POLITECNICO
MILANO 1863

SafeStreets

Marco Premi (941388)
marco.premi@mail.polimi.it

Fabrizio Siciliano (939895)
fabrizio.siciliano@mail.polimi.it

Giuseppe Taddeo (928360)
giuseppe.taddeo@mail.polimi.it

Computer Science and Engineering

2019/2020

Software engineering 2

DD

Design Document

Version 1.0 - [Data da inserire]

Reference professor:

Matteo Giovanni Rossi
matteo.rossi@polimi.it

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to give more technical details than the RASD about SafeStreets system. The RASD presented a more abstract and general view of the system and of the functions is supposed to execute. Indeed, this document presents more details about the design, run-time processes, deployment and algorithm. It also provideds more information about implementation, integration and testing with a testing plan.
In particular, the document presents the following topics:

- Overview of the high-level architecture
- The main components, their interfaces and deployment
- The run-time behavior
- The desing patterns
- Requirements on architecture components
- Implementation plan
- Integration plan
- Testing plan

1.2 Scope

Here it's presented a review of the application scope, made referring to what has been stated in RASD document.
With SafeStreets users can notify the authorities when traffic violations occur, and in particular parking violations. Both user and authorities must register to the application and agree that SafeStreets stores the information provided, completing it with suitable meta-data. The whole system, because it tracks users information, must respect the standards defined for processing of sensitive information such as GDPR if it is used in Europe. The user sends the type

of the violation to the municipality and direct proofs of it (like a photograph). The system runs an algorithm to read the license plate and also asks the user to directly insert the license for a better recognition. Obviously, other information are required, like the name of the street when the violation has occurred, which can be retrieved from user's direct input or from the geographical position of the violation (using Google Maps API). Furthermore, the system, by cross referencing data from third party services, automatically can highlight the streets with the highest frequency of violations or the vehicles that commit the most violations. SafeStreets crosses information about the accidents that occur on the territory of the municipality with his own data to identify potentially unsafe areas and suggest possible interventions. Because municipality could generates traffic tickets from the information about violations SafeStreets should guarantee that information is never altered (if a manipulations occurs, the application should discard the information). Such features are made possible trough the use of one mobile application with two different UIs which are determined by the kind of customer that logs in (user or PO). The collected information are sent to a back-end and they all of those can be accessed by municipality employees in order to execute different actions (emit ticket, analyze unsafe areas, etc...).

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **User:** it is identified as a civilian customer of the product. It will be the main source for the SafeStreets initiative to obtain information about traffic violations and therefore to be successful;
- **Third parties:** those kind of organization/company that could provide services useful to SafeStreets;
- **Customer:** it defines both authorithy users (police officers or municipality employees) and civilians;
- **Authority user:** all of those customers who have a responsibility role in regard of the streets' safety and the SafeStreets initiative. Example of these category are: police officers, municipal employees, director and basically anyone in charge and able to issue fines and deal with road violations;
- **Ghiro:** image manipulation detection software, used by authorithy users in order to detect any image manipulation and assess the veracity of the hard evidence connected to the traffic ticket

1.3.2 Acronyms

- **UI:** User Interface
- **GDPR:** General Data Protection Regulation
- **API:** Application Programming Interface
- **GPS:** Global Positioning System
- **PO:** Police Officer
- **ME:** Municipality Employee
- **PaaS:** Platform as a services
- **VPC:** Virtual Private Cloud
- **AWS:** Amazon© Web services
- **Amazon© EC2:** Amazon© Elastic Compute Cloud
- **CPU:** Central Processing Unit
- **ACID:** Atomicity, Consistency, Isolation, Durability
- **DBMS:** DataBase Management System
- **REST:** REpresentational State Transfer

1.3.3 Abbreviations

- **Gn:** nth goal;
- **Dn:** nth assumption;
- **Rn:** nth requirement;
- **ID:** identifier (Fiscal Code for Users, a municipality identifier for Authorithy Users)

1.4 Revision history

1.5 Reference Documents

- **Amazon© Web Services**
- **Amazon© Virtual Private Cloud**
- **Amazon© Elastic Compute Cloud**
- **RESTful Architecture**

- **Relational DataBases**
- **Multi-tier Architecture**

1.6 Document Structure

Chapter 1 - Introduction

Chapter 1 is an introduction of the design document. It describes the purpose and the scope of the document and it highlights the differences with the RASD. It also shows some abbreviations, definitions and acronyms in order to provide a better understanding of the document to the reader.

Chapter 2 - Architectural design

Chapter 2 deals with the architectural design of the system and it's the core section of the document.
It provides an overview of the architecture and it contains the most relevant architecture views:

- Component view
- Deployment view
- Run-time view

It also shows the interaction of the interfaces and the selected architectural styles and patterns, with an explanations of each one of them.

Chapter 3 - User interface design

Chapter 3 specifies the user interface design and refers to the mock-ups already presented in the RASD. Furthermore it shows some UX diagrams to describe the interaction between the customer and the application.

Chapter 4 - Requirements traceability

Chapter 4 explains how the requirements defined in the RASD map to the design elements defined in this document.

Chapter 5 - Implementation, integration and test plan

Chapter 5 specifies the description and the order of implementation, integration and testing plan of the sub-components of the system.

Chapter 6 - Effort spent

Chapter 6 shows the effort spent by each member of the group working on this project.

Chapter 7 - References

Chapter 7 includes the reference documcuments.

2 | Architectural design

2.1 *Overview*: High-level components and their interaction

The whole software that will become the main core of the SafeStreets initiative will be developed as a distributed application, which means that the software will be executed (or run) on multiple devices within a network. It will have a three-layers logic and be divided as following:

- **P**: The ***presentation*** layer will handle all *incoming* (*and outcoming*) relations with the customers
- **A**: The ***application*** layer will work as a "man in the middle" between the **P**resentation layer and the **D**atabase layer and will hold all the needed logic for the software to correctly work;
- **D**: The ***database*** layer will be needed in order to store and manage all needed (and requested) information of the initiative;

Each and every one of the layers the architecture will be composed by a (group of) machines. By doing this, it is meant to provide, to each layer, its own dedicated hardware, for either scalability, failure handling and flexibility reasons. The following image shows the high-level architecture of the system without providing any detail of the components which will form the structure of the software itself, which will be tackled later in this document.

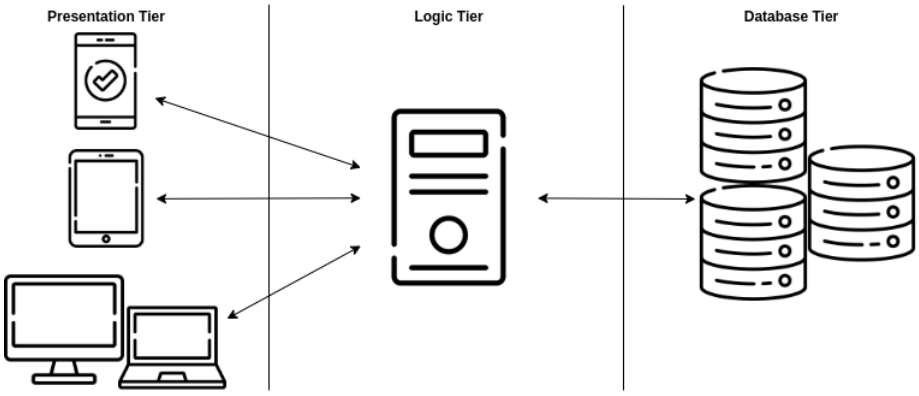

Figure 2.1: High-level architecture

The introduction of the **P**resentation layer has been considered in order to allow a thin client architecture and let less performing devices access SafeStreets initiative and give a smoother feeling when using either the mobile or web version of the software. By doing so, this allows also for an high reusability of the code, since the logic is all implemented in the single **A**pplication layer, which allows different devices to access the same logic. The latter is the only tier which deals with two other tiers at the same time and is in charge of accessing data from the **D**atabase layer and pass it to the clients back and forth. In addition, the *man in the middle* communicates with the Data tier synchronously when it comes to access the needed data, but asynchronously when storing and writing actions are required.

The software will exploit a **P**latform **as a S**ervice (***PaaS***) provided by Amazon©. It has been chosen to create a **VPC** in order to attend the requirements stated in section 3.5 *"Software System Attributes"* of the RASD. This will help also to augment the system scalability and improve the perfomances, as well as the reliability and security of the information stored in the system.
The system is going to be modeled on a *scale out* architecture: this will be improve the performance, as well as the failure management, by replicating nodes. This kind of architecture has been chosen, instead of a scale up architecture, as the latter is not suitable for a system thatplans to be eventually expanded and furtherly serve more and more customers. It comes without saying that the chosen kind of architecture will require the implementation of a *load-balancing system* as well as a ***S**hared **D**isk **C**onfiguration* (**SDC**) in order to let all hardware write and read the same information at any moment. The latter has been chosen over a Shared Memory Configuration as it provides a certain degree of fault tolerance and does not create a bottleneck on the memory bus.
Furthermore, in order to comply to the correct chain of custody that the system requires and to protect all sensible information of all customers, the installation of a proper **D**e**M**ilitarized **Z**one (***DMZ***) is needed. This is accomplished by a

series of firewalls created around the core tier of the software, the *Application Tier*, the one that, if compromised by malicious users, could provide access to both other layers.

