University of Southern Denmark

Software Engineering of Mobile Systems

Group M

Give a Tip

Authors:
Simon H. Larsen
Thomas A. Lemqvist
Veronika Zemanova

Supervisor:
Mikkel B. Kjærgaard

Author emails: {sila114, thlem14, vezem17}@student.sdu.dk

December 21, 2017

SDU

Abstract

This report documents the implementation of a prototype application, which builds upon the already existing application 'Giv et tip'. The purpose of the existing application is to crowdsource the reporting of issues, which the municipality parks department is responsible for fixing. The prototype application focuses on some of the problems of the existing application, which includes validation of reports, prioritization, and giving users an incentive to use the application.
A battery test was conducted on the prototype to determine the viability of the application. As the app will need precise location data continuous in the background the battery consumption might be an issue. It showed that the application will be responsible of increasing the discharge rate by 7%.
The prototype solves the validation of reports by making end users validate each other's reports. The validation is done by pushing an Android notification, asking them to confirm the reported issue, whenever they pass near the geographic location of the report.

Keywords: *Mobile Sensing, Gamification, crowd sourcing of data, crowd sourcing of validation for data, Application development.*

Contents

1 Introduction

This section introduces the motivation and context of the project, it describes the problem and the objective of the project, and shortly outlines the rest of the report.

1.1 Motivation and Context

This project has covered case number three, which works with the application 'Giv et Tip' [1]. The application is used by citizens of Odense Municipality, to report issues around the city, which the parks department is responsible for fixing. These issues include everything from potholes, to overfilled trash cans.

The objective for this case is to use mobile sensing and gamification to foster crowd sourcing of information, and keep people returning to the platform. Mobile sensing is used in the existing application, to add extra data to the reports, such as a GPS location, to give the reports a precise geographic location, and to add images to the reports from the phone camera.

Gamification is needed in the current application, because there is no incentive to return to the platform, and users only use the application to report issues. By having a gamification system in place, the user will gain an incentive to return to the platform. This is done by adding game-like elements, such as progression, where users increase in rank and unlocks new features or content.

1.2 Problem and Objective

This project has worked with three problems with the current application, firstly validation of reports, which is how the municipality validates that the report is actually something they have to react upon, and fix. Secondly prioritization of reports, which is how the municipality prioritize the reported issues, and in which order they fix the reported issues. Lastly, the current application does not have an incentive to keep users on the platform, or gain new users, other than it being a easier way of reporting issues.

The objective for this project is to produce a prototype application, which will solve the report validation problem, by crowdsourcing the validations from the end users. The project will furthermore attempt to prioritize the reports, based on activity around reports. As well as as introducing some gamification elements to the platform, in order to incentivise new users, and to keep current users active on the platform.

1.3 Report Outline

The report is structured as following: section 2 describes the problems in greater detail. Section 3 will describe the solution to the problem on a high level. Section 4 will analyze the problem domain. Section 5 will build upon the analysis and describe the design decisions made. Section 6 will describe the implementation of the solution. Section 7 will reflect upon the achieved results, and discuss them. Section 8 will conclude the project and specify the directions for future works.

2 Problem Description

This section will analyze and describe the problems, which was introduced in the introduction section, into greater detail.

The validation of reports is a problem, as we assume that municipality workers have to spend time reading through reports, and validate them either based on submitted images, or travel to the location of the report to personally validate it. This process seems very time consuming and like a waste of municipality resources.

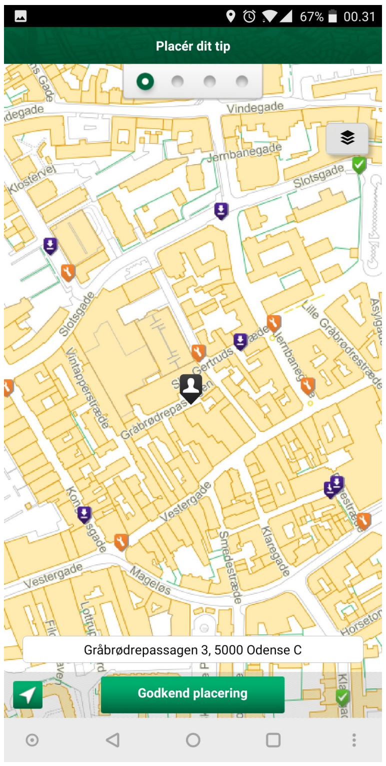


Figure 1: Image of the map screen in the current application.

In the current application, as Figure 1 shows, it is possible to see where other reports has been created, but there seems to be no way to get any information about these reports, other than the icons showing the report status.

The municipality also have a website which shows an overview of the current reports [2], but again, it does not offer any other functionality, for users to interact with the already reported issues. In order to contribute data to an already reported issue a user has to create a duplicate report with some more (accurate) information. This will eventually lead to more reports that the municipality workers have to validate.

The prioritization of reports is a problem, as we assume that the municipality workers have to dig through multiple reports and prioritize them. This process can cause a potentially dangerous issue, or an issue which affects more citizens, to become buried between the many other reports.

The platform lacks an incentive for the users to use the platform, other than it being an easier way to report issues with. If there is no incentive to use the application, the amount of active users on the platform will decrease over time, which will result in a decrease in the amount of reports the municipality will receive.

This section has described the three problems that the project will try to solve. The solution for these problems will be elaborated upon in the next section.

3 Solution Approach

This section will describe the solution approaches for the problems described in the previous section, which has been taken in this project on a high level.

To reduce the amount of resources needed in order to validate the reports, this project will focus on implementing functionality for crowdsourcing validation of reports.

Whenever an issue report is created, its location will be added to the map, and when users pass by these reports, they will receive a notification, which asks them to confirm or deny that the reported issue is real, and relevant. The user will have the option to add comments to further improve the details of the report, and add new images as well.

This way the users of the application can contribute to the validation of the reports, this could potentially save the municipality a lot of time and resources spent on validating reports.

To help prioritizing the reports, we believe that the amount of activity for a report e.g. the reports with a lot of confirmations, or pictures and comments, will be prioritized highest, as they have more activity and affects more people. Whereas the reports which have been denied by multiple users will not be shown, or at least will be prioritized less.

To create an incentive for using the application, this project proposes a reward system, where users are rewarded an amount of points, each time they interact with reports on the system. It could work like the following: if a user creates an issue report, he will receive 10 points, if he confirms or denies a report he will receive 2 points, and if he adds more information by commenting or adding an image to an existing report, he will get 5 points.

The point system can be used in different solutions, and this report describes three different ways of using it.

Leaderboard: in this solution the platform will use the points to compare the users of the platform with each other. This could create a competitive spirit, and motivate users rise to the top of the leaderboard.

Lottery: in this solution the platform will use the points as lottery tickets for a lottery of prizes provided by local businesses or the municipality. The more points a user has, the more lottery tickets, and chance he has to win a prize. This could motivate users with the chance of winning free items.

Charity: in this solution the platform will use the points as votes, which the users can pledge to a charity organization or event. The municipality or local businesses will then donate money to a pool, which is either goes fully to the majority vote, or is split between the charities, according to their share of the votes.

Some of these gamification solutions require a donation from the municipality or local businesses. We believe that these will be willing to take part, in order to keep users on the platform, which could lead to a more issue free city.

This section has introduced the solution at a high level, it has introduced the report validation feature, the prioritization feature, and the three different solutions for the gamification element.

4 Analysis

This section will analyze the problem domain of our solution, in order to be able to come up with an optimal implementation. We will analyze the current application, present the actors of the system, define the use cases which have been chosen for the prototype, present the requirements specified for the system, and finally, describe the domain model for the application.

4.1 Actors

This section will present the actors of the system. There are four actors of the prototype, the unregistered user, the registered user, location, and Server.

Unregistered user: is a user who is not registered in the system i.e. he does not have an account.

Registered user: also known as 'user', is a user, who has an account, and is an active user on the platform.

Location: is the location of the registered user. This actor is responsible for triggering events when the registered user is near a report.

Server: is the server which serves the data to the android application, and accesses the database.

These actors will be used in the following section in the use cases, which presents the features of the system.

4.2 Use Cases

With the actors from the previous section, we created use cases in order to figure out what services our prototype application should offer.

UC1 Register user

UC2 Login

UC3 Create report

UC4 Change settings

UC5 View Report

UC6 View Leaderboard

UC7 Validate report

UC8 Comment on report

UC9 Display report notification

UC10 Logout

These use cases can be seen in the use case diagram in Figure 2. The use case diagram shows the relation between actors and use cases, and shows what actors have access to which use cases.

