Team members:
Waldo van der Merwe; u15018556 (Team leader)
Joan Mwaniki; u16159323
Brian Ndungu; u15322913
Dewald van Hoven; u15030378

Taxi Boss SRS

Supreme Internect

July 2019

Contents

1 Introduction

1.1 Purpose

This Software Requirements Specification (SRS) aims to provide a detailed overview of the software product, Taxi Boss, to be developed and delivered, as well as the parameters and goals of the project.

1.2 Scope

A new system will be developed that will be used on mobile devices and web. This system will aim at monitoring taxi drivers through their mobile device and report any violations that the taxi drivers commit to a monitoring sub-system.

The public (commuters as well as any other citizen) will be able to report traffic violations that taxis commit through our system. However, the main aim of the system is to gather data regarding traffic violations that taxi drivers commit with the aim of enforcing better road usage by the drivers. The purpose of this is to provide a platform to improve the behaviour of taxi drivers, as well as the states of their vehicles. The approach is to empower the public by giving them a platform on which they can help make the taxi industry safer for everyone.

1.3 Definitions, Acronyms, and Abbreviations

TDA: Taxi Driver app
WM: Web monitor
SSP: Server-side processing
WA: Web App
PU: Public User (Any member of the public)
UC: Use Case
R: Functional Requirement

1.4 Overview

This document contains all of the software requirements and specifications, as well as an overview of the software product and its functions. User characteristics, constraints, assumptions, and specific requirements will also be outlined in this document.

2 User Characteristics

There are three types of intended users that will use the taxi boss system; the taxi driver, taxi monitor and the public.

The taxi driver will need to have a smart mobile device which will enable him to see the driving violations that he has committed. This user does not need any high technical level of understanding to use the app.

Members of the public who want to report traffic violations will need to have access to a web browser or cellphone that can run USSD codes.

The monitor needs access to any device that has access to an web browser which will enable this user to monitor the drivers that are currently registered with this user. They need to be internet literate and have some understanding on the different traffic violations and the severity of each one of them.

Other intended users includes the software engineers who will ensure that the system runs smoothly, as well as take note and fix any problems that may occur. They will require a high educational level and good technical skills in software engineering in order to interact with the system at the lowest level to ensure that it is functioning as required.

3 System Architecture

3.1 System Type

- The system is an Interactive System. The system is driven by users interacting with the system to perform business transactions. Commuters and drivers both interact with the system through their direct input and GPS information when using the app. The transfer of data to and from the server forms the core functionality of the system.

3.2 Architectural Style

The app's primary objective is to provide comprehensive reporting about the taxi drivers to the monitor. Thereby meaning that it would need to gather, analyse and visualize taxi related data. Taking this into consideration we noted that the core of this application relies on business logic, user interfaces (to interact with the data gatherers and viewers) and some form data repository. This abstract understanding of the interactive system provided us with the lowest level granularity architectural pattern of the system, the MVC pattern. This pattern is essential for the development of the app because the abstraction from the business logic made provisions for code sharing and reuse, as the application is meant to be accessed on android, iOS and web. It's modularity contributes to essentially making the application significantly more maintainable when there is need for change, which is important because the app will introduce a lot of features that would introduce new data analytics metrics iteratively.

3.3 Architectural Requirements

3.3.1 Database Server

A scalable database server is required. Cloud Firestore will provide the required database functionality. It is a highly scalable real-time database hosting service by Google. It uses no SQL and stores data in Documents in JSON format.

3.3.2 Server

A server will be hosted on Firebase Hosting. This server will be used to analyze the data that is in the database.

3.3.3 Web Framework

The monitor portal will be written using the Angular 7 web framework. The public reporting subsystem will also be hosted on Firebase Hosting. These apps will both interact with database.

3.3.4 Native app

The native app will be on Taxi Drivers' phones. This will be used to gather location data and send it to the server. It will also receive notifications if a lot of bad driving is picked up