The following image shows a detailed representation of the concepts above explained. Thanks to the decision of exploiting Amazon©'s services and its VPC, all the above mentioned architectural characteristics are automatically implemented, as the whole service is customizable and easily implementable.

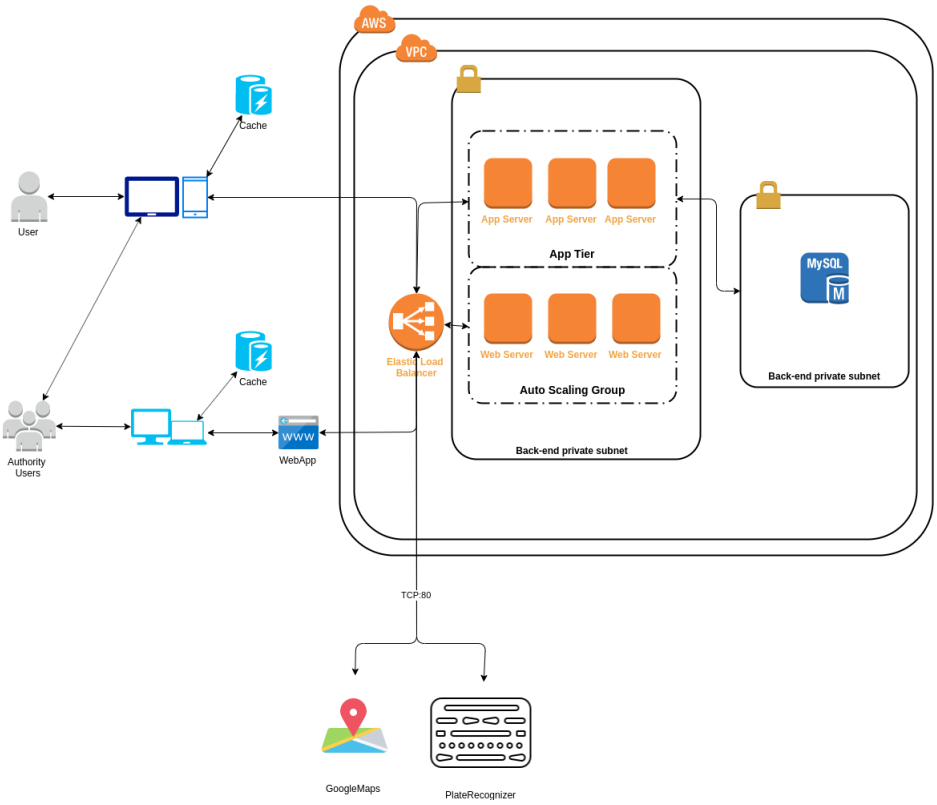

Figure 2.2: Architectural structure

It is easily noticeable that there is no mention of the photo forensic software (*Ghiro, per instance*) as this general architecture focuses on the software itself. Any additional photo forensic software, like the one proposed, has to be considered part of the "package" sold, but not needed to be developed as an already functioning software has been chosen. The sole purpose of SafeStreets is to ensure that the program is actually used and allow an automatic redirection to it.

2.2 Component view

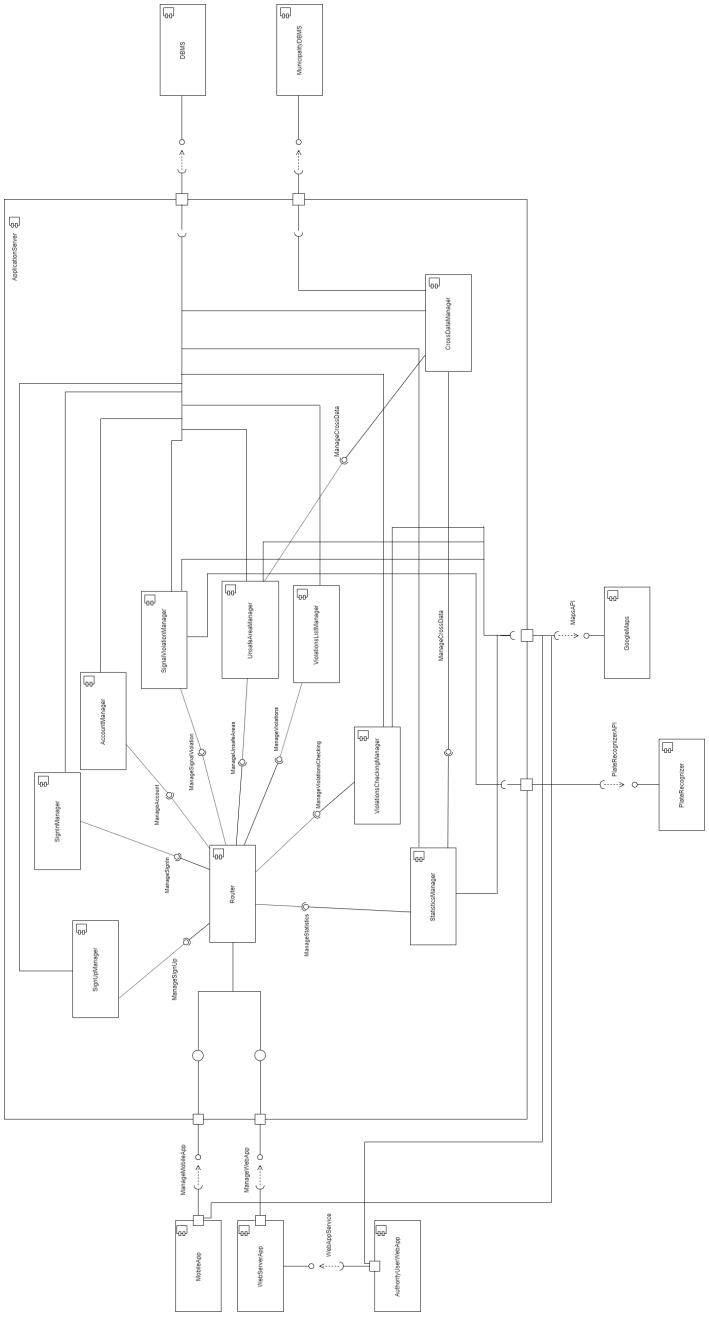

Figure 2.3: Component Diagram

- **Router**:it manages messages and function calls coming from other subsystems in order to pass the data to the correct element of the system. It eventually calls the correspondent method/function on it. Furthermore, the router is partitioned according to the type of the interacting components because of the different functionalities.
- **SignUpManager**: this component provides all the procedures to allow customers to register to SafeStreets. Obviously, this components has also to interact with SafeStreets DBMS to store the registration data and to run a check about the chosen email, password and AuthoritiesID or fiscal code.
- **SignInManager**: it contains all the logic devoted to the authentication of the customers. It check the authentication parameters using the data stored on SafeStreets DBMS.
- **AccountManager**: this component provides all the procedures to manage the account and interacts with SafeStreets DBMS.
- **SignalViolationManager**: it deals with the signalations of violations made by the users. It interacts with SafeStreets DBMS.
- **UnsafeAreasManager**: this component provides all the users the possibility to see the unsafe areas. It receives all the information from SafeStreets DBMS and municipality DBMS through CrossDataManager.
- **ViolationsListManager**: with this component the user can see all the violations he/she has reported. The authority user can see all the violations reported in his/her area. It interacts with SafeStreets DBMS.
- **ViolationsCheckingManager**: this component provides the authority user the ability to check che violations. It interacts with SafeStreets DBMS.
- **StatisticsManager**: this component provides the authority user the possibility to see all the statistic generated crossing the data on violations. It receives all the information from SafeStreets DBMS and municipality DBMS through CrossDataManager.
- **CrossDataManager**: it crosses data from SafeStreets DBMS and from the municipality DBSM to generate the statistics and the unsafe areas.

