of importance:

- View map over the area with the users own position marked
- View other user's positions on a map
- View observations from central command or other users on the map
- Be able to send a textual observation of the users current position to the server
- Be able to attach a photo from the phone's gallery or one that the users takes to the text observation.
- Be able to send in observations using other positions.
- Continuously update the map to view changes and additions.
- Have a login function to authenticate user, and stay logged in for a certain amount of time so that it can use the app without network connection.
- Be able to download/cache the map so that the desired map area is available in offline mode.
- Able to view a list of observations, and there see what observations has been successfully sent to the server.
- Able to individually change settings like how often one wish to send and receive automatic updates with server.
- The application should try to limit the consumption of the battery from the device
- If the device has been stolen/lost it should be possible to block the device from the server side, so that it will not be able to run the app.

Chapter 5

Design

An iterative design methodology was used for the design of the system. Based on the requirement analysis it was decided that a Client-Server architecture would be best suited for the system. The class and database diagrams were used as a starting point. During each sprint components were either added or refined to incorporate the evolving requirements and improve the performance and quality of the system.

Since the application has the properties of a client/server system, the system was separated into a back-end and front-end from the very beginning. On this basis, the system could implement a publish/subscribe pattern to further match the requirements from the customer.

Back-end:
The back-end consist of a web server, database, and different modules. The server handles the requests coming from the front-end and responds to the client. The modules will:

- process the incoming or outgoing data
- validate the data
- communicate with the database to permanently save/delete existing data, or get information currently residing in the database.

Front-end:
The front-end represents the client side, which is the android application. Users do not need to know anything about the server (back-end), all functionality and services are presented by the client(front-end). The client can run independently from the server when there is no network connections, and will automatically communicate with the server to report its observations and locations. It can also download observations from other users. The client also contains its own database to save all the information. By communicating with the server, the client can present situations to the users.

5.1 Architectural and Design patterns

There are many benefits of using Architectural and design patterns:

- they provide ways to solve problems with proven solutions
- they make the overall system easier to understand and maintain
- and they make design, implementation and team communication more efficient.

Client-Server Architecture pattern

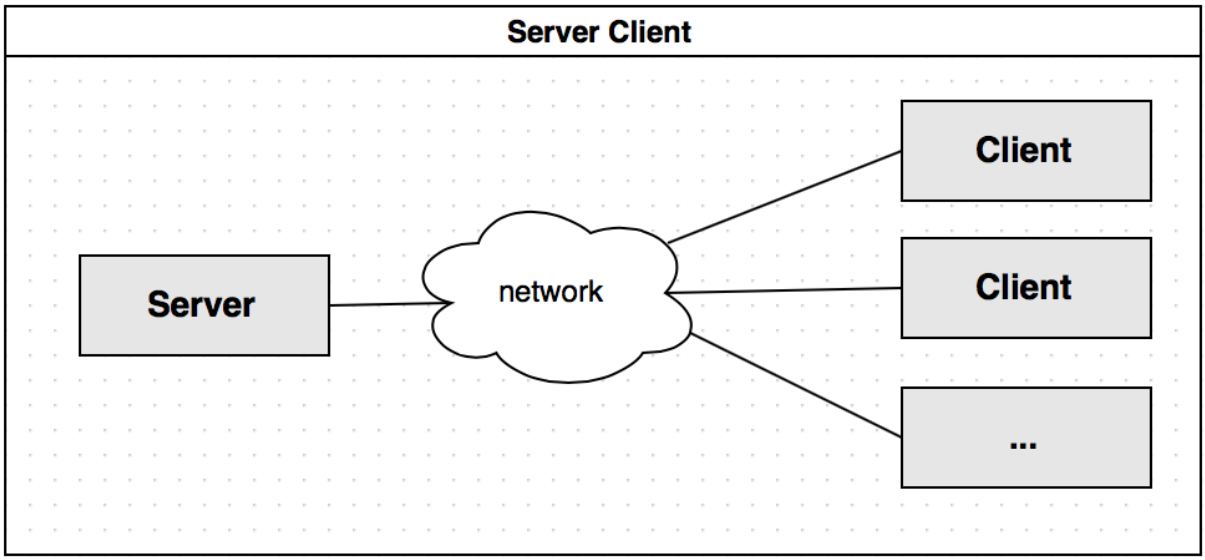


Figure 5.1: Client/Server draft

The Client and Server are two independent components, but they have to cooperate to make the whole application work. The client side is implemented to collect, hold information and do all the interactions (report locations, observations, show situations of co-workers) with users, and the requests sent to the server. The server handles the request and response from and to a client and validate the incoming data. By applying the Client Server Architecture pattern we are able to divide system into two models (components) and assign them to two groups so we can work parallel.

MVC Architecture pattern

The architecture incorporated on the client side is the MVC Architecture Pattern. MVC is the traditional Model View Controller pattern. The Model part holds the logical states of the application, the View part offers graphic interface to the user, and the Controller part does the control.

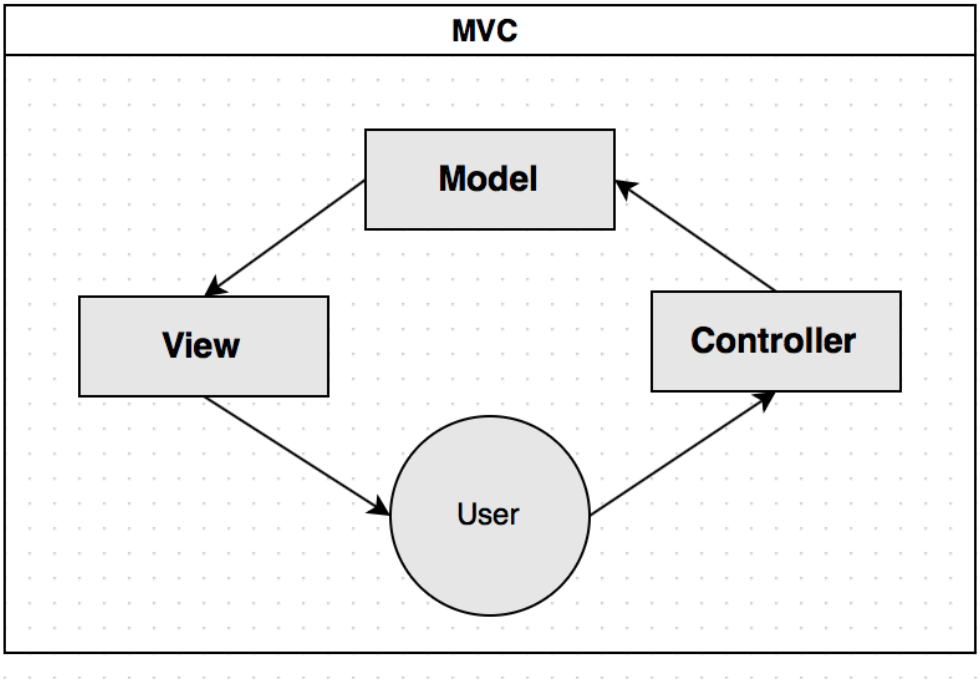


Figure 5.2: MVC draft

Model View Controller(MVC):

- All view files are in the layout folder
- Controllers hold all android activity classes that are used by the user
- Model is split into different parts
  - util: contains all utilities the service need
  - service: responsible for communication with the server
  - localdb: contains the local database, data access object
  - Data handling: handles data and synchronizes with the server in the background

Applying MVC gives the team some advantages:

- Easier to maintain, edit, change and update functionality
- Easier to make a plan for implementation and dividing the work between the team members

Design patterns Design patterns used in the system:

- **Observer pattern**
  - Used to observe or listen to a function, or an activity to trigger other functions

- **Singleton pattern**
  - Used to ensure that the program does not create a new instance when it is not needed
  - For instance: Auto sync object can only be created once and will then be used during the whole application's lifetime
- **Inheritance**
  - Used to reuse code, which makes it easier to update and maintain code
  - For instance: photo report, text report, location report and data access object for photo report, text report and location report. The common attributes are in super classes, so when editing or adding new attributes, we only need to do it in the super class if it is common for all classes.

5.2 ER Diagram

The entity–relationship model (ER model) is a data model for describing the data or information aspects of a business domain or its process requirements, in an abstract way that lends itself to ultimately being implemented in a database such as a relational database.

By designing the ER diagram at the beginning, all the team members could understand the general models of the application. After allocating the assignments, each member could design parts of the application individually based on the common models, so that the conflicts of the design would be reduced to a minimum.

The application has the the models as shown below; Member, Location, TextReport, PhotoReport and EventLog. The figure below shows the ER diagram.

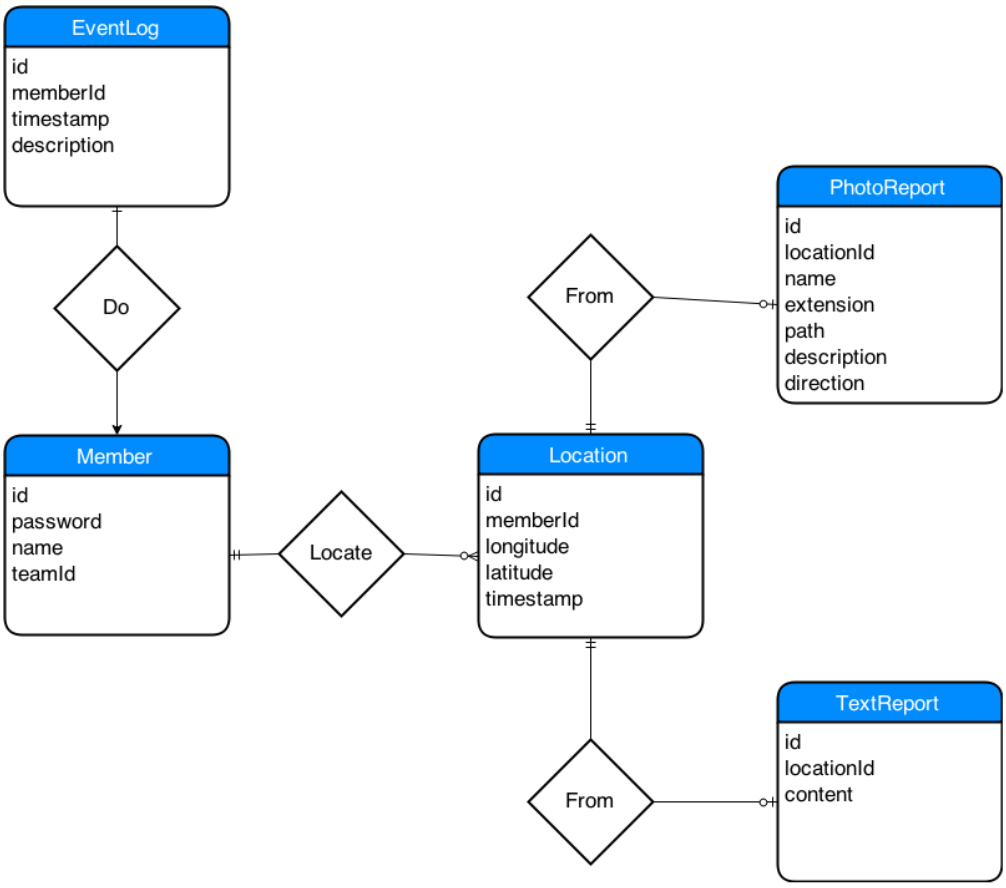


Figure 5.3: Entity Relationship diagram

Member: Member is the user of the application. It contains the basic information about the user.

Location: Location records the latitude, longitude and time stamp of a location. Each one of the locations must have only one owner which is the member, while one member could have zero, one or more locations.

TextReport: TextRport records the content and the location of a text report. Each one of the text reports must have only one location, while one location could have zero or one text report.

PhotoReport: As the same as TextRport, PhotoReport records the content and the location of a photo report. Each one of the photo reports must have only one location, while one location could have zero or one photo report.

EventLog: EventLog takes every action of a member into the log so that everything can be tracked.

5.3 Class Diagram

The Class Diagram was used to show the design structure of attributes, operations and methods we thought might be needed for the application. This helped for early design, discussion with team members, and it was also used as a starting point for the implementation. Class Diagrams were redesigned and reorganized during development, the final Class Diagrams are represented in chapter 6.

Server side Planned to have 3 packages, where each hold functions for service, Data Access Object and Model separately.

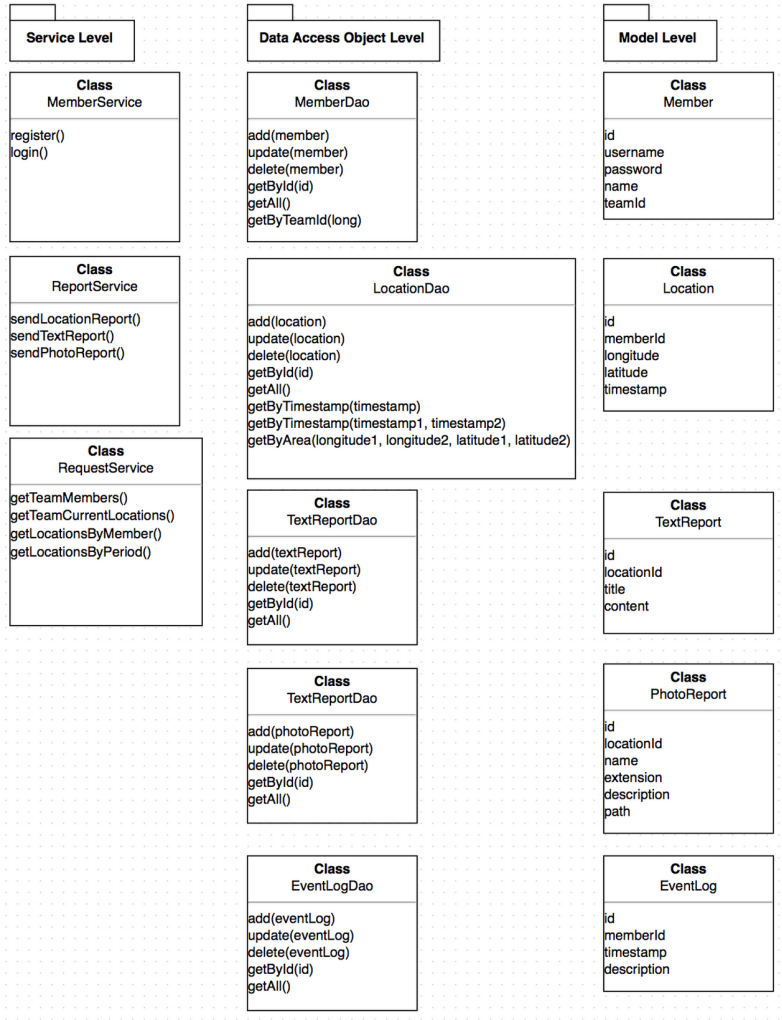


Figure 5.4: Server side main classes

Client side Client side also start with 3 packages, which hold functions for User Interactions, core functions, and server client communication functions.

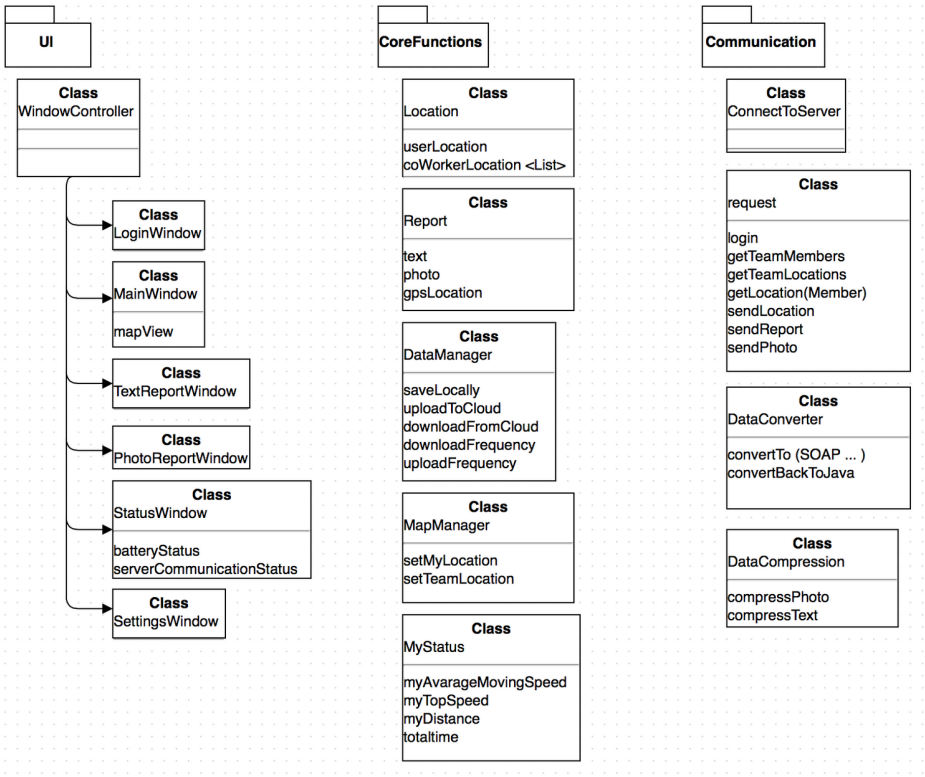
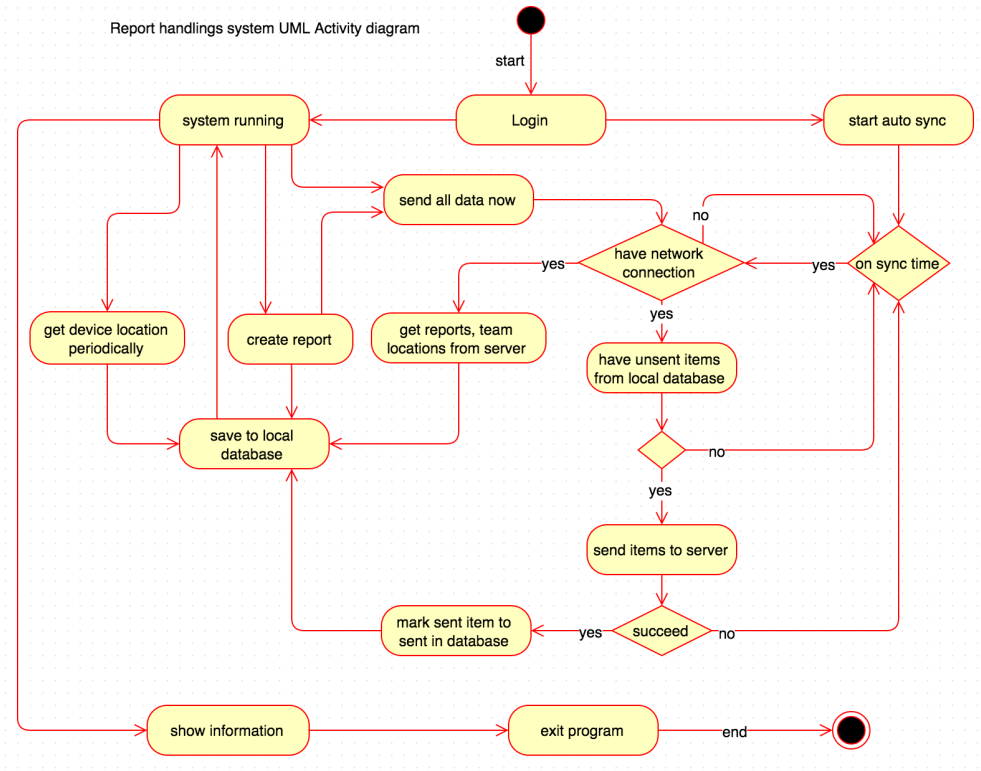


Figure 5.5: Client side main classes

5.4 Report handling system

The Report handling system needs to handle difficult situations. These include: limited network connection, slow connection speeds, and an intermittent network connection. For the purpose of handling difficult connection situations, all the outgoing and incoming information will be saved in a local database. The sending function will handle the outgoing information either by an auto sync to server function, or by allowing the user to click a send button to dispatch the information immediately. The outgoing messages (reports) which have been sent successfully will be marked as sent so that the system knows how to handle it. The user will be notified which message items have been successfully sent to the server. In order to save battery, the sending activity and get device location activity is limited by loop functions and will only run in specified intervals.