3.4 Architectural Design

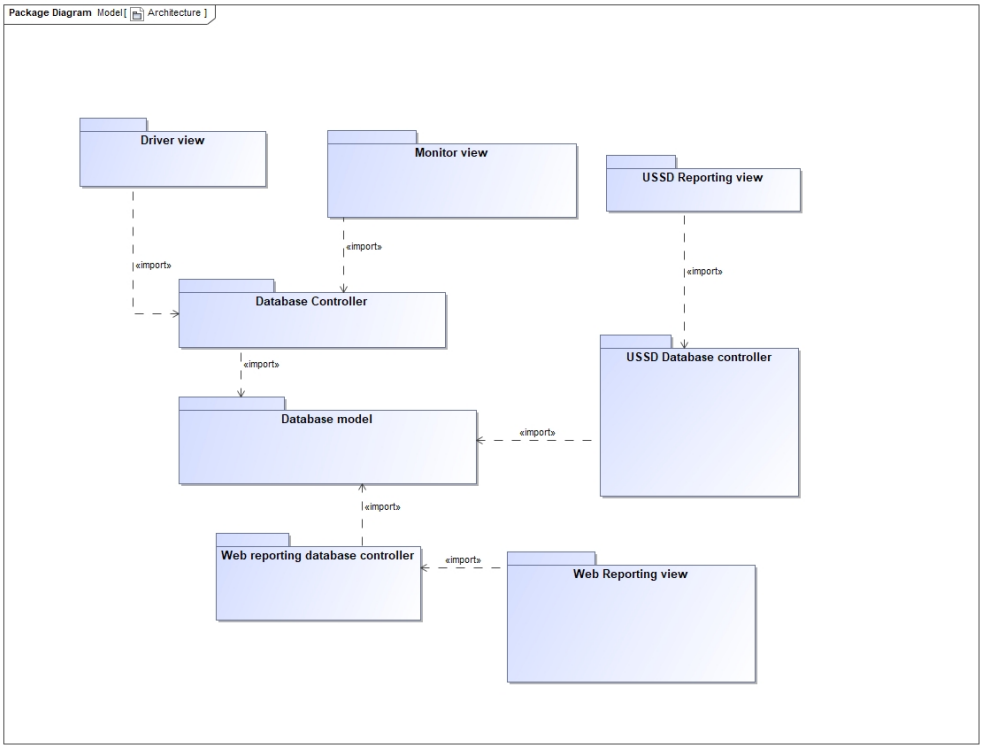

Figure 1: System Architecture

3.5 Deployment Model

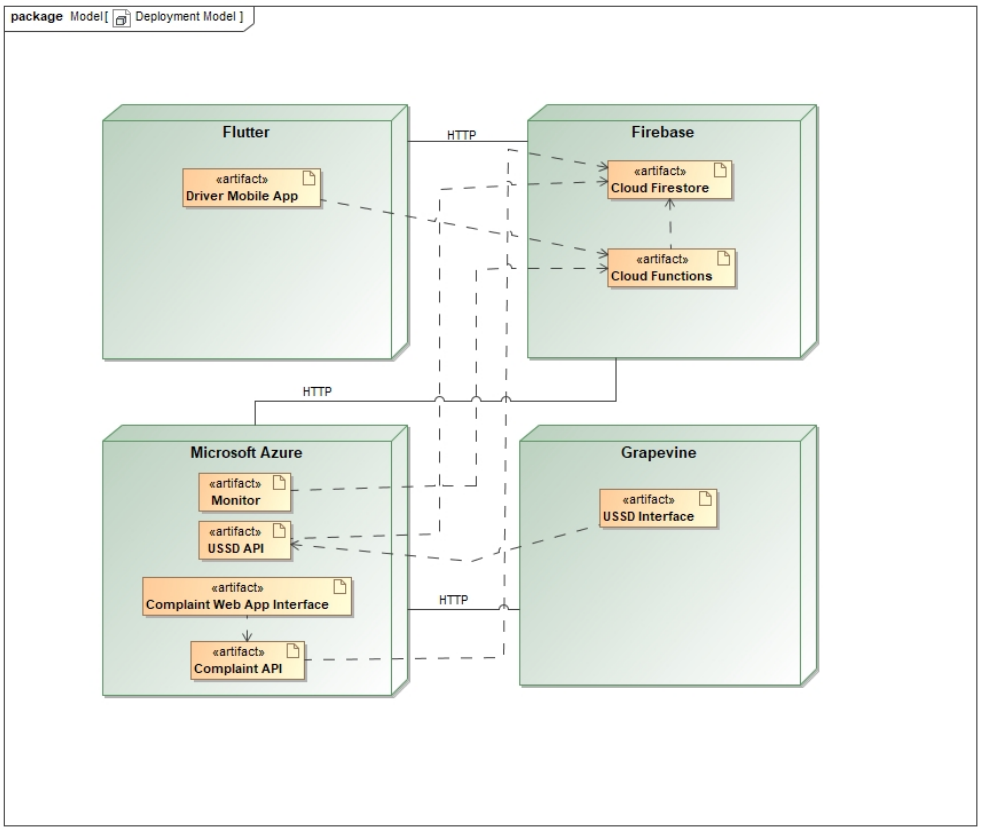

Figure 2: Deployment Diagram

4 Product Perspective

4.1 System Interfaces

The system requires cellphone or internet browser client and there is a central database server.
There will be a driver interface in the form of a cellphone app in Android and iOS and a web based app for the public. These apps will interact with the database. Interaction from the phone to the central database server will be handled with the Flutter framework. The database itself is a Cloud Firestore database.

The following data will be communicated with the server from the public web app:

- Any violations that a taxi has committed

The following data will be communicated with the server from the driver app:

- Location data
- Receive a notification of bad behaviour that must be corrected
- View his average driving behaviour rating

The following data will be communicated to the monitor browser interface:

- View drivers under this monitor
- View a driver's violations
- View statistics based on the data gathered

The following data will be communicated between the web server and the database:

- Location information regarding taxis
- Driver violations reported by the public
- Driving violations that are automatically detected by the system

4.2 User Interfaces

The public will interact with the cellphone web app.
There will be a USSD interface for the public to interact with.
There will be a taxi driver app that is available on Android and iOS.
There will be a monitor web app.

4.3 Hardware Interfaces

The following hardware interfaces will be used:

- A smart phone with a mobile network connection and a GPS controller
- A cellphone with a mobile network connection
- A computer
- Internet connection to the computer
- Mobile networks
- Server side hardware

4.4 Software Interfaces

Software interfaces that will be used:

- Android
- iOS
- Flutter UI framework.
- Firebase's Cloud Firestore database interface
- Angular
- Any web browser
- Firebase Hosting

4.5 Communication Interfaces

The application and website will make use of the TCP/IP protocol and asynchronous HTTP requests to communicate with the server. This connection will be encrypted for security. Smart phones will communicate GPS location information with GPS satellites.

4.6 Memory

In terms of the memory requirements of the users' mobile device, it will depend on the memory of the actual device. In terms of the software application itself, it will require somewhere between 60 MB - 90 MB of storage (this is an estimate). The user's mobile device will therefore need to be able to accommodate this requirement.
The only other consideration to make in terms of the memory, is the amount of memory the database will require on the actual server.

