POLITECNICO
MILANO 1863

Computer Science and Engineering

A.A. 2019/2020

Software Engineering 2 Project:

"SAFE-STREET"

Desgin **D**ocument

December 9, 2019

Prof.Rossi Matteo Giovanni

Amirsalar Molaei

karim Zakaria Saloma

Erfan Rahnemoon

Contents

Figures

Tables

1 Introduction

The Software Design Document is a documentation of the intended system design used to convey the expected output of the development phase. In this section, an overview of the content and intended use of the document is discussed.

1.1 Purpose

The Software Design Document is built to describe a detailed description of the *Software To Be* from the architectural and technical aspects. This document specifies the manner in which the software shall be built through the use of narrative and graphical tools to aid in the communication of the necessary information to the concerned audience.

This document is intended to provide a clear and complete description of the system to the persons who shall be developing the system. In order to, assist in the understanding of how the system should be built and how the end product should function in accordance with the previously decided upon requirements and specification of the system.

1.2 Scope

In this section, the scope of the system previously described in the *Requirements and Specification Document* shall be revisited. As well as, consider some more in-depth aspects of the system.

As previously stated in the *Requirements Document*, the *SafeStreets* system shall be providing four main functions to various users; in this section, the system boundaries and scope used to define the limitations and different responsibilities of the S2B.

The first of the main functionalities is the enabling of users to report traffic violations. Regarding this, some phenomena are regarded as world phenomena not viewed by the system due to its limitations such as the fact that the system does not directly detect a violation. However, it can be accounted for by the system through a traffic report made by the users. Moreover, another functionality that has to do with the users is the publishing of collected data to be viewed by the users in a refined representation to help them consider the safety of various areas based on traffic violations. The data is also communicated to the authorities but with different levels of details.

The other two main functions have to do with the *SafeStreets* system providing services to government authorities. The domain limitations of the system affecting this interaction are also discussed in this section. Such as, the fact that the system is only able to make suggestions for preventive measures to the authorities based on the accident data that have been communicated. Meaning, that the system does not have any knowledge of accidents unless they are reported by the authorities and that the system can only suggest interventions and neither put them into place nor can detect them being applied. Moreover, a second function to the authorities would be the communication of traffic reports received from users to be later used by government officials to give out traffic tickets, the system responsibilities to support this process is to prevent the users from tampering with images *digitally* and to provide the collected reports to the authorities proactively. In other words, physical tampering with license plates to mislead authorities and the actual process of giving out tickets is not part of the application domain.

Moreover, in this document, some more technical issues regarding the functioning of the system need to be discussed. Primarily, the security aspect of the system; more specifically, data security. Since the system will be dealing with the collection and communication of sensitive data while performing more than one of the main functions; the system must, at all times ensure the safe transmission and storage of data and the application of measures to prevent any means of data tampering. The data being discussed includes but is not limited to, user personal data and detailed data of traffic reports.

1.3 Definitions,Acronyms,Abbreviations

1.3.1 Acronyms

S2B *Software To Be*

GPS *Global Positioning System*

UI *User Interface*

GDPR *General Data Protection Regulation*

UML *Unified Modeling Language*

RASD *Requirements Analysis and Specification Document*

OS *Operating System*

SMS *Short Message Service*

MVP *Model View Presenter*

REST *Representational State Transfer*

API *Application Program Interface*

1.4 Revision history

Revision	Date	Author(s)	Description
0.0	24/11/2019	karim Zakaria Saloma, Amirsalar Molaei, Erfan Rahnemoon	First document issue

1.5 Reference Documents

References

[1] *Standard for Information Technology–Systems Design–Software Design Descriptions IEEE 1016.* 2009.

[2] UML,
`https:omg.org/spec/UML`

[3] Specification Document. *SafeStreets Mandatory Project Assignment.*

[4] Microservices Architecture,
`https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices`

1.6 Document Structure

The Software Design Document(**DD**) is comprised of six main chapters. Which shall be described in this section of the document:

Chapter 1 (Introduction): provides an overview of the document as a whole; describing, the various sections constituting this document, as well as, the intended use of this document.

Chapter 2 (Architectural Design): a detailed description of the architecture to be developed during the implementation phase of the system; spanning from a high-level component view to a detailed run-time description of the different modules of the system. This is used as a guideline for the development team in order to have a clear idea of how the system should be built.

Chapter 3 (User Interface Design): an overview of the design of the different interfaces that the users of the system shall be interacting with the system through; in order to utilize the functionalities

of the system according to their needs. This overview is concerned with the visual aspects of the user interfaces.

Chapter 4 (Requirements Traceability): provides a link between the design decisions in this document and the requirements of the system described in the *Requirements and Specification Document*. This is done by providing an explanation of how the system design described in this document fully satisfies the requirements the system must abide by.

Chapter 5 (Implementation, Integration and Test Plan): describes the approach to be followed during the development and testing phase of the system. This is also provided as a clear guideline for the development team to follow.

Chapter 6 (Effort Spent): summarizes the efforts of the team members in developing this document in terms of time spent on each of the sections of the document.

2 Architectural Design

2.1 Overview

2.2 Component View

In the component diagram, there are four-part of the system which are the API from other companies that provide some functionalities as service. The google map API provides the map and Geo-location services and Firebase provides easy authentication for mobile users by push notification, plus the storage API for the storage of the violation images. Storing the amount of image that the program is going to work with it needs too much space and handling a database specialized for multimedia usage, so Firebase Storage is a more pertinent decision. In the same manner authentication by email or phone number (SMS) has it is own problems, so the application uses google firebase authentication.Sentry system helps to log the system in necessary parts and also makes a bug tracking mechanism for the problem happing in the android app and also server-side. Moreover, the database it seld is another component based on the architecture of the system which should be more scalable, as a result, the system will work with the database through the API provided by the ORM used in the system, which is the SQLAlchemy. The authentication, storage, and map API are connected to the android app and server, which as shown in the diagram each of these interfaces has their own usage and the functionality of them is not the same. Furthermore, two authority API is assumed in the system because in one of them the manipulability sends the data for the system, and in another they the system will send the data related to violation reports and interventions by their request. The authorities usually work with a lot of APIs, and usually, they are not integrated into one system, therefore to make the system as compatible as possible with existing systems this architecture is chosen.
On the server-side, for more isolation, the system is broken into seven-part which help to easy deployment and further development in the system. The only connection between the subsystems is the API provided by the "RegistrationAuthentication and login" part which the ViewMe and ReportViolation part are using this API. The reasons to provide internal API instead of directed access by the functions to this subsystem is first to make the system more scalable in future development; second, for security which this part has sensitive information about the user is better to be isolated from the whole system and has more strict rules to access to it.
The reset of subsystems so their job without any knowledge about the other parts of the system, and there is no direct connection between the part outside of the system and database. Even for the data which comes from outside first, the data will be stored in the database by responsible component for that request or response and then it can be used by other components and no direct connection between the components. The ReportViolation will handle the functionalities related to reported violations, and the ReportSubmission is responsible for the request from authorities. Furthermore, the ViewMe is part which allows users to change the data of their profile and see their previous report and ViewSafety will manage the request for the data system needs to provide the safe areas on the map. The "RegistrationAuthentication and login" is in the head of user registration and credential of the users to work with the system. There is two part of the system which they do not provide any service for the normal end-users one is the interventions recommender which will analyze the data and recommend the necessary intervention to authorities to decrease the number of violation and accidents, and next is the mechanism to check the images of the violation report from users and detect those who are manipulated and they are not original.
In the android-side of the system, there is one important component, which is the plate number recognition which detects the plate number by images is taken by users and shows the plate number in pre-report. The last component in the android app is the local database, which is necessary to store some data about the user and its reports. Nevertheless, the android app has some other part which they just representer and can not assume them as a component.

