SafeStreets
DD document

Dario Miceli Pranio
Pierriccardo Olivieri

Academic year: 2019 - 2020

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide a functional description of *SafeStreets* application.

1.2 Scope

SafeStreets is a service that aims to provide *Users* with the possibility to notify *Authorities* when traffic violations occur, and in particular parking violations. The application's goal is achieved by allowing *Users* to share photo, position, date, time and type of violation and by enabling *Authorities* to request them.

SafeStreets requires the *Users* to create an account to access its services, the functionalities unlocked after registration depend on the type of account created.
If a *User* creates an account as *Citizen*, he/she must provide name, surname and a fiscal code in order to prove that he/she is a real person. Furthermore, he must provide an email with which he will be uniquely identified and a password. Once the account has been activated, *User* can finally start to report parking violations and can also see statistics of the streets or the areas with the highest frequency of violations.

On the other hand, an officer will create an account as *Authority* and he will need to provide his name, surname, work's Matricola, a password and as for *Citizen*, will be uniquely identified by an email. Once the Matricola has been verified and the account has been activated, the officer can retrieve the potential parking violations sent by *Citizen* that have not been taken into account yet by other officers, analyze them and, if it is the right case, generates traffic tickets. *Authorities*, can see the same statistics of the *Citizen* and can also see statistics about vehicles' license plate that commit the most violations.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

- *Users*: can be either *Citizen* or *Authority*
- *traffic violation*: generic violation that can occur in a street
- *parking violation*: a violation caused by a bad parking
- *violation*: general violation, identity both traffic or parking violation
- *unsafe areas*: areas with an high rate of violations
- *Driver*: a piece of software that simulates a component that depends on the module undergoing testing
- *Stub*: a piece of software that simulates a components on which the module undergoing testing depends
- *Client*: Software *System* on the user's device that requests services to the Server
- *Server*: Software *System* that handles requests from different clients

- *n-tier*: Distributed architecture composed of n hardware components, each containing one or many layers
- *n-layer*: Distributed architecture composed of n software levels, each distributed on one or many tiers.
- *Design*: Pattern: Reusable software solution to a commonly occurring problem within a given context of software design.

1.3.2 Acronyms

Table with all acronyms used in document.

ACRONYM	COMPLETE NAME
DD	Design Document
RASD	Requirements Analysis and Specification Document
GPS	Global Positioning Systems
S2B	Software To Be
STNP	Simple Time Network Protocol
FC	Fiscal Code
DB	Database
ORM	Object-Relational Mapping
MVC	Model View Controller
DBMS	Database Management System
REST	REpresentational State Transfer
UI	User Interface
UX	User Experience
HTTP	HyperText Transfer Protocols

1.3.3 Abbreviations

- **Rn**: n-th Requirement

1.4 Revision History

1.5 Reference documents

- ISO/IEC/IEEE 29148: https://www.iso.org/standard/45171.html
- Specification Document: "SafeStreets Mandatory Project Assignement"

1.6 Document Structure

This section is a guide through the document, a short explanation on what we will discuss.

- **Chapter 2: Architectural Design**
  In this chapter, starting from an informal overview, is described deeply the *System* architecture, with the use of components diagrams and component interface diagrams. It is also described using sequence diagrams how *System* behave in a runtime perspective.

- **Chapter 3: User Interface Design**
  As already described in the RASD document with mockups, in this section is analyzed the UI flow that shows how those mockups interact in a dynamic context.

- **Chapter 4: Requirements Traceability**
  This chapter is focused on the relation between the requirements from the RASD and the design choices of the DD.

- **Chapter 5: Implementation, Integration and Test Plan**
  Explain the implementation for *SafeStreets* application and how it is tested, the integration of all component and how is organized the testing phase.

- **Chapter 6: Effort Spent**
  A table where is described the effort spent for each group member for this project.

2 Architectural Design

2.1 Overview: High level components and their interaction

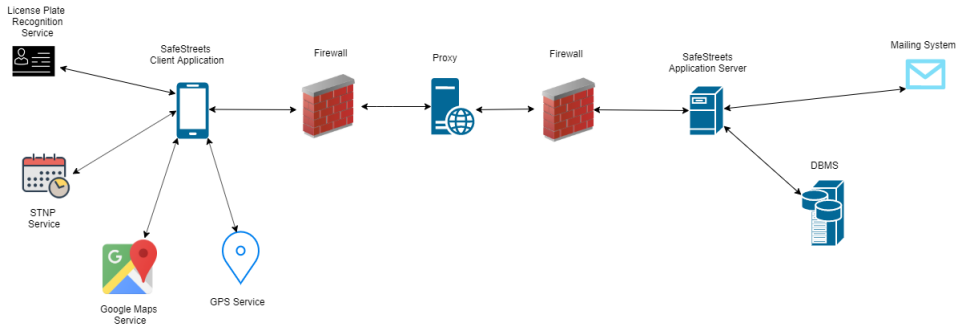

Figure 1: *System's* Overview

In this graphical representation of *SafeStreets* we describe at an high level the main interaction between the components that are involved. Focusing on client side *SafeStreets* provides a Client application, in this view case we don't make a distinciton between the *Users* type, this is primarily devoted to the fact that the requests are similar. We identify then a 3-tier architecture, composed by the client, a proxy in the middle to manage properly all the requests and finally a back end part in which the *System* store the information (in a DBMS) and generate statistics thanks to the data submitted by the *Citizen* and confirmed by *Authorities*. In order to do this we need an application server. Since we will authenticate the *Users* through a confirmation email, and we give the possibility to the *Authority* to receive data via mail, the *System* needs also a Mailing System. Finally we also need some service in the client like GPS Service, Maps Service, STNP Service and License Plate Recognition Service. These services are essential to create a proper violation and to send it correctly.

2.2 Component view

Component High Level View
The following 3 images will show at an high level how the *System* is organized, as we can see the choice we made is a classic client-server architecture.

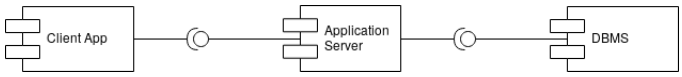

Figure 2: Component Diagram High Level View

The client presents various components each one covering a different role: the Presentation Manager that corresponds to the "View" in the MVC Pattern is the medium between the application and the *User*, Logic Manager is the component designed to catch all the application Logic, it orchestrates the send report action for *Citizens* and the retrieve report action for *Authorities*, Logic Manager is also connected to a cache storage in order to save momentaneously data. The Security Manager, crucial for the application, is the component used to ensure the message security during the trasmission to server, his main job is to apply a digital signature. The Network Manager handles the connection and is the component that communicates with server.

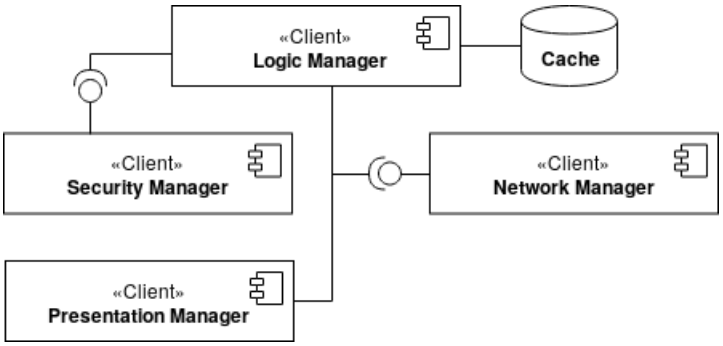

Figure 3: Client view

The server presents the Network Manager for communication, and various components for *SafeStreets*'s main actions like report, statistics and settings that are embedded in the Report Manager, Statistics Manager and Settings Manager respectively. To handle all the auth operations we will use the Authentication Manager, that with the help of Privacy Manager will authenticate an user in order to give access to actions. All this components needs to communicate with database in order to store permanently or modify data, and this is possible thanks to the Database Manager, infact database is system-independent, and all the communications pass through this component. The Statistics Manager has also a cache in which can store the generated statistics to send to the requestor, this is a way to reduce database access and to fulfill the request in a rapid way.