5 Constraints

- There are no constraints relating to the type of software to be used to implement any functionality, the decision is left to the design team.
- There is a budget constraint that the project will include no additional costs for software to be used. Exceptions can be made if it is really necessary, but should be avoided. If the database scales to a large enough scale, the costs of this scaling will be covered as necessary

6 Domain Model

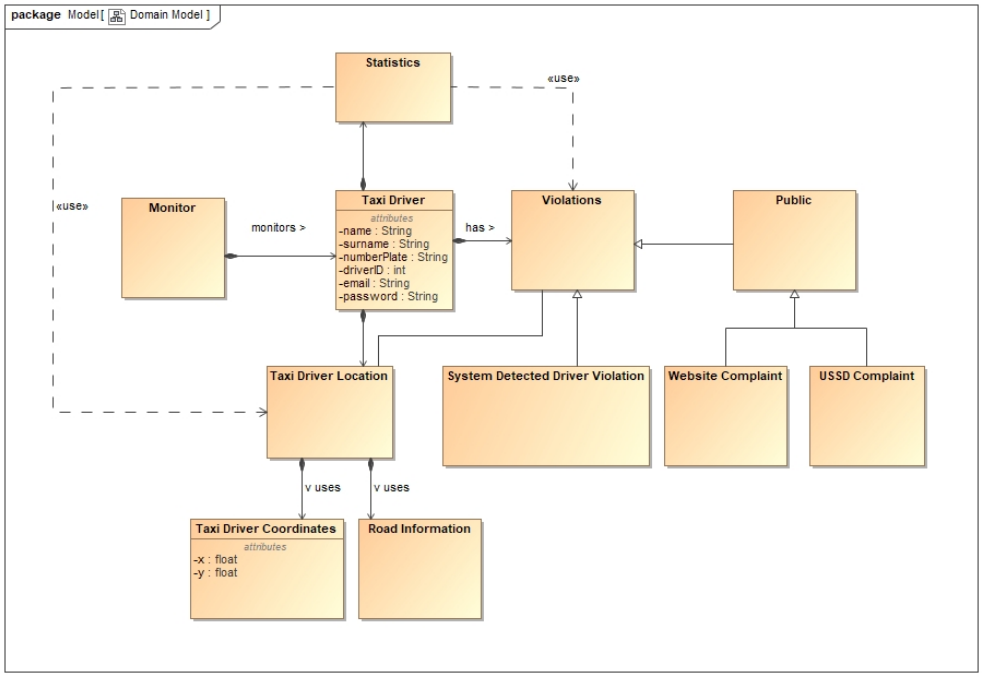

Figure 3: Taxi Boss Domain Model

7 Functional Requirements

Functional requirements are broken up into the various types of interfaces. The number of the functional requirement corresponds to the numbering scheme used in this list.

7.1 Subsystem: Taxi Driver app

1. The app will allow for a specific driver to log in and out of his/her account.
2. The app will allow for a specific driver to register an account. The registration information that will need to be provided is:
   (a) Name and surname of the driver
   (b) Email address of the driver
   (c) Driver's ID number
   (d) Registration number of the taxi the driver drives
   (e) The User ID of the monitor the driver is linked to
3. The app will allow the driver to change the registration number of his taxi.
4. The app will allow the driver to delete his/her account.
5. The current location of the driver will be communicated to the server.
6. The app will receive notifications of bad driving.

7.2 Subsystem: Public web app

7. The app will allow a PU to report any traffic violations. The following information will be required to report a traffic violation:
   (a) Registration number of the taxi
   (b) Time of incident
   (c) Location of incident
   (d) Type of violation by the following categories:
      i. Illegal stop
      ii. Using the emergency lane
      iii. Driving on the wrong side of the road
      iv. Illegal or unsafe overtake
      v. Unroadworthy vehicle
      vi. Violent behaviour
      vii. Skipping robot

viii. Skipping a stop sign
ix. Speeding
x. Incorrect use of indicators or hazards
xi. Cutting off other drivers
xii. Other

7.3 Subsystem: USSD interface

8. The USSD code interface will allow a PU to report any traffic violations. The following information will be required to report a traffic violation:
   (a) Registration number of the taxi
   (b) Time of incident
   (c) Location of incident
   (d) Type of violation by the following categories:
      i. Illegal stop
      ii. Using the emergency lane
      iii. Driving on the wrong side of the road
      iv. Illegal or unsafe overtake
      v. Unroadworthy vehicle
      vi. Violent behaviour
      vii. Skipping robot
      viii. Skipping a stop sign
      ix. Speeding
      x. Incorrect use of indicators or hazards
      xi. Cutting off other drivers
      xii. Other

7.4 Subsystem: Web monitor

9. The website will allow a monitor to create an account with the following credentials:
   (a) Name and surname
   (b) Email address
10. The website will allow a monitor to manage his/her account.
11. The website will allow a monitor to log in and log out of his/her account.
12. The website will allow a monitor to add or remove a taxi driver from his/her account.

13. The website will allow a monitor to view the taxi drivers under his monitoring account.

14. The website will allow a monitor to see traffic violations committed by the driver.

15. The website will allow a monitor to see statistics and graphs regarding his drivers' traffic violation statistics.

7.5 Subsystem: Server-side processing

16. If the server picks up that the driver has committed a traffic violation, it will notify the driver of the violation. This will be handled with an AI module.

