POLITECNICO
MILANO 1863

SafeStreets

Software Engineering 2 Project - Prof. Matteo Rossi

DD Document

Salvatore Fadda - 944786
Adriano Mundo - 944684
Francesco Rota - 948714

A.Y. 2019/2020
Version 1.0

December 9, 2019

Contents

1 Introduction

1.1 Purpose

This document represent the *Design Document* (DD). It aims at providing an in-depth description of the architecture below *SafeStreets* application and its services. It will present a section related to the architectural design with different perspective on the components of the *System*, how they interact and how they will be implemented. All the requirements of the RASD document are mapped with the components to explain how they will be satisfied. Finally, a section for the testing plan for Q&A team is provided.

1.2 Scope

SafeStreets is a crowd-sourced application that aims at keeping safe the city's streets. The idea behind this service is to allow *Users* to notify the *Municipality* when a violation occur on the streets under its jurisdiction. The *User* can notice and notify the violation by sending a photo of the violation including date, time and position. *SafesStreets* stores all the data and uses a plate recognition algorithm to recognise the image content.

The **Basic Service** allows *Users* and *Authorities* to mine the information collected by the service, so they can access statistics built from the data.

As **AF1**, the application *SafeStreets* identifies potentially unsafe areas and suggests possible interventions to *Authorities* to solve the founded issues.

As **AF2**. the application *SafeStreets* allows the *Municipality* to generate traffic tickets directly from the application data. Also, using the data of issued tickets the *System* can build statistics and find insights to suggest to *Municipality* in order to improve their service.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Client:** a piece of computer hardware or software that accesses a service made available by a Server.
- **Server:** a computer program or a device that provides functionality and handle the requests of other programs or devices, called Clients.
- **N-Tier:** or multilayer is an architecture in which presentation, application processing, and data management functions are physically separated in n-layers.

1.3.2 Acronyms

- **GPS:** Global Positioning System
- **API:** Application Programming Interface
- **RASD:** Requirements Analysis and Specification Document
- **DBMS:** Data Bases Management System
- **GDPR:** General Data Protection Regulation
- **MVC:** Model-View-Controller
- **REST:** REpresentational State Transfer
- **Q&A:** Quality and Assurance
- **UI:** User Interface
- **HTTPS:** Hyper Text Transfer Protocol Secure

1.3.3 Abbreviations

- **[Rn]:** n-th Functional Requirement
- **A1:** Advanced Function One
- **A2:** Advanced Function Two

1.4 Revision history

Version	Date	Description
1.0	09/12/2019	First Delivery

Table 1: Revision History

1.5 Reference Documents

- Mandatory Project Assignment
- RASD Document of *SafeStreets* application

1.6 Document Structure

The other sections of the Design Document (DD) are organised in this way:

- **Architectural Design** (Section 2): an in-depth description of the System's architecture. It defines the main components, the relationship between them and the deployment of components. There are different views and levels of analysis of the components plus some subsection useful for identifying how the components interact and the architectural styles and patterns.
- **User Interface Design** (Section 3): a complementary section of what was included in the RASD. It includes the definition of the UX process through a model that represents the flows of the interfaces.
- **Requirements Traceability** (Section 4): a complementary section of what was included in the RASD. It contains all the identified requirements and show the relationship between them and design choices in order to satisfy them.
- **Implementation, Integration and Test Plan** (Section 5): shows the order of the implementation and integration of all the components and subcomponents, providing how the application will be tested.
- **Effort Spent** (Section 6): a section containing a table for identifying the hours and the effort spent by each team member to deliver the DD.

2 Architectural Design

2.1 Overview

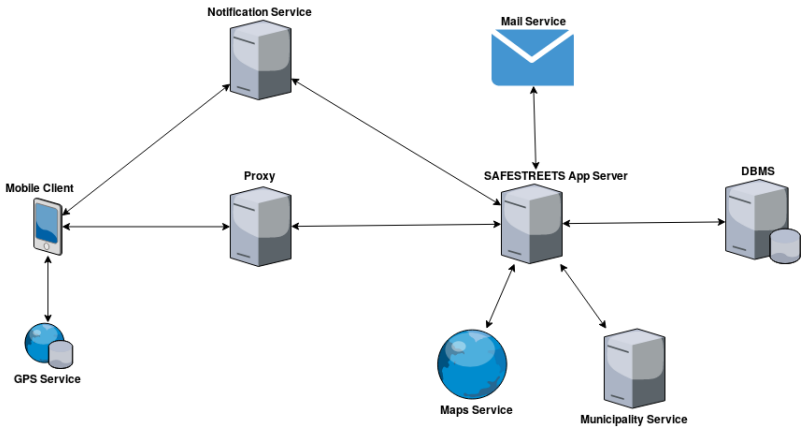

Figure 1: Overview of the System

The image above shows an high-level overview of the *System*'s architecture. The components interact with some external services.
The Mobile Client application accesses the GPS service in order to retrive geographical information and communicates with the Application Server through a Proxy.
The Application Server uses an external Notification Service to send notifications directly to the Client. It uses a Mail System service and a Map service to execute all the functions. Finally, it accesses the Municipality Service to retrieve data and to communicate with the Municipality through the an API service offered by the Municipality itself.
Further details on the *System* components will be explained in the next sections.

2.2 Component View

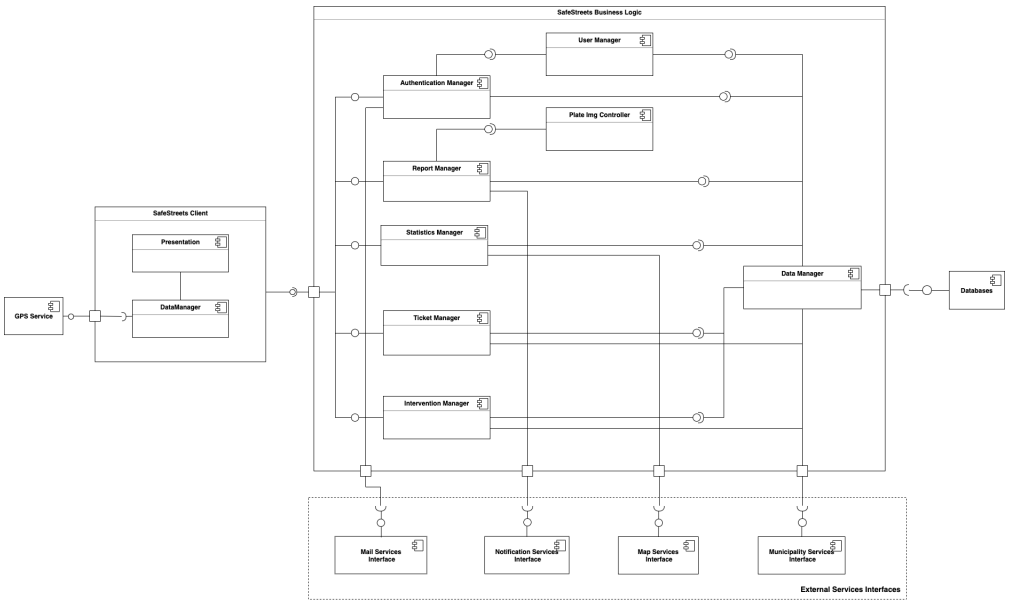

Figure 2: *SafeStreets* Component Diagram

The purpose of the UML component diagram is to capture the internal architecture of the *System*, showing the structure pf components, how they are connected together as a part of larger components.
It's divided in three main components: the *SafeStreets* Client Application, the Business Logic and the external Database.
Each main component is divided in modular components to provide specific functionalities. Components communicate between each other by providing interfaces with the required information.
This diagram uses assembly connectors for internal interfaces and delegation connectors for external interfaces.