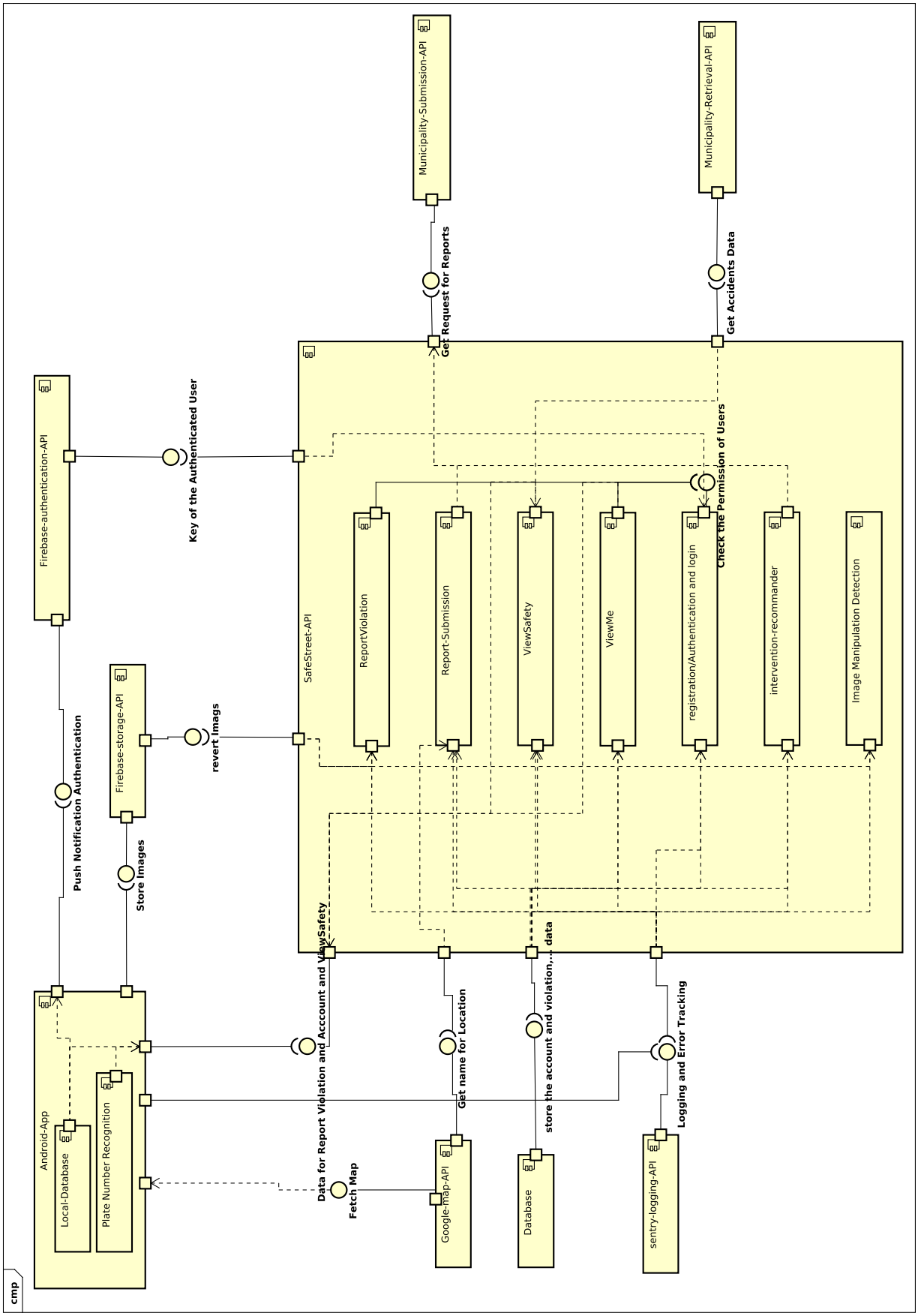


Figure 1: Entity Class Diagram

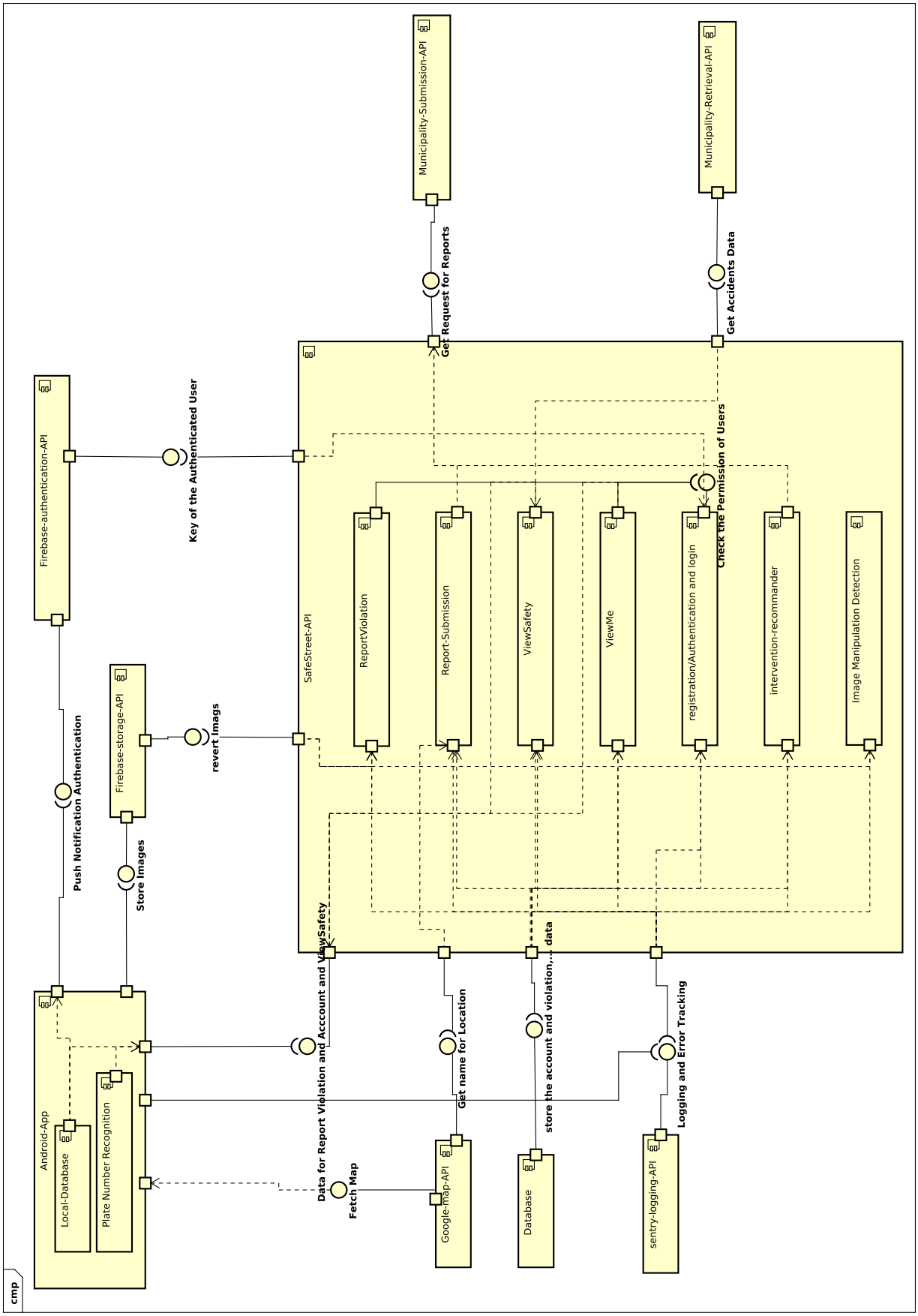


Figure 2: Entity Class Diagram

2.3 Deployment View

2.4 Runtime View

This section of the document is concerned with the dynamic interactions between the various components of the system during runtime to achieve the desired functions of the system. The section is constituted of four major subsections grouping together various runtime views of different parts of the system. These subsections are as follows *general functions, violation reports, view safety* and *municipality interaction*. In the following diagrams some simplifications were implemented in order to remove redundant details; however, they should still be mentioned, such as, that each and every server module interaction should correspond to a *log entry*, and all exceptions that may occur in the system should be error logged. Note that in the diagrams, some examples are used for logging and error logging interactions. Moreover, it is noteworthy that all external incoming and outgoing data passes through the *Security module* encryption and decryption and the same process occurs on the mobile app.