17. Location data will be analyzed and maintained on the database.

7.6 Use Cases

7.6.1 PU Reports an incident with the Taxi Boss Web App

Report Incident with web app.jpg

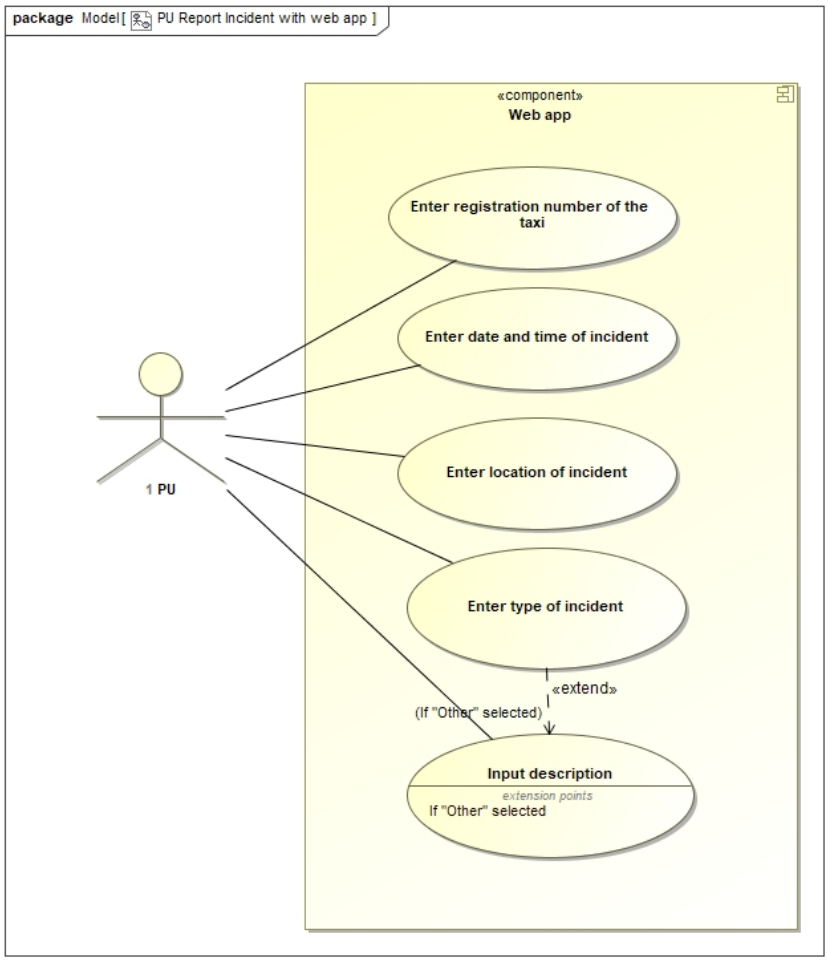

Figure 4: PU Reports incident with web app

7.6.2 PU Reports an incident with the Taxi Boss USSD code

Report Incident with USSD code.jpg

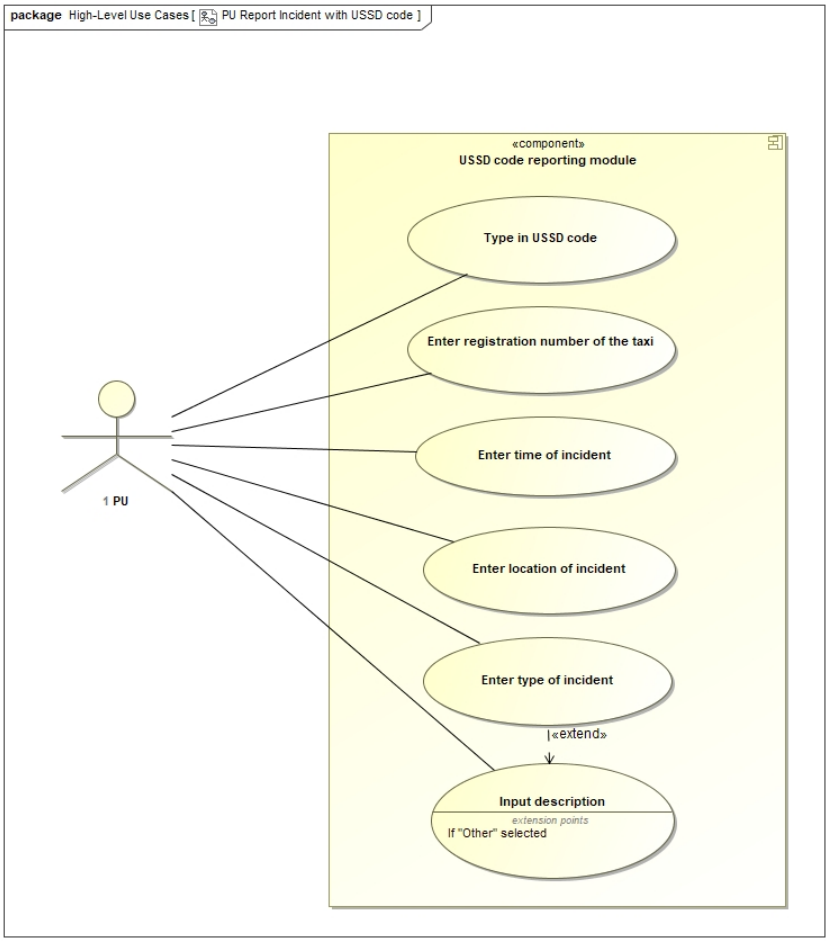

Figure 5: PU Reports incident with web app