SafeStreets Client
This component located on the *User*'s device represents the client machines that access to the Business Logic container. Its sub-modules are the Presentation and Data Manager. It is implemented as a thin-client, so does not contain any logic of the *System*.

- **Presentation:** this module corresponds to the View of the MVC Pattern, requires data from the Business Logic to display the correct UI to the *User*.
- **Data Manager:** this module has to communicate with the GPS Service in order to retrieve the data required to the application.

SafeStreets Business Logic
This component contains all the *SafeStreets* application logic. It's a module between the Client Application and the Database. It collects together all the components needed to satisfy the application functionalities. In the section below each component will be described.

- **Authentication Manager:** this component contains all the methods needed to access the application, so for the purpose of the system authentication. It's responsible for both the registrazione and log-in process. It continuously interacts with the Database through the Data Manager interface. It guarantees that all the constraints are respected and finally, when you need to confirm to the *Users* they have successfully registered, it communicates with the Mail Service Interface to send the confirmation e-mail.

- **User Manager:** this component handles all the functionalities related to the User account, both for simple *Users* and *Authorities*. Its services are performed by interacting with the Database through the Data Manager.

- **Report Manager:** this component contains the logic needed to perform and handle the violation reported by the *User*. Its service are linked to the image loading, the violation indication type and so on. It interacts with another component that's responsible for analysing the image, the Plate Img Controller and stores all the data in the Database through the Data Manager. Finally, the Report Manager communicates with the Notification Service Interface because it gives a direct feedback to the *User* about the loaded image through a push notification.

- **Plate Img Controller:** this component handles the request coming from the *User* that wants to load a violation image. The request comes from the Report Manager, in fact the component exposes the methods to verify that the image was not modified by third parties, so it makes a consistency check. In order to do this task, the component runs an algorithm.

- **Statistics Manager:** this component manage and calculate all the statistics that the application is intended to provide to both *Users* and *Authorities*. It has two main task: retrieve and store the data from the Database through the Data Manager, where are stored all the data inserted from the application's users; and to calculate all the statistics that are accessible from the UI. It manipulates all the data in order to show simple graphs or to find useful insights. It accesses to Map Services Interface because it needs to retrieve geographical information in order to show statistics directly associated with the city's zone.

- **Ticket Manager:** this component is responsible for the ticket verification by the *Authorities* thanks to reported *User* violation data. In fact he retrieve from the Database through the Data Manager all the violation reported from the *Users*, then let the verification to be done manually by

an *Authorities*, exposing all the needed functions. Finally, it communicates with the Municipality Service Interface in order to advise that the ticket is verified. Therefore the Municipality is able to generate a ticket from this information.

- **Intervention Manager:** this component handles the information coming from the Municipality, in fact it interacts with the Municipality Service Interface to retrieve data about accidents. Then, it cross the information with the data stored in the Databases and retrieved through the Data Manager in order to provide suggestions on interventions.
- **Data Manager:** this component provides all the methods to interact with the Database such as data retrieval, storage and update, it is the unique point of access to the Database.

Data Base
This component represents the DBMS, which provides the interfaces to retrieve and store the data. Data about the application and data about Users are securely stored and encrypted.

External Services Interfaces
Some components of the diagrams that represent the *SafeStreets System* and described above communicate with external components. This are are third party services that expose their API. These communications are bilateral and essential to guarantee all the functionalities.

- **GPS Service:** this interface communicates with the Client Application that have to access data from the *User*'s device, in particular GPS data through the Data Manager interface. GPS data are essential for tracking position and retrieve all suitable metadata necessary to the application to work properly.
- **Mail Services Interface:** this interface is responsible for the interaction with the *User* when it's sent a confirmation e-mail after the registration phase. It's accessed by the Authentication Manager.
- **Notification Services Interface:** this interface is needed to send push-notification to the *User* as an alert when the image processing is not done correctly, so it needs to communicate with the Report Manager in order to advise the *User* that have to re-do the process of reporting.
- **Map Services Interface:** this interface is essential for *SafeStreets*, in fact access to a mapping service is necessary to show and calculate statistics with respect to the different city's zone. It's a service accessed by the Statistics Manager.
- **Municipality Services Interface:** this interface is essential to have a means of communication with the Municipality services. It lets to retrieve data and to pass information to the Municipality in order to do all the services offered by the *System*. It's accessed by Ticket, Intervention Mngs.

2.3 Deployment View

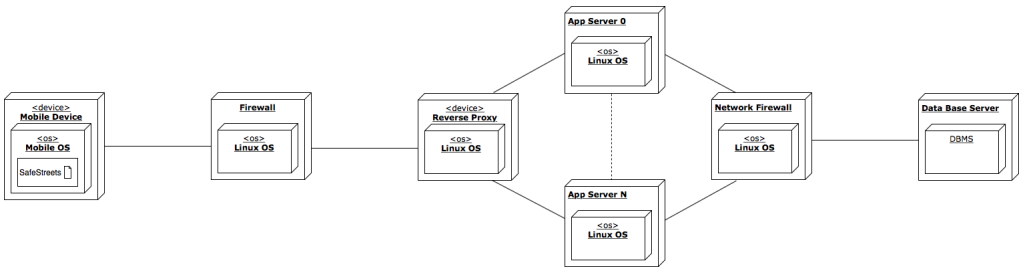

Figure 3: *SafeStreets* Deployment Diagram

The *System* presents a Multi-tier architecture. The role of each node will be specified in the next section.

Mobile Device
It represents the Client in the architecture, hosting the System's mobile application. This is equally valid for both Users and Authorities.

Firewall
It filters the access to the Reverse Proxy and is used to protect a trusted network from an untrusted network. A firewall provides protection from unauthorised requests or from malicious attacks.

Reverse Proxy
It retrieves resources on behalf of a client from the servers and balances the load of the various requests. It helps to achieve increased parallelism and scalability of the application.

Application Servers
They include all the business logic of the system, which is completely replicated to allow workload balancing.

Network Firewall
It does the same job of the Proxy Firewall but protecting and filtering the access to the DBMS.

Data Base Server
All data are stored in the Data Base Server equipped with a relational DBMS and can be retrieved with appropriate queries.

2.4 Runtime View

Account Creation Runtime View

The first sequence diagram describes the order of the events that occurs when a *Guest* tries to register as a *User* of the *SafeStreets* application. The actors involved in this scenario are the *SafeStreets* mobile application on the Client side, while on the Server side the Authentication Manager, the User Controller and the Data Manager. The external component are the Database and the Mail Service Interfaces.

The *Guest* selects the register button and fill the the registration form, then the request is validated, if not there's an alter. If the validation process is done correctly it's created a new *User*, inserted in the Database and e-mail confirmation with a code is sent to the new *User*.

