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

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
GDPR	General Data Protection Regulation
FC	Fiscal Code
DB	Database

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

2 Architectural Design

2.1 Overview: High level components and their interaction

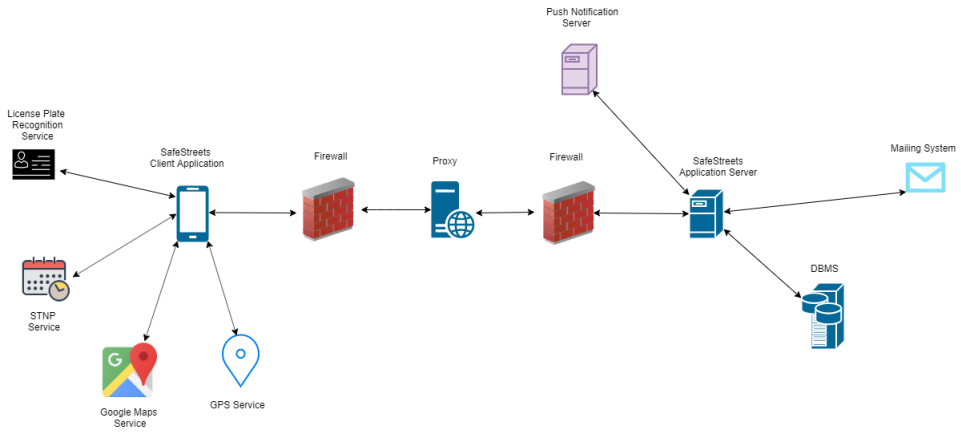

Figure 1: *System's* Overview

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. For notify the *Users* with some relevant informations *System* will use a notification System. Finally we also need some service in the client like GPS Service, Maps Service, SMTP 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 showing at an high level how the *System* is organized, as we can see the choice we made is simple, it's a basic client-server architecture. The client presents various components each one covers a different role: the Presentation Manager is the medium between the application and the *User*, Logic Manager is the component designed to catch all the application Logic, it orchestrate 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 some data. The Security Manager is 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 firm. The Network Manager handle the connection and is the component that communicates with server.

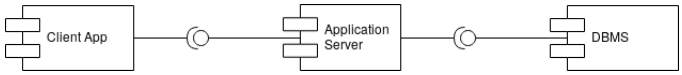

Figure 2: Component Diagram High Level View

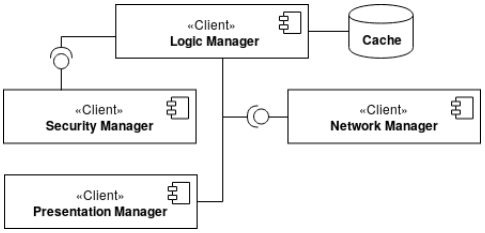

Figure 3: Client view

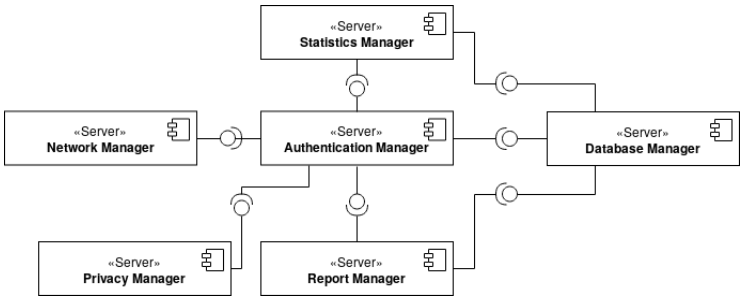

Figure 4: Server view

Component High Level View Here we go deeper in the Component Diagram, are decribed 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 with Push notification System for the server.s