4.2.1 Use Case Descriptions

This section will describe the use case UC9 into greater detail. For descriptions of all use cases look in Appendix A.

Table 1 shows the description of UC9. It shows the main flow with its alternative flows, which ends the flow. The reason for the alternative flows to end the use case, is that this is supposed to happen in the background, and the user should not be interrupted, unless there actually is something to notify him about.

This section has introduced the use cases, which has been worked with within this project. These use cases will be used in the following section to lay basis for some of the requirements to the system.

4.3 Requirements

This section defines the requirements set up for the prototype system. These requirements has been based upon the use cases from the previous sections, and based on the development knowledge within the team.

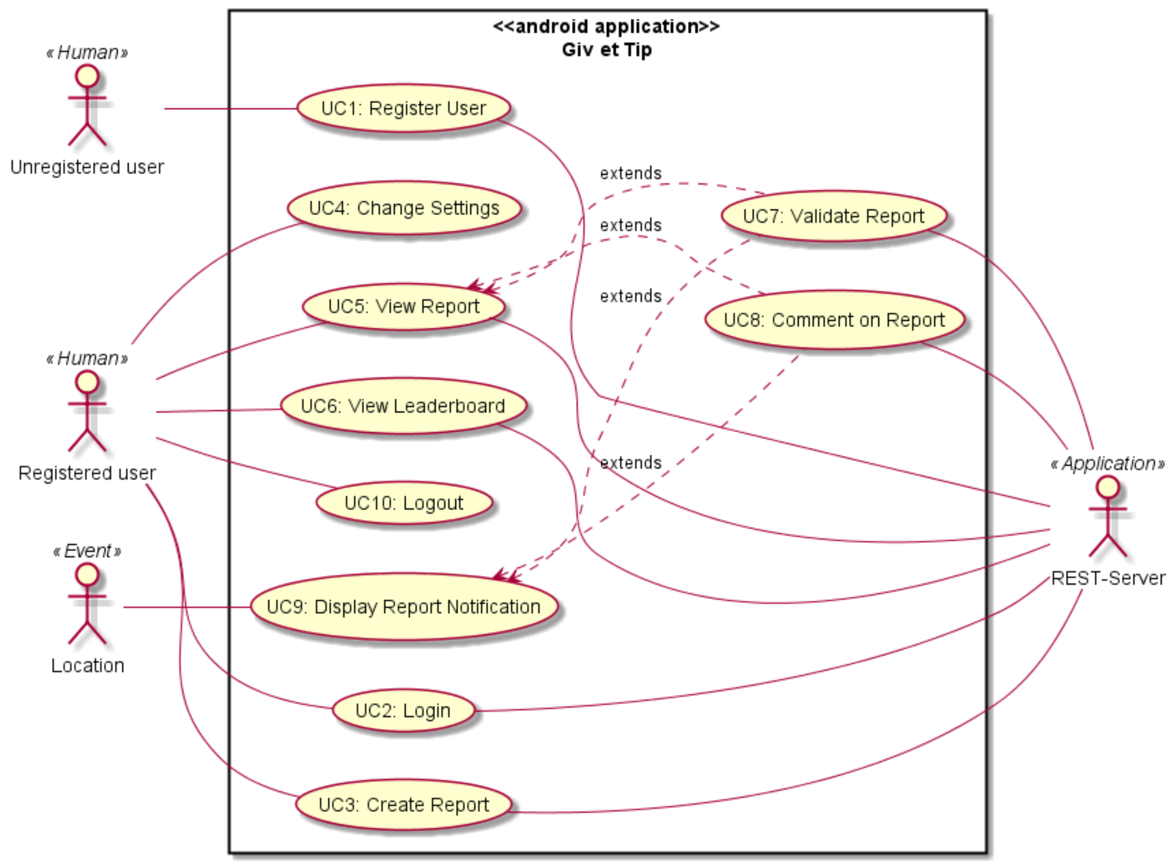


Figure 2: Use case diagram for the system

The requirements has been divided into the Functional and Nonfunctional requirements.

4.3.1 Functional Requirements

The functional requirements are the requirements which defines the functionality of the system. The functional requirements can be seen in the list below.

F1 Unregistered users should be able to register as users.

F2 The user should be able to login to their account.

F3 The user should be able to logout of their account.

F4 The user should be able to create a report.

F5 The user should be able to change the location of the report.

F6 The user should be able to change the distance threshold to reports, for notifications.

F7 The user should be able to view reports inside the application.

F8 The user should be able to view the user points leaderboard.

Table 1: Use case description for UC9.

Name	Display Report Notification
Id	UC9
Description	Allows the application to send notifications to the user, if he is within his set distance threshold of a report.
Primary Actors	Location
Secondary Actors	NA
Preconditions	The user has location services enabled.
Main Flow	1. The distance from the user's current location is within the set threshold 2. The application checks if the user has already contributed to this report. (a) Alternative flow: Alternative Flow: User already contributed to report: 3. The application checks if the user has been shown this report before (a) Alternative Flow: Report already shown. 4. The application gets the latest information about the report. (a) Alternative Flow: No connection to server. 5. The user is asked to validate the report. 6. end.
Postconditions	A notification is shown to the user, and he can respond to ut with UC7 or UC8.
Alternative Flows	User already contributed to report: 1. end. Report already shown: 1. end. No connection to Server: 1. end.

F9 The user should be able to validate a report bu voting up or down for it.

F10 The user should be able to comment on reports.

F11 The system should notify the user, whenever he is within the distance threshold of a report, and ask for a validation.

4.3.2 Nonfunctional Requirements

The nonfunctional requirements has been divided into what parts of the system they belong to, e.g. the requirements for the server is in the server category, and the requirements for the mobile application is in the mobile application category. The categories and requirements can be seen in the lists below.

Server

NS1 The Server should be RESTful, in order to serve both the application and a web client.

NS2 The Server should be able to run on Debian linux.

NS3 The server should be developed in java.

NS4 The server should have a connection to a MySQL database.

Mobile application

NA1 The application should be developed with minimum SDK 23.

NA2 The application should use as little powers as possible.

NA3 The application should contain a user friendly UI.

NA4 The application should show the reports on a map.

NA5 The application should check the distance to nearby reports in the background.

NA6 The application should hash all passwords, using the SHA-256 algorithm.

The key requirements, which is necessary in order to solve the problems, which this project focuses on, are the following:

NA5, without this the application will not be able to ask the users to verify reports, **F10** is directly dependent on this requirement.

F8 and **F9**, without these functional requirements ensures that there will be features for verifying the reports.

4.4 Domain Model

The domain model is a conceptual model of our platform. It identifies objects or entities in the real world which will be a part of the system.

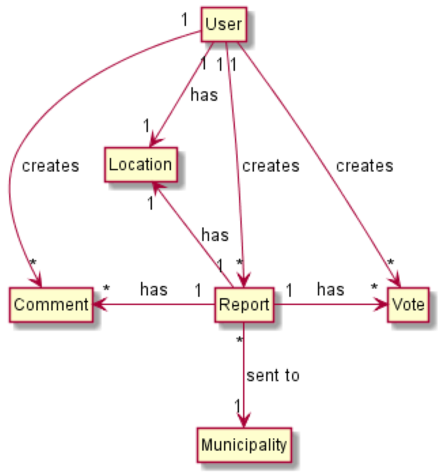


Figure 3: The Domain model for the Giv et Tip prototype

Figure 3 shows a simple domain model for our prototype, we can see the user can create reports, comments and votes, and has a location. We can see the report has comments, votes and a location, and that the report is sent to the municipality, which is represented by the Server actor.

In this analysis section the actors of the system have been identified, the use cases, which has been used throughout the development phase of the prototype system, have been discovered, the requirements to the system has been established, and the domain model have been presented.

5 Design

This section will describe the design decisions, which was made throughout the project. It will start by describing the model of the system with a class diagram, then the architecture will be described. It also introduces some quality attribute scenarios, and android device tactics. And finally it will describe the design of the UI, and the application flows.

5.1 Class Diagram

Extending upon the domain model in figure 3 a class diagram has been made as shown in figure 4. The domain model is in "customer language" while the class model is a Software Engineering discipline.