7.6.3 Taxi Driver Violation Notifications

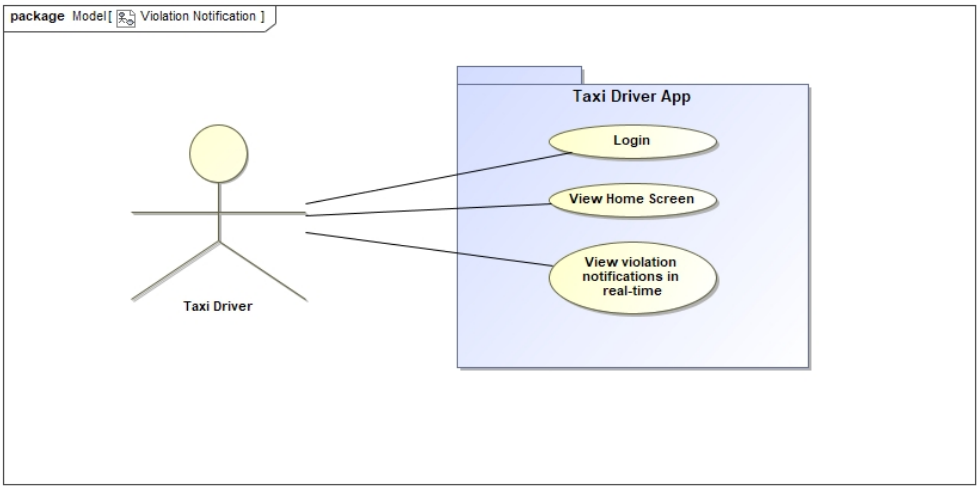

Figure 6: Taxi driver views violations in real-time

7.6.4 Monitor Account Registration

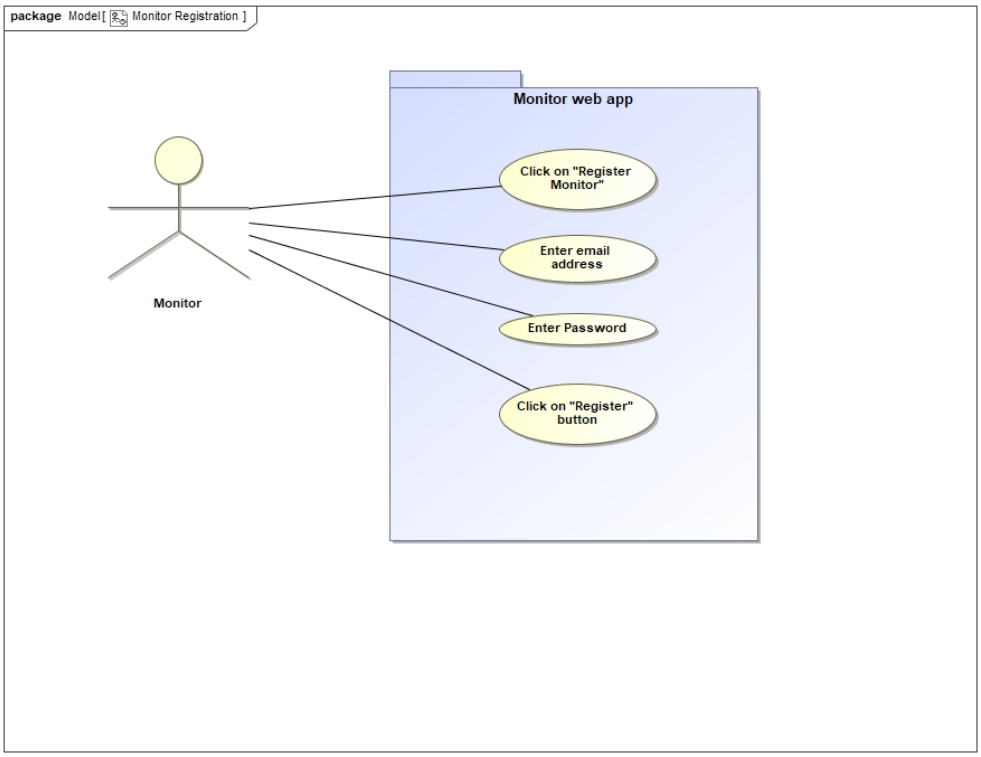

Figure 7: Monitor registration

7.6.5 Monitor login

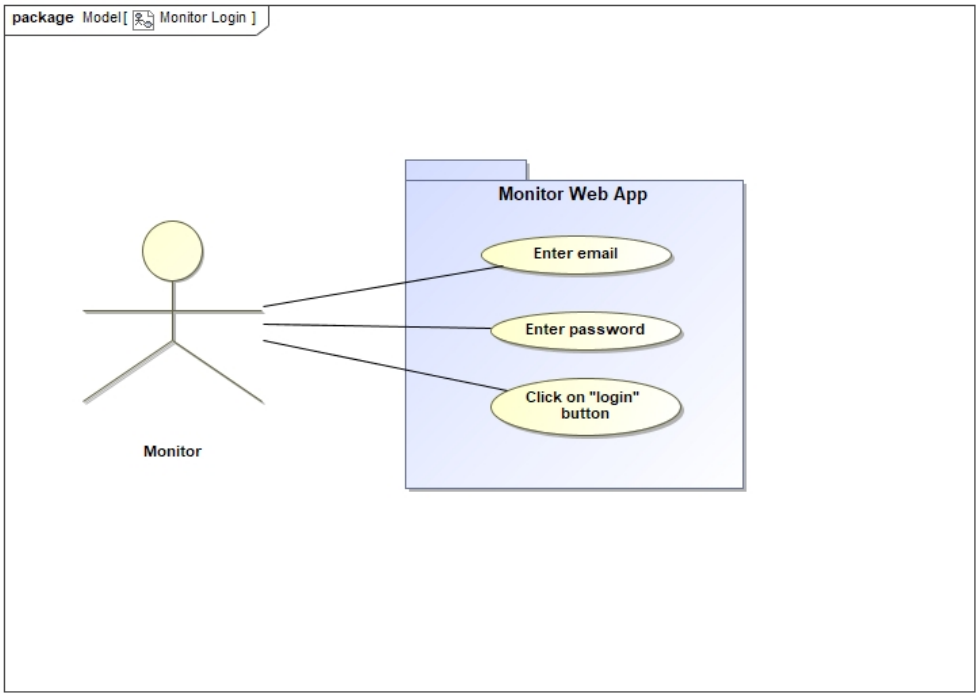

Figure 8: Monitor login

7.6.6 Register a Driver

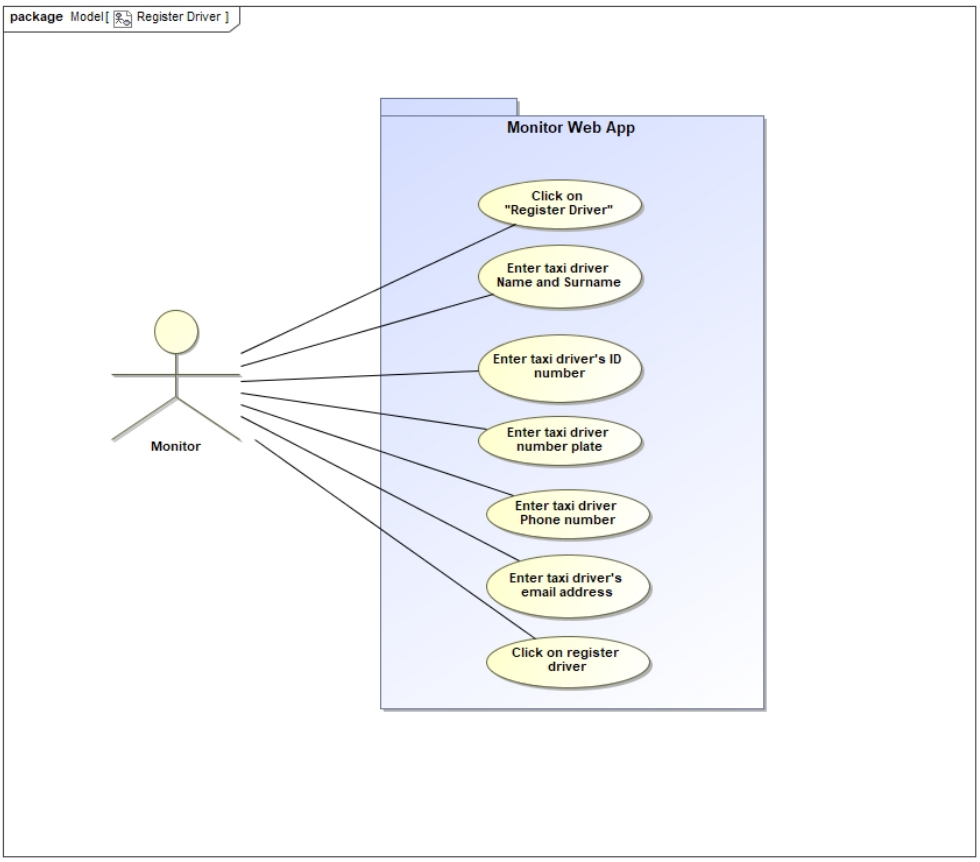

Figure 9: Registration of a driver

7.6.7 View Registered Drivers List