Description:

- When a user is logged in to the system is running and all functions are available. At the same time the "auto sync" function will start automatically in the background of the system which runs at specified time intervals that can be defined by the user
- The System will get the GPS location periodically and all the retrieved locations will be saved in the local database. The process is running automatically.
- When the "Send all data now" function is called, the system will first check if there is a network connection
  - if there is no connection, the mission will be passed to the “auto sync” function, which runs when it is “on sync time” or the “send all data now” is called again.
  - if a network connection is available, the system will further check if there are unsent items in local database:
    - if no unsent item at local database, the system will wait to next “on sync time”, or the “send all data now” is called again to run.
    - if there is unsent items, system will try to send unsent items to server
      - if the mission failed, the system will do it again on next “on sync time”, or the “send all data now” is called.

· if the mission succeeds,the system will locate the sent items in the local database and mark them as sent.

5.5 Login system

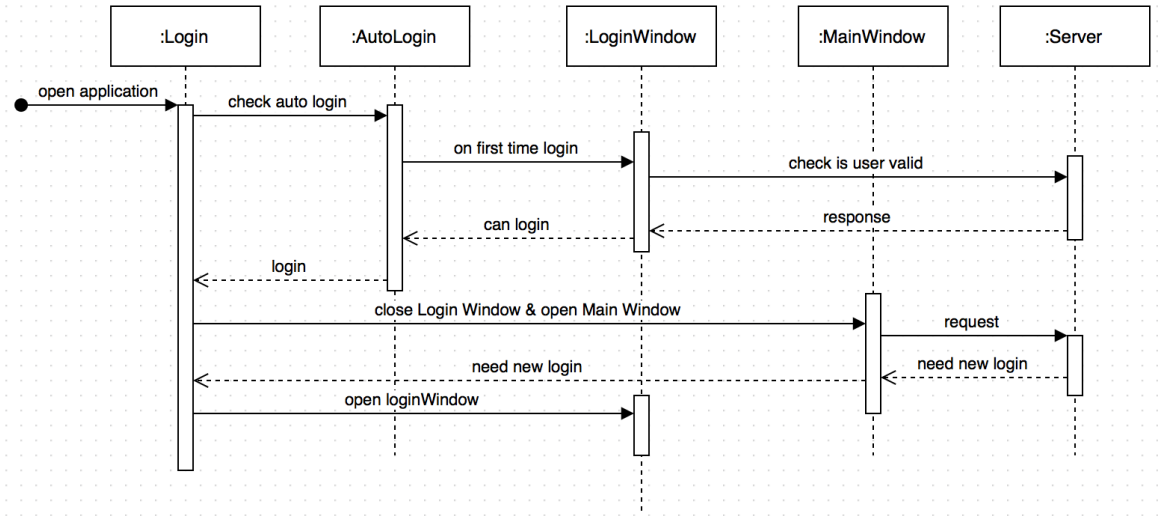


Figure 5.6: Sequence diagram for login

The first time the user logs in an id and password will have to be provided. The device also needs to be connected to a network for the user to be able to login and connect to the server.

- when login succeeds the user will be automatically logged in the next time, and no network connection is needed to open and use the application at this point.
- if logout is activated the user has to login again(need network to communicate with server for validation when logging in, but logout is remembered locally).
- if user has not used application for two days, a new login is needed(need network and validation from server). The application will jump to login window when it starts to communicate with the server again.

5.6 User Interface

The main view is based mainly on a map view. Here the user can see its own position on a map, as well as the position of all other users and the location of text or photo reports. The intention is that the user can then click on the individual markers on the map to get more detailed information about them.

There are also several other windows:

- A login window where the user can log in to the app.
- One for sending in photo and text reports
- A window where the user can view all his/her reports and easily see which ones has been successfully sent to the servers.
- One for seeing the status of different things, etc. Your own GPS coordinates, when you last sent an update to the server etc.
- A settings window where the users can change various settings

The login screen: Here is where the user can login with his user ID and password. The login button logs the user in and takes the user to the map screen (Main Menu).

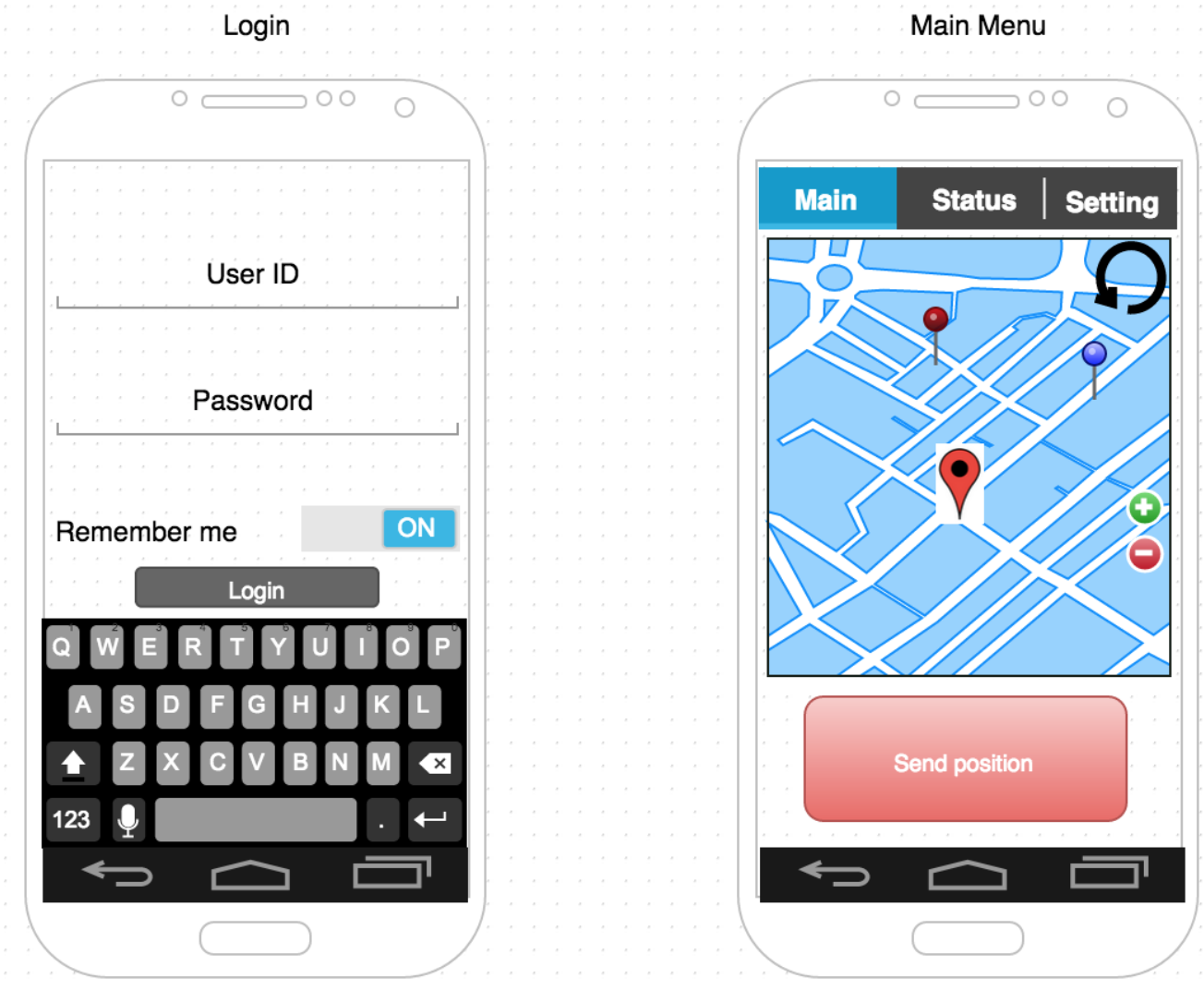


Figure 5.7: Login screen and map activity draft

The Main Menu screen: Shows user, team members, and observation locations on the map.It also has access to the other screens of the application via a menu interface.

The Report (Send) screen: The report screen gives the user an option to write a text description to a location. The new photo report button takes the user to the photo report screen. The save text report button will save and send the report.

Figure 5.8: Send report screen and List observations screen

The Observations screen: show observations as list. Both your own observations and observations from others.

The Settings screen: This screen gives the user a chance to change the options on the frequency on the updates that needs to be sent to the servers and default server if needed.

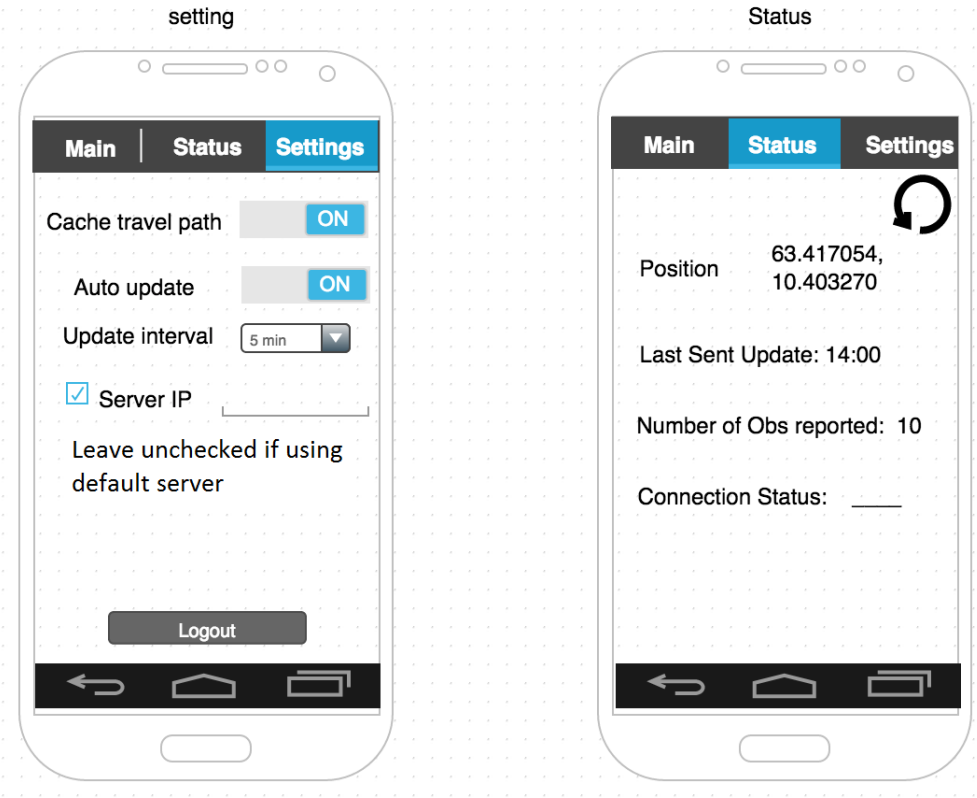


Figure 5.9: Settings screen and status screen

The status screen: This screen gives the users an overview of the status of the sync to the servers and the location of the user.

5.7 Security

1. Password lock screen: Android has a feature that allows the user to setup a password or a pin number to prevent unwanted access.
2. Encrypt phone: newer android versions has a encrypt phone feature. This allows the user to encrypt all saved data on the phone (including our app data). The encryption algorithm is 128 Advanced Encryption Standard (AES) with cipher-block chaining (CBC) and ESSIV:SHA256. The master key is encrypted with 128-bit AES via calls to the OpenSSL library.
3. Remote wipe: newer android versions has Android Device Manager which supports remote wipe of device with the associated google account.
4. Auto logout: If App has been inactive for more than 48 hours it will automatically logout.
5. Server block account: as a device has been reported missing we can remove the device from our database and prevent it from sending or receiving any more information.

Chapter 6

Implementation

Chapter 5 Design, described how the system was designed and why certain methodologies and patterns were chosen. This chapter documents how the system was actually implemented. Some parts, like the class diagrams, are different from the original design. Since the application consists of two separate components, the implementation is split into a server and a client section.

6.1 Server Implementation

Implementation on server side was conducted by several parts. Each part is in a certain layer which implements a set of similar functions. Here in the application it mainly includes the model layer, data access object layer, service layer and controller layer. Model layer contains all the models needed in the application. DAO(data access object) layer contains the classes which implement the data connection to the database. Service implements all the workflow functions. Controller handles the communication between the server side and the client side.

6.1.1 Model layer

The Model layer contains all the models needed according to the ER diagram. These models are Member, Location, TextReport, PhotoReport and EventLog. One extra model Result, is used for sending responses to client side.

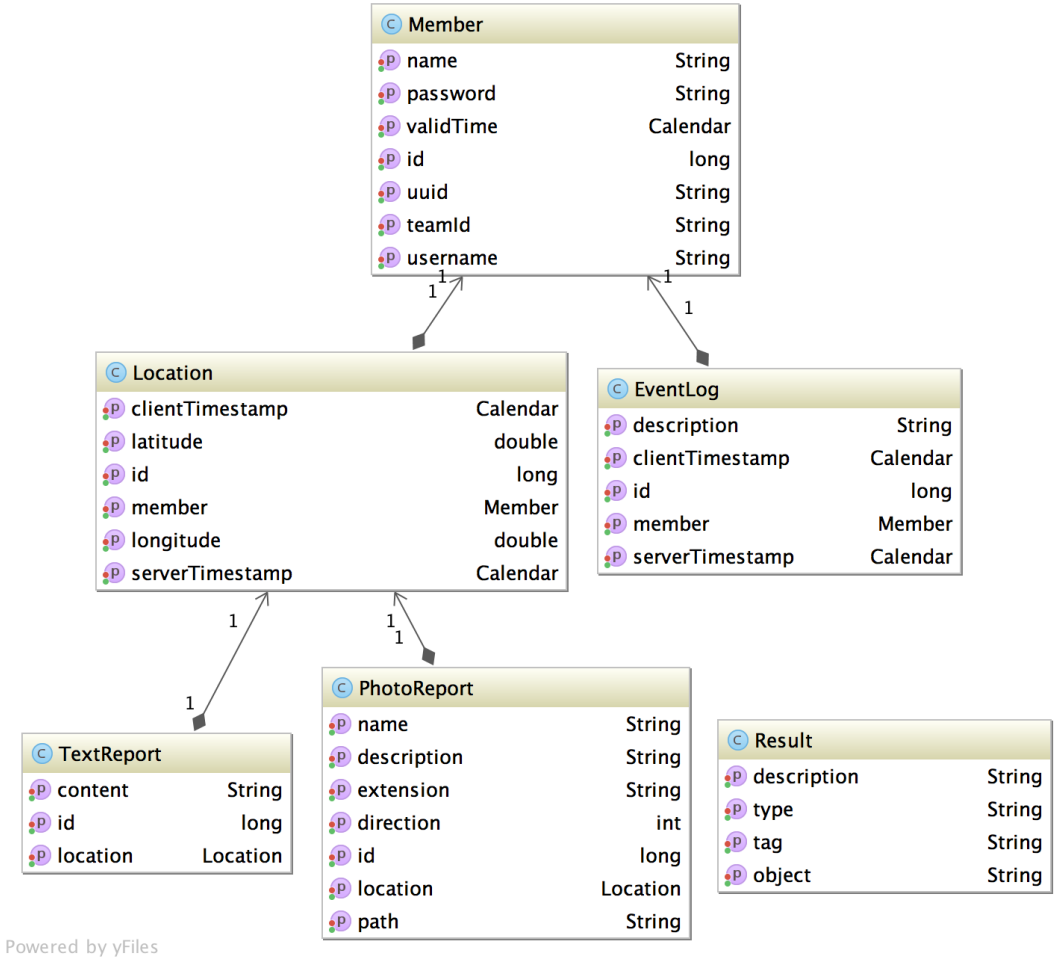


Figure 6.1: Model layer

6.1.2 Data Access Object layer

The DAO layer contains the classes connecting with the database. In this layer the interface is defined at first so that a different implementation could be done for the same interface. Currently the application implements only one implementation for each interface.

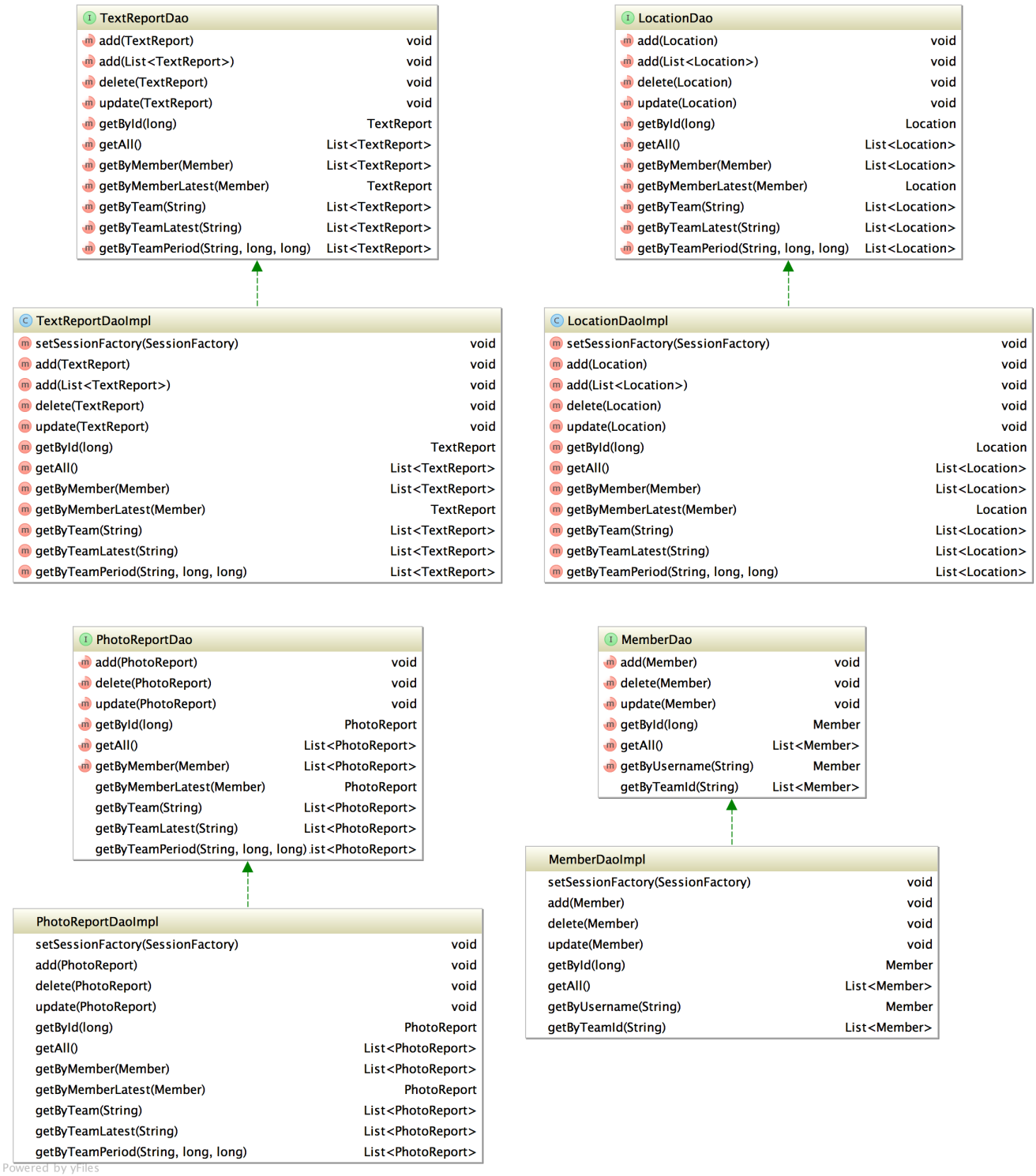


Figure 6.2: DAO layer

6.1.3 Service layer

This layer implements the services: UserService, ReportService, RequestService.