2.4.1 General Functions

This first subsection considers the runtime behavior of system components for the delivery of basic general functions. Specifically, *registration and verification, login* and *profile viewing and info editing*. The first two runtime diagrams below describe the user registration and verification process. As can be seen in the diagram a new user fills in the registration form with his information the submits the form. The data is relayed to the *Safestreets-API* which saves the user data and prompts the *Firebase-Authentication* server to verify the new user. Which in turn, sends the verification message to the user and returns feedback of the user confirmation.

Figure 3: Registration Runtime View

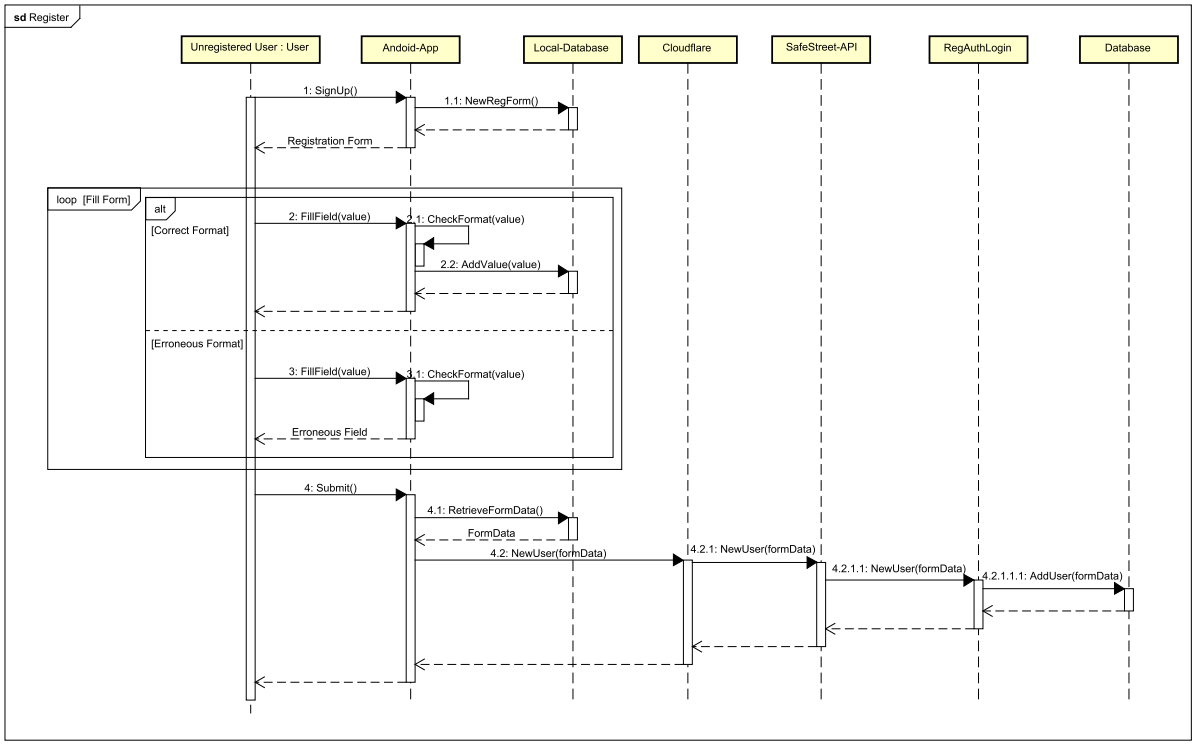
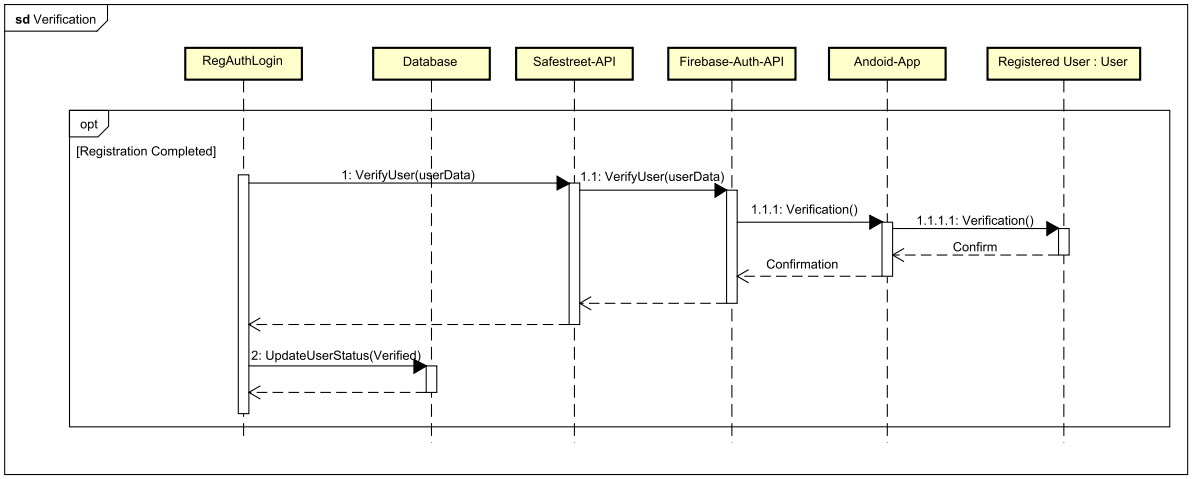
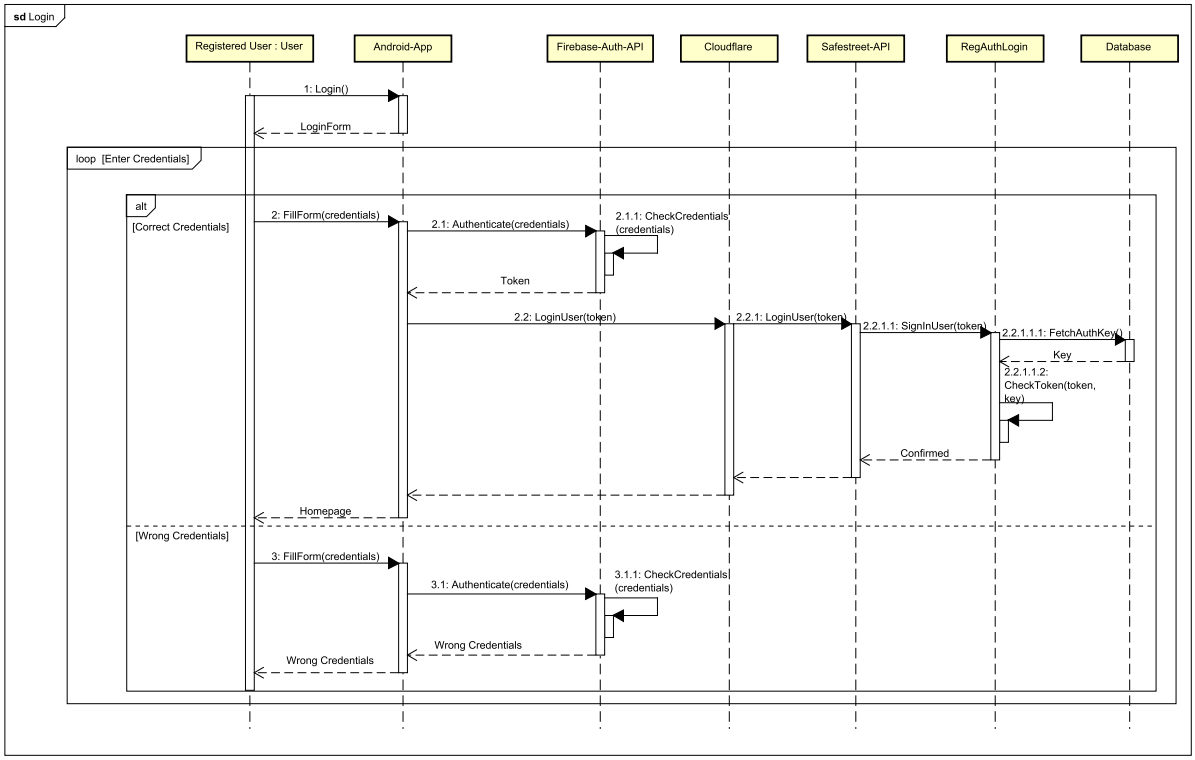