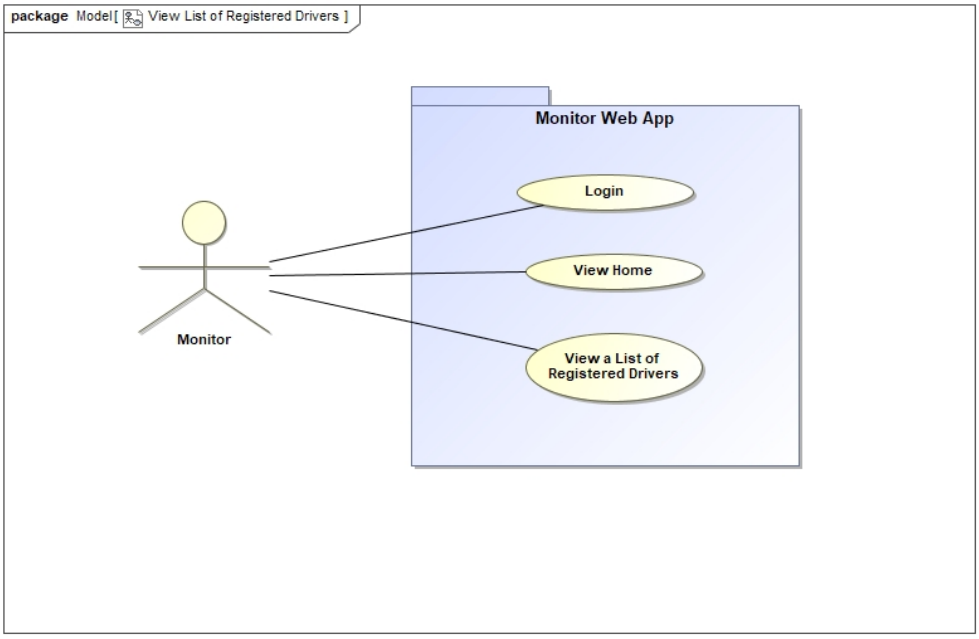

Figure 10: Viewing all drivers registered to a monitor

7.6.8 High Risk Drivers

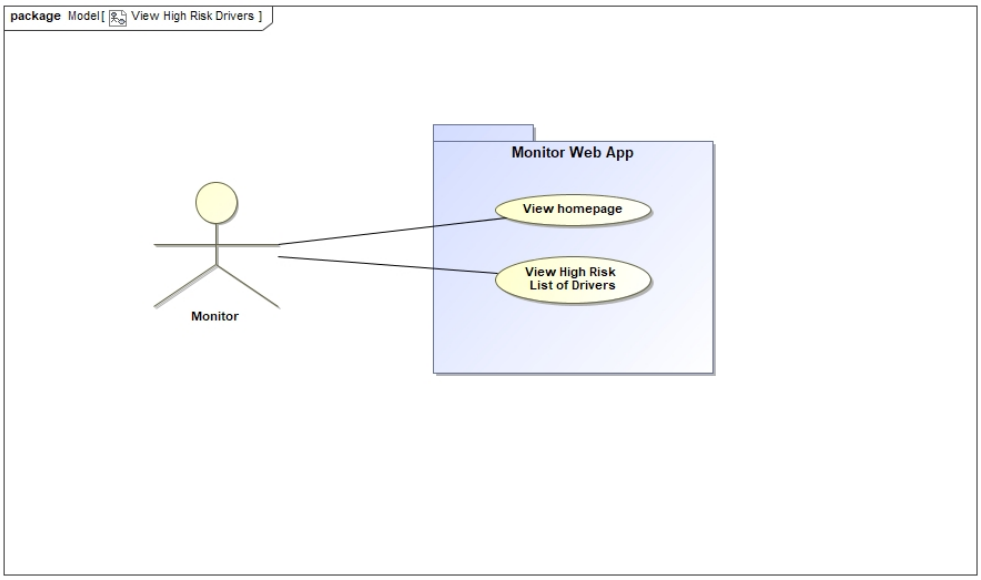

Figure 11: View a list of high risk drivers

7.6.9 View Number of Registered Drivers

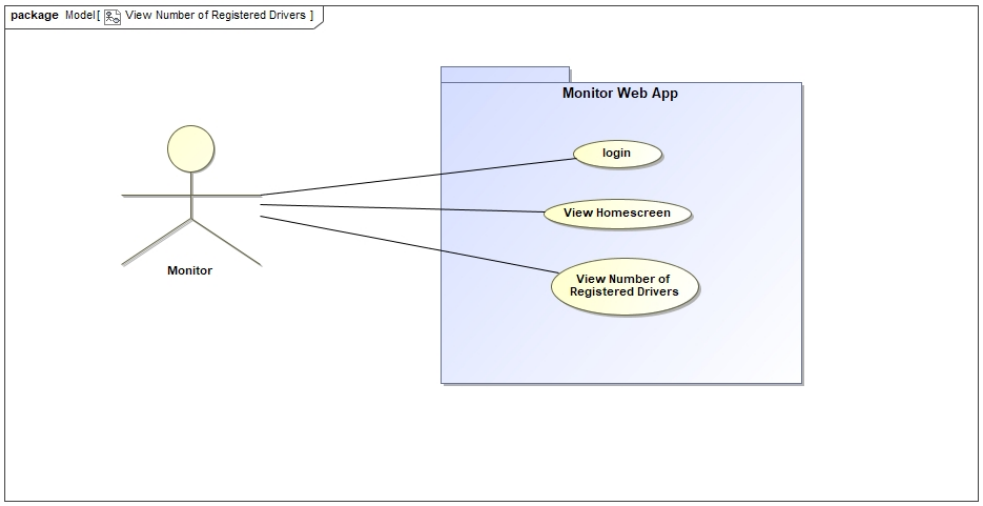

Figure 12: Viewing the number of registered drivers

7.6.10 View Driver Violations

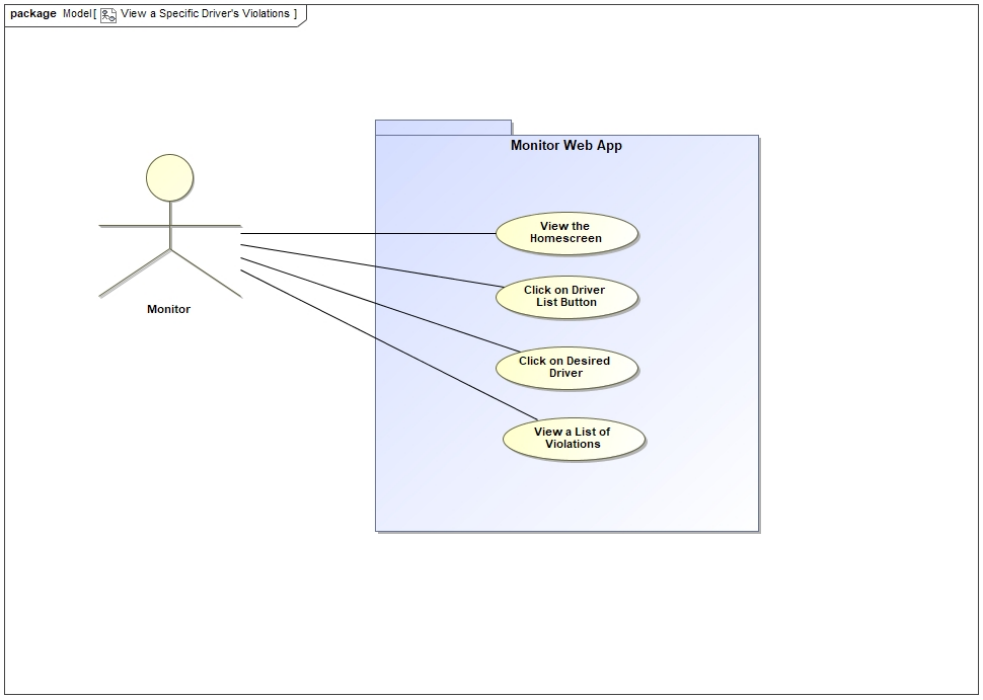

Figure 13: Viewing all violations of a specific driver

8 Quality Requirements

8.0.1 Security

1. Only monitors that have been authenticated will be allowed access to data.
2. FirebaseUI provides a drop-in authentication solution that handles the UI flows for signing in users with email addresses and passwords, phone numbers, and with popular federated identity providers, including Google Sign-In and Facebook Login.
3. Firebase takes care of security for us, making it easier to protect our data.