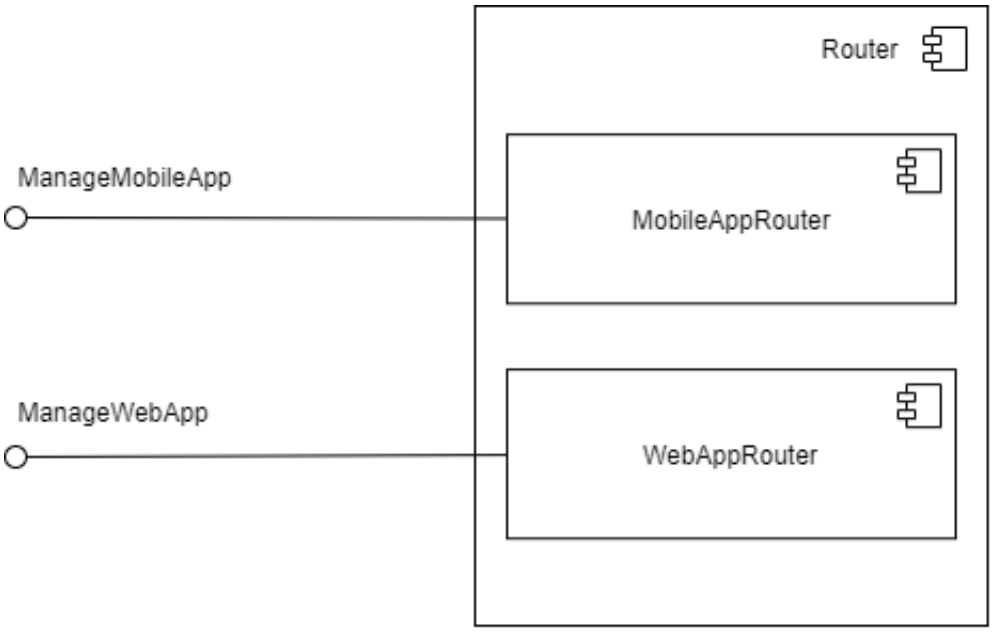

Figure 2.4: Detailed view of router components

2.3 Deployment view

The following image is a deployment diagram which represents the architecture of the system as distribution(deployment) of software artifacts to deployment targets(node). Artifacts represents elements obtained with a development process. Nodes can represent either hardware or software environments.

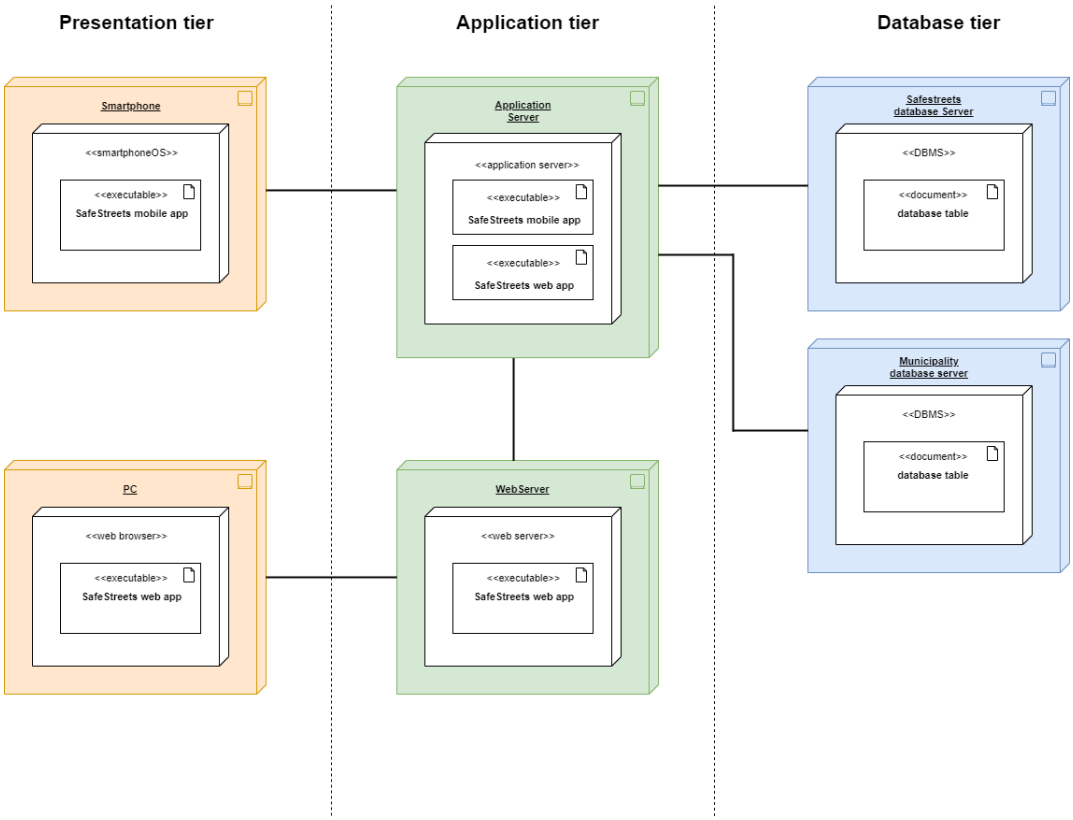

Figure 2.5: Deployment view

The three tiers contain:

- **Presentation tier**: in this tier the presentation logic is deployed. Users are provided with a mobile application on their mobile devices and authority users are provide with both a mobile application and a web application accessible from a common browser. The mobile application must be developed for most of the devices (both iOS and Android version). Both user and authority user ask to communicate to the application server in order to retrieve data, signal a violation, check violations or unsafe areas.

- **Application tier**: in this thier the application logic is deployed. The application server allows the mobile application and the web application to access data stored into the SafeStreets database. The application server also implements the business logic and handles the requests. The mobile application directly addresses the application server. The web server allow authority users to use SafeStreets services. If it can't provide some information it forwards the requests to the application server.

- **Database tier**: in this tier data access must be deployed. The internal Safestreets' database is going to be a relational database (which is going to be furtherly explained in this document). In the meantime, the software is going to require an exclusive access to third party databases (eg. municipality databases), which are going to expose a series of API calls in order to retrieve relative data on certain areas of said municipality, traffic violations and so on.

2.4 Run-time view

2.4.1 Signal a violation

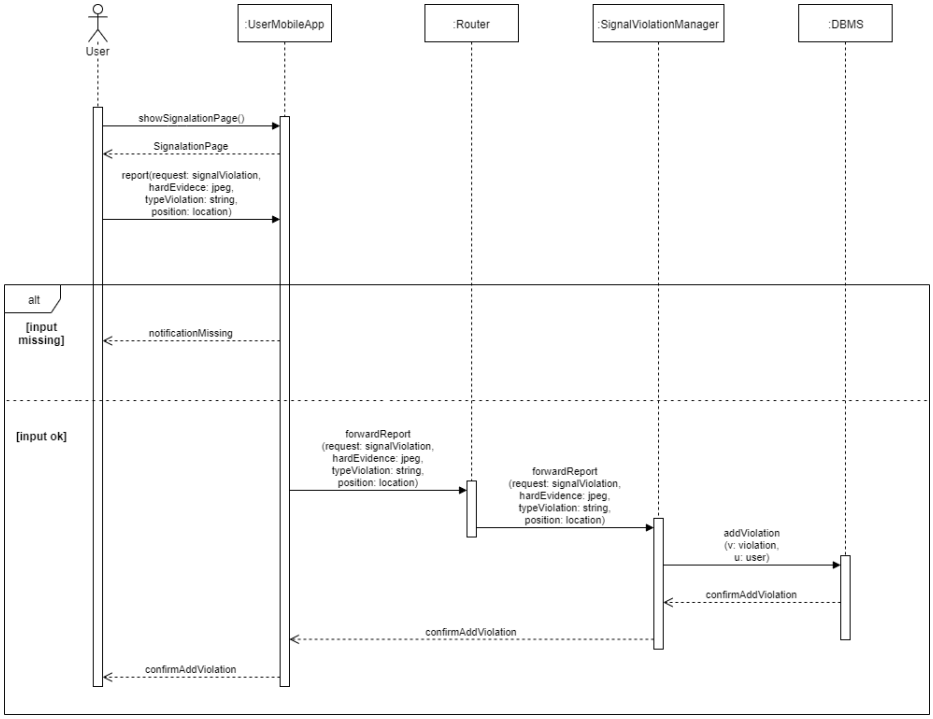

Figure 2.6: Runtime View - Signal a violation

In this sequence diagram the process through which a user, already logged in, can signal a violation to the authorities is shown. Once the web app has rendered the page to report a violation, the user can insert all the needed input data to perform the action (the position, photos of hard evidence, type of violation are here thought to be contained in the request object. The position is assumed to be entered by the user manually, the detection of the online location by Google Maps is neglected). When submitted, the request is sent to the Router, which forwards it to the right component, i.e. the 'SignalViolationManager'. The latter is responsible for checking the input inserted by the user: if there are some missing or invalid input, an error message is sent back to the user who can recompile the form and resubmit the new data. Otherwise, if all the inputs are valid, the request with all data of the violation and the parameter user set to the user who have done the report is sent to 'ViolationListsManager', which forwards it to the DBMS (which adds it to the list of reports of that municipality). At this point, the DBMS send the confirm of the operation to

the components up to the user who it's ready to send other reports or to return to his menu.

