# Requirements and Analysis Document for PlankaGBG

Lucas Karlsson, Joakim Ohlsson, Seif Bourogaa,
Joakim Tubring, Filip Hanberg

2020-10-11
version

## 1 Introduction

The applications purpose is to report current positions of ticket controllers in near real-time. The model will map reports to geographical coordinates and position in the public transportation network. The reports in this case will create alerts for when ticket controllers are in the nearby area or on the same route as the user. Furthermore the applications purpose is also to gather information from the reports and store that information anonymously for the future.

### 1.1 Definitions, acronyms, and abbreviations

1. **GUI**: The graphical interface, how the functionality is displayed to the user.
2. **Model**: The package of the program that will manage all functionality of the application.
3. **View**: The package of the program that that contains the GUI.
4. **Controller**: 1: (**Code**) The package of the program that connects the View package to the Model package. 2: (**Model**) An individual responsible for checking whether or not a public transit passenger has a valid ticket.
5. **Incident**: Multiple reports that takes place on the same route or station.
6. **Report**: When the user sees controllers the application lets the user compile the information and submit it. A report.
7. **Reporter**: The user.

8. **Non-functional requirement**: Requirement that defines how a system should be.
9. **Functional requirement**: Requirement that defines what a system should do.
10. **Java-doc**: Java-doc is a documentation generator that generates HTML documents from java source code.

# 2 Requirements

## 2.1 User Stories

Story Identifier *US1* - **Make a Report**

1. Description:
   - As a User, I want to be able to report when I see controllers so that other users can see where the controllers are.
2. Acceptance Criteria:
   - A Report is made.
   - Said report is connected to an incident.
     - If there is no incident, create a new one.
   - The incident is updated and displayed in the GUI.
3. Functional Requirements:
   - Can I click a button to make a Report?
   - Can I make different kind of Reports?
4. Non-functional Requirements:
   - **Reliability:** Application should not crash when creating a new Report.
   - **Usability:** It should be easy for the User to create a new Report.

Story Identifier *US2* - **Endorse Incident**

1. Description:
   - As a User, I want to be able to endorse an Incident so that other Users can see that it is trustworthy
2. Acceptance Criteria:
   - An Incident is endorsed.
   - Said Incident is ranked as more trustworthy, if the User reports it as such.

- The incident is updated and displayed in the GUI.

3. Functional Requirements:
   - Can I repel or dispute an Incident?
   - Can I backup and Incident?
4. Non-functional Requirements:
   - **Reliability:** Application should not crash when endorsing an incident.
   - **Usability:** It should be easy for the User endorse an incident.

Story Identifier *US3* - **Search and Browse the Network**

1. Description:
   - As a User, I want to be able to see where the controllers are so that I can be more careful in my travels.
2. Acceptance Criteria:
   - Latest Incidents shows up in GUI.
   - See if there are active incidents on my tram line.
3. Functional Requirements:
   - Can I click on a button to search for Incident relevant to a tram line?
   - Can I see recent Incidents in a the GUI?
4. Non-functional Requirements:
   - **Scalability:** It should be easy for Developers to scale the functionality.
   - **Usability:** It should be easy for the User browse the controllers in the Network.

Story Identifier *US4* - **Identifying Image**

1. Description:
   - As a User, I want to be able to attach an image to my report to further strengthen its credibility.
2. Acceptance Criteria:
   - Image shows up in GUI where Incidents are displayed.
   - User Report and its connected Incident credibility is strengthened.
3. Functional Requirements:
   - If I create an Report can I add an image?

- Can I change the image afterwards?

4. Non-functional Requirements:
   - **Usability:** It should be easy for the User attach an image to a report. .

Story Identifier *US5* - **Trust Factor**

1. Description:
   - As a User, I want to have a trust factor that impacts the credibility of my Reports.
2. Acceptance Criteria:
   - When a Report is made and is backed up by other Reports the User's trust factor changes.
   - When a Report is created and auto generated trust factor is assigned to that Report, based on the trust factor the User hold.
3. Functional Requirements:
   - Can I change my trust factor?
   - Can I change others trust factor?
4. Non-functional Requirements:
   - **Security:** One user should not have direct access to another User's private information.
   - **Reliability:** Trust factor should not be changed randomly for each user.