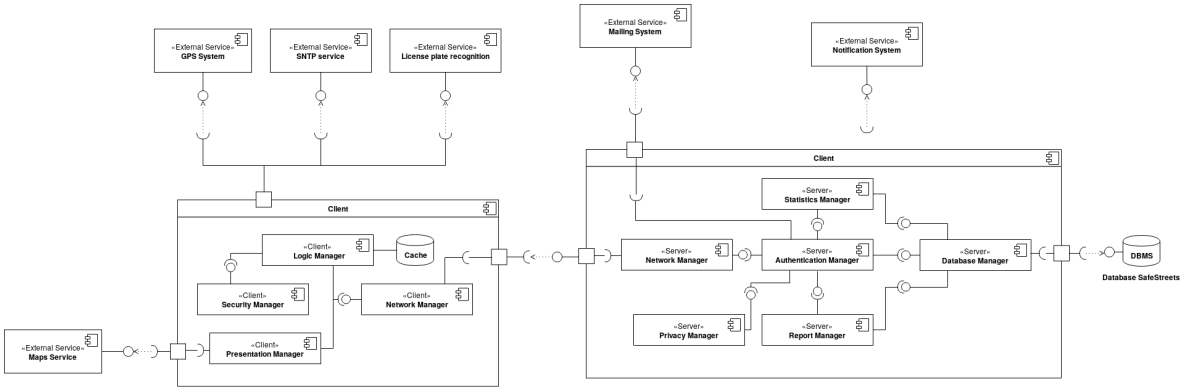

Figure 5: Component Diagram Detailed view

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 Business Logic, External interfaces. Below we will describe each component.

***SafeStreets* Client Application** This component is located on the *User's* device. Its modules are the Network Manager, the Presentation Component and Security Manager. The role of the first component is to dispatch all incoming and outgoing communications with the application server. The Presentation Component, instead, corresponds to the "View" in the MVCS Pattern.

***SafeStreets* Business Logic**
This component describes core logic of *SafeStreets* application. It is a stateless module that lies between the Client application and the central Database. Now we will describe each component:

- **Network Manager**
  It handles all incoming messages exchanged between Clients and Server. It does two foundamental things; it sends all incoming messages from client application to the correct handler in Business Logic and on the contrary it collects all the outgoing messages and sends to the corresponding Client Application.

- **Authentication Manager**
  The Authentication Manager exposes all methods related to the access to the platform. It handles the phase of *User* registration and login and in order to do this it has to continuosly interact with the database through the Data Storage Interface. It also checks all the costraints in order to garantee creation of correct account. Furthermore it sends email to all the *Users* that have been registered by accessing the Mailing *System*'s Interface.

- **Citizen Manager**
  The *Citizen* manager handles all functionalities related to a *Citizen* Account. In order to do this it has to continuosly interact with database through the Data Storage Interface. It also allows *Citizen* to update his setting.

- **Authority Manager**
  The *Authority* manager handles all functionalities related to an *Authority* Account. In order to do this it has to continuosly interact with database through the Data Storage Interface. It also allows *Authority* to update his setting.

- **Statistics Manager**
  The Statistics Manager handles the management of a Retrieve Statistics performed by *User*. Retrieving is possible by querying the central Database through the Data Storage Interface.

- **Privacy Manager**
  The Privacy Manager exposes methods to encrypt sensitive data. This operation is important in order to avoid data leaks and data alteration.

- **DataBase Manager**
  The DataBase Manager provides all methods to interact with the central Database such as data retrieval, storage and update.

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 Data Storage 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.