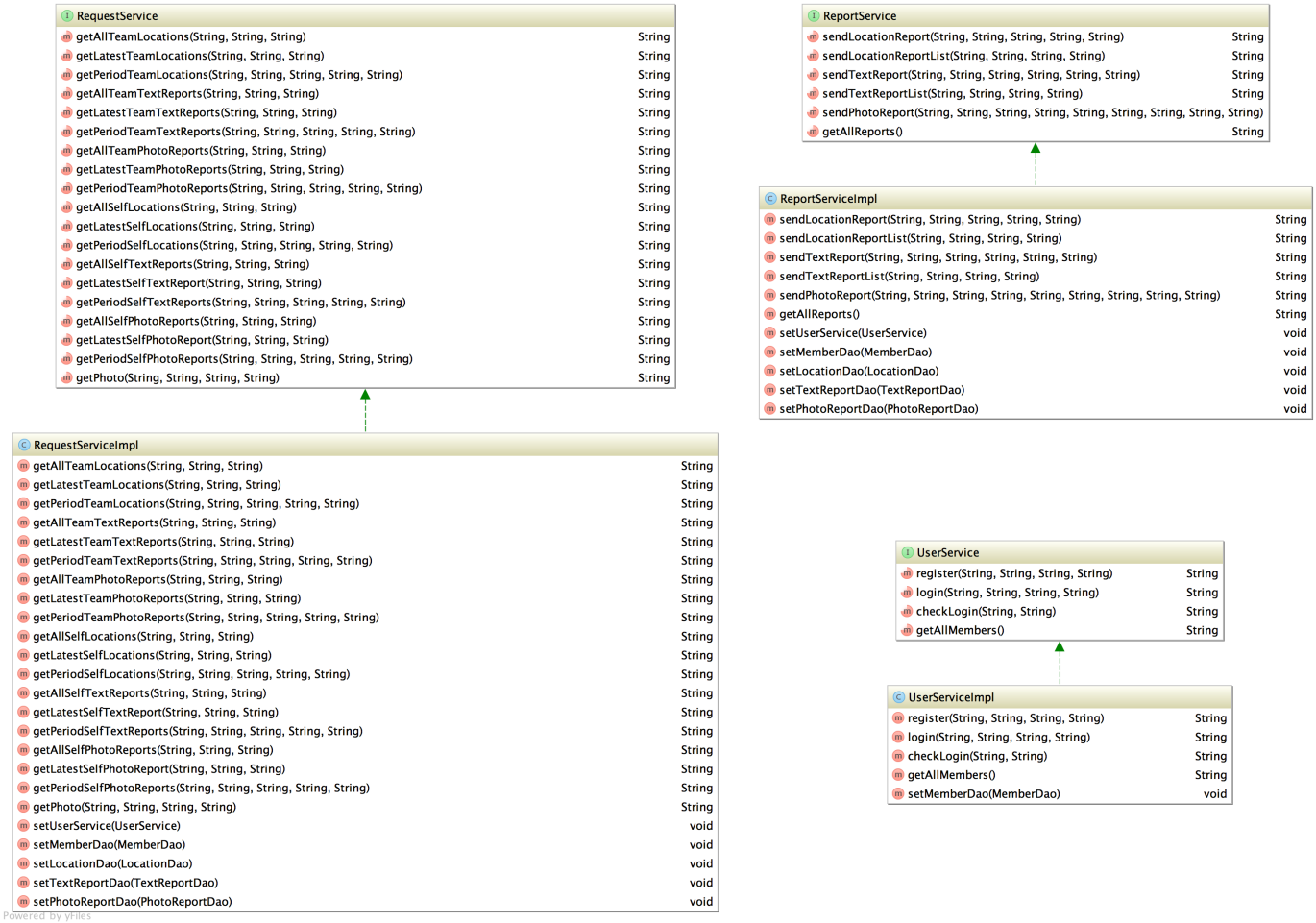


Figure 6.3: Diagram for the service layer

6.1.4 Controller layer

This layer implements all the controllers that manage the communication between the server and the client, including UserController, ReportController, RequestController.

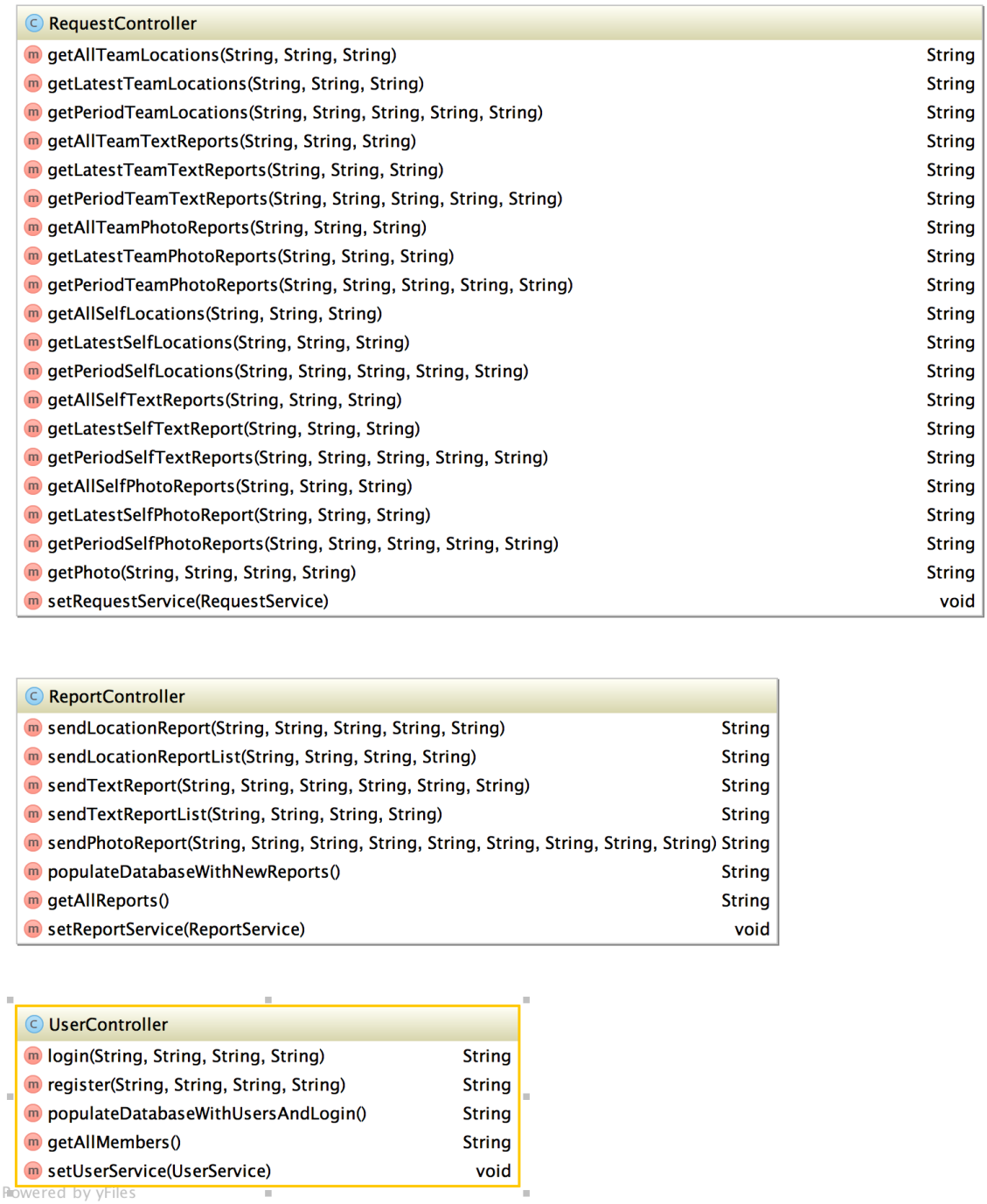


Figure 6.4: Diagram for the controller layer

6.1.5 Utility

The utility is implemented for encryption, decryption and storing constants.

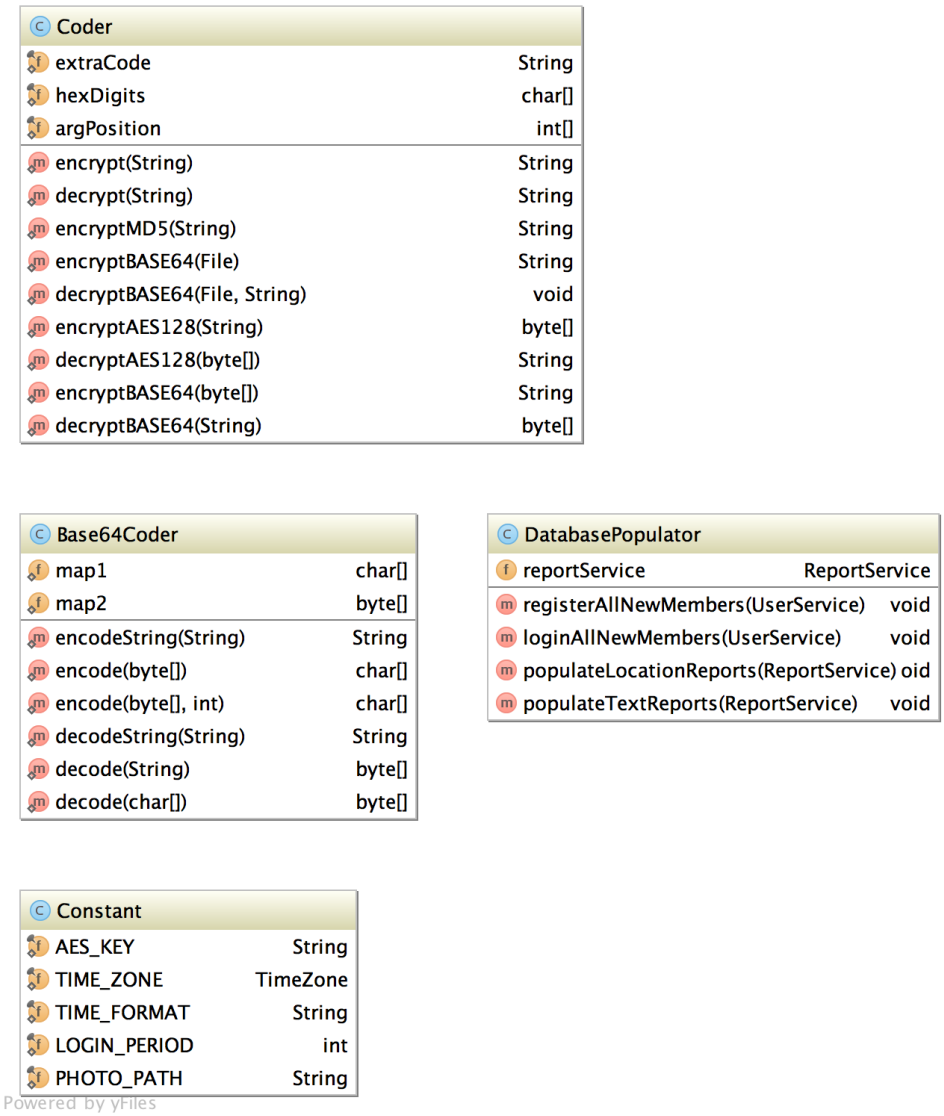


Figure 6.5: Diagram for the utility

6.1.6 Database

The database is designed for storing all the required information. On the server side, the database should store the member information, which is used for authenticating the request whether it is permitted or not from the client side. The database should also store all the reports' information in order to send them to the other teammates later on. In addition, an event log is stored for later checking. All the tables match the models defined above in the model layer.

Name	**Description**
id	Id is the primary key which is generated automatically for the table
Password	The password made by the user when registered and used on login
Name	The full name of the user
Uuid	Uuid is an identified code which is generated from a certain member and a certain device. It is used to keep the login status for a certain member on a certain device
ValidTime	ValidTime is used to hold the login status for a period. As long as the member has not used the application before the valid timestamp, the user would have to login in again to continue communicating with the server
TeamId	TeamId is used to separate members into different teams

Table 6.1: Member table attributes

Location Table

Name	**Description**
Id	Id is the primary key which is generated automatically for the table
MemberId	MemberId relates this location tuple to a member in the member table
latitude	Latitude records the latitude of the location related to the report
longitude	Longitude records the longitude of the location related to the report
ServerTimestamp	The timestamp at which the server received this location
ClientTimestamp	The timestamp at which the client created this location report

Table 6.2: Location table attributes

TextReport Table

Name	Description
Id	Id is the primary key which is generated automatically for the table
LocationId	Relates this textReport to a location in the location table
Content	the textual content of the report

Table 6.3: TextReport table attributes

PhotoReport Table

Name	Description
Id	Id is the primary key and is generated automatically for the table
LocationId	Relates this photoReport to a location in the location table
Name	Name is used to store the name of the photo file which is stored in the server
Extension	The file extension of the photo
Description	A textual description of the report
Direction	Describes the direction of the member when the photo was taken

Table 6.4: PhotoReport table attributes

EventLog Table

Name	Description
Id	Id is the primary key and is generated automatically for the table
MemberId	MemberId relates this event to a member
Description	A textual description of the event that has occurred
ServerTimesamp	Records the timestamp when the server handled the operation
ClientTimestamp	Records the timestamp when the client handled the operation

Table 6.5: EventLog table attributes

6.2 Client Implementation

Implementation on Client side was strongly influenced by iterative design. MVC was not designed at the very beginning, and the reporting system was re-implemented to better fit the requirements. This part documents how the Client was implemented at the end.

6.2.1 System structure

Three packages were implemented as we use a MVC Architecture pattern.

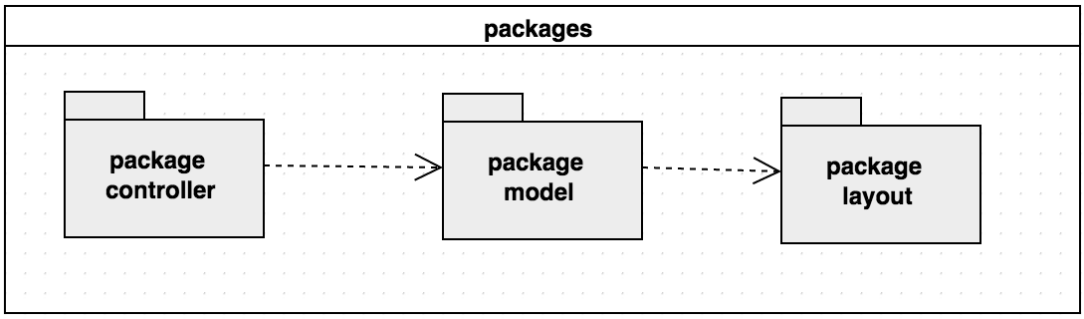


Figure 6.6: Package overview

Package controller There are 8 controllers which controls different activities separately.

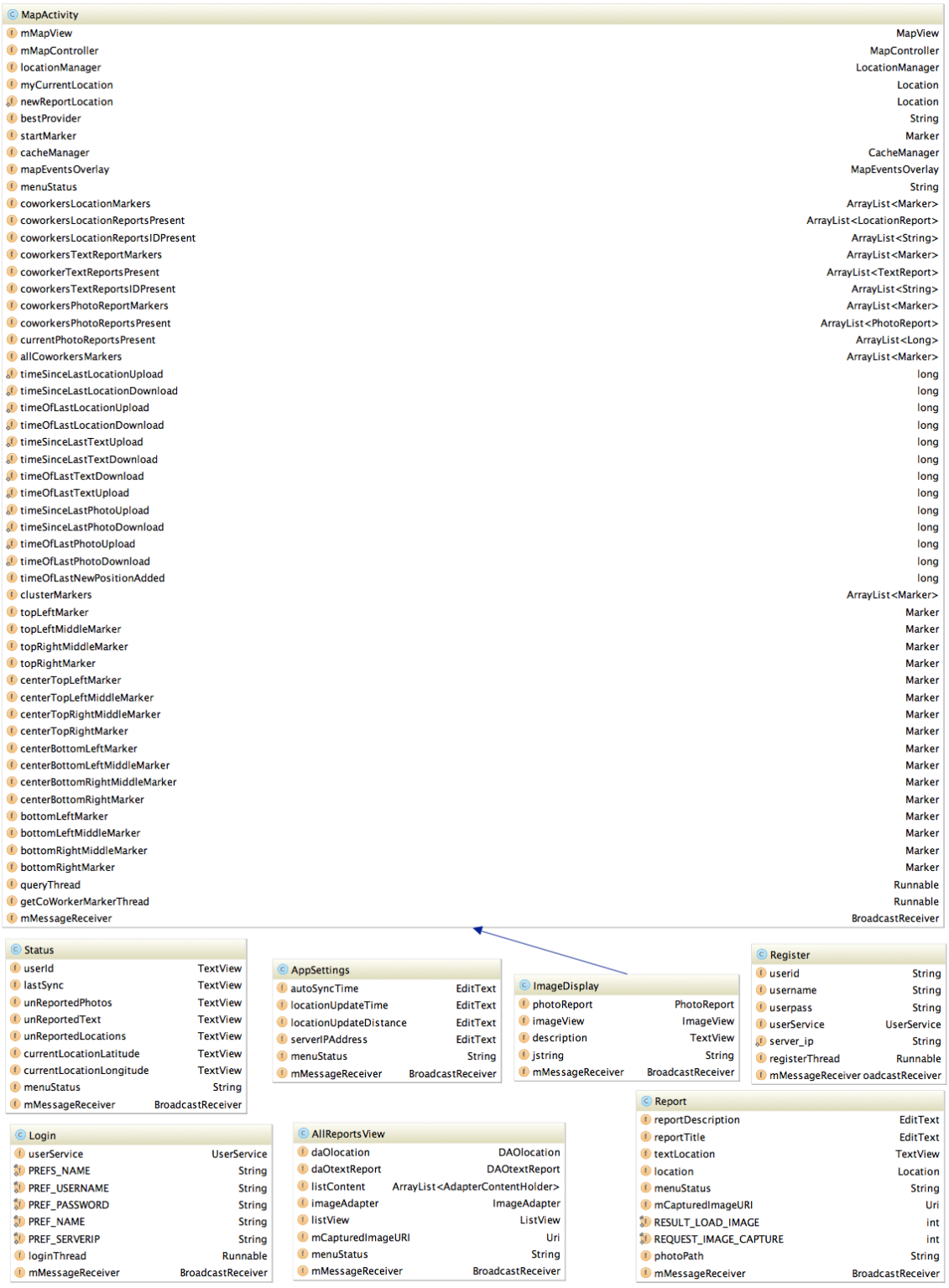


Figure 6.7: Classes in controller package

Package model As described in Design, model has a big package, so we divided it into four smaller packages, which makes our code more organized.

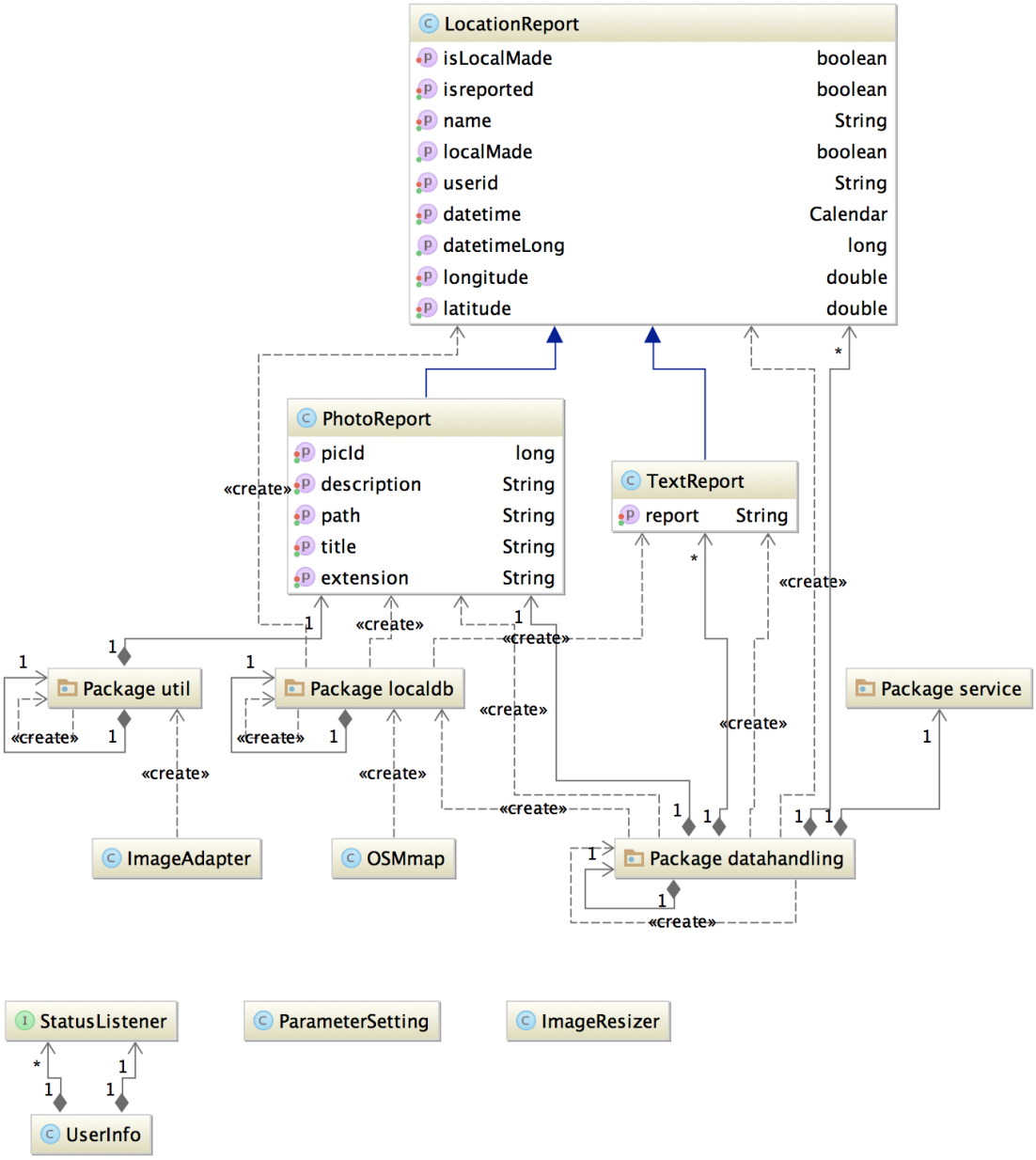


Figure 6.8: Content of model package

Package model: sub-package datahandling Handling data, synchronise data between server and client.