Story Identifier *US6* - **Modify Report**

1. Description:
   - As a User, I want to be able to change or modify a Report after it has been made.
2. Acceptance Criteria:
   - When a Report is made I want to be able to modify it.
   - The Report is updated.
   - The updated Report is displayed in the GUI.
3. Functional Requirements:
   - Can I add or remove something from a Report I have created?
4. Non-functional Requirements:

- **Security:** One user should not have direct access to another User's report, meaning they should not be able to change or modify it.
- **Reliability:** A report should not be modified unless the User modifies it.

Story Identifier *US7* - **Vouch for another User's Report**

1. Description:
   - As a User, I want to be able to vouch for another User's Report in cases that helped me.
2. Acceptance Criteria:
   - Users want to be able to vouch for a report by voting.
   - User that created the Report has their trust factor increased.
3. Functional Requirements:
   - Can I vouch for a report?
   - Can a User's trust factor be changed by a vote?
4. Non-functional Requirements:
   - **Reliability:** A user should be able to vouch for another Report without performance issues, such as application crashes.

Story Identifier *US8* - **Notification**

1. Description:
   - As a User, I want to be notified if a controller is on my route and, or, near me.
2. Acceptance Criteria:
   - User's should have the option to be notified.
   - User's should be notified about controllers on their route, or near them, if they have said option enabled.
3. Functional Requirements:
   - Can I enable a notification function?
   - Is the notification function affected by Reports?
4. Non-functional Requirements:
   - **Manageability:** The User should easily be able to turn the Notification on and off.
   - **Reliability:** The application should not crash when sending the User notifications.

Story Identifier *US9* - **Map**

1. Description:
    - As a User, I want a graphical representation of where reports has been made.
2. Acceptance Criteria:
    - User's should have access to a map that displays Reports.
    - Map should display where and when the Reports has been made.
    - Map should correspond to all Reports.
3. Functional Requirements:
    - Can I see where Reports has been made on a map?
    - Can I see when the Reports was made on a map?
4. Non-functional Requirements:
    - **Manageability:** The User should easily be able to switch to the Map view.
    - **Reliability:** Map should not force the User to close to app to switch view.

## 2.2 Definition of Done

The entire code should be commented using Java-doc standard, methods and variables should be declared in such a way they are easy to understand and the code should have been tested and peer-reviewed by the group. Any visual elements that are part of the code should be implemented in the GUI of the application. After every feature branch has been approved they should be merged to master to allow for continuous development and integration.

## 2.3 User interface

Our initial GUI sketches can be seen in *Figure*1 and *Figure*2 below. The GUI will consist of multiple views. Live Reports, here the User will be able to see the active reports in Göteborg. The Live Reports view will consist of the location of the reports, how many reports there are and what time the incident occurred. Another view is the Report View, here you fill in information on the incident you witnessed, where the controllers are and when, and the application updates the live reports. The application also has a login view, here you register an account and you will also be able to change information your information.

The view that the program launches in depends on if you have opened it before, if its your first time launching the app you will have to create an account. If not you

will directly launch into the Live Reports section. From here you can navigate to either Account Settings view or Report Incident view. The thought behind this simple interface is that the application is supposed to be easy to use under stress or pressure, as we reckon that people usually are heavily stressed in the public transport network.