2.3 Deployement view

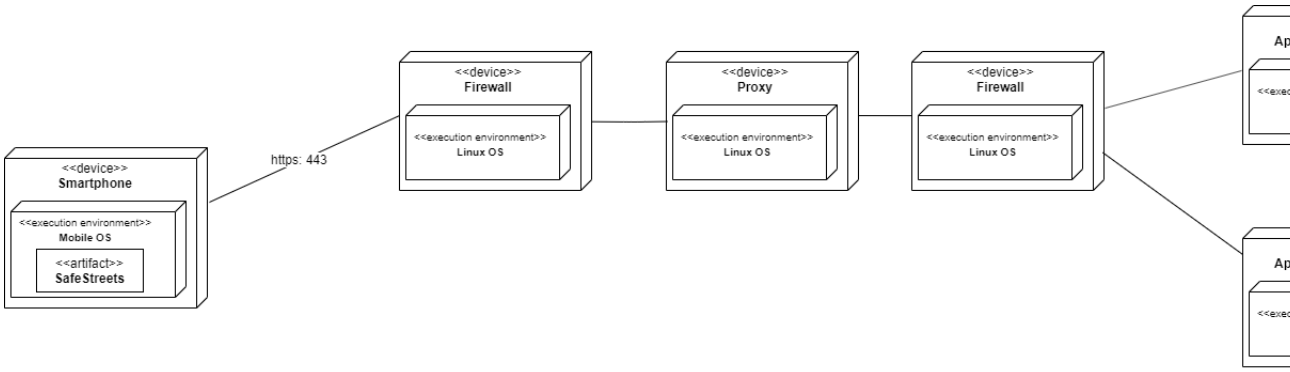

Figure 6: Deployment view

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

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.

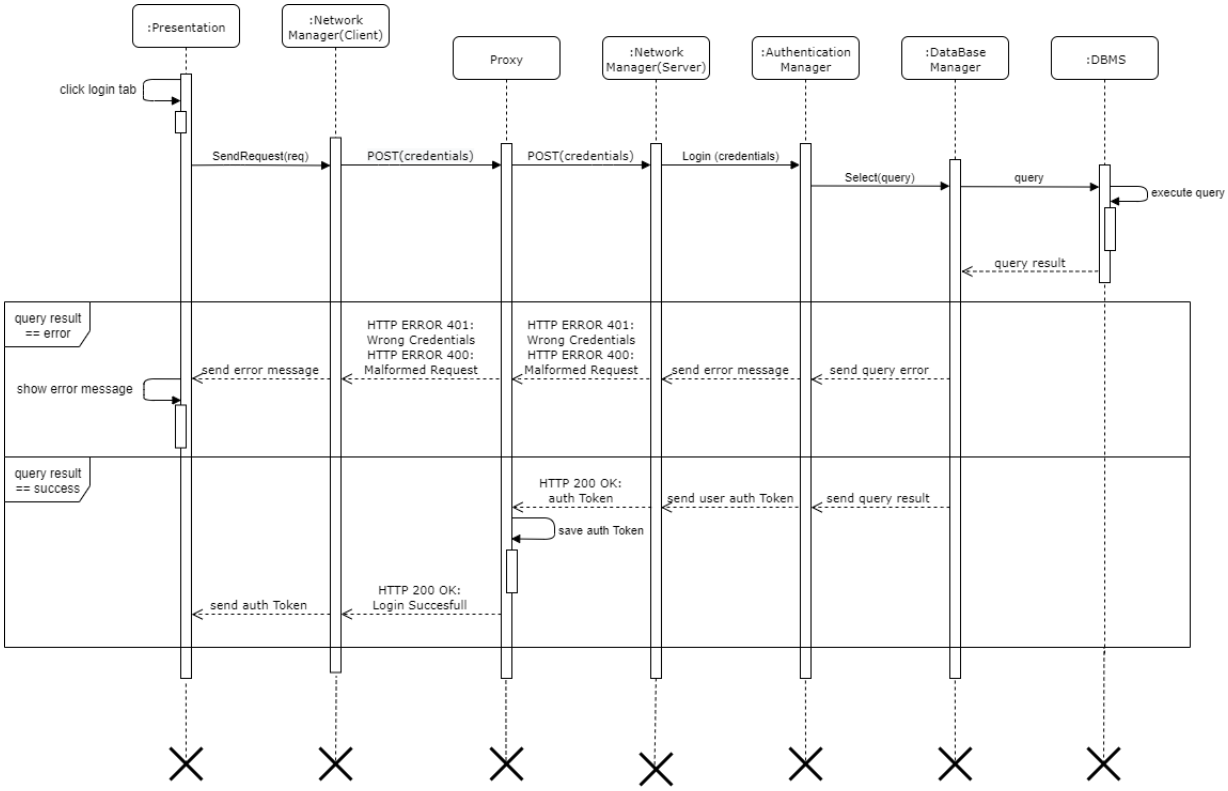

Figure 7: Login Runtime View

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.