Figure 6.9: Content of model sub-package: datahandling

Package model: sub-package localdb contains local database, data access object.

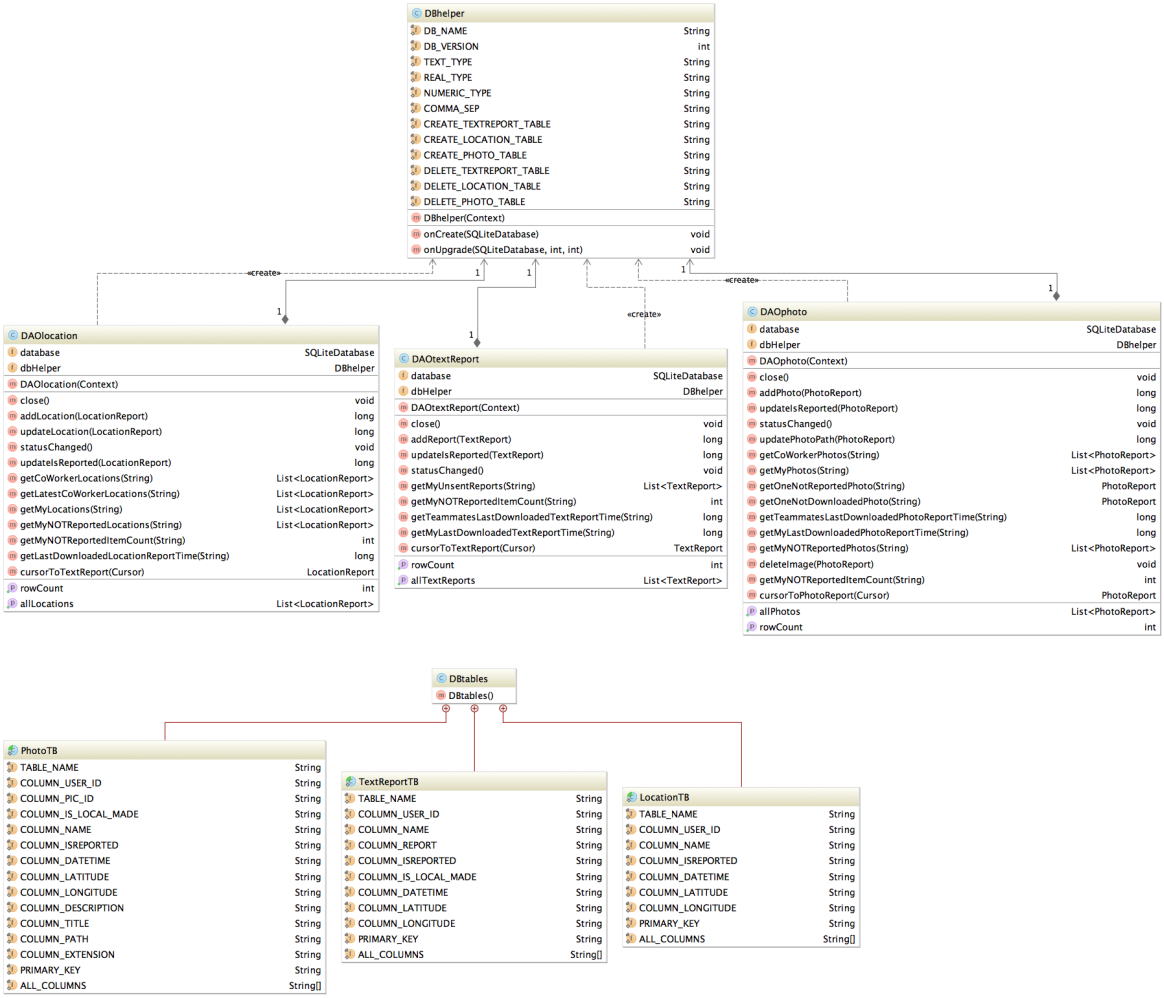


Figure 6.10: Content of model sub-package: localdb

Data Access Objects

Data Access Objects were used to read and write data to database.

Tables

DBtables is a static Class which has 3 inner class to define the table names and column names, which were used to keep the code organized, give easy access and reduce the chance of writing wrong names for tables and columns.

Package model: sub-package service Interface for service, and implementation is in sub package: imp. service package is responsible to communicate with server.

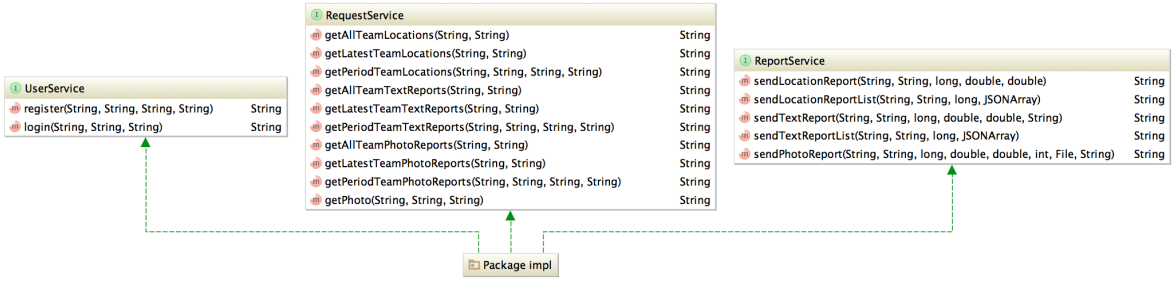


Figure 6.11: Content of model sub-package: service

Package model: sub-package service: sub-package imp
The implementation for service.

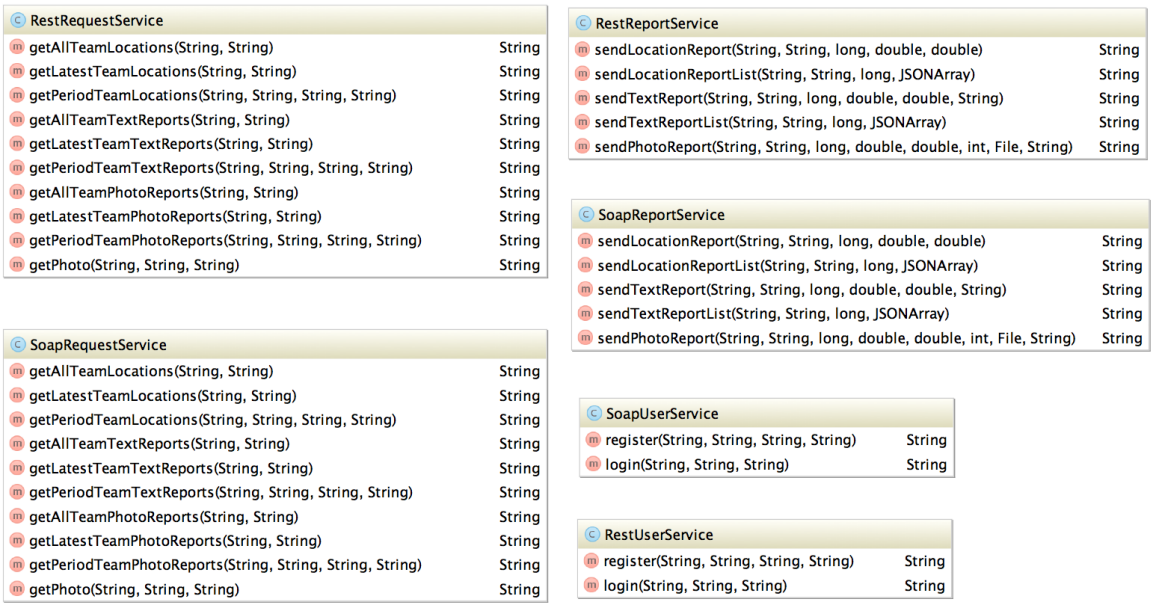


Figure 6.12: Content of model sub-package: service/imp

Package model: sub-package util

Contains all utilities that service need.

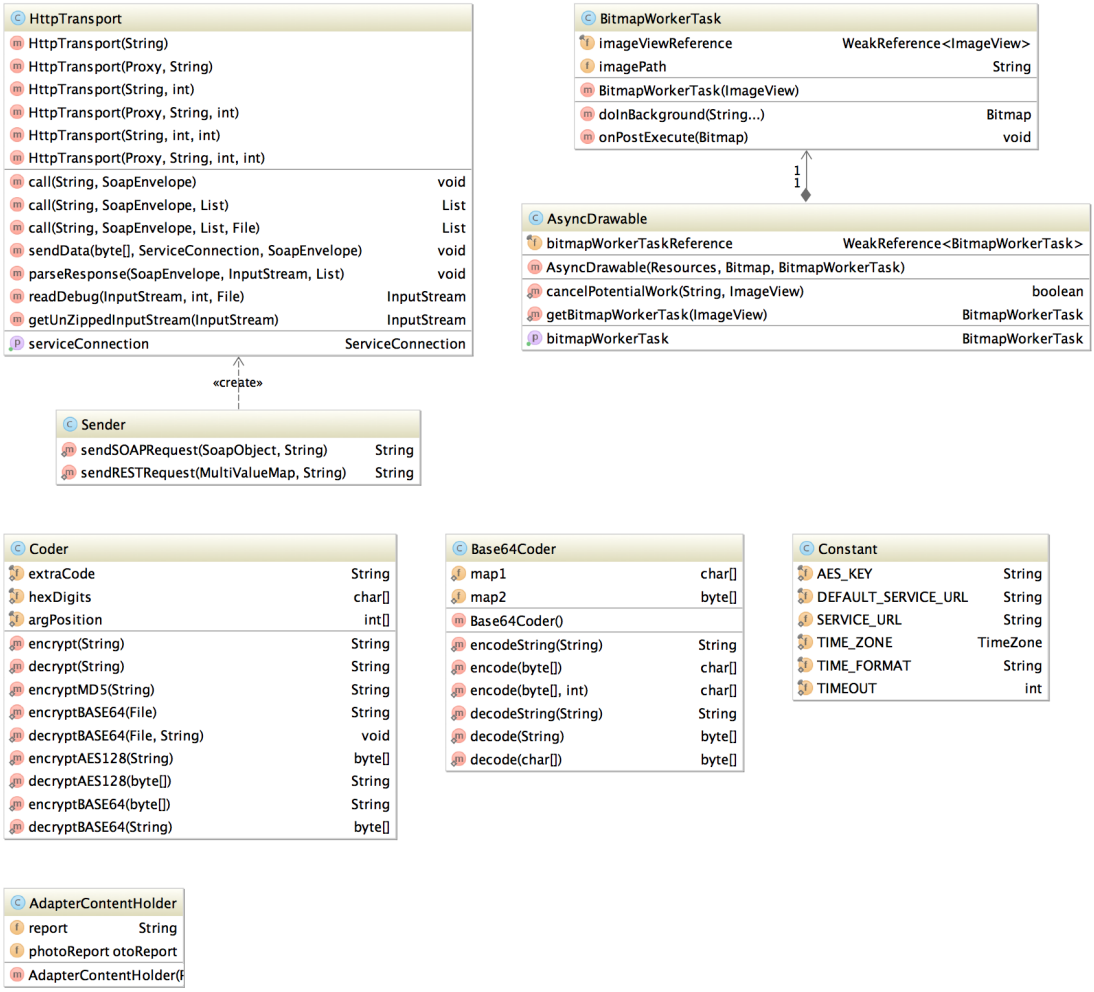


Figure 6.13: Content of model sub-package: util

Package layout (views) Represented as xml files. We have screenshots of how the layout (views) look like and how it can be used in the user menu part.

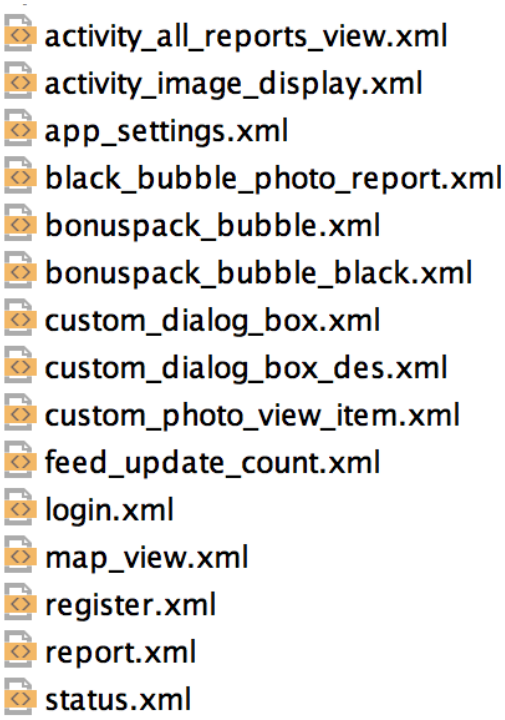


Figure 6.14: Content of layout package

6.2.2 Reporting system

The Reporting system handles reporting and data uploading to the server.

When a report is made:

- It will first be saved in the local database
- Then marked as not reported

If an active user send the report to the sever(by clicking the send button)

- System will check if there is network available
- System will also check if there are reports marked as not reported to the sever
- If both are true the report which has not been reported will be sent to the server
  - If the server received successfully, the report will be marked as reported
  - If the server did not successfully receive the report, the report will remain as not reported and wait for the next send action function to be called

If the user did not trigger the "send to server" function, the system will do it automatically (auto sync). The system will periodically run the "send to server" function at a set time interval. (see Figure: Reporting system flow)

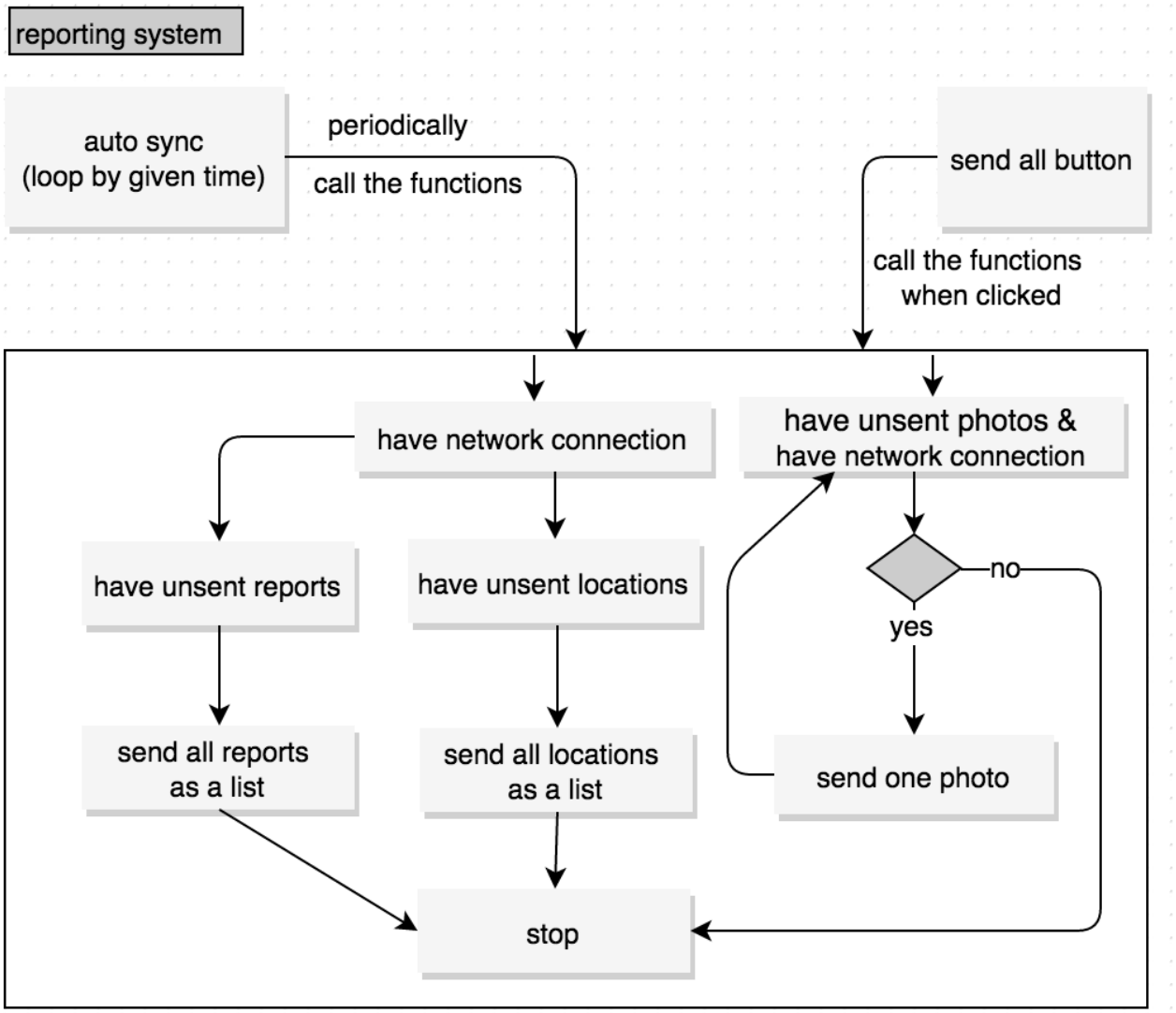


Figure 6.15: Reporting system flow

- Location report and text report are usually very small in size and can be sent as a package to the server
  - All not reported text reports will be packaged as a list and sent to server
  - All not reported location reports will be packaged as list and sent to the server
- Photos are usually very big in size compared to text or location reports. The best way that ensures photo reports being sent to server in all network conditions, is by sending them one by one

- Photo report has its own sending system which checks unsent photos and the network connection. It will send one photo to the server at a time when both conditions are true. If the server successfully receives the photo, it will be marked as reported.

Downloading system The Downloading system handles the downloading of reports (observations) from team members which are saved on the server. When the download succeeds, the data (reports) will be saved to a local database.
The Downloading system works similar to the reporting system (see figure downloading system). When the user activates the "send to server" function (send button), the downloading function will also be activated.

- If the user did not trigger the "send to server" function, the system will do it automatically (auto sync), where it will periodically run the "send to server" function and the "download report" function. (see figure downloading system)

If there is network connections the system will try to download all the not downloaded location reports and text reports as a list separately.

- The system uses a time stamp to identify which data(reports) are not downloaded from the server. A last succeeded download from server time stamp, will be sent to the server and asks the server for any items (reports) after this time.

Photo downloading also has it own downloading method there.

- Client will first download a list of photo reports with all the information(location of the photo, text descriptions, time). Photos will not be downloaded here, this list will be very small in size, but not the photos.

- When the client got the list of photo reports, the system will start to download the photos one by one, and mark the succeeded downloaded photos as downloaded, otherwise the mark will remain as not downloaded from server. This process will be repeated until all photos are successfully downloaded(if there is a network connection).

The whole downloading process will be periodically repeated.