2.4.2 SignUp and SignIn

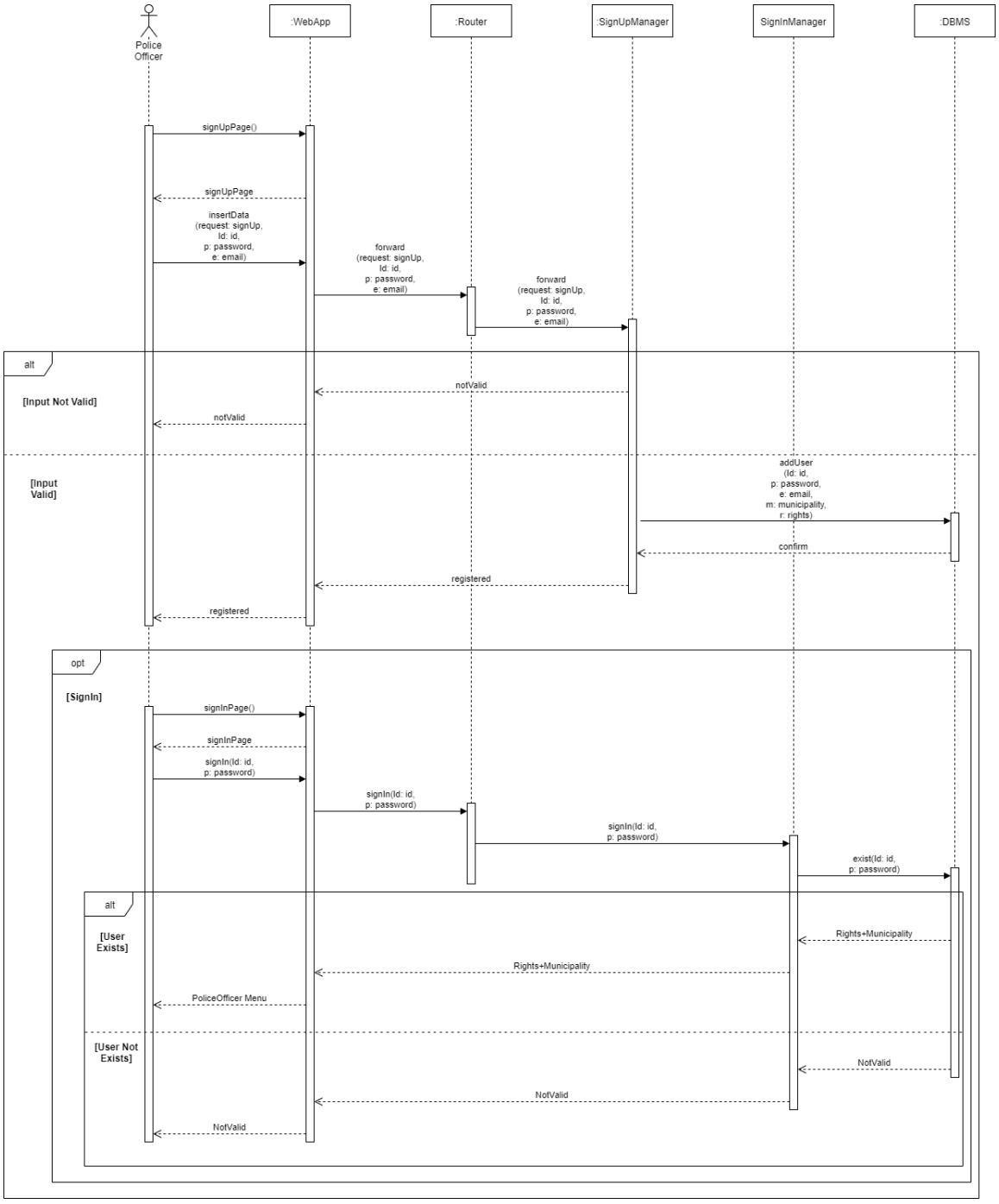

Figure 2.7: Runtime View - SignUp and SignIn

In this sequence diagram the process of sign up and subsequently sign in is shown. Once the WebApp has rendered the page to signup, the Police Officer insert his data (AuthorityId, email, password). When submitted, the request is sent to the Router, which forwards it to the right component, i.e. the 'SignUpManager'. The latter is responsible for checking the input inserted by the Police Officer: if there are some missing or invalid input or the Authority results already registered, an error message is sent back to the user. Otherwise, if all the inputs are valid, SignUpManager analyze the AuthorityId and understand the municipality of competence of the Police Officer and his rights, so it adds the parameters of municipality and rights to the other data and send them to the DBMS which add a new AuthorityUser. At this point, the DBMS send the confirm of the operation to the components up to the user who can, if he want, sign in to the service. He therefore, once the WebApp has rendered the page to signin, inserts his Id and password. When submitted, the request is sent to the Router, which forwards it to the right component, i.e. the 'SignInManager'. The latter is responsible for checking the input and asks to the DBMS if there exists a couple <Id,password> into the database. If there no exists, SignInManager send an error to the user, otherwise, once understand the right of the user (i.e. PoliceOfficerRights) and his municipality, he send these data to the WebApp which shows the personal menu to the Police Officer.

2.4.3 Analyze Unsafe Areas

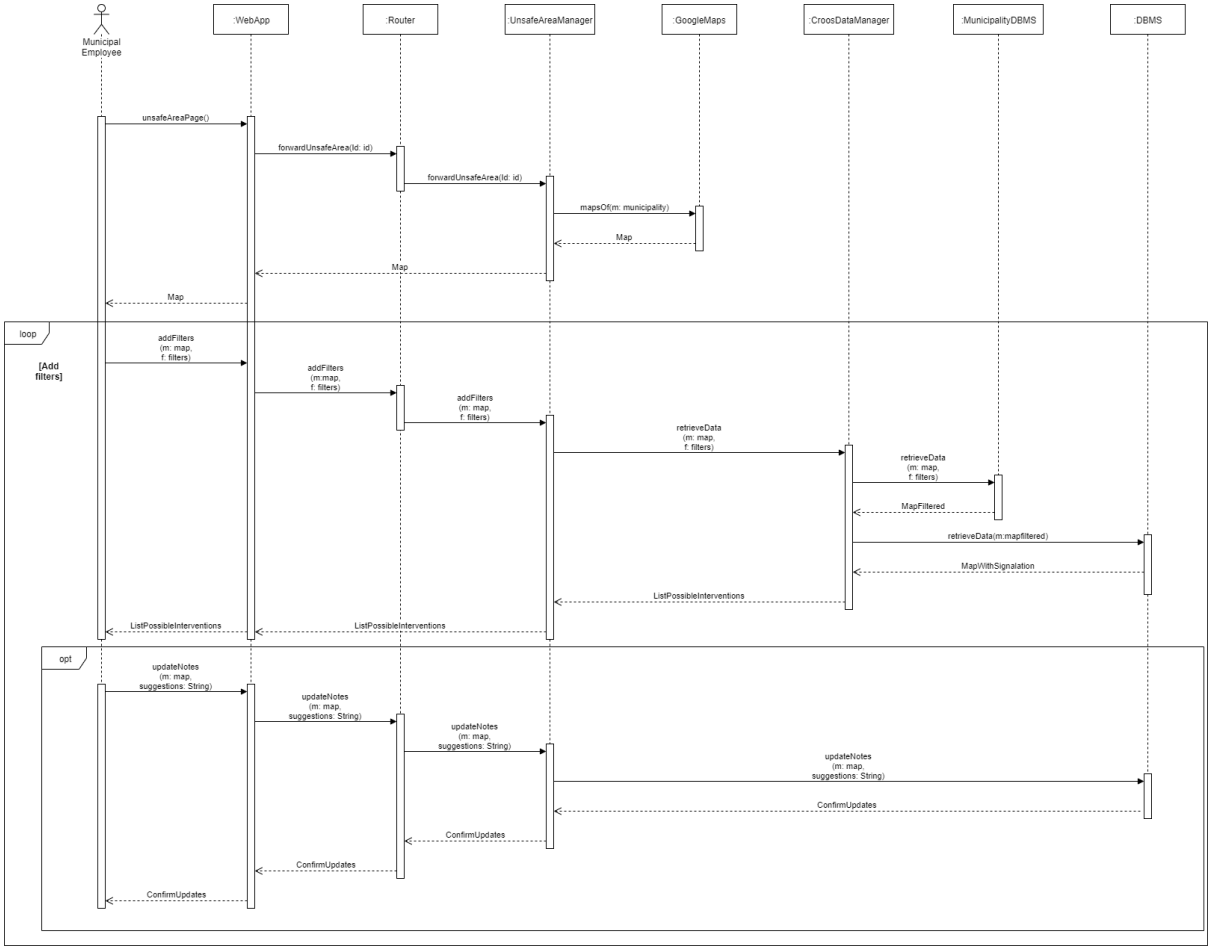

Figure 2.8: Runtime View - Unsafe Areas

In this sequence diagram the process of analyze the Unsafe Areas is shown. The Municipal Employee request the UnsafeAreaPage to the WebApp which forwards the request to the Router adding the Id of the municipal employee. The Routher forwards it to the right component, i.e. the 'UnsafeAreaManager'. The latter understand, through the Id, the municipality of competence of the Municipal Employee and asks the map to Google Maps. Therefore the map of the municipality is sent up to the employee. At this point, the employee can add filters to his maps and, through the same chain, UnsafeAreaManager receives the map from Google Maps and forwards it and the filters to CrossDataManager. The latter forwards all the data to the Municipality Database which responds