Figure 4: Verification Runtime View



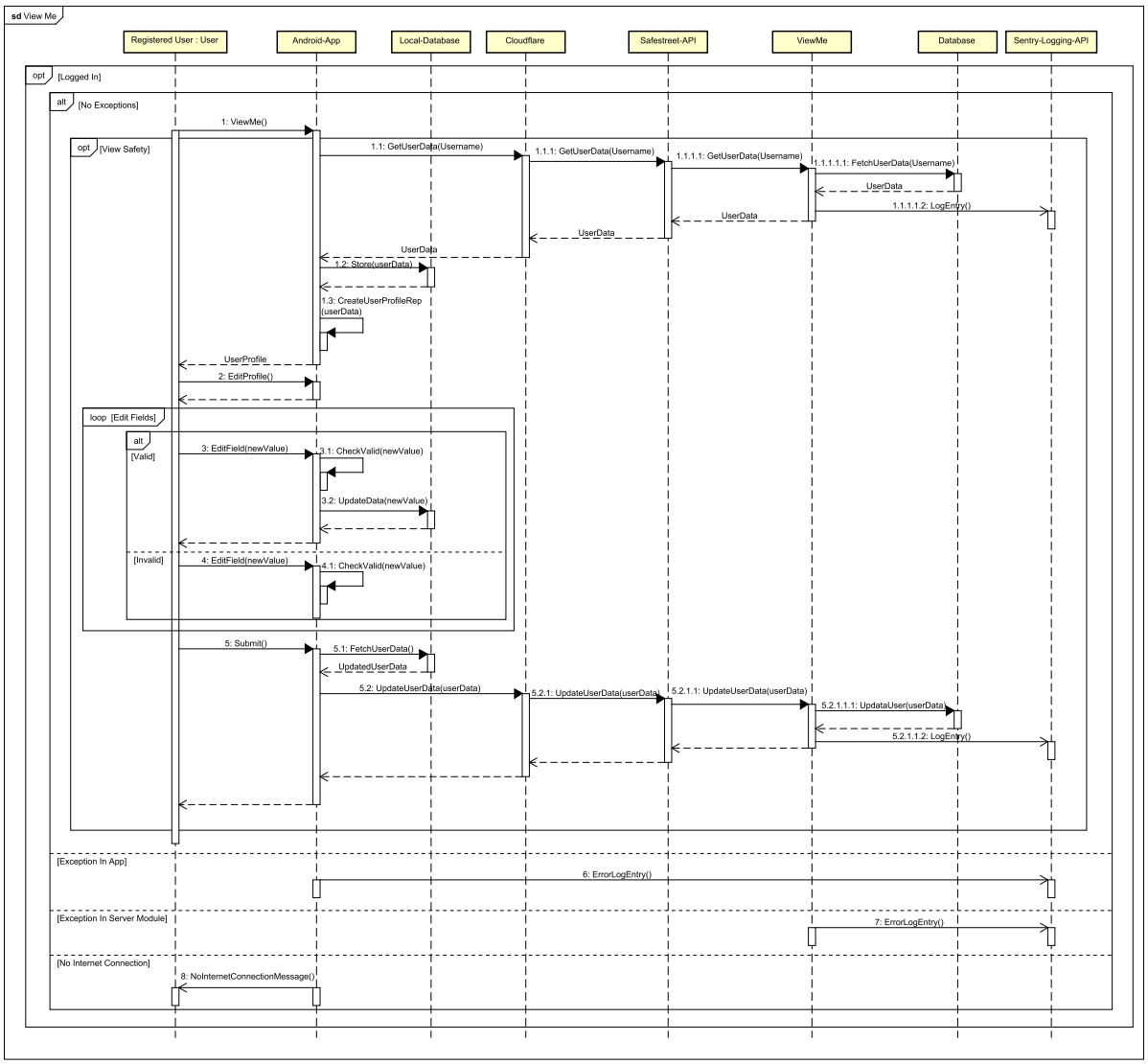
The below figure describes the user login process. Which again, relies on the *Firebase-Authentication* server to authenticate the user credentials and issuing a token to be sent to the app server to confirm the authentication process.

Figure 5: Login Runtime View



The last of the general functions in the figure below is the *ViewMe* functionality. Where the user can view their profile and edit their personal information.