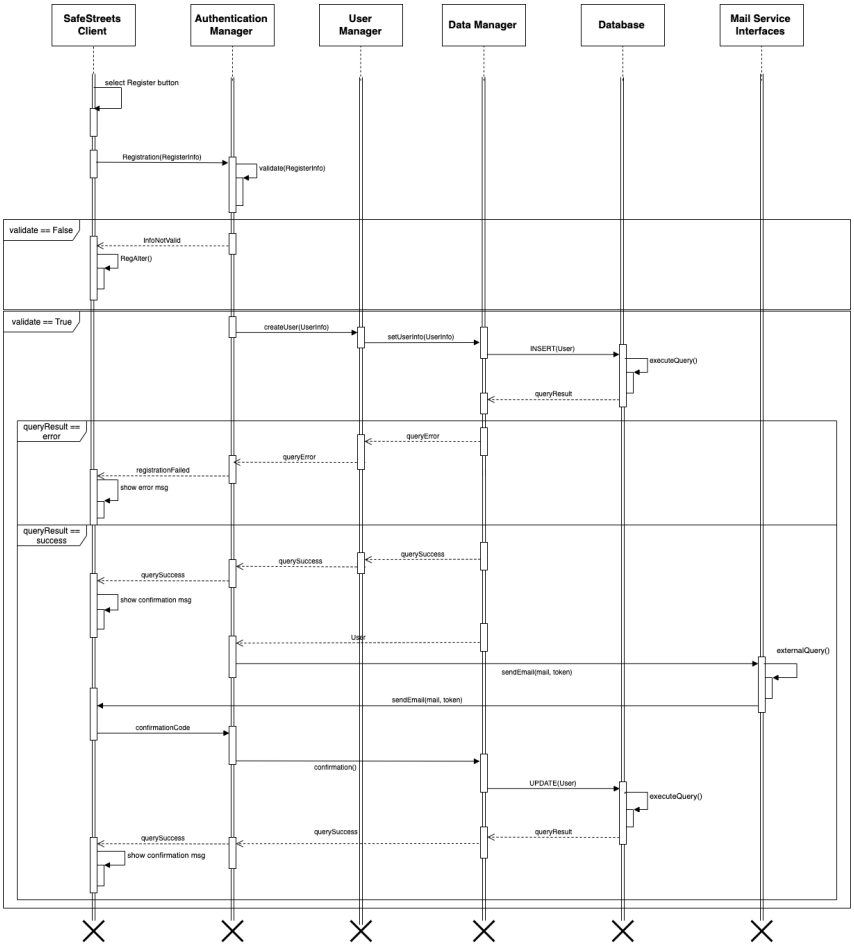

Figure 4: *SafeStreets* Account Creation Runtime View

Log-in Runtime View

The second sequence diagram describes the order of the events that occurs when a *User* or *Authority* tries to log-in to the application. The actors involved in this scenario are the *SafeStreets* Client and on the Server side the Authentication Manager and Data Manager. The external component is the Database.
The *User's* application selects the login tab and fill the the form with credentials, then the request is validated, if not there's an alter. If the validation process is done correctly it's created an access token and the *User* obtain access to the application.

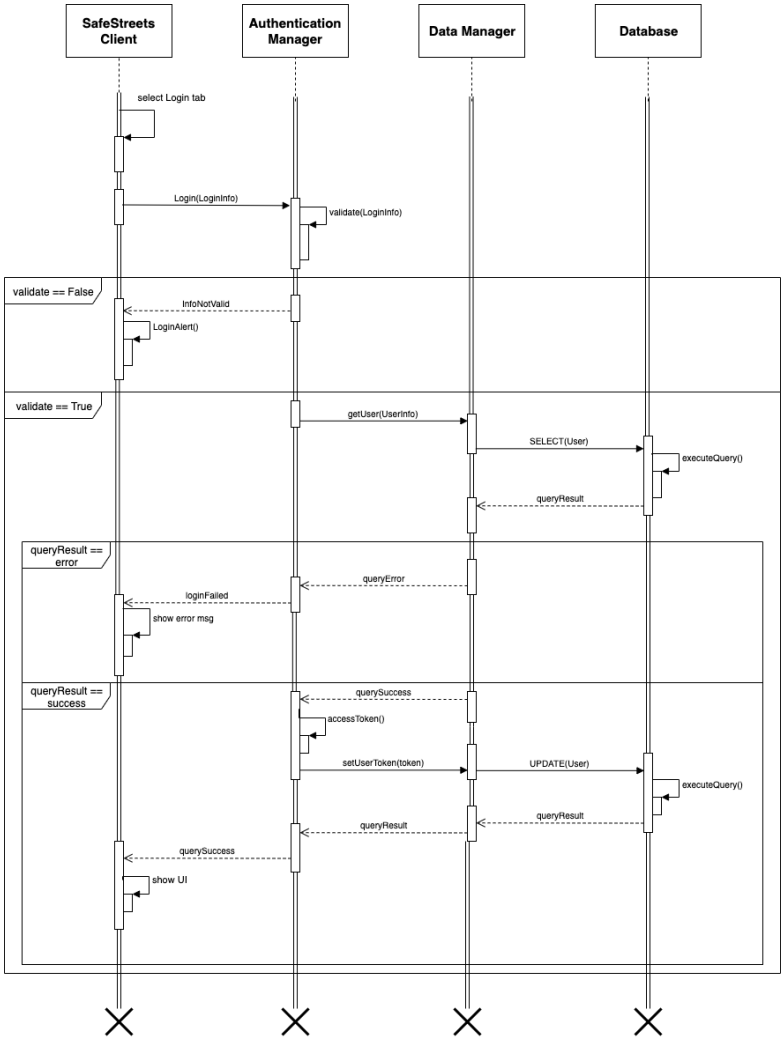

Figure 5: *SafeStreets* Log-in Runtime View

Report Violation Runtime View

The third sequence diagram describes the order of the events that occurs when a *User* tries to report a violation. The actors involved in this scenario are the *SafeStreets* application on the Client while on the Server side the Authentication Manager, Report Manager, Plate Img Controller, Data Manager. The external components are the Database, GPS Service and Notification Service Interfaces. The *User* selects the report tab, the authorisation is checked, then the *User* insert all the data needed for reporting the violation. Position data are retrieved from the GPS Service. Once the *User* has loaded the data, the image is processed by the Plate Img Controller in order to check if the image has been modified or not. If the validation process is done correctly it's created a notification in the Database, otherwise the *User* must re-take the photo and submit again the report.

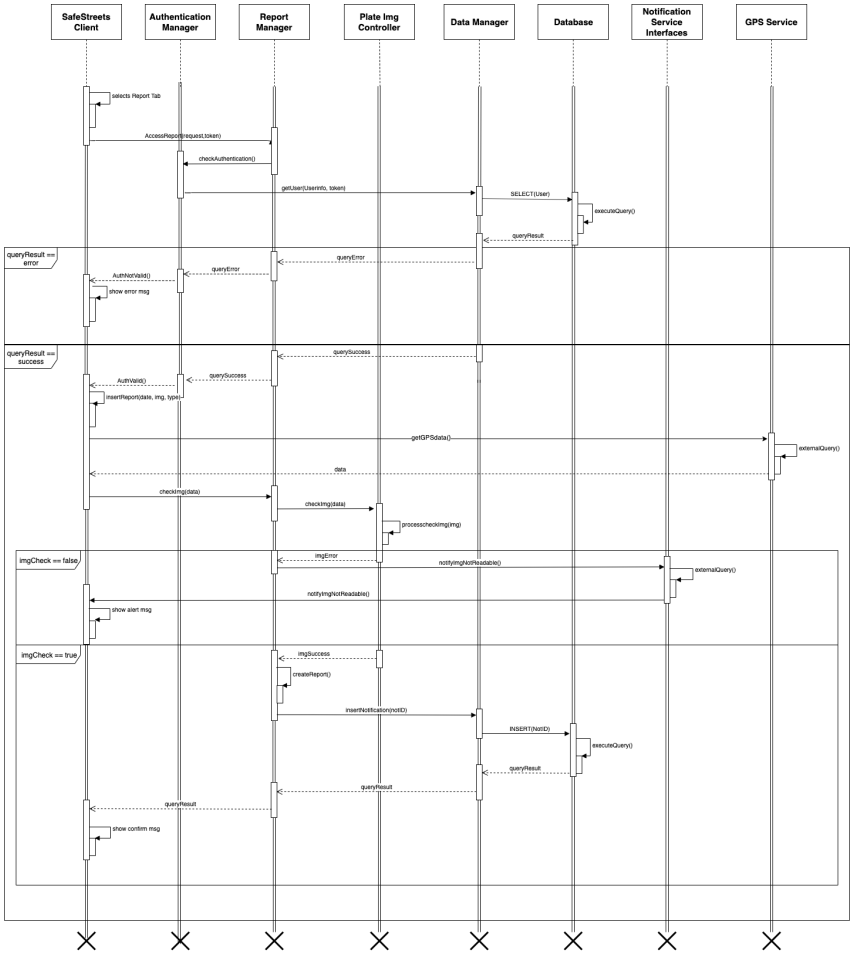

Figure 6: *SafeStreets* Report Violation Runtime View

Maps Access Runtime View

The fourth sequence diagram describes the order of the events that occurs when a *User* or *Authority* tries to access Maps Statistics. The actors involved in this scenario are the *SafeStreets* application on the Client while on the Server side the Authentication Manager, Statistics Manager and Data Manager. The external components are the Database and Map Service Interfaces.