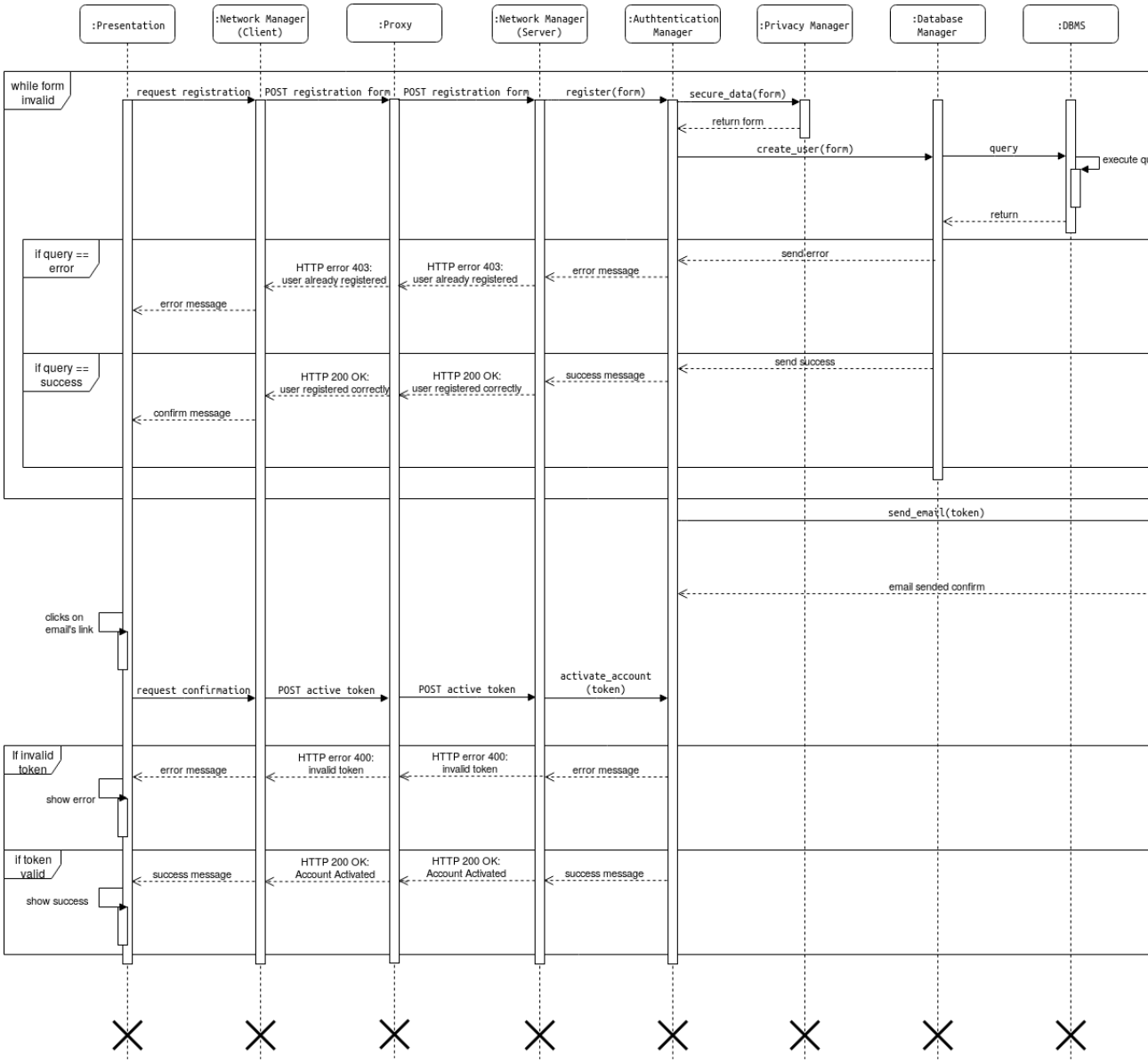

Figure 8: Registration Runtime View

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.

