POLITECNICO
MILANO 1863

Software Engineering 2

A.Y. 2019/2020

# SafeStreets

## DD – Design Document

Version 1.0

**Authors:**

Daniele Comi 10528029 - 944534
Anton Ghobryal 10501942 - 945577

**Professor:**

Elisabetta Di Nitto

# Sommario

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to give more technical and design details than the RASD about SafeStreets application system. RASD showed a general view of the system and described the functions which supposed to be executed, while DD aims to present the implementation of the system including design viewpoints; such as context viewpoint, composition viewpoint, logical viewpoint, dependency viewpoint, information viewpoint, patterns use viewpoint, interface viewpoint, structure viewpoint, interaction viewpoint, state dynamics viewpoint, and finally resource viewpoint. It also presents in more details the implementation and integration plan, as well as the testing plan. More precisely, the document presents:

- Overview of the high-level architecture
- The main components and their interfaces provided one for another
- The Runtime behaviour
- The design patterns
- Additional details about UI
- Mapping requirements on the defined components
- Implementation plan
- Integration plan
- Testing plan

The purpose of this document is to provide an overall guidance to the architecture of the software product.

## 1.2 Scope

Here's a review of the application's scope that refers to what is already explained in the RASD.

SafeStreets allow users to report a violation to the authorities when they spot one. In order to obtain the ability of using SafeStreets the user will have to register himself into the application system. Users have two different modes to register themselves into the system: the first one is the proprietary authentication which also requires email validation and the second one consists of SPID[9] authentication. Generally, they will have to subscribe with their full name and fiscal code since they're mandatory to be able to fill certain reports. Registered Users obtain points that indicate their integrity through their continuous voluntary participation in order to provide the possibility of achieving the goal of making the streets safer. These points are called integrity points. Initially, users, who have registered with SPID[9], have more integrity points than the proprietary authentication (according to demonstrating more integrity into the society verifying his own identity through a public system of digital authentication). Moreover, when a report is verified by the authorities, integrity points of the notifier increase. Users can see also, through a map, the security level of a zone. Allowing users to mine general information about notified violations doesn't violate the privacy of the reporting user according to the Legislative Decree 196/03[1] and the regulation 2016/679[3] given since they aren't authorized to access other users' private information such as fiscal code, name, surname etc. Security level is calculated being based on the statistics of the types of violations committed in the interested area. Of course, any user

will have the possibility to change the password in case it is forgotten through the normal process of password change link sent to their email address.

As it is in the specification of the S2B, Reports are composed of date, time, position, a note (with a maximum fixed number of  characters) and a clear picture of the committed violation in which the license plate should be included, but it isn't a restricted requirement because, in the worst case, there are two possible situations: in the first one the license plate isn't clear (e.g. poor quality or blurry image) the user is allowed to do one out of two possible actions that consist of re-take the picture of the violation or modify the license plate number, and if the user chooses to do the second action, the system shall recognize the report as one, instead, with a modified license plate number and this induces minor level of credibility; instead, in the second situation, if the system doesn't recognize a vehicle in the taken picture it will take an immediate action to discard this picture and it will eventually ask the user to take a new clearer picture to be able to proceed, and that precludes the fact that user might send pictures that are not in accordance with the domain of the application (e.g. photos that don't contain a vehicle such as selfies).

Since the violation must be notified in real-time domain, the user is not allowed to upload a picture at all. So that, situations as creating a false violation or manipulating data of a certain violation. For the same reason the user is not allowed to modify a photo. If the user notices something that should be mentioned, there's a note that he can fill in briefly with possible observations. Also, the user must have a stable active connection to be able to submit the violation.

A report should satisfy the application domain before it becomes in hands of authorities and in order to realize this fact a report should include the preconditions described earlier. When a report is filled in completely the authorities must be able to receive it through the application. Within this context, the authorities are defined as Italy's law enforcement agencies. The authorities, interested in the application willing to use it for increasing local security, must have a valid digital certificate provided by the police forces through the Ministry of the Interior[4] and the Ministry of the Defense[5]. An authority must register making a formal and direct request to the available Certified Email[6, 7] address of SafeStreets through his Certified Email[6, 7] which will give him in a secure way the credentials generated to be able to use the application or by just using SPID[8] with his Certified Email. The login process with the authority credentials requires also a valid digital certificate. Once an authority is registered and SafeStreets has added his credentials in the system, he will be able to receive notifications about the committed violations. Registered authorities have the maximum authorization to access all the data notified by users. They also have access to all normal user functionalities, thus the capability of reporting violations. The authorities can also verify and validate the visualized reports depending on the veracity of the notified violations. The authorities are also guaranteed a second access to SafeStreets through a web service which require the same login process.

Either the registration process or the reports made and of the user who carried it out are respects the terms established by the Legislative Decree 196/03[1], Legislative Decree 82/05[2] and the regulation 2016/679[3].

SafeStreets offers also the possibility to be an important participant as an independent entity which can provide suggestions to the improvement of a certain area. In order to realize such a functionality, SafeStreets should have access to accident records of the applied areas. Interested municipalities, in order to let the authorities benefit this functionality, must guarantee access to

those data records because it helps the application to cross the provided data about accidents with its own data to provide suitable suggestions depending on the identified situation. It will then notify the authorities regarding those suggestions.

## 1.2 Definitions, Acronyms, Abbreviations

### 1.2.1 Definitions

- **Violation**: a subset of anything that is classified as a traffic violation by the Traffic regulation and laws document. This subset is composed of:
    - Double line parking
    - Expiry of the parking time limit
    - No parking area
    - Parking in places reserved to people with disabilities
    - Parking in the middle of bike lanes
    - Parking near bus stops
    - Parking on crosswalk
    - Parking on residents reserved spots
    - Parking ticket missing
    - Possible vehicles damage by third parties (e.g. broken glass)
- **Vehicle**: any terrestrial identifiable vehicle subject to Traffic regulation and laws document, like cars, motorbikes, trucks, etc…
- **User**: any citizen registered in the system who is using any of SafeStreets functionalities.
- **Violation report, notification**: acknowledgment in SafeStreets system of a new violation occurred.
- **Authority**: any registered law enforcement using SafeStreets application alongside its authority-restricted functionalities
- **Municipality**: any central administration of a city or a town which may or may not give open access to its incidents data.
- **Reliability score**: score assigned to any user account which gives a sense of how much a user I reliable in giving information regarding violations.
- **Safe area**: a low radius geographical area where violations are lower than a certain threshold or lower than other areas.
- **Suggestion**: an automatically inferred hint given to the authorities by SafeStreets regarding how they could improve, with the help and permission of their municipality, area marked as high-risk area due to a high correlation of violations and incidents reported from the same municipality. Possible suggestions are:
    - Add a barrier between the bike lane and the part of the road for motorized vehicles
    - Install a towaway zone sign
    - Increase parking slots
    - Increase local police controls
- **Galileo**: Global localization system based on a network of 24 satellites commissioned by European Union and ESA (European Space Agency)

- **SPID**: is the unique system of access with digital identity to the online services of the Italian public administration and of private members: citizens and companies can access services with a unique digital identity in a secured way
- **Certified Email**: A certified email is an email that can only be sent using a special Certified Email Account provided by a registered provider. When a certified email is sent, the sender's provider will release a receipt of the successful (or failed) transaction. This receipt has legal value and it includes precise information about the time the certified email was sent. A certified email account can only handle certified email and can't be used to send regular email.
- **Design viewpoint:** The specification of the elements and conventions available for constructing and using a design view.
- **Amazon Web Services:** The public cloud system which comes in help to SafeStreets for recognizing objects and inferring data.
- **Cloudflare:** The system which enhances SafeStreets response time and fault tolerance on the web servers side.
- **AU10TIX:** The system which SafeStreets will use to automatically check identification documents validity.

### 1.2.2 Acronyms

- **API**: Application Programming Interface
- **AWS**: Amazon Web Services
- **AVD**: Android Virtual Device
- **D.L.**: Legislative Decree
- **DCPM**: Decree of the President of the Council of Ministers of the Italian Republic
- **DD**: Design Document
- **DDoS**: Distributed Denial of Service
- **DMV**: Department of Motor Vehicles
- **DMZ**: Demilitarized Zone
- **DNS**: Domain Name System
- **EEA**: European Economic Area
- **EU**: European Union
- **GDPR**: General Data Protection Regulation
- **GPS**: Global Positioning System
- **IEEE**: Institute of Electrical and Electronics Engineers
- **QL**: Query Language
- **S2B**: Software to Be
- **SMTP:** Simple Mail Transfer Protocol
- **SPID**: Public Digital Identity System
- **UI**: User Interface

### 1.2.2 Abbreviations

- $G_n$ = $n^{th}$ goal
- $D_n$ = $n^{th}$ domain assumption
- $R_n$ = $n^{th}$ requirement

## 1.4 Revision history

- Version 1.0: first release
- Version 1.1:
    - Improvements in the test plan section
    - Improvements in the description of the component interfaces

## 1.5 Reference Documents

- D.L. 196 of 2003 (196/03) https://www.camera.it/parlam/leggi/deleghe/Testi/03196dl.htm
- D.L. 82 of 2005 (82/05) https://docs.italia.it/italia/piano-triennale-ict/codice-amministrazione-digitale-docs/it/v2017-12-13/index.html
- General Data Protection Regulation (EU) 2016/679 https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679
- IEEE 1016-2009 - IEEE Standard for Information Technology--Systems Design--Software Design Descriptions https://standards.ieee.org/standard/1016-2009.html
- Specification document "Mandatory Project Assignment AY 2018-2019" https://polimi365-my.sharepoint.com/:b:/g/personal/10528029_polimi_it/EXR1gN6gBoxJgMC86Ow45gMBFwZzkRSWuoaf5K7t1wZutA?e=SPnVkI
- Ministry of the Interior and digital certificates released http://politichepersonale.interno.it/itaindex.php?IdMat=1&IdSot=35&IdNot=386
- Ministry of the Defence and digital certificates released http://www.pkiff.difesa.it/#secEN
- Certified Email https://www.agid.gov.it/it/piattaforme/posta-elettronica-certificata
- Certified Email RFC https://tools.ietf.org/html/rfc6109
- SPID https://www.agid.gov.it/it/piattaforme/spid
- Italian license plate verifier http://www.targa.co.it/data/doc.aspx
- Police State license plate verifier https://www.crimnet.dcpc.interno.gov.it/crimnet/ricerca-targhe-telai-rubati-smarriti/FAQ
- RASD
- AWS https://docs.aws.amazon.com/
- Cloudflare https://developers.cloudflare.com/docs/
- Google Firebase https://firebase.google.com/docs
- AES https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
- RSA https://community.rsa.com/docs/DOC-60094
- Argon2id https://www.cryptolux.org/images/0/0d/Argon2.pdf
- Oracle DB https://docs.oracle.com/en/database/
- QSM http://www.qsm.com/resources/functionpoint-languages-table
- AU10TIX https://www.au10tix.com/
- GraphQL https://graphql.org/
- Message passing protocol in Bayesian Belief Networks http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf
- SonarQube https://docs.sonarqube.org/latest/
- Jenkins https://jenkins.io/
- JUnit https://junit.org/junit5/docs/current/api/
- Intel HAXM https://github.com/intel/haxm

- Android Studio and tools https://developer.android.com/studio/profile
- Arquillian http://arquillian.org/
- JMeter https://jmeter.apache.org/
- Telerik Test Studio https://docs.telerik.com/devtools/teststudiodev

## 1.6 Document Structure

- **Chapter 1** is an introduction to the design document. Its goal is to explain the purpose of the document and to highlight the differences with the RASD, whilst showing the link between them.
- **Chapter 2** aims to provide a description of the architecture design of the system, it is the core section of the document. More precisely, this section is divided in the following parts:
    - Overview
    - Component view
    - Deployment view
    - Runtime view
    - Component interfaces
    - Selected architectural styles and patterns
    - Other design decisions
- **Chapter 3** specifies the user interface design. This part is already contained in the RASD in the mockups' section. However, we decided to insert some UX diagrams to better describe the interaction between the customer and the application.
- **Chapter 4** provides the requirements traceability, namely how the requirements identified in the RASD are linked to the design elements defined in this document.
- **Chapter 5** includes the description of the implementation plan, the integration plan and the testing plan, specifying how all these phases are thought to be executed.
- **Chapter 6** shows the effort which each member of the group spent working on the project

# 2. Architectural design

## 2.1 Overview

The SafeStreets system to be developed is a distributed application following the multitier architecture paradigm with a completely scalable multitier and data tier as shown in the Figure 1. The architecture is basically composed, without going in details yet, with three main layers which are the Presentation Tier, the Middle tier and the Data Tier. The Presentation Tier is the layer near the user where information is presented, and the user can start or receive interaction with SafeStreets. The middle tier, as will be better explained later, is the layer managing all the application and business logic behind SafeStreets coordinating all its functionalities. Finally, the Data Tier is the layer which its purpose is data storage of the SafeStreets system. So, all these different hardware layers that represent different computers and servers needed to do the respective tier work. This architecture will grant the system the characteristics of high scalability and high flexibility as it will be shown later.

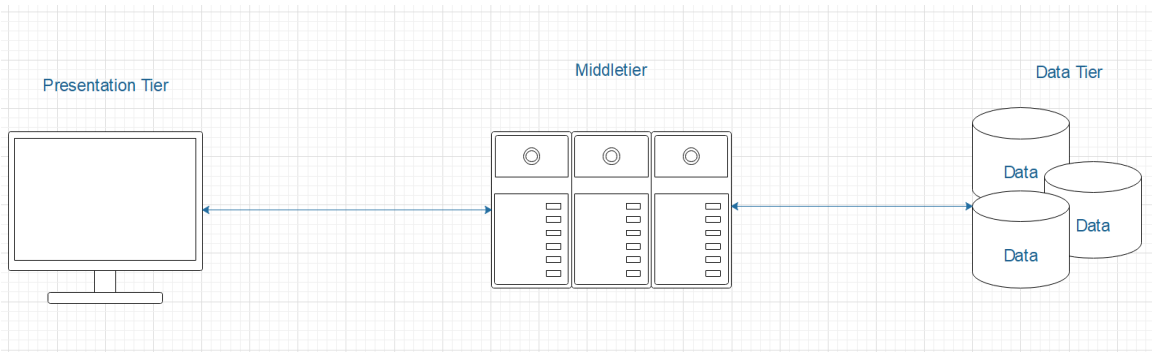



*Figure 1 – High level multitier architecture*

Now, going in more details as shown in the Figure 2, a general diagram but not yet precise in every part of the architecture we can see starting from the Presentation Tier that is represented by the users using the application which can be Authorities or Normal users. As shown Normal Users will only be able to access SafeStreets through a mobile application where the Authorities will also have the possibility to use a SaaS through web access. Their requests towards SafeStreets will be asynchronously or synchronously depending on which action they will take in account. After sending a violation report they will be able of course to use any other kind of services offered by SafeStreets, as it will happen when for example using the Safeness are map functionality. Instead when composing a violation report in the process of taking the violation picture, they will have to wait, so a synchronous interaction, SafeStreets to control the validity of the image before proceeding in any further composing of the

violation report. Also, interaction starting from SafeStreets to users are asynchronous, when for example sending to the selected Authorities notification on new reported violation by some users. This notification interaction will be better described later.

Moving on the middle tier as shown in the Figure 2 is highly scalable and flexible. Application servers and web servers are also decoupled to acquire even more stability and more security which will be later discussed. Even if the middle tier shows for clarity reasons only three distributed aggregated nodes in the diagram, it will be able to have many more of them. Each distributed node represents indeed a set of server geographically dislocated, in terms of hundreds of kilometers distant with each other in order also to avoid that natural causes could harm multiple clusters. Each of them will so have different static IP addresses assigned by IANA, referencing the third level of the TCP/IP stack, but associated to the same DNS records. DNS, not part of the architecture, will be able to choose which node is better to initiate communication to. But this is not all, all main nodes containing various servers will have a load balancer to allow the workload to be correctly distributed without overwhelming any server during heavy workloads. Both web servers and application server instances which will be discussed later, are thought to have different kind of services or even replicated services that can work in parallel: a parallelized system will allow SafeStreets to get the requested availability level for each functionality allowing it to have a high fault resistance characteristic.

The middle tier will have also access to external services provided by third parties to allow SafeStreets functionality to work correctly. As shown, we can spot the Google Firebase[14] service, the SPID[8] service, the License Plate API[9], the Police State services[10], the CA/TA Authorities for digital certificates, the Maps services and the Municipalities incident data access. All these various third parties will offer specific APIs used in the middle tier to access their services.

It is also present a Cloud architecture instantiated remotely on Amazon Web Services which will be used to work with heavier workloads regarding the image validity verification, license plate OCR system and the suggestions functionality of SafeStreets. As shown this deployed architecture to be defined during its instantiation will have an elastic load balancer which will grant to deploy any new hardware needed to keep the workload under control granting SafeStreets to function normally even when there are lots of data or requests regarding these functionalities. From the diagram can be spotted various Amazon SageMaker deployed instances like for Convolutional Neural Networks, for Natural Language Processing, for OCR or Bayesian Networks: their need will be explained in the further sub chapters.

Security is a topic even in the architectural design, where later on will be discussed security in software and network terms, in the architecture to better secure SafeStreets WAN there are present various Hardware Firewalls between Internet accessed by clients and the first layer they will encounter which is the web servers which are responsible for exchange information even if not in a direct browsable format for every users as explained before. This second hardware firewall is between this previous sub layer of the middle tier and the distributed application server layer: this will create a DMZ, Demilitarized Zone, for the application servers so that the external network can access only to the resources exposed in the DMZ.

The web servers are not guaranteed the same level of security because their functionality is the management of information representation and their forwarding or receiving to/from

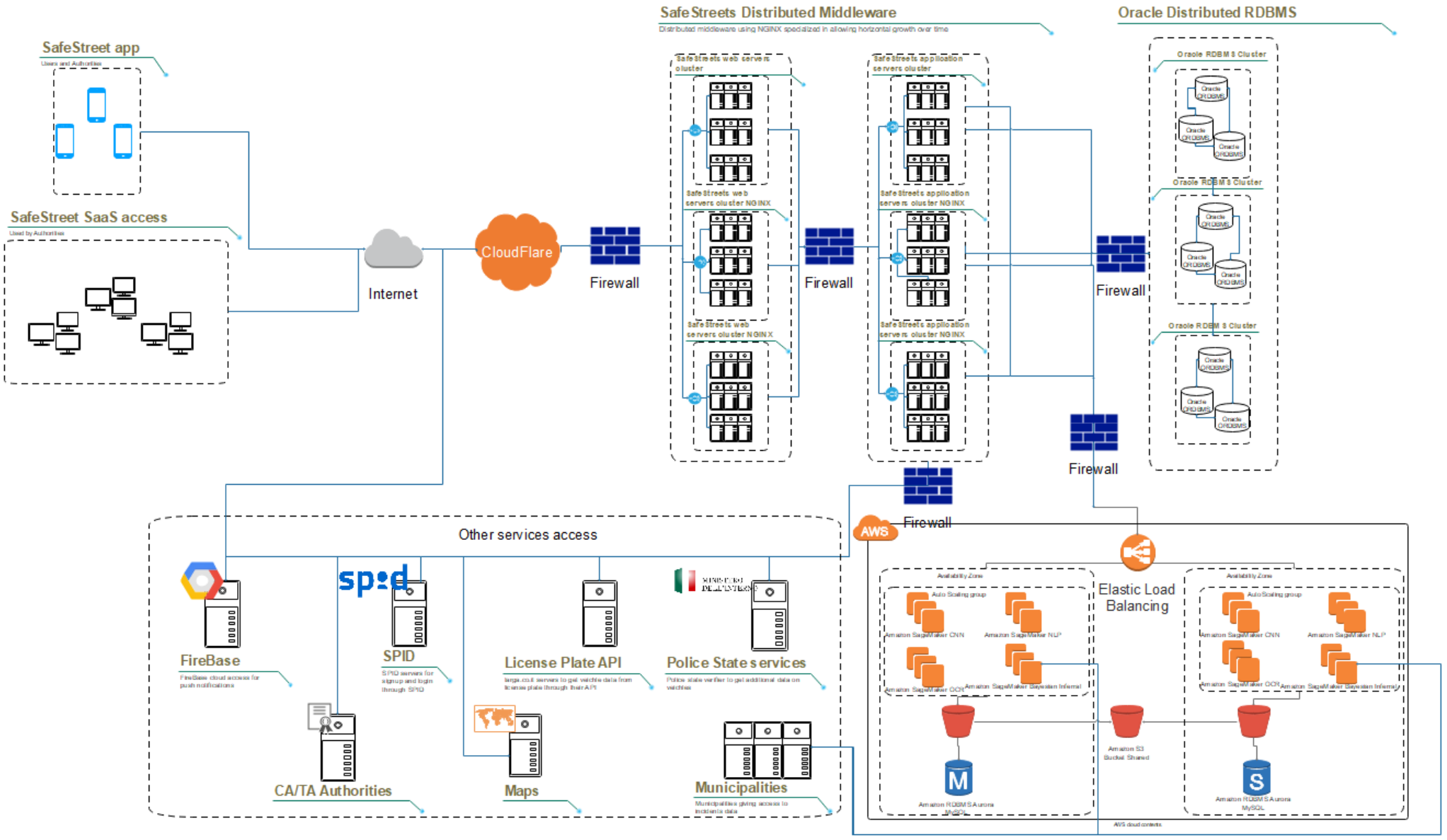
users, and upon functionalities requests they will forward the requests to the application servers in the DMZ in a distributed way letting the DNS and the load balancer of the distributed application servers choose the most suitable node. This level of security is required, as mentioned, since the offered service deals with sensitive data of the users. The lower level of security of the more exposed layer of the distributed web servers is covered by the used service of Cloudflare[13].

Cloudflare[13] can solve this issue and add also more services like DDoS protection, a Web application firewall, an Authoritative DNS and a Content delivery network. Their specific details will be discussed later.

On the last layer called Data Tier we can find a distributed DBMS, Database Management System, based on Oracle DB which will grant a geographically advantageous access to data depending on user request and on the middle tier requests. Their specific details will be discussed later but it's important to say that a hardware firewall between each layer of the multitier architecture is needed to get more security as possible. Indeed, even between the second sublayer of the middle tier and data tier there is a need of the firewall to better protect what's, already secured, inside database: user and SafeStreets data.

Of course, to make the middle tier correctly communicate, there is the need to adopt a middleware. Where clients communicate through one of the distributed web servers, the application servers need to communicate with various heterogeneous distributed systems and remote third party services. So, a middleware is present, and it'll be using some already existing frameworks in order to have a correct communication between the distributed web servers, the private cloud and the distributed DBMS to the middle tier. To grant better performances the various network nodes associated with the distributed web servers but also with the distributed application servers will be better indexed through the best routing and switching algorithm available in terms of software defined networking of SafeStreets. To speed up the finding of the various nodes available it will be used, and configured, at a lower level in the ISO/OSI (TCP/IP) stack than the presentation level, an OpenFlow switch.

*Figure 2 – General system architecture of SafeStreets*


SafeStreets Distributed Middleware
Distributed middleware using NGINX specialized in allowing horizontal growth over time
Oracle Distributed RDBMS
SafeStreet app
Users and Authorities
SafeStreet SaaS access
Used by Authorities
Internet
CloudFlare
Firewall
Firewall
Firewall
Firewall
Firewall
SafeStreets web servers cluster
SafeStreets web servers cluster NGINX
SafeStreets web servers cluster NGINX
SafeStreets application servers cluster
SafeStreets application servers cluster NGINX
SafeStreets application servers cluster NGINX
Oracle RDBMS Cluster
Oracle RDBMS Cluster
Oracle RDBMS Cluster
Other services access
FireBase
FireBase cloud access for push notifications
SPID
SPID servers for signup and login through SPID
License Plate API
Police State services
CA/TA Authorities
Maps
Municipalities
Municipalities giving access to incidents data
AWS
Elastic Load Balancing
Availability Zone
Auto Scaling group
Amazon SageMaker CNN
Amazon SageMaker NLP
Amazon SageMaker OCR
Amazon SageMaker Bayesian Inferral
Amazon S3 Bucket Shared
Amazon RDS MS Aurora MySQL
AWS cloud contents

## 2.2 Component view

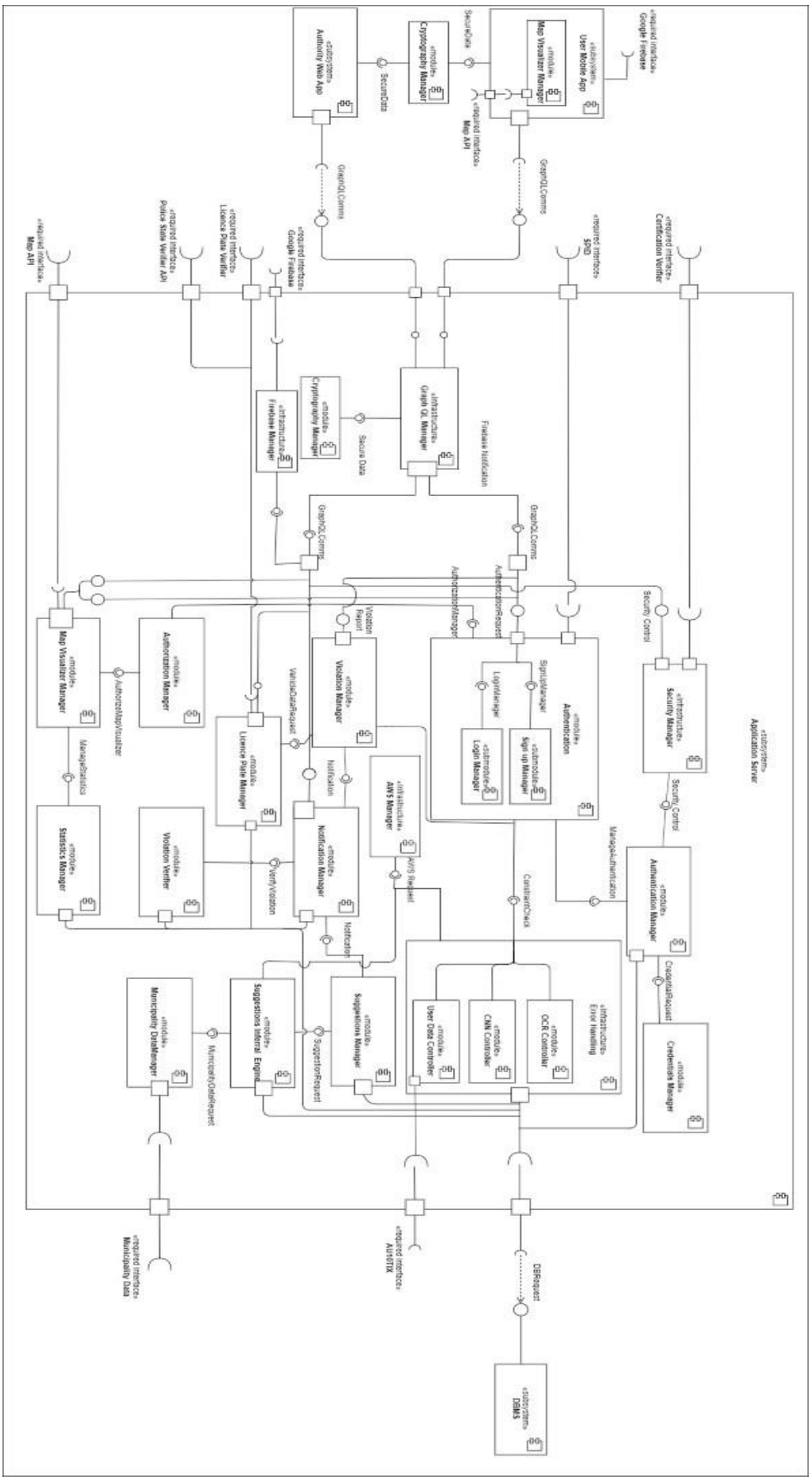


*Figure 3 – Components View diagram*

Here all the components will be described in detail, there is the need to say these components are obviously composed of further elements such classes which will be better described in the chapter 2.5 using nearly after the component interfaces a main UML Class Diagram which will include components and their detailed design.