6.2.3 Map implementation

Initially Google maps api v2 was intended to be used for the implementation of map features. This was because it was the most known to the group, and the google maps application is well known. After a short time the team found out that there was some key features needed for the Application which was very hard, or impossible to implement using the Google maps api. The main problem here was the use of caching or pre-fetching maps to use in offline mode, as this is not allowed in the way the app intended in the google maps api's terms:

"10.1.3 Restrictions against Copying or Data Export. (b) No Pre-Fetching, Caching, or Storage of Content. You must not pre-fetch, cache, or store any Content, except that you may store: (i) limited amounts of Content for the purpose of improving the performance of your Maps API Implementation if you do so temporarily (and in no event for more than 30 calendar days), securely, and in a manner that does not permit use of the Content outside of the Service; "

This means that a different way to implement maps needed to be found the group's application development . The best alternative to google maps was to use OpenStreetMap.

The main reason for choosing OSM is that it it possible to give your own local tilesets to the app, so that it will run without a connection with the internet. In that way the users could before starting the mission receive the relevant map tiles that would be used, and then be able to use the map functions in the app at any times regardless of the connection status. Also it is possible to give your own tile sets that are independent of the online OSM maps. This means that if there for instance was a mission in the mountains, you could instead load up the maps that are created for hiking and navigating in the mountains instead of the standard maps that are better optimized for urban areas.

6.2.4 Client database

The client database was implemented to make the client work independently when there is no network. When the app encounters a Disconnected, Intermittent and Limited environment, the local database can store all the reports locally. This includes location,text, and photo report. This was implemented to avoid loss of data which has been sent from the client side, but has not been received by the server yet.

There are three tables in the database:

LocationReport

Name	Description
UserId	Userid tells which user it belongs to
Datetime	Datetime records the time when the report was created
latitude	Latitude records the latitude of the location related to the report
longitude	Longitude records the longitude of the location related to the report
isReported	Represents the current status of the report

Table 6.6: Location Report attributes

TextReport

Name	Description
UserId	Userid tells which user it belongs to
Report	Represents the actual textual report
Datetime	Datetime records the time when the report was created
latitude	Latitude records the latitude of the location related to the report
longitude	Longitude records the longitude of the location related to the report
isReported	Represents the current status of the report

Table 6.7: Text Report attributes

PhotoReport

Name	Description
UserId	Userid tells which user it belongs to
PhotoId	PhotoId relates this report to a specific photo
Datetime	Datetime records the time when the report was created
latitude	Latitude records the latitude of the location related to the report
longitude	Longitude records the longitude of the location related to the report
isReported	Represents the current status of the report

Table 6.8: Photo Report attributes

Chapter 7

Testing

This chapter presents the various tests that were performed to ensure the stability and robustness of the application. Only some of the test cases are included in this section, the rest can be found in appendix F. The tests included focus on functionality, user interface, server response times, and compressions and battery testing. Unit tests were considered, but are not included due to reasons that will be explained in chapter 8.

7.1 Functional Testing

This section contains a summary of the functional tests that were performed. These tests were based on the textual use cases(scenarios) in section 4.2, to assert that the necessary functionality had been implemented. The tests were run by following the scenarios as described in 4.2 and ensured that the system responded as expected.

7.1.1 Login and Logout

Various tests had to be performed to ensure the stability of the login functionality in table 4.1. The logout functionality from table 4.7 was also tested. A couple of the test cases are included here, the rest can be found in Appendix F.

7.1.1.1 Login Success

Title	Login Success
Description	The user wants to login to the application
Preconditions	The application is running and the user is logged out. Network connection
Sequence of steps	1.Enter valid username 2.Enter valid password 3.Click the login button
Expected Results	The login Succeeds and the user is directed to the main map view
Pass/Fail	Pass
Comment	

Table 7.1: Login Success

7.1.1.2 Log out

Title	Log out
Description	The user wants to log out of the application
Preconditions	The application is running and the user is logged in
Sequence of steps	1.Click Log out in the menu
Expected Results	The user will be sent back to the login screen where he can log in again if wanted, or he can close the app if wanted
Pass/Fail	Pass
Comment	

Table 7.2: Log out

7.1.2 Map interaction

These tests were based off the scenarios in table 4.3 to test how the map responded to user input.

Title	Map Interaction
Description	The user wants to interact with the map
Preconditions	The user is logged in and in the map view
Sequence of steps	1. Swipe finger 2. Pinch the map 3. Click a user location 4. Click a report location
Expected Results	1. The map moves according to the direction of the swipe 2. The map zooms in or out 3. A popup window displays the coordinates and user id 4. A popup window displays the current text and photo reports from the chosen location
Pass/Fail	1. Pass 2. Pass 3. Pass 4. Pass
Comment	

Table 7.3: Map Interaction

7.1.3 Update Map

These tests were based off the scenarios in table 4.3. Since the application should update the map autonomously at specified intervals, tests had to be performed from both a system and user viewpoint.

7.1.3.1 Update Map, User

Title	Update Map User
Description	The user wants to update his own and the team's positions, as well as observations on the map
Preconditions	The user is logged in and in the map view
Sequence of steps	Click the status button
Expected Results	The map is updated with all the new observations, his own and the team's locations
Pass/Fail	Pass
Comment	

Table 7.4: Update Map User

7.1.3.2 Update Map, System

Title	Update Map System
Description	The system should update the map periodically
Preconditions	The user is logged in and in the map view
Sequence of steps	Get data from server and update the map
Expected Results	The map is updated with all the new observations and user locations
Pass/Fail	Pass
Comment	

Table 7.5: Update Map System

7.1.4 Report Observations

These tests were based off the scenarios in 4.4. The user should be able to send a textual or photo report via the report screen. Since the application might be utilized in a DIL environment the tests also had to be performed without a network connection. Only the text reports are included here, the tests for photo reports can be found in appendix F.

7.1.4.1 Textual Report, With Connection

Title	Send Textual Report
Description	The user wants to send a textual observation
Preconditions	The user is logged in and on the report screen. Network connection
Sequence of steps	1. Click the text field 2. Input the desired text 3. Click report observation button
Expected Results	The report is sent to the server
Pass/Fail	Pass
Comment	

Table 7.6: Textual Report, With Connection

7.1.4.2 Textual Report, No Connection

Title	Send Textual Report
Description	The user wants to send a textual observation
Preconditions	The user is logged in and on the report screen. No network connection
Sequence of steps	1. Click the text field 2. Input the desired text 3. Click report observation button 4. Connect device to network
Expected Results	The report is stored in the local database, then sent to server when the device is reconnected
Pass/Fail	Pass
Comment	

Table 7.7: Textual Report, No Connection

7.1.5 View Observation

These tests were based off the scenarios in 4.5. An important aspect of the application is to allow the user to view information about relevant observations.

7.1.5.1 View Observation from Map

Title	View Observation from Map
Description	The user wants to view information about an observation
Preconditions	The user is logged in and on the Map screen
Sequence of steps	1. Click an observation on the map
Expected Results	A popup displays observation information including, user id, GPS coordinates, and content
Pass/Fail	Pass
Comment	

Table 7.8: View Observation

7.1.5.2 View Observation from Report View

Title	View Observation from Report View
Description	The user wants to view information about an observation
Preconditions	The user is logged in and on the report view screen
Sequence of steps	1. Click an observation from the list
Expected Results	A popup displays observation information including, user id, GPS coordinates, and
Pass/Fail	Pass
Comment	

Table 7.9: View Observation from Report View

7.1.6 Automatic background tasks

These tests are based of the scenarios in 4.8 and 4.9 which is about automatic tasks running in the background.

7.1.6.1 Automatically sync with server

Title	Automatically sync with server
Description	The user wants the data to be updated automatically after a certain amount of time, so that the information shown is relevant without the user specifically calling for updated information manually.
Preconditions	The application is running and has a connection to the server and user is logged in
Sequence of steps	1. Be logged in and on the Map screen 2. Go out of the app(without closing it), turn of the screen or just wait for a while (at least longer than the set time interval) 3. Make sure that something is changed (Team member's positions, new observations etc.) 4. Go back in and check the map and see if the new or updated information is showing on the screen
Expected Results	All changes happened during the time will be updated and shown correctly on the map
Pass/Fail	Pass
Comment	

Table 7.10: Automatically sync with server

7.1.6.2 Automatically store files locally

Title	Automatically store files locally
Description	The user wants to be able to use the application also when there is no network connection with the server. To enable this the data needs to be stored in a local database with the latest data it has access to
Preconditions	The application is running and has a connection to the server initially
Sequence of steps	1. Be logged in and on the Map screen 2. Refresh the data and take notes of how it looks 3. Turn of the network connection on the device 4. Make some changes to the data on server(Move team member's positions, create new observations etc 5. Check if the application is still running with all its function, but instead of showing the changes and the newest data, it will only show the previously stored version until connection is regained
Expected Results	App will work as normal, but not update any of the data from external sources until it regains connection to the network.
Pass/Fail	Pass
Comment	

Table 7.11: Automatically store files locally

7.2 Usability and UI testing

A Mock up UI was implemented for the purpose of user interface testing before the actual app was completed. The test was carried out by asking a couple of friends to test the UI. We received constructive feedback that has aided us in crafting and refining the user interface.

Tasks	Create user and login
Result	The test subject successfully logged in
Comment by the tester	There is nothing to say, this is just like any other registration/login screen you would find

Table 7.12: Login usability testing

Tasks	Navigate to different screens inside the application. The tester gets a few things thats needs to find(last sent location, settings, look at sent photo report etc)
Result	The tester eventually got around to every screen
Comment by the test	The screens were pretty easy to find since all of it is in the dropdown menu, but the descriptions was a little bit confusing

Table 7.13: Views usability testing

Tasks	Send a picture and a text report
Result	The tester had trouble finding where the photo button was
Comment by the tester	I didn't find the add photo button at first because I thought that the button would be in the report screen but it was in the photo view screen

Table 7.14: Sending report usability testing

7.3 Performance Testing and Analysis

7.3.1 Analysis Purpose

The purpose of this project is to develop an app for FFI that applies military functions to civilian commercial products. Some important indicators will describe the quality of the performance for the app, such as battery consumption, responding time, bandwidth usage. In order to select proper ways to improve the quality of performance, the analysis should be conducted by using the app according to those different ways.

7.3.2 Analysis Variables

Different variables affect the performance of the applications in different ways. To find the different affections, the variables are set into three groups, SOPA vs. REST, With GZIP vs. Without GZIP and MySQL vs. H2.

MySQL vs. H2

MySQL and H2 are two different databases. MySQL is widely used and H2 is a database which is developed by Java. Since performances of both databases are uncertain in the application, they are set as two variables in the same group.

SOAP vs. REST

SOAP and REST are two different implementations for web service. The application uses Ksoap2 and Apache CXF to implement client and server side for SOAP. The communication between client and server side is based on XML format documentation. Compared with SOAP, the application implements REST with Spring for Android and Spring MVC. The communication is based on JSON format documentation. Since both implementations are totally different, the performances for each are therefore uncertain. To find a higher performance way, these two implementations are set to be in the same group for analysis.

With GZIP vs. Without GZIP

For sending request to server and receiving response from server, app would spend less energy by using compressed messages than using original ones, However, compression by GZIP itself consumes energy. So which way spends less energy is uncertain in this case.

7.3.3 Analysis Process

In order to make results more comparable, the general environment will be set as equal as possible. To avoid affection of unstable network, test will be conducted locally in LAN. Server will connect wired to the router while android will connect wireless to the same router.

Sever	....
Operation System	Debian Release 7.7 (wheezy) 64-bit Kernel Linux 3.2.0-4-amd64 GNOME 3.4.2
Hardware	Memory: 3.9GB Processor: AMD A10-5800K APU with Radeon(tm) HD Graphics x 2

Table 7.15: Server Attributes

Client	....
Mobile	Xiaomi MI 4
Operation System	Android Version: 4.4.4 KTU84P
Hardware	Memory: 3.0GB Processor: Processor: 2.5GHz quad-core

Table 7.16: Client Atrributes

Specified conditions for database test
Client application: AndroidTest
Database1: MySQL
Database2: H2 embedded(together with web service in the same server)

Based on these conditions, two tests have been conducted as following:

1. Time reporting 100000 TextReport
2. Time requesting 100000 TextReports

Specified conditions for web service performance test
Client application: AndroidTest
Database1: H2 embedded
Web service application: sair-1.0-SOAP-GZIP-RELEASE
Web service application: sair-1.0-REST-GZIP-RELEASE
Web service application: sair-1.0-SOAP-NoGZIP-RELEASE

Web service application: sair-1.0-REST-NoGZIP-RELEASE

Based on these conditions, five tests have been conducted as following:

1. Time reporting 1 TextReport
2. Time reporting 100000 TextReports
3. Time requesting 1 TextReport
4. Time requesting 100000 TextReports
5. Battery usage running for 1 hour

7.3.4 Analysis Results

Some analysis results for the database, performance, and battery tests are included in this section More tests can be found in Appendix F.

Results for database test

Time reporting 100000 TextReports (ms)	MySQL Server	H2 Embedded
1	205420	28886
2	202499	26480
3	199878	27871
4	197643	28241
5	200340	26578
6	197890	26858
7	198003	28928
8	200011	28248
9	204835	27484
10	199099	28856
Average	200561.8	27843

Table 7.17: Time reporting 100000 TextReports (ms)

Results for web service performance test

Time reporting 100000 text report (ms)	SOAP with GZIP	SOAP without GZIP	REST with GZIP	REST without GZIP
1	28886	25493	58136	33282
2	26480	22008	34725	32311
3	27871	23077	45090	31883
4	28241	21830	44622	31239
5	26578	23249	43924	31878
6	26858	22196	46529	34566
7	28928	24817	48304	33447
8	28248	24964	46457	35752
9	27484	22385	46802	31596
10	28856	22368	43749	32741
Average	27843	23238.7	45833.8	32869.5

Table 7.18: Time reporting 100000 text report (ms)

Battery usage running for 1 hour (J)	SOAP with GZIP	SOAP without GZIP	REST with GZIP	REST without GZIP
1	49.1	49.9	40.6	39.9
2	47.9	52.3	38.7	42.1
3	53.1	56.7	38.5	41.2
4	49.5	54	37.9	43
5	48.3	53.9	39	40
Average	49.58	53.36	38.94	41.24

Table 7.19: Battery usage running for 1 hour (J)

7.3.5 Analysis Conclusion

The results for database test show that by using H2 Embedded the time reporting 100000 text reports and the time requesting 100000 text reports are much less than by using MySQL server. Therefore using H2 Embedded is more effective than using MySQL server in the application.

The results of the web service performance test indicate that by using SOAP the time spent reporting and requesting text reports are in general less compared to REST. However, the results also show that using GZIP takes a little longer time than without using GZIP for reporting text reports while using GZIP takes a little shorter time than without using GZIP for requesting text reports. In addition, from the result of battery usage running for one hour, it points out that by using REST consumes less power than using SOAP in the test cases.
Hence the conclusion is using H2 embedded is the priority compared with MySQL server in the application for a target of high performance. For the web services, SOAP is more efficient than REST but consumes more battery power. GZIP improves the performance more for big size content such as a long list of reports or an image report. But it is not worth using when sending only few location reports or text reports.

Chapter 8

Evaluation

This chapter evaluates the project and most importantly conclude what we have learned from the project. We start by evaluating the project management, which we think is an important aspect for future projects. That is followed up by the Design and Implementation. Finally we conclude and evaluate the whole project.

8.1 Project Management

8.1.1 Scrum

The Scrum methodology worked well for us. Although we did not follow scrum as strictly as a massive scale project might require, the version we utilized suited the needs of the project. As an example, we did not have a scrum master to organize the scrum meetings, since we felt it was unnecessary. The group agreed that a flat group structure would better suit the work environment of the group. This encouraged members to take initiative and enhanced the group dynamic. Although this worked well for the most part, we could have benefited from a scrum master in the early stages as it took some time for us to get started with the sprints.

Scrum meetings were carried out every time we met(not at the beginning), and it was used more as a tool to ensure each group member had an assigned task, and everybody is up-to-date with the on going project. During the shared work hours we also utilized pair programming. This allowed members to get support if they were stuck on a certain task. This occurred quite often as everyone on the team was new to android programming and proved to be an especially useful tactic during the development of the application.

The fact that we chose an iterative development methodology helped us to a great extent during the development of the application. Frequent meetings with the customer ensured we felt safe concerning the product requirements. Although some changes had to be made (more on this in section 8.2) based on the feedback from the client, it didn't result in unnecessary complications as we felt suitably

prepared.

8.1.2 Group Communication

Since we spent a lot of the time working together in the same room, communication was never much of an issue. The times when we were not working together we used Facebook to keep each other up to date. This worked well for the most part although there was a couple of instances when messages could have been conveyed more clearly. An example is the user manual which was supposed to be attached to the report sent to to the customer before the demo of the application but then was left out because of a communication error. Luckily this was not a major issue according to the client as changes were made to the application before the physical demonstration.
As mentioned earlier we had fixed working days every week to allow for frequent face-to-face communication. This meant that we could have discussions instantly whenever problems arose or decisions had to be made. This proved efficient during development and we felt it was necessary because during the project:

1. Continuous and frequent changes were made.
2. Continuous design and redesigns were made.
3. The complexity of the application was continually increased.
4. The functionality was continually growing.
5. Different models, layers and packages were made and assigned to different team members.
6. Most of our team members had other courses or projects. Everybody could not always remember what was done the last time, what were the on-going problems that needed to be fixed, what the other team members were doing and how the whole project process is going.

Improvement of communication The Group communication could have been more efficient at the start of the project. Scrum meetings were not carried out at every meeting, and we were not used to summarize and share with group members about what has been done since last time, what was the problems/challenges we met and what was the present days plan etc. During the project work we were progressively learning. Our scrum meetings increased in both frequency and efficiency. From having meetings on an irregular basis, we started having them at the beginning of the day every time we worked together. At some of the first scrum meetings we had, it could take up to 40 minutes to finish them. This was was way too long, and people were not able to stay concentrated during the whole meeting, resulting in not everyone getting all the information. After some weeks of working we were able to finish the meetings with only using about 10-15 minutes, while stile being able to clearly communicate what we wanted.

8.1.3 Time management

Time management is a significant part of the project. By following the plan, we estimated the time and tried to complete the tasks in time for every sprint. We used WBS to make an outline of the project and estimate how much time we would use on the different parts at the beginning. Sadly we did not use it as much as we should have in the beginning, and we later on realised that we should have used it for more detailed planning of the separate parts. Some minor problems occurred during the sprints but these were solved easily.
There were certain issues:

1. At the start planning could have been better, and therefore we didn't get a sprint back-log for the first sprint, or burn-down charts for the first 2.
2. Because of the poor planning, some of the first sprints ended up either being finished way ahead of time, or taking longer than expected.
3. We also had some problems during sprint 6 because of difficulties with some of the features in the application taking a lot longer to implement and fix so that they worked satisfactory, so that we used a lot more time during this sprint and had to extend some of the tasks for the next sprint.
4. The WBS was used but not to the extent that is to be expected in Scrum.

Luckily, our supervisor noticed us about the problem and gave us the feedback we needed to get back on track so that we could avoid the problems about time management later on.
Overall we did not have too many problems with time management. We were able to finish all the work we wanted to finish before each important deadline, and we managed to deliver everything on time without having to work too heavy hours in front of each deadlines. This made the group able to not get too stressed, and we were able to be satisfied about what was delivered.

8.2 Design and Implementation

Since we used an iterative and incremental software methodology, changes were made both to the design and implementation during the development. This gave us many advantages but there were also some challenges during the development.

Problems and difficulties
Since only one person in the group had experience with android development beforehand, the rest needed to learn it from scratch. However, since everyone had experience from java programming, were eager to learn, and Android studio was very intuitive and well made, it did not take too long to learn the basics.
One of the problems we faced at the beginning was about changes. We used a Client-server model, where the client sends data or requests to a server, and gets a response, or downloads data from the

server. When the process is done, the client and server needs to convert data to its own format so that the data can be used (server: Java Servlet + H2; client:Android + SQLite). Client and server need to speak a common language(data format) so they can understand each other. When the application gets more complicated, the language also need to be more complicated (more complicated data format and more data). Once the server changed its data format, the client could not understand it correctly, even though server told client changes were made.

To solve this issue, the people responsible for the server and the client sat together for several days trying to come up with a solution. Finally the problem was solved by clearly defining interfaces for communication and data transfer. This ensured we did not face the same problem again throughout the rest of the project. It even allowed us to make switches between REST and SOAP to communicate between server and client without too many modifications to the code..