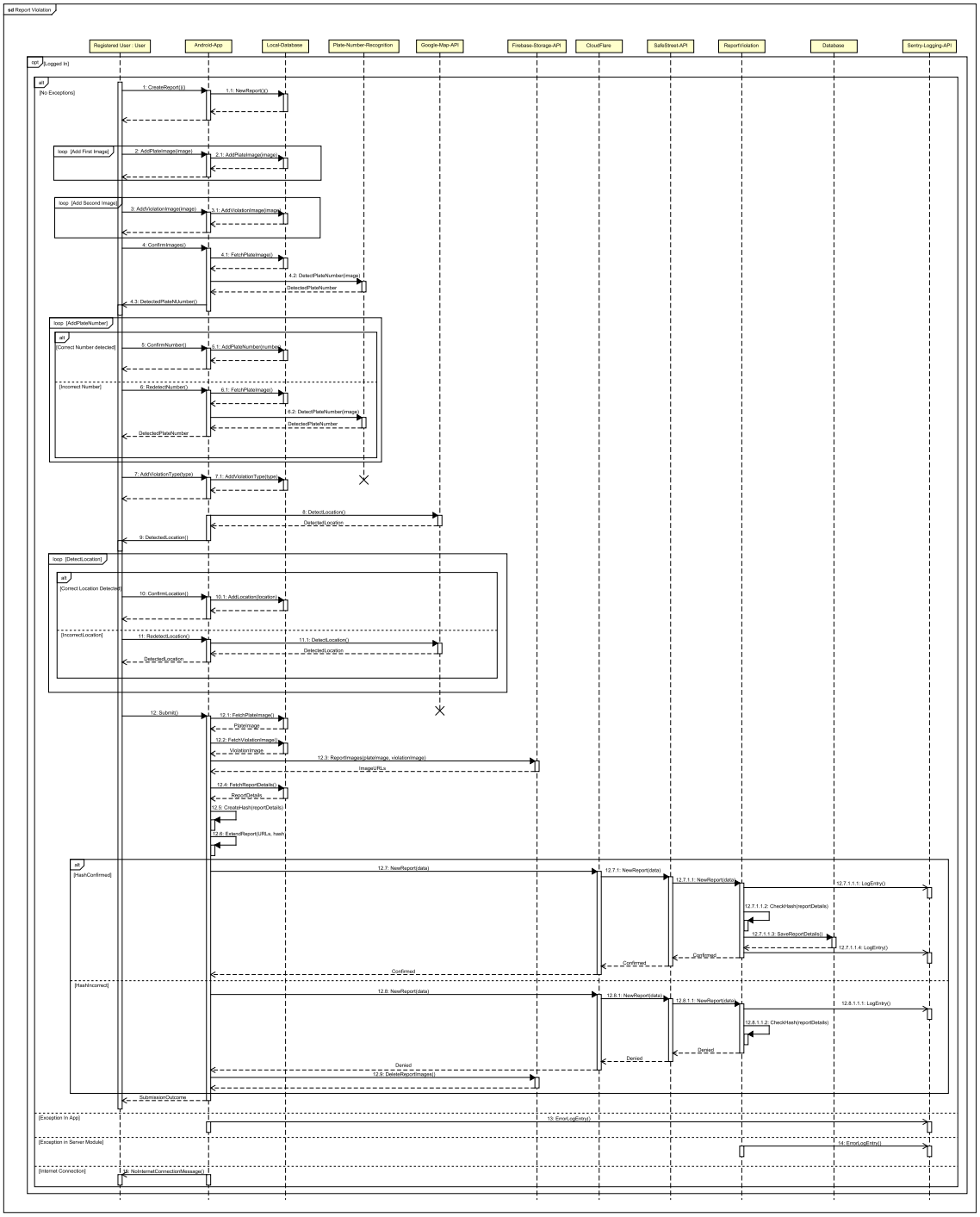
Figure 6: ViewMe Runtime View



2.4.2 Violation Reports

The process of *Violation Report* submission performed by the users is constituted of the user report formulation including images showing the violation, the location, and other useful information. The images are saved on the *Firebase-Storage* and their respective URLs are used to access them for any future use. Finally, the report details, image URLs and a computed hash which is verified to ensure data integrity are sent to the server. A detailed breakdown of the step-by-step process of violation report submission is described in the following figure.

Figure 7: Report Violation Runtime View



The report history function described in the following figure is a function offered to the user to search through reports that were previously submitted by them according to criteria that they define. Such as, reports that were submitted in a certain time period or a certain type of violation reports.

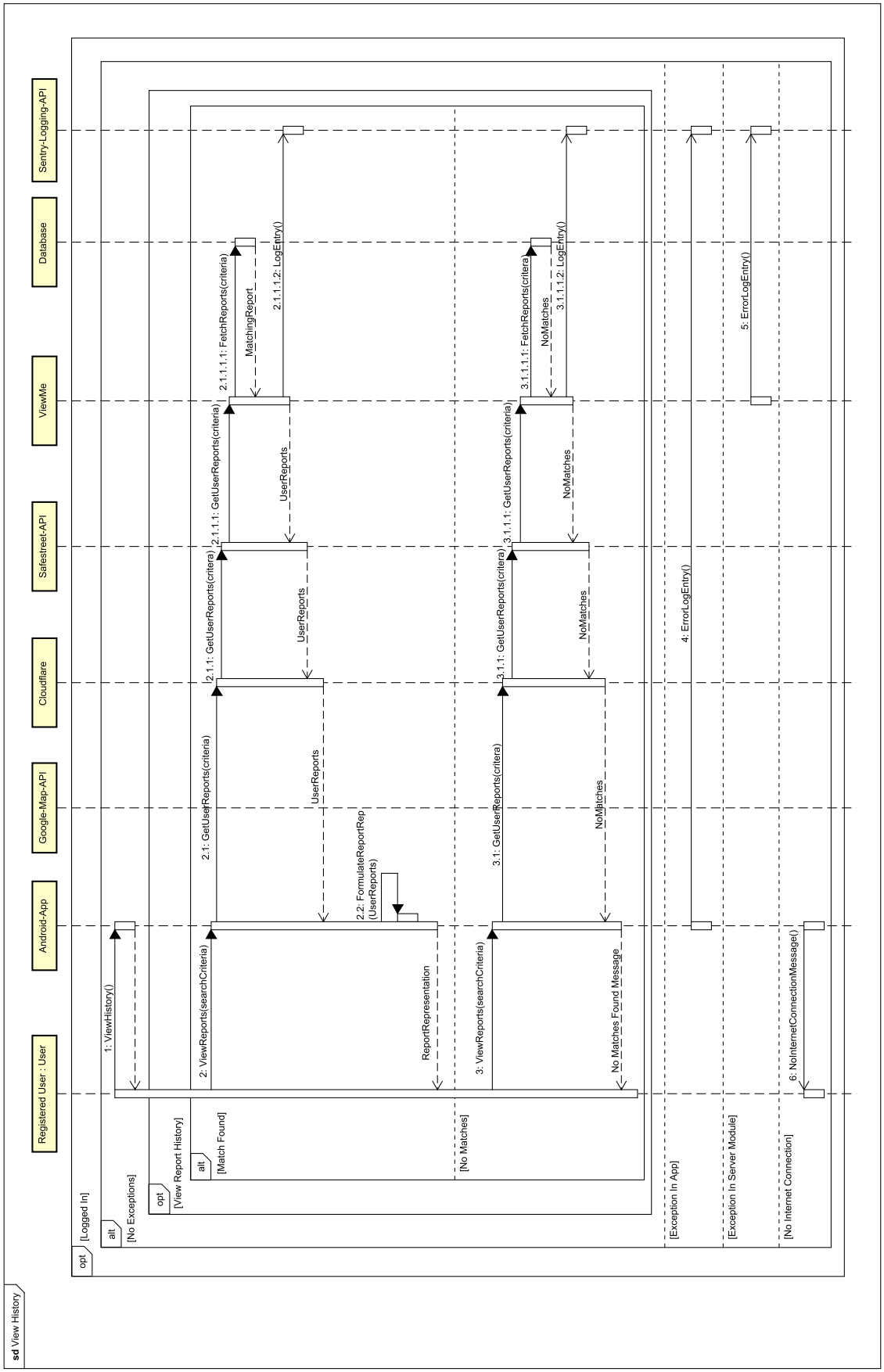


Figure 8: Report History Runtime View

2.4.3 View Safety

This subsection discusses one main feature of the *Safestreets* that is the *View Safety*. This feature entails the graphical representation of the safety of the various areas according to the user-specified search criteria. Where the user would search for a certain area and the app would retrieve report data of both violation reports filed by the users and accident reports retrieved from the municipality and formulate a geographical representation using the *Google Map API* as can be seen in the figure below.