- **User Mobile App**: It represents the mobile application Graphic UI. Earlier, as mentioned, it could be used by both normal user and authority. It incapsulates **Map Visualizer Manager** module as it's necessary for map management and development. Such UI allows the user report violations and make requests for safeness map and the visualization of statistics which describe the percentage of violations and violation types committed in the requested zone (user's current location by default). The users are also allowed to possess a profile on which, in addition to provided personal information, there's his reliability points which is proportional to how many violations he reported correctly and the same was verified by authorities. Only normal users (not authorities) can sign up from the application but both types of users can login from the application once registered.
- **Authority Web App**: It represents the system on which authorities request for multiple operations. This component is mandatory for security reasons as its main (and most important) functionality is authority recognition. Authorities use this API to sign up providing a valid digital certificate provided by the minister of intern or its subordinates. Then, it could be also used for all the other functionalities as the application as they're described in the RASD[11].
- **Map Visualizer Manager**: This module is used to manage the visualization on maps requests of the user. Those requests can be a safeness map which provides information about types of violations committed in several areas and describes how much safe an area is with a colour grading joint with various available statistics. From the user point of view, it will be a map where major violation areas are pointed out with a respective indication of both the frequencies of violations in a certain area and their importance. Instead, from the application server point of view all this data will be delocalized in terms of components in a single server and will be organized together by this very component, which will ask the support of other components which will provide data and business logic. Generally, it requires an external interface which provides a **Maps API** that could facilitates the visualization of the output on a geographical map.
- **Cryptography Manager**: Since the user provides private information at every transaction such as the content of the violation report, this component provides end-to-end encryption which means SafeStreets and third parties can't read the content of each transaction; this includes also the registration content. It could adopt an encryption standard such as RSA[16] or AES[15]. AES is slower but it's recommended since the information is provided to recognized authorities such as local police.
- **Graph QL Manager**: It's a server runtime for executing queries by using a user-defined type system. Since it isn't tied to database or storage engine, it'll be backed by SafeStreets existing code and data. This module will drastically facilitate user identification since graph QL service can be used, also, to identify the logged in user. This module uses functionalities of Google Firebase[14] and because of that, it makes use of the **Firebase Manager** module; this will be explained with further details later.
- **Security Manager**: Since it's an entity that requires an external interface, it's been considered an infrastructure because it requires communication between the application server and a certification authority in order to recognize the certificate provided by authorities.

- **Credentials Manager**: This module manages the storing and generation of the credentials of a user, including the generation of the credentials that will be provided to recognized authorities and their Certified Email upon successful web app registration, and also to generate a valid unique code to create a valid URL containing a GET request to reset on demand a user password or in case it was lost.
- **Authentication**: This module is essential to cover both sign up and sign in sessions. From a normal user prospective, it's used for both signing up and signing in; instead, from an authority prospective, it's used only for signing in. For the user type identification, normal user or authority, this module makes use of **Authentication Manager** module. This module requires an external interface in order to provide the possibility to be recognized via SPID[8]. This module makes use of the **Error Handling** module in order to verify that the data provided by the user are compliance with the requirements (fields need to be filled). It also interacts to AU10TIX for the validation of the identification documents.
- **Authentication Manager**: this module manages the user type identification and it's an intermediator between the **Authentication** module and the **DBMS**. It controls the validity of the provided credentials at every login and so also manages the sending through the internal SMTP server of the registration confirmation link and the one for a password reset request. At the registration via web app when authority provides a valid certificate, this module employs **Credentials Manager** to generate credentials in order to send those credentials back to the authority. Such credentials can be structured as two random strings: username and password. Those credentials can be used to sign in SafeStreets. The hashed password is necessary in order to be able to login, but the username is not. An authority can use a provided username to login with, in addition to the password provided by this module.
- **Violation Manager**: This module represents the handler of the report with its required fields and communicate the non-compliant fields to the user. This module makes use of the **Error Handling** module in order to verify that the data provided by the user are compliance with the requirements (fields need to be filled); it uses two main modules to guarantee the correctness of the report, which OCR and CNN controllers.
- **Error Handling**: This infrastructure recognizes the non-compliances to the requirements of SafeStreets. It contains three main modules which are **OCR Controller**, **CNN Controller** and **User Data Controller**. The **OCR Controller**, given a picture, recognizes automatically the license plate and auto-fills respective field. The **CNN Controller**, given a picture, recognizes automatically if it contains a vehicle. **OCR Controller** and **CNN Controller** employs **AWS Manager** module in order to provide the functionalities mentioned earlier; they also refuse the provided information if those aren't compliant to requirements, such as a non-containing vehicle (or license plate) picture. Finally, the **User Data Controller** controls if any of the provided data from the user are compliant to the requirements and further additional definable rules.
- **Authorization Manager**: It's an intermediator between user and **Map Visualizer Manager** module because the authority has more visibility privileges of some attributes, such as license plates, than the normal user. Given a generic user, this module recognizes the user type and provides access to selected attributes of the viewed violations on the map based on the user type.
- **Statistics Manager**: This module calculates the statistics of safeness of certain area based on the existing data of SafeStreets database. The choice of the used models, distributions and confidence intervals and so on can be chosen and modified freely in the implementation side.

- **Violation Verifier**: When an authority verifies a violation, this module works as intermediator between the authority verification and SafeStreets database in order to update and manage the reliability score of the user therefore to the verification action, and the violation status to verified.
- **License Manager**: This module works as an intermediator between SafeStreets violation reporting and storing, and the two required external interfaces **License Plate Verifier** and **Police State Verifier API**. Such interface works as an intermediator between SafeStreets application server and DMV and Police State database in order to give the possibility to authorities to mine information about the license numbers provided by SafeStreets.
- **Municipality Data Manager**: This module standardizes the format of the data provided by the municipality and prepare data for the **Suggestions Inferring Engine** in order to cross the data provided by the municipality with SafeStreets data and so also, how the data is retrieved from the various municipalities.
- **Suggestions Inferring Engine**: This module handles the data crossing and in order to handle a huge amount of data it uses services provided by AWS[12].
- **Suggestions Manager**: This module handles the extraction of suggestions data, after the periodic data crossing, and prepare the data to be provided to the authorities as possible suggestions in certain area.
- **Notification Manager**: This module prepares the data to be notified, such as reported violations and possible suggestions, then passes the prepared structures to **Graph QL Manager** module in order to handle the actual notification to the selected authorities through the **Google Firebase Manager**. It will also have the job to create a queue for each violation notification where the nearest authority will be first notified and then if he'll refuse by not opening the notification after a certain empirical time it will send the notification to the second authority and so on.
- **AWS Manager**: This module handles all the communication to AWS[12] which will be using the provided AWS APIs[12] to let SafeStreets get but also order new computed data from the various deployed models in Amazon SageMaker[12] such as the CNN to recognize images, OCR for license plates reading, NLP and Bayesian Network for better suggestions and so on.
- **Google Firebase Manager**: This module handles all the communication to Google Firebase[14] which is the cloud platform for handling push notifications. It will be using the provided Firebase APIs to let SafeStreets send a notification requests to Firebase which will be handling the correct delivery to the users. It will also handle incoming response which are basic for what concerns SafeStreets and they are just the fact that an authority has opened/accepted the notification of the violation and is going to verify it.

## 2.3 Deployment view

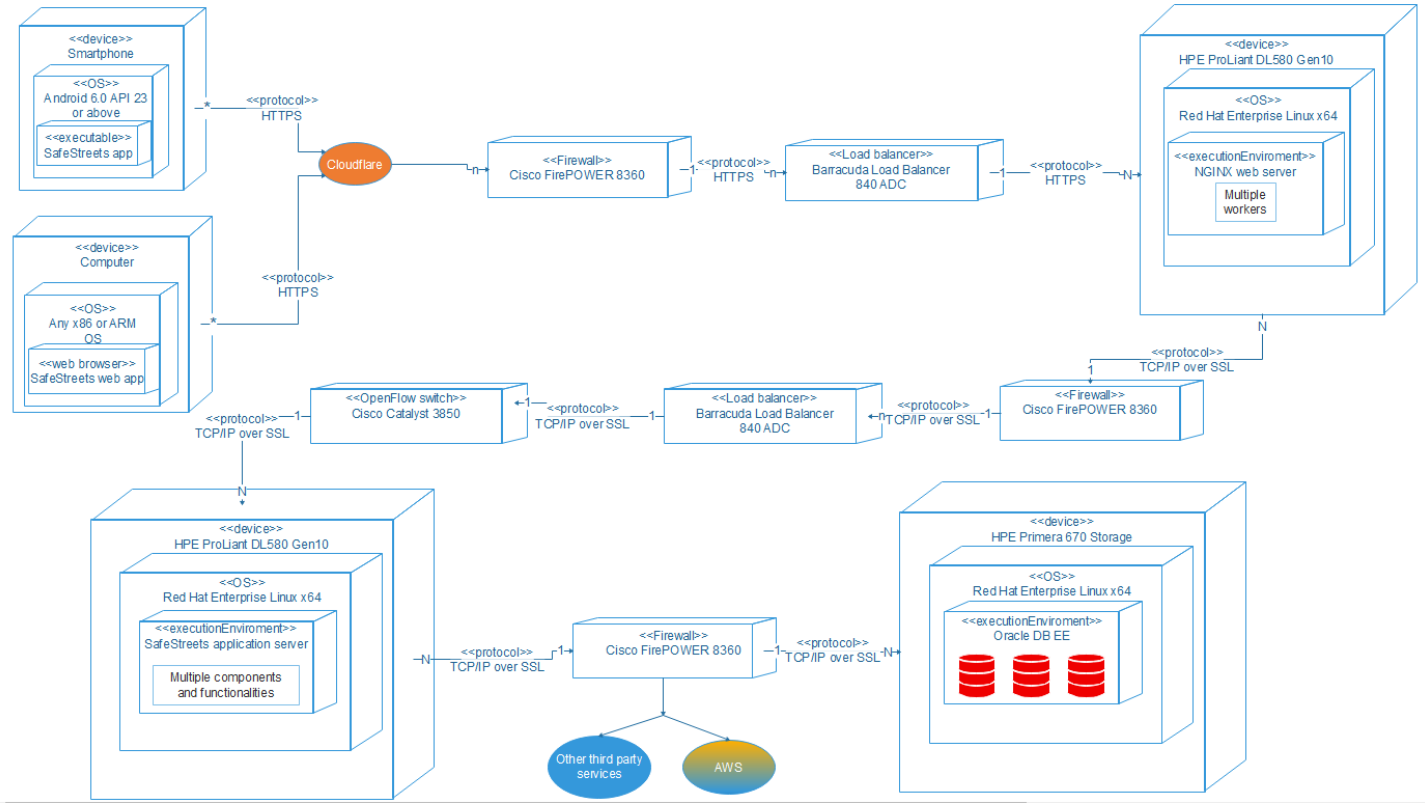



*Figure 4 – Deployment diagram*

The deployment diagram shows how is intended the physical deployment of artifacts on nodes.

In the Figure is shown the deployment diagram of SafeStreets, note that it is not displayed anything which is not under control at the design level so, for example, no deployment node of third-party services or of the public cloud.

Starting from the first tier where clients are located, we can see two kind of deployment are available as already known.

The computer access will be deployed using an internet browser running on any hardware platform x86 or ARM available. It can be so generalized on different hardware deploying platforms because of its kind of interaction only through a web browser.

The users in general, comprehending the authorities, instead will have to access SafeStreets through a smartphone application. SafeStreets app will be deployed as executable using as operating system any Android ARM x64, ARM x32, x86 or x64 compatible hardware running Android 6.0 Marshmallow API 23 or higher. The Android application will be better explained in its details in the Chapter 5 of Implementation, integration and test plan. What can be already said is that it will use the Android SDK available with last APIs. It will have an Android activity for the main screen and other sections will be under another activity or an Android fragment. Any other functionalities needed to run into background or upon notifications will use Android services and Android broadcast receivers.

Moving to the middle tier the first device indicated as node in the deployment diagram it's the external firewall. It'll be deployed using a Cisco Firepower 8360 running on its IOS allowing a bandwidth of 60 Gbps. This will grant the best security in terms of unauthorized access to the network of the various servers. This very deployment will be the same in the various firewalls. Both web servers and application servers will be running their various executable services of SafeStreets on Linux Red Hat Enterprise x64 operating system in an HPE ProLiant DL580 Gen10 running multiple scalable Intel Xeon 9200 series with up to 12 TB of DDR4 RAM and 512 GB of static RAM. The web server will be running NGINX natively or on VMware server instances to guarantee the required performance levels thanks to, first the various virtualized instances which reduces drastically the cost of failures and increment drastically the availability of the services and secondly thanks to NGINX internal structures using various kind of workers. The communication between web servers to application servers and from application to the, till to be analyzed, data layer deployment diagram will be done through a secure TCP channel, while, if needed, distributed communication between application servers will be done using already existing frameworks such as Remote Procedure Calls protocols using Java EE RMI. The communication with the Oracle DB will be done by using the Java EE RDMI and the Java EE Persistence APIs to communicate with an object/relation mapping fashioned way to the DBMS. To gain ulterior enhance on performances of the middle tier, an OpenFlow controller switch to better manage internal cluster packets exchange through various switches which are controlled by the SDN defined in the controller which will use a Cisco Catalyst 8350.

The third layer corresponding to the data tier has various kind of nodes corresponding to the various distributed DBMS. As it will be stated they'll be running Oracle DB[18] as DBMS to guarantee very high performances. Indeed, the DBMS has been chosen because of the high relational level of the tables for which data finds a secure storage. Data such as every proprietary authentication encrypted data, or all the violations and so on like it will be displayed, in the further pages, by the ER model. A DBMS is also needed due to its powerful skills of being very good in optimizing queries, especially the ones handling tons of data. It will use their algebraic representation together with internal statistics distributions of the data to make them faster and simpler to be executed, on an already fast DBMS. SQL indexes will also be set using specific main and secondary memory structures to guarantee the best performance available. The DBMS, which is coherent with respect to the ACID properties, will also have various triggers to guarantee the perfect balance of data consistency and business logic implemented in the DBMS to lighten the work on the middle tier when possible.

The various hardware available in the second and third tier for the web and application server and for the distributed DBMS servers will have enough computational power, main and mass memory to handle everything. As computational power will be used HPE Primera 670 Storage based on Intel Xeon 9200 CPUs with about 6 PB of storage divided up between very high quality SSDs and HDDs, with also 1 TB of dedicated cache and 32 Gbps maximum throughput. Every Database Server will have a level 6 RAID system.

What was not already explained was the load balancer. Its instance will be deployed on a third party system because of the adoption of a hardware load balancer solution like for the various firewalls. It will be used the Barracuda Load Balancer 842 ADC which is the best suitable for large systems which can have very high traffic requests to handle in the fastest

way possible having the maximum theoretical throughput at a rate of 20 Gbps.

## 2.4 Runtime view

The following sequence diagrams want to focus on the main and critical points of different use cases listed in the RASD, for a clarity purpose some trivial conditions are omitted like the fact that there is a need for internet access, such trivial cases are already covered in the use cases' exceptions list already fully shown in the RASD. Such diagrams are designed such that there's a hard connection between them and both component diagram and UML diagram as it's between component diagram and UML diagram. The following sequence diagrams shows mainly the interaction between the user and the mobile application, the interaction between the mobile application and the application server, the logic behind the application server, the interaction between the application server and the external interfaces which are needed in order to provide additional functionalities, and finally the interaction between the application server and the DBMS.

Will be also shown GraphQL used through the GraphQL manager which will be able to understand incoming external requests from users and to route them to the right components in order to build a proper response.

### 2.4.1 Signup Normal User

The first time the normal users open the application, they must sign up in order to use the functionalities provided by the application. When a normal user requests a sign up, the application must show two alternative and exclusive choices that are accessible for the sign-up process. The first one is the proprietary mail. The first step, the application controls if the user already exists in SafeStreets DB; and if so, the mobile application returns such fact in addition to offer the possibility of logging in the system or signing up with possibly another different data. The second step, the user provides to the application an existent e-mail, in addition to all the required documents, such as a picture of the ID documents and the fiscal code. The third step, the application server controls if the user provided an ID document and a fiscal code document using AU10TIX service through their APIs to recognize the type of the provided document, and the user eventually succeeds to sign up only if he passes this process. The second choice is signing up through SPID. When the user selects this alternative, the application must forward the user to another interface, SPID external interface, on which he can directly log into the system.

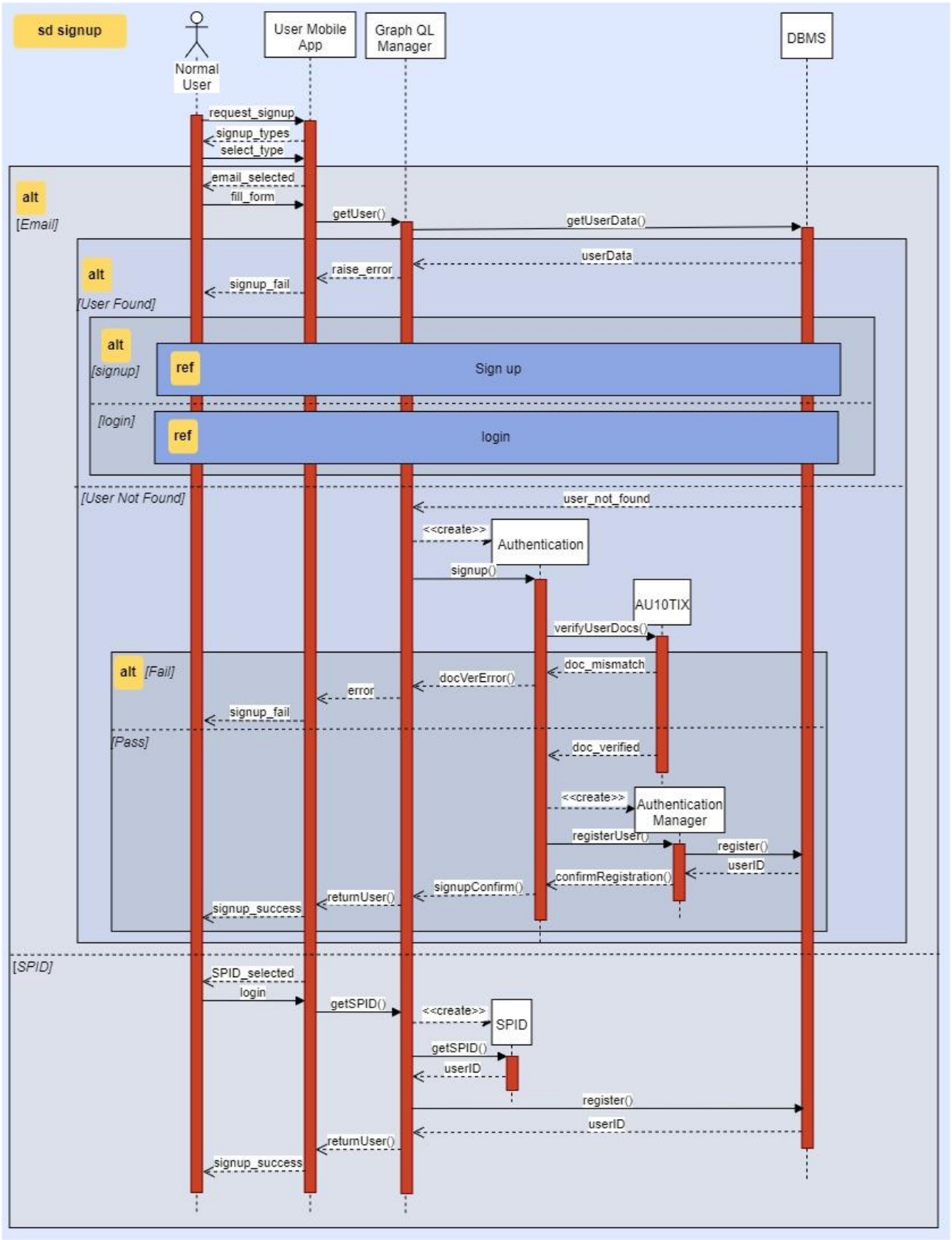



*Figure 5 – sign up normal user sequence diagram*

### 2.4.2 Sign-in Authority

The first time an authority signs into the system, they must use the web application. An authority can't sign in the first time from the mobile application. This design choice is made in order to provide more security as it's been already explained in this document and the RASD. For an authority to sign into the system for the first time, a valid digital certificate must be provided. If the provided digital certificate exists already in SafeStreets DB, it means that the authority who requested to sign-in as a first time already exists in the DB; moreover, the system has already provided to the authority valid credentials to login, so the authority should login instead. If the provided digital certificate doesn't exist in SafeStreets DB, the application server controls if the provided digital certificate is actually valid through an external interface which connects SafeStreets to a certification authority or a trusted authority. If the provided digital certificate is valid, the system generates random sequences to be used as credentials to login directly into the mobile application, otherwise the authority must provide a different certificate, which is valid, in order to complete the sign-in process.

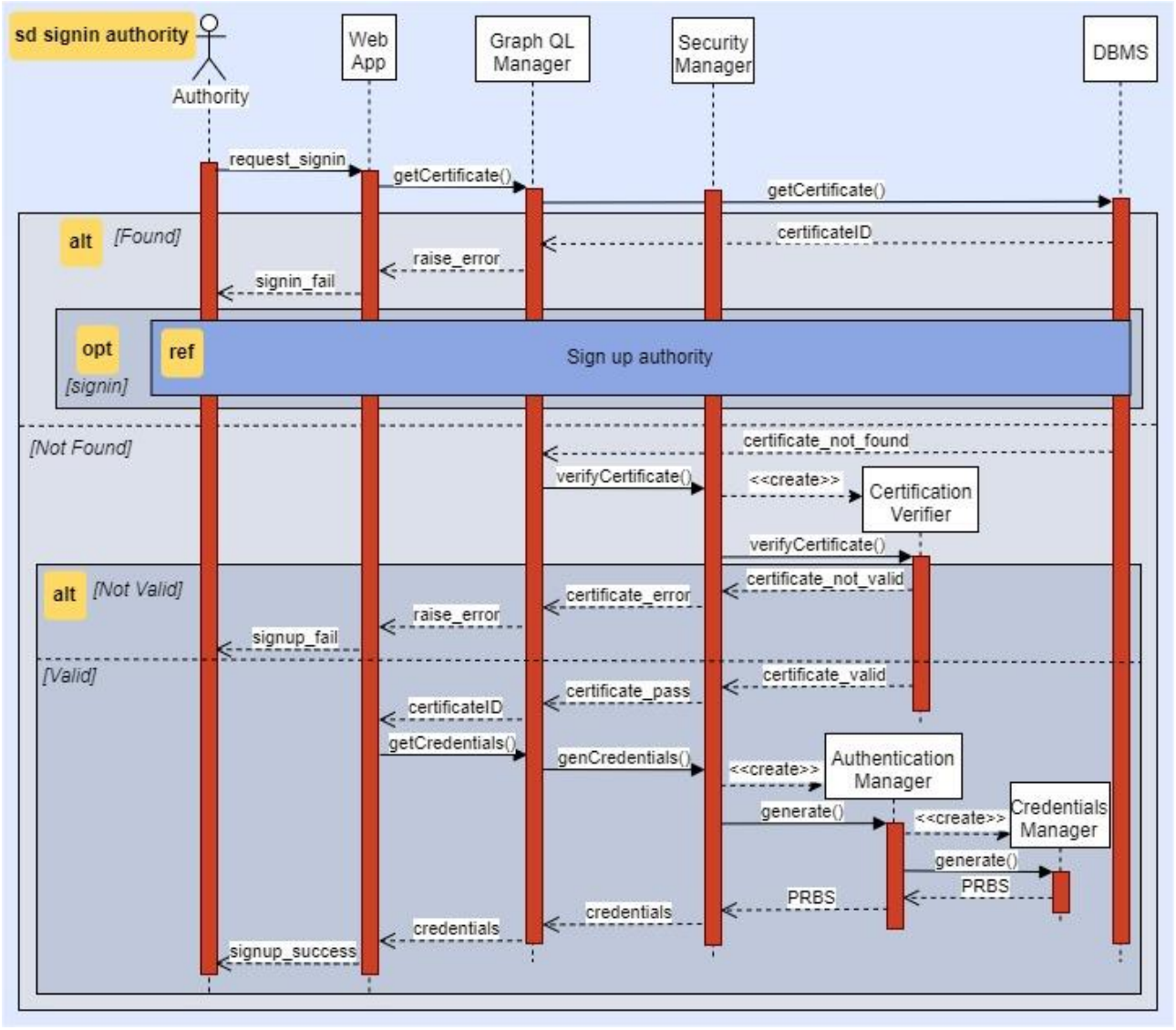



*Figure 6 – sign in authority sequence diagram*

### 2.4.3 Login

The following sequence diagram represent the usual login process as any other application. When a generic user (normal user or authority) requests a login, such user must have been signed up/in already to the application as shown in the previous sequence diagrams. The users must provide their credentials: username or e-mail and a password in order to login. If it's the first time for the user on the system, they can't login, so the mobile application provides the possibility of signing up or logging into the system with valid credentials.