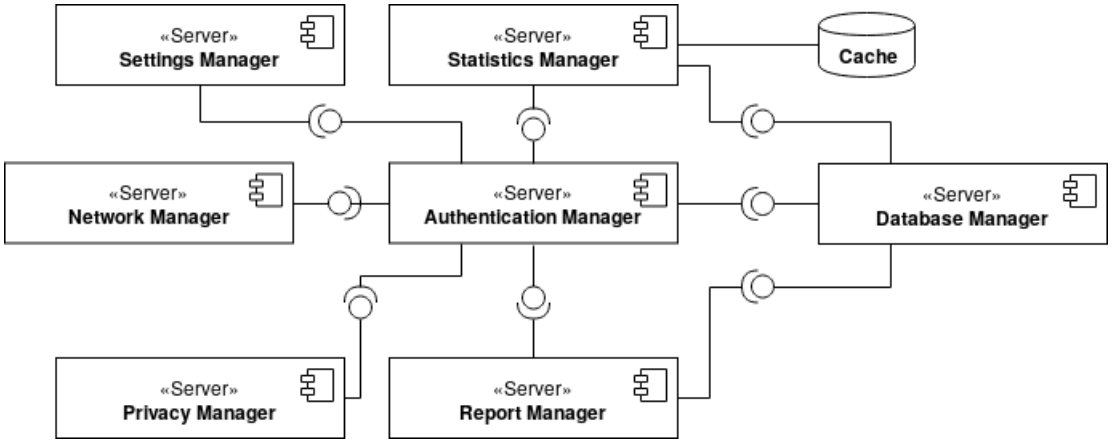

Figure 4: Server view

Component View
Here we go deeper in the Component Diagram, are described all the interactions between the component described above and all the external services like the Maps service, license plate Recognition Service and STNP service for the client and Mailing System for the server.

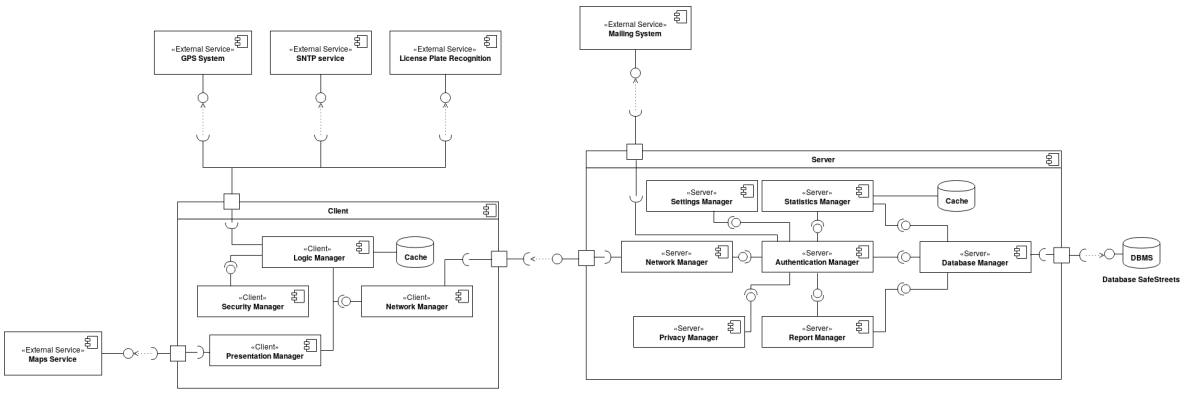

Figure 5: Component Diagram Detailed view

The purpose of this UML diagram is to show the internal architecture of the *System*'s software. It's divided in three component: *SafeStreets* Client Application, *SafeStreets*'s Server, External interfaces. Below we will describe each component.

Network Manager:
The network manager is the component designed to manage the sending and receiving of messages exchanged between client and server, it is located both in the client and in the server, this component also has the task of correctly routing the messages to the correct component.

Client:

- **Presentation Manager**:
  This client side component is the main intermediary between User and *SafeStreets*.

Interface management is totally entrusted to this component, in terms of scalability it is advantageous to separate the interface from the rest of the application. Encapsulating everything in the Presentation Manager will be helpfull to reduce problems whether in future will be necessary to change part of the interface or to develope the client for a new type of operating system.

- **Security Manager**:
  Security in report transmission is critical. Separating security techniques into a separate component *SafeStreets* can ensure greater scalability in terms of security by implementing the best methods. The component currently deals with the digital signature of the report before sending and also checking the validity of the digital signature of a report.

- **Logic Manager**:
  This is the main component of the client, it communicates with the presentation manager, security manager, network manager and with all the external services, it is the central unit of the Client. The purpose of this component is to perform all the operations that the User requires through an interface such as report sending operations (*Citizen*) or request report (*Authority*) but also the statistics request and the edit settings. The first two (send and receive report) operations are very important, the Logic Manager is therefore connected to a cache that has multiple uses, in the case of the Citizen this cache is used to temporarily save the report, this step is necessary because if the user sends the report and this is subjected to a malicious attack the server will notice upon arrival that the digital signature does not match, discarding that report. To avoid loosing it, the Logic Manager saves it in the cache and sends it periodically until the confirmation is received. The Logic manager also deals with the creation of the report, carrying out all the necessary operations, collecting data from external services, executing the license plate recognition algorithm and using the services of the Security Manager to apply the digital signature to the report to guarantee its security in the way to server, in the same way for the report retrieval function, with the help of the Security Manager, the validity of the digital signature will be checked.

Server:

- **Database Manager**:
  The database access intermediary resides in this component, every access to the database in fact passes through the Database Manager, this brings advantages in scalability and flexibility since it makes the server independent from the database. Among the various operations it deals with the authentication of the user, therefore the operations of login and registration, of statistics recovery and report insertion or recovery. This component will use an ORM in order to abstract the implementation details of the Database.

- **Authentication Manager**:
  This component is of fundamental importance on the server side, not only does it take care of all the login and user registration procedures, it is also the access port for which all the other components must pass in order to execute their procedures, in fact no component can perform an operation coming from a request from an unauthenticated user.

- **Privacy Manager**:
  This component deals with the security of users' sensitive data, especially passwords and personal data must be encrypted before being inserted into the database, for greater flexibility and any changes it was preferred to separate that component. It communicates exclusively with the Authentication Manager and has the task of check the hash of the password stored in the DB with the hash of the password sent by the client.

- **Settings Manager**:
  The user may want to change his data, update the email or name and surname. To do this, these requests, on the server side, are forwarded to the Settings Manager which communicates directly with Authentication Manager and Database Manager to ensure the success of the action.

- **Statistics Manager**:
  The statistics of the application are generated at time intervals, in fact being updateable only when other data have been evaluated by the competent *Authority* does not make sense to constantly update them. This component then takes care of the creation of the statistics and saves them in the appropriate cache where they are updated periodically. The Statistics Manager generates all types of statistics for both Violations and Vehicle and forwards the most updated from the cache to the Client when requested.