The *User's* application selects the map tab, the authorisation is checked, then the Statistics manager get the Notification Data from the Database and the Map from the external interface. After that it computes the Map Statistics and the Client show the interface to the *User*, based on the visibility level, because *User* and *Authority* can see different data.

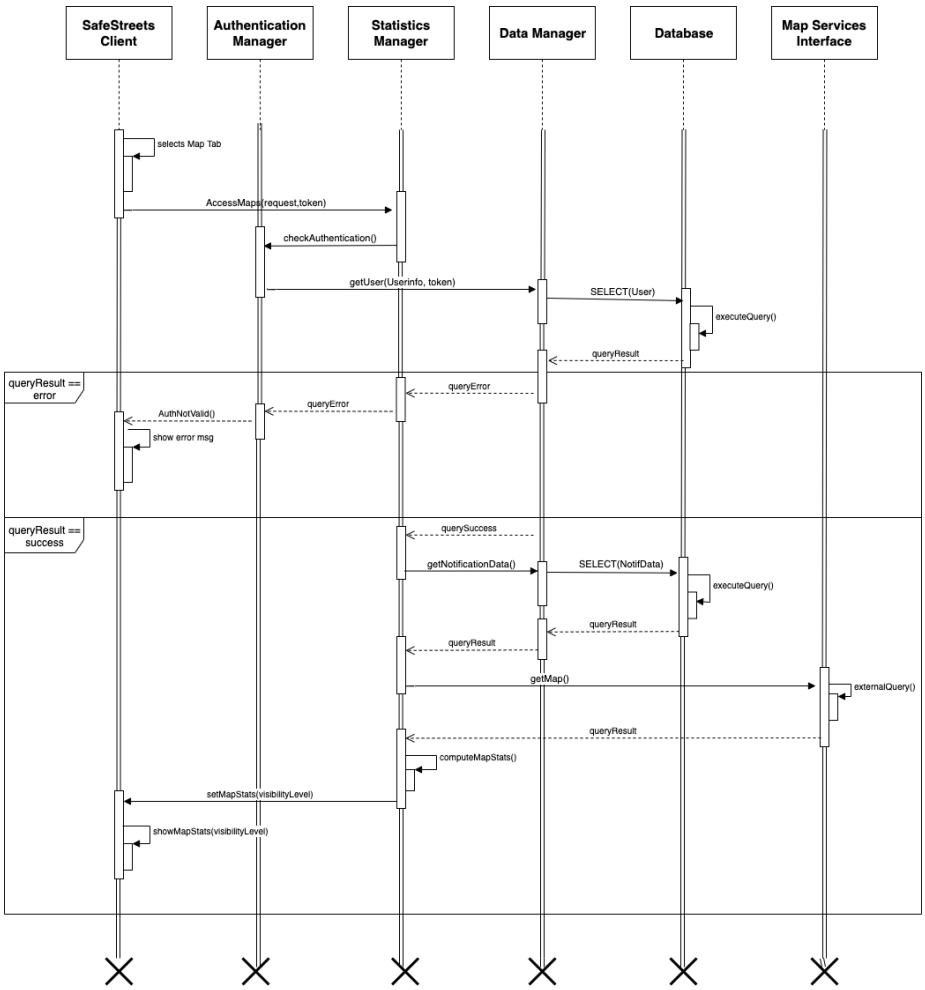

Figure 7: *SafeStreets* Maps Access Runtime View

Statistics Access Runtime View

The fifth sequence diagram describes the order of the events that occurs when a *User* or *Authority* tries to access Statistics. The actors involved in this scenario are the *SafeStreets* application on the Client while on the Server side the Authentication Manager, Statistics Manager and Data Manager. The external components is the Database.

The *User's* application selects the map tab, the authorisation is checked, then the Statistics manager get the Stats Data from the Database and computes the Statistics that must be shown to the *User*. Based on the different visibility level, *User* or *Authority* can see different statistics and insights generate from the notifications and/or tickets data.

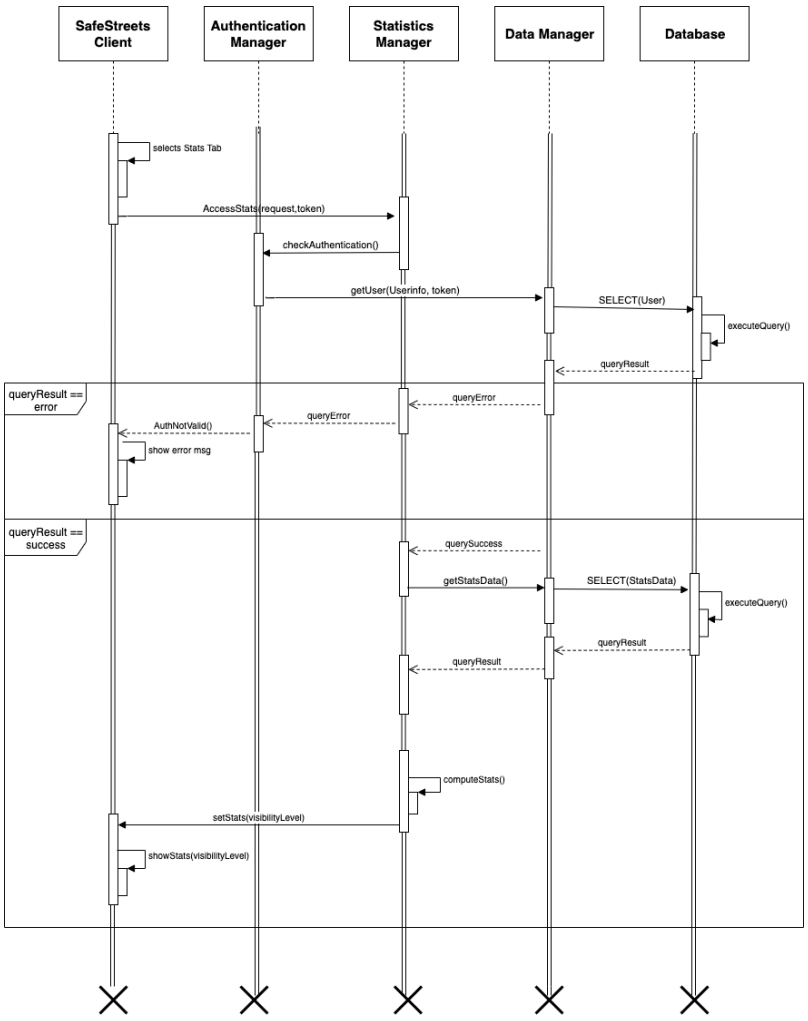

Figure 8: *SafeStreets* Statistics Access Runtime View

Tickets Generation Runtime View

The sixth sequence diagram describes the order of the events that occurs when an *Authority* validates a ticket that could be generated from the notifications' data. The actors involved in this scenario are the *SafeStreets* mobile application on the Client side while on the Server the Authentication Manager, Ticket Manager and Data Manager. The external component are the Database and the Municipality Service Interfaces.