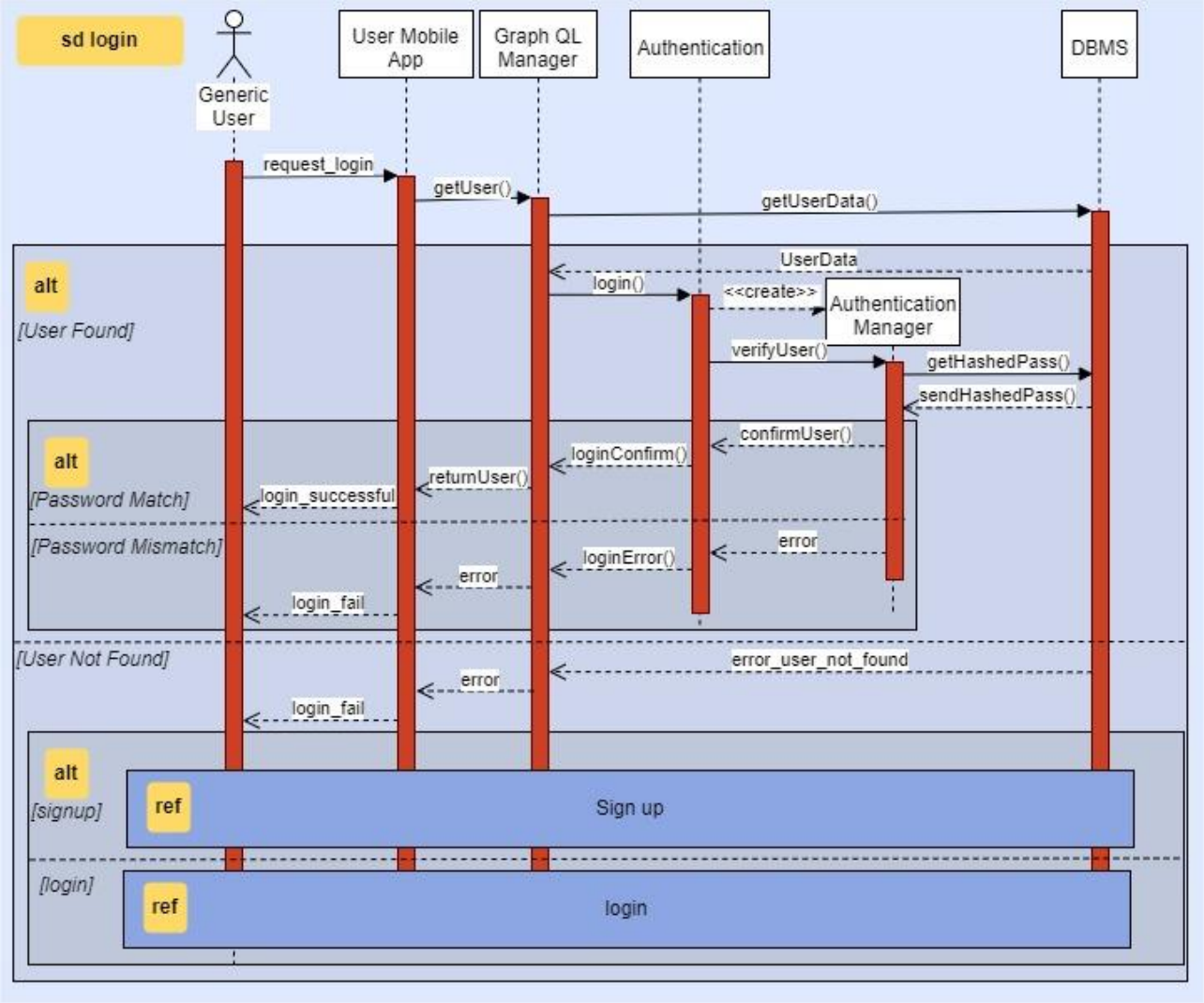



*Figure 7 – generic user login sequence diagram*

### 2.4.4 Map Visualizer

The following sequence diagram shows how the mobile application interacts with the application server in order to provide a map API with various information the safeness level of the requested area, the statistics of how frequent a violation type is committed. Initially, the mobile application must recognize the user type (normal user or authority) in order to provide a certain granularity level of visibility of the requested data. For example, an authority can show information about the license owner through a DMV provided external interface in order to access private information such as full name, address and fiscal code that could be helpful to address fines as a legal reaction for the commitment of a violation. In order to provide a result in a short period of time optimizing the threads between the various tasks, two main processes must be executed in parallel. The first process will be getting the needed information on different granularities based on the user type; the authority can visualize more information, as it's been explained earlier, than the normal user; but every time the authority requests to visualize more information about the committed violations the application server controls if the provided certificate is still valid. If the request is made by the authority a certificate control must be applied because the application provides sensitive information. Eventually, the second process calculates the statistics which are implemented and chosen, based on the existent data in the DB.

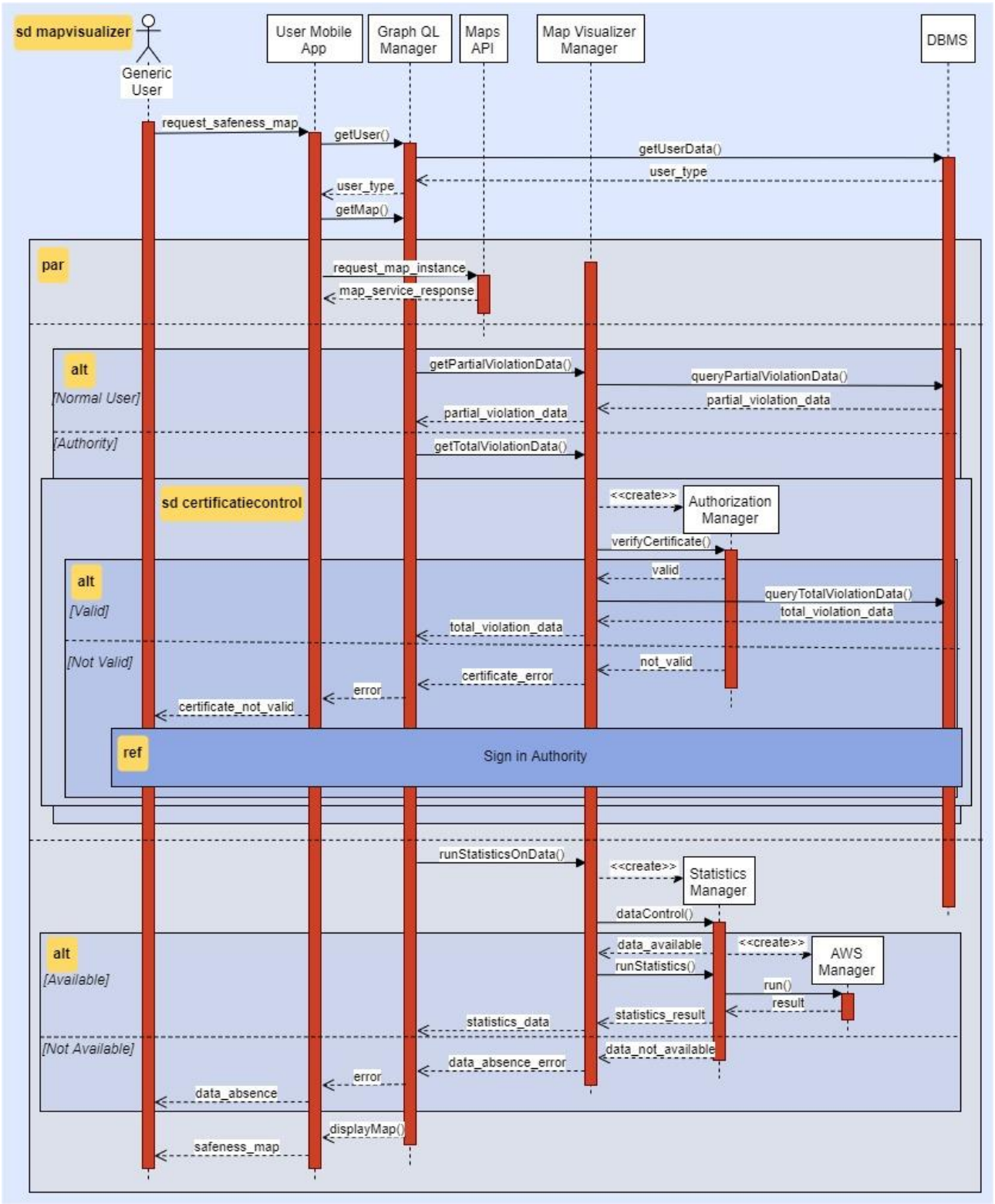



*Figure 8 – map visualization sequence diagram*

### 2.4.5 Violation Report and Notification

The following diagram shows mainly, in addition to the process, the required external interfaces which point out the controlling process of the validity of each report and the main component of the notification process. The violation report request contains main elements to the controlling process such as the picture of the violation and the correctness measure of the license number. When a generic user (normal user or authority) provides a violation report, the application server initiates a controlling process of the validity of the provided report. The first control of the controlling sequence of processes is that the taken picture must contain a vehicle. This kind if control is processed by AWS subsystem called CNN which initiates image analysis process recognizing the elements of the provided picture. If the provided picture doesn't contain a vehicle the application server doesn't go further until the user either provides another picture or cancels the whole report request. Otherwise, the application server proceeds with the second control which extracts the license plate from the picture (if there any) and analyse it to extract the textual part which represents the actual license number then it controls if it's a valid license number. This type analysis is done by another subsystem of AWS which is called OCR. The recognized license number is inserted automatically into the report and it's modifiable by the user; if the license number is modified by the user, such information must be included in the report in order to show minor credibility of such report. At last, if the report passes all such controls it will be notified to the nearest authority based on the position of the committed violation since the GPS is automatically calculated by the application server and it's inserted automatically into the report. The notification process is pushed to the authority through an external interface which Google firebase. In this sequence diagram, the notification process follows the same flow independently from the notification type which may has two different contexts: either violations or suggestions.

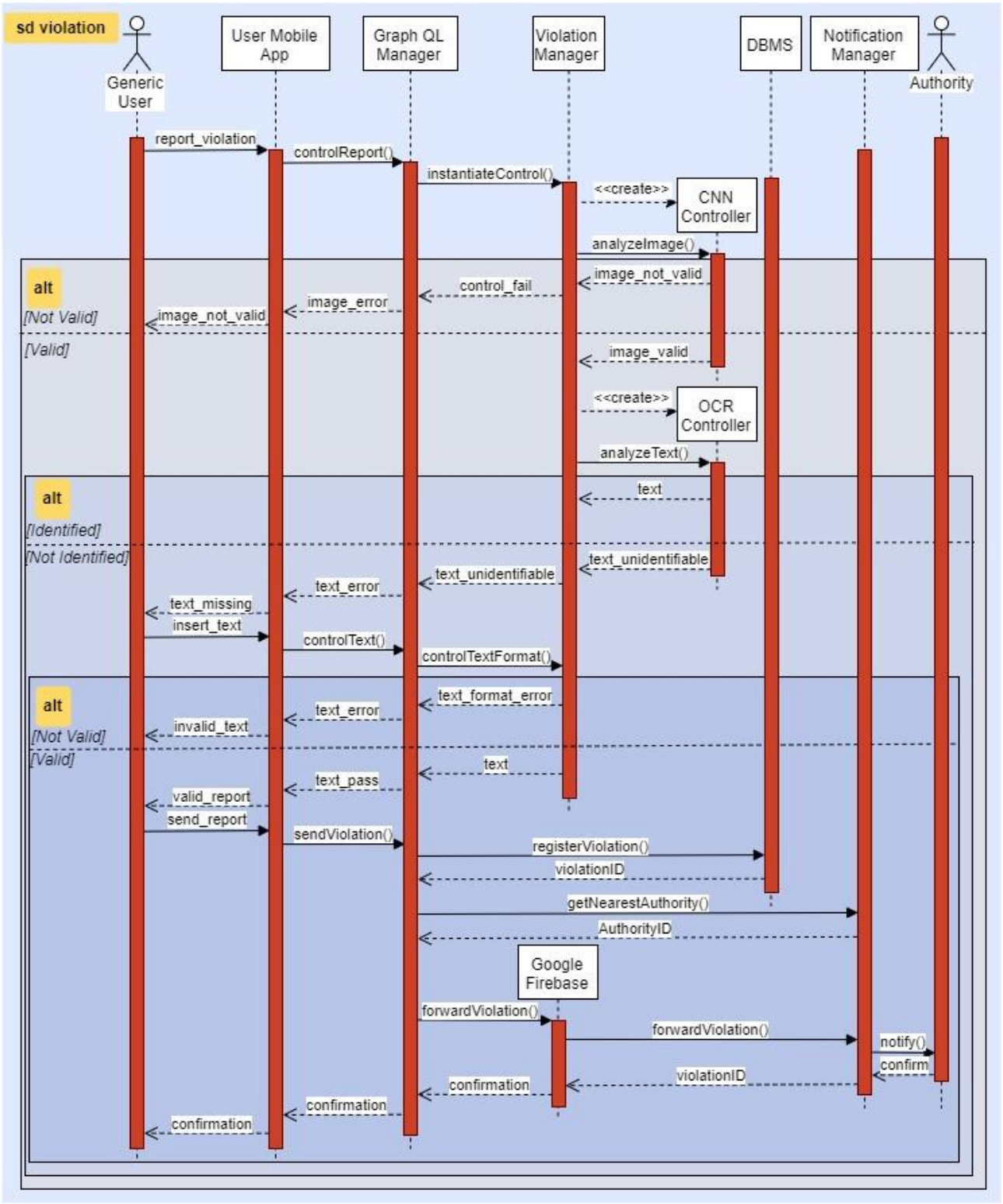



*Figure 9 – violation report and notification sequence diagram*

### 2.4.6 Notification Authority Access and Verification

The following diagram is the logical continuance of the previous one. When an authority requests access to a notification about a violation report, the application server automatically verifies if the vehicle with such license number is stolen through an external interface, license verifier, which the police state license plate verifier. If so, it returns immediately to the authority with report accompanied by a signal which indicates that the reported vehicle is stolen. Then, a certification control (validity check, as shown previously) is made between the application server and the authority in order to provide further information about the vehicle owner through the license number via an external interface which is connected to a license engine that retrieves information directly from the DMV DB. Such information may include full name, address and fiscal code of the vehicle owner in order to eventually address a fine. Finally, if the report is precise, SafeStreets gives the possibility to the authority to verify a report to increase the credibility of the user who reported the committed violation. If the notification type context was about suggestions the flow of this sequence diagram doesn't change much; this fact will be shown more clearly in the next sequence diagram.

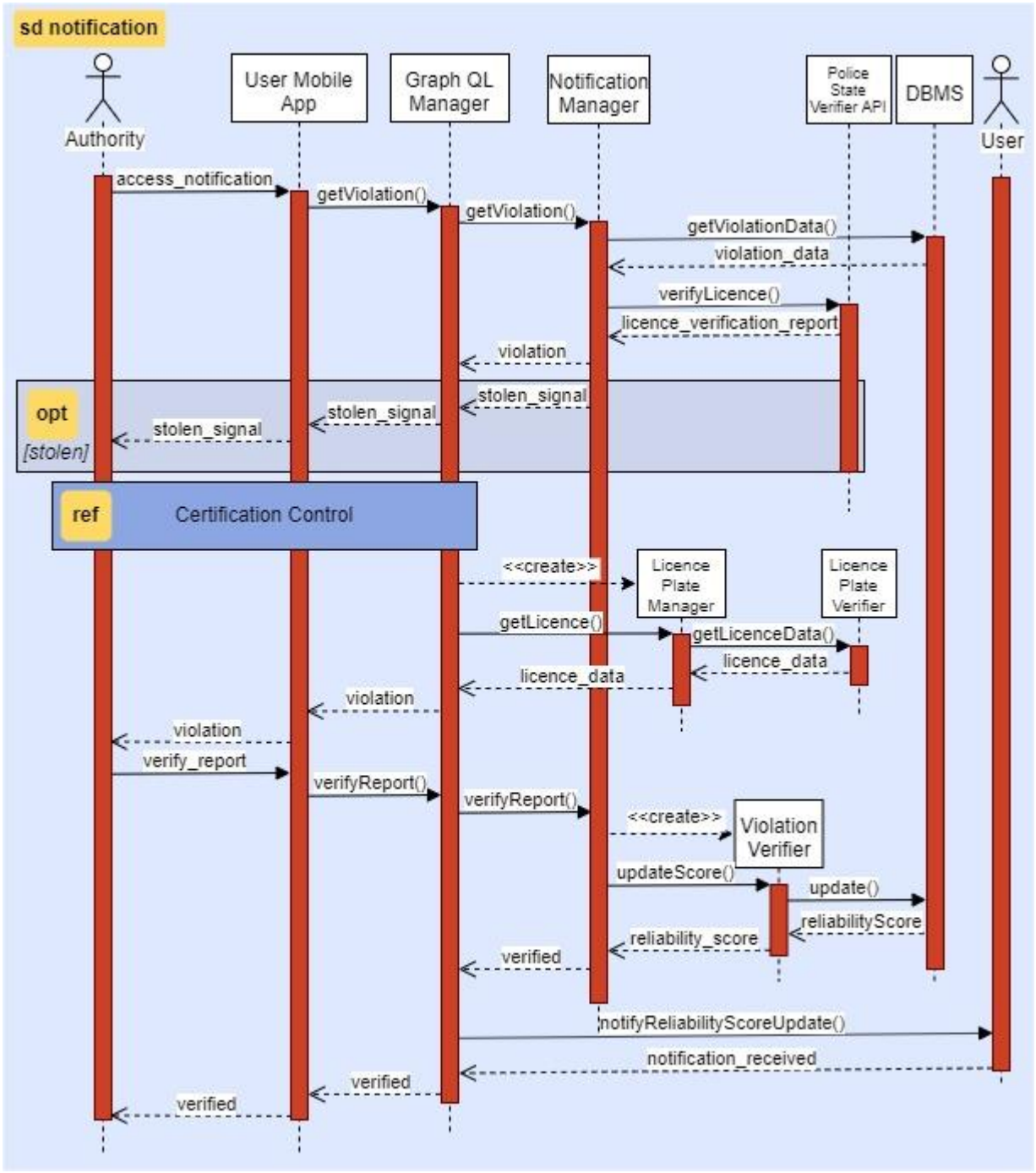



*Figure 10 – notification authority access & verification sequence diagram*

### 2.4.7 Suggestions Authority Access

Accessing suggestions is evaluated in two different contexts which is the period notification about suggestions or accessing the current suggestions provided by the system at any time. This functionality is exclusively for the authorities, so a certification control is necessary, as shown earlier. If the first context is evaluated, the application server calculates the suggestions and notifies them to the authorities, otherwise the authority can visualize the current suggestions provided by the system earlier, in the last period in which the application server calculated the suggestions to be notified to authorities. In this flow, there are mainly two assumptions to be made necessarily in order to cover all possible exceptions. The first assumption points out the fact that there are data provided by the municipality. The second assumption is about SafeStreets DB which must contain data in order to compute the request. The application server calculates the suggestions in various steps. The first one is creating a set of violations of municipality's area. The second step is preparing a set of data about incidents provided by the municipality and adapting its format in order to be compatible with SafeStreets data about violations. The third step is to use suggestions inferring engine in order to run data crossing between violations' set and incidents' set on AWS. The fourth and last step is to create a set of suggestions based on the result of the data crossing computed at the third step.

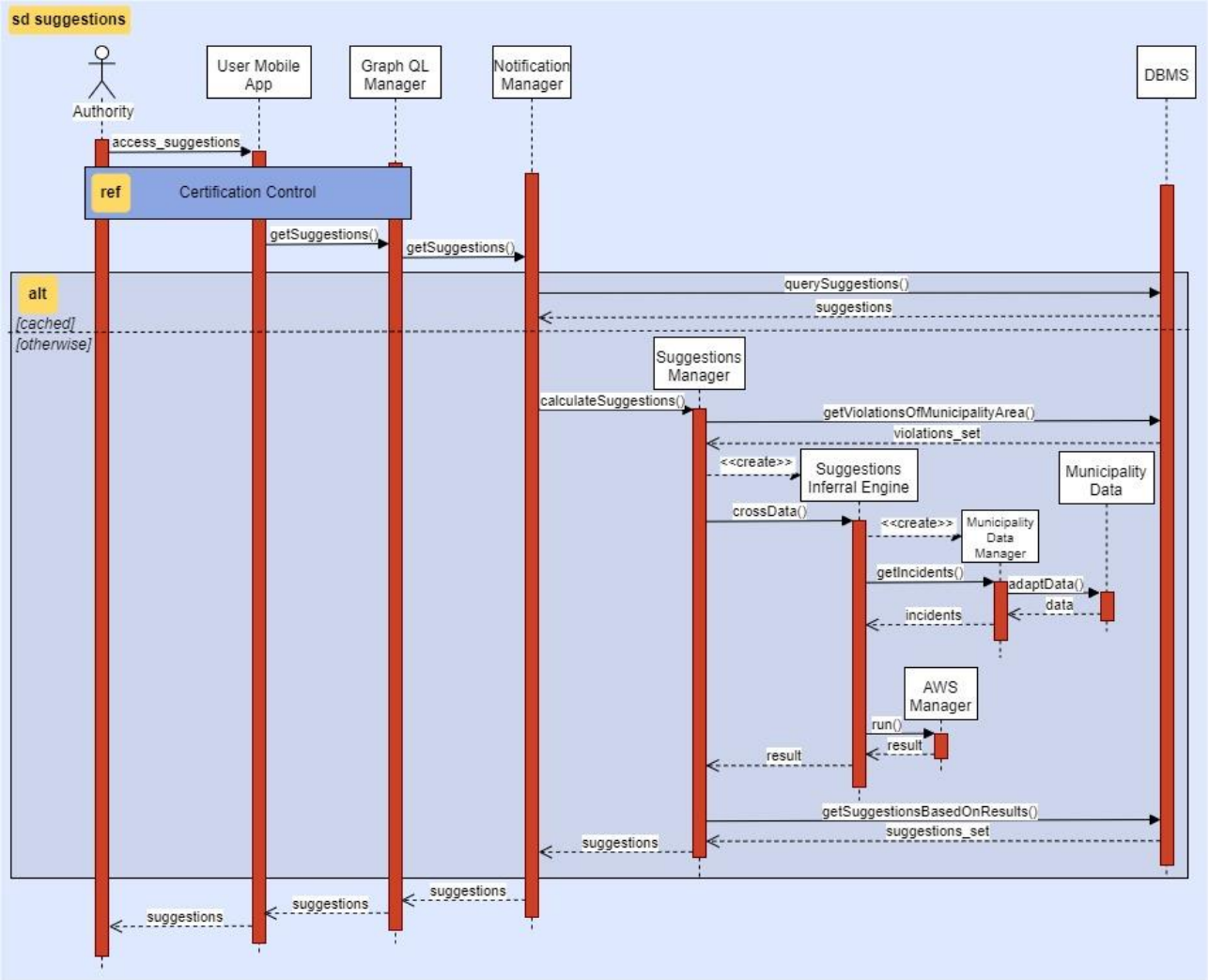
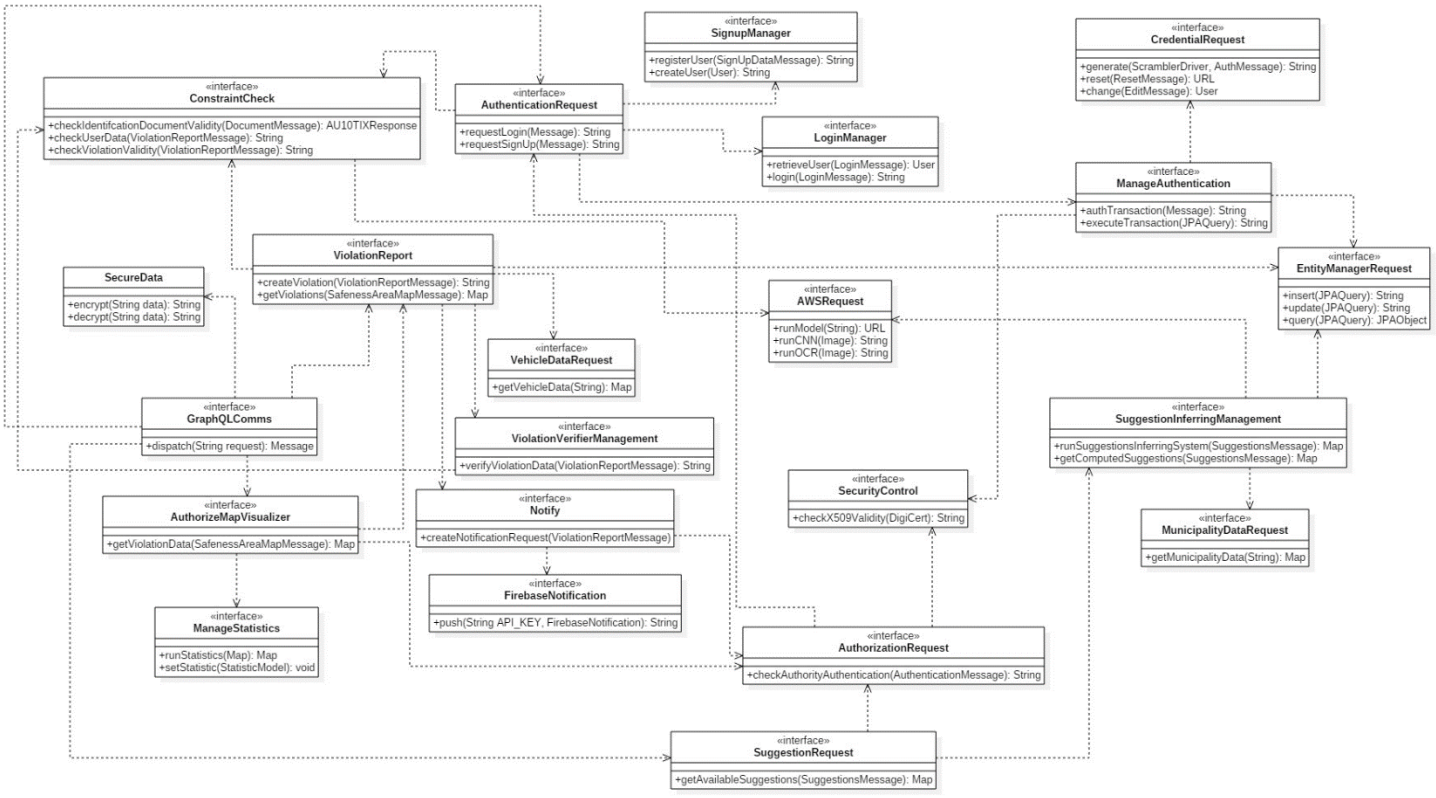



*Figure 7 – suggestions authority access sequence diagram*

## 2.5 Component interfaces

The component interfaces are here presented but to ensure a better and vaster understandability it was chosen to show how the interaction between every component is done through the next Class Diagram. Other disclaimer are also that it was not chosen, again for clarity to not show in the component interfaces the ones of the client and the ones related with the remote procedure calls which are left for a better explanation in the following Class Diagram.




In the above diagram it is shown the interaction between various interfaces. The interaction between the interfaces can be summarized in the various functionalities present in SafeStreets.

- During authentication of a user, authority or normal user, an incoming message will be delivered to the GraphQLComms where through a later described dispatching system will deliver the request firstly to the AuthenticationRequest on which it will have to call, depending on the interface, a login or a registration procedure. They will then unleash a chain of requests to the LoginManager and to the ManageAuthentication which will oversee communicating through the RDMI to the EntityManager to manage JPA objects mapped on the Oracle DB. The login manager will then allow the system actual login this user. The same procedure upon login will follow for the authorities but within addition the control on his provided digital certificate.
- During violation report, an incoming message will be delivered from GraphQLComms to ViolationReport which will be using various interfaces such as the ViolationVerifierManagement, the ConstraintCheck, the EntityManagerRequest and the Notify to allow the starting of the procedure to let the authorities know a new violation has occurred.

- During the visualization of the Safeness Area Map, this precise request will be delivered through GraphQLComms to AuthorizeMapVisualizer. It will depend also on the ManageStatistics interface to also include them in the response message. To guarantee the delivery of a more detailed report it will use the AuthorizationRequest on which, depending on the user requesting this data, will allow a more detailed response or not. Then to gather those data will use the ViolationReport interface.
- During the request or notification of new suggestion it will used the chain of interfaces regarding suggestions which are the SuggestionRequest, SuggestionInferringManagement and the gather of incidents data through the MunicipalityDataRequest. The actual execution of the request will be done through the AWSRequest interface which will be used to infer new data if possible.