with the map filtered. Moreover the CrossDataManager forwards all the data to the DBMS of SafeStreets which responds with the map and signalation. Now CrossDataManager has all the data that it needs to cross the data and he sends the resulting map up to the employee. The employee can decide to update or delete or add suggestions and he send them to UnsafeAreaManager which forwards them to the DBMS of Safestreets, the process ends with the confirm of the DBMS sent up to the employee.

2.4.4 Statistics

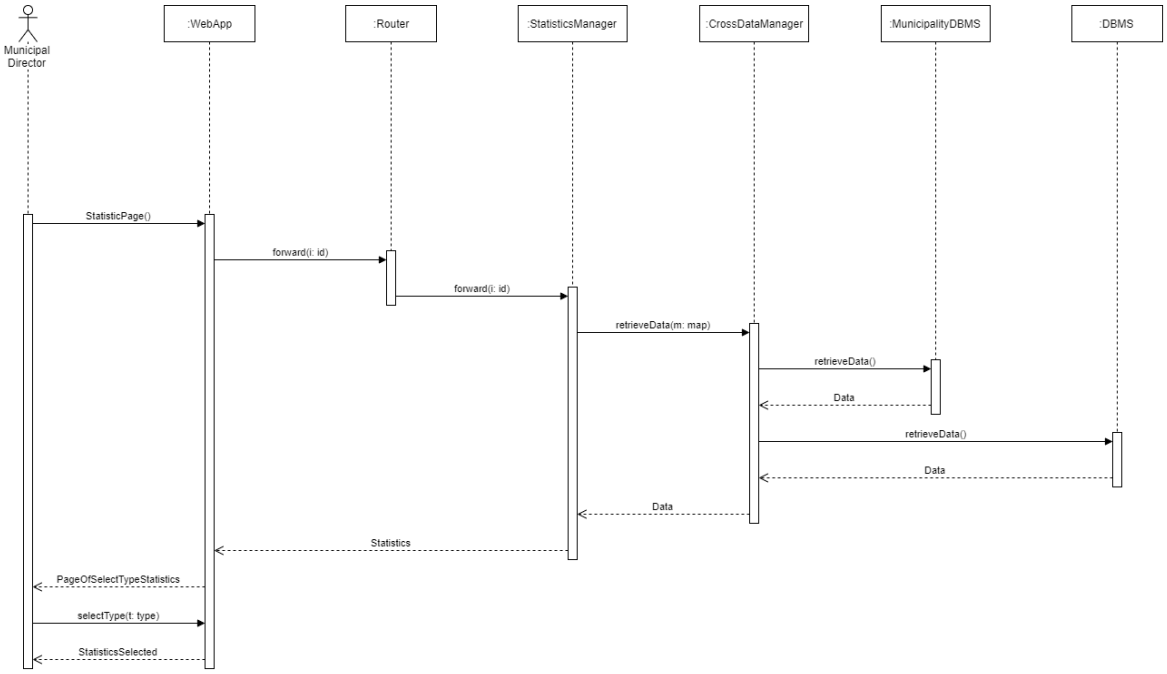

Figure 2.9: Runtime View - Statistics

In this sequence diagram the process of analyze the Statistics is shown. The Municipal Director request the StatisticsPage to the WebApp which forwards the request to rhe Router adding the Id of the municipal director. The Router forwards it to right component, i.e. the 'StatisticsManager'. The latter understand, through the Id, the municipality of competence of the Municipal Director and forward the request to the CrossDataManager. The latter retrieve the data from the municipality and from the DBMS of SafeStreets. Now CrossDataManager has all the data that it needs to cross the data and he sends the resulting data up to the employee. The employee can decide which type of statistics to select and the process enda with the WebApp that show to the director the statistics selected.

2.5 Component interfaces

Figure 2.10: Component Interfaces

The above picture shows an ideal representation of how the main component interfaces, which main functions they should be implementing and how they should be communicating with each other.
The first thing that jumps to the eye is the existence of the two biggest interfaces: *ManageMobileApp* and *ManageWebApp*. Both of them will work as connectors by taking input from the customers and forward them the actual logic tier which processes those inputs and returns the correct outputs, which are once again forwarded by these two components to the UI.

Furthermore, the *ManageCrossData* component is worth of mention as it is a critical point for the correct functioning of two main parts of the SafeStreets initiative. This component manages all interactions that are going to happen with third party databases and the information they hold, cross reference it with the SafeStreets' acquired info and redirects the output to either the *ManageUnsafeAreas* or *ManageStatistics* components.

2.6 Selected architectural styles and patterns

2.6.1 Architectural styles

2.6.1.1 RESTful architecture

It has been decided to use a RESTful architecture in order to develop SafeStreets website. REST architectural style allows requesting systems to access and manipulate web resources by using a uniform and predefined set of rules. REST is very useful in systems like SafeStreets where there are different clients who request resources and a server which provides the resources. In a RESTful architecture we must satisfy some different constraints:

- **Uniform Interface**: there exists a uniform way to interact with SafeStreets server regardless user's device or application.
- **Stateless**: the client hanldes the state for the request and the server doesn't store anything related to the session. This constraint allows simple server design.
- **Cacheable**: this constraint accellerates client-server communications because it completely eliminates some interations.
- **Client-server**: client doesn't know anything about the business logic and server doesn't know anything aboyt the client. and it's UI. In SafeStreets system authority users uses a web app that communicates with the web server which in turns communicates with the application server where is located the business logic.
- **Layered system**: an application architecture with multiple layers allows the presence of a lot of multiple servers between the client and the end server. The architecture is then more flexible and secure because each layer doesn't know anything about any layer other than that of immediate layer.
- **Code on demand**: is an optional feature and it isn't used in SafeStreets system.

2.6.1.2 Three tier architecture

The architecture of SafeStreets is multilayered, composed of three tiers:

- **Presentation layer**: is used to present the data in a way that the user can understand. It enables the usage of the services offered to the user. The presentation layer of the the users is the smartphone and for authority users are both the smartphone and the browser.
- **Application layer**: is used to coordinate the application. It receives and computes the requests send by the presentation layer and it also interacts with the database.

- **Database layer**: stores the information provided by the application layer. The information is also passed back to the application tier for processing.

Three tier architecture is very useful because it allows to change or upgrade one of the three tiers without any problem and so it makes the system more flexible and reusable. Furthermore it makes the system safer because it separates the access to data from the other layers.

2.6.2 Design Patterns

Model View Controller (MVC)