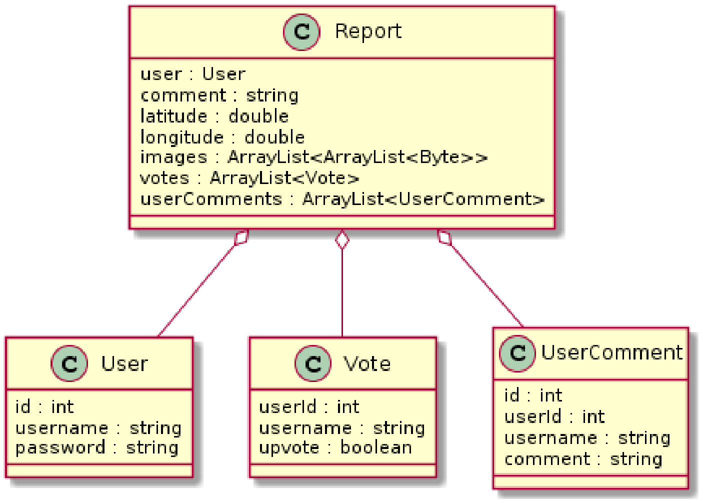


Figure 4: Class diagram, showing the model of the system.

The class diagram in figure 4 shows the fields of each class and each of their types. The *Report* class aggregates a *User*, a list of *Vote*, and a list of *UserComment* to support the demands for reports having comments and votes as described in the domain model **??**. The domain model also describes that a report belongs to a location, this is covered by the fields latitude and longitude.

5.2 Architecture of prototype diagram

5.2.1 Deployment Diagram

The deployment of the system requires a specific set of computation units. The required units have been mapped the deployment diagram figure 5.

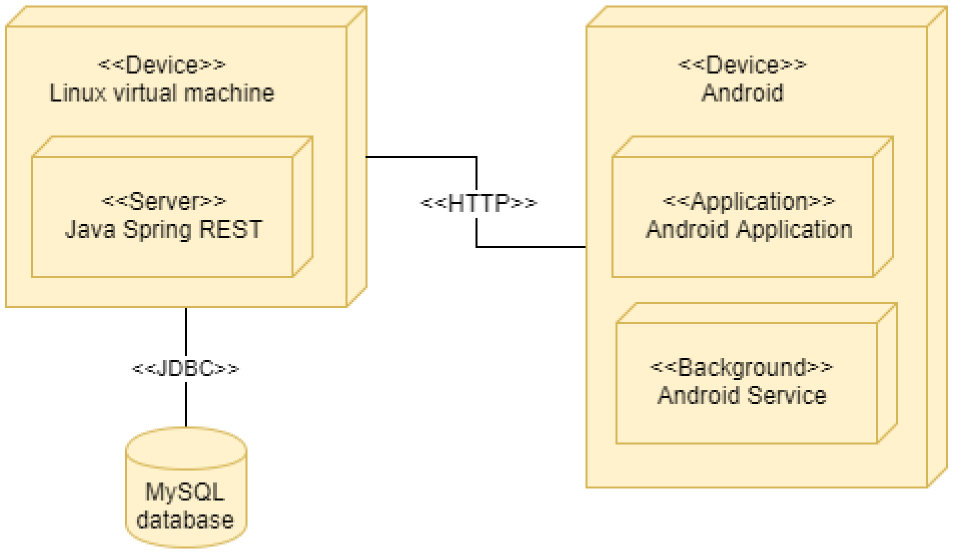


Figure 5: Deployment diagram of the system.

Two computational units is the minimum required units to run the system. A Linux virtual machine to host the backend REST service - including a MySQL server to achieve persistence - and at least one Android device. The Android device will act as a client that will consume the REST service provided by the Linux virtual machine. The Android client will be running a continuous background process and will have a frontend app. Both of which will be communicating with the backend through HTTP. The communication protocol between the client and the backend will be HTTP as the server will be a REST service. The communication between the MySQL database and the REST service will be JDBC since this is very well supported in Java.

5.3 Quality Attribute Scenarios

This section will introduce some quality attribute scenarios, which will be implemented in order to increase the quality of the system. The quality attribute scenarios in this section focuses on mobile sensing, because of that we only include the quality attributes Energy Efficiency and Resource Adaptability [3].

5.3.1 Energy Efficiency

Table 2: QAS I

Source of Stimulus	System
Stimulus	Activity recognized as walking, checking if within a geofence
Environment	Not within geofence
Artifact	Client Application
Response	Decrease location service precision.
Response Measure	Maximum 1% battery drain per hour, caused by this process.

Table 3: QAS II

Source of Stimulus	System
Stimulus	Activity recognised as walking, checking if within a geofence
Environment	within geofence
Artifact	Client Application
Response	Increase location service precision.
Response Measure	Maximum 5% battery drain per hour, caused by this process.

Table 4: QAS III

Source of Stimulus	System
Stimulus	Battery charge below 15 %
Environment	Normal operations
Artifact	Client application
Response	Disable direct location service, to only receive updates from other services location requests.
Response Measure	Maximum <1% battery drain per hour, caused by this process.

5.3.2 Resource Adaptability

Table 5: QAS IV

Source of Stimulus	User
Stimulus	User reports an issue, without client application having permission for location data.
Environment	No location data available.
Artifact	Client application
Response	User is asked to permit location data, or submit a nearby address for reff
Response Measure	increased the reporting time by maximum 2 minutes.

5.4 Android Device Tactics

To address architectural design challenges mentioned above in Quality Attribute Scenarios and to support non-functional requirements, we propose using several architectural, energy efficiency and resource adaptability tactics.

5.4.1 QAS I and II

Energy efficiency quality attribute scenario to either increase or decrease location service precision, based on user activity recognition and checking whether user is inside certain geofence radius, can be satisfied by using ***Event-based*** tactic. This tactic schedules energy consumption of location service when a threshold or raw sensor data indicates an event. The application, continuously checking user activity, will trigger an event when detected activity is recognized as ON FOOT. This, in turn, will start method to check reports proximity to user. If the application discovers any reports nearby, the location request precision will be increased. If there are no reports, the precision will be lowered.

5.4.2 QAS III

Disabling direct location service, to trigger only updates from other applications once the battery charge is below 15% can be done through setting PRIORITY_NO_POWER for setPriority(int) of location request. By setting this, this will greatly limit the functionality of the application, but it will still take advantage of location when available. The tactic used for this is called ***Sensor Selection***. It balances quality of service delivered by sensor with energy cost.

5.4.3 QAS IV

Resource availability tactic ***Increase resources*** can be used for our last QAS table. To avoid delivering a degraded quality of service, the tactic increases number of available resources. In our case, we can consider user input as additional resource. When the application fails to deliver location, be it any location or just a correct one, user will have an option to manually submit location data. This could be done by writing down the exact location or referencing nearby address/building/structure for reference.

Additionally, to have even more efficient application we can consider using tactics outside of our quality attribute scenarios. Continuous location sensing while the service is active consumes a lot of energy. To improve optimization, we can implement ***Dynamic Duty Cycle*** by setting radius for geofence - in code, by user input through frontend settings, or both. By adjusting this parameter we can lower, or increase if that is needed, the boundary around reports that will trigger method to increase location service precision.

To continue with the thought that location sensing has high demands on energy, we can use activity recognition service as a trigger for starting location service in chain reaction with tactic named ***Sensor Replacement***.

Communication Selection tactic works parallel with Event-based tactic mentioned at the start of this subsection. By reducing the location service precision when the user is not within proximity of any report, the application selects the least energy-consuming communication option to transfer data between requester and provider. The same can be said about the tactic used for QAS III.

Selecting resources for delivering location updates with the highest quality – in this case we want the latest known user location, helps us deliver augmented service. ***Resource Selection*** can make use of two methods of Location Request: setInterval(long millis) and setFastestInterval(long millis). By fine-tuning both, we can benefit both from the application's and other applications location updates.