The AWSRequest is also used to run OCR and CNN regarding the reporting of a new violation, on the notification side will be used the Notify interface which is dependent on the FirabaseNotification interface which will allow the interaction with Google Firebase APIs and send a push notification. A SecureData interface is also present to allow the encryption or decryption when needed of the data provided or requested, which are stored in the Oracle DB.

Here the SafeStreets class diagram with almost every detail is shown, almost every detail because further implementation analysis is required. The class diagram is shown firstly in full form to give a sense of the general architecture, although due to being big it was better to shown even some cut and more zoomed parts, but it is not to be intended as a ready to code class diagram, due to the nature of this document it was preferred clarity over precision. It was chosen to add also this diagram to give a better understanding of the project even if the relation between the Component Interfaces to the Class Diagram is not in scale 1:1 for clarity reasons.

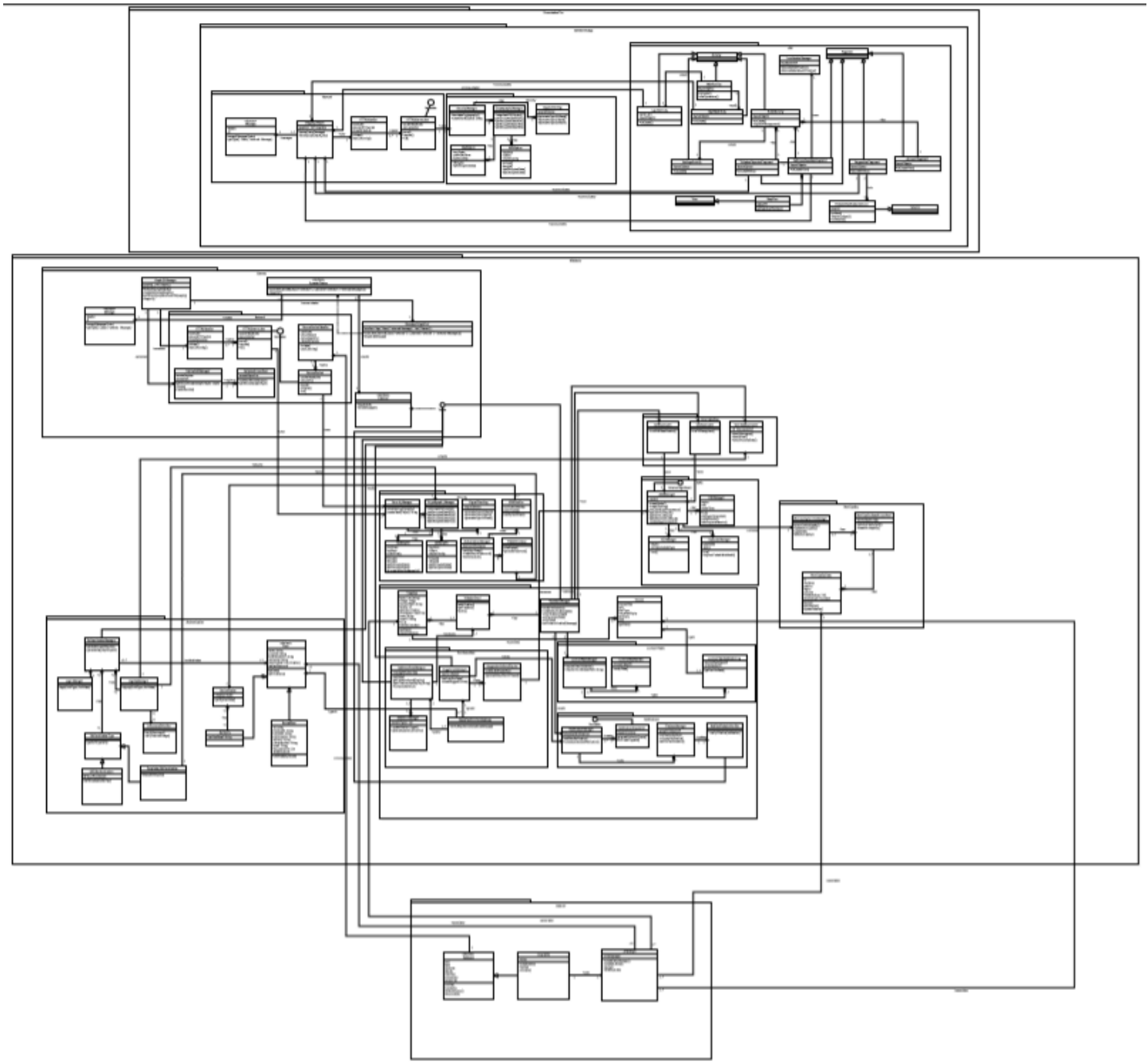


*Figure 5 – Full Class Diagram*

Now, for a better understanding zoomed parts are shown. Starting from the lower level corresponding to the Data tier, in the Figure 6 is shown how the interaction with the distributed ORDBMS, Oracle DB[18], is managed within a cluster. The available API for a better entity-object management are also used which are part of the Enterprise Framework available.

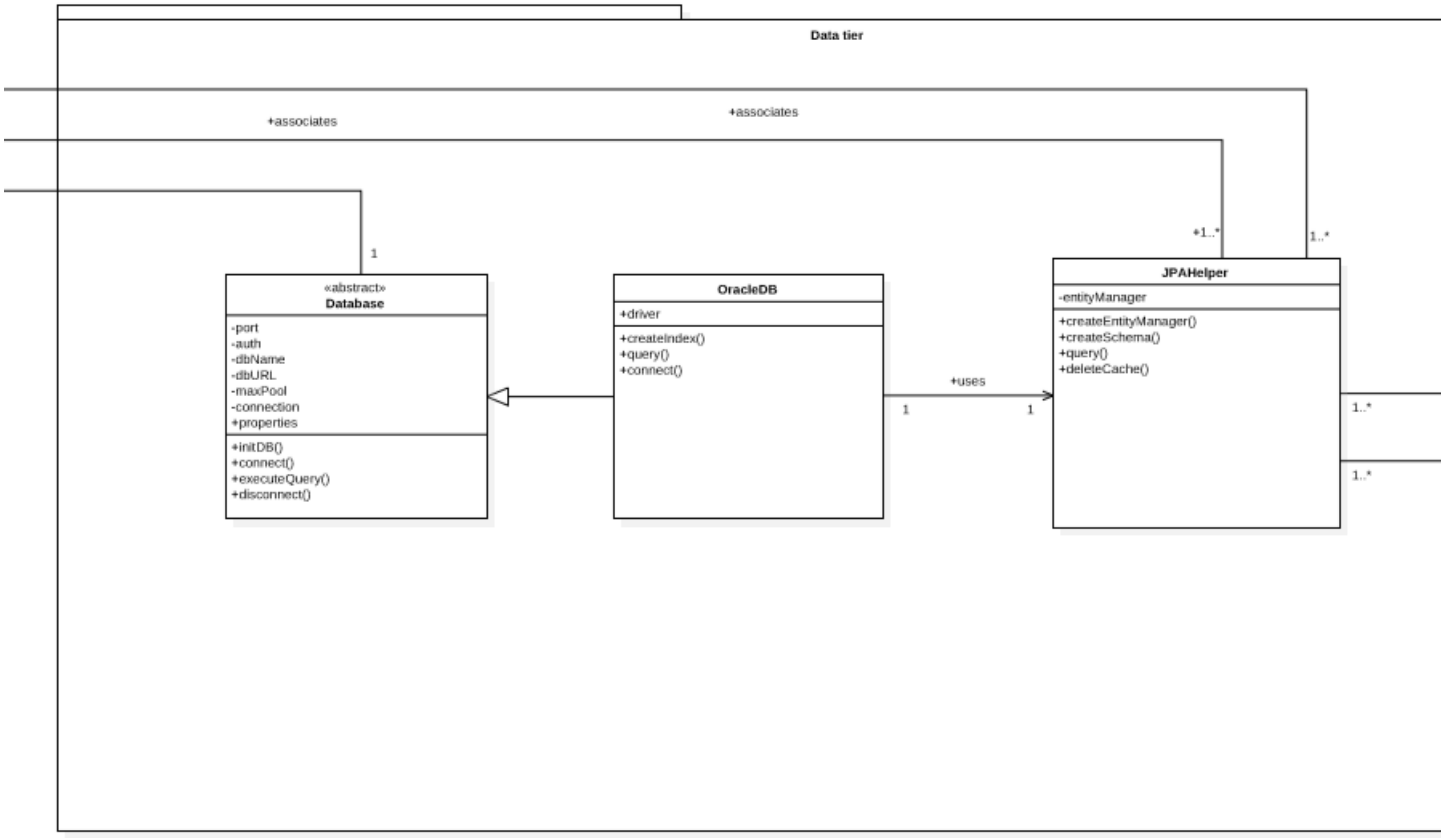



*Figure 6 – Class Diagram of Data tier*

The next part is now focused in the middle tier which due to being big was split in various parts.

In the Figure 7 it is shown the communication and security package. In the communication package are present the various classes for communicating according to the architecture through HTTPS, TLS and other types of secure communication which are using a standard dynamic scheduler for the various connections. Also, the RMI communication is present which is guided by the standard APIs to let, in case of necessity, the inter communication between different cluster to improve the fault tolerance. The logical representation of the communication between users and the system is represented using GraphQL[21] which will be better presented in the following paragraphs.

The visitor pattern in a little more abstract and general way is shown and how indeed communication needs to pass through the implemented DynamicRouter using the MessageDispatcher class.

In the security package the various symmetric and asymmetric encryption algorithm used are present as classes and other security functionality functions regarding digital certificates X509 and hashing are present.

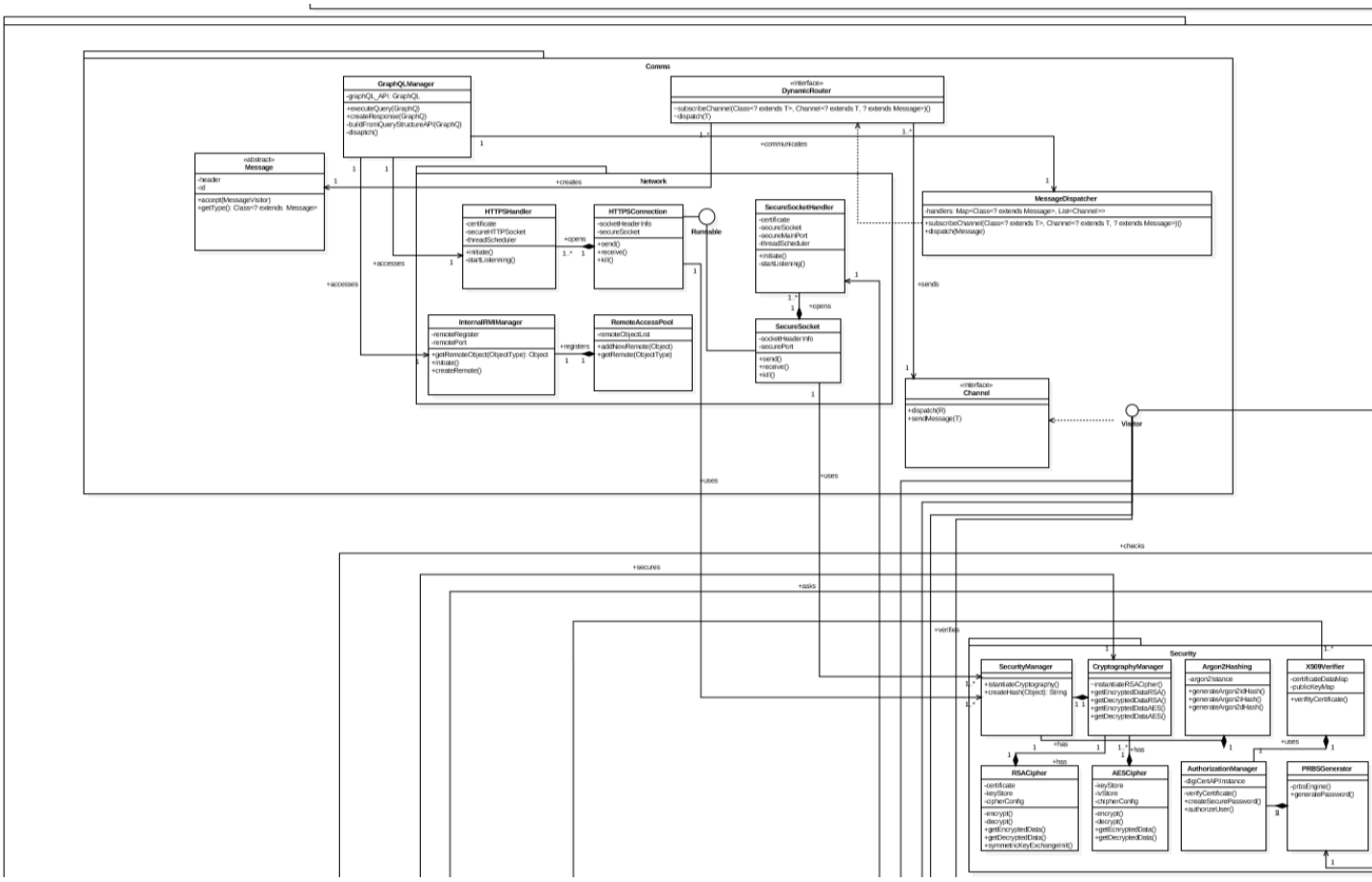



*Figure 7 – Class Diagram of middle tier – part 1*

In the following 8th figure the authentication part of the system and the one regarding the various functionalities are shown, with also the presence of the SMTP server to send confirmation and restore emails. As shown in the RASD[11] the authentication is composed of two types, one proprietary and one through SPID[8]. The user class and its children are present as a form of entity-representing object through the previous shown Enterprise framework API.

The functionalities are accessed through a constantly present layer of security to also grant or not what a user can get in response and which authorities receives a notification automatically regarding some event through the Google Firebase[14] push notifications API.

Both authentication and functionalities access are ruled under the main dynamic visitor pattern.

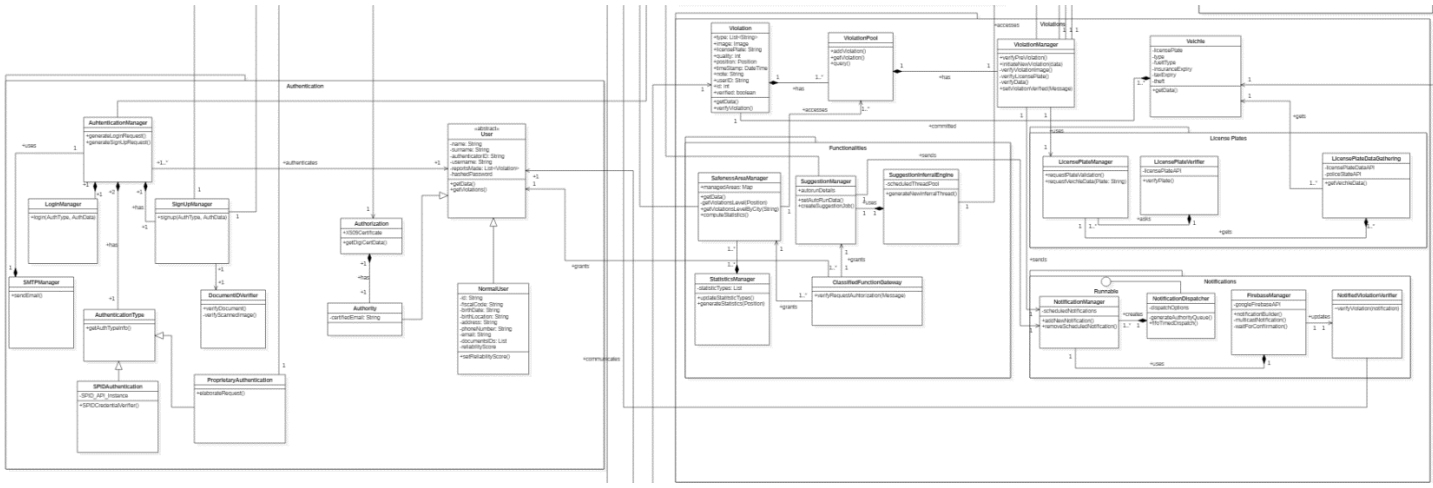


*Figure 8 – Class Diagram of middle tier – part 2*

In the last part of the middle tier shown in the figure 9 it is shown the interaction with the public cloud through AWS[12] and with Municipality access.

AWS SageMaker[12] grants SafeStreets access through the APIs which are then used in the main AWSManager class which is the node through which any kind of AWS requests and responses are managed. Requests can also be autonomous Inferring on new data, a better fine tuning of hyperparameters, an improving of the already presents model and so on. As shown, in case it is asked to verify some data like a picture taken that will be asked to the CNNController which will then start a request to AWS through the AWSManager. The same is shown for the OCRController.

For the municipalities instead it is shown as an architecture a pool of data access for the various municipality which contains various metadata on the municipality itself and then the incidents data queried from their server, which specifications is of course present.

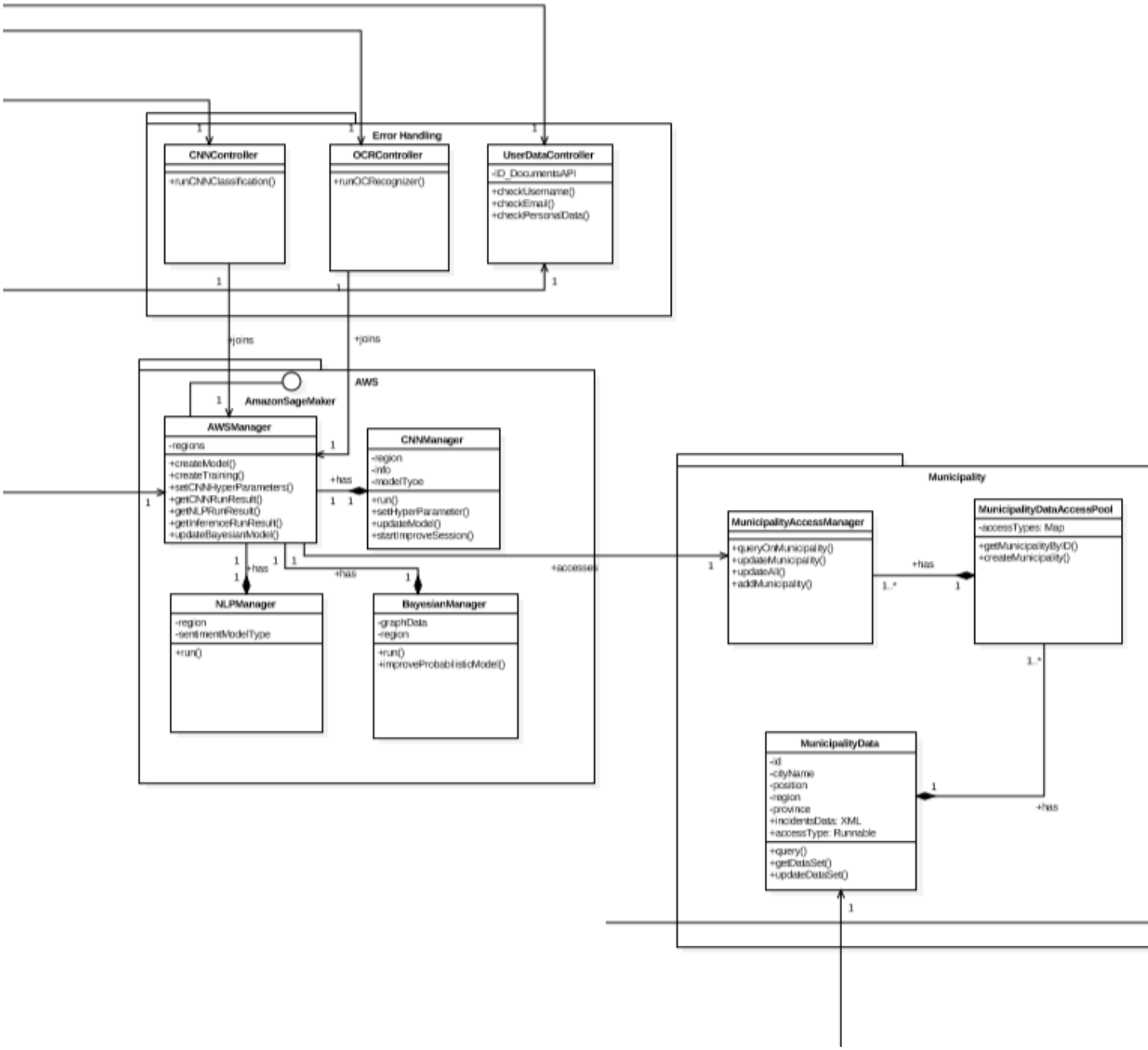
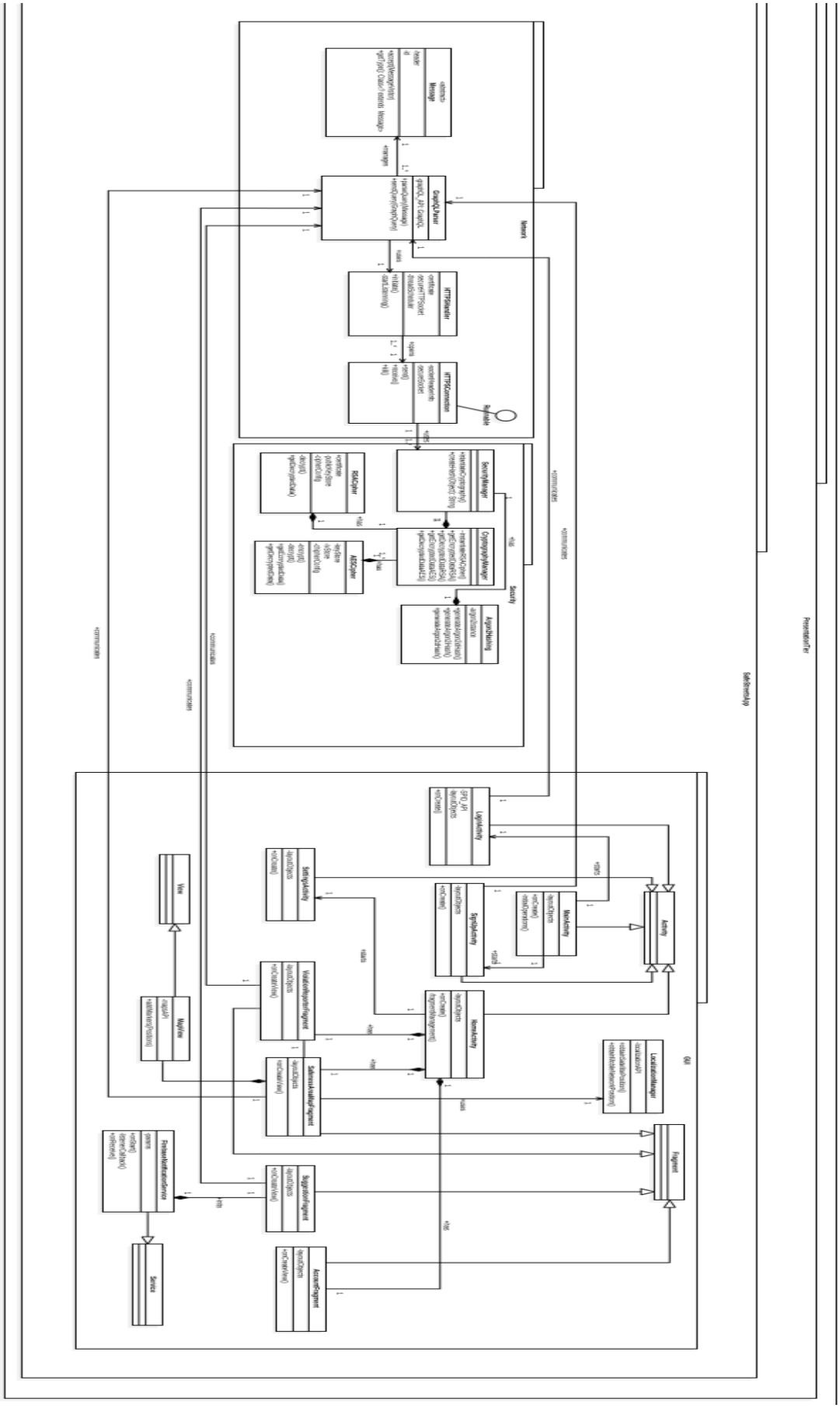



*Figure 9 – Class Diagram of middle tier – part 3*

The last part of the system is leaved to the presentation tier shown in the figure 10 where there is the standard client-side network communication software architecture and then then a package for security and for actual presentation is present. The security package is needed also here to grant an actual end-to-end communication. In the GUI package are present the various Activities, Fragments, services and APIs used in the client side with a basic inner representation which is left for the further implementing phase part.

*Figure 10 – Class Diagram of presentation tier*



## 2.6 Selected architectural styles and patterns

Here are presented the architectural styles and patterns used in designing SafeStreets, their presentation will start from the architectural patterns, then to the architecture styles. That is because there is not a unique pattern and style but a variety of these to allow the design and future development of a better application both on end user level for satisfaction and on the developer level regarding the implementation, the power of the architectures combined and the accordingly easily maintainable and testable project.

As for Architectural Patterns for SafeStreets have been chosen these two main patterns:

- Multitier Architecture
- Dynamic Visitor Messaging Pattern

### *Multitier Architecture*

The Multitier Architecture is intended to allow any of the n-tiers to be upgraded or replaced independently in response to changes in requirements or technology. For example, a change of operating system in the presentation tier would only affect the user interface code because they are physically separated.

The main presentation tier is the topmost level of the application. It is intended to display information related to users request upon various SafeStreets functionalities like Safeness area map, violations report, and statistics on violation in a certain area and so on. It communicates with other tiers and it is a layer which users can access directly using the SafeStreets app, or in case of Authorities using also the web access.

The application tier which corresponds to the distributed middleware running on different clusters in SafeStreets, is the tier that controls an application's functionality by performing detailed processing regarding various functionalities, in this case it will be able to obtain data and communicate with other services and infrastructure as discussed before.

The data tier includes the data persistence mechanisms and the data access layer that encapsulates the persistence mechanisms and exposes the data. In this case the data tier is also distributed to grant even more scalability, flexibility and general performance improvements.

A graphical general and generic representation of the Multitier Architecture is here displayed in the Figure 11.