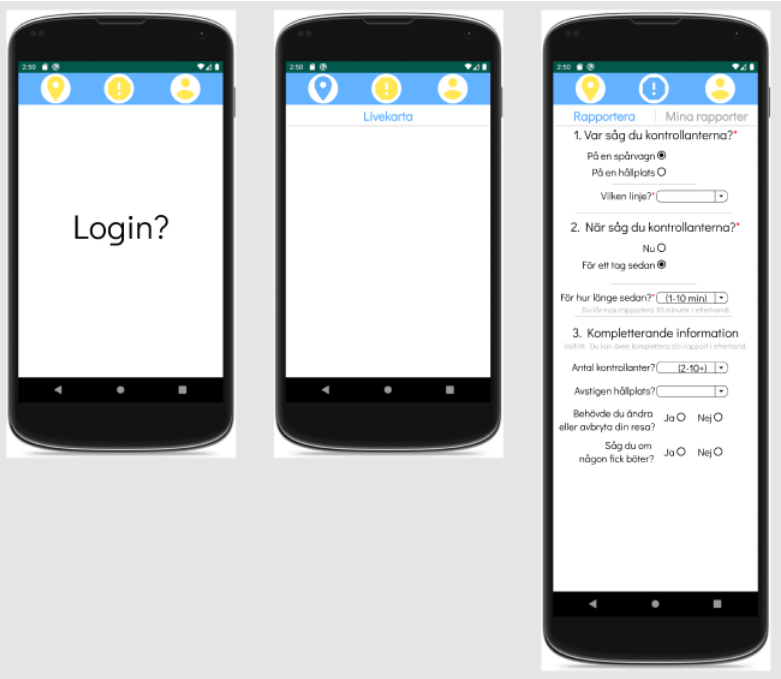


Figure 1: Preliminary GUI

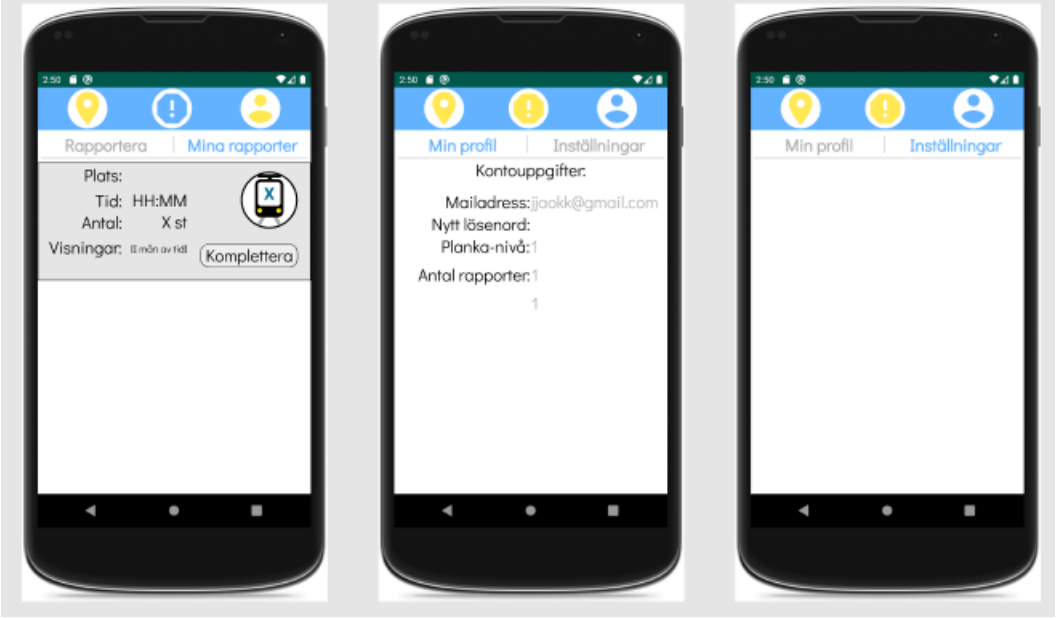


Figure 2: Preliminary GUI

# 3 Domain model

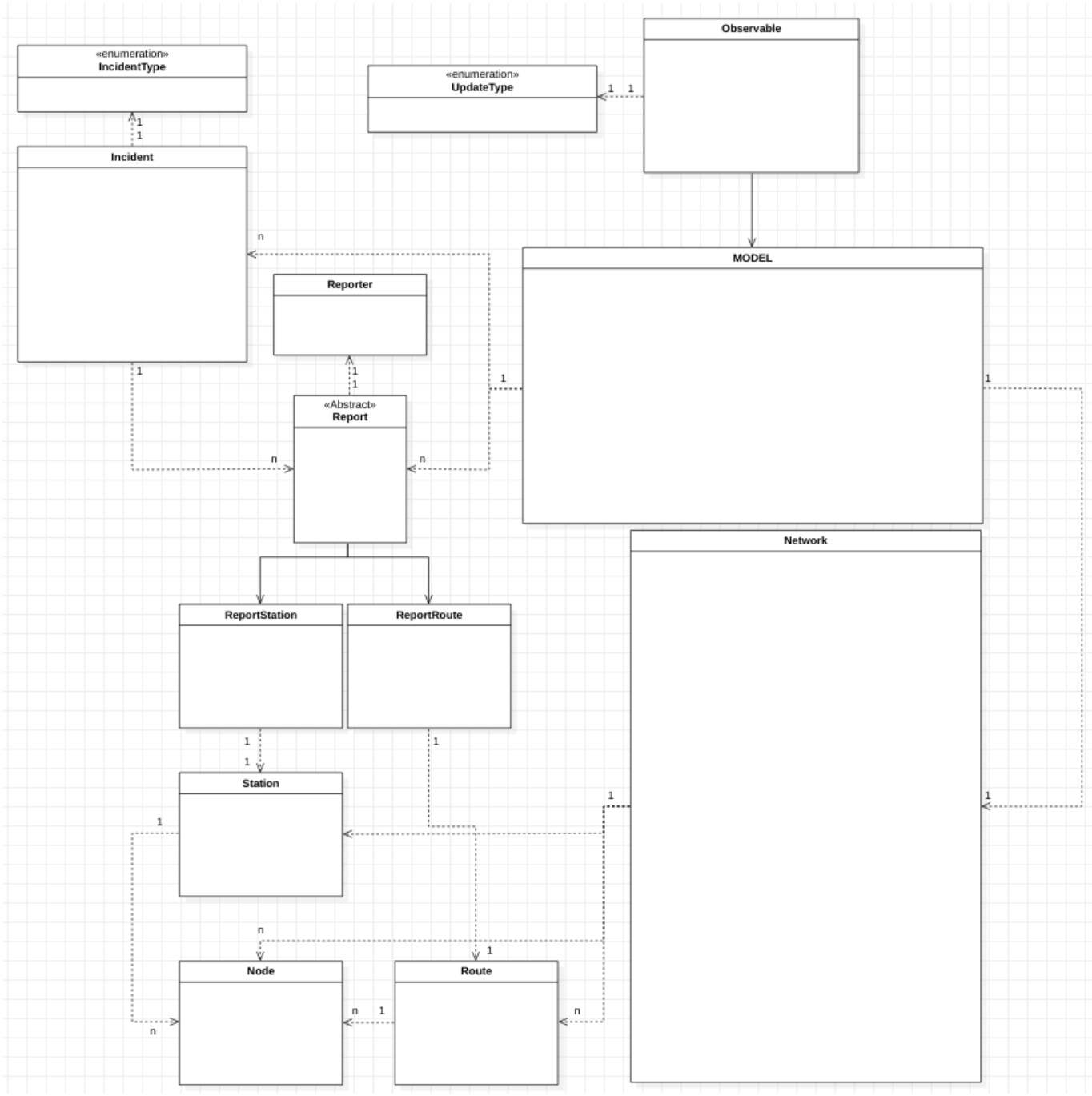


Figure 3: Domain Model

### 3.1 Class responsibilities

Explanation of responsibilities for the classes in *Figure*3 :

1. Node
   - Class responsible for each tram station in the network.
   - Contains information such as name and alarm state.
2. Route
   - Class responsible for each tram line in the network.
   - Takes a list of Nodes, one Node for each station the tram passes by.
3. Network
   - Class responsible for the entire tram network and its connections. Built up by adding every route to the data structure.
4. Station
   - Class responsible for representing the stations in our tram network, a station contains a list of individual stop positions for a station.
   - For example, Station Centralstationen contains Centralstationen A, Centralstationen B etc.
5. Reporter
   - Class responsible for representing the User in a report.
   - Protects the Users integrity and only contains an email address and a factor for how trustworthy the User is.
6. AbstractReport
   - Abstracts content from the RouteReport class and StationReport class and represents the reports for the Incident class.
7. RouteReport
   - Class responsible for Reports that involve both a Station and a Route.
8. StationReport
   - Class responsible for Report that only involves a Station.
9. Incident
   - Class responsible to gather Reports that all involve the same Station and, or, Route.

## 4 References

List all references to external tools, platforms, libraries, papers, etc.