5.5 User Interface Design

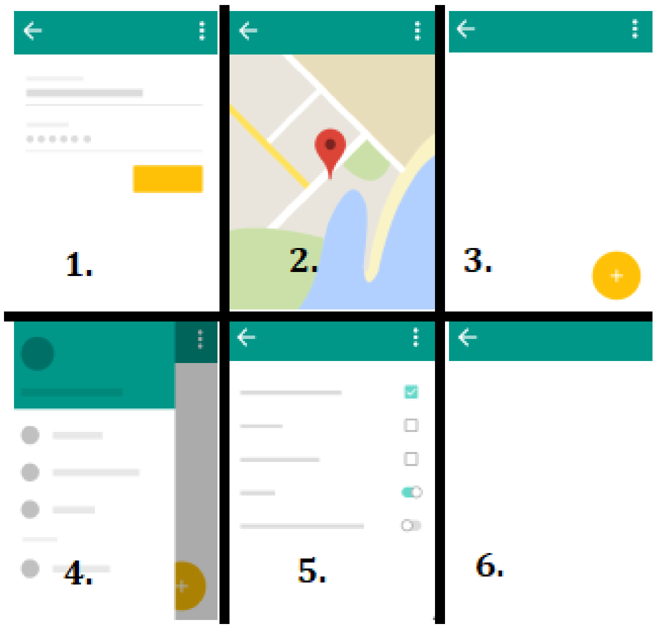


Figure 6: Activity Mockups for the application.

The first prototype of design we will draw inspiration from is figure 6.

After user logs in (1), he will be redirected to an activity with Google maps (2). There, he will see his location and reports noted by other users. He can then create his own report (3). The user can navigate with the help of sidebar navigation (4) any time. To adjust the application, user can change settings (5). Finally, some view of user statistics or gamification idea could be displayed in the last activity (6).

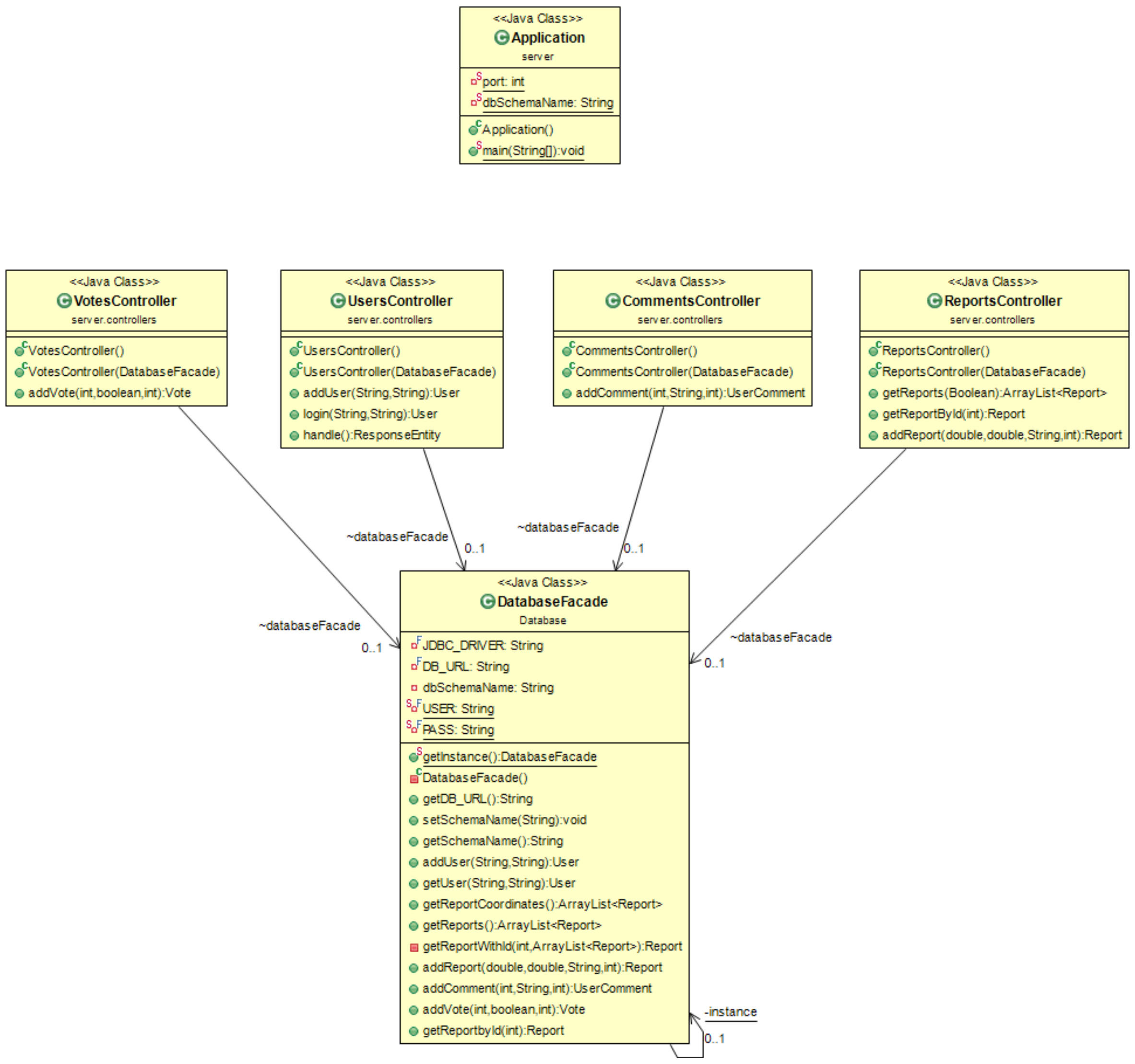
The requirements of the project have been established and the preliminary software design has been created. The desired quality attribute scenarios and Android device tactics have been defined.

6 Implementation

This section will describe the implementation of the backend as well as the frontend. Implementation diagrams, code examples, and technical descriptions will be used.

6.1 Java Spring REST Backend

To support the system functionality a Java RESTful service was developed using Spring. The reasoning behind using a REST backend is that the service can be consumed by any client that supports HTTP. Thus allowing for a web client as well as an android application to communicate with the backend. The endpoints for the service was implemented in accordance to the implementation class diagram 7.



<<Java Class>>
Application
server
port: int
dbSchemaName: String
Application()
main(String[]):void
<<Java Class>>
VotesController
server.controllers
VotesController()
VotesController(DatabaseFacade)
addVote(int,boolean,int):Vote
<<Java Class>>
UsersController
server.controllers
UsersController()
UsersController(DatabaseFacade)
addUser(String,String):User
login(String,String):User
handle():ResponseEntity
<<Java Class>>
CommentsController
server.controllers
CommentsController()
CommentsController(DatabaseFacade)
addComment(int,String,int):UserComment
<<Java Class>>
ReportsController
server.controllers
ReportsController()
ReportsController(DatabaseFacade)
getReports(Boolean):ArrayList<Report>
getReportById(int):Report
addReport(double,double,String,int):Report
~databaseFacade
0..1
~databaseFacade
0..1
~databaseFacade
0..1
~databaseFacade
0..1
<<Java Class>>
DatabaseFacade
Database
JDBC_DRIVER: String
DB_URL: String
dbSchemaName: String
USER: String
PASS: String
getInstance():DatabaseFacade
DatabaseFacade()
getDB_URL():String
setSchemaName(String):void
getSchemaName():String
addUser(String,String):User
getUser(String,String):User
getReportCoordinates():ArrayList<Report>
getReports():ArrayList<Report>
getReportWithId(int,ArrayList<Report>):Report
addReport(double,double,String,int):Report
addComment(int,String,int):UserComment
addVote(int,boolean,int):Vote
getReportbyId(int):Report
-instance
0..1

Figure 7: Server side class diagram.

Figure 7 shows the classes implemented in the backend. All the controller have two constructors; an empty constructor and a constructor that takes a DatabaseFacade class as a parameter. This is used to enable unit testing of the REST controllers by allowing dependency injection to mock the database facade. The database facade contains the logic for communicating with a MySQL database using JDBC thus achieving persistence for the application.

The VotesController lets users add an up- or downvote to reports.

The UsersController lets unregistered users sign up and lets registered users login.

The CommentsController lets users add comments to reports.

The ReportsController lets users get all reports, get a report by id, and add a new report.

Code Snippet 1: The ReportsController in Java Spring REST

```java
@RestController
@RequestMapping(value = "/api")
public class ReportsController {

    @RequestMapping(value = "/reports", params = {"only-coordinates"})
    public ArrayList<Report> getReports(@RequestParam(value="only-coordinates")
    Boolean onlyCoordinates) {
        if (onlyCoordinates) {
            return databaseFacade.getReportCoordinates();
        } else {
            return databaseFacade.getReports();
        }
    }
}
```

In Spring a REST endpoint is defined by using annotations. Specifically the annotation "*@RestController*" is used to define an endpoint known as a controller. This controller does not default to any URI, but another annotation is used for specifying the endpoint URI. The "*@RequestMapping*" annotation is used for specifying URI's - the specific methods as well as the controller itself can use this annotation. All of the controllers maintain a reference to a database facade which is responsible for making the transactions with MySQL database to maintain persistent data.