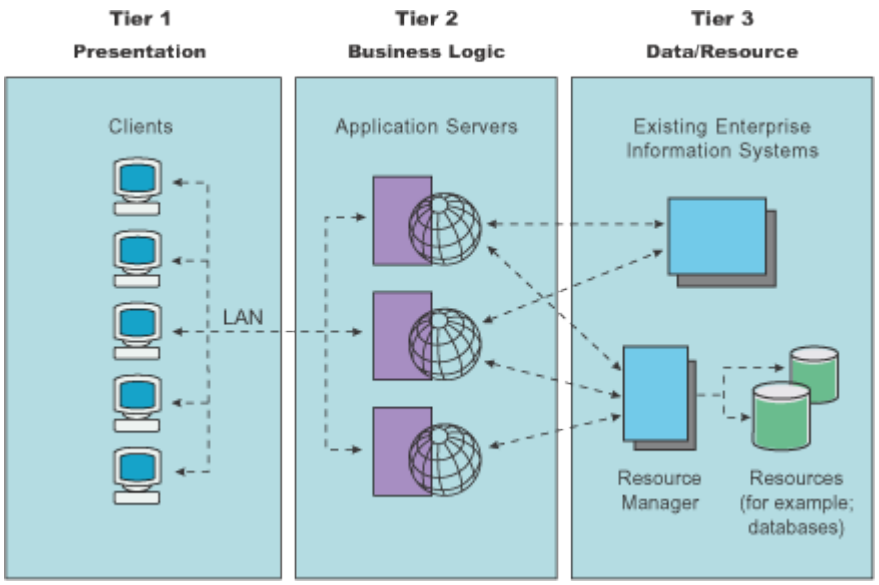



*Figure 11 – General and generic multitier architecture*

### *Visitor Pattern*

The visitor pattern is a way of handling inner communication in the middle tier from ingoing and outgoing messages and requests. This pattern is very powerful indeed it will be able to developers to add new functionalities without changing the main classes of SafeStreets, thanks to its separation between objects and algorithms. This approach will make the project more and more:

- Maintainable
- Testable
- Easier approach to test driven development (see Implementation and testing in chapter 5)

An important advantage is also that the various visitors are easy to separate and assign to different developers during the actual implementation phase.

The visitor pattern can be also better understood by looking at the Figure 12.

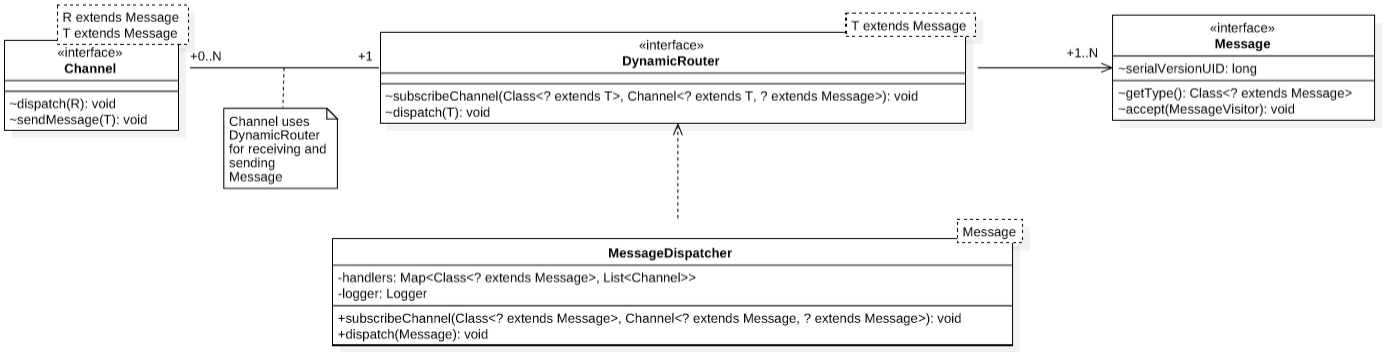



*Figure 12 – General and generic Dynamic Visitor Pattern*

As for Architectural Styles for SafeStreets have been chosen these three main styles:

- Web app - service oriented architecture style
- Public cloud computing architecture style
- Representational state style using GraphQL

### *Web app - Service oriented*

The Service oriented architecture style (SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. The basic principles of service-oriented architecture are independent of vendors, products and technologies. A service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently.

A service has four properties according to one of many definitions of SOA:

- It logically represents a business activity with a specified outcome.
- It is self-contained.
- It is a black box for its consumers.

- It may consist of other underlying services.

Service-orientation promotes loose coupling between services which is what we want in general for SafeStreets and what will be having especially in the SafeStreets web access for Authorities. SOA will separate various SafeStreets functionalities for Authorities into distinct services for example the service for checking the various kind of violations in multiple areas or the service for the suggestion inferred through municipalities data which can be dynamic over the time. This choice of using a web app for the authorities accessing SafeStreets it is a better approach because it will grant to deploy to different kinds of authorities the same access and will help do decouple the deployment of the various services for the authorities. The service-oriented architecture is well suitable in the SafeStreets architecture thanks to its large scalability, flexibility and performance characteristics.

### *Public Cloud computing*

The second style adopted is the public cloud computing architectural style. It will have a direct link with SafeStreets given the fact that is used to check for image validity, license plate reading and suggestions computation with municipality data. It will be based on AWS[12] on which there will deployed various SageMaker instances:

- Convolutional Neural Networks: it will be used to recognize if an image is valid, so if it contains a vehicle and its identifiable license plate using a pretrained model with millions of tested images and having a 99% success ratio. Each instance will be improved upon any new image upload in addition to the Transfer Learning already in use. The chosen instance would be the ml.p3.16xlarge.
- Natural Language Processing: it will be used to add more context and to aggregate in a better way incident data with their description which municipalities give SafeStreets access to. It will create so a model tree of an incident when possible to improve the suggestions capabilities and to do so it will be used the BlazingText Algorithm in AWS.
- OCR: it will be used to read autonomously the license plate using text recognizing algorithms based on continuous learning and improvement.
- Bayesian Networks: it will be used to create suggestions, it's based on a conditional probabilistic model represented as a probability graph where it is possible to get the more probable event following a certain event conditioning it. It will be allocated in the Machine Learning section of AWS SageMaker[12].

AWS public cloud is composed of an elastic load balancing which will automatically instances new deployable instances for SafeStreets functionalities depending on the requests. SageMaker will be configured in using the best hardware available for Depp Learning which depending on the actual economical availability can be the NVIDIA DGX-2 or some NVIDIA Tesla V100 GPUs.

*Representational state pattern using GraphQL*

As for the third style, Representational state pattern using GraphQL[21] it was chosen this composite style because in SafeStreets there will be a huge number of users that will access its functionalities and on which no server will have to save any kind of client state, which also does not have any control over. Although a REST standard query style has the disadvantages to be lower performant than GraphQL, to don't have the possibility querying multiple different resources in an efficient way and to be server dependent on the query result representation. This does not mean that the REST architectural style does not need to be used: it guarantees, through its constraints, various advantages. So, it was chosen to combine it with GraphQL which compensate all those various disadvantages said above like performance, multiple different resource queries, client defined which led to a series of advantages which are useful to the actual SafeStreets system. Given the fact that Authorities and normal users will have different level of authorized access it is effective to be capable to make requests in a specific form defined from the caller of the query and receiving the result in that precise format. But this is not the end: it is also very suitable for relational data because GraphQL will by itself get the right data put together without the need of the developers to do various REST calls to different endpoints. Having also different kind of access methods for Authorities and normal users, a GraphQL approach will allow to simply adapt the specific request for visualizing data on the Service oriented style for Authorities using the SaaS and on the mobile app for both users and authorities and it will be able to better manage of push notification through Firebase[14]. Also, adopting GraphQL will led to better maintainability because there is no need of a common interface for requests.

Having an underlying REST approach will create constraints which are already satisfied:

- Uniform interface: the goal is to have a common approach to access the resources, so that being familiar with one API means being familiar with all the other APIs.
- Client - Server: client and server are two different entities, which evolve separately without any dependency.
- Stateless: the client is responsible for managing the state of the application and this entails a simpler server design.
- Cacheable: this allows to avoid some interactions between the client and the server, speeding up the communication.
- Layered system: A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. This means that the client doesn't know if it's talking with an intermediate or the actual server. So, if a proxy or load balancer is placed between the client and server, it wouldn't affect their communications and there wouldn't be necessities to update the client or server code.

All the communication in question exploit the HTTPS protocol through POST requests, HTTP over SSL, in order to guarantee the security and the reliability of the connection in any kind of situation. Regarding the format in which the data are transmitted, JSON is used because it is suitable for the data interchange between client and server and GraphQL services typically respond using JSON. JSON may seem like an odd choice for an API layer promising better network performance, however because it is mostly text, it

compresses exceptionally well with GZIP. A graphical simple explanation of its working process is shown in the below Figure 13.

The utilization of this approach will ensure high future improvements, maintainability, and flexibility and will ensure a lower testing effort required to reach the desired quality. This is possible thanks to the improved of the already high decouple between server and client implementation and architecture. Any modeling change on the server side will not effect in any way the various client requests nor their security.

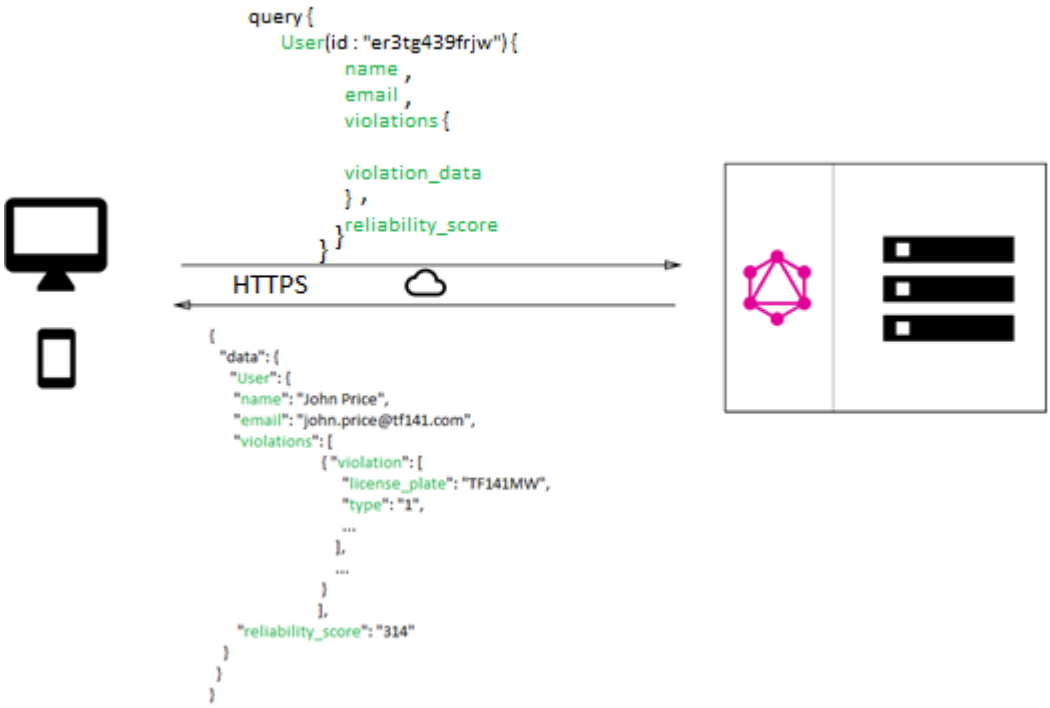



*Figure 13 – GraphQL, one of the advantages displayed*

## 2.7 Other design decisions

There are also other design decisions which were briefly shown in the overview of this chapter:

- Distributed DBMS using Oracle DB
- Thin client
- Cloudflare
- Firebase for push notifications
- End-to-end encrypted communication
- End-to-end encryption for sensitive data
- Network architecture overview

### *Distributed RDBMS*

The distributed RDBS is actually a pretty good choice for SafeStreets. Having already a distributed middle tier it was quite natural to couple it with a distributed RDBMS because it will allow to gain performance on data retrieval and their actual availability. A relational database was chosen given the data structures presents in SafeStreets which needs to be related with one another, for example a violation will be obviously related to the user who made it, to a vehicle, to certain city and so on. For the RDBMS it was chosen Oracle DB[18] EE that offers one of the most performant RDBMS available, supporting ACID properties and a variety of SQL functionalities like triggers and data integrity upon table creation and indexes.

In the following Figure 6 is displayed the ER diagram for the Oracle DB[18] EE Distributed RDBMS.

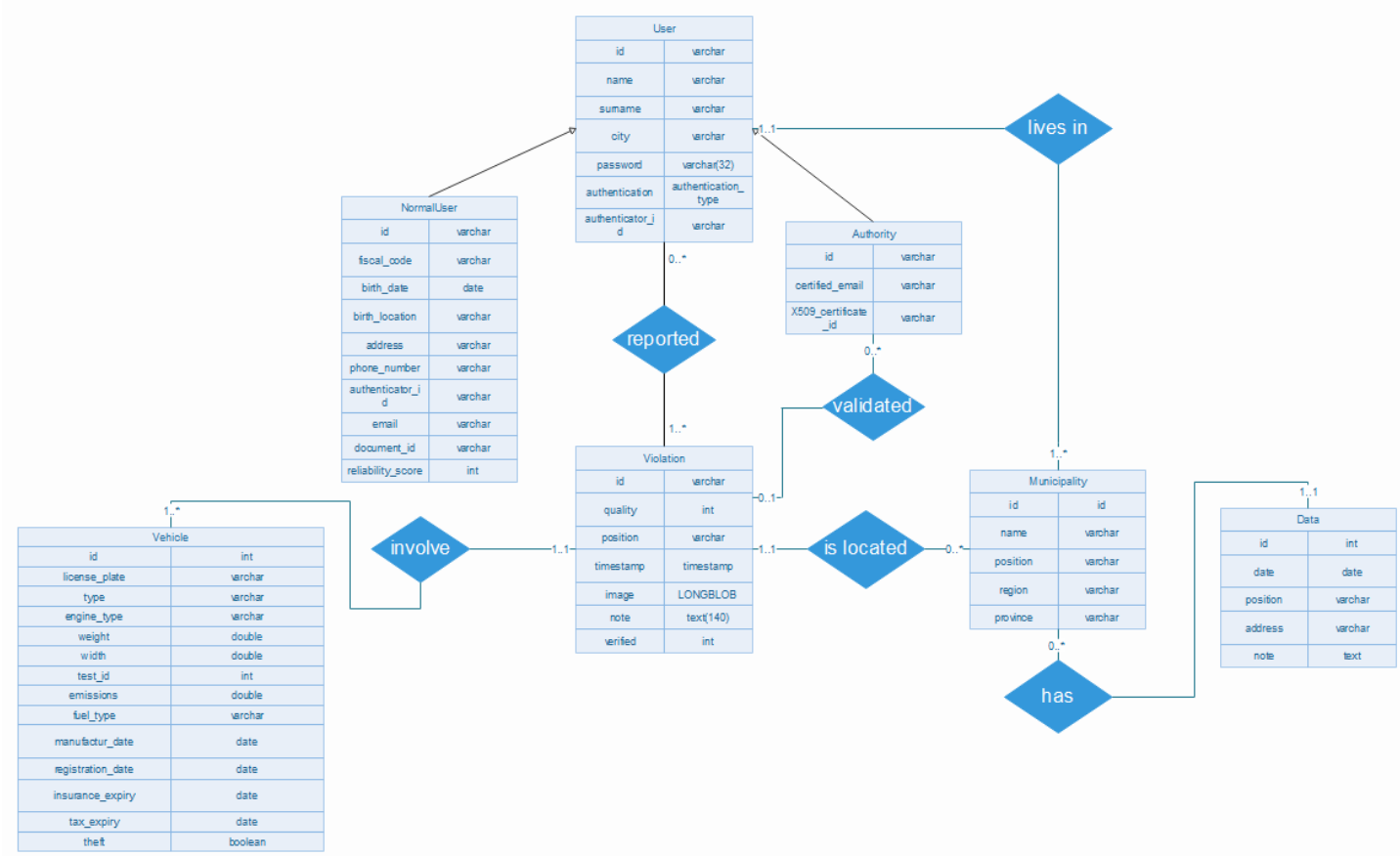



*Figure 14 – ER diagram*

As shown in the figure 14 there are multiple entities with various kind of relationship and respective cardinalities. There are some peculiar points which merit to be explained. The password field is limited to 32 characters because as will be stated below in the

following sections, only its hash will be saved for security and privacy reasons, accordingly to the Legislative Decree 196/03[1], Legislative Decree 82/05[2] and the regulation 2016/679[3], and the hash function that will be used has as outputs hashes of length of 256 bits which is translated to 32 chars. The image data will be saved as LONGBLOB type (Long Binary Large Object) to ensure that any image with any kind of quality or compression will be accordingly saved. The note attribute is limited as discussed in the requirements for clarity reasons regarding user explanation to exactly 140 characters. Vehicle entity has lots of data thanks to the license plate APIs[9] and the Police State APIs[10], those data will be made visible to authorities and in a limited part to normal users. The authority will be as stated in the previous document capable of reporting a violation as a normal user can but also to verify violations which were notified to him. The verification process will output an enumeration which is represented as an integer in the attribute so that's possible to state if no verification has been done or if there was a successful verification or a failed verification which will then change the reliability score for the reporting user through the internal business logic partially implemented as Oracle SQL triggers.

### *Thin client*

The thin client decision was straightforward. In SafeStreets all main computation is made on the middle tier so however a user is accessing SafeStreets, mobile app or SaaS, they won't need to do any kind of heavy workload but the actual presentation of information and its rendering. However, some needed tasks are required in order to guarantee the desired high degree of security, like for example encryption/decryption especially in the end-to-end encryption for sensitive data explained below.

The client application, differently from the web app the authorities can access, will take advantage of the framework available for the local environment in which is being executed and it will use it to guarantee it can correctly communicate with the various servers and to be sure it will be able to do the few important operations requested.

### *Cloudflare*

Cloudflare[13] was adopted to improve principally the web servers security but also to add more services like DDoS protection, a Web application firewall, an Authoritative DNS and a Content delivery network which are not so easy to add natively in a complex architecture.

- DDoS protection: Cloudflare offers an "I'm Under Attack" mode for customers experiencing cyberattacks. It will be needed in case SafeStreets were under attack from malicious users making millions of requests very rapidly to try to disrupt SafeStreets services and other user experiencing their SafeStreets access.
- Web application firewall: Cloudflare will allow to utilize a web application firewall service. By default, the firewall has the OWASP ModSecurity Core Rule Set alongside Cloudflare's own ruleset and rulesets for popular web applications which will mitigate any non-compliant incoming packet.

- Authoritative DNS: Cloudflare will also offer a faster user accessing SafeStreets services by using their DNS which has one of the fastest DNS lookup speeds worldwide, with a reported lookup speed of 5.6ms.
- Content delivery network: Given that Cloudflare's network has the highest number of connections to Internet exchange points of any network worldwide it is possible to exploit Cloudflare cache content in its edge locations to use it as a content delivery network (CDN); all web requests from Authorities through the web access SaaS are then reverse proxied through Cloudflare with cached content served directly from Cloudflare when needed. So, in case of web server malfunction, which is very rare given their highly scalability characteristics and fault tolerance, no web access for the Authorities will be compromised.

### *Google Firebase*

Google firebase[14] is instead used for push notifications, push notifications are needed for the authorities to know whenever a new violation is reported, or a new suggestion is available. Push notifications are managed in the middleware but then it will be Firebase to send them to the right chosen users by the servers. A user will retrieve push notifications using even here, as described before, GraphQL[21]. Developers will use Firebase available API, and this will also lower the testing and implementation cost together with simple yet powerful management.

The utilization of this cloud service will ensure high future improvements, maintainability and will ensure a lower testing effort required to reach the desired quality.

### *End-to-end encrypted communication*

Every communication will be encrypted as said before, in order to do that it will be chosen an encryption based on asymmetric and symmetric keys. RSA[15] with 4096 bits keys will be used alongside with AES 256[15], having in SafeStreets safely stored a KeyStore encrypted with a user dependent key, which contains the private key used together with the public key of SafeStreets to exchange a symmetric key for AES 256, gaining better performances and to allow secure communication on any network. This will make SafeStreets compliant with the Legislative Decree 196/03[1], Legislative Decree 82/05[2] and the regulation 2016/679[3].

### *End-to-end encryption for sensitive data*

For sensitive data, so for the user personal data upon registration there will be the need of a further security layer. To do so it will be used end to end encryption, where not even SafeStreets will be able to read user personal and private data. To accomplish this will be used a symmetric encryption algorithm, which its key will be based on the actual password

of the user salted with a function which has as domain a set of characters that are the unique identifier of a user. This end to end encryption works because the key is not known even to SafeStreets since in the database the password will be saved with its hash, using as hash function Argon2id[17] which is the most powerful and resistant over attacks hash available. So, any user personal data is encrypted in the database using this procedure and sent over the network as it is, allowing over the actual owner to decrypt it as requested the Legislative Decree 196/03[1], Legislative Decree 82/05[2] and the regulation 2016/679[3].

### *Network architecture overview*

In the following image is shown the already discussed architecture but shown in a more network fashion way so to be helpful in the future implementation at the network level for understating better the clustering and the configuration needed for the hardware network components such as the firewall, the load balancer and the OpenFlow switch defining an SDN. OpenFlow switch will be so used for a better performance at a single cluster level, which is configured using the OpenFlow protocol using the API provided using Python OpenFlow APIs and then configuring via script deployed configurations the various OpenFlow switch which needs the knowledge of the inner cluster topological network schema to improve performances.

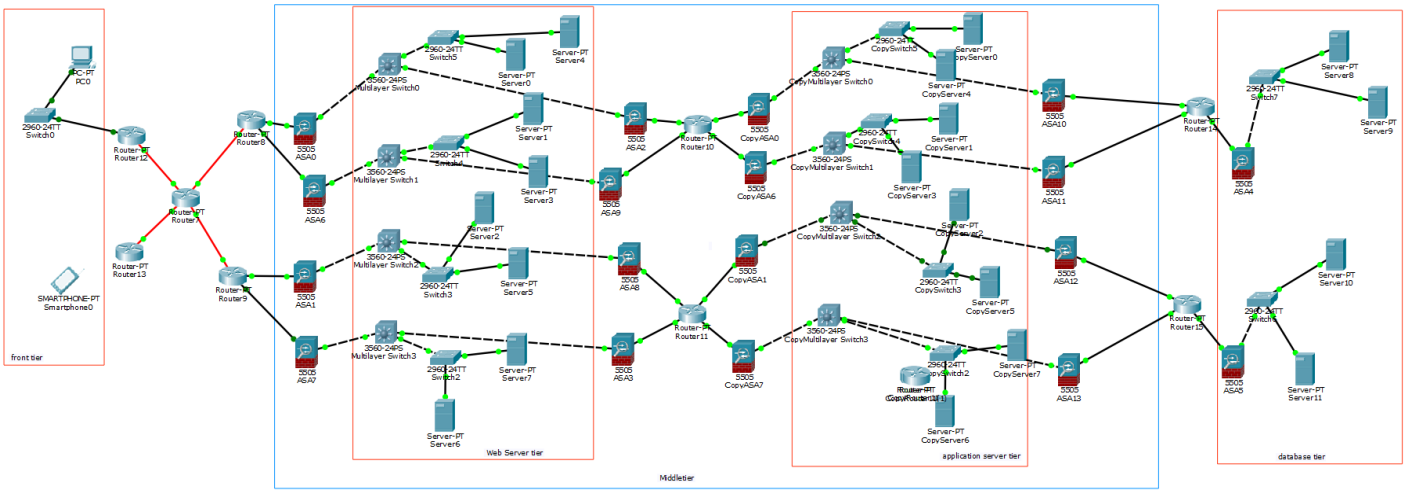



*Figure 15 – Network architecture diagram overview*

# 3 User interface design

The following design of the smartphone app of SafeStreets for both users and authorities and the design for the desktop application exclusively available for authority are here shown. They are just an intuitive way to lead the design towards this direction but any necessary changes or adjustments or new components can be added to ensure a better user experience and improve both usability and simplicity. UX diagrams will follow to understand user interaction with SafeStreets app and the authorities interaction even with the web app.