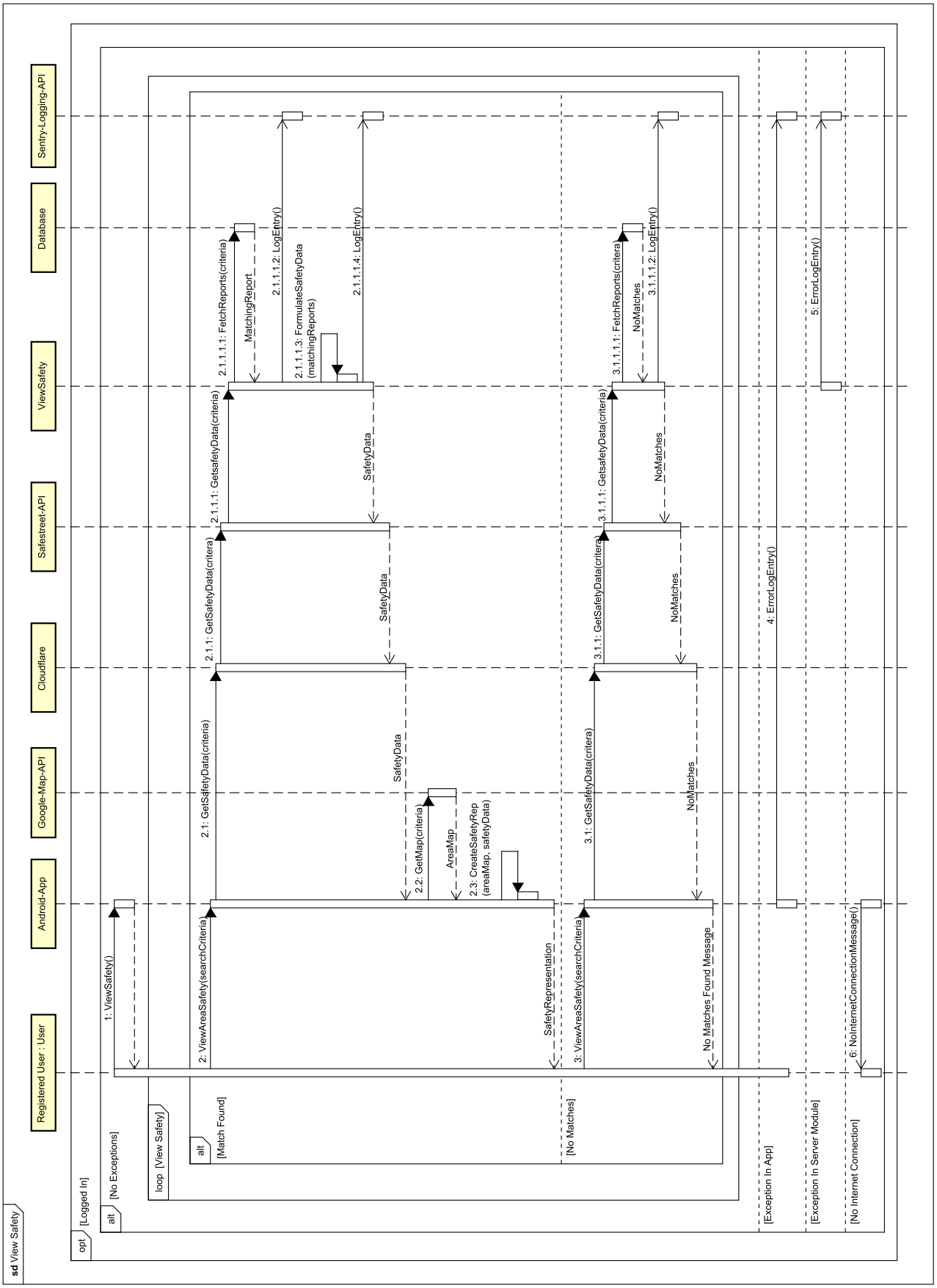


Figure 9: View Safety Runtime View

2.4.4 Municipality Interaction

This last subsection is concerned with the various interaction between the *Safestreet* system and the municipality. To be precise, there are kinds of interactions between the system and the municipality; which are as follows, firstly, the communication of user-generated violation reports to the authorities through periodic submissions to the municipality submission API. Second of the three features, the retrieval of accident reports from the municipality retrieval API to be used in the *View Safety* feature and in the intervention suggestion which is the last feature described in this subsection. In the intervention suggestion process, the user-generated reports and the retrieved accident reports are processed together to gain insight into the traffic violations which are causing a high accident rate in various areas and suggest appropriate measures to resolve these issues. The following figures provide a more detailed description of the aforementioned processes.