- **Report Manager**:
  The Report Manager takes care of the two key functions of *SafeStreets*. The send report and the retrieve report, in fact by communicating with the database through the Database Manager it can save the incoming reports or manage the sending of those that should be evaluated. It also deals with the security logic, in fact if the report received does not correspond with the digital signature this will be immediately discarded. When a report is evaluated by an *Authority*, this component will take care of updating the respective report and notifying the statistics manager.

External interfaces

Some of the described components in our *System* are also dedicated to communicating with external services through specific interfaces. It is essential that the communication works properly in order to fulfill application's functionalities. This external services are:

- Database
  The component devoted to interacting with the central database on cloud is the Database Manager.
- GPS
  On the Client side of the application, *SafeStreets* access data from the *User*'s device's GPS through the Data Manager Interface. GPS information are important because are used in report to locate a position.
- Mailing System
  It used by Authentication Manager to inform an *User* that his account has been correctly created.
- Maps Service
  It's used to show unsafe areas in statistics.
- STNP service
  To retrieve the correct Time and Date to attach at report.
- License Plate Recognition Service
  This service taking in input the violantion's photo will recognize if possibile the license plate.

2.3 Deployement view

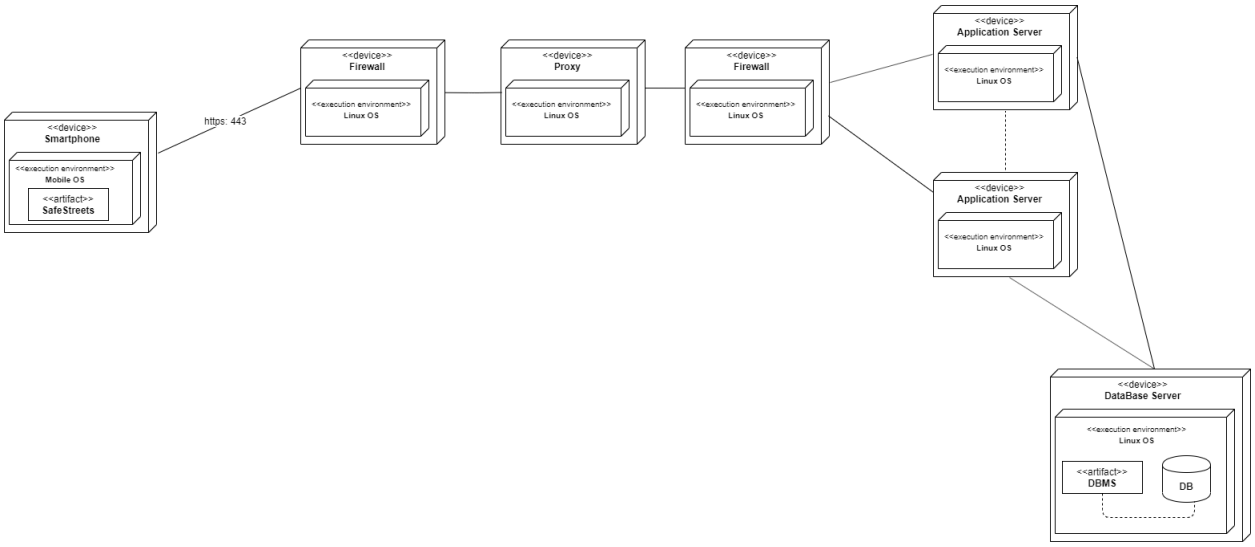

Figure 6: Deployment view

The architecture choosed for *SafeStreets* is a multi-tier architecture. Below all the nodes involved will be described.

Smarthphone
Represents the *User's* smarthphone in which the client will run. This is the client machine that will run *SafeStreets* application.

Firewall
Necessary to provide protection between local network and world network, so not-allowed third parties will not be able to access data. In particular we decided to use two firewalls to create a DMZ. The first firewall must be configured to allow traffic destined to the DMZ only. The second firewall only allows traffic to the DMZ from the internal network.

Application Server
Is the central unit of *SafeStreet* all the other components refer to this. It contains all the logic and provide bidirectional access to Database i.e. manages data acquisition and requests. We decided to fully replicate Application Servers in order to balance the workload

Proxy
System will use a proxy to receive requests. This solution is more scalable for the eventuality of further improvements and guarantee a better stability of the *System*. Furthermore Proxy acts as additional shield against security attacks.

Database
A Database is necessary in order to store all the informations about personal data, registration and data submitted by *Citizen* and retrieve information in order, for instance, to build statistics.

2.4 Runtime View

Proxy Runtime View

The following sequence diagram describes how the Proxy interacts with the Client and the Application Server. When an *User* send a request, checks whether such request conforms to its defined schema and, in case the *User* has to be logged in, checks if the provided authToken is associated to an actual logged in *User*. This last check, in particular, is efficient to be done on the Proxy instead of each Application Server, because if a *User* logs in with a particular Server which then crashes, the *User* doesn't have to log in again, since the information is stored on the Proxy.

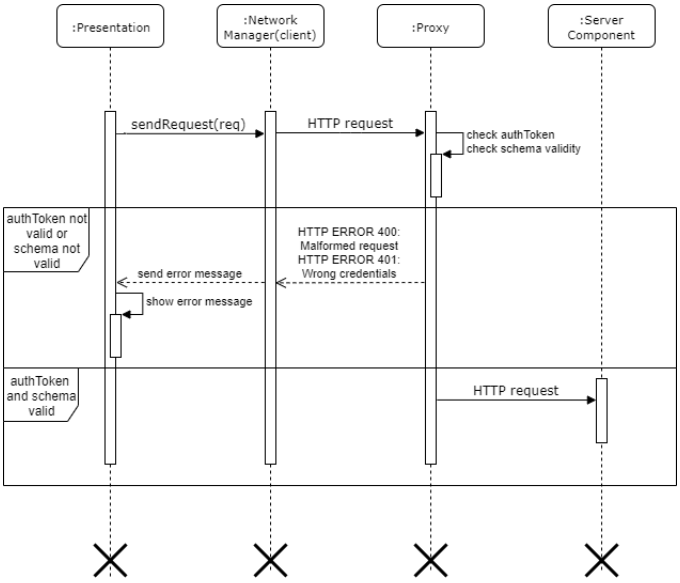

Figure 7: Proxy Runtime View

Login Runtime View

The following diagram describes the sequence of events when a *User* tries to login in *SafeStreets* application. The actors involved are the Presentation and the Network Manager on the client and the Network Manager, Authentication Manager and the Database Manager on the Server. When a *User* wants to login in his account, an HTTP request is sent to the application Server containing his credentials. Database Manager checks the validity of credentials by quering DB. If the query success then *User* can correctly be authenticated. The two errors that can occur are HTTP ERROR 400: Malformed Request and HTTP ERROR 401 Wrong Credentials. The first one occurs when request's format is invalid, such as having empty parameters. The second one occurs when credentials are wrong.