8.3 Conclusion

We are happy with the final product and the development process. An application for situation awareness and reporting incidents were implemented, and it is functioning well in a 'disconnected, intermittent and limited environment', which meet the main requirements. The later added requirements like encrypt information, to implement and test REST and SOAP to find the better one to use, redesign and re-implementing of the application to improve the performance and reduce the battery usage, redesign and re-implement user interface to meet customer needs. A few parts may still have space to improve[1], but the most important parts were successfully implemented.

Lessons learned:

- It would be nice to get to know each other better at the beginning, maybe spend some time for 'team building' or some other activities before or at the beginning of our project. As the success of the project is largely dependent on the skills and strengths of every individual. By knowing each other, we can improve the communication and set a common goal at an early stage; by knowing each others skills and strengths we can improve the working efficiency, and happily work with project.
- We should have made a better and more detailed plan for each task or module of the project. During some of the sprints, especially at the beginning of the project, some tasks were over- or underestimated, the actual time used were different from what we estimated. Extra time were used to catch the sprint, and time were wasted because we did not always have clear tasks ready. Luckily we did have some amount of extra time, so we were able to catch up whenever that was needed. We need to learn more about project management, and learn to better adjust if something differs from the original plan.

[1] Se appendix G for more on future work

- We should work more closely with the customer in the sense of testing the software and getting more frequent feedback on the product.
- We neglected parts of the report midway through. This meant we ended up having to play catch up alongside the application development. It would have been easier to continuously write the report in parallel with the programming.
- It would be good to have a leader for meetings and discussions to ensure that there is no misunderstandings and that all members are allowed to participate. Although we did not introduce this in our process we could have benefited from this in the early stages of the project.

Appendix A

Acronyms

3G Third Generation (Third generation of mobile telecommunications technology)

API Application Program Interface

App Application

AES Advanced Encryption Standard

CBC Cipher-Block Chaining

CPU Central Processing Unit

DAO Data Access Object

DIL Disconnected Intermittent Limited

DoD United States Department of Defence

ER Entity–Relationship

FFI The Norwegian Defence Research Establishment

GPS Global Positioning System

GUI Graphical User Interface

GZIP GNU Zip

HTTP The Hypertext Transfer Protocol

ID Identification

IDE Integrated Development Environment

IP Internet Protocol

JSON JavaScript Object Notation

LED Light-Emitting Diode

MVC Model-View-Controller

NTNU Norwegian University of Science and Technology

ORM Object/Relational Mapping

OS Operating System

OSM OpenStreetMap

RDBMS Relational Database Management System

REST Representational State Transfer

SMTP Simple Mail Transfer Protocol

SOAP Simple Object Access Protocol)

SQL Structured Query Language

SSL Secure Sockets Layer

UI User Interface

WBS Work Breakdown Structure

XML Extensible Markup Language

Appendix B

Report from customer meeting

Meeting with customer 26-01-15
Group: FFI-android
Meeting form: Skype meeting
Date: 26-01-2015
Time: 09.00 - 10.00
Next meeting: 09-02-2015

This was the first meeting with the customer and here the team got an overview over the task. Following is our current understanding of the task after the meeting. The application are to be made from scratch. This application shall communicate with a server which provides information to the user about incidents reported by other users. The main types of information that will be used is but not limited to: GPS location, Text(description) and images. There are some critical requirements concerning the application that must be considered when designing and building the product. the team have to take into account that the product will be used with high mobility or in areas with low bandwidth meaning the app have to be efficient in transferring data and finding solutions when interrupts in the connection occurs. Because of the mobility of the user or the chance of being located in remote places where the possibilities of charging the device is not available the application need to be cheap when it comes to the usage of battery power.

Answered questions:

- The team can use a public GitHub.
- A new application will be built.
- Information that will be passed: images, text, GPS location.
- The team are free to choose database and server technology.
- The main focus is on the client-side of the application.

- Critical constraints: Low bandwidth and scarce charging opportunities.

Meeting with customer 09-02-15
Group: FFI-android
Meeting form: Face to face
Date: 09-02-2015
Time: 13.00 - 14.00

This was our first face to face meeting with our customer. The customer explained us in detail what the specifications for the application was:

- The OS must not be rooted.
- The unit must be identified with a unique ID.
- Our app must be able to operated in DIL environment:
  - Disconnected
  - Intermediate
  - Limited
- The position the app send back must have:
  - A unique ID
  - position
  - timestamp
- If the app send in a observation it must either have text or picture or both, in addition to the position and its requirements stated above.
- Command must be able to see the latest positions and observations of all the units from the operation area.
- Battery life must be at least 4 hour without power bank.
- Our database backend must have a interface for further connect to other resources.
- The units must also be able to view the observations of other units.
- The map must be preloaded.
- The app must have intuitive feedback to the user:
  - If it is connected to the server

  - Was the observations sent correctly

- The app must be able to cache observations that failed to send due to limited connectivity.
- Autosend the observation once connection is restored.

The team have also received the following testing platforms from the customer:

- 2x Samsung galaxy tab 2
- 1x Samsung galaxy tab 4
- 1x ASUS Google nexus 7
- 1x Sony Xperia Z2

Meeting with customer 02-03-15
Group: FFI-android
Meeting form: Skype meeting
Date: 02-03-2015
Time: 12.00 - 12.30

This was the first meeting after the face to face meeting. The meeting was short and Frank was not present. The client reminded us that the report is as important as the product and that the team could use probing software to measure the battery usage.

Meeting with customer 09-03-15
Group: FFI-android
Meeting form: Skype meeting
Date: 09-03-2015
Time: 12.00 - 12.30

In this meeting the client got a rough status rapport about the application and the server. the client agreed to have a look at our report before the next meeting. The team will send our report before friday morning.

Meeting with customer 16-03-15

Group: FFI-android
Meeting form: Skype meeting
Date: 16-03-2015
Time: 12.00 - 12.30

1. Move figure from 5.1 to 2.4
2. Better description of the ER-diagram and other diagrams.
3. Create use cases for the rest of the functions from 3.2.1 with the automatic functions.
4. Describe how the login will handle no network connection
5. Describe what the consequences of being in a DIL environment has for each of the use cases
6. In use case 2, Define what locations means, or update it to be a more accurate description. Also add in that one can also use multi-touch controller, like dragging, pinching, and double clicking the map for zooming and navigating the map.
7. Add in the automatic updating and how it works from use case 4. Create a use case for checking observations, and adding updates or extra information about existing observation
8. For use case 5, add a description of how you should be able to click on a point in the map, and then create an observation with that as the location. Also User should be able to use a picture that he has already taken and has stored on his device, and not only take a picture right now.
9. For UML diagram 5.4: add text description about how it works.
10. Think about how we should test the battery usage of our application. Explore if there is any good techniques or profiling tools for accurate testing of battery usage.
11. Add possibilities to configure settings like: How often different things should be refreshed If you after being disconnected for a long time would send the position of all your recent locations, or only the last one.

Meeting with customer 08-05-15
Group: FFI-android
Meeting form: Skype meeting
Date: 08-05-2015
Time: 12.00 - 14.00

This was our second and last face to face meeting with the customer.

The customer goes through our report and discusses inconsistencies and final changes.
Some specific points include:

- citation to requirements chapter 4 figure
- more information on DIL, not just bandwidth
- more use cases for android functionality
- GPS coverage for use case1 login wrong
- Use case 2 id or username?
- use case 3 GPS coverage
- use case 4 automatics update
- use case 5 comments user should get feedback on data sent or not sent
- use case 7 add for report view
- use case 6 logout: no connection to server during logout
- use case scenarios could have been more specific
- more functionality in design
- report is not fluid enough to read
- a bit restrictive with a user tied to an observation
- relation between observations
- location for observations and users are always the same
- activity diagram discussion photo being sent one by one
- user interface be more specific
- more explanation for the implementation diagrams, variables especially

- team ID in implementation should be mentioned earlier
- testing how robust the application
- more description on sending report behaviour
- synchronized clocks could be problematic, mention it(evaluation)
- captions for functional tables in testing
- need to test the syncing
- database over all registered devices
- design needs to be expanded, more needs to moved up
- mention that the database tests and analysis are back-end and not local
- more justification for some of the decisions we made
- database testing caption
- mention how and why the tests are performed
- more details of software and libraries used in implementation, version number etc

After we went through the report, the customer went on and tested the software. After playing around with the software the customer was more or less satisfied with the software, there were a couple of bugs in the software that the customer wanted us to fix for the final delivery. There were a couple of Gui improvements mentioned by the customer that needs to be done.

Appendix C

Sprint Backlogs

C.1 Sprint 2

2 Implement first version android application
2.1 Implement android UI
2.1.2 Implement main activity layout
2.1.3 Implement first OpenstreetMap
2.2 Response/request handler client-side(SOAP)
3 Implement backend draft
3.1 Implement database handler draft
3.2 Create the database(MySQL) backend-side
3.3 response/request handler (SOAP) backend
3.4 Implement (Very) simple authentication system

Application prototype: Done on client side:

- register new user
- user login
- report location to server periodically
- download coworkers location periodically from server, all need to be done with communication to our server and those simple functions are tested and all works, which also give credit at our server side worked.
- OpenstreetMap implemented
- show users own location with a marker
- and show coworkers location with other markers

C.2 Sprint 3

Sprint 3 Client:
Finished jobs on Sprint 3 client side:

- restructure architecture:
  - save and read file locally first
  - use asynctask technique to upload and download files from server periodically in background
- implement local database: sqlite
  - save & read file to and from local database
  - test report table for text report
  - location report table for location report
- Improved UI
  - new dark theme style
  - add action bar button
  - new listview to show location report & text report
- OSM offline maps

Server

- Entity models
- Database object access functions
- Authentication logic design
- UserService for SOAP on server
- ReportService for SOAP on server
- RequestService for SOAP on server
- UserService for SOAP on client
- ReportService for SOAP on client
- RequestService for SOAP on client
- Gzip for SOAP on server
- Gzip for SOAP on client
- Setting up REST

C.3 Sprint 4

Sprint 4 Client: Finished jobs on Sprint 4 client side

- UI:
  - app settings window: user can set application parameters
  - status view window: show user status
  - Photo report view window: a list view with photo and its descriptions
  - Popup window for options choose: take photo or add from gallery
  - Popup window for write photo descriptions: title and description for photo
  - Photo view window: show a selected photo in full screen
- design pattern:
  - abstract class for upload and download
  - inheritance for photo and report
- functionality:
  - implement download, upload text report
  - add DAO photo: to handler local database
  - add photoAdapter: to handle photo list view
  - add imageResizer: to reduce memory use
  - add PhotoReport: to hold photo information
  - add PhotoView, PhotoDisplay . . . ect to view photos and photo report.
  - View own and other users location on the map.

Server:

- Optimize database access object
- Analysis database performance

Testing:

- Test connecting server and client
- Test all SOAP functions
- Test all REST functions

C.4 Sprint 5

Sprint 5 Client:

- UI:
  - Show observerations on map
- functionality:
  - new upload photo system for photo upload:
  - photo upload uses its own upload functions to ensure the upload get done
  - upload one photo at a time, until there is no network or no un-uploaded photos.
  - new photo download system:
  - download latest photo report list(all informations about a photo report: location, time, description ...)
  - save the list into local database, when it is done,
  - download each photo file from server.
  - each of the download photo file are saved private to the application
  - only this application can use the photos downloaded from server, but the photo taking by user are saved publicly as the copyright belongs to the user).
  - it works! All major functions works :)

C.5 Sprint 6

merge 1:

- add remember me function: auto login with or without network connection after first time login(need network to confirm)
- fix bug:
  - bug: logout did not close all activities, user can come back with back button.
  - fixed: all activities will be closed after logout is clicked and use will be send to login window.
- Reorganize structure (MVC).
- delete test package and all classes inside, as we are certain they are not needed anymore.
- fix bug:
  - bug: when logout, login window opened many times.

  - fixed: when logout, login window opened only once, when user hit back button app will close.

merge 2:

- add auto logout function(security reason): user have to login again if server tell the app that this user need to do a login
- dynamic show status at menu
  - add total unreported items count to title
- dynamic status view
- add observer pattern to observe the changes;
- Working prototype with all basic core functions

C.6 Sprint 7

- All functions has been implemented in the client, still a decent amount of bugs
- User manual has been made
- Chapter 1,2 and 4 of the report has been finished, still needs proof reading and minor adjustments.

C.7 Sprint 8

- Remaining bugs is fixed
- Software has been completed and is ready for release
- Javadoc has been created.
- Report is finalized and ready for delivery
- Group and self evaluation forms has been filled

Appendix D

Time Management

D.1 Burndown charts

Sprint 3		
Task	Time estimated	Time spent
Documention	80	68
Gui design	10	8
Gui implementation	20	14
Gps Fix	10	4
Map implementation	30	20
Local database	20	12
Gzip	20	9
Server side	40	41
Testing	10	5
Total	240	181

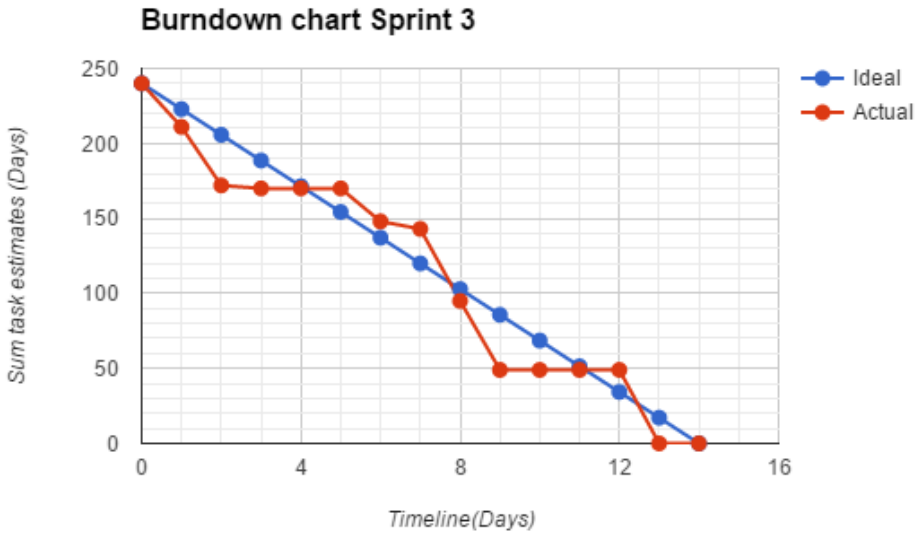


Figure D.1: Burndown chart sprint 3

Sprint 4		
Task	Time estimated	Time spent
Documention	90	85
Gui design	20	25
Gui implementation	20	15
Server side	10	11
Server - Client communication	20	15
Client report	10	4
Debugging	20	23
Client side	40	36
Testing	10	8
Total	240	222

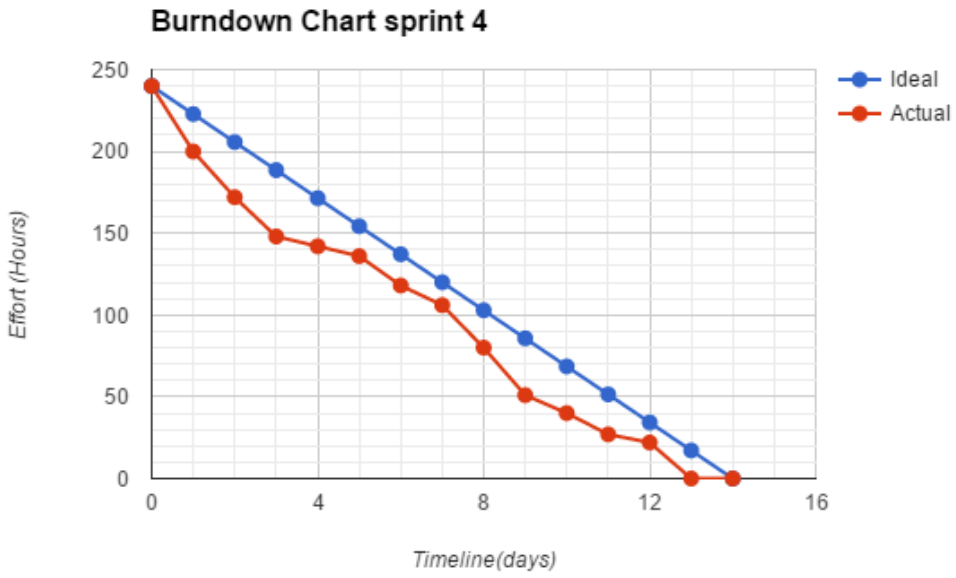


Figure D.2: Burndown chart sprint 4

Sprint 5		
Task	Time estimated	Time spent
Documention	80	76
Finish Photo report UI	15	9
Edit+ locations and obs.	15	24
Team locations	20	26
Photo handling client-server	25	36
Report obs. activity	20	27
Create mock UI for test	15	17
Server side handling	30	24
Total	220	239

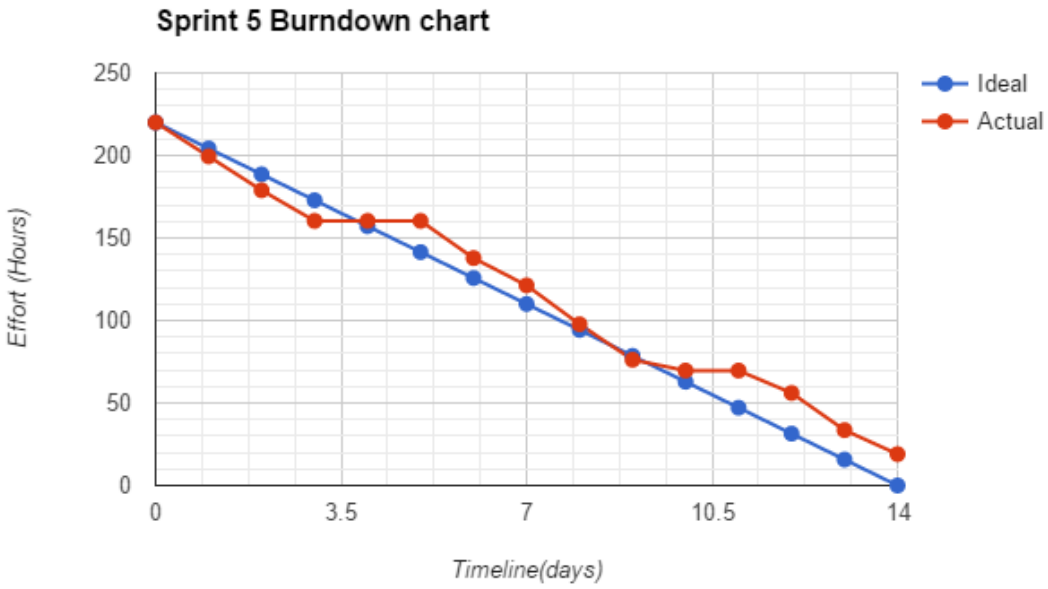


Figure D.3: Burndown chart sprint 5

Sprint 6		
Task	Time estimated	Time spent
Documentation	100	91
Rendering and handling photos	10	28
Testing	25	28
Merging photo and observation report	13	37
Login/autologout functions	30	24
Dynamic status window	10	22
FInalize working prototype	10	8
Reoganise structure (mvc)	5	10
Bugfixing	20	-
Total	223	248