The *Authority* selects the Tickets tab, the authorisation is checked, then the Ticket Manager request the list of the notifications and show them to the User. In order to validate or refuse the ticket the User have to select one of them and confirm or not that is an effective violation. The Municipality has the privileges to check and generate the effective ticket based on the *Authority*'s validation.

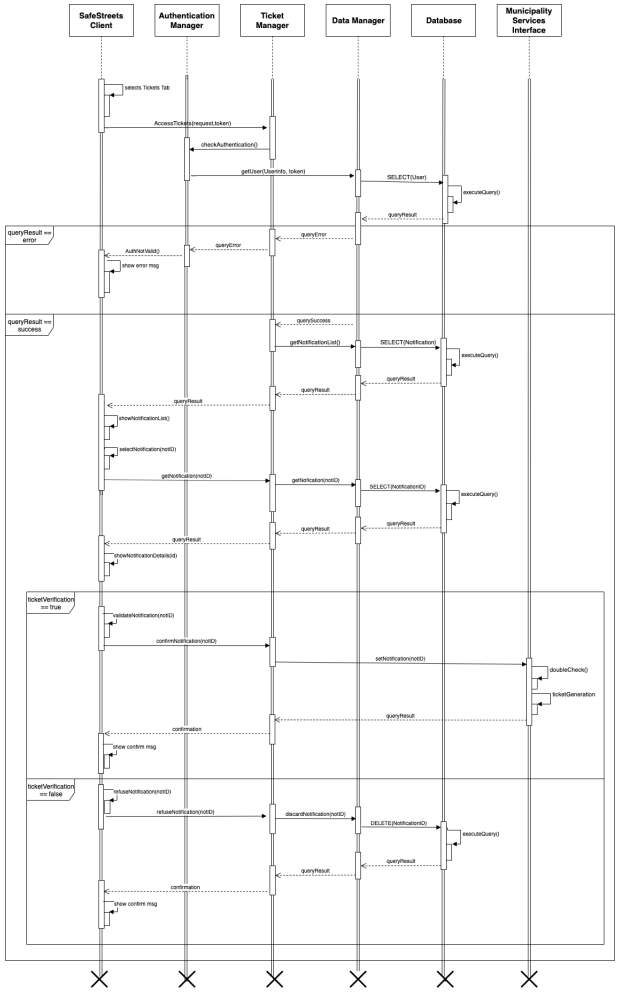

Figure 9: *SafeStreets* Tickets Generation Runtime View

Intervention Suggestion Runtime View

The seventh sequence diagram describes the order of the events that occurs when is suggested to the *Authority* and he/she can confirm the solution as feasible or not. The actors involved in this scenario are the *SafeStreets* mobile application on the Client side while on the Server the Authentication Manager, Intervention Manager and Data Manager. The external component are the Database and the Municipality Service Interfaces.

The *Authority* selects the Interventions tab, the authorisation is checked, then the Intervention Manager request the intervention data from the Database showing the list of the suggestions to the User that has the task to confirm one or more of them as feasible or not. Then the confirmation is sent to the Municipality that starts the procedure to solve the issue.

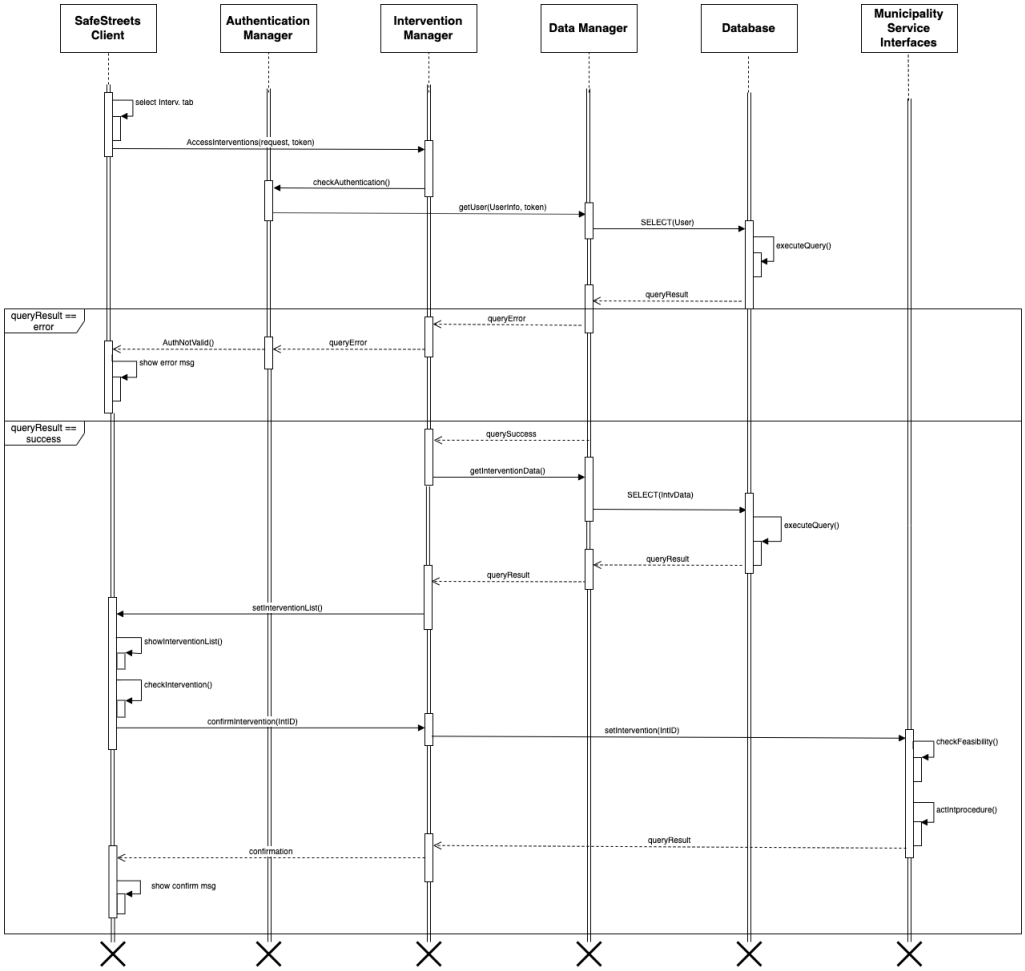

Figure 10: *SafeStreets* Intervention Suggestion Runtime View

2.5 Component Interfaces

Interface Diagram

The diagram below represents the Component View of the *System*, with methods that have been shown in the Runtime View. Some self calls are omitted.

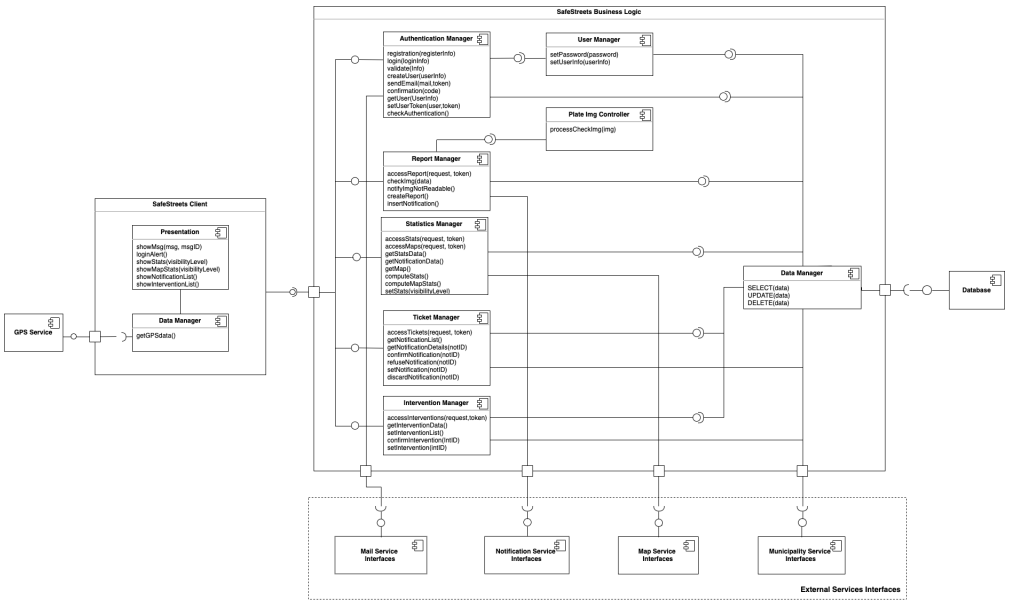

Figure 11: Component Interface Diagram

SafeStreets Client

In this section we will describe all the methods of the Client, in particular related to the Presentation layer.