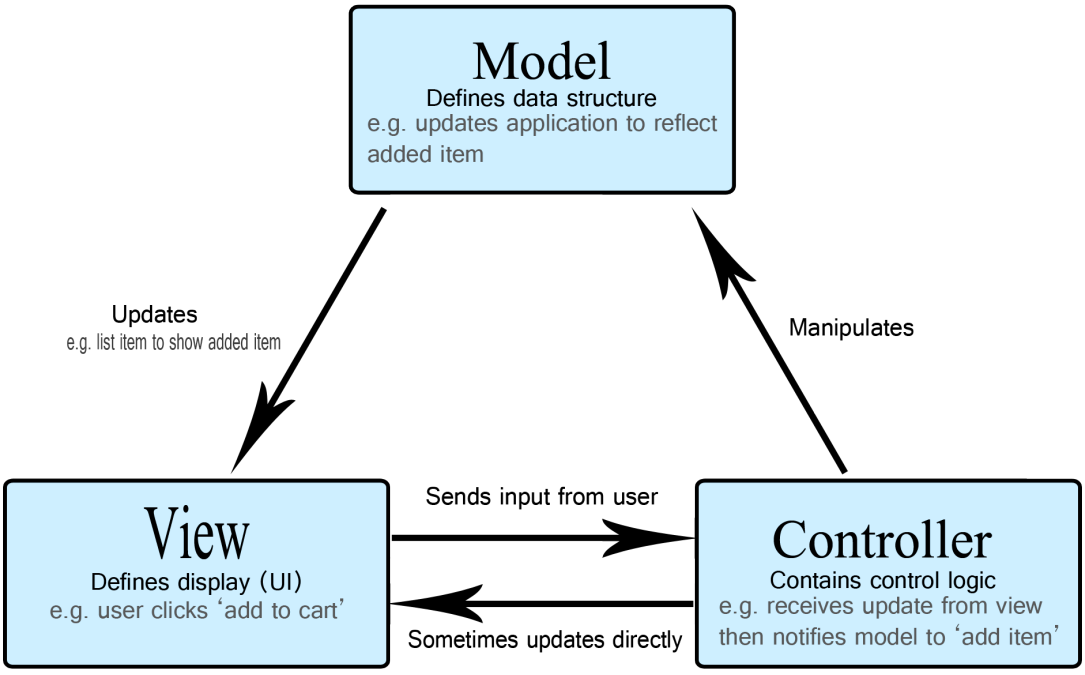

Figure 2.11: Model view controller

Model view controller is a very useful and quoted design pattern. MVC is used to separate the fundamental parts of the application and in particular it emphasizes a separation between the software's business logic and display. The three parts of Model View Controller design pattern are:

- **Model**: manages all the data and the business logic
- **View**: handles the GUI used by all the users
- **Controller**: acts on both model and view, it routes commands to the view and model elements

MVC is also very useful because the decoupling of these three components allows parallel development and code reuse.

2.7 Other design decisions

2.7.1 Virtual Private Cloud

It has been chosen to exploit the services of Amazon© in order to simplify the process of creation and future modification of the architecture, in case any of the constraints should be needed to adapt to the constantly changing market's requirements. *Amazon© VPC* is the networking layer for Amazon© **EC2** (**A**mazon© **E**lastic **C**ompute **C**loud), which provides scalable computing capacity on the **A**mazon© **W**eb **S**ervices cloud (**AWS**). This service provides a series of features as, for example, the ones important to SafeStreets' initiative:

- Virtual computing environments (*or instances*);
- Various configurations of CPU, memory, storage and networking capacity (*instance types*);
- Firewalls that enable developers to specify protocols, ports and source IP ranges that can reach the *instances* using *security groups*;
- Static IPv4 addresses for dynamic cloud computing, known as Elastic IP addresses. This is used in case of failure of an instance by rapidly remapping the failed address to another existing instance;

2.7.2 Thin Client

In a "Thin Client" architecture the server does most of the work while the client is lightweight. In this architecture the client is designed to be online all the time and to communicate with a server. If network connection is down the application of course doesn't work, but we don't need to implement any offline modules because the core idea of the application is comunication itself. Furthermore thin client architecture allows the application to keep all the real business logic protected on the server.
Google Maps APIs could be considered an exception to this paradigm because are used directly by the client.

2.7.3 Relational Database

Relational databases are the most common type of digital databases. The model is based on relations between data and it is solid, avoids duplication of data and enables developers to work with safe and predictable results. By building the relational model, the data are organized into one (or more) tables (or *relations*) of columns and rows, with a unique key identifying each row (or *tuples*) which can be linked between to each other. By doing so, this model allows to handle higly structured data and provides support for ACID transactions. The structure is very flexible and it can be scaled up without problems because adding data without modifying the existing ones is simple. In addition relational databases use very expressive and mature query languages such as SQL and the whole management of the database is left to handle to the DBMS (**D**ata**B**ase **M**anagement **S**ystem).

3 User interface design

3.1 Interface mockups

Both the web and mobile mockups have already been presented in the RASD in chapter 3.

3.2 UX Diagrams

Here three UX diagrams are presented in order to show how the User and the Authority User will navigate through both the mobile and the web application. It's important to underline that the mobile application is the same for both user and authority user: the app is going to recognize the different kind of customer that has been logged in and is going to show both different functionalities and screens.

3.2.1 Mobile application

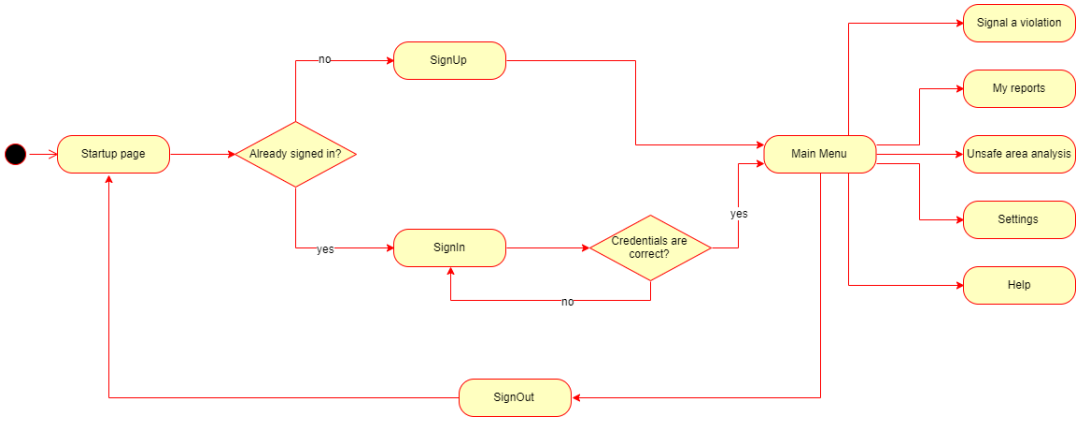

Figure 3.1: UX Mobile - User

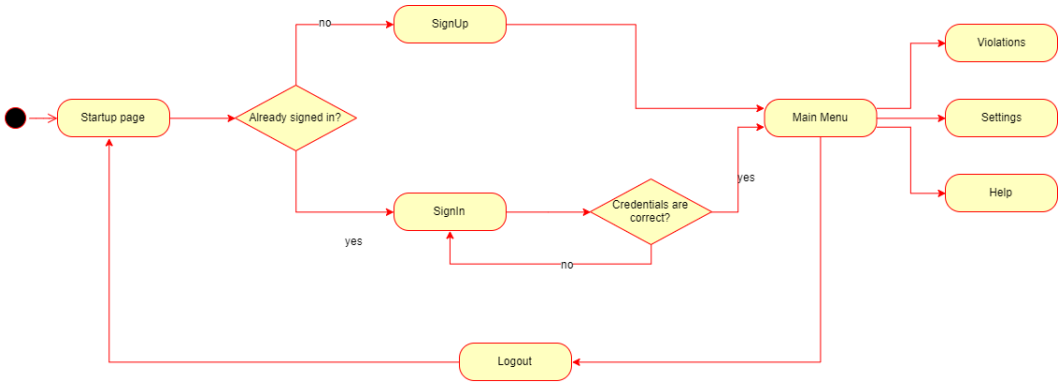

Figure 3.2: UX Mobile - Authority User

3.2.2 Web application

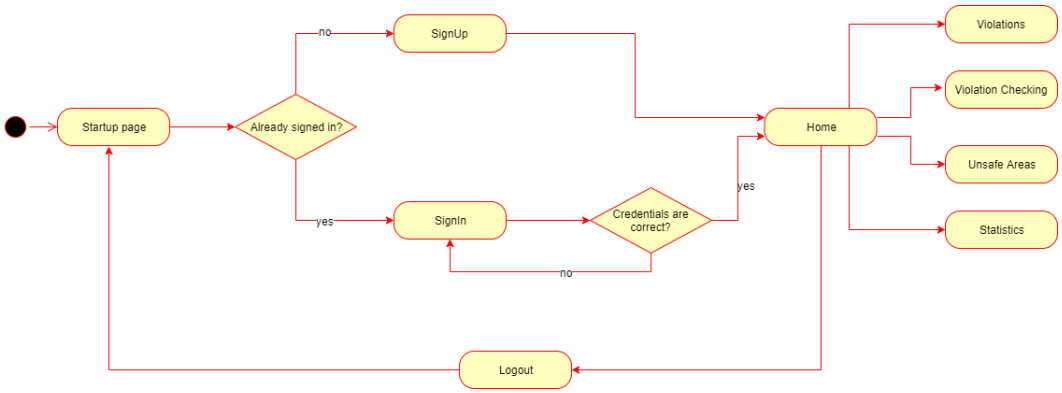

Figure 3.3: UX - Web

4 | Requirements traceability

In this sections it's higlighted the mapping between the requirements defined in RASD and the design component in the application server.
This is necessary to fulfill the goals.