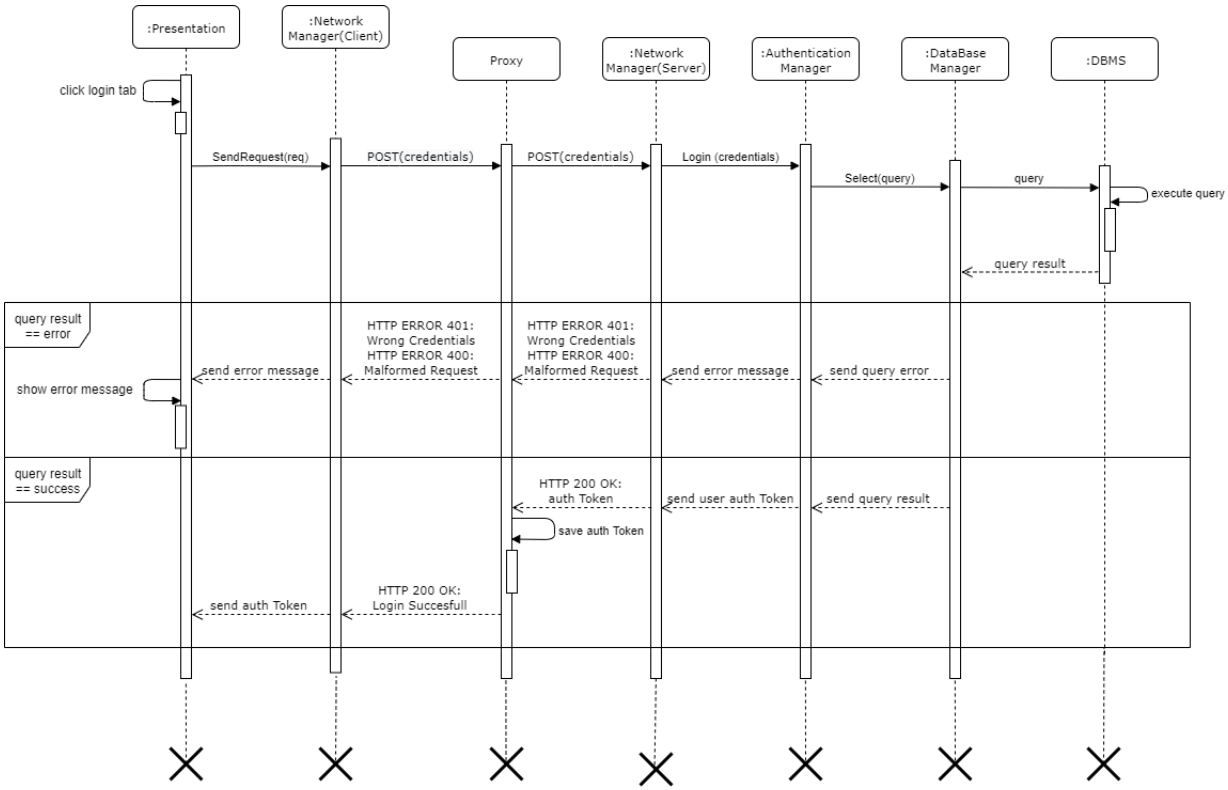

Figure 8: Login Runtime View

Registration Runtime View

The registration sequence shows the step followed for the registration of a new *User*. The two registration procedures for *Citizen* and *Authority* are similar, the only thing that change is the code used, for the first is needed the fiscal code for the second the Matricola number, to avoid a duplicate we show here the general procedure. After filling the form data are sended through the proxy at the main server, here data will be secured and stored in the database. Here 2 situations may occur, the *User* data aren't already present in the database so *User's* data will be stored and *System* can continue with the process sending client a success message or data inserted have been already used by another *User* and the client will receive an error message. This process will be repeated until success message. When *User's* data are correctly stored the *System* will send a confirmation email with a token in order to activate the account. Then *System* wait until the *User* clicks the confirm link in the mail received, clicking the link will send the token to server, again two situation may occur, the token is valid and the procedure terminate with the account activation or the token isn't valid and the *User* need to request another confirmation email.

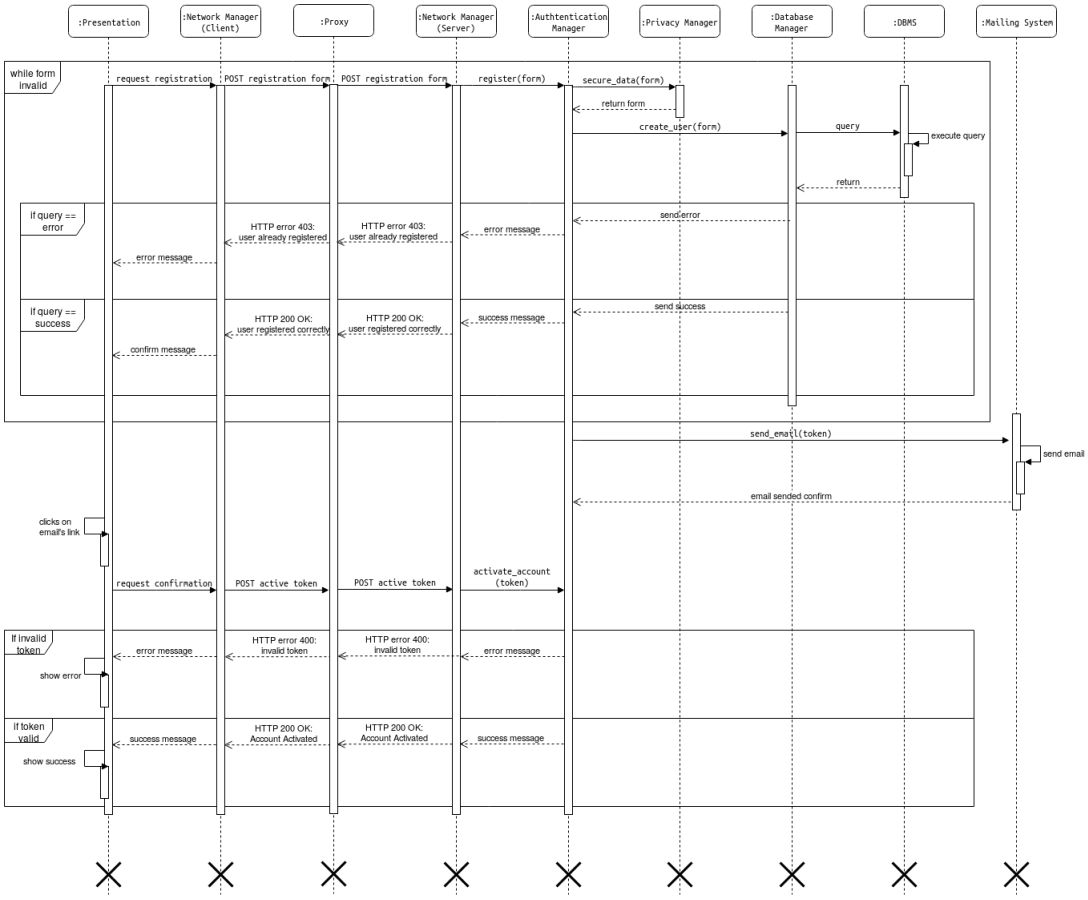

Figure 9: Registration Runtime View

Send Report Runtime View

The following diagram represents the sequence of actions executed on the *Citizen's* send report, first all the necessary data is gathered, the nex sequence will describe in a more detailed way how it's done, here is simplified for a better readability. Then data collected is passed at the Logic Manager, it will secure the data applying a digital firm through the Security Manager, then the violation is saved momentaneously in the cache and sent at the server. As described in the picture, Logic Manager save the violation in the cacheto avoid loss, if there is a malicious attack in the way to the server in order to modify the violation then the digital firm will be compromised, in this case the server will reject the violation and it will be lost, to avoid this inconvenience the client saves the report in the cache and will send it at every timetout until it recevies from the server the success message. In the other hand the server will save the report and send the success message if only if the digital firm is correct.