- **Presentation**
  - *showMsg(msg, msgID)*: this method has to show the alert or confirmation messages to the *User*, using as parameter the type of message and the different id on the basis of the case that has been taken into account.
  - *loginAlert()*: display the alert when the Authentication Manager does not validate the credentials entered to access the application.
  - *showStats(visibilityLevel)*: display the statistics generated from the notifications' data and tickets' data based on the visibilityLevel of User or Authority.
  - *showMapStats(visibilityLevel)*: display the statistics on the map generated from the data based on the visibilityLevel of User or Authority, in order to show sensitive information.

- *showNotificationList()*: display to the Authority the list of tickets that can be validated or refused.
- *showInterventionList()*: display the intervention list suggested by the System to the Authority.

- **Data Manager**
  - *getGPSData()*: get the coordinates data in order to retrieve the position of the physical device.

SafeStreets Business Logic
In this section we will describe the methods of the Business Logic, divided per each component.

- **Authentication Manager**
  - *registration(registerInfo)*: method that processes the registration using the information filled by the Guest.
  - *login(loginInfo)*: method that processes the login using the information filled by the User or Authority.
  - *validate(Info)*: method that validate the authorisation of the User's application to use the functionalities of the application.
  - *createUser(userInfo)*: method that create a new User using the information filled by the Guest in the form, if the process has been done correctly.
  - *sendEmail(mail, token)*: method that invoke the external service to send the confirmation e-mail to the new User.
  - *confirmation(code)*: method used to confirm some actions.
  - *getUser(userInfo)*: method that retrieve information about privileges of the User starting from his/her information.
  - *setUserToken(user, token)*: method that set the token linked to the User.
  - *checkAuthentication()*: method that process the check to be sure that the User has all the authorisation to proceed with the use of the application.

- **User Manager**
  - *setPassword(password)*: method that sets the new password to a User.
  - *setUserInfo(userInfo)*: method that sets the user information.

- **Report Manager**
  - *accessReport(request, token)*: method invoked to access the report function of the application.

- *checkImg(data)*: method that start the process of image checking.
- *notifyImgNotReadble()*: method that invoke the external Notification Service to alert the user to re-enter the image of the violation because it's not readable.
- *createReport()*: method that create the effective report with all the information provided by the User in the notification.
- *insertNotification()*: method invoked to memorise the notification if the process has been done correctly.

- **Plate Img Controller**
  - *processCheckImg(img)*: method that using as input the image of the violation, analyse the image and determine if it has been modified or not, and that's readable or not.

- **Statistics Manager**
  - *accessStats(request, token)*: method invoked to access the statistics function of the application.
  - *accessMaps(request, token)*: method invoked to access the maps function of the application.
  - *getStatsData()*: method that get all the data needed for the statistics generation.
  - *getNotificationData()*: method that get all the data generated from the Users' notification.
  - *getMap()*: method that invoke the external Map Service to cross the data with the map visualisation.
  - *computeStats()*: method that compute the Statistics to show the User or Authority.
  - *computeMapStats()*: method that compute the Map Statistics to show the User or Authority.
  - *setStats(visibilityLevel)*: method that set the Statistics to show to the User or Authority, based on the visibility level.

- **Ticket Manager**
  - *accessTickets(request, token)*: method invoked to access the tickets function of the application
  - *getNotificationList()*: method that get all the notification list to show to the User
  - *getNotificationDetails(notID)*: method that get the information about a specific notification based on the notID.
  - *confirmNotification(notID)*: method invoked when a notification is confirmed by the Authority, based on the notID.

- *refuseNotification(notID)*: method invoked when a notification is refused by the Authority, based on the notID.
- *setNotification(notID)*: method that invoke the external Municipality Service to set the notification in the list of the feasible one, so that the process of ticket generation can be done.
- *discardNotification(notID)*: method that discard the notification after it has been refused, based on the notID.

- **Intervention Manager**
  - *accessInterventions(request, token)*: method invoked to access the intervention function of the application.
  - *getInterventionData()*: method that get all the data needed for the interventions suggestion function.
  - *setInterventionList()*: method that set the intervention list to show to the User.
  - *confirmIntervention(intID)*: method invoked when the Authority confirm that the intervention is feasible based on the ID of the intervention.
  - *setIntervention(intID)*: method that invoke the external Municipality Service to set the Intervention in the list of the feasible one, in order to start the phase before the physical intervention, based on the notification ID.

- **Data Manager**
  - *SELECT(data)*: method that select the data as input from the external database service.
  - *UPDATE(data)*: method that update the data as input from the external database service.
  - *DELETE(data)*: method that delete the data as input from the external database service.

2.6 Selected Architectural Styles and Patterns

Multi-tier Architecture
As stated in the Overview section, the *System*'s architecture is a multi-tier architecture. A multi-tier architecture is a client-server architecture where presentation, application logic, and data management functions are physically separated. This division let each tier perform specific tasks, avoid the dependancy from a single node for all the architecture and facilitates the modification of a single tier. This guarantees a better maintenance of *System*.

SafeStreets implements a three-tier architecture, with the following layer: Presentation tier, Business (or Domain) logic tier, Data management tier. We'll now analyse each one in details:

- **Presentation tier:** this is the top level of the application; this tier is completely managed by the Client device, which is a mobile application, both for *Users* and *Authorities*. The main task is to display to the User's application all the information provided by the Application Server, thanks to the GUI of the mobile device. It also displays also the information of the external services. This is the only tier directly accessible from the User

- **Business Logic tier:** this tier host the Application Servers of *SafeStreets*. As explained by the name it implements only the Business/Domain logic layer and not the storage. The deployment is done by replicating each server as many times as necessary to guarantee the non-functional requirements. Also, each on the server is stateless. In case of many request, it's possible to add more machines to handle the traffic rise. It controls all the application functionality with all his processing. This tier contains the reverse proxy.

- **Data management tier:** this tier is represented by the Database. A cloud solution is adopted in order to eliminate from the *System* tasks like the reliability and management of the physical database. So, it is the data access layer and encapsulate all the related mechanisms.

Thin Client
The thin client paradigm is implemented for the interaction between the physical device machine and the *System*. A thin client needs to perform very few computation, very close to no computation, but it has to handle the communication. The main advantage of this architecture is that it's easier to keep data synchronised across multiple clients. It also requires less effort to create new client implementation because there's no logic. A thin client does not rely neither on local storage since all the business logic is on the application servers, which have enough computing power. Even if our Client implementation has a Data Manager and it's used to retrieve the GPS data and because the contribution to the data access is very low, we consider the Client tier as a Thin Client, which host the Presentation layer.

MVC
The *System* architecture follow the MVC (Model-View-Controller) pattern. This pattern of functionality separation has numerous advantages like simultaneous development, high cohesion and loose coupling between classes, multiple views for the model and ease of modification. It's considered a best practice in the development process and respect the fundamental design principles of software engineering.