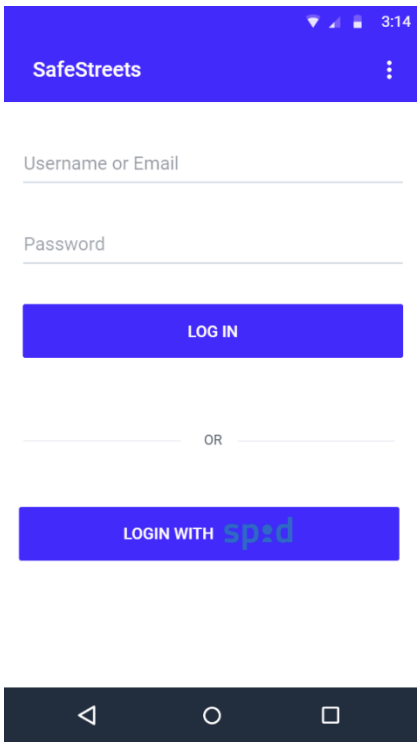



*Figure 16 – Login*

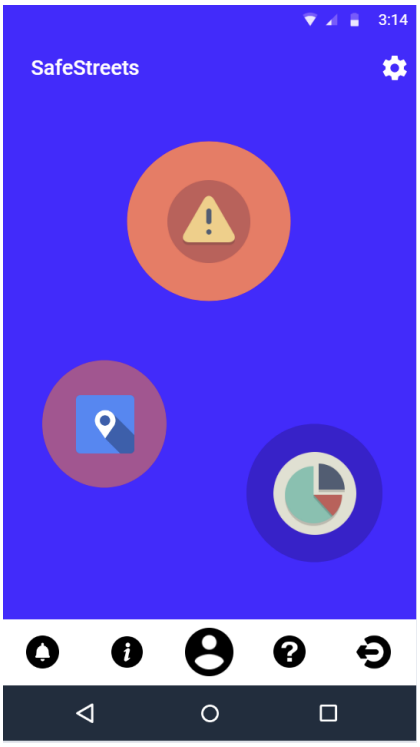



*Figure 17 – Home screen*

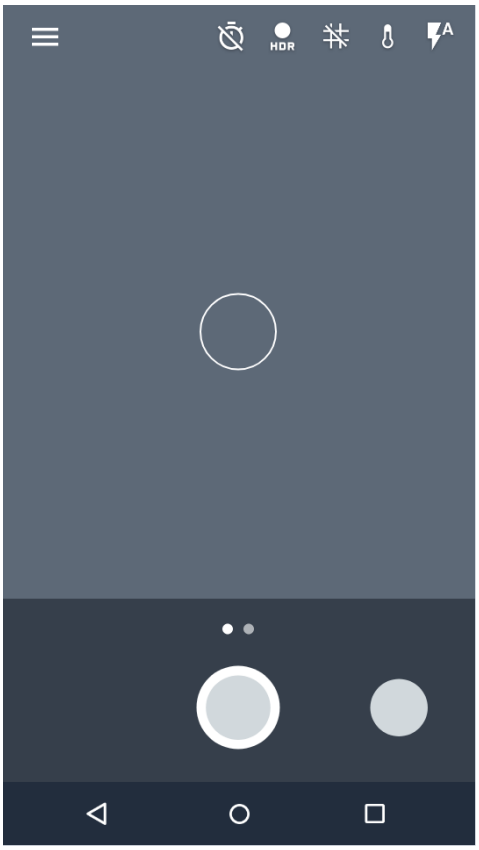


*Figure 18 – Camera for violation picture*

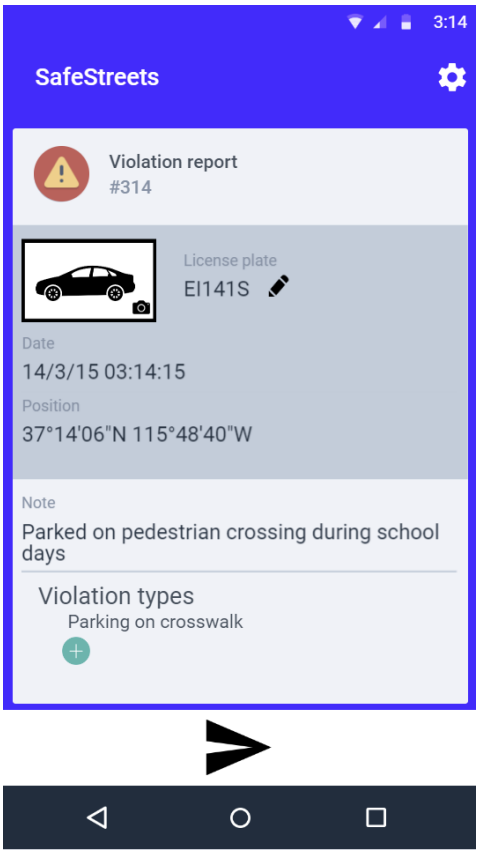



*Figure 19 – Violation report UI*

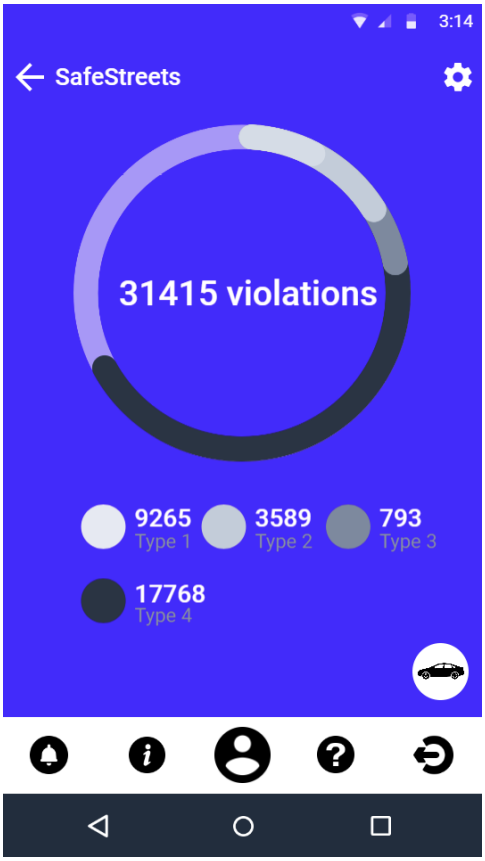



*Figure 20 – Statistics*


*Figure 21 – Safeness Map*

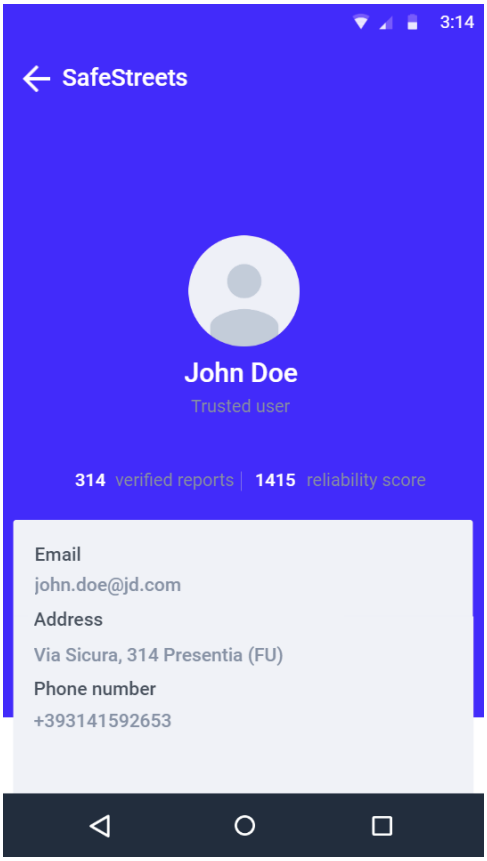



*Figure 22 – User account details*

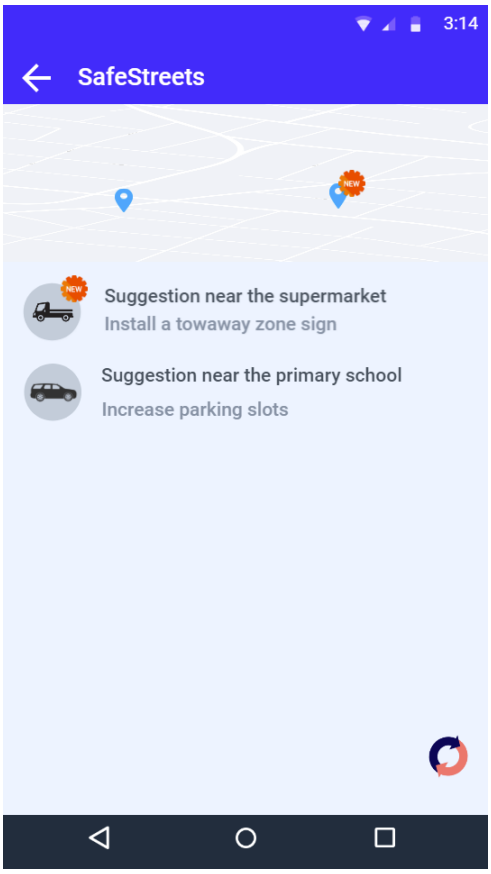



*Figure 23– Inferred suggestions to authorities*

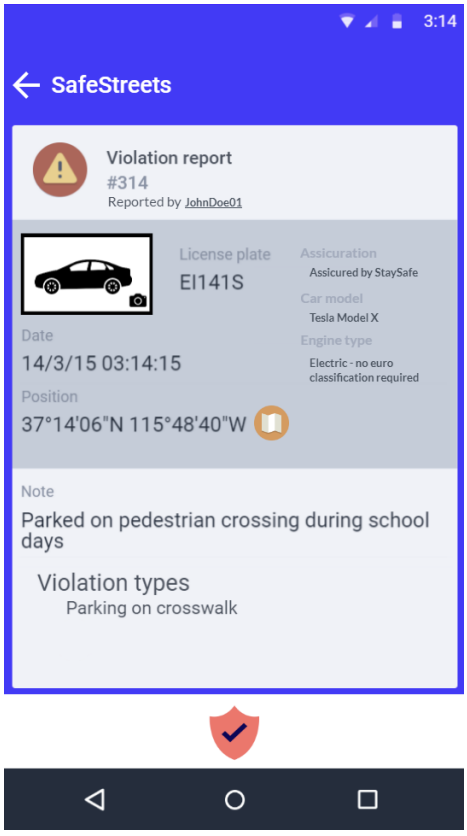



*Figure 24 – Violations reported to authorities*

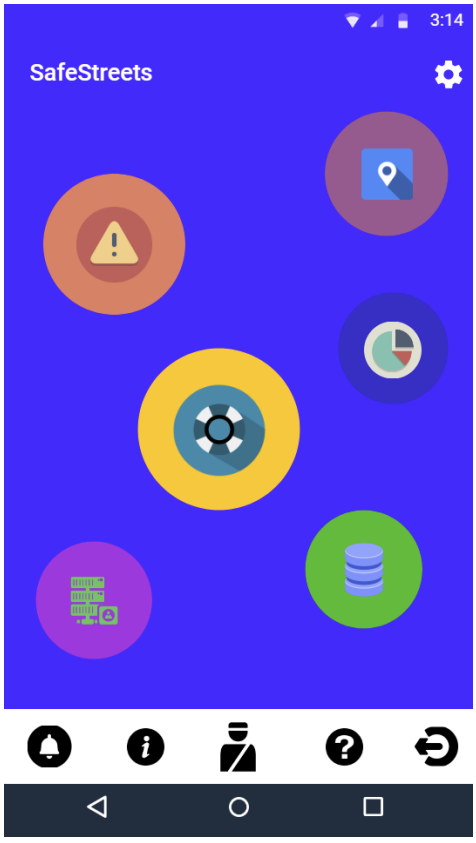



*Figure 25 – Authorities home screen*

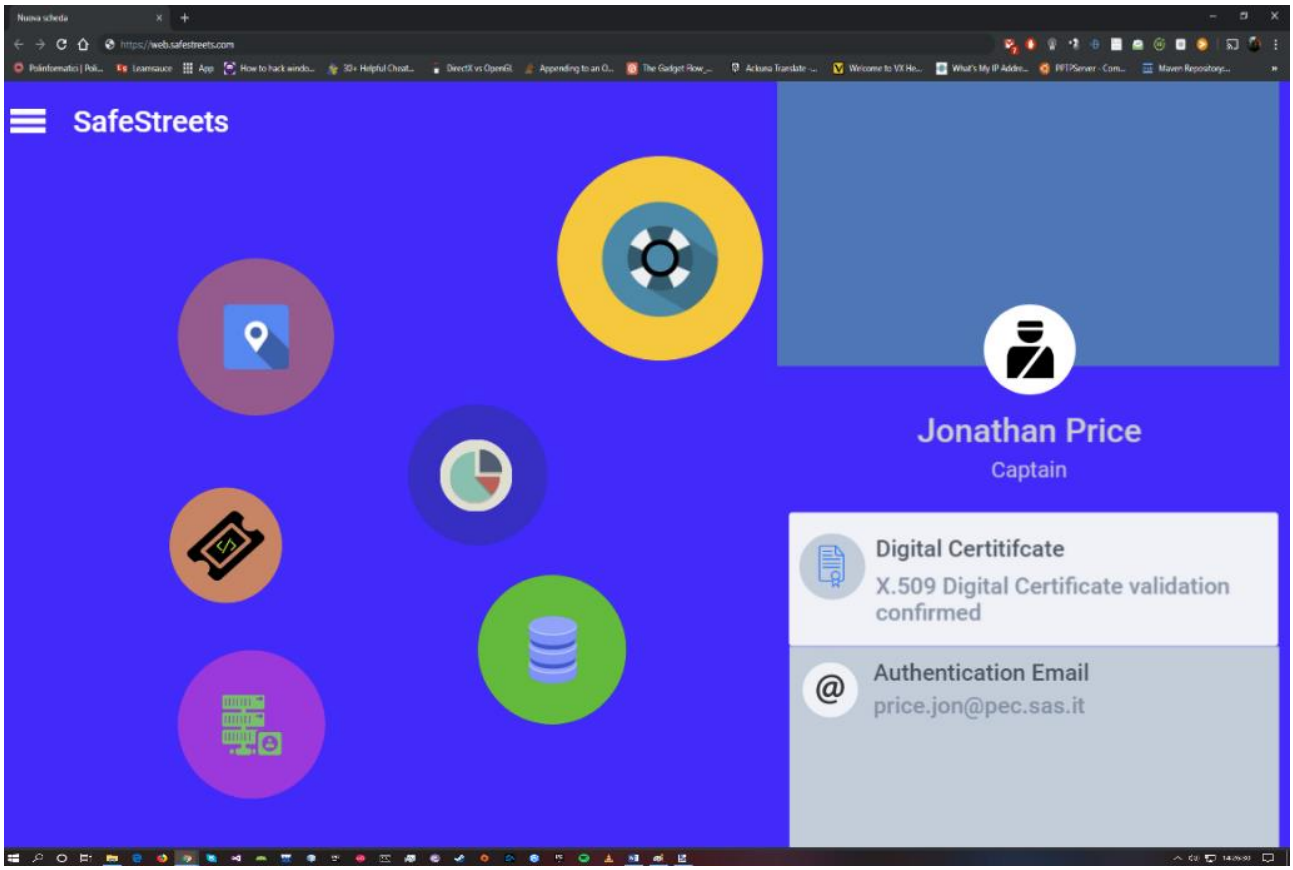



*Figure 26 – exclusive authority SafeStreets desktop access through secure web-app*

The following UX diagrams show the interaction flow between the user and the provided API. There's a new notation as a helper to point out more clearly how functionalities are distributed at different granularities to differ between, in this case, a normal user and an authority. The "X" notation positioned on an interface shows the fact that there is no use for the considered functionality for the user in question.

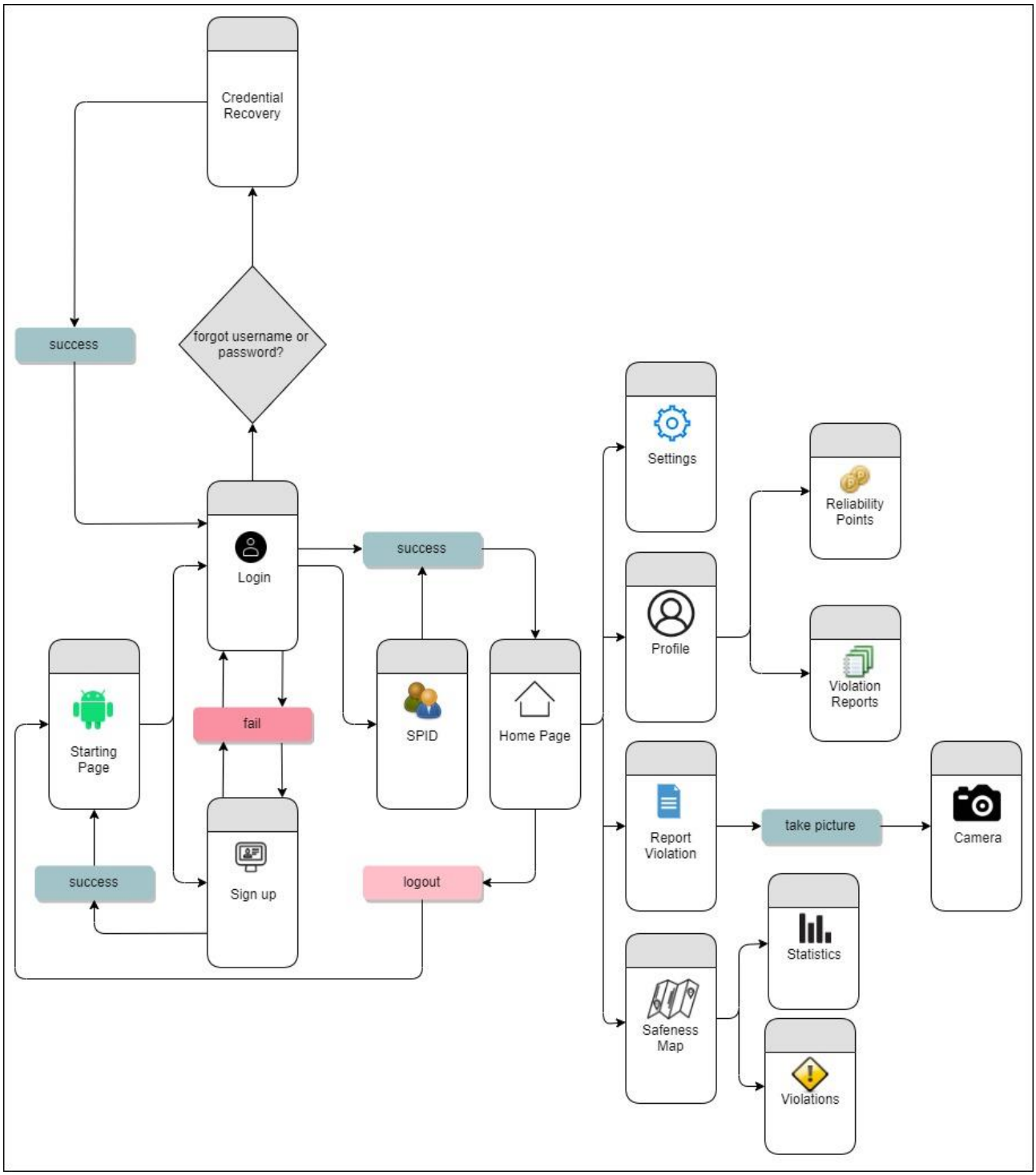



*Figure 27 – Normal User Mobile Application UX Diagram*

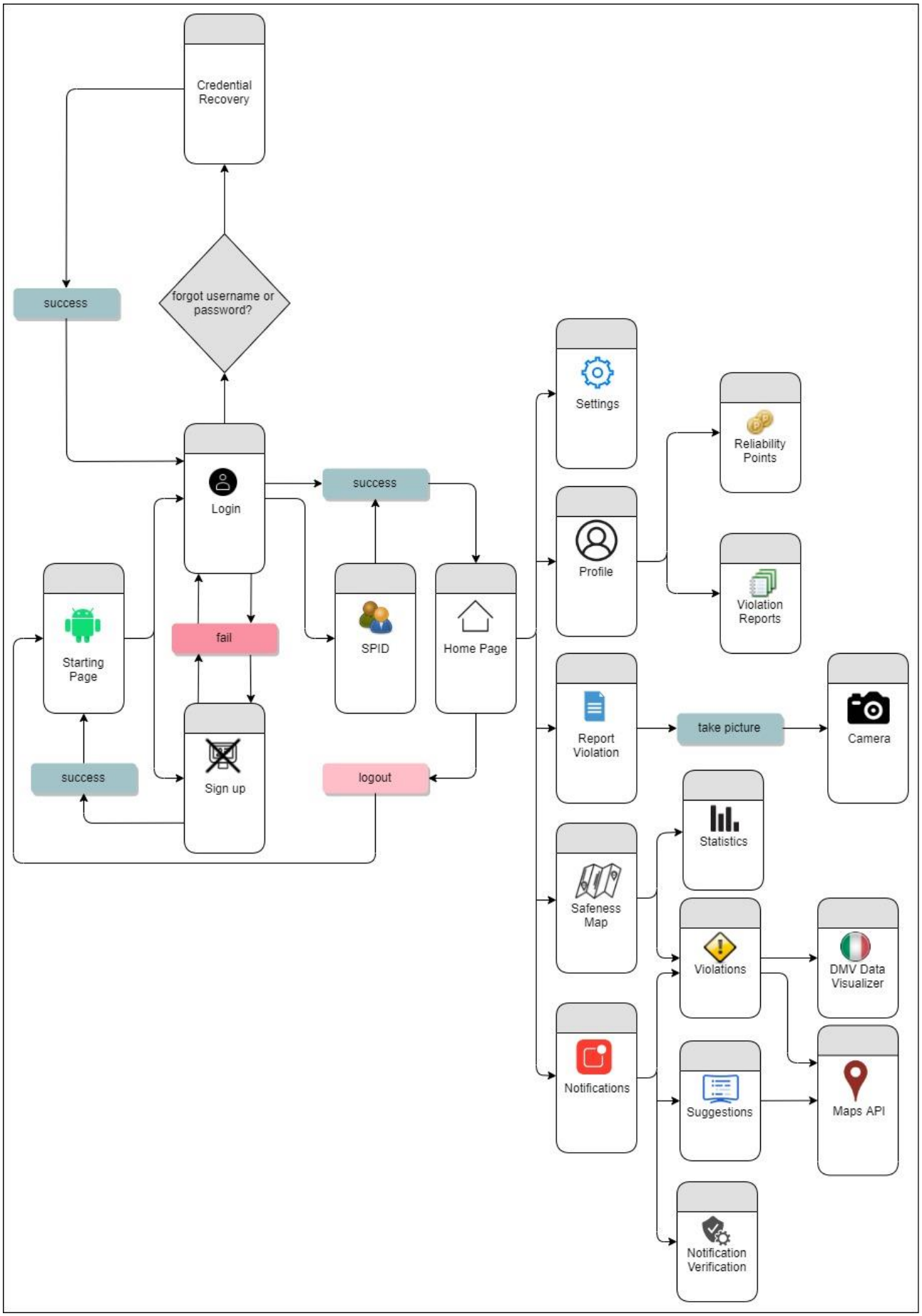



*Figure 28 – Authority Mobile Application UX Diagram*

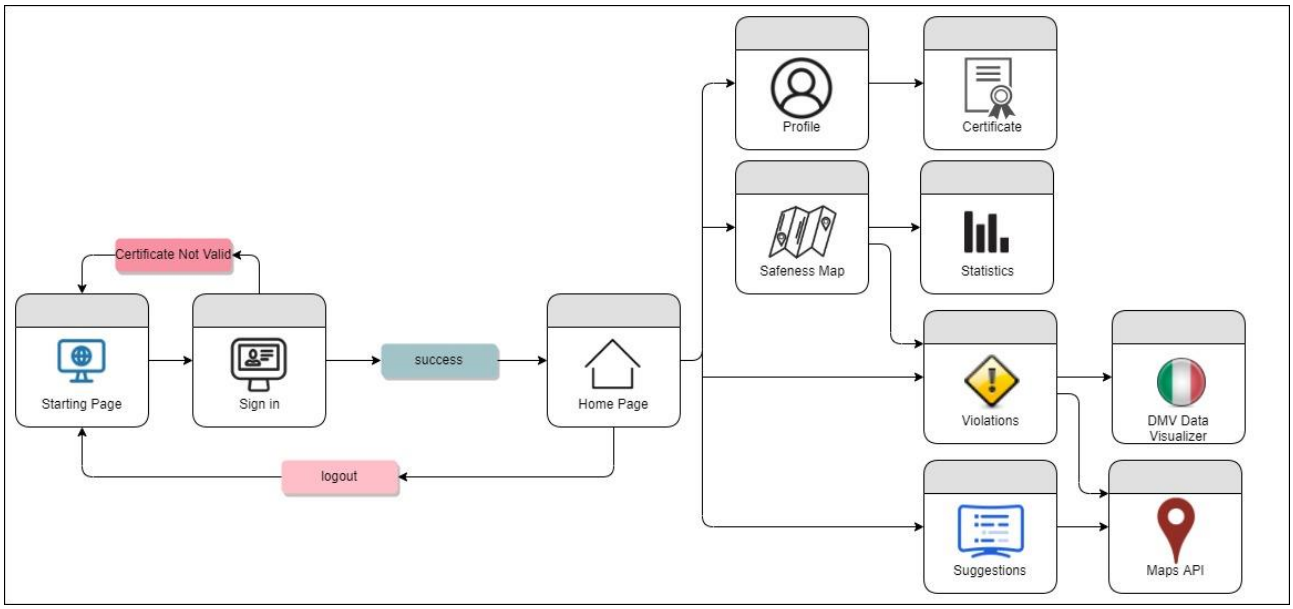



*Figure 29 – Authority Web Application UX Diagram*

# 4 Requirements traceability

Design choices are not just for the sake of the various advantages and improvements already shown but also to grant the fulfilment of the project which was first described in the RASD[11], especially the goals and the various requirements.

Here is shown the complete list of the various requirements with the various associated components previously described which will grant their feasibility.

- [R1] The user must be registered to use the application.
  - **Authentication:** it will handle the login and registration to allow a user to be registered and logged in.
  - **Authentication Manager:** it will handle the communication with the data tier to finalize the login/registration process which was done by the Authentication component.

- [R2] The user can register, and access securely, through two different authentication methods: SPID[9] and proprietary authentication which includes the scanning two different identification documents.
  - **Authentication:** it will handle the login and registration to allow a user to be registered and logged through SPID or the proprietary authentication.
  - **Authentication Manager:** it will handle the communication with the data tier to finalize the login/registration process which was done by the Authentication component.
  - **Credentials Manager:** it will be use for the authority registration as previous stated both in the DD and in the RASD[11] that a password for his credentials will be generated.
  - **User Mobile App:** it will be used by the user to choose which authentication method to use.

- [R3] The user registered with SPID[8] has a higher initial integrity score than a registered user with proprietary authentication.
  - **Authentication:** it will handle the login and registration to allow a user to be registered and logged through SPID or the proprietary authentication.
  - **Authentication Manager:** it will handle the communication with the data tier to finalize the login/registration process which was done by the Authentication component and correctly set the initial integrity score value.

- [R4] Each user has an integrity score.
  - **Authentication:** it will handle the login and registration to allow a user to be registered and logged through SPID[8] or the proprietary authentication.
  - **Authentication Manager:** it will handle the communication with the data tier to finalize the login/registration process which was done by the Authentication component and set the initial score for each user.

- [R5] Each user can access the details of his own, view his data, integrity score and reports made and access and edit its secret authentication credential and receive account updates.
    - **Graph QL Manager:** it will be used to protocolize the data to exchange to the user and to answer correctly to his query.
    - **User Mobile App:** it will be used from by user to request his account details and to visualize them, ask for changes and to receive update on his score.
    - **Violation Manager:** it will be used to retrieve data on his notified violations.
    - **Security Manager:** it is used to access the specific security functionalities which are, in this case, handled by the Cryptography Manager.
    - **Cryptography Manager:** it will be used both from the client and from the middle tier. Middle tier will have to encrypt searched data on violations and the client will have to decrypt those data but also to decrypt its personal data on which no else can do.
    - **Authentication Manager:** it will handle the communication with the data tier to access his personal information and to edit or retrieve his password through the sending of an email via SMTP. All will be done in a secure way through the Cryptography Manager and the Security Manager.
    - **Credentials Manager:** it will be used to generate a valid link to reset the password through the Authentication Manager via SMTP.

- [R6] Each registration made by a user follows the indications imposed by the Legislative Decree 196/03[1] and the Regulation 2016/679[3], which are shown to the user.
    - **Authentication:** it will handle the login and registration to allow a user to be registered and logged in.
    - **Authentication Manager:** it will handle the communication with the data tier to finalize the login/registration process which was done by the Authentication component. All will be done in a secure way through the Cryptography Manager and the Security Manager.
    - **User Mobile App:** it will be used by the user to choose which authentication method to use.
    - **Security Manager:** it is used to access the specific security functionalities which are, in this case, handled by the Cryptography Manager and to handle the hashing of user password for a correct end-to-end encryption.
    - **Cryptography Manager**: it will be used both from the client and from the middle tier. Middle tier will have to encrypt searched data on violations and the client will have to decrypt those data but also to decrypt its personal data on which no else can do.

- [R7] Each authority can securely access the application through his pre-given credentials and its digital certificate provided by the police forces through the Ministry of the Interior[4] and the Ministry of the Defense[5].

- **Authentication:** it will handle the login to allow an authority to be logged in.
- **Authentication Manager:** it will handle the communication with the data tier to finalize the login process which was done by the Authentication component. All will be done in a secure way through the Cryptography Manager and the Security Manager.
- **User Mobile App:** it will be used by the authority to correctly request to be logged in and so to authenticate.
- **Authority Web App:** it will be used by the authority to correctly request to be logged in and so to authenticate.
- **Security Manager:** it will manage thorough the certificate verifier to correctly verify the authority.
- **Cryptography Manager:** it will be used both from the client and from the middle tier. Middle tier will have to encrypt searched data on violations and the client will have to decrypt those data but also to decrypt its personal data on which no else can do.

- [R8] Each authority can have access to the application features available for users without privileged access.
  - **Authorization Manager:** it will be used to verify from who a functionality requests will come from and decide whether can or cannot access a specific functionality or a super set of data.
  - **Graph QL Manager:** it will be used to protocolize the data to exchange to the user and to answer correctly to his query.

- [R9] Each authority has full access to the reports made.
  - **Authorization Manager:** it will be used to verify from who a functionality requests will come from and decide whether can or cannot access a specific functionality or a super set of data.
  - **Graph QL Manager:** it will be used to protocolize the data to exchange to the user and to answer correctly to his query.
  - **Violation Manager:** it will be used to retrieve data on the specific requested violation.