6.2 Databse

The REST service is responsible for enabling HTTP access to the backend while the database facade is responsible for communicating with the MySQL database through JDBC. The database schema required to support the application is shown in figure 8.

The database diagram in figure 8 shows how the issue reports are connected to users and how the details of a report is interconnected.

The **Image_report** table contains a primary key *id*, highlighted by underlining the column name, a foreign key *id_report*, highlighted by *italic text*, and an *img_path* column. The *img_path* column is a string containing the relative path to an image file stored on the system.

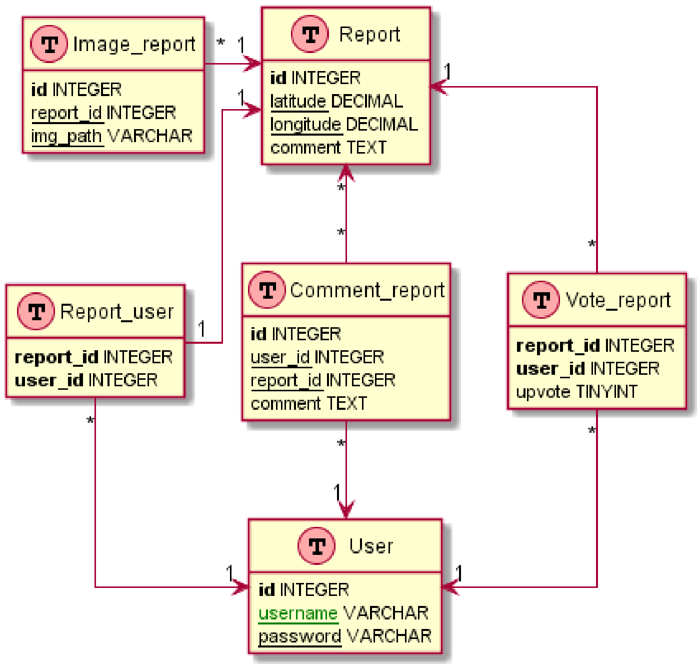


Figure 8: Relational Model that shows the database structure

The **Comment_report** table contains comments on reports submitted after the publishing of a report. The table contains two foreign keys that references the *Report* and *User* tables as well as a comment column.

The **Vote_report** table contains the votes that users have cast on reports. The vote is saved as a TINYINT which is converted to a boolean value in Java. The table also contains two foreign keys - which is both primary keys as it is a weak entity - that references the *Report* and *User* tables.

The **Report_user** table is a weak entity which bind reports to users by using two primary keys which are also foreign keys to the the *Report* and *User* tables.

The **User** table contains the username and password of a user.

The **Report** table contains the reports submitted by the users. The report table itself will contain the *longitude* and *latitude* of a problem as well as a problem description *comment*.

6.3 Android Client

This section depicts how we implemented the android client. It describes which mobile sensing features we implemented and how we applied them into service. Secondly, it lists employed energy efficiency and resource adaptability tactics. Finally, we show user interface and describe some of the application flow with the help of screenshots of the android activities.

6.3.1 Sensing

We utilize two kinds of mobile sensing, activity and location. We use them to share data and inform people.

Activity Sensing with ActivityRecognitionClient - extends GoogleApi Once the user is logged in, the application will start with activity detection inside the TipNotificationService. Six types of user activities – *UNKNOWN*, *IN_VEHICLE*, *ON_BICYCLE*, *ON_FOOT*, *STILL* and *TILTING* - are assessed continuously in the background.

ActivityRecognitionContainer is responsible for categorizing the *ON_FOOT* activity by fusing activities *ON_FOOT*, *RUNNING* and *WALKING*. It also serves as an interface for detected activities, storing type and confidence.

Sensor data processing is done inside TipDistanceHandler class. If the activity is recognized as *ON_FOOT*, the class will determine if it should check for new report updates from the server. The default update interval for checking for new updates is 2 hours. If those 2 hours passed and the person is walking somewhere, the application will automatically download new reports. If the two hours have not passed, and the activity is still *ON_FOOT* , the class will calculate user distance to the reports. If the distance is less than 150 meters, the location service will increase its quality of delivery, precision. Simultaneously, it will compare the distance to the notification radius and if it's less, notification will be triggered.

If the detected activity is any other than *ON_FOOT*, the TipDistanceHandler will attempt to decrease the location precision, to preserve power.

Location sensing with FusedLocationProviderClient - extends GoogleApi TipLocationService is a background service which is started with user login and is stopped when the user logs out. First, googleApiClient is build and location request parameters (interval and priority) are set. The application then checks to see if it has the permissions to access location. If it does have that, the service starts the location updates through callbacks within the class and broadcasts it outside the class.

Location is forwarded to ReportsMapFragment (where Google map is displayed) and to TipNotificationService (to calculate distance between user and reports).

6.3.2 Tactics

The implemented tactics from design section are the following:

- Energy efficiency in sensing: Dynamic Duty Cycle, Sensor Replacement
- Energy efficiency in processing: Event-based
- Energy efficiency in sharing: Communication Selection
- Resource adaptability in resource availability: Resource Selection

To show an example, Figure 9 displays a sequence diagram of an Event-based tactic scenario, where the application changes the location service accuracy based on the users geographic proximity to nearby reports.

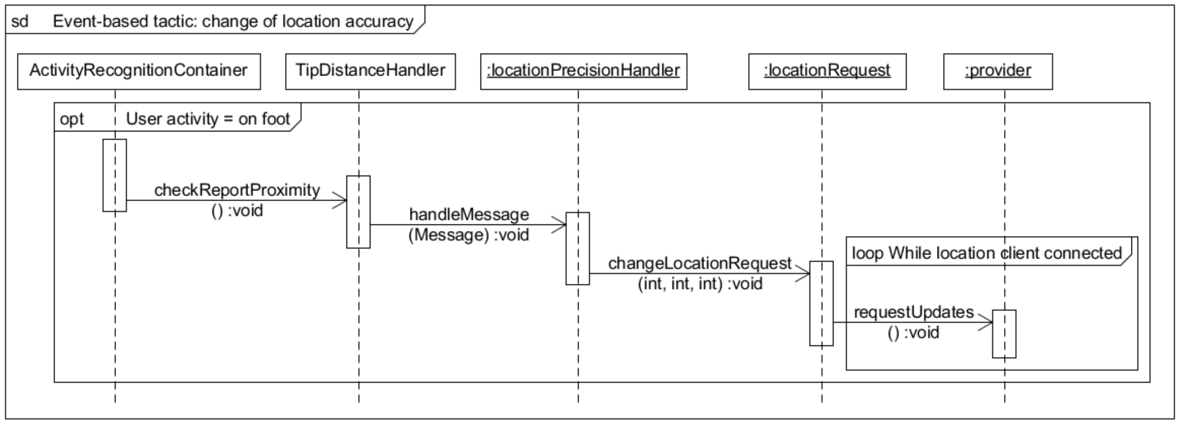

sd Event-based tactic: change of location accuracy
ActivityRecognitionContainer
TipDistanceHandler
:locationPrecisionHandler
:locationRequest
:provider
opt User activity = on foot
checkReportProximity
() :void
handleMessage
(Message) :void
changeLocationRequest
(int, int, int) :void
loop While location client connected
requestUpdates
() :void

Figure 9: Sequence diagram of the Location change Tactic

6.3.3 User Interface

In the implemented User Interface, have built upon the User Interface from subsection 5.5. It now have functionality, handling of notifications of nearby reports, and commenting on reports. The implemented leaderboard is for demonstrative purposes only, and does not have any functionality.