- **Model**: that's everything related to the application's data and integrity. It's implemented in our *System* by the external database.
- **View**: this corresponds to the Presentation tier, that's the only interface between the User's application and the *System*.
- **Controller**: that's responsible for the application logic. In our case is stateless and replicable and corresponds to the business logic layer of the *System*.

2.7 Other Design Decisions

Database
It's used a relational model for the *SafeStreets* database because it's the right schema for the purpose of our *System*. Even if the *System* has to compute a lot of calculation, the majority of the operations needs to select small amount of data; these are repetitive operations that should be computed fast, so the chose of a non-relation model is not take into account. Also, a relation model could be accessed using SQL that's a standard across the industry, instead of using a specific language that a non-relational database needs.

Firewall
The *System* architecture has two firewalls that guarantee a separation between the mobile device and the reverse proxy and between the application server and the database. There's an application proxy that provide one of the most secure types of access you can have in a security gateway, in fact every time an application makes a request, the application intercepts the request to the destination system. There's also a network firewall that filters the packets trying to pass through them, keeping the application safe.

3 User Interface Design

In the section 3.1 of the RASD document are explained in details all the User Interfaces and their design. So, for further information about the UI Design, refers to the RASD document.
In this section is provided an overview of the UX Application flow, using the UI mockups of the RASD. It's explained the flow of the application from the point of view of the *User* and from the point of view of the *Authorities*.

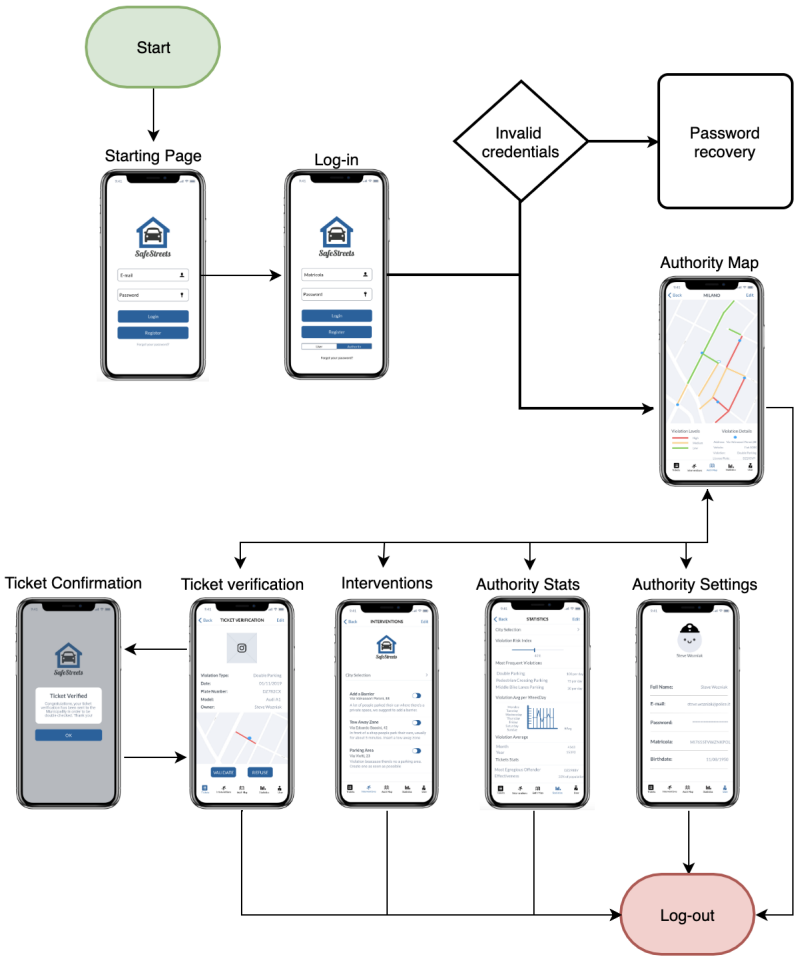

Figure 12: *SafeStreets* User UX flow diagram

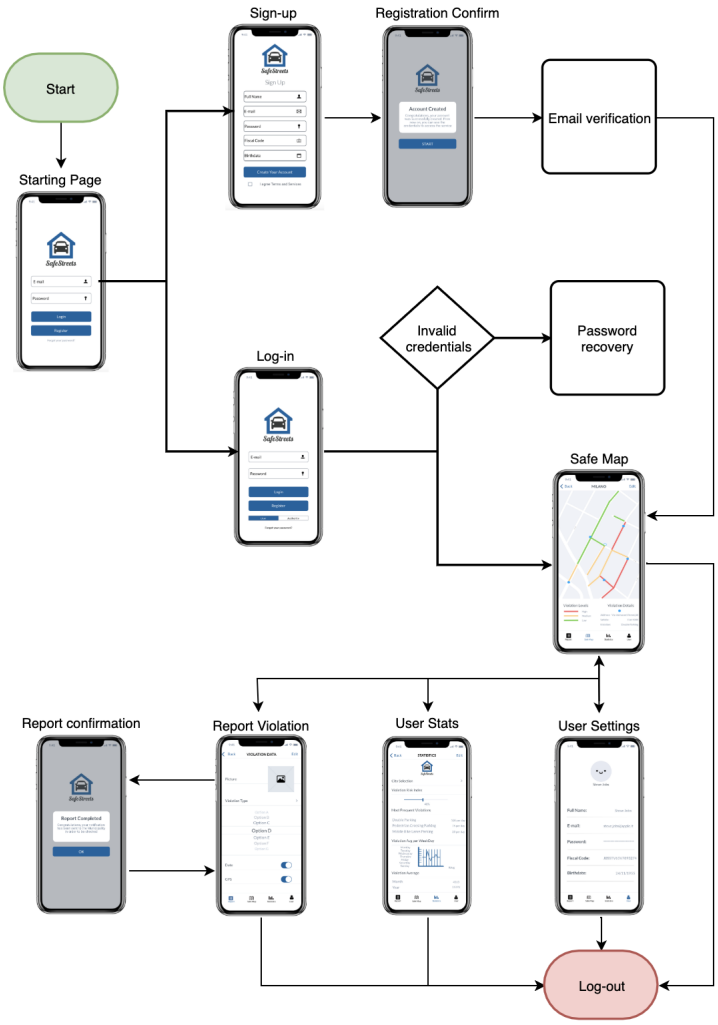

Figure 13: *SafeStreets* Authority UX flow diagram

4 Requirements Traceability

Component (DD)	Requirements (RASD)
Data Manager (Client)	• **[R13]** Metadata, such as date and time, must be automatically re- trieved from the device.
Authentication Manager	• **[R1]** The System allows account to be created as a simple User or Authority. • **[R2]** A User account can be created if the User provides the correct data: unique email and fiscal code. • **[R4]** Users and Authority can access the service if they log-in with their credentials. • **[R5]** The System must be able to check if the credentials are valid. • **[R9]** Only User with an account can create and send a report.
User Manager	• **[R6]** The System must store all User credentials.
Report Manager	• **[R8]** The System gives a feedback to the User if the report process is done correctly. • **[R9]** Only User with an account can create and send a report. • **[R10]** The System must allow the User to take a picture of the violation. • **[R12]** The System must allow the User to select the violation type. • **[R14]** The System must allow the User to edit information before sending the report.

Plate Image Controller	• **[R7]** The System must be able to analyse the picture and recognise the plate number. • **[R8]** The System gives a feedback to the User if the report process is done correctly. • **[R11]** The System can accept or refuse the image loaded from the User.
Statistics Manager	• **[R18]** The System must allow the User to visualise statistics derived from the data. • **[R29]** The System must allow the Authority to visualise statistics derived from tickets' data.
Ticket Manager	• **[R23]** The System must allow the Authority to access information about the violation notification. • **[R24]** The System must allow the Authority to validate a ticket. • **[R25]** The System must allow the Authority to access sensible data about the violation. • **[R26]** The System must advise the Authority that the violation de- tails process is done correctly.