- [R10] Each authority has full access on vehicle information pinpointed by the external Italian license plate verifier[9] and by State Police portal[10].
  - **Authorization Manager:** it will be used to verify from who a functionality requests will come from and decide whether can or cannot access a specific functionality or a super set of data.
  - **Graph QL Manager:** it will be used to protocolize the data to exchange to the user and to answer correctly to his query.
  - **Violation Manager:** it will be used to retrieve data on the specific requested violation.
  - **License Manager:** it will be used, if it was not already done or the needs an update, to retrieve various information from the vehicle's license plate.

- [R11] Each authority can access the details of the report made and the user who carried it out according to the terms established by the Legislative Decree 196/03[1] and the regulation 2016/679[3].
    - **Authorization Manager:** it will be used to verify from who a functionality requests will come from and decide whether can or cannot access a specific functionality or a super set of data.
    - **Graph QL Manager:** it will be used to protocolize the data to exchange to the user and to answer correctly to his query.
    - **Violation Manager:** it will be used to retrieve data on the specific requested violation.

- [R12] Any authority shall receive notifications regarding the municipality in his current position.
    - **Notification Manager:** it will be used to manage the notifications to be sent and the order on which the authorities should receive them.
    - **Google Firebase Manager:** it will send through the Google Cloud Platform the push notifications.

- [R13] A report must consist of an image, date, time, location and metadata.
    - **Violation Manager:** it is the component which handle all reports data.
    - **User Mobile App:** it will guide the user in composing a correct violation report.

- [R14] The metadata of a report is the type of report, the quality of the report and the notes entered by the user.
    - **Violation Manager:** it is the component which handle all reports data.
    - **User Mobile App:** it will guide the user in composing a correct note of the report.

- [R15] The notes entered by the user cannot be longer than 140 characters.
    - **User Mobile App:** it will manage to no let the user insert any more than 140 characters through its UI.

- [R16] Date, time and location must be added automatically via the Internet and GPS/Galileo satellites.
    - **User Mobile App:** it will manage through the localization APIs the user precise position and through the available internet access the time and date of the report.

- [R17] The user can proceed with the violation signalling if the GPS location is precise and obtained securely.
    - **User Mobile App:** it will check the presence of a correct satellite link.

- [R18] It is possible to report in the presence of an Internet connection only.
    - **User Mobile App:** it will check the presence of an available internet access.

- [R19] User reporting image is recognized as valid for reporting only if it contains a vehicle that can be identified through the license plate.
    - **AWS Manager:** it will manage the request to run the CNN model on the received image and it will ask that to the CNN controller inside the Error Handling.
    - **Error Handling:** it will be used, inside this infrastructure composed of various modules, the CNN Controller to request to AWS[12] through its API a validation on a specific received image.

- [R20] The system must be able to recognize the vehicle registration number.
    - **License Manager:** it will be used to retrieve information on the vehicle committing a violation given its license plate.

- [R21] The user can decide to modify the result of the reading of the license plate made by the system.
    - **User Mobile App:** it will let the user through its UI to edit the text representing the license plate automatically recognized.

- [R22] The system includes for each report a warning if the user has modified the vehicle license plate in which the notification will have reported a lower quality.
    - **Violation Manager:** in the case a violation report has been modified by the user during its composition, the information exchanged will allow this module to archive this violation report as edited by the user.

- [R23] The user can't send multiple times the same violation report.
    - **Violation Manager:** during an incoming violation report requests, it will cross check for any violation the user has already done which has a high degree of similarity to the one received in a time, location and vehicle sense. As already stated in the RASD[11].

- [R24] Each user can access a map showing the security level in certain areas.
    - **Map Visualizer Manager:** it will handle the visualization of the safeness area map on the user mobile app and on the web app for the authorities on their side, on the middle tier side it will handle the process of getting all the data elaborated.
    - **User Mobile App:** though the app any user will be able to choose to visualize the safeness area map.
    - **Authority Web App:** though the web app in the browser any authority will be able to choose to visualize the safeness area map.
    - **Authorization Manager:** it will used to handle the authorization level for a given user.

- [R25] Each user can have limited access to reports by viewing information that does not violate the privacy of the reporting user according to the Legislative Decree 196/03[1] and the regulation 2016/679[3].
    - **Authorization Manager:** it will used to handle the authorization level for a given user.

- [R26] Each user can view statistics based on reports made in certain areas.
    - **Statistics Manager:** it will handle all the statistics that has been implemented and their computation.
    - **User Mobile App:** through the user mobile app any user will be able to requests and then access the statistics in various selected areas.

- [R27] Authorities can indicate an alert as verified through the application.
    - **Violation Verifier:** it will receive a violation verification from other modules, but this will then request the classification of verified violation through the Violation Manager.
    - **User Mobile App:** it will allow an authority to mark a violation as verified.

- [R28] Each alert verified by an authority will give the user who has indicated it a higher reliability score.
    - **Violation Verifier:** it will receive a violation verification from other modules, but this will then request the classification of verified violation through the Violation Manager.
    - **Violation Manager:** it will automatically, upon request, increment the reliability score of a specific user.
- [R29] The system must be able to access the accident data present in a specific municipal area if present.
    - **Municipality Data Manager:** it will get incident data from the municipality depending on the specific access given by a certain municipality.

- [R30] The system must analyse accidents and violations data to produce a suggestion to be notified to the authority to improve road safety.
    - **Suggestions Inferring Engine:** it will handle the requests of new suggestion to be computed by the Amazon SageMaker Bayesian (Belief) Network.
    - **Suggestions Manager:** it will handle suggestion notification to the right authorities.
    - **AWS Manager:** it will handle the communication to AWS[12] back and forth, so upon a suggestion computation requests or on any update received from Amazon SageMaker model.

- [R31] Any authority shall receive suggestions regarding the municipality in his current position.

- **Suggestions Inferring Engine:** it will handle the requests of new suggestion to be computed by the Amazon SageMaker[12] Bayesian (Belief) Network.
- **Suggestions Manager:** it will handle suggestion notification to the right authorities.
- **AWS Manager:** it will handle the communication to AWS[12] back and forth, so upon a suggestion computation requests or on any update received from Amazon SageMaker[12] model.
- **Google Firebase Manager:** it will handle the sending of push notification through the Google Cloud Platform.
- **Notification Manager:** it will handle the subset of authorities to send these notifications to.

# 5 Implementation, integration and test plan

In the chapter 5 we will focus on the main aspect regarding implementation, integration and test plan. These chapters will be further divided to create a better and more precise understanding of each part. This chapter also concerns various project management problematics which will be further discussed in the implementation part.

## 5.1 Implementation plan

In this chapter 5.1 we'll talk about the implementation of the various subsystem, in an enough detailed manner, about time and cost managing using various methods of project management and about enlightening milestones to better guide the team which will implement SafeStreets.

### 5.1.1 Subsystem implementation

In SafeStreets there are various subsystems present which are required to be developed so that SafeStreets we'll be able to work properly. From the components diagram we can check them out and from the UML Class diagram we can see in much more detail how they are composed. The interested subsystems are these one below:

- User Mobile App
- Authority Web App
- Web Server
- Application Server
- DBMS

Having a high hierarchy in the structures the main subsystems are highly composed from various sub parts. The implementation of each one of these subsystems will be subject to a certain order, both at this level and at the inner levels of each subsystems. In this upper level it is mandatory that the first subsystems to be implemented and tested are in this order, the DBMS, the Application Server, the Web Server and the two client-side subsystems. All these subsystems will follow a top down approach as better suited for an Object-Oriented approach.

A further description on the order will be done in this chapter focusing also on the implementation details and it will be better explained using the Gantt chart.

Starting from the User Mobile App we will focus on each subsystem and their implementation decisions with also the help of the UML Class diagram.

The User Mobile App representing the thin client in the presentation tier is mainly composed of the presentation part for the user, the user interface, and the communication with SafeStreets so the networking part and the security part, for a secure communication.

The User Mobile App developed for Android OS targeting API 29 but being compatible from Android 6 with API 23 and above, will be implemented using Java with Android Studio, still better choice than Kotlin due to a higher number of experienced developers in Java. The user interface part is being defined in XML with various kinds of layouts, sub-layouts and views, which will be controlled the various Activities and Fragments in the app.

The Activities and Fragments will be following the standard Android app lifecycle and they were identified in this group:

- MainActivity
- LoginActivity
- SignUpActivity
- HomeActivity
    - SettingsActivity
    - ViolationReporterFragment
    - SafenessAreaMapFragment
        - MapView
    - SuggestionFragment
        - FireBaseNotificationService
    - AccountFragment
        - FireBaseNotificationService

As can be seen they were divided in a hierarchical structure which indicates who can call who and to which activity a subpart can come back to.

Each activity and fragment have their own layout description in XML where can also be found custom view components and styles described in XML which are in the resources folder, where also other graphical components are present such as drawables.

Each activity being called from another one will follow the standard Android call procedures with the Intent starting and management API found in the Android SDK, also fragments will follow this standard using the Fragment Transaction Manager.

The SafenessAreaMapFragment together with the MapView will allow the development on the client side of the safeness area map, on this part there is no actual workload present other than just visualizing received data from the middle tier of SafeStreets, then to visualize data on the map will be used, while being on Android, Google Maps API.

The SuggestionFragment indicated will not of course be available for Normal User and will be able to be called, created and used only in the presence of a successful authority access.

The networking part of the User Mobile App is quite standard. The communication as stated in the previous chapters is HTTPS based, so the Apache HTTP library for Android is necessary. This part will be mainly composed by these parts:

- HTTPSHandler
    - HTTPSConnection
- GraphQLParser
- Message

The HTTPS Handler will have to manage a pool of different HTTPSConnection which can be so managed in parallel, depending on which resource they are asking to. It's important to say that those networking components, which will be then directly used by the Activities and Fragments have the need to be called inside AsyncTasks present in the Android SDK, so that the user experience will not be affected from the networking workload and from the time consuming operation which will definitely affect the user interface usability if not

well managed as stated here. But, to send a request there is the need to translate them in a GraphQL representation. This component will just be using the GraphQL API available and will be able to perform any kind of requests while being completely decoupled from the server side of SafeStreets regarding data representation and logical location. GraphQLParser will manage to parse Messages generated from Activities and Fragments containing requests with various kind of data to be store and evaluated.

The Security part of the User Mobile App will be concerning on how to manage, at the client side, an end-to-end encryption. It will be used so encrypt outcoming requests and incoming responses from SafeStreets. The RSA[16] and AES[15] ciphers will be implemented the already present API in the JDK to manage various kind of encryption system with a secure and advanced key store management for both symmetric and asymmetric encryption algorithms. RSA is being mainly used to guarantee server authenticity and exchanging a symmetric AES key. Their specifics were further discussed in the previous chapters. Also, the hashing function used, Argon2(id)[17], can be managed directly from the JDK which will be used to exchange a hashed version of the password to decrypt sensible personal data on which only the user owning them has the access to.

The Authority Web App will follow mainly the development of the User Mobile App in terms of functionalities but it will be developed thinking in a web manner, so using a dynamic web development language which can also be in this case Java using JSP which will be useful even to internal code reuse and so to enhance development efficiency while decreasing development cost and time, which will be better explained in the future chapters.

Moving on the middle tier the main subsystems are in fact the Application Server and the Web Server.

The Web Server will be, as stated in the previous chapters, the first middle tier sub part any client will encounter and to which it will communicate to. It will be developed using as stated in the Deployment diagram NGINX Web Servers which are composed of various workers managed in a highly advanced way to enhance performances. Their main purpose is to exchange data from and to the Application Server using HTTPS. Their actual implementation, of the workers, will be done JSP to manage the various responses and to let the Authority Web App visualize data.

The Application Server is instead highly composed of various which are shown in the Components diagram and even better in the Class diagram.

Starting from the network interchange management we can see that the actual implementation quite follows the networking part on the client in a logical way but it's way more advanced which will allow this subsystem to communicate in the distributed environment in which it will be present. In fact, there will also be an RMI management system which is based on the JDK EE APIs to allow remote procedure calls when needed on other servers in the same cluster or not, depending on the situations, to allow the best possible fault tolerance system.

Messages are here transformed back and forth using GraphQLManager which will be able to understand using GraphQL APIs which requests were asked by the users. This will transform requests in an internal Message representation using the MessageDispatcher

and DynamicRouter together composing the Visitor Pattern to exchange messages back and forth from and to any kind of channel subscription in an advances and efficient way which will allow high future maintenance and easier and more powerful Junit tests.

From the various other components, we can now identify the implementation of the main different functionalities which have to respect the order of development expressed for each one of them and better explained in the Gantt chart.

Starting from the **Registration and Login functionality** which is implemented using a main AuthenticationManager in charge of routing requests to the LoginManager or to the SignUpManager. A login requests from a normal user can be made in a proprietary manner or using SPID[8], either way each one of the two will have its kind of authentication system. In case of a new registration the SignUpManager will come in help with its DocumentVerifier which will be able to scan and verify the two identification documents requested through the usage of the AU10TIX APIs[20] which guarantees a fully automated identification document verification in seconds though their continuous work and improvement on their deep learning model. This functionality has also to manage the authentication process of authorities, it will be done by using their Certified Email[6, 7] and their Digital Certificate X.509 which will be verified through the proper APIs available for this kind of RFC and IEEE standard. User data will be represented in the data tier, which will be better explained later, but also at running time using Java Persistence APIs where each entity is represented as a class to better manage interchange of stored data. For its implementation and testing it is needed the implementation and complete testing of the GraphQL Manager, with all its various parts previously described, and of the Cryptography Manager. It of course also needs the implementation and complete testing of the DBMS component to ensure a correct communication with the Oracle DB[18] and of the Error Handling infrastructure because there is the need to check identification documents and to check data correctness. Also, in order to have a correct authority authentication system is required that the Security Manager implements and tests the Credentials Manager to create the initial password for a registering authority with a previous request through its Certified Email[6, 7].

This functionality of course includes also the recover, which is more of a reset, of the password credential which is done in a standard way by sending a link through an internal SMTP server to the user requesting this reset of the password. This requires the implementation and testing of Authentication Manager which will handle the communication with the data tier to edit or retrieve his password through the sending of an email via SMTP. But this requires the full implementation and testing of Credentials Manager**:** it will be used to generate a valid link to reset the password through the Authentication Manager via SMTP.

All will be done in a secure way through the Cryptography Manager and the Security Manager.

Moving on the **Violation Report functionality** it's based on the Violation Manager, composed of various elements, but depending on other modules which are required to

be tested and implemented in order to implement and test correctly this part. The Violation Manager is basically composed of a ViolationPool which a helper for various operations like checking whether a violation has already been reported by a certain user or not, or for any other query purpose. It's then linked in its usage to the Violation JPA's class and so to Vehicle JPA's class. Also, it will need the LicensePlateManager. So, the license plate manager which uses two other different APIs to work, is required to be fully implemented and tested in order to implement this part, alongside with the CNN Controller, OCR Controller, Notification Manager and the GraphQL Manager.

The CNN Controller and the OCR Controller are needed for a successful image recognition and text extraction from a reported violation, so a correct deployment of Amazon SageMaker models is required. The Notification Manager is needed to create the queue of notifiable authorities in order to let them physically manage this violation, which is also composed of the Notification Dispatcher which creates the dispatch queue and of the Firebase Manager which through the Google Firebase APIs[14] sends the notifications and this is what's requested for the **Authority violation notification functionality**. The GraphQL Manager is instead the component which let us receive the violation report which is well mandatory to work because it also contains the internal Message exchange system with the channel subscription ruled by the DynamicRouter. It of course also needs the implementation and complete testing of the DBMS component to ensure a correct communication with the Oracle DB[18]. Linked to this there is also the need of the Violation Verifier which is the component strictly linked with the Notification Manager in order to know which authority has chosen to go physically check out a violation and to know whether a specific a violation has been marked as verified or not, to correctly update the reliability score of the user who sent a specific violation report and letting the user know through a Firebase push notification.

Here for more clarity purpose is displayed the creation of the Amazon SageMaker instance for SafeStreets, with the selection of the largest and better performant cloud instance as already previously said, on which the AWS Manager will communicate to with AWS APIs[12].

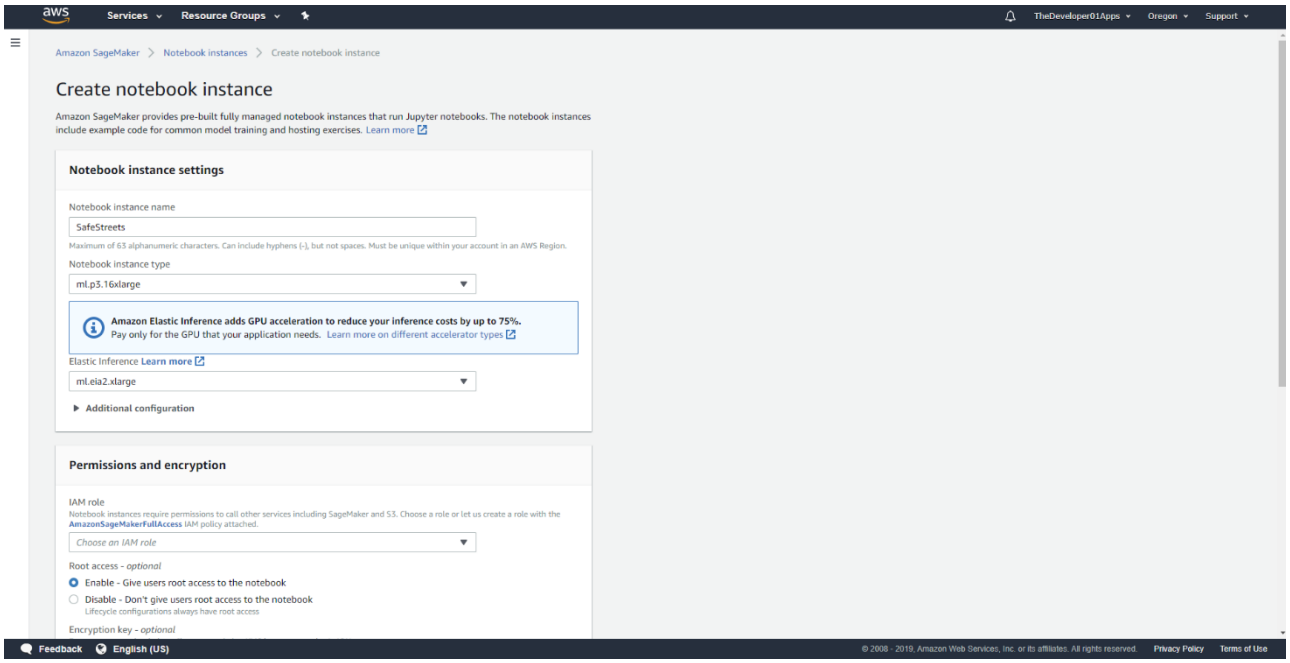



*Figure 30 – AWS instance creation*

Looking at the **Safeness area map functionality**, we can easily see which is directly linked with Violation Manager, so it is mandatory to let that implementation and testing be finished first before this one can be also implemented and correctly tested. It also requires the implementation and the testing of the Statistics Manager component, to allow visualize various statistics and of the Authorization Manager which is needed in order to give a different response to different level of clearance. It of course also needs the implementation and complete testing of the DBMS component to ensure a correct communication with the Oracle DB[18] and of data retrieval.

Then, the **Suggestion inferring functionality** in order to work has the need of a completely implemented and tested Municipality Access Manager to get all the incidents data available for a certain municipality, of the Notification Manager to let the authorities know, this time with no specific dispatching order, the presence of new suggestions available and of the AWS Manager because the suggested data will be elaborated by the model present in the cloud, in Amazon SageMaker, using the Bayesian Belief Network deployment and the Natural Language Processing for a better understanding of the context of any single incidents data, so a correct deployment of Amazon SageMaker models is required. Basically, it works by creating an acyclic graph containing in the nodes the correspondent conditional probability given by the upper nodes as parents. The nodes represent the probability of something to happen which is, for example, a certain decrease of a certain kind of incidents given a certain street improvement approved by the authorities. The probability of the parent nodes is computed empirically given the municipality data and its weight, its importance, computed through the contextualization of the incidents via the NLP. It's very powerful because it can create, through the message passing protocol[22], an exact inference on cause and effect based on real data. Where this data, which comes from the municipality, will be taken over a time range of one year to

guarantee a vastly quantity of data and to have the most correct situation of the incidents occurring in a municipality.

It of course also needs the implementation and complete testing of the DBMS component to ensure a correct communication with the Oracle DB[18].

Another functionality is the **Account information functionality** which oversees the displaying for any users, and so for any authorities too, the details of their specific accounts. In order to make this functionality work there is the need of a fully implemented and tested DBMS component which with the usage of the JPA and of the EntityManager which are present in the Java EE will allow the retrieval and the query of any data regarding a specific user. This is also needed in case of any user would like to change something on his access information, like for example its password.

Lastly, the DBMS subsystem which has been visually better described in the Class diagram is taking advantage of the Java EE Persistence API as already said, to communicate with Distributed Oracle[18] ORDBMS. For this to happen has been chosen to dispatch a two-layer Database Access classes, this has been to encourage further future developments in various and different DBMS systems if they were needed, without touching any other part of the data tier represented in this part of the software architecture.

For a better performance increase has been selected the correct indexes to create which are the primary key of the Users schema and of the Violation schema, that's because they are highly used and large tables on which the related queries need to be as fast as possible in order to get all the data really quick. A secondary index is instead required for the Vehicle schema because it contains too large quantities of data for a primary index, but it's needed to be faster for displaying vehicle information on authorities as fast as possible and is also a frequently used foreign key with respect to the Violation schema.

In the next image is present the Gantt chart representing the whole order and management of the development of SafeStreets. Deadlines are mandatory and are well shown in the Gantt chart.

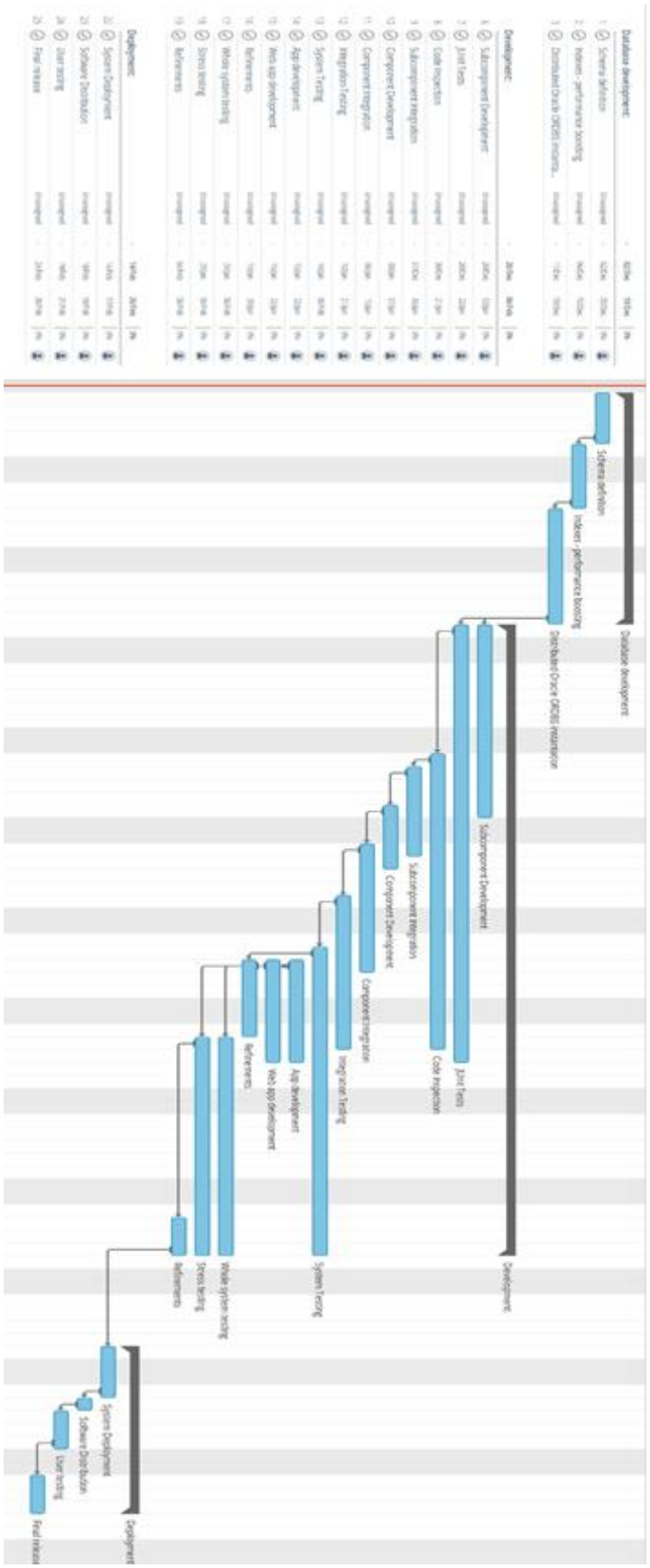



*Figure 31 – Gantt chart*

And here we can summarize the importance and the difficult of a feature.

| Feature | Importance for the customer | Difficulty |
|---|---|---|
| Signup and login | Low | Medium |
| Report a violation | High | Medium |
| Safeness area map | High | Medium |
| Statistics | Medium | Low |
| Suggestions | High | High |
| Notification of reports | High | Medium |
| View account details | High | Low |
| View violations data | High | Medium |

### 5.1.2 Time and cost management

This section is specifically focused on providing some estimations of the expected size, cost and required effort for SafeStreets. For the size estimation part, we will essentially use the Function Points approach, considering all the main functionalities of SafeStreets and estimating the correspondent amount of lines of code to be written in Java. This estimation will only consider the parts of the project that concur to the implementation of the business logic and will disregard the aspects concerning the user interface. For the cost and effort estimation we will instead rely on the COCOMO 2 approach, using as initial guidance the amount of lines of code computed with the FP approach.
Here is presented the COCOMO 2 model which is used:

| Scale Factors | Very Low | Low | Nominal | High | Very High | Extra High |
|---|---|---|---|---|---|---|
| PREC | thoroughly unprecedented | largely unprecedented | somewhat unprecedented | generally familiar | largely familiar | thoroughly familiar |
| $SF_j$: | 6.20 | 4.96 | 3.72 | 2.48 | 1.24 | 0.00 |
| FLEX | rigorous | occasional relaxation | some relaxation | general conformity | some conformity | general goals |
| $SF_j$: | 5.07 | 4.05 | 3.04 | 2.03 | 1.01 | 0.00 |
| RESL | little (20%) | some (40%) | often (60%) | generally (75%) | mostly (90%) | full (100%) |
| $SF_j$: | 7.07 | 5.65 | 4.24 | 2.83 | 1.41 | 0.00 |
| TEAM | very difficult interactions | some difficult interactions | basically cooperative interactions | largely cooperative | highly cooperative | seamless interactions |
| $SF_j$: | 5.48 | 4.38 | 3.29 | 2.19 | 1.10 | 0.00 |
| PMAT | The estimated Equivalent Process Maturity Level (EPML) or | | | | | |
| | SW-CMM Level 1 Lower | SW-CMM Level 1 Upper | SW-CMM Level 2 | SW-CMM Level 3 | SW-CMM Level 4 | SW-CMM Level 5 |
| $SF_j$: | 7.80 | 6.24 | 4.68 | 3.12 | 1.56 | 0.00 |

*Figure 32 – COCOMO 2 model*

The driver choices were:

- Precedent: low
- Development flexibility: low
- Risk resolution: very high
- Team cohesion: high
- Process maturity: Level 2

Which lead to a total amount of 17.29 .