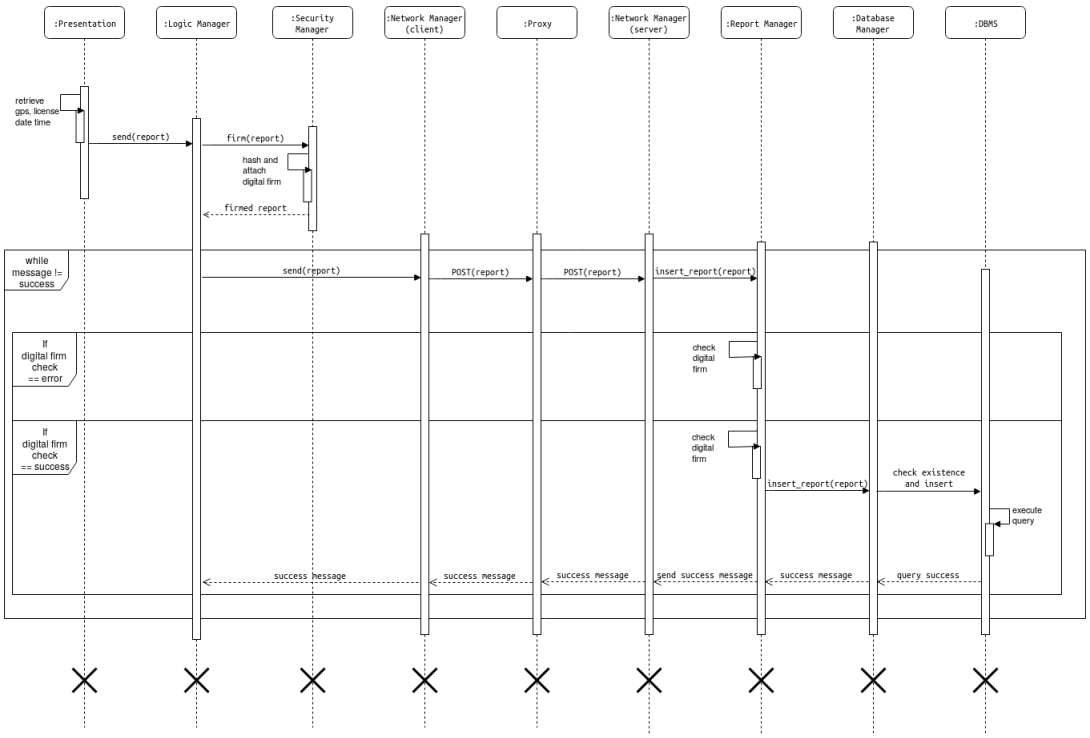

Figure 10: Send Report Runtime View

Collect External Data Runtime View

This is a "zoom" on the self call "retrieve gps, license, date time" that Presentation does in the previuos sequence, it's only a graphical semplification in aim to better understanding the process, separating this fase make more readable the previous image. This steps show how information are retrieved from external services, those information will be used to create the report by the Logic Manager.

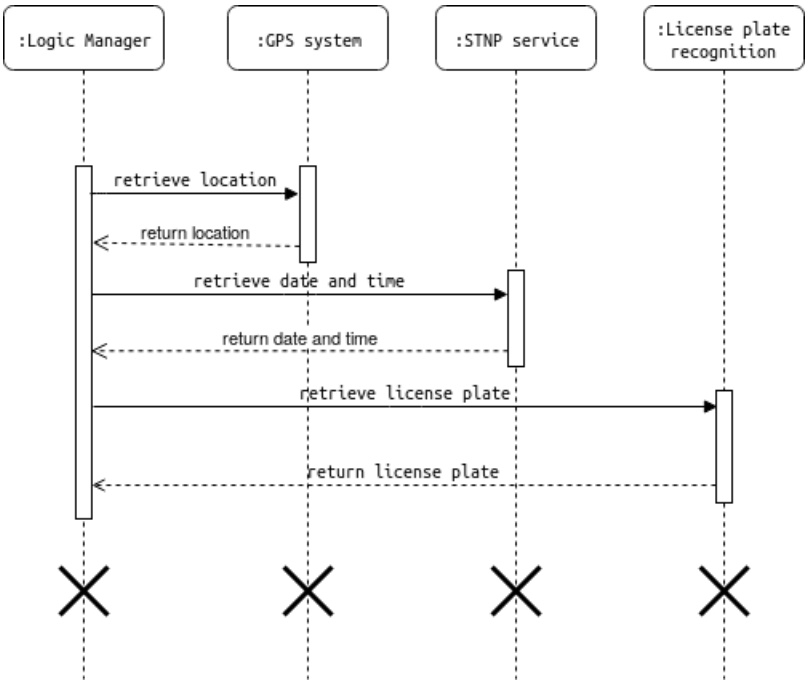

Figure 11: Collect External Data Runtime View

Retrieve Report Runtime View

The following diagram describes the sequence of events when an *Authority* tries to retrieve a violation in *SafeStreets* application. The actors involved are the Presentation, Security Manager and the Network Manager on the client and the Network Manager, Report Manager and the Database Manager on the Server. When an *Authority* clicks the retrive button an HTTP request is sent to the Application Server in order to query the DB and retrive the violation. Once the violation has been retrieved, Securith Manager checks violation's validity by computing the hash and matching it with the one from the digitally sign. If the two hash are the same then Security Manager sends the violation to *Authority*. When *Authority* clicks the "YES" or "NO" button a final HTTP request is sent to Application Server containing the response. It is important because in this way *System* can build Statistics. If, instead, the two hash are not the same then a malicious attacker is trying to alterate information so Security Manager discards the violation and waits for the timeout. In this way Report Manager can sends the violation again.

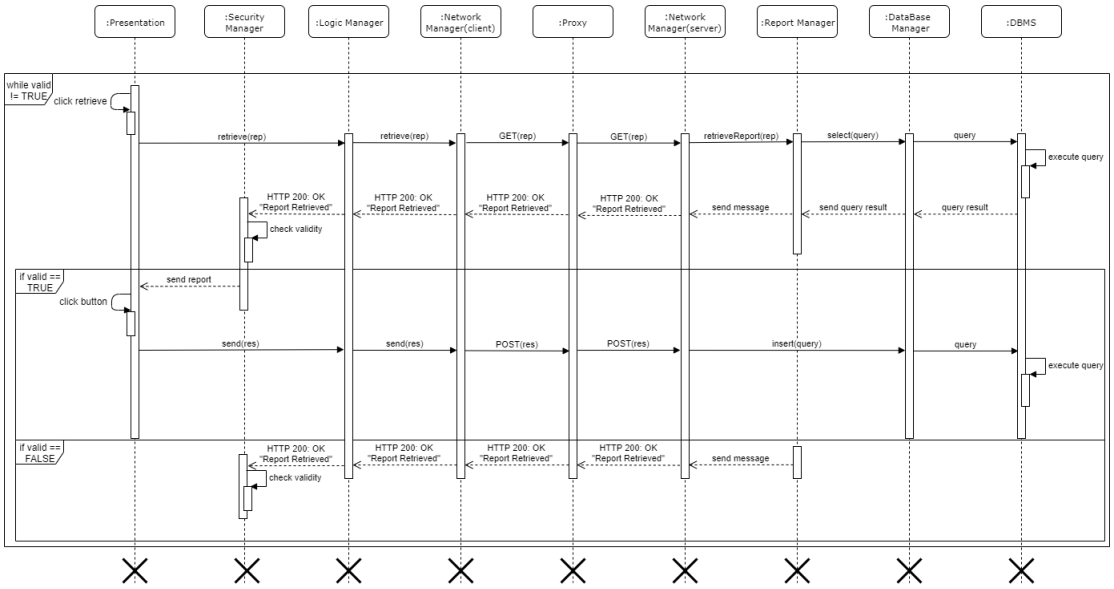

Figure 12: Retrieve Report Runtime View

Retrieve Statistics Runtime View

The following diagram describes the sequence of events when a *User* tries to retrieve statistics in *SafeStreets* application. The actors involved are the Presentation, GPS, Google Maps and the Network Manager on the client and the Network Manager, Statistics Manager and the Database Manager on the Server. When a *User* clicks the retrieve button two HTTP requests are sent, one to GPS Service and the other one to Google Maps Service, in order to know the exact position of the *User* and the area in which he is using the app. After that an other HTTP request is sent to Application Server in order to retrieve statistics by quering DB.