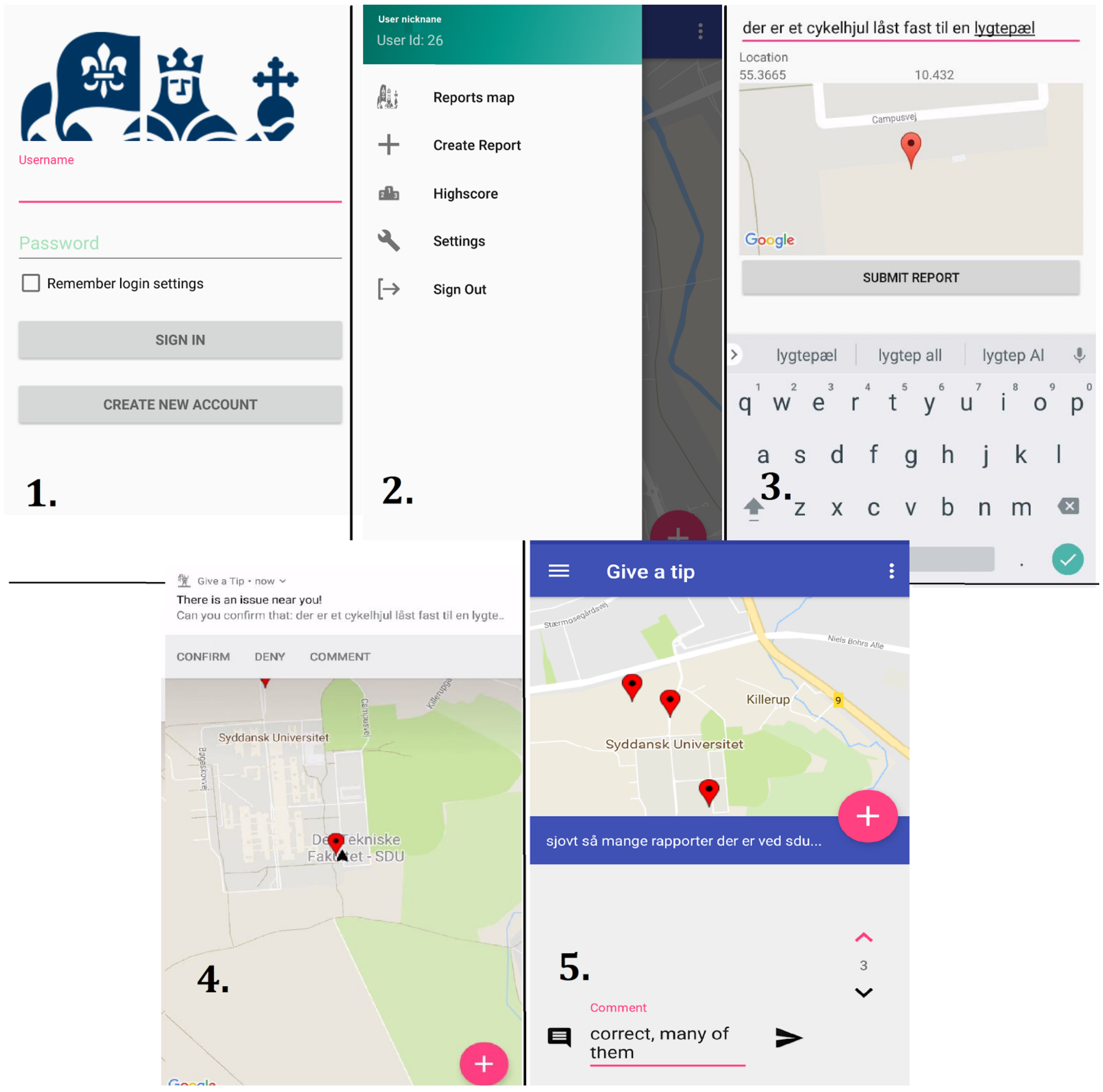
The App Flow

After the non-registered user registers an account, he has the option to log into his account (1). When he logs in, he will be shown the map view (4) with his current location marked by a black triangle, and markers of existing reports will be displayed.

The user can either use intuitive navigation by clicking on things, or sidebar navigation (2).

To create a report, the user has to be in the location he wants to report it at (3). The location will be shown in the map view fragment inside activity, where description is also put before submission.

Once the user walks by an existing report, the phone will vibrate and play a sound, and a notification will be shown (4). There, the user can confirm or deny the report, therefore helping us to validate it, or comment further on it.



der er et cykelhjul låst fast til en lygtepæl

55.3665 10.432

There is an issue near you!
Can you confirm that: der er et cykelhjul låst fast til en lygte...

sjovt så mange rapporter der er ved sdu...

correct, many of them

Figure 10: Screenshots from the final prototype

This section described the technical implementation of the system. A RESTful web service communicating with a MySQL database was developed to support HTTP clients - including the Android frontend. Android client was developed with activity and location sensing features, and with some of the architectural tactics described in design section.

7 Reflection and Discussion

This section will reflect and discuss the reached solution, based upon tests and the results therefrom. It will elaborate on the parts of the problems, which was not solved. And finally reflect upon the project as a whole.

7.1 Not Solved

We wanted to create a web client, which should include a suite of administrative tools, for the municipality workers to use, to select issues to solve. In this web client, we would have solved the prioritization problem. But as we have not created this web client, as we deemed it out of the scope of the course, which focuses on mobile sensing.

Although it has not been implemented, we have made the architecture of our server application ready to serve a web client, and we believe that it would not be of great risk to implement such a client.

The lacking incentive for using the platform, has not been implemented in this prototype application, but it have been addressed, with the three solution suggestions from section X (ref 3). We have analyzed the three solutions and found both pros and cons for all implementations, and found that the lottery solution might promote cheating for personal materialistic gain, and would therefore not pursue that solution. And we found that the charity solution, was a good idea, as it could engage users, but it requires external investors in order to become a successful solution. Therefore, we find that the leaderboard solution is the easiest solution, as it do not require any external investors, and does not promote cheating by materialistic gain, but it might not be the most engaging gamification element. We therefore suggest that a combination of the three would be a great solution.

The reports in the current Giv et Tip application, has categories and images, both of which have not been implemented in this prototype, as it did not add any value to the problem domains. Although the categories could be used for weighing the prioritizations in the web client, so that more demanding categories would be weighed higher that less demanding categories.

7.2 Project Reflection

We have tried to utilize an agile approach to the development of the application, where we started by identifying the use cases, and assigned them to weekly sprints, and used a kanban system on GitHub, to keep track of the development progress. This system has worked out well for us, although we moved away from the weekly sprints, and just implemented the features, as we had time for it.

The project pitch at Odense Castle went well, this project won the "Giv et Tip" project category, and one of the judges, a developer from sweco, mentioned that we had a useful focus, with the crowdsourcing of verification.

The teamwork within the team, has worked out great, there has been great information and knowledge sharing throughout the process of this project.

7.3 Known Issues

Location permissions: There is an issue with the location service, the first time the application is started, which renders the location service useless, as it never updates, unless the application is force-stopped, through the android application settings menu. This issue is caused by the lacking confirmation whether GPS is enabled on the first startup of the application. It could probably be fixed by waiting to start the location service until it is known that GPS is enabled.

Shared preferences as storage: The prototype makes excessive use of the shared preferences as data persistence. This is not optimal, and it would be much better to use SQLite, which is already included in android, for this purpose.

Disabled location services: There is another issue, which renders the location service useless. If the user disables the location services, in the android settings menu, the application will not prompt the user to re-enable it, when in the application is in focus, in order to make it usable again

Download and storage of reports: The background service is responsible for keeping report data up to date, by downloading report coordinates over time. It is currently configured to do this on a two hour interval. It would be nice to have some kind of manual update, which would offer users the option to force update the reports. Furthermore, these report coordinates is the only stored data about the reports. This means that every time the user needs to be shown report details, the application needs to download data for that report, and it is then thrown away afterwards.

Settings: The settings menu in the prototype only offers the option to tune the distance threshold to reports before a notification is shown. It could be great if there was some kinds of settings profiles in stead, which could offer e.g. a power saver mode, a normal mode, and a power hungry mode. These profiles would be easier for an end user to understand the effects of.

Geofences: The prototype does not utilize the android api geo-fences, for figuring out which reports to calculate the distance to, but instead just calculates the distance to each and every report. This means that the application uses unnecessarily much processing power to calculate distances to reports, which are multiple kilometers away.

Increased precision around already notified reports: The prototype do not discriminate between new reports and reports it has already notified the user about, when it comes to calculating the distance to the reports. This means that it increases the GPS sensing precision, which is power consuming, for nothing, as the user will not be notified about the report anyways.

7.4 Discussion

The project aimed to solve the problem of validation of reports, prioritization of reports, and the lacking incentive of using the platform. All three problems have been addressed in this report, but only the validation problem solution has been implemented.