Figure 10: Municipality Report Submission Runtime View

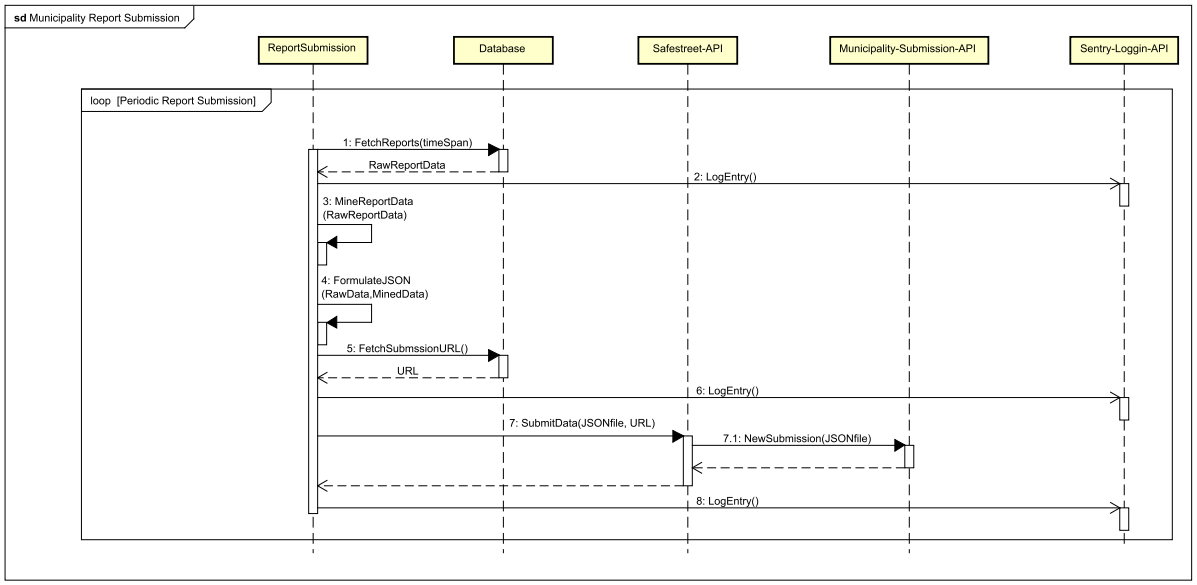


Figure 11: Accident Report Retrieval Runtime View

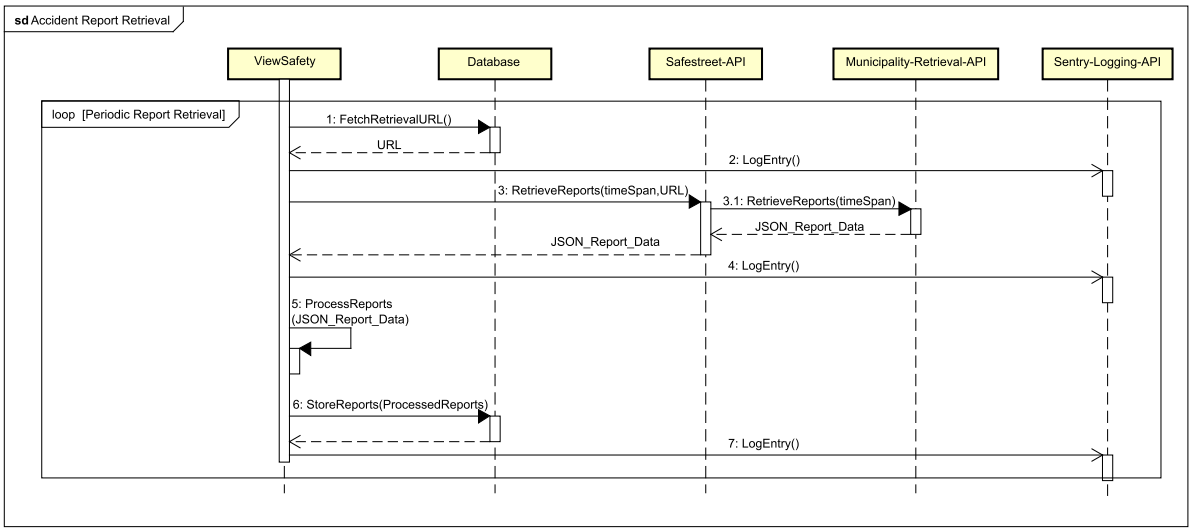
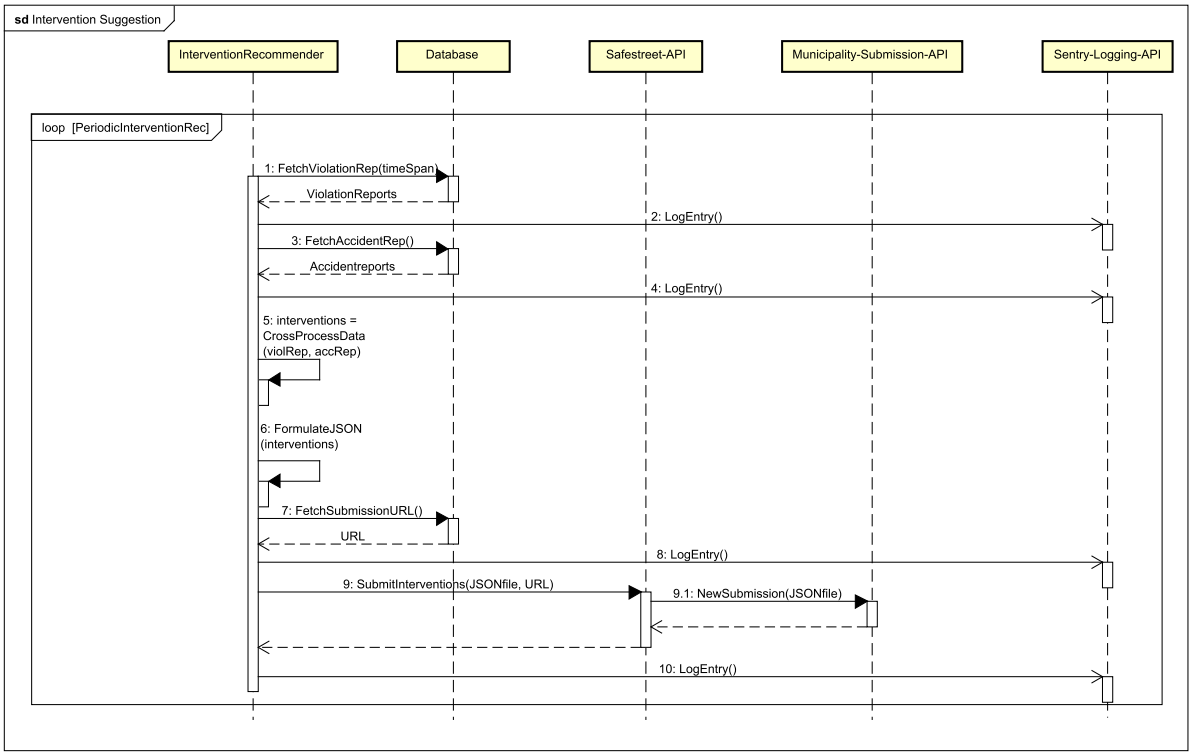


Figure 12: Intervention Suggestion Runtime View



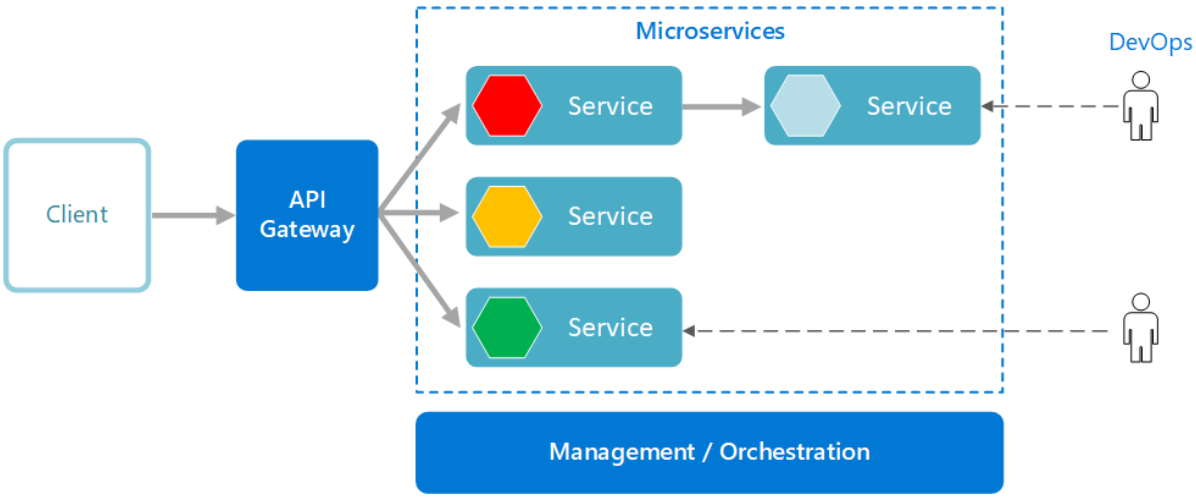
2.5 Selected Architectural Styles And Patterns

In the following section, the selected architectures and design patterns for the implementation phase of the system shall be discussed. Due to nature of the system as a whole and of the services expected to be provided to the users of the system; particularly, the fact that the system provides multiple services which are only connected through the data being produced and consumed by each of them, the system design decisions provided in this section are mostly due to the clear need for decoupling of the various services among each other and their decoupling from the client-side of the system. To be more precise, this system shall adopt the *Microservices Architecture* and the *MVP* design pattern, with the use of *RESTful APIs*. In the hereinafter subsections these design decisions shall be explored.

2.5.1 Microservices Architecture

The first of the design decisions to be implemented is the *Microservices* architectural style which subdivides the system into smaller services each of which has limited interaction with other services and provides a unique service to the clients. It is evident that this architectural style perfectly fits the previously described nature of the system. This architectural style not only provides decoupling of the services from the user but also among themselves. Moreover, the architecture at hand is rather scalable with the respect to the number of users since multiple instances of the various microservices may be instantiated to serve the users which is rather simple due to their stateless nature. Below is a figure representing the *Microservices* architectural style, in which the previously discussed aspects of the architecture can be seen.

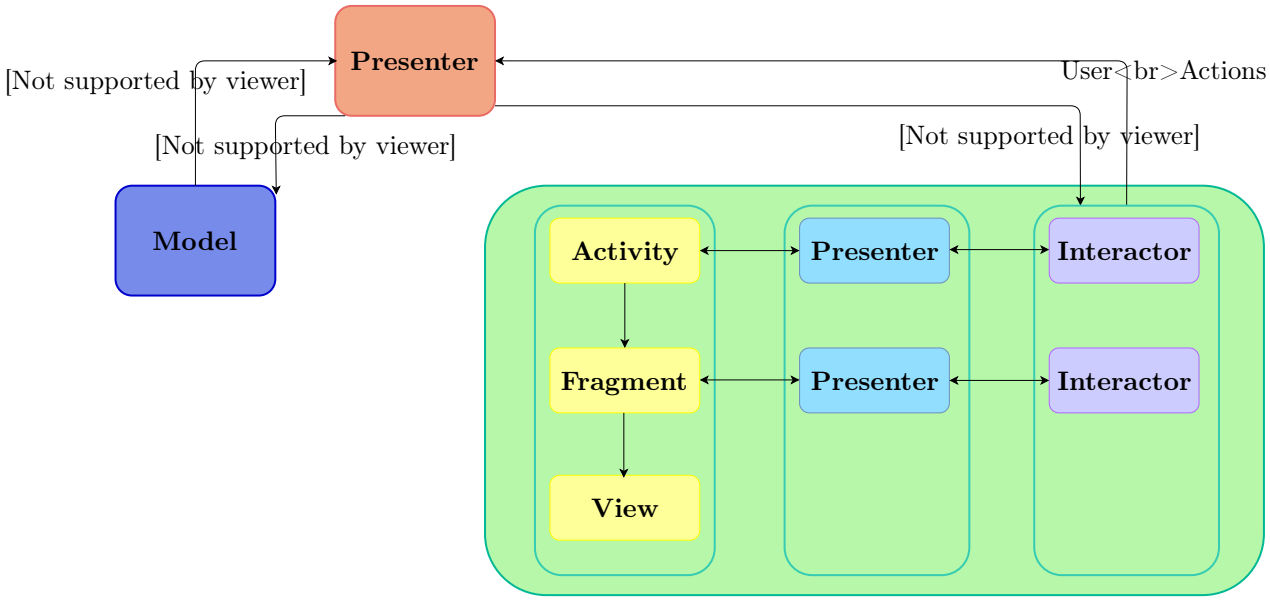
Figure 13: Microservices Architectural Style (Microsoft Azure 2019)