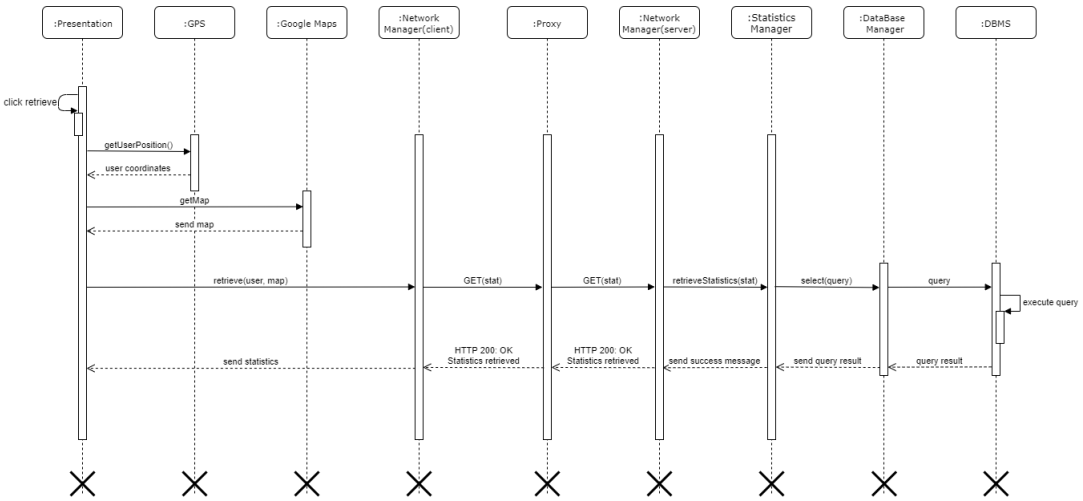

Figure 13: Retrieve Statistics Runtime View

2.5 Component Interfaces

Below are showed 3 picture representing the component interfaces diagram, the first is for client, the second for server and the third is the complete diagram with all the components. Again we divided the perspective of the image for the sake of readability.