The validation problem was solved by implementing an infrastructure for crowdsourcing the validation. The infrastructure utilizes a background service, which runs continuously on the phone, and checks if there are nearby reports. The service then notifies the user when within 30 meters of a report. This service works as intended, members of the group have been receiving notifications, while not aware that the service ran in the background, about nearby reports. This proves that the service works as intended, as it should not bother the user other than when there are nearby reports.

As for the other two problems, this report has suggested different solutions, which have not been implemented, due to constraints in the scope and time available.

When developing for battery dependent devices like Android it is important to consider the energy efficiency of a solution for the device. A list of QAS and tactics is described in the tactics section 5.4. In order to validate the tactics Battery Historian has been taken into use [4]. This allows for measuring all of the activity on the Android device to be able to determine what is going on while using our app.

While using battery historian a test was conducted. The device used for testing was a OnePlus 5T running Android 7.1.1.

Thomas went out for a walk while having the app running in the background. Google Maps Timeline feature was used to track the path Thomas took while testing the application - it is worth mentioning that Google Maps nor any other location tracking service was running while Thomas was walking. Figure 11 shows the path as well as a red circle, which marks the spot where reports were placed in our app.

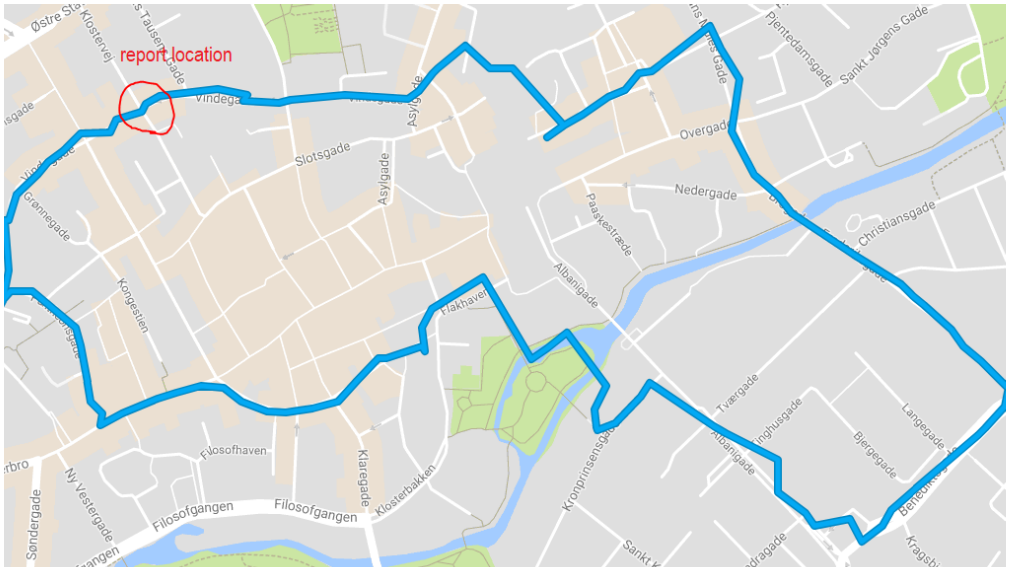


Figure 11: Map of the Thomas walk

Thomas got a notification as soon as he approached the red circle marked on figure 11. He approached the red circle somewhere around 1:27-1:29 PM according to Google Timeline. It is seen in figure 12 that the GPS is turned on at 1:27:08 PM until 1:30:50 PM. This fits very well into the timestamp provided by Google Timeline - this means that the GPS was turned on around the same time as Thomas was near the report.

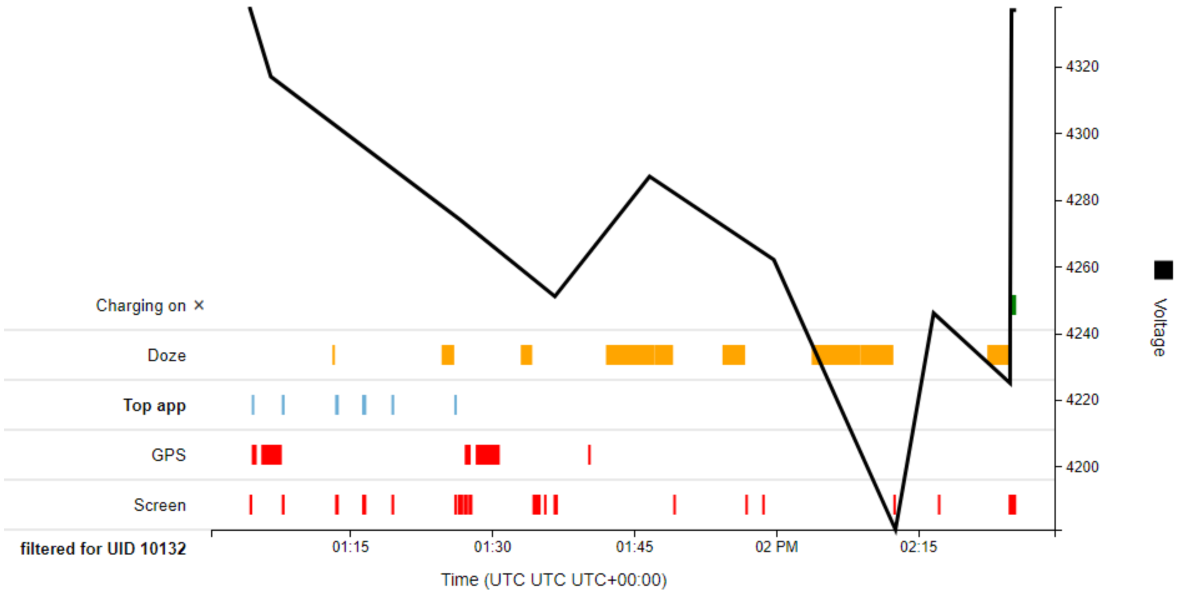
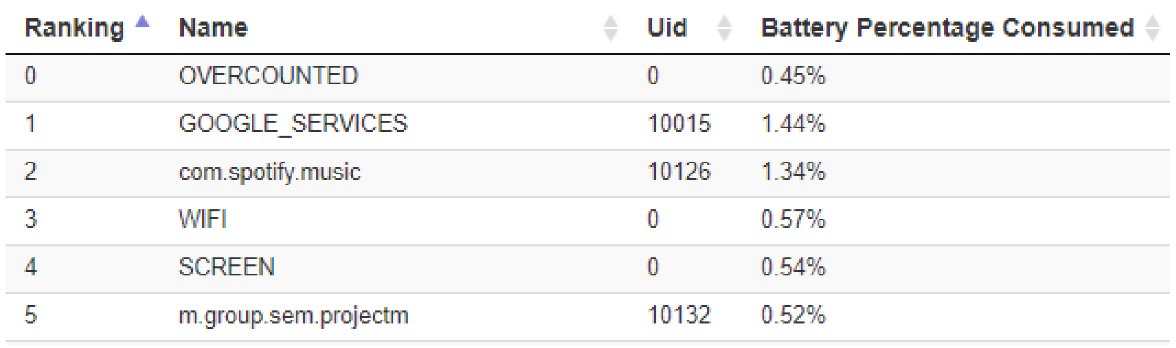


Figure 12: Image from Battery Historian.

This confirms that the implementation of dynamic duty cycling works as the location sensor accuracy was changed when Thomas approached and moved away from the report.

The accumulated discharge throughout the test was 7 percent with a test duration of 1 hour and 20 minutes. The power estimates throughout the test are shown in table 13.



Ranking	Name	Uid	Battery Percentage Consumed
0	OVERCOUNTED	0	0.45%
1	GOOGLE_SERVICES	10015	1.44%
2	com.spotify.music	10126	1.34%
3	WIFI	0	0.57%
4	SCREEN	0	0.54%
5	m.group.sem.projectm	10132	0.52%

Figure 13: Power estimates

Our solution comes in on a top 5 of the most battery consuming activities with 0.52% battery consumption. This means that our app is responsible for 7.14% of the battery consumption throughout the 1 hours and 20 minutes test. Compared to playing music using Spotify for the same duration, the battery consumption of our app is close to just one third of Spotify.

Increasing the battery consumption by 7.14% using our application would reduce the battery lifetime - of the device in question - by an estimated 1 hour and 20 minutes. Taking it from a 19 hour battery life to a 17 hour and 50 minutes lifetime. This seems like a fairly acceptable battery consumption considering that the application is relying on very precise GPS positioning.