- ⟨**R1**⟩ the customer must not be already registered in the system
  - ⋆ **SignUpManager**
- ⟨**R2**⟩ the customer must provide a valid ID and email
  - ⋆ **SignUpManager**
- ⟨**R3**⟩ the customer must agree to the Terms of Use
  - ⋆ **SignUpManager**
- ⟨**R4**⟩ the customer must be already sign up
  - ⋆ **SignInManager**
- ⟨**R5**⟩ the customer must insert its email and password
  - ⋆ **SignInManager**
- ⟨**R6**⟩ the user must be able to insert one or more photos of the violation
  - ⋆ **SignalViolationManager**
- ⟨**R7**⟩ the user must send information about its location
  - ⋆ **SignalViolationManager**
- ⟨**R8**⟩ the user can add the type of violation that is being reported
  - ⋆ **SignalViolationManager**
- ⟨**R9**⟩ the user is shown the unsafe areas around him

    - ⋆ **UnsafeAreaManager**
- ⟨**R10**⟩ the customer is allowed to filter the unsafe areas
    - ⋆ **UnsafeAreaManager**
- ⟨**R11**⟩ the customer is allowed to search unsafe areas
    - ⋆ **UnsafeAreaManager**
- ⟨**R12**⟩ the customer must be able to modify its account information
    - ⋆ **AccountManager**
- ⟨**R13**⟩ the customer must be able to delete his/her account
    - ⋆ **AccountManager**
- ⟨**R14**⟩ the authorithy user must be able to check all the new and past violations details
    - ⋆ **ViolationsListManager**
- ⟨**R15**⟩ the authorithy user must be able to report the validity of the violation
    - ⋆ **ViolationsCheckingManager**
- ⟨**R16**⟩ the authorithy user is shown a list of possible improvements to be made to the shown areas by the system
    - ⋆ **StatisticsManager**
- ⟨**R17**⟩ the authorithy user must be able to visualize ad-hoc analysis of the Safestreets initiative created by the system
    - ⋆ **StatisticsManager**

5 Implementation, integration and test plan

5.1 Implementation plan

SafeStreets system is composed by different subsystems:

- MobileApp
- WebServerApp
- AuhtorityUserWebApp
- ApplicationServer

All SafeStreets subsystems need to be implemented, tested and integrated using a bottom-up approach. With a bottom-up strategy it's possible to develop lower level components first and then to reuse them in order to create higher level elements. Bottom-up approach also allows better testing of all the different components. It's important to show a visible application feature during all the steps of the development.
There also are some external subsystems:

- SafeStreets DBMS
- Municipality DBMS
- Google Maps
- Plate Recognizer
- Ghiro

All the external subsystems, of course, are not implemented and tested because are developed by third parties, therefore, can be already considered stable and trustworthy.
Follows a list with all the features, their importance for the customer (therefore to the SafeStreets initiative itself) and their complexity of implementation in order to have a better estimation of the order and the time needed to meet the given requirements.

Feature	Importance for the customer	Implementation difficulty
SignUp and SignIn	Low	Low
Signal a violation	High	High
Manage previous reports	Medium	Medium
Visualize unsafe areas	High	High
Manage Account Settings	High	Low
Violation Checking	High	High
Visualize Statistics	High	High

5.2 Integration and testing

5.2.1 Entry criteria

In order to start the integration testing process and to produce meaningful results, there are a number of conditions on the progress of the development of the project that have to be met. As a matter of fact the integration process should start only when the estimated percentage of completion of every component with respect to its functionalities is:

Component	Entry %	Comments
AccountManager	90-95%	The functionality of 'Manage Account Settings' is important for the customer but we can see it as an extra accessory that does not affect the other features of the system, and for this reason the corresponding component 'AccountManager' can be implemented and tested later than the others.
SignUpManager and SignInManager	80-90 %	The sign up and sign in features are obviously an entry condition for the right functioning of the system, but they are not core features and they are not very complex, so the testing and implementation of his corresponding components 'SignUpManager' and 'SignInManager' can be delayed.

StatisticsManager	50-60 %	The functionality of 'Visualize Statistics' needs to be implemented all the part relating to the reporting and management of them, so its entry percentage can be subsequent to the previous one. The component 'StatisticsManager' can just postpone a bit its testing.
UnsafeAreasManager and CrossDataManager	50-60 %	The functionality of 'Visualize Unsafe Areas' has the same role of the previous one, but instead about the Director it's about the Municipal Employee. The components 'UnsafeAreasManager' and 'CrossDataManager' can just postpone a bit its testing.
ViolationsListManager	20-30 %	The functionality of 'Manage Previous Reports' it's important both for the users and for the authorities. The authority needs this component to do his job and the user needs this component to track his signalations. So this component needs to be implemented before ViolationsCheckingManager and to be tested as soon as possible.
ViolationsCheckingManager	10-20 %	The functionality of 'Violation Checking' is very important and affect many other features, so it's crucial to implement and test it. The component 'ViolationCheckingManager' it's hard to implement and it needs to be tested as soon as it is ready.

SignalViolationManager	10-15 %	Signal a violation is the core functionality of the system and without it most other features would make no sense, it requires an implementation and integration with other functionalities as soon as possible. The component 'SignalViolationManager' is tested as soon as it is ready.

5.2.2 Integration testing strategy

It is stated, throughout the document, that the integration strategy for the SafeStreets software is a bottom-up strategy. Here follows a brief description of what this strategy is.

A *Bottom Up approach* is testing strategy that allows it to be as user friendly as it can possibly be and provides a high deployment coverage, especially in early phases of the development. The whole software is broken into small submodules that can be indipendently developed and then, after have been correctly tested, can be linked between each other in order to finalise the software requested. By doing so, the main faulty interactions are easily exposed and can promptly be fixed. Furthermore, by separating the final product in smaller modules, the reusability of said modules can be augmented and simply the process of creation, in both resources and time.

5.2.3 Sequence of component/function integration

The following section shows which components will go through the process of integration for further clarification. As a matter of further explanation, the arrows start from the component which uses the targeted one.