Intervention Manager	• **[R20]** The System must be able to identify viability issues based on data. • **[R21]** The System must suggest solution to address viability issue. • **[R22]** The System must allow the Authority to access the list of possible interventions. • **[R23B]** The System must allow the Authority to establish if the inter- vention is feasible solutions or not.
Data Manager	• **[R6]** The System must store all User credentials. • **[R15]** The System must be able to store all the notifications' data. • **[R27]** The System must be able to store all the tickets' data.

Table 2: Requirements Traceability

5 Implementation, Integration and Test Plan

5.1 Overview

The *System* is divided in many components, that can be divided in the following sub-systems:

- Frontend Components: this is Client application subsystem, containing the presentation and the data manager components.
- Back-end Components: this is the Business Logic subsystem with all its components that interacts with the database.
- External Components: this are the external components that interacts with the *System*, such as the Map and GPS services.

The strategy used to design the *System* was *Top-Down*, defining the sub-systems described above at a high-level, then refining and detailing them with all the details needed.
The implementation and testing approach will follow a combination of both *Top-Down* and *Bottom-Up* strategies because it's the most reasonable choice for relatively small components and sub-systems.
In the following paragraphs it will be presented the order of the implementation and of the integration testing between components.

5.2 Implementation

The implementation of the different components will be parallelised in order to divide the work between developers and for speeding up the implementation process.
The order of the implementation is:

- Implementation of the Database component
- Implementation of Business Logic and Client components
- Integration of component with external service interfaces

The implementation of the Database consist of instance creation on an external platform provider and the creation of tables and schemes. The implementation of the Back-end Components is parallelised, so the Business Logic (Server) and Client will be developed by different developers and then linked together. The integration with external services consists of a series of API calls to the external services that are already implemented.

The Business Logic implementation will follow this order. If there are some components on the same line means that they will be implemented in parallel:

1. Data Manager
2. Authentication Manager, User Manager
3. Report Manager, Plate Img Controller
4. Statistics, Ticket, Intervention Manager

The Client implementation will follow this order:

1. Data Manager
2. Presentation

The Data Manager is critical because it represents to unique point of access to the Database and it is the component that interacts with all the components that compute the application logic and operate on the basis of the Data Manager retrieval capacity.

5.3 Integration and Testing

Throughout all the implementation activity every single component, with the exception of the external ones, will be tested as a single Unit, in order to immediately correct as many defects as possible and to save time and effort costs. Fixing bugs and malfunctions gets in fact increasingly harder as the development of the *System* proceeds. There is no need to test the external components because the reliability of their services will be the main factor in the process of choosing them; instead they will be used for testing the implemented components as Unit and the subsystems derived from their integration. In order to Unit-test a single component, most of the time the behavior of the modules it has to interact with will have to be simulated. This can take some time, but still it is critical, to proceed with the integration of components, to be able to rely on the internal functionalities of the modules.

The integration phase will take place following the temporal order of the implementation and a flow that can be considered *bottom-up*. This means there will first be an integration between some components of the back-end and the test of the resulting subsystems. Then the integration of all components part of the front-end and back-end subsystems and their test. Eventually the integration of front-end and back-end and the test of the *System* operation, before integrate it with the external components interfaces as well.
Specifically:

1. Integration of the Auth Manager and the User Manager and test of their interactions (for example during the Registration process);

2. Integration of the Report Manager and the Plate Img Controller and test of their interactions;

3. Integration of the two subsystems above with the Data Manager and test of its functionalities;

4. Integration of the subsystem and the Statistics Manager, Ticket Manager and Intervention Manager, which are responsible for the Advanced Services provided by the System (see RASD section 1.1), and test of the remaining functionalities of the back-end;

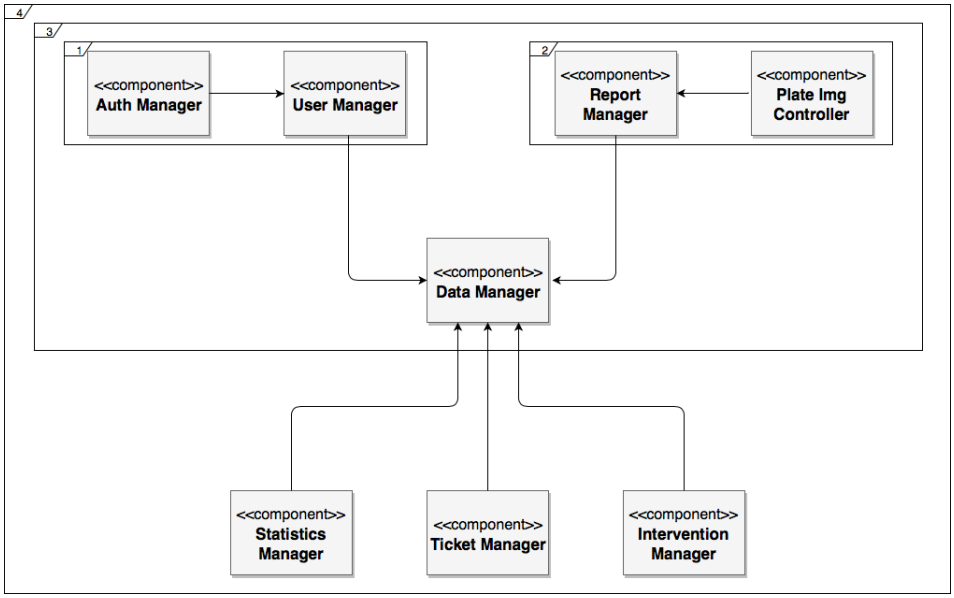

Figure 14: Integration of the back-end components

5. Integration of the components of the front-end, the Presentation and the Data Manager, and test of the front-end operation;

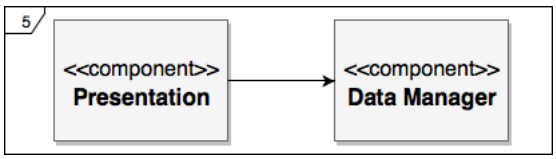

Figure 15: Integration of the front-end components

6. Integration of back-end and front-end and test of the interactions between the two;

7. Integration of components with external services interfaces and final test of the System.

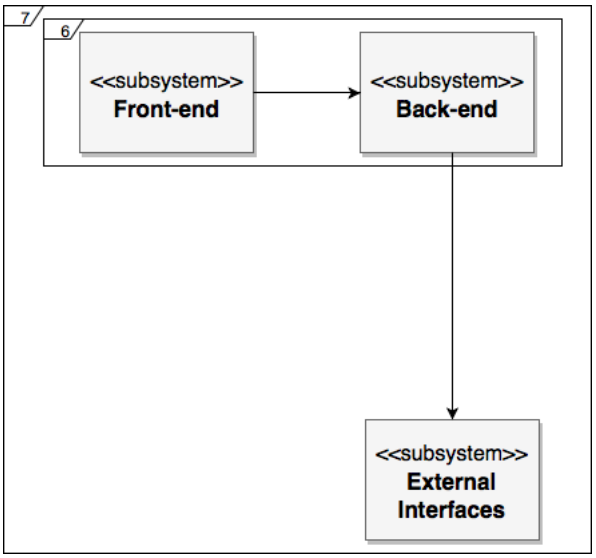

Figure 16: Final integration

6 Effort Spent

Date	**Task**	**Hours**
XX/XX/XX	XX	X
		Total
		XX

Table 3: Adriano Mundo's effort

Date	**Task**	**Hours**
XX/XX/XX	XX	X
		Total
		XX

Table 4: Francesco Rota's effort

Date	**Task**	**Hours**
XX/XX/XX	XX	X
		Total
		XX

Table 5: Salvatore Fadda's effort