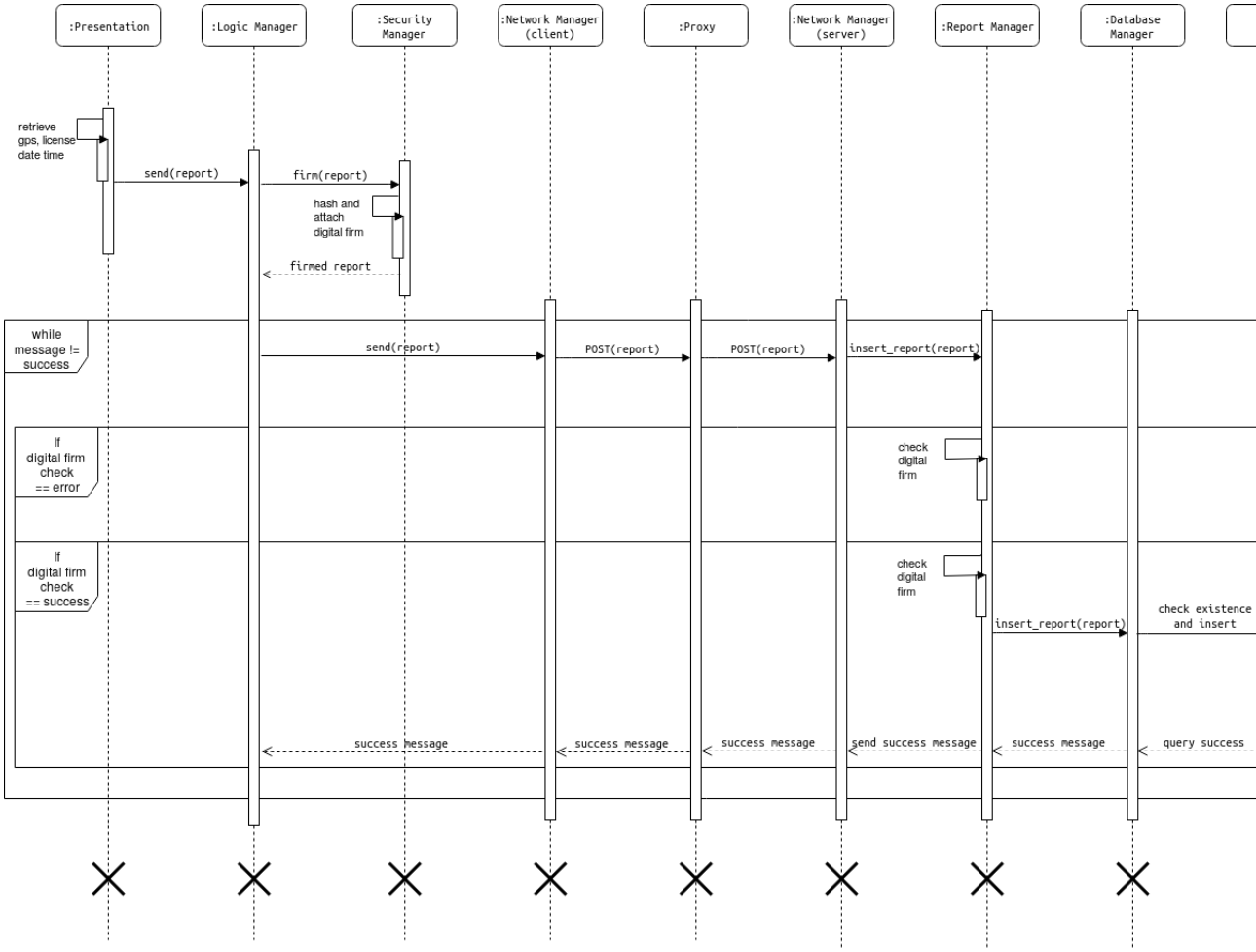

Figure 9: Send Report Runtime View

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.