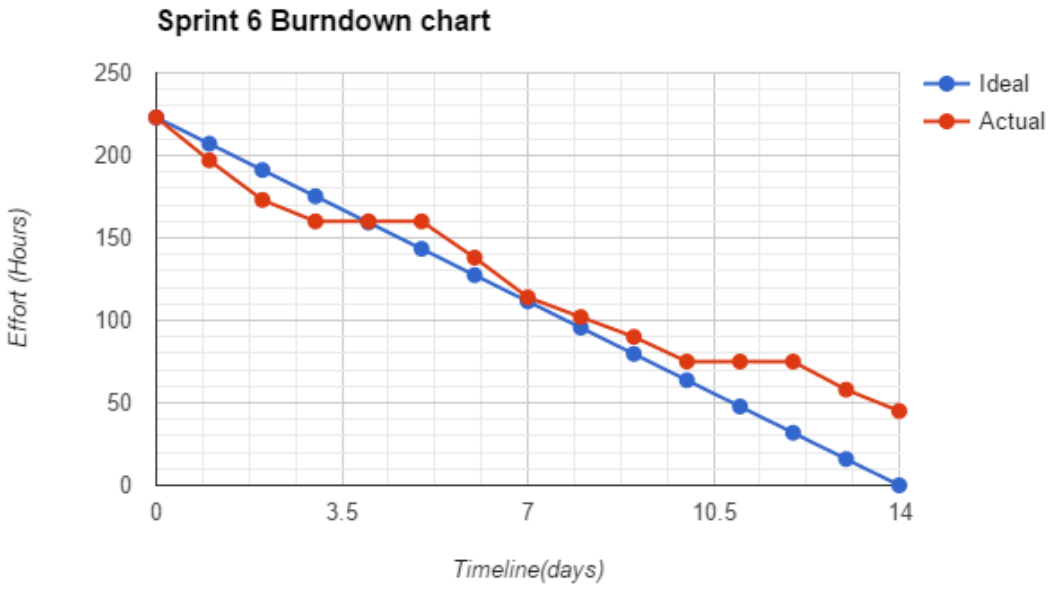


Figure D.4: Burndown chart sprint 6

Sprint 7		
Task	Time estimated	Time spent
Documentation	120	115
Implement remaining functions	50	60
Polish App UI	40	34
User Manual	20	17
Total	230	226

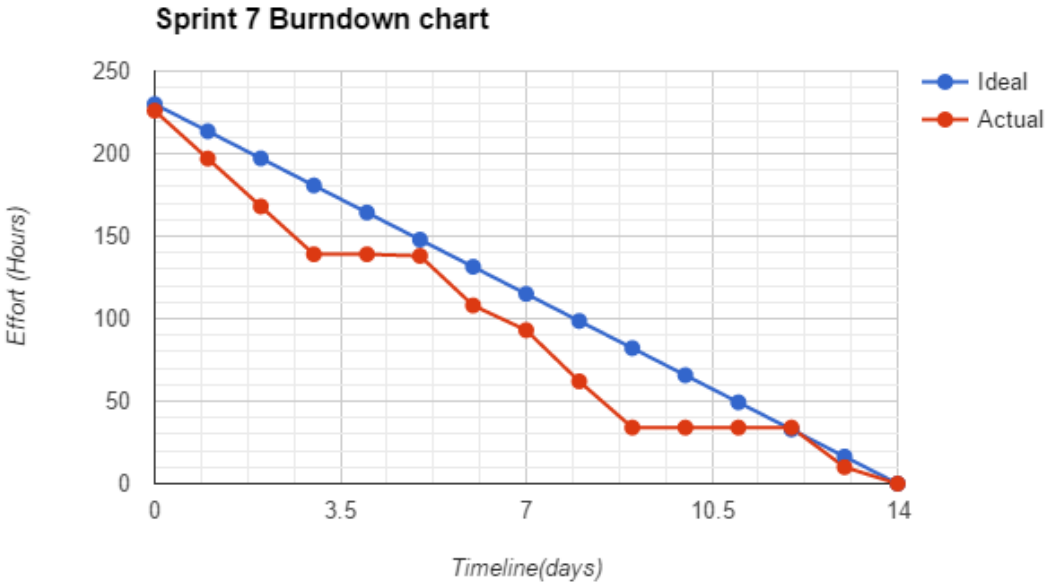


Figure D.5: Burndown chart sprint 7

Sprint 8		
Task	Time estimated	Time spent
Bugfixing	50	46
Create final release	20	18
Finish Writing the report	230	226
Proof reading and last changes	24	25
Group evealuation	9	9
Individual evaluation	9	9
Total	342	333

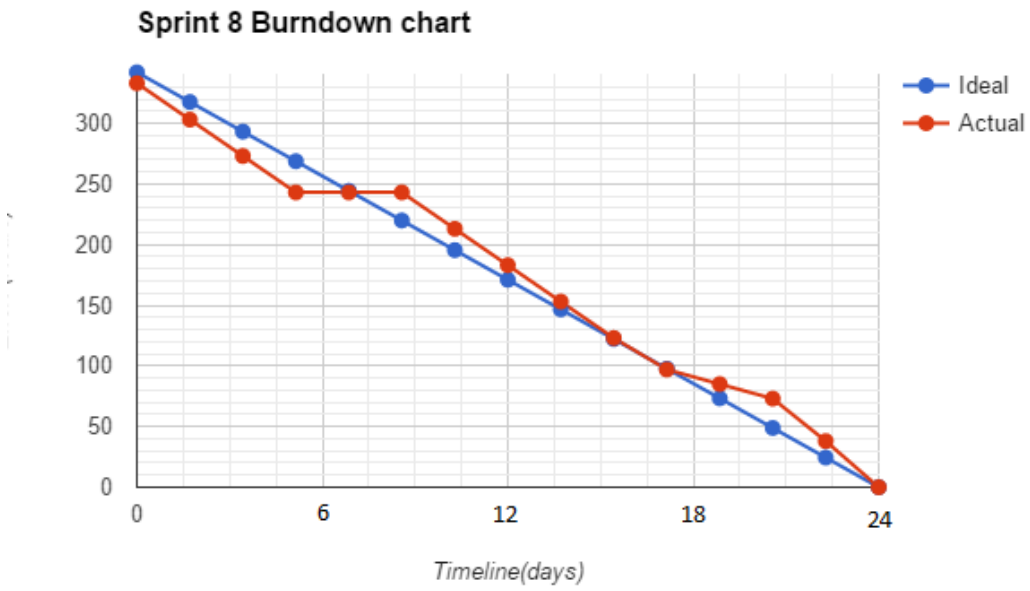


Figure D.6: Burndown chart sprint 8

Appendix E

User and Developer manuals

Situation Awareness and Incident Reporting

User Manual

NTNU 2015

Table of Content

1 Introduction

This is the user manual for the Situation Awareness android application. It contains information on how to install, and run the application. It also presents screenshots with complementary explanations of how each screen operates, and instructions on how to perform specific actions.

2 Install Instructions

2.3 Install and run

2.3.1 Requirements

Device using Android OS 3.0 or newer.
A computer that the device can be connected to.

2.3.2 Installing instructions

First we need to allow installation of apk files from unknown sources on the device:

1. Go into settings menu on your device.
2. Click on the Security tab.
3. Under Device Administration check the Unknown sources box.

Then we can install the app.

1. Connect the device to your computer, and open the device with your file explorer (On windows the device can be found under my computer).
2. move our app .apk file to a folder of your choice on to your device.
3. locate the apk file on your device using the file browser on the device.
4. run the .apk file.

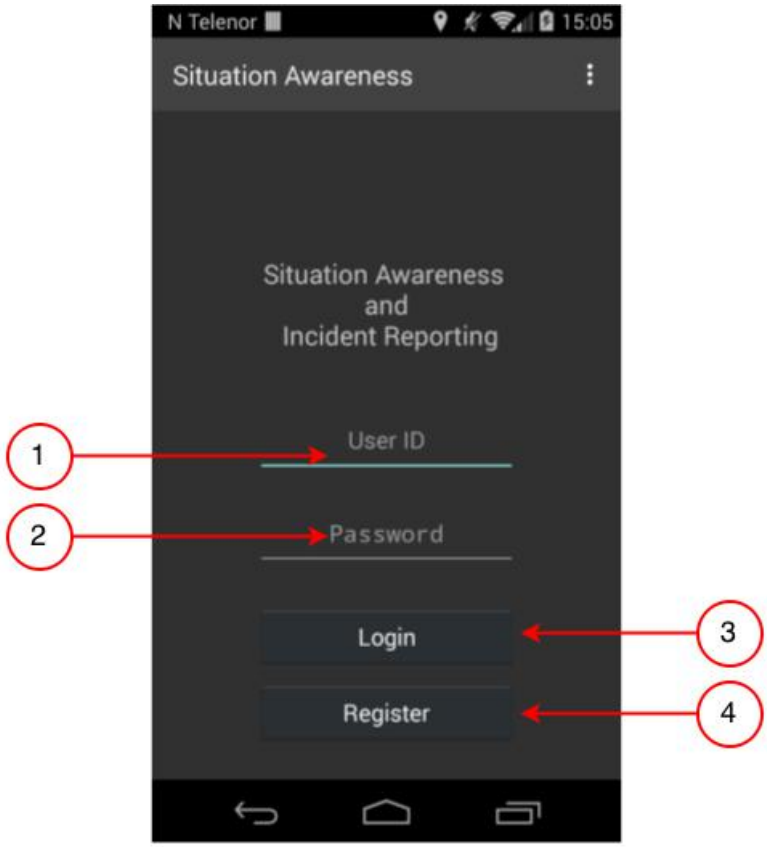
Note! This uses the default online server. To use your own local server, look at the developer manual for more information.

3 Screen Overview

This section contains illustrations of all the screens utilized in the application and how they function.

3.1 Login Screen

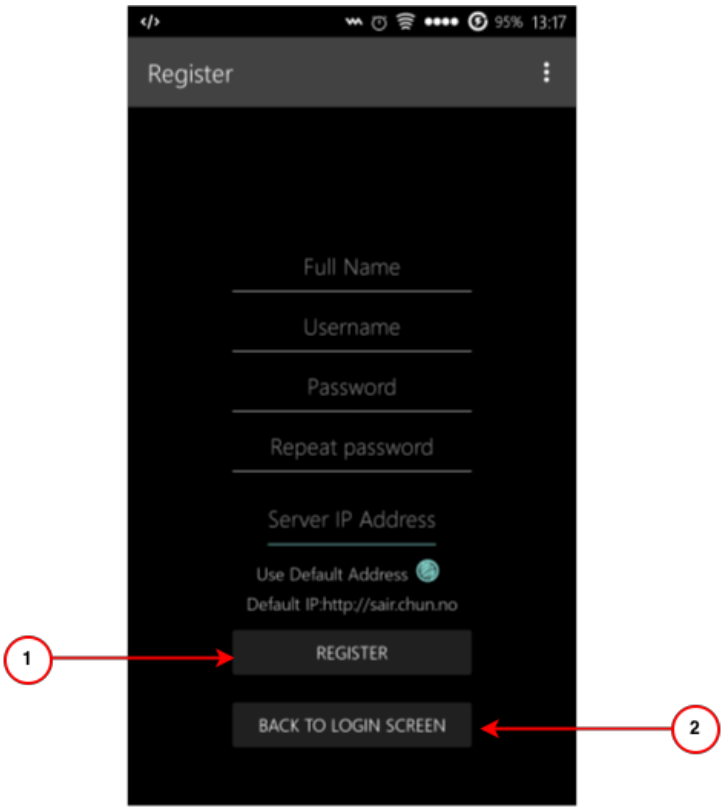
Description: The first screen that is shown when the application is run. Allows the user to login into the main application.



1) Click the field to input a valid user id.
2) Click the field to input the corresponding password that was registered with the user id.
3) Login to the application as long as the credentials are verified. A popup will display a success message if the username and password is recognized.
4) Registers a new user.

3.2 Register Screen

Description: Allows a user to register with a new user id and password. The input fields are fairly standard. To register a full name, usernam and password is required. The username is the id that will be used to login to the application(3.1). The aditional field, server IP address, allows the user to connect to a custom server.

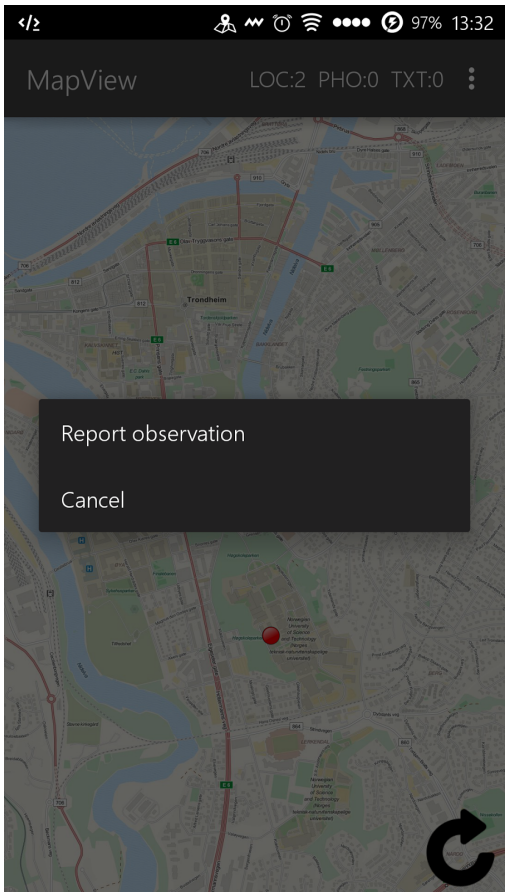


1) Click to save the information and register as a new user as long as the input is valid.
2) Click to return back to the main login window(3.1)

3.3 Map Screen

Description: The main map screen displays all relevant information such as, own, team, and observation locations. It includes touch based functionality common in many maps. The functions include:

- Scrolling the map can be achieved by swiping the finger up,down,left or right. The map will scroll in the opposite direction of the swipe.
- To zoom the map perform a pinching motion or press the zoom butons at the bottom of the map view.
- It is also possible to report an observation directly through the use of the map. By clicking and holding at a specific point on the map, a pop up window(shown below) will open with options for reporting an observation at the given location.



- The positions on the map can also be interacted with by touching the corresponding icons. The map legend that follows explains this in more detail.

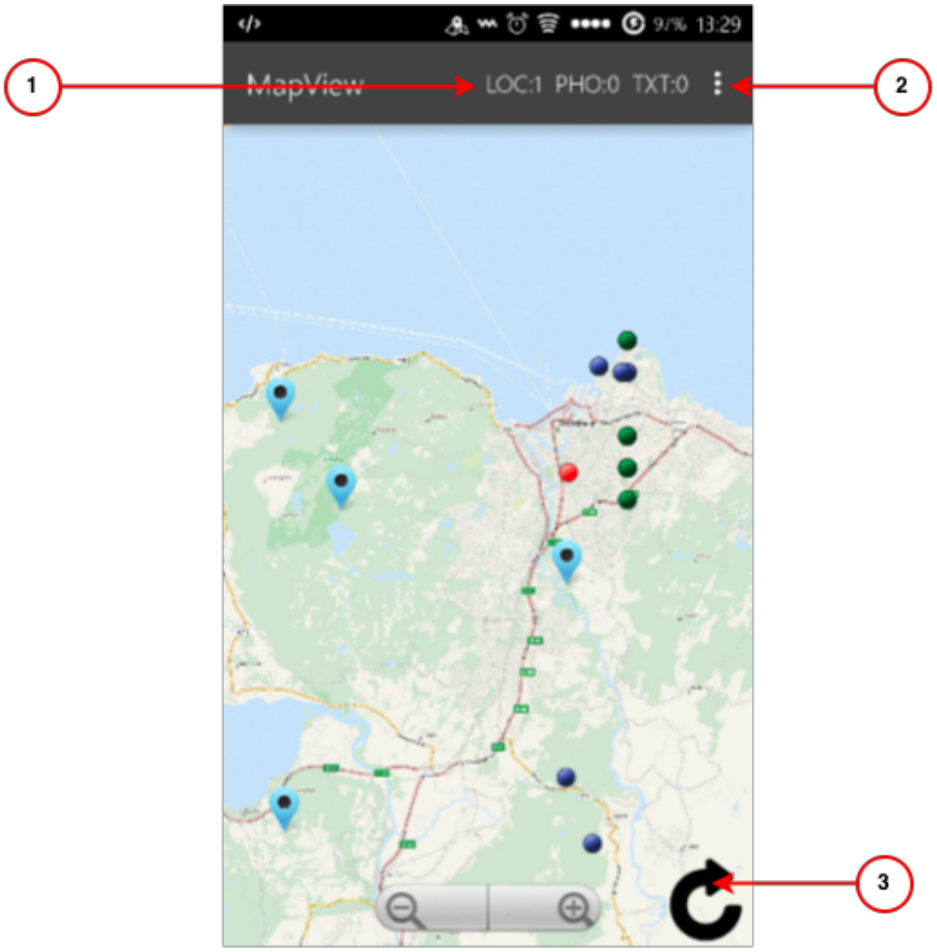
Map Legend

This icon is the user location. By clicking on this, a pop up window will display the current GPS coordinates of the device.

This icon is the location of other teams. By clicking on this it is possible to view the GPS coordinates of the selected team.

This icon represents text reports. By clicking on this, a pop up window will display the GPS coordinates of the report as well as the content.

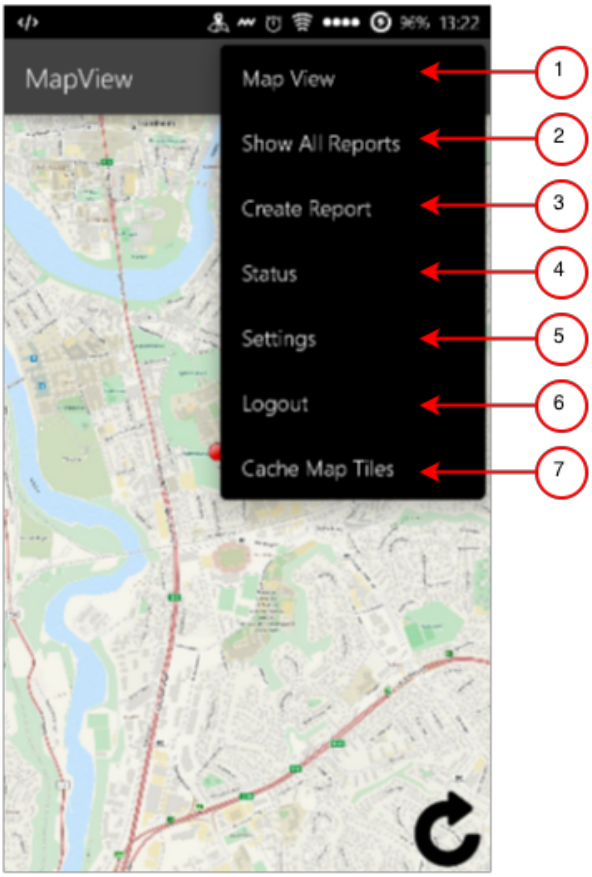
This icon represent photo reports. By clicking on this, a pop up window will display the GPS coordinates of the report as well as the content.



1) Displays number of current unsent reports
   a) LOC: number of unsent location reports
   b) PHO: number of unsent photo reports
   c) TXT: number of unsent text reports
2) Click to access the drop down menu in 3.4
3) Click to refresh the map, updating user and team locations, as well as, new observations.

3.4 Drop down Menu

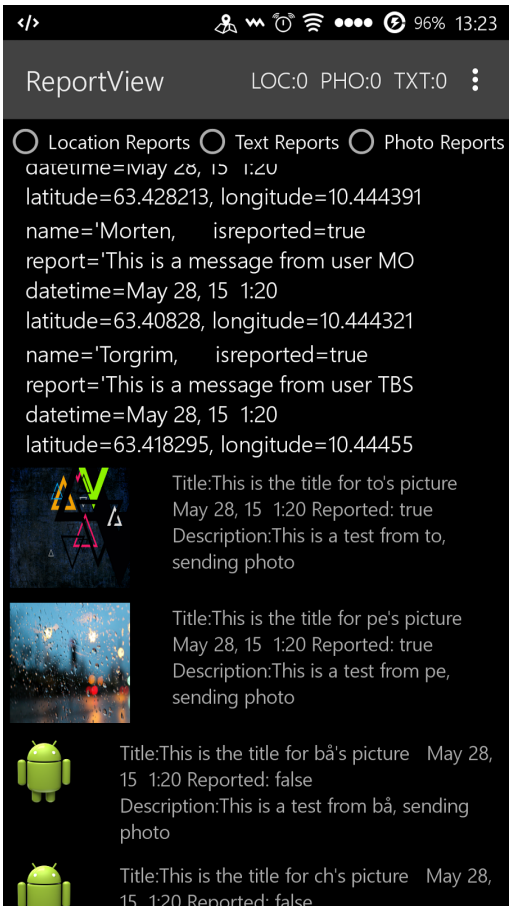
Description: The drop down menu is used navigate the application.‘



1) Click to open the map screen(3.3).
2) Click to open the report view screen(3.5) to display a history of all observations
3) Click to open the report screen(3.7) to register a new observation
4) Click to open the status window (3.8)
5) Click to open the settings screen(3.7)
6) Click to logout of the application and return to login screen(3.1).
7) Click to download the newest map tiles.