5.2.3.1 Integration of frontend and backend

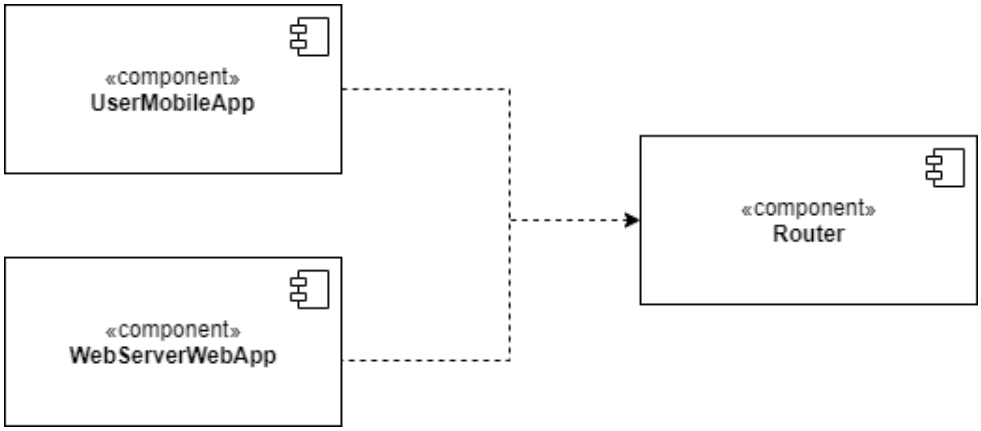

Figure 5.1: Router integration

Figure 5.2: AuthorityUserWebApp-WebServerWebApp integration

5.2.3.2 External services integration

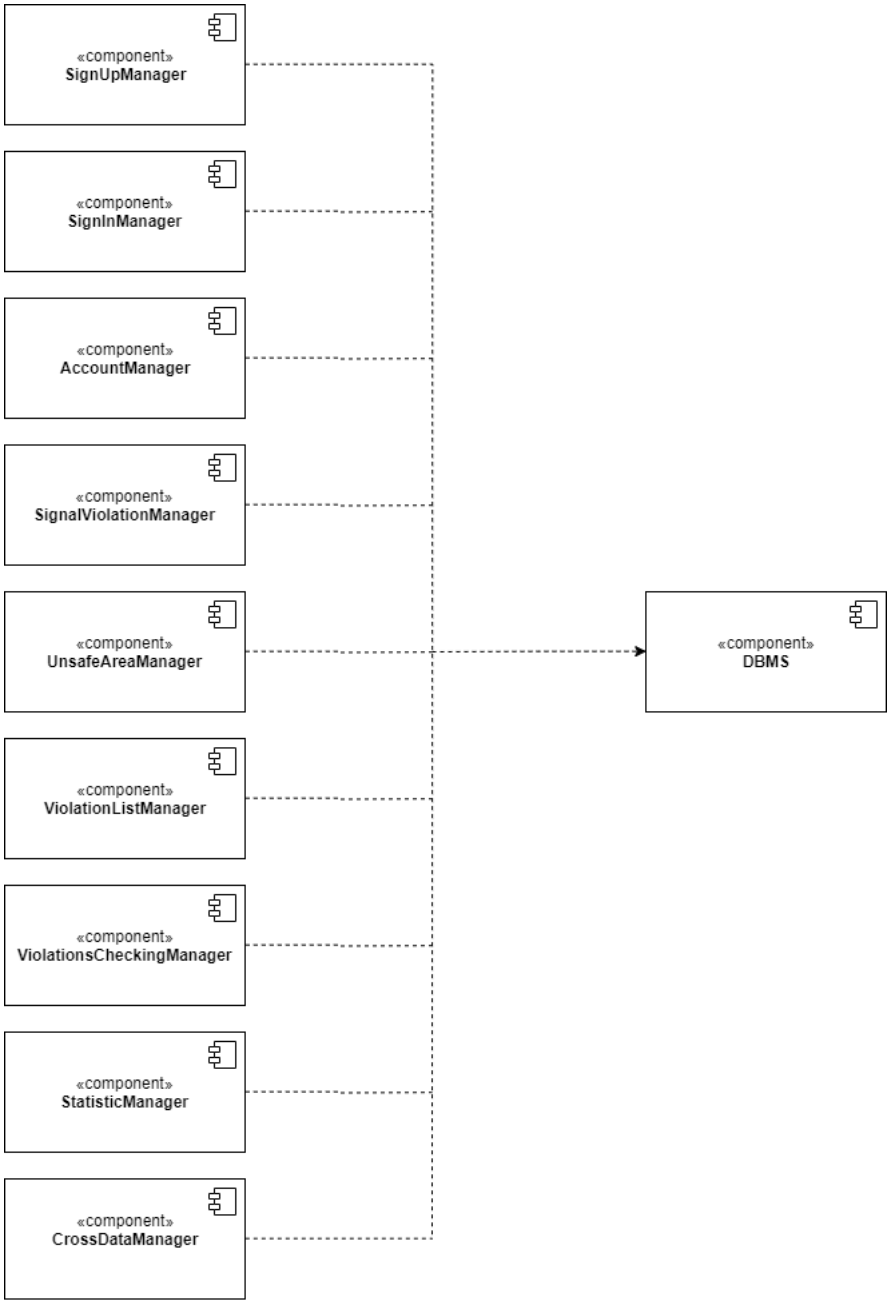

Figure 5.3: DBMS integration

Figure 5.4: MunicipalityDBMS integration

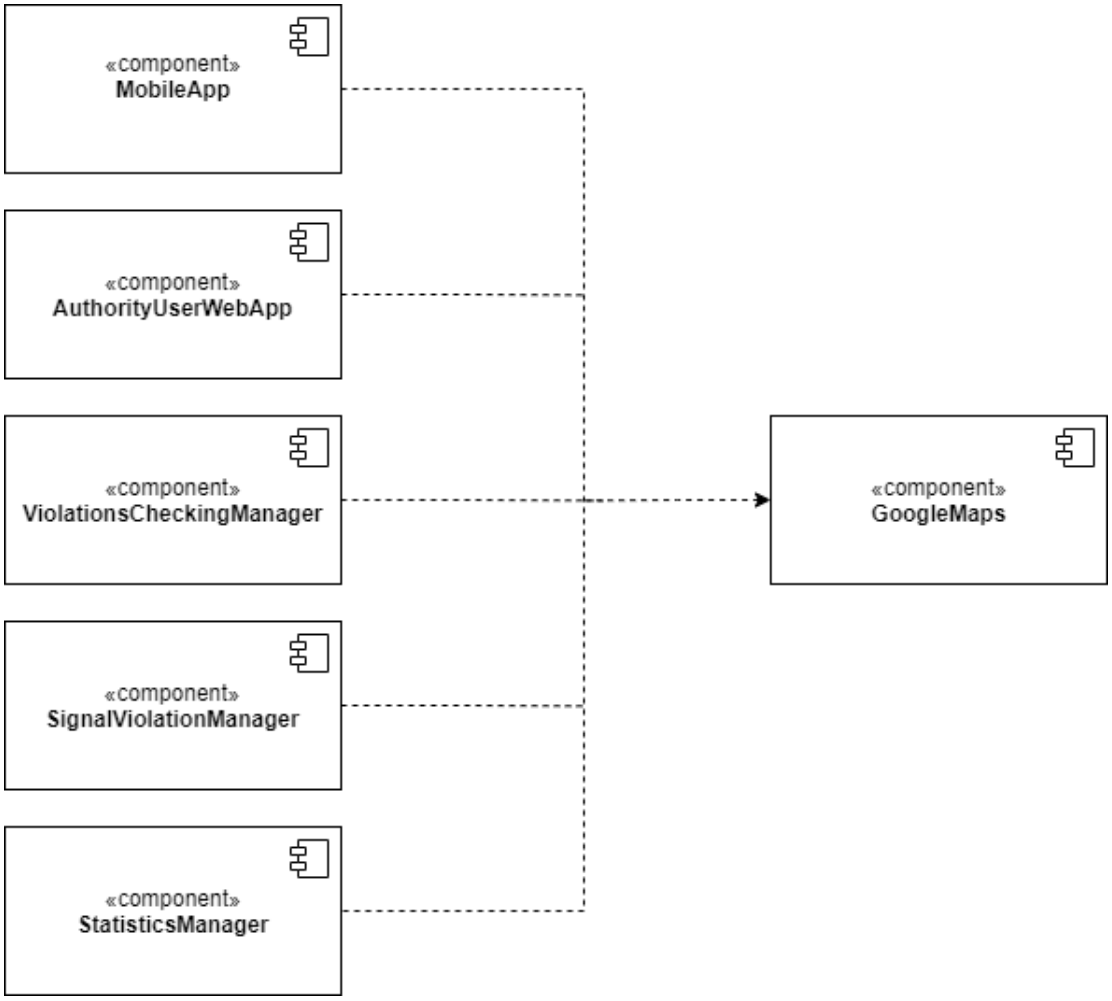

Figure 5.5: Google Maps integration

Figure 5.6: Plate Recognizer integration

5.2.3.3 Internal services integration

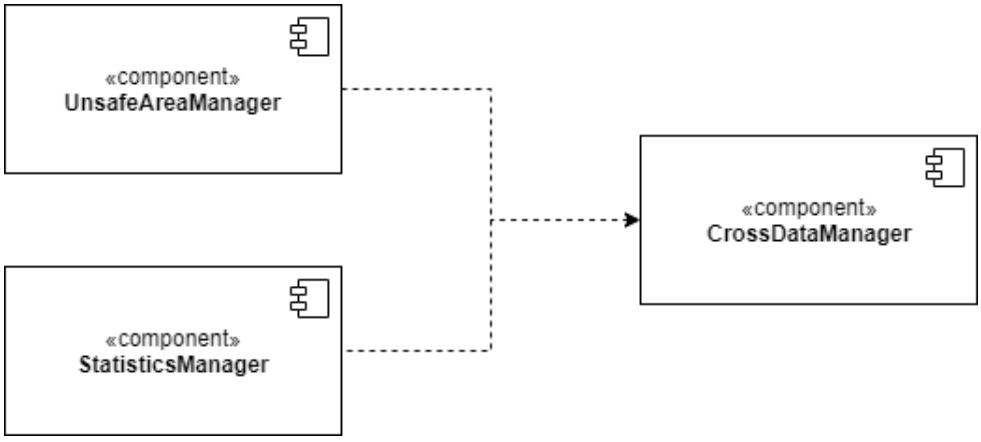

Figure 5.7: CrossDataManager integration

6 | Effort spent

Marco Premi

Chapter	Effort (hours)
Chapter 1 - Introduction	0
Chapter 2 - Architectural design	0
Chapter 3 - User interface design	0
Chapter 4 - Requirements traceability	0
Chapter 5 - Implementation, integration and test plan	0
Total (hours)	0

Fabrizio Siciliano

Chapter	Effort (hours)
Chapter 1 - Introduction	0
Chapter 2 - Architectural design	0
Chapter 3 - User interface design	0
Chapter 4 - Requirements traceability	0
Chapter 5 - Implementation, integration and test plan	0
Total (hours)	0

Giuseppe Taddeo

Chapter	Effort (hours)
Chapter 1 - Introduction	0
Chapter 2 - Architectural design	0
Chapter 3 - User interface design	0
Chapter 4 - Requirements traceability	0
Chapter 5 - Implementation, integration and test plan	0
Total (hours)	0

7 | References

- **Amazon© Web Services**
- **Amazon© Virtual Private Cloud**
- **Amazon© Elastic Compute Cloud**
- **RESTful Architecture**
- **Relational DataBases**
- **Multi-tier Architecture**