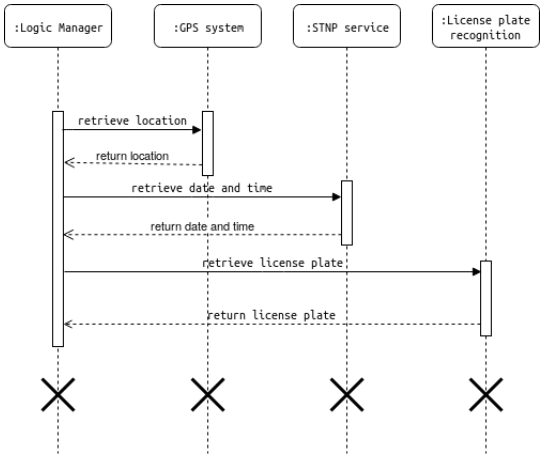

Figure 10: Collect External Data Runtime View

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.

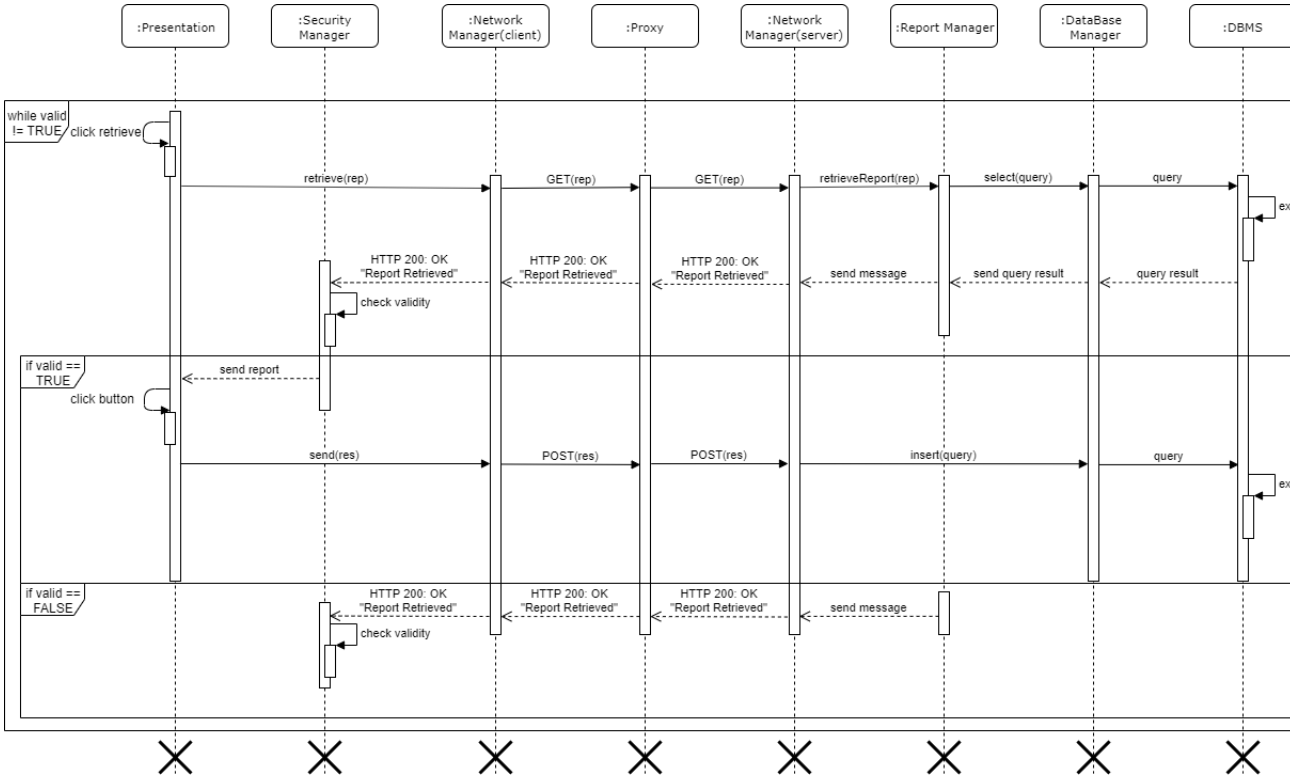

Figure 11: Retrieve Report Runtime View

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.