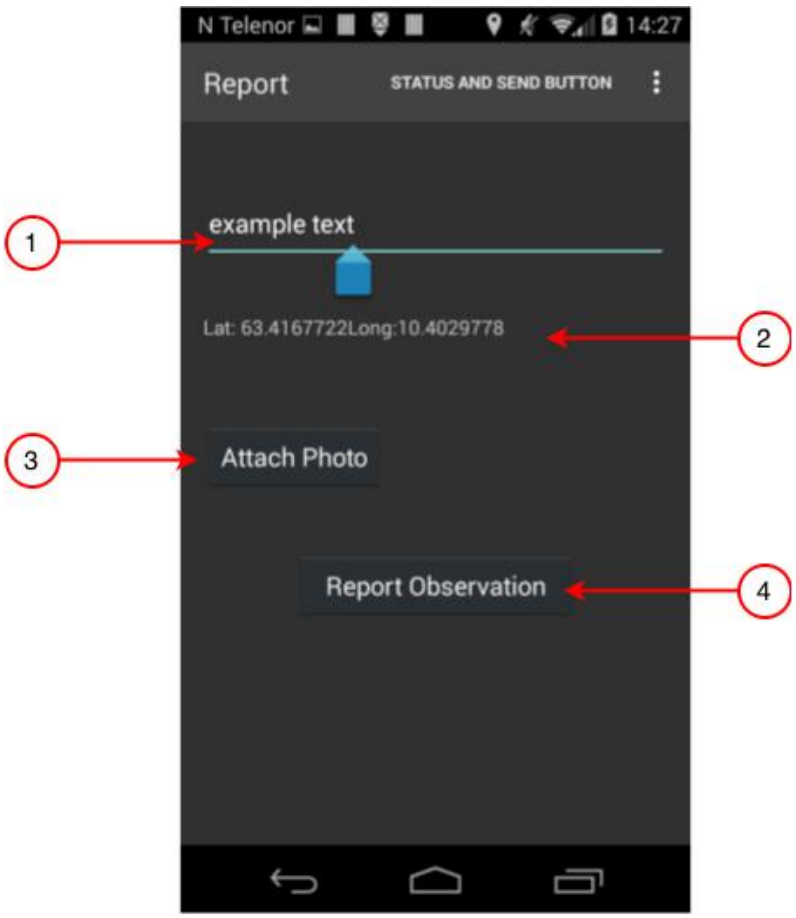
3.5 Report View Screen

Description: The report view contains a history of all the textual reports sent from the device. This view is implemented to keep track of reports that were sent. The report view can be accessed from the “Show All Reports” tab in figure 3.4



3.6 Report Screen

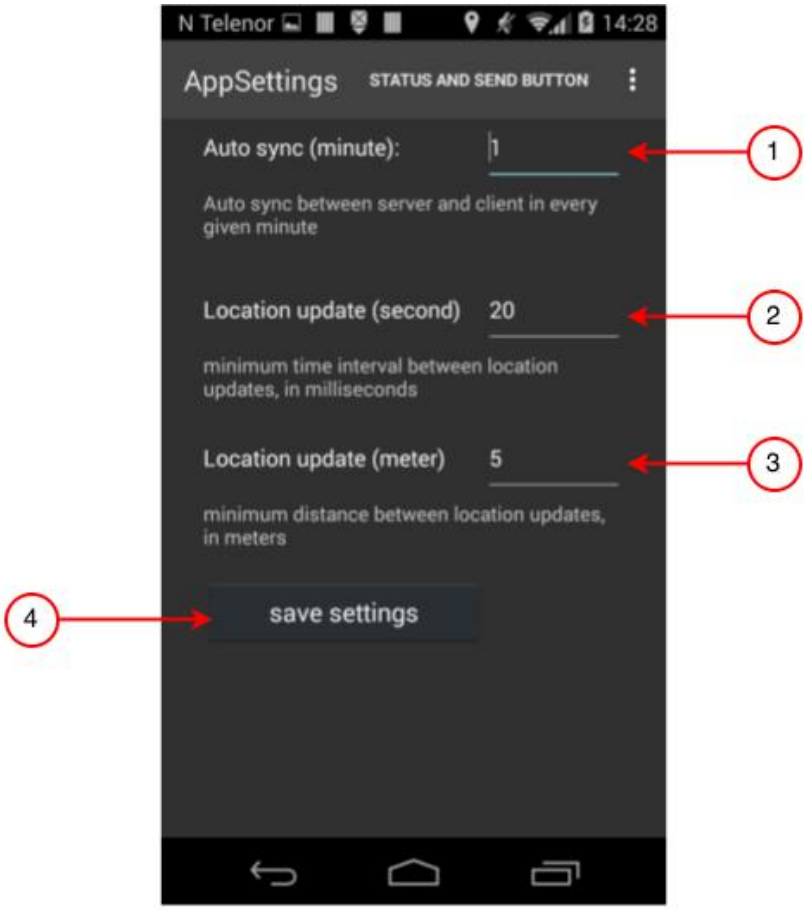
Description: The report screen is used for registering an observation from the current location of the device. If there is no network connection while trying to report an observation, the observation will be stored and sent next time the device gains a connection.



1) Input the textual description of the report by clicking the field.
2) Displays the current GPS coordinates of the device
3) Click to attach a photo to the report. This will open a popup window with some additional actions. Here a picture currently on the phone can be selected, or a new one can be taken and added to the observation.
4) Click to report the observation and send it to the server.

3.7 Settings Screen

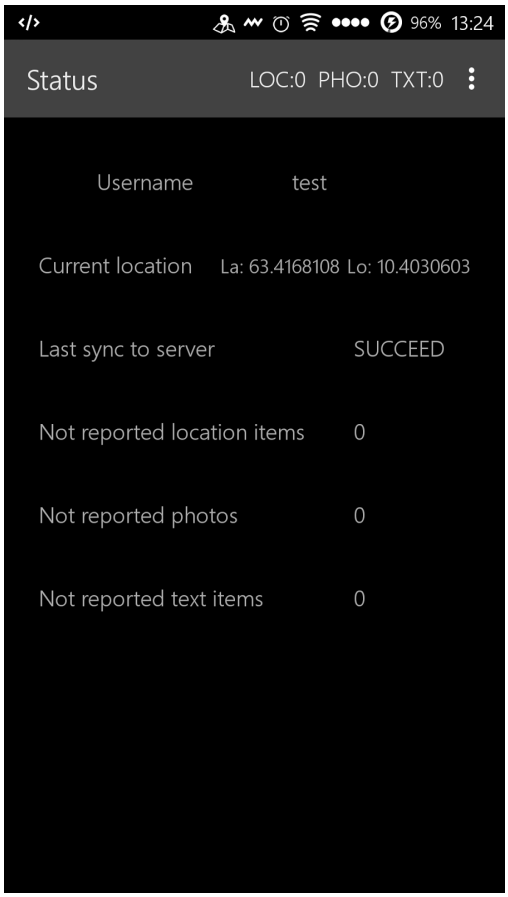
Description: The settings screen is used for changing various settings with respect to update times. It allows for reduction in battery consumption by reducing the activity of the application though less frequent syncing with the server.



1) Click the field to alter the interval at which the client should sync data with the server.
2) Click the field to alter the interval at which the client should report the devices current
3) Click the field to alter the desired movement distance at which the device should update its location.
4) Click to save the current settings.

3.8 Status Screen

Description: The status screen shows the current location of the device, as well as, an overview of how many items have yet to be sent to the server.



Situation Awareness
Developer Manual

NTNU 2015

Table of Content

1 Introduction

This is the developer manual for the Situation Awareness android application. It contains information on how to configure and compile the application.

2 Requirements

Intellij (Commercial edition) or Eclipse (J2EE edition).
Tomcat
Android Studio
Device using Android OS 3.0 or newer.

3 Install Instructions

3.1 Compile

3.1.1 Compiling instructions for server

1. Download the zip file and extract it to the desired location.
2. Open Intellij or Eclipse and select File-->Import from the top menu.
3. Choose import project from external model and choose maven below.
4. Click next and choose a directory where you want to set the root directory.
5. Click next until finish.
6. In the menu bar, select Build -> Build Artifacts -> sair:war

3.1.2 Compiling instructions for client

1. Download the zip file and extract it to the desired location.
2. Open Android studio and select File-->Open from the top menu.
3. Navigate to the location where the folder was extracted and open build.gradle in FFI_Android-->Client--->SituationAwareness.
4. Hit next on the following screen and wait for the gradle to build.
5. Run the application by clicking the green play icon at the top.
6. If you have an android device already plugged into the computer you can choose to run the application on this, if not choose the emulator.

3.2 Install and run

3.2.1 Installing instructions for server

1. Go to the directory sair/target/, copy the sair.war to the tomcat directory tomcat/webapp/, and rename it as ROOT.war. Also you need to remove the previous ROOT directory and ROOT.war.
2. Go to the directory tomcat/bin/, run the startup.sh in Linux or Mac, or run the startup.bat in Windows.

3.2.2 Installing instructions for client

First we need to allow installation of apk files from unknown sources on the device:

1. Go into the device settings.
2. Click on the Security tab.
3. Under Device Administration check the Unknown sources box.

Now to install the app.

1. Connect the device to your computer, and open the device with your file explorer (On windows the device can be found under my computer).
2. move our app .apk file to a folder of your choice on to your device.
3. locate the apk file on your device using the file browser on the device.
4. run the .apk file.

4 Configuration

4.1 Configure the server

Go to the package sair -> src -> main -> java -> edu -> ntnu -> sair -> util, and modify constant.java.

1. Key for encryption and decryption. This is the key for encryption and decryption by using AES.
   protected static final String AES_KEY = "01FFIAndroid2015";

2. Timezone for time. This is the setting for Timezone.
   public static final TimeZone TIME_ZONE =
   TimeZone.getTimeZone("Europe/Oslo");

3. Time format. This set the format for the time.
   public static final String TIME_FORMAT = "yyyy-MM-dd'T'HH:mm:ss.SSSZ";

4. Login period. This set the valid login period. After the period the use is set to be logout automatically. The unit is hour.
   public static final int LOGIN_PERIOD = 48;

5. Path of photos. This set the path for storing the photos of the photo reports. Set it wherever you want.
   //Linux or Mac path
   public static final String PHOTO_PATH = "/home/ProjectFFI/photos/";
   // Windows path
   //public static final String PHOTO_PATH = "c:/ProjectFFI/photos/";

6. public static final int TRUE = 1;

7. public static final int FALSE = 0;

4.2 Configure the client

Go to the package SituationAwareness -> app -> src -> main -> java -> ffiandroid -> situationawareness -> util, and modify constant.java.

1. Key for encryption and decryption. This is the key for encryption and decryption by using AES.
   public static final String AES_KEY = "01FFIAndroid2015";

2. URL of the server. This is the URL for the server. If you set your own server locally, you can set it for the local IP, i.e. http://192.168.1.101:8080. Note, the protocol "http://" is necessary and if you are using tomcat without change its configuration then the port "8080" is necessary.
   // Public Server
   public static final String SERVICE_URL = "http://sair.chun.no";

3. Timezone for time. This is the setting for Timezone.
   public static final TimeZone TIME_ZONE =
   TimeZone.getTimeZone("Europe/Oslo");

4. Time format. This set the format for the time.
   public static final String TIME_FORMAT = "yyyy-MM-dd'T'HH:mm:ss.SSSZ";

5. Timeout in ms. This set the timeout for the connection to the server. If the time is longer than the timeout, it will disconnect from the server.
   public static final int TIMEOUT = 360000;

4.3 Advanced Configuration

The product supports SOAP/REST and GZIP. The following configuration is about configuring for these things.

4.3.1 Set SOAP with GZIP

By default, the configuration is set to be SOAP with GZIP.

4.3.2 Set SOAP without GZIP

1. Use the original source code which is SOAP with GZIP.
2. In the server source code, go to the package sair -> src -> main -> java -> edu -> ntnu -> sair -> service -> impl.
3. Open files ReportServiceImpl.java, RequestServiceImpl.java and UserServcieImpl.java. Remove @GZIP(force = true, threshold = 0) in each file.
4. In the client source code, go to the package SituationAwareness -> app -> src -> main -> java -> ffiandroid -> situationawareness -> model -> util.
5. Open file HttpTransport.java, uncomment the commented code, and remove the GZIP part.

6. Open file Sender.java and remove the code below
   HeaderProperty headerProperty;
   headerProperty = new HeaderProperty("Accept-Encoding", "gzip");
   headers.add(headerProperty);
   headerProperty = new HeaderProperty("Content-Encoding", "gzip");
   headers.add(headerProperty);
7. Recompile everything and reinstall.

4.3.3 Set REST with GZIP

1. Use the original source code which is SOAP with GZIP.
2. In the server source code, nothing to change.
3. In the client source code, go to the package SituationAwareness -> app -> src -> main -> java -> ffiandroid -> situationawareness -> datahandling.
4. Open all files and replace all “new SoapReportService()” with “new RestReportService()”.
5. Recompile everything and reinstall.

4.3.4 Set REST without GZIP

1. Do the configuration as setting REST with GZIP.
2. In the server source code, go to the package sair -> src -> main -> webapp -> WEB-INF
3. Open file SpringMVC-servlet.xml, and remove the code below
   ```
   <filter>
     <filter-name>gzipFilter</filter-name>
     <filter-class>edu.ntnu.sair.filter.GZIPFilter</filter-class>
   </filter>
   <filter-mapping>
     <filter-name>gzipFilter</filter-name>
     <url-pattern>/REST/*</url-pattern>
   </filter-mapping>
   ```
4. In the client source code, go to the package SituationAwareness -> app -> src -> main -> java -> ffiandroid -> situationawareness -> model -> util.
5. Open file Sender.java and remove the code below
   ```
   // Add the gzip Accept-Encoding and Content-Encoding headers
   HttpHeaders requestHeaders = new HttpHeaders();
   requestHeaders.setAcceptEncoding(ContentCodingType.GZIP);
   requestHeaders.setContentEncoding(ContentCodingType.GZIP);
   // Support GZIP
   restTemplate.getMessageConverters().add(new StringHttpMessageConverter());
   ```
6. Recompile everything and reinstall.

Appendix F

Testing

F.1 Login Failure

Title	Login Failure
Description	The user wants to login to the application
Preconditions	The application is running and the user is logged out
Sequence of steps	1.Enter incorrect username 2.Enter valid password 3.Click the login button
Expected Results	The login does not succeed and the user is asked to re enter valid credentials
Pass/Fail	Pass
Comment	Tested with variations of invalid username or passwords

Table F.1: Login Failure

F.2 Login Failure(No network)

Title	Login Failure(No network)
Description	The user wants to login to the application
Preconditions	The application is running and the user is logged out. No network connection
Sequence of steps	1.Enter correct username 2.Enter valid password 3.Click the login button
Expected Results	The login does not succeed and the user is asked to connect to a network
Pass/Fail	Pass
Comment	

Table F.2: Login Failure(No network)

F.3 Photo Report, With Connection

Title	Send Photo Report
Description	The user wants to send a photo observation
Preconditions	The user is logged in and on the report screen. Network connection
Sequence of steps	1. Click the attach photo button 2. Take a photo 3. Click report observation button
Expected Results	The report is sent to the server
Pass/Fail	Pass
Comment	Was also tested with choosing a picture from gallery

Table F.3: Photo Report, With Connection

F.4 Photo Report, No Connection

Title	Send Photo Report
Description	The user wants to send a photo observation
Preconditions	The user is logged in and on the report screen. No network connection
Sequence of steps	1. Click the attach photo button 2. Take a photo 3. Click report observation button 4. Connect device to network
Expected Results	The report is stored in the local database, then sent to server when the device is reconnected
Pass/Fail	Pass
Comment	Was also tested with choosing a picture from gallery

Table F.4: Photo Report, No Connection

F.5 Test data of MySQL and H2

Time requesting 100000 TextReports (ms)	MySQL Server	H2 Embedded
1	75653	15636
2	71545	13448
3	71431	12441
4	70769	13347
5	72630	15602
6	70380	13216
7	68896	13412
8	71283	14056
9	71769	15219
10	70911	12898
Average	71526.7	13927.5

Table F.5: Time requesting 100000 TextReports (ms)

F.6 Test data of time requesting 1 text report

Time requesting 1 text report (ms)	SOAP with GZIP	SOAP without GZIP	REST with GZIP	REST without GZIP
1	43	50	132	169
2	43	49	152	164
3	41	32	132	139
4	78	43	130	129
5	58	53	120	134
6	36	46	141	148
7	52	53	152	146
8	36	33	143	161
9	42	40	136	131
10	44	61	136	158
11	32	37	116	125
12	24	47	142	144
13	57	43	124	163
14	32	53	108	149
15	44	37	125	128
16	40	45	121	176
17	41	56	108	138
18	37	41	110	128
19	61	38	113	126
20	44	56	114	139
Average	44.25	45.6	127.75	144.75

Table F.6: Time requesting 1 text report (ms)

F.7 Test data of time reporting 1 text report

Results for web service performance test

Time reporting 1 text report (ms)	SOAP with GZIP	SOAP without GZIP	REST with GZIP	REST without GZIP
1	49	44	113	100
2	35	35	104	105
3	41	31	104	105
4	41	38	122	100
5	42	36	97	100
6	40	55	116	117
7	38	35	83	103
8	39	23	115	95
9	54	41	102	111
10	40	45	124	94
11	52	39	110	110
12	43	38	101	120
13	48	32	104	109
14	66	32	130	107
15	45	35	87	97
16	30	44	116	84
17	45	40	101	84
18	45	38	105	99
19	34	48	127	96
20	55	34	90	84
Average	44.1	38.15	107.55	100.4

Table F.7: Time reporting 1 text report (ms)

Appendix G

Future works

If we would have more time and resources to improve our project we would have liked to see these improvements in our application:

- Improved cluster function so that the application scales better when there are large amounts of information, including thousands of observations
- If there are many observation at the same point, find a way to easier show them all to the user
- More thorough testing of the application for better efficiency, possible bugs, and especially to be able to lower the energy consumption of the device so that the battery life is extended
- Some better formatting on some of the views to make them more user friendly
- Ability to import your own maps
- Add relations between observations and a possibility to edit old observations
- Info window with relevant information showing up when clicking on an observation either on the pop-ups in the map, or in the list in reportView
- Center screen when app opens in the middle, and not the top left corner
- A button to center on your own position
- Hide or delete your own old location reports from the reportView list.
- Improve UI for different screen sizes, and both horizontal and vertical orientations.
- The status of reports is not updated properly, needs to be improved
- Possibilities to have multiple teams where you would only get the relevant information for your own team
- Possibility to reports with things like time, location etc.
- An admin interface

Bibliography

[1] Android Studio. http://developer.android.com/tools/studio/index.html. Accessed: 2015-08-02.

[2] Apache CFX. http://cxf.apache.org/. Accessed: 2015-15-02.

[3] Apache Tomcat. http://tomcat.apache.org/. Accessed: 2015-15-02.

[4] Base64. http://en.wikipedia.org/wiki/Base64. Accessed: 2015-11-03.

[5] FFI. http://www.ffi.no/en/Sider/default.aspx. Accessed: 2015-28-01.

[6] GZIP. http://www.gzip.org/. Accessed: 2015-10-03.

[7] H2. http//www.h2database.com/. Accessed: 2015-15-02.

[8] ksoap2. https://code.google.com/p/ksoap2-android/. Accessed: 2015-17-02.

[9] MySQL. https://www.mysql.com. Accessed: 2015-15-02.

[10] OpenStreetMap. https://www.openstreetmap.org/about. Accessed: 2015-22-02.

[11] OSMdroid. https://osmdroid.net/. Accessed: 2015-22-02.

[12] REST. http://www.service-architecture.com/articles/web-services/representational_state_transfer_rest.html. Accessed: 2015-16-02.

[13] Saving Data in SQL Databases. http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html. Accessed: 2015-30-03.

[14] SOAP. http://en.wikipedia.org/wiki/SOAP. Accessed: 2015-15-02.

[15] Spring Framework. http://projects.spring.io/spring-framework/. Accessed: 2015-16-02.

[16] SQLite. https://www.sqlite.org/. Accessed: 2015-15-02.

[17] United States Department of Defense. Department of Defense Mobile Device Strategy. http://www.defense.gov/news/dodmobilitystrategy.pdf.