2.5.2 Model View Presenter

In this subsection the design pattern to be used in the system implementation shall be discussed; in particular, the ***M**odel **V**iew **P**resenter (MVP)* design pattern. This design pattern subdivides the system into three general segments; the *Model* representing the business logic and the data, the *View* which interacts directly with user, and the *Presenter* which has a two way interaction with both the view and model; presenting data from the model to the View and manipulating the model based on the user interaction with view. By definition, the *MVP* completely decouples the View and the Presenter and their communication is completely through interfaces; which brings us to the third of the decisions to be discussed in this section later on. The following figure shows the interaction between the system sub-parts as defined by the *MVP* pattern.

Figure 14: MVP Design Pattern



2.5.3 RESTful APIs

This third and last subsection is concerned with the use of *RESTful APIs* for the interaction between the client and server sides. These APIs can be viewed as the final piece of the puzzle that is the system architecture as a whole. Since the nature of the *RESTful APIs,* falls so perfectly into the system architecture described up to this point. Some of the main properties of these APIs which prove to be rather relevant to this architecture is the fact that they are stateless and flexible in the sense that the data is tied to neither resources nor methods enabling simple scaling with concurrent calls. Therefore, these aspects enable *RESTful APIs* to be a perfect fit for the role of interfacing the View and the Presenter described in the previous subsection.

3 User Interface Design

In the following page, the UX diagram is presented in the form of a Wireframe. The Wireframe is a specific type of UX representation which demonstrates significant abstract outline about the application and indicates all the flows that user can experience. Taking a closer look at the diagram, each user flow corresponding to a specific macro functionality of the system is represented with a unique color. The various flows represented in the diagram are to now be discussed. The light green flow has to do with the user violation report main functionality. Functionalities such as login and sign up, and sign out are represented by the pink and turqoise flows respectively. The view history, view profile and app info are depicted by the light blue flow which flows from the homepage (report violation) through the menu to each of the app sections. Lastly, as is illustrated by the lilac and orange flows the user may return to the report page from each of the app's main sections. Finally, it should be brought to your attention that the specific mockups of the user interfaces are included in the *Requirement Analysis and Specification Document (section 3.1.1.*

Figure 15: Entity Class Diagram

4 Requirements Traceability

This section shall present a traceability matrix to ensure the satisfaction of all functional requirements and therefore the satisfaction of the underlying goals which are expected of the system. The *Requirements traceability matrix(RTM)* presented below shows a correlation between all the functional requirements previously defined in the *RASD* document and the system modules that satisfy and perform these functionalities. Apart from the components mention in the *RTM* some other components were defined in previous sections; though these modules are not direct contributors to the realization of the requirements they do however perform some roles ensuring the smooth and secure operating of the system as a whole.

REQ ID	**REQUIREMENT**	**COMPONENT**
[R-1]	Users should be allowed to register to services provided by the system	*Registration, Authentication and Login-Firebase Authentication API*
[R-2]	Users should provide unique identification to the such as fiscal code during registration	*Registration,Authentication and Login-Firebase Authentication API*
[R-3]	Registered users should be allowed to login	*Registration,Authentication and Login-Firebase Authentication API*
[R-4]	Each registered user should have a unique username used for logging in chosen at registration time	*Registration,Authentication and Login-Firebase Authentication API*
[R-5]	System should enable registered users to report traffic violations	*ReportViolation*
[R-6]	When reporting a violation, users should be able to take an image of the violating vehicle's license plate	*Android App- Firebase Storage API*
[R-7]	When reporting a violation, users should be able to fill in the details of the reported violation such as the type of the violation	*Android App*
[R-8]	The system should be able to detect the current user location when reporting a violation	*Android App-Google Map API*
[R-9]	The system should extract the plate numbers from the image taken by the user	*Plate Number Recognition*
[R-10]	The received reports must be stored by the system to be used by other services	*Report Violation-Database*
[R-11]	Registered users should be allowed to view a representation of the safety of selected areas possibly with the help of a map API	*Android App-Google Map API-View Safety*
[R-12]	The system should implement a means to measure the safety of various areas based on reported violations in said areas	*View Safety*
[R-13]	Incoming reports should be integrated and used to update the safety of areas	*View Safety-Report Violation*

[R-14]	If accident reports are provided by authorities the system should take that data into account when calculating the safety of a certain area	*View Safety*
[R-15]	The system should present on-demand to the users a record of all the reports previously submitted by them	*Report Historyy*
[R-16]	The system should keep records regarding the submission dates of violations to the municipality	*Report Submission-Database*
[R-17]	The system should aggregate the data regarding reported traffic violations since the last submission to the municipality	*Report Submission-Database*
[R-18]	The aggregated traffic violation data should be converted to a form acceptable by the municipality interface	*Report Submission*
[R-19]	The system should periodically submit the new traffic violation data to the municipality interface	*Report Submission-Database*
[R-20]	The system should be able to store submitted reports coming from users with proper metadata	*Report Violation-Database*
[R-21]	The system should be able to export reported violations in form of specific file	*Report Submission*
[R-22]	The system should be able to filter data based on desired requests from authorities	*Report Submission*
[R-23]	The logging functionality has to be implemented in the system	*Sentry Logging API*
[R-24]	The system should extract insights from the users' traffic violation reports such as the most frequent types of violations in certain areas	*Report Submission-View Safety*
[R-25]	The system should be able to decide on appropriate interventions to minimize frequent traffic violations in the various areas	*Intervention Recommender*
[R-26]	The system should formalize the interventions to be suggested in a form acceptable by the municipality interface	*Intervention Recommender*
[R-27]	The system should submit the interventions regarding the areas with a high frequency of violations to the municipality interface	*Report Submission*
[R-28]	The system has to be able to mine reports to find insights based on types of violation and different areas	*View Safety*
[R-29]	The system should analyze the data from the reports to produce statistics such as the most frequent plate numbers that commit violations	*Report Submission*
[R-30]	The system should formalize the refined data and submit them to the municipality interface periodically	*Report Submission*

Table 2: Traceability matrix

5 Implementation Integration And TestPlan

6 Effort Spent

Discription of the Task	**Hours**
karim Zakaria Saloma	
Introduction	5
Architectural Design	7
User Interface Design	7
Requirements Traceability	2
Implementation, Integration and Test Plan	3
Amirsalar Molaei	
Introduction	5
Architectural Design	7
User Interface Design	7
Requirements Traceability	2
Implementation, Integration and Test Plan	3
Erfan Rahnemoon	
Introduction	5
Architectural Design	7
User Interface Design	7
Requirements Traceability	2
Implementation, Integration and Test Plan	3

Table 3: Effort Spent by Each Team Member.