8.0.2 Scalability

1. The system should be able to operate effectively and efficiently under high loads of concurrent entries.
2. This is achieved through our Firestore real-time database, that is highly scalable and offers automatic horizontal scaling in and out, in response to our application's load.

8.0.3 Testability

1. Flutter provides a robust testing framework which provides memory leak finders, test case generators and page object generators. This provides for an effective means of doing static analysis and core generation.
2. Flutter also comes with a variety of testing packages which provide core frameworks for writing unit tests, as well as additional utilities for testing widgets.
3. Travic CI offers continuous integration and should launch our app after all tests have been passed.

8.0.4 Portability

1. Our app will be portable across platforms, because it is developed in Flutter.
2. Flutter delivers ARM binaries for android and IOS. Which is very similar to native technology, thereby allowing for quick compilation.
3. We will also offer a web-based application for users as well as USSD functionality.

8.0.5 Usability

1. Our apps will be highly usable, with a simple, yet aesthetically pleasing user interface.
2. We will incorporate Angular's component library, which offers great UI designs integrated with Goggle's material design.
3. We will also incorporate an Angular module for FusionCharts, which will generate responsive graphs and charts, for graphical representation of data. This will make it easier for users to understand data and statistics.

8.0.6 Reliability

1. All data is encrypted in transit by Firebase, using HTTPS, making sure all the data generated remains authentic.
2. This also protects data from Man In The Middle Attacks.
3. All the data is further encrypted on the server side by Firebase.

8.0.7 Maintainability

1. We plan on designing for maintainability, by writing readable code, that is easy to understand.
2. We also plan to highly document our code, by using compodoc to generate decent documentation regarding our code. This allows for code reuse.

8.0.8 Auditability

1. Our app data will store all the data that has been pushed to the database and will be accompanied with a time stamp, so we can know when data was sent.
2. Firestore also keeps our user history, which will keep track of all users that make changes to the data.
3. It will know who has altered data and when data has been altered.

8.0.9 Compatibility

1. Our app is developed in Flutter, which allows for cross platform development.
2. It is also highly compatible with Firebase, because, while Flutter uses Google's SDK for app development, Firebase provides the back-end services, without maintaining your own services.
3. Further, all those backend tasks, can be easily implemented into our angular project, using firebase functions.

8.0.10 Availability

app will offer USSD code functionality, that will allow users to send reports through USSD codes, whenever they have no internet connection, or if their devices do not have internet access.

8.0.11 Server Factors

1. Uptime: The Cloud Firestore database provides 99.999% uptime. The monitor website is currently hosted on Firebase Hosting. It provides 99.5% uptime. If maintenance needs to be performed on the server, it should be done between 00:00 midnight and 04:00. This is the time when drivers and commuters are the least active.
2. Speed: The database updates in real time with a latency of 2 seconds on average.
3. Cloud Firestore is a scalable database. As traffic and storage size increases, it will be easy to upscale the server based on payed accounts. Currently it is free.
4. Sensitive data on the server will will be encrypted.
5. Monitors will have special accounts to view statistics of their various drivers.
6. Firestore Hosting is a scalable server service. As processing requirement and number of hits increases, the server can be upscaled easily with payed accounts. Currently it is free.

8.0.12 User Interface Factors

1. The commuter and taxi driver interfaces will be easy to understand so that everyone is capable of using the app to its full advantage.
2. The monitor interface will have a small learning curve to it, but given that the data to be viewed by the monitor is not very complex, it will be easy to learn.
3. All interfaces will have a logo and accompanying styling to give the app a pleasurable and recognizable feel.

8.0.13 Quality Metrics

1. Smart phones using android should ideally have the Android Pie operating system installed.

2. Smart phones using iOS should ideally have the iOS 12.2 operating system installed

3. The smart phones being used should have more than 2GB of Memory and more than 5GB of storage as well as 3G or LTE mobile network data transfer speed capabilities.

4. The Cloud Firestore database should scale according to the amount of transactions being performed and the amount of users using it. Cloud Firestore offers this in that it is free when there is only low-scale data and traffic going through it, but can easily be upgraded to a payed account that will allow all necessary transactions to take place and space be used.

5. Most laptops and desktop PC's that will be used by the monitor will have a web browser installed. The monitor view will be easily accessible.

8.0.14 Newly added

1. Google Cloud Functions are hosted in London, UK, in the europe-west2 Google Cloud Platform Region because based on a latency test done on 2019/07/16, this region has the lowest latency back to South Africa, where our HTTP requests will originate from. This is also ideal because our Cloud Firestore database is hosted in the Google Cloud Platform europe-west region, which will keep communication latency between requests to the database from the functions to a minimum and keep billing as low as possible.

2. CORS restrictions has been circumvented in the Google Cloud Functions because the Monitor web interface as well as the Reporting web page is hosted on Microsoft Azure and the API functions are hosted on the Google Cloud Platform.

8.1 Functional Requirement to Use Case Traceability Matrix

	UC1	*UC2*	*UC3*
R1	x		
R2		x	
R3			
R4			
R5			
R6	x		x
R7			x
R8			x
R9			
R10			
R11			
R12			
R13			
R14			
R15			
R16			x
R17			