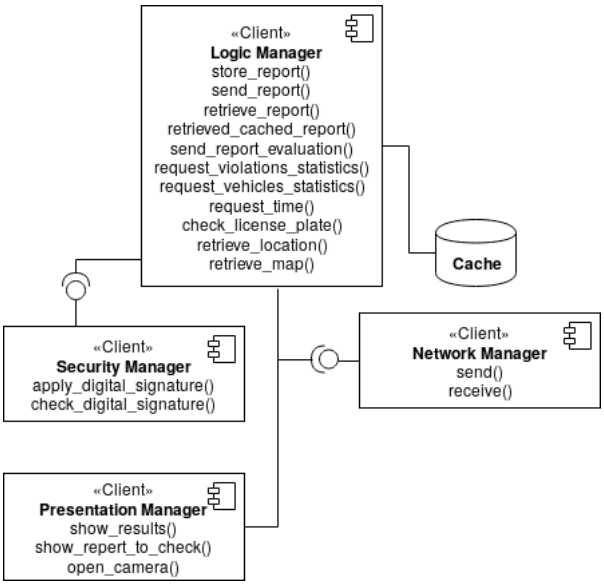

Figure 14: Client Component Interface Diagram

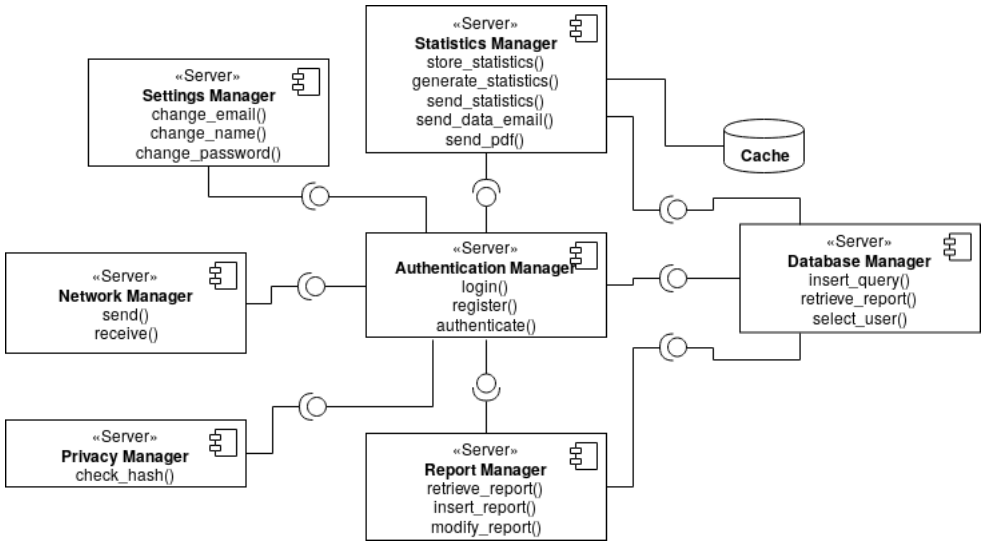

Figure 15: Server Component Interface Diagram

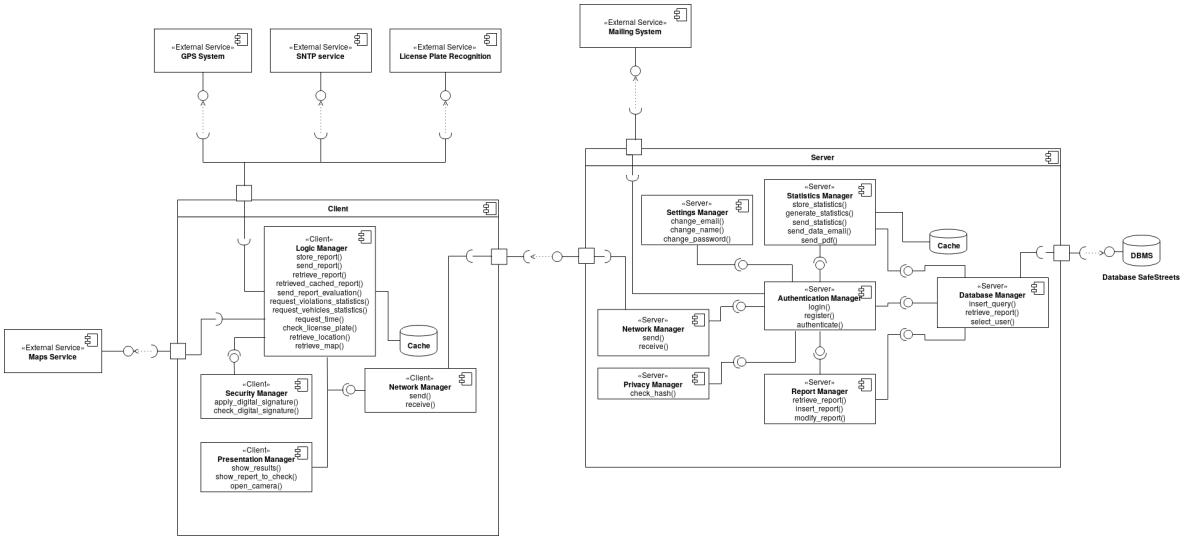

Figure 16: Component Interface Diagram

2.6

The architecture chosen for *SafeStreets* is multi-tier, namely a 3-tier architecture. This choice is weighted based on the requirements that the *System* must fulfill trying to optimize infrastructure costs at the same time. The tiers are divided into Client, Server and Database. The Client allows Users to interact with *SafeStreets*, takes care of the presentation, the digital signature, the exchange of messages with the server and the generation of the report. The logic in the client concerns only the temporary caching of the report and a possible resubmission in the case of a malicious attack on the report (both in the case of the *Citizen* to avoid loss in case of error and for the *Authority* that must evaluate it), collecting data from various external services and securing the report during trasmission to server.
From these premises and considering that all the other operations, such as the generation of the statistics or the update of these are made, necessarily, on the server side, the client is to be considered a thin client. The Server on the other hand will have a greater workload, the chosen architecture is NginX, which has a more flexible architectural design than the Apache process-driven one. Furthermore NginX is much more efficient than apache in presenting static content, this is in favor of *SafeStreets* since the statistics, which may seems dynamic, actually arrive to the user in the form of static content. This is because, as the *SafeStreets* logic was designed, the statistics are updated only thanks to the control of an *Authority*, this implies that the process of updating the statistics is not very fast, and that these can be generated at regular time intervals saving them in cache and sending them to the clients in static form. This brings great advantages to the application in terms of workload and service speed.
In fact, generating statistics at each client-side request would be unsustainable, so the server-side load is drastically reduced. In order to have greater *System* portability and scalability in case of required memory space growth, the best choice for the database is entrusting to an external service. This choice not only saves on costs, maintenance and the skills required but also guarantees a remarkable scalability in performance.

REST

Our *System* uses the Representational state transfer (REST) architectural style in order to achieve performance in component interactions, scalability, modifiability of components to meet changing needs, portability of components by moving program code with the data and reliability in the resistance to failure at the *System* level in the presence of failures within components, connectors, or data. HTTP is the communication protocol used underneath, so CRUD operations are implemented through GET, POST, DELETE and PUT primitives.

Pattern
In the following paragraph will be described the design patterns used in *SafeStreets*.

MVC:
Our *System* follows the MVC paradigm. The acronyms stands for Model View Controller

- The Model corresponds to the component containing all the application's data and integrity constraints. In our case the Model is described in the external database.
- The View corresponds to the presentation layer, the client's only interface with the *System*.
- The Controller is responsible for the logic of the application and in our case corresponds to *SafeStreets*'s Server.

Visitors Pattern:
Both Network Manager client and server side will adopt thi patter in order to route correctly the messages. *SafeStreets* router logic is embedded in The Network Manager, this choice aims to simplify the sending of a message to a component from another. It will be easy and flexible to add new type of message and, adding a new component, we only needs to add it on the network manager.

Adapter Pattern:
In order to separate interfaces from main logic we need to use this pattern. This separation is necessary to guarantee scalability and to simplify future improvements.

2.7 Other design decision

Firewall
Our *System* use two firewalls to create a DMZ. The first firewall must be configured to allow traffic destined to the DMZ only. The second firewall only allows traffic to the DMZ from the internal network. This setup is considered more secure since two devices would need to be compromised. Network firewalls operate by filtering the packets trying to pass through them.

DataBase
It was decided to use a relational database model for *SafeStreets*, as opposed to a non-relational model, because it allows to represent the intrinsic relationships present in the data needed to be stored. Also, many complex operations will be made on the database, including Joins and Updates, which are not well handled by a non-relational model. The database will use an ORM to simplify the interaction with Managers. Also the Database is acquired on premise as an external service.

ER Diagram
This is an oversimplified representation of the database schema. This is only a general view on the mandatory attributes needed focusing on the relevant attributes and tables. We omitted for simplicity other attribute less important, statistics for example may need some other attributes useful to perform searches more efficiently. In the ER schema there are four main entities: *Citizen* and *Authority* identified as *Users*, the two entity Vehicle and Location separated in two distinct table in order to make the process of statistics generation more efficient as possible and the entity Violation that is identified by date, time, location and license plate to guarantee unicity.

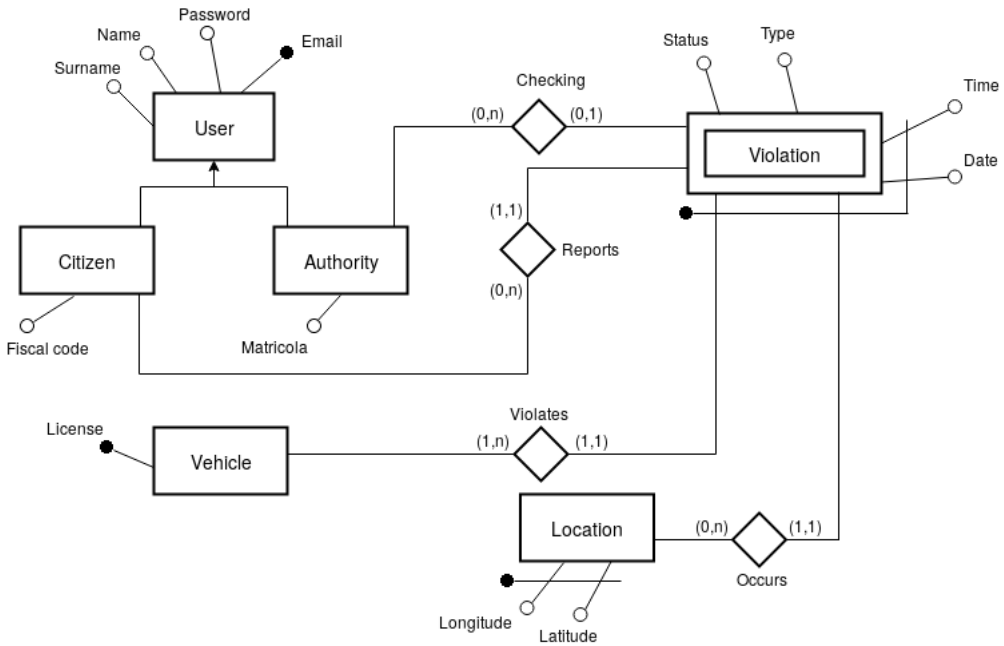

Figure 17: ER Diagram

3 User Interface Design

This is the application main flow either for the *Citizen* and *Authority*, they both start with login or registration and then they have access to the respective pages. *Citizen* will be able to visualize violations statistics, settings and send report page. The *Authority* can access same statistics as *Citizen* plus vehicles statistics and the page in which can retrieve report information and also a settings page.