Before going in this computation details, we now focus on the function points which provide an approach to estimate the size of a project taking as inputs the number of functionalities to be developed and their complexity. The estimation is based empirical values which are commonly used among different kind of projects and are defined by different frameworks such as COCOMO 2. So, any empirical value not chosen during the analysis in this document comes from those very frameworks.

The Internal Logic Files (ILFs) in this project are here synthetized:

| **ILF** | **Complexity** | **FPs** |
|---|---|---|
| Login data | Low | 7 |
| Normal Users data | Low | 7 |
| Authority data | Average | 10 |
| Violations | Average | 10 |
| Vehicles | Average | 10 |
| Notification queue | Low | 7 |
| Municipality data | Average | 10 |
| API permissions | Low | 7 |
| **Total** | | 68 |

The External Logic Files (ELFs) in this project are here synthetized:

| **ELF** | **Complexity** | **FPs** |
|---|---|---|
| AWS result retrieval | High | 10 |
| Digital Certificate retrieval | Average | 7 |
| ID Documents retrieval | High | 10 |
| License plate data retrieval | Low | 5 |
| Maps data retrieval | Low | 5 |
| Geo Localization | Low | 5 |
| Municipality data retrieval | Average | 7 |
| **Total** | | 49 |

The External Inputs (EIs) in this project are here synthetized:

| **EI** | **Complexity** | **FPs** |
|---|---|---|
| Login | Low | 3 |
| Logout | Low | 3 |
| Normal user registration | Average | 4 |
| Identification Documents | Average | 4 |
| Authority digital certificates | Average | 4 |

| | | |
|---|---|---|
| Authority authentication | Average | 4 |
| Violation report | High | 6 |
| Safeness map request | Average | 4 |
| Statistic data request | Low | 3 |
| Suggestion requests | High | 6 |
| Statistics request | Average | 4 |
| Violation verification | Low | 3 |
| **Total** | | 47 |

The External Inquiries (EQs) in this project are here synthetized:

| **EQ** | **Complexity** | **FPs** |
|---|---|---|
| Retrieve violations from map | Average | 4 |
| Retrieve statistics data | Average | 4 |
| Retrieve violation data | Average | 4 |
| Retrieve vehicle data | Average | 4 |
| Retrieve user data | Low | 3 |
| Retrieve suggestions data | High | 5 |
| **Total** | | 24 |

The External Outputs (EOs) in this project are here synthetized:

| **EO** | **Complexity** | **FPs** |
|---|---|---|
| Violation report notification | Average | 5 |
| Authority credentials | Average | 5 |
| Suggestions notification | High | 7 |
| Violation verification | Low | 4 |
| Account notification | Low | 4 |
| **Total** | | 25 |

The overall estimation can be here synthetized:

| **Function Type** | **Value** |
|---|---|
| Internal Logic Files | 68 |
| External Logic Files | 49 |
| External Inputs | 47 |
| External Inquiries | 24 |
| External Outputs | 25 |
| **Total** | 213 |

Then, considering Java Enterprise Edition as a development platform we can estimate the total number of lines of code. We will use the upper bound of SLOC which is SLOC = 213 ∗ 137= 28542, where 134 is taken as empirical value from the QSM[19] function points language reference.

We can now choose the values for the various cost drivers:

- RELY: Nominal 1.00
- DATA descriptors: Very High 1.28
- Rating level: Very High 1.34
- Required reusability: Nominal 1.00
- DOCU descriptors: Nominal 1.00

- TIME descriptors: Extra High 1.63
- STOR descriptors: Nominal 1.00
- PVOL descriptor: Nominal 1.00
- ACAP descriptor: High 0.85
- PCAP descriptor: High 0.88
- APEX descriptor: Low 1.00
- LTEX descriptor: Low 1.09
- PCON descriptor: Nominal 1.00
- TOOL descriptors: Very High 0.78
- SITE descriptor: Extra High 0.80
- SCED descriptor: High 1.00

And the total amount is equal to 1.42237810311168, as the product of all the drivers.
The effort is now computed as:

$$Effort = A * EAF * KSLOC^{E}$$

E is computed as:

$$B + 0.01 * \sum SF[i] = 0.91 + 0.01 * 17.29 = 0.91 + 0.1729 = 1.0829$$

We can now compute the effort as

$$Effort = A * EAF * KSLOC^{E}$$

where A is 2.94, equals to:

$$2.94 * 1.42237810311168 * 28.542^{1.0829} = 157.58\, PM$$

So, a schedule estimation can be done as:

$$Duration = 3.67 * Effort^{F}$$

Where F is $0.28 + 0.2 * (E - B) = 0.28 + 0.2 * 0.512 = 0.3824$.
So, the duration is estimated as:

$$3.67 * 157.58^{0.3824} = 25.4 \frac{person}{months}$$

Which can be better estimated in an expected duration depending on the team available and its size. What can be done to do that is by estimating the number of people knowing the time available to develop the project, if we assume a total available time of 3 months, we can assume to, an average of 8 to 9 people working actively on the project. It is also important to denote that the presence of such a complex architecture due also to its distribution and of the various technology present, is not well considered in this limited computation so it's wise to say that this is a lower and improbable bound of working people for obtaining a 3 months result.
It's important to denoted one more time that this is an upper bound, a worst-case effort and duration estimation.

### 5.1.3 Milestones

The milestone of this project is indicated as the most important achievements in order to make the project prosecute its development in the right way and to assess its progresses.

The milestones are here indicated in the following table:

| **Milestones** |
|---|
| Successful creation and installation of the distributed DBMS and of the distributed system architecture |
| Successful implementation and testing of the internal messaging system |
| Successful implementation and testing of the authentication system |
| Successful implementation and testing of the violation reporting system |
| Successful implementation and testing of the suggestions inferring system |
| Successful full system test |
| Successful deployment of SafeStreets |

## 5.2 Integration plan

The following diagrams show which components will go through the integration process which include:

- Integration of external interfaces to the application server
- Integration of the internal components of the application server
- Integration of external interfaces to the user mobile application
- Integration of external interfaces to the authority web application
- Integration of the DBMS

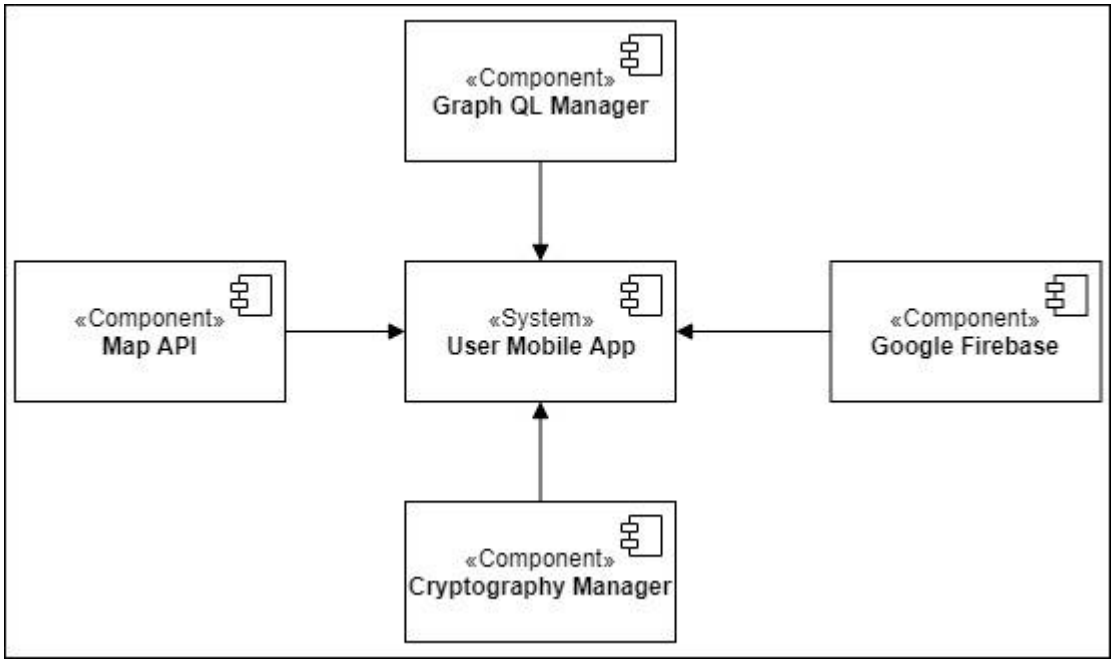



*Figure 33 – User Mobile Application Integration*


*Figure 34 – Authority Web Application Integration*

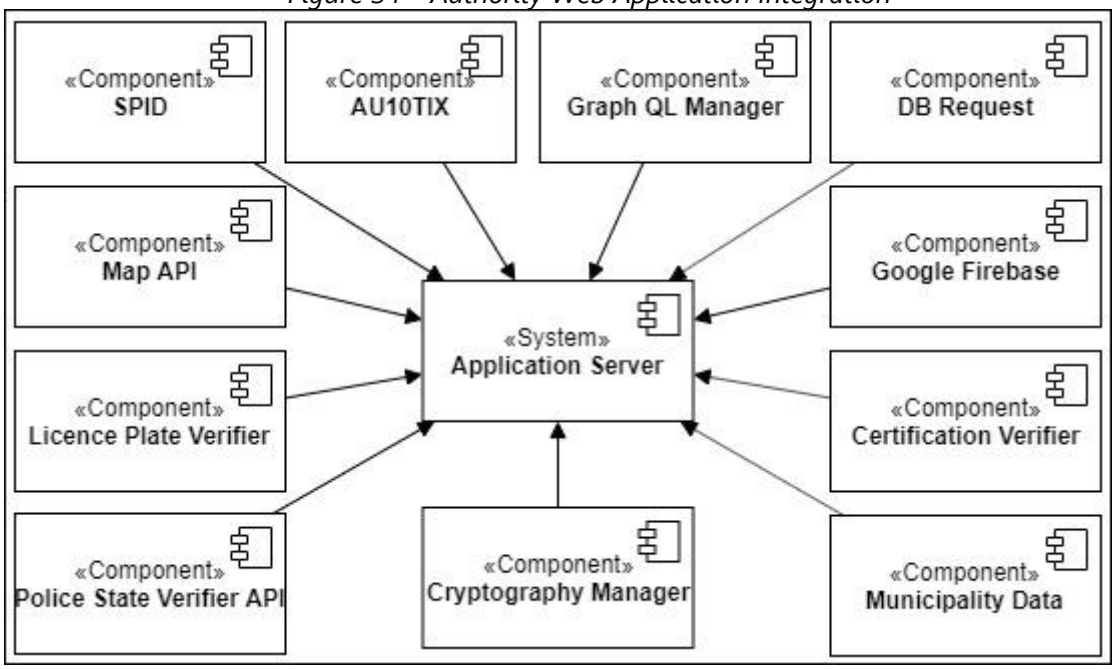



*Figure 35 –Application Server Integration*

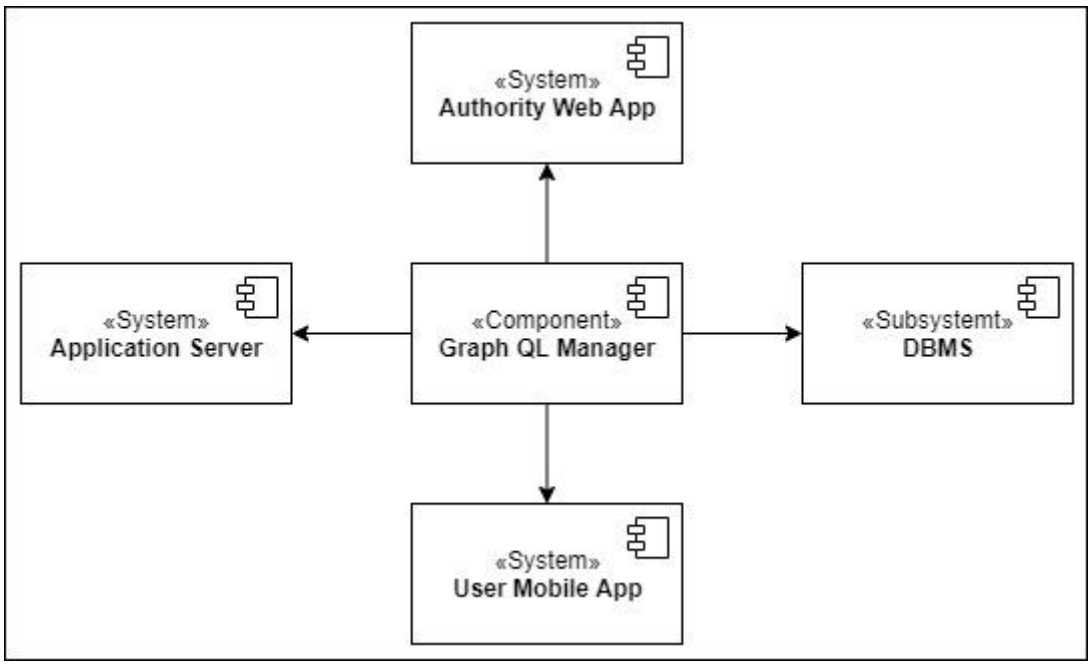



*Figure 36 – Graph QL Integration*

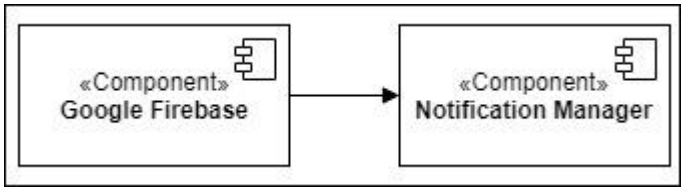


*Figure 37 – Google Firebase Integration*

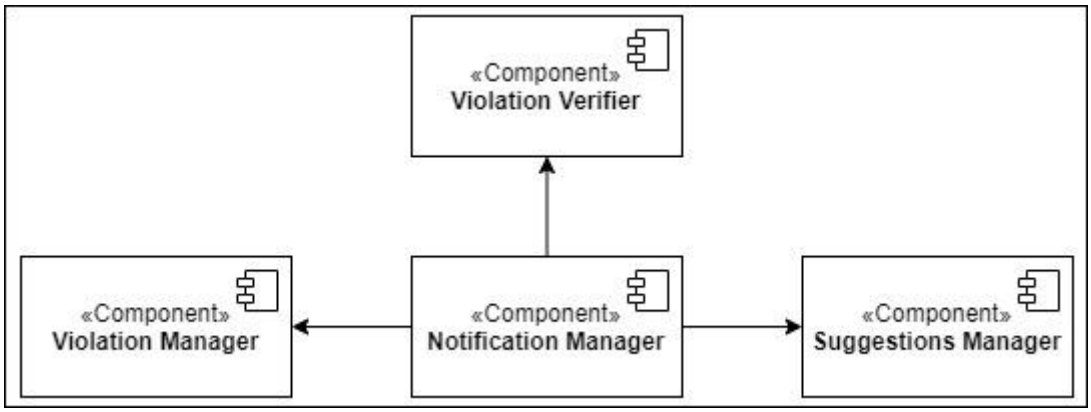



*Figure 38 – Notification Manager Integration*


*Figure 39 – AWS Manager Integration*

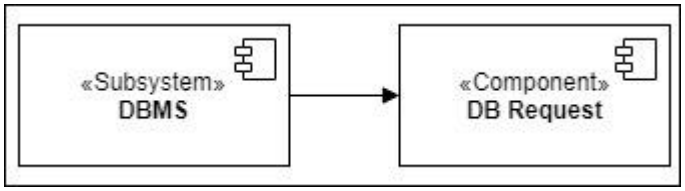



*Figure 40 – DBMS Integration*

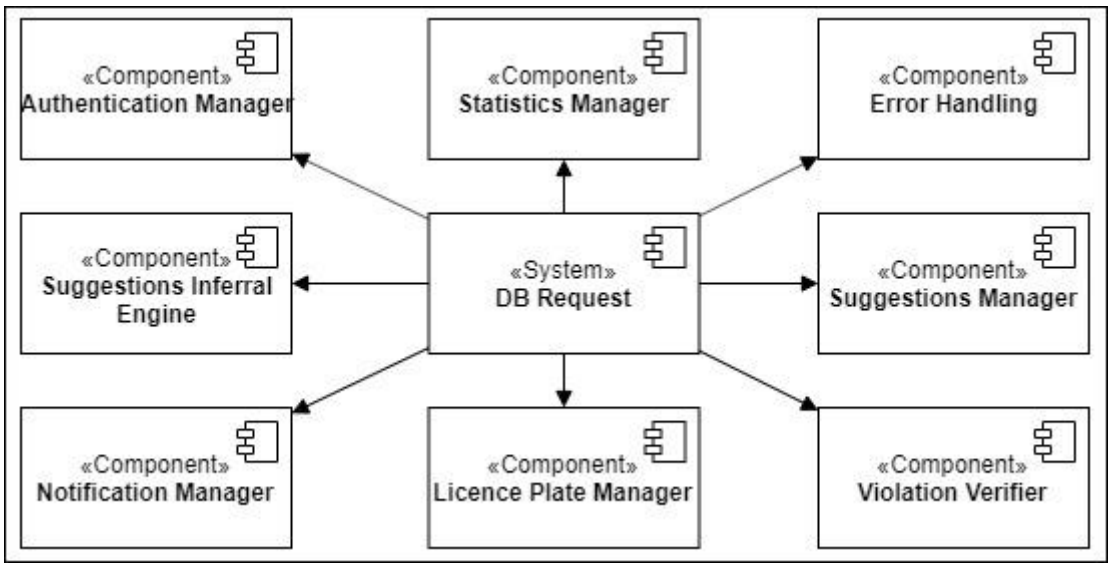



*Figure 41 – DB Request Integration*

*Figure 42 – Map Visualizer Integration*

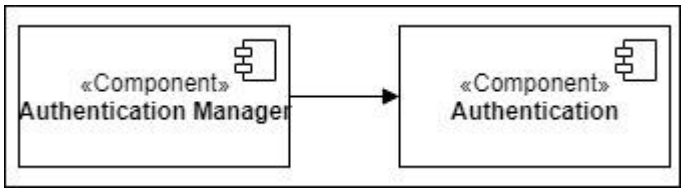



*Figure 43 – Authentication Integration*


*Figure 44 – Authentication Manager Integration*

## 5.3 Testing Plan

SafeStreets development strategy follows a test-driven development approach in a bottom up fashion. So that the testing will force the development to focus first on critical and small modules which development is guided by the selected test suit case and will proceed forward if and only if the testing results with the various assertions are satisfied or a certain level of accomplishment has been reached.

The tests will be implemented through Junit tool. Also, the integration of components as described in the previous section will be tested but of course being a bottom up approach it will happen after all the dependencies will have been successfully tested.

The testing suite will be mostly precise selected in order to cover all the critical parts of the project without losing too much time in covering huge amount of input space with a random approach.

The testing plan will be advantaged on how SafeStreets has been designed, as previous said various times in the document, because it will make the testing easier having already an usable set of components working which will ensure an easy creation of drivers for the various stubs, such as for example the internal messaging system designed using the visitor pattern with the channel subscription but also various other choices already discussed in the previous pages.

Regression testing will also be considered as there would be improvements of the technology used to implement SafeStreets such as the various needed APIs calls. JUnit provides automatic execution of tests. New test cases are added to the regression test suite as a new version is developed. For the test analysis, there will be automated approaches such as data flow analysis, control flow graph, def-use pairs, path conditions and branches. Since the adopted approach is test-driven, incremental integration and testing is better than the big bang approach and as a consequence the components integration testing will follow as already said a bottom-up approach.

It is important to denote that for a successful integration testing these following tasks are necessary:

- Designing the integration test
- Designing a driver (if it was not made at the unit test)
- Designing input test data (if it was not made at the unit test)
- Setting up a system, the components involved, the driver and the input test data
- Performing the integration test

SafeStreets is designed over the middle and data tier as a distributed system so it would be appropriate to apply a performance testing to make sure that everything is working at the required performance standards, and so in case to identify any bottlenecks that may affect response or computation time under a normal workload.
It's always followed by a load testing which will be done in order to cover any memory management bugs present especially in the middle tier in the application servers. Stress testing is considered to be done periodically but not so often and will cover a randomly selected space of events, such as a simulated huge amount of incoming requests and simultaneously the shutting down of various server inside clusters or also the storage server running the Oracle DB and then the result will be analysed to ensure the system recovers gracefully after various failure. The following points show the most important test cases (for the accurate definition, see the RASD):

- **Violation signaller and automated violation validity control**
  The testing unit should be composed of a set of components which aim for the critical points of such functionality. Test cases must measure how accurately the violation manager accepts and refuses pictures and recognizes licence plates and must ensure that the notifications functionality works correctly. The note maximum number of characters and maximum number of violation types must be tested. The time between two different reported violations on the same issue must be tested.
- **Safeness areas map with different visualization levels**
  The testing unit must be online, so that statistics are calculated on different sets of pseudo-random data. Test cases must measure how accurate the statistics are. Security level must be tested also on different granularities such that the data visualization limit is verified.
- **Suggestions Inferring System**
  As in the previous testing unit, it must be online. Test cases must include the fact that there are no data provided by the municipality. The behaviour of the system must be tested on different quantity of data from both SafeStreets and the municipality and must ensure that the notifications functionality works correctly.
- **User reliability score improvement**
  The testing unit must observe the behaviour of the user's profile in case one of the violations are verified and must ensure that the notifications functionality works correctly.

These test cases are reported are only the ones which are considered most important with respect to other, but it's so important to note that there is the need of a lot of other tests as was already suggested from the typology of tests presented above. Also, their input and expected output will be more formally reported in tables. The tests will run in a precise

order: there will be various groups of tests where each group hold common tests together. During a complete test the first group of tests to run will be of the most basic tests, as the bottom up approach suggests, and then it will move to more functionality levels, then integration ones and after others one, ending with the End to End test where everything, starting with the user interaction, will be tested. So, there is an actual Testing Pipeline to be executed, where test in the same group can be done in parallel but different groups must respect the order of execution and run sequentially. The keyword that needs to be use as much as possible is: automation. Automate as possible is the best approach to run better tests, even if there is an initial cost of developing such testing platform or integrating an existing one.

### *5.3.1 Testing tools*

In order to test in the more effectively way it was chosen a set of tools and helpers.

Regarding the middle tier which components will be running on the Java Enterprise Edition runtime environment, four different tools will be used which are here below listed:

- Arquillian[29] integration testing framework: this tool enables us to execute tests against a Java container in order to check that the interaction between a component and its execution environment is happening as it is supposed to be. Specifically, we are going to use Arquillian to verify that the right components are injected when dependency injection is specified, the required APIs work correctly and that the connection via JPA to the Oracle DB works correctly.

- Junit[25] framework: this tool is primarily devoted to unit testing activities, which is very useful to do a correct bottom up test because it will verify that the interactions between components are producing the expected results. In principle it will use to ensure that proper sub components are working properly with their specified algorithms.

- Jenkins[24]: this tool will help the development through an automatic testing, continuous integration approach, execution of source code deployed on the codebase so to ensure everything is working properly before actual committing anything to the master branch of the codebase

- SonarQube[23]: this tool will ensure a complete coverage of any kind of problems in the project such as missing legacy code, possible leaks which will only be verified through load testing but also to know the testing coverage SafeStreets.

- JMeter[29]: this tool will ensure the best possible performance and stress tests possible. It will handle the various request and simulate interaction to the system while logging data to be later analysed from the development team.

- Telerik Test Studio[30]: this tool will ensure to automate everything, as much as possible, regarding tests execution in the sense of interaction so to simulate also any kind of user interaction without also having the need to have different people as testers.

On the mobile side, since Android will be used there are the usual common tools present in Android Studio[26] such as:

- Memory Profiler and Allocation Tracker: this tool will ensure a tracking on the mobile application regarding memory usages to avoid any kind of leaks or overused memory which can be an important part to consider avoiding over-usages on low level smartphone.
- CPU Profiler: this tool will ensure a better performance testing result thanks to the recording of CPU usage in various parts of the application, such as the networking pat done in an asynchronous way and to reduce also battery usage which is an important part to consider in mobile applications.
- AVD: this tool will ensure to test the mobile application over a large a set of different Android devices and Android operating system version and a variety of hardware configurations which will be done through the Intel HAXM[27]
- XML Layout Inspection: this part of Android Studio will help to ensure a better layout development.
- Instant Run: this part of Android Studio will help to enhance the speed up of mobile application deployment and testing.

On the web app part will be used the instrument provided by the browsers such as Google Chrome and Firefox inspection tools but also Selenium to automate the interaction with the web interface.

# 6 Effort spent

Daniele Comi

| **Description of the task** | **Hours** |
|---|---|
| Overview | 13 |
| Deployment view | 11 |
| Selected architectural styles and patterns | 13 |
| Other design decisions | 7 |
| Requirements traceability | 3 |
| Implementation plan | 11 |
| User interface design | 2 |
| Testing plan | 2 |

Anton Ghobryal

| **Description of the task** | **Hours** |
|---|---|
| Introduction | 2 |
| Components view | 13 |
| Runtime view | 23 |
| User interface | 2 |
| Integration plan | 7 |
| Testing plan | 3 |
| User interface design | 2 |

# 7 Reference


1. D.L. 196 of 2003 (196/03) https://www.camera.it/parlam/leggi/deleghe/Testi/03196dl.htm
2. D.L. 82 of 2005 (82/05) https://docs.italia.it/italia/piano-triennale-ict/codice-amministrazione-digitale-docs/it/v2017-12-13/index.html
3. General Data Protection Regulation (EU) 2016/679 https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679
4. Ministry of the Interior and digital certificates released http://politichepersonale.interno.it/itaindex.php?IdMat=1&IdSot=35&IdNot=386
5. Ministry of the Defence and digital certificates released http://www.pkiff.difesa.it/#secEN
6. Certified Email https://www.agid.gov.it/it/piattaforme/posta-elettronica-certificata
7. Certified Email RFC https://tools.ietf.org/html/rfc6109
8. SPID https://www.agid.gov.it/it/piattaforme/spid
9. Italian license plate verifier http://www.targa.co.it/data/doc.aspx
10. Police State license plate verifier https://www.crimnet.dcpc.interno.gov.it/crimnet/ricerca-targhe-telai-rubati-smarriti/FAQ
11. RASD
12. AWS https://docs.aws.amazon.com/
13. Cloudflare https://developers.cloudflare.com/docs/
14. Google Firebase https://firebase.google.com/docs
15. AES https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
16. RSA https://community.rsa.com/docs/DOC-60094
17. Argon2id https://www.cryptolux.org/images/0/0d/Argon2.pdf
18. Oracle DB https://docs.oracle.com/en/database/
19. QSM https://www.qsm.com/resources/function-point-languages-table
20. AU10TIX https://www.au10tix.com/
21. GraphQL https://graphql.org/
22. Message passing protocol in Bayesian Belief Networks http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf
23. SonarQube https://docs.sonarqube.org/latest/
24. Jenkins https://jenkins.io/
25. JUnit https://junit.org/junit5/docs/current/api/
26. Intel HAXM https://github.com/intel/haxm
27. Android Studio and tools https://developer.android.com/studio/profile
28. Arquillian http://arquillian.org/
29. JMeter https://jmeter.apache.org/
30. Telerik Test Studio https://docs.telerik.com/devtools/teststudiodev