Potentially the battery consumption of the application could be reduced by optimizing the background service. Some of the ways to optimze the application are mentioned in the known issues section of the discussion. This includes using the geofencing to reduce activity recognition in the background. Other factors, methods, and tactics can also be investigated further to optimize battery usage.

Unsolved tasks has been stated as well as known issues. Project reflection described the agile approach to the project and the discussion section shows the validation and battery consumption test of the application.

8 Conclusions

We have successfully implemented a prototype, with infrastructure capabilities for crowdsourcing of report validations. It does this by notifying end users, whenever they are geographically near a report, and allowing users to comment and confirm or deny reports. The prototype also allows users to create

There's a lot of reports in Odense [2] and prioritizing the reports can become a tedious task. By letting users validate reports it gives the municipality a way of prioritizing reports.

To keep users engaged work related to gamification has been considered. A suggestive leaderboard has been developed, but no logic has been implemented to make the leaderboard function.

A battery consumption test using the application have been performed. While having the application in the background for 1 hour and 20 minutes the application increased the discharge rate (% per hour) of 7%.

Future Works

In the future it would be great to pursue the rest of the addressed problems, and implement the suggested solution for the prioritization problem, and the incentive problem. It would also be interesting to work with other kinds of clients, such as the suggested web client, but also an iOS and maybe windows phone clients.

References

[1] Sweco Danmark A/S. Giv et tip - Odense. https://play.google.com/store/apps/details?id=com.grontmij.givetpraj.odense, 2017. Online; accessed 18 December 2017.

[2] Sweco Danmark A/S. Giv et tip - Odense (web client). http://dw3.dk/citizen/69f51374-13dd-49a9-a34a-3ee268ce5056, 2017. Online; accessed 19 December 2017.

[3] Mikkel Baun Kjærgaard and Marco Kuhrmann. On Architectural Qualities and Tactics for Mobile Sensing. *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures - QoSA '15*, pages 63–72, 2015.

[4] Google. Battery Historian. https://github.com/google/battery-historian, 2017. Online; accessed 19 December 2017.

A Use Case Descriptions

Name	Register User
Id	UC1
Description	Allows unregistered users to create an account, and become a registered user.
Primary Actors	Unregistered User
Secondary Actors	Server
Preconditions	The unregistered do not have an account
Main Flow	1. The unregistered user enters a username, and password. (a) Password has to be entered twice to make sure the user did not mistype it. 2. The password is hashed for security reasons. 3. The username and hashed password is sent to the server. (a) Alternative flow: No connection to server. 4. The server stores the password in the database. (a) Alternative flow: Username is already in use. 5. end.
Postconditions	The unregistered user has become a registered user.
Alternative Flows	No connection to Server: 1. The user is informed of the missing connection. 2. end. Username is already in use: 1. The unregistered user is prompted to pick a new username. 2. Return to step 1 in main flow.

Name	Login
Id	UC2
Description	Allows user to login to the application.
Primary Actors	Registered user
Secondary Actors	Server
Preconditions	The user is not logged into the application
Main Flow	1. The user enters his username and password and selects the login option. 2. The password is hashed to match against the password hash in the database. 3. The username and hashed password is sent to the server. (a) Alternative flow: No connection to server. 4. The server matches the username and hashed password with data from the database. (a) Alternative flow: Username or password does not match data in database. 5. The user is logged in, and receives a user object from the server. 6. end.
Postconditions	The registered user is now logged into the application.
Alternative Flows	No connection to Server: 1. The user is informed of the missing connection. 2. end. Username or password does not match data in database. 1. The user is prompted to pick a new username. 2. Return to step 1 in main flow.

Name	Create Report
Id	UC3
Description	Allows a registered user to report issues to the municipality
Primary Actors	Registered user
Secondary Actors	Server
Preconditions	A registered user is logged in, and has encountered an issue, which the municipality is responsible for fixing.
Main Flow	1. The user selects the 'create report' option. 2. The user enters a short description of the issue. 3. The user confirms the location of the report (a) Alternative flow: Wrong location. 4. The user selects the submit report' option. 5. The report is sent to the server. (a) Alternative Flow: No connection to the server. 6. The user is informed of a successful submission. 7. end
Postconditions	The report has been created.
Alternative Flows	Wrong Location: 1. The user corrects the location by selecting the correct location. 2. Return to step 4 in main flow. No connection to Server: 1. The user is informed of the missing connection. 2. end.

Name	Change Settings
Id	UC4
Description	Allows users to change the distance threshold for receiving notifications.
Primary Actors	User
Secondary Actors	NA
Preconditions	User wishes to change the distance needed to reports, in order to receive notifications
Main Flow	1. The user selects the 'settings' option. 2. The user selects the distance threshold. 3. end.
Postconditions	The distance threshold needed is updated.
Alternative Flows	NA

Name	View Report
Id	UC5
Description	Allows users to view report details, such as description, comments and votes.
Primary Actors	User
Secondary Actors	Server
Preconditions	The user wishes to see information about a report.
Main Flow	1. The user selects a report on the map to see. 2. The application collects the newest data on the report from the server. (a) Alternative flow: No connection to server. 3. The report is is shown to the user. 4. end.
Postconditions	The user can see the report details.
Alternative Flows	No connection to Server: 1. The user is informed of the missing connection. 2. end.

Name	View Leaderboard
Id	UC6
Description	Allows users to see the leaderboard standings.
Primary Actors	User
Secondary Actors	Server
Preconditions	The user wants to see the leaderboard
Main Flow	1. The user selects the 'view leaderboard' option. 2. The application collects the newest information from the leaderboard. (a) Alternative flow: No connection to server. 3. The leaderboard is shown to the user. 4. end.
Postconditions	The user can see the leaderboard.
Alternative Flows	No connection to Server: 1. The user is informed of the missing connection. 2. end.

Name	Validate Report
Id	UC7
Description	Allows users to validate reports, by giving them an upvote or downvote.
Primary Actors	Registered user
Secondary Actors	Server
Preconditions	The user is viewing a report, or have received a notification from based on his proximity to a report.
Main Flow	Precondition = View report: 1. The user selects either the 'upvote' or the 'downvote' option on the report. 2. The vote is sent to the server. (a) Alternative Flow: No connection to server. 3. end. Precondition = notification: 1. The user selects the 'confirm' or the 'deny' option in the notification. 2. The vote is sent to the server. (a) Alternative Flow: No connection to server. 3. end.
Postconditions	The user has validated the report
Alternative Flows	No connection to Server: 1. The user is informed of the missing connection. 2. Validation option is stored, for later retry 3. end.

Name	Comment on Report
Id	UC8
Description	Allows users to comment on each others reports, to add more information to the reports
Primary Actors	Registered user
Secondary Actors	Server
Preconditions	The user is viewing a report, or have received a notification from based on his proximity to a report.
Main Flow	Precondition = View report: 1. The user enters a comment for the report. 2. The user selects the 'submit comment' option 3. The comment is sent to the server (a) Alternative Flow: No connection to server. 4. end. Precondition = notification: 1. User selects the 'comment' option in the notification. 2. The user enters a comment for the report. 3. The user selects the 'submit comment' option 4. The comment is sent to the server (a) Alternative Flow: No connection to server. 5. end.
Postconditions	The user has submitted a comment to the report.
Alternative Flows	No connection to Server: 1. The user is informed of the missing connection. 2. Comment is stored, for later retry 3. end.

Name	Display Report Notification
Id	UC9
Description	Allows the application to send notifications to the user, if he is within his set distance threshold of a report.
Primary Actors	Location
Secondary Actors	NA
Preconditions	The user has location services enabled.
Main Flow	1. The distance from the user's current location is within the set threshold 2. The application checks if the user has already contributed to this report. (a) Alternative flow: Alternative Flow: User already contributed to report: 3. The application checks if the user has been shown this report before (a) Alternative Flow: Report already shown. 4. The application gets the latest information about the report. (a) Alternative Flow: No connection to server. 5. The user is asked to validate the report. 6. end.
Postconditions	A notification is shown to the user, and he can respond to ut with UC7 or UC8.
Alternative Flows	User already contributed to report: 1. end. Report already shown: 1. end. No connection to Server: 1. end.

Name	Logout
Id	UC10
Description	Allows the user to logout.
Primary Actors	User
Secondary Actors	NA
Preconditions	The user wishes to log out of the application.
Main Flow	1. The user selects the 'logout' option in the menu. 2. The application removes stored account information. 3. The user is redirected to the login screen. 4. end.
Postconditions	The user is now logged out.
Alternative Flows	NA