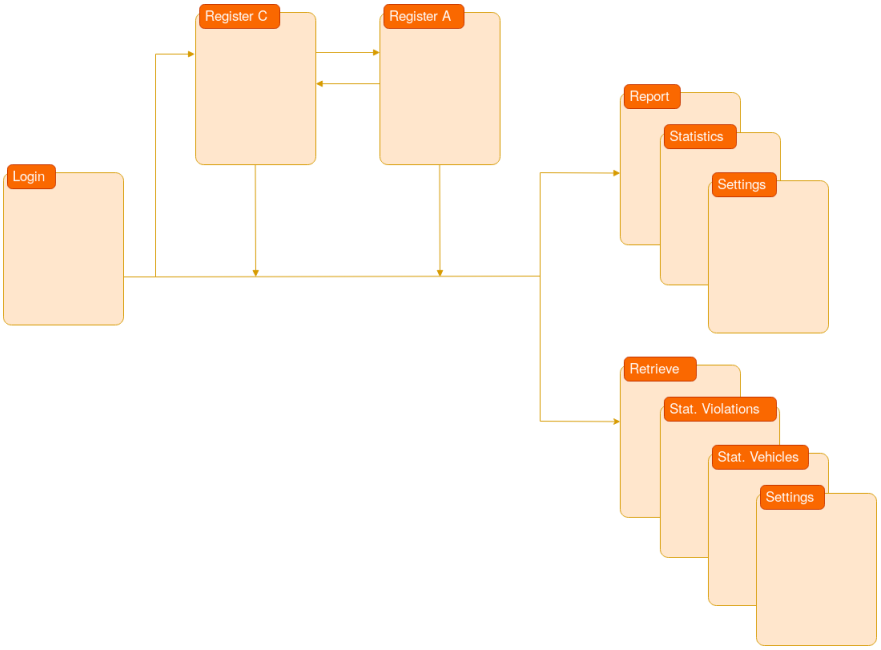

Figure 18: UI flow

4 Requirements Traceability

COMPONENT(DD)	REQUIREMENTS(RASD)
Authentication Manager	[**R1**]: Account can be created if and only if User provides unique email and password [**R2**]: The *System* allows Guest to create *Citizen* or *Authority* account
Report Manager	[**R4**]: The *Citizen* has to take the violation's photo with the application [**R5**]: The *System* allows *Citizen* to input some violation's data [**R6**]: The photo taken must be recognizable by the *System* [**R7**]: The *Citizen* has to be able to discard the photo taken [**R8**]: The *System* has to be able to attach the correct date, time and position to the report [**R9**]: The *Citizen* can't change date, time and position in the report [**R10**]: *Citizen* can change the license plate if it isn't recognised properly [**R11**]: *Citizen* has to be able to choose the correct type of violation [**R18**]: The violation retrieved can only be seen by the *Authority* that retrieves it
Statistics Manager	[**R12**]: Users can change the area of visualization [**R13**]: Users can change the date of visualization [**R15**]: *Authority* can search for a specific license plate [**R19**]: The *System* must update the statistics with the most recent data
Privacy Manager	[**R17**]: The *System* must use HTTPS to safely communicate
Settings Manager	[**R3**]: The *System* allow *User* to modify his settings.
DataBase Manager	[**R14**]: The *System* has to save the confirmed violation in the DB.
Security Manager	[**R16**]: Violations sent must be digitally signed [**R17**]: The *System* must use HTTPS to safely communicate

5 Implementation, Integration and Test Plan

5.1 Overview

For what concern the part of implementation we identify 3 macros subsystems:

- The client subsystem, containing the presentation.
- The server subsystem, containing components of Server and the interaction with DB.
- External Services, such as DB, Maps Service, STNP Service, License Plate Recognition Service.

Our implementation and testing will be incremental and will follow the bottom-up approach.

5.2 Implementation

The order of implementation we decided to follow is necessary in order to obtain consistency:

1. creation of the Database and its costraints.
2. server and client implementation
3. integration with external services

The implementation of the server and the client will be in parallel in order to increase the efficiency. The implementation of the server will follow this order:

1. Database manger
2. Network manager (to receive request from a fake client)
3. Authentication manager (to authenticate some test users)
4. Privacy manager
5. Settings manager
6. Report manager
7. Statistics manager

In particular DataBase Manager and Network Manager will be implemented first because they represents the access point to the Clients and the DB

For what concern the implementation of the client:

1. User Interface implementation (we will implement the same graphic for both iOs and Android) (view dell'MVC)
2. Network manager
3. Presentation manager
4. Security manager
5. Logic manager

5.3 Unit testing

During the whole phase of implementation component's unit tests will be performed in order to find any possible kind of bugs. Unit tests will be added incrementally as new code is written. In this way is possible to fix bugs with the lowest cost of repair in terms of effort and time. Using the bottom-up approach we need to simulate the components that are missing in order to test the component just added. To do so driver will be used to simulate calls to a certain component. Stubs, instead, will be used to simulate external services such as Maps Service or STNP service.

5.4 Integration testing

In this section we will show how components will be integrated. The arrows start from the component which 'uses' the other one. **Integration of internal components of the Application Server**

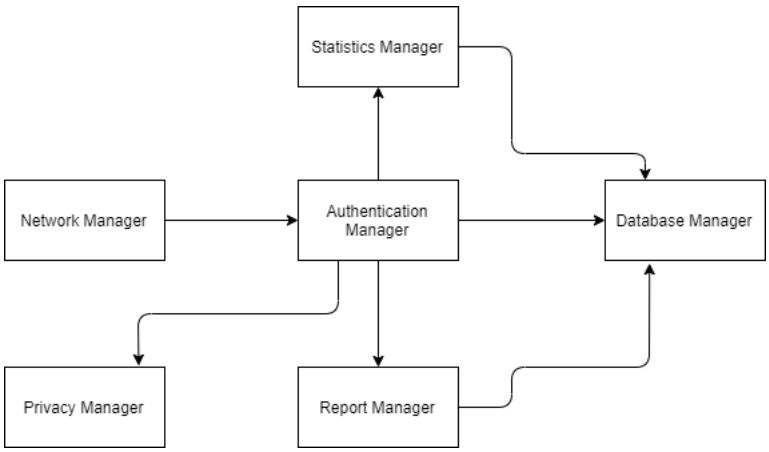

Figure 19: Server Integration

Integration of Client

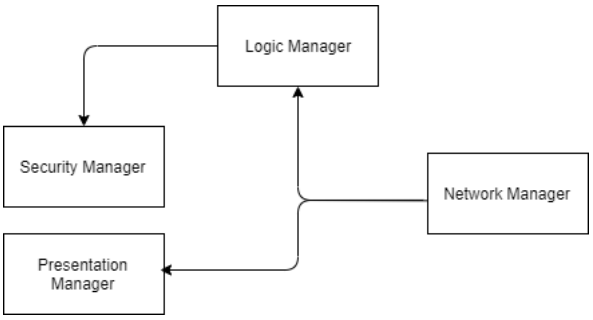

Figure 20: Client Integration

Integration of Client-Server

Figure 21: Client Server Integration

Integration of External Services

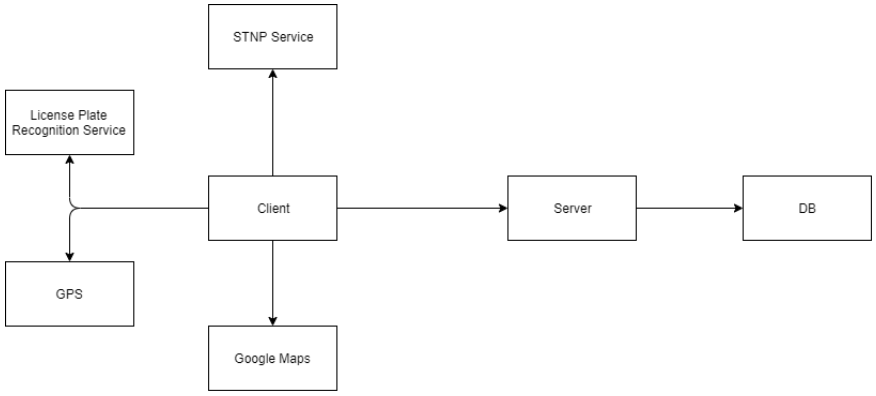

Figure 22: External Service Integration