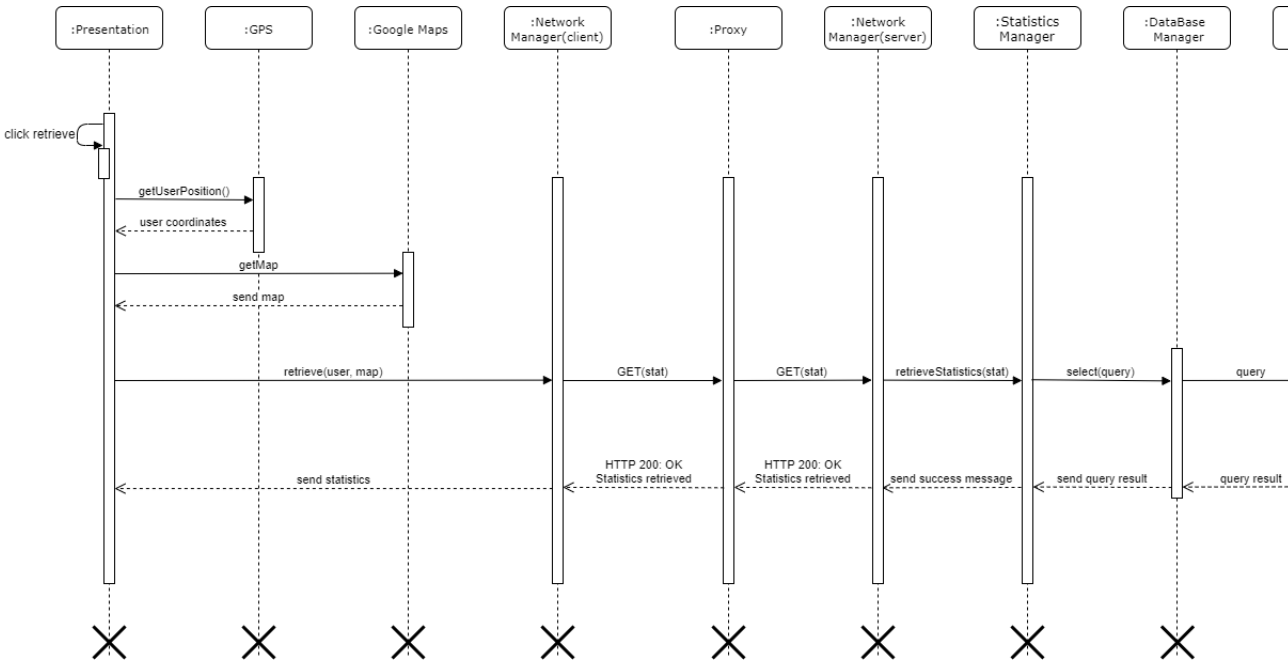

Figure 12: Retrieve Statistics Runtime View

2.5 Component Interfaces

2.6 Selected architectural styles and patterns

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.

MVC Our *System* follows the MVC paradigm. As shown in the picture The acronyms stands for Model View Controller

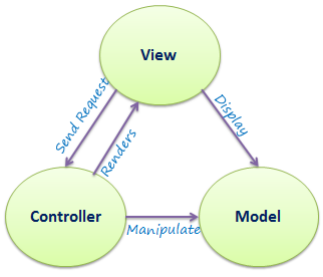

Figure 13: MVC Architecture

- 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* Business Logic.

2.7 Other design decision

Firewall

ER Diagram This is an oversimplified representation of the database schemas. This is only a general view

on the mandatory attributes needed the focus is 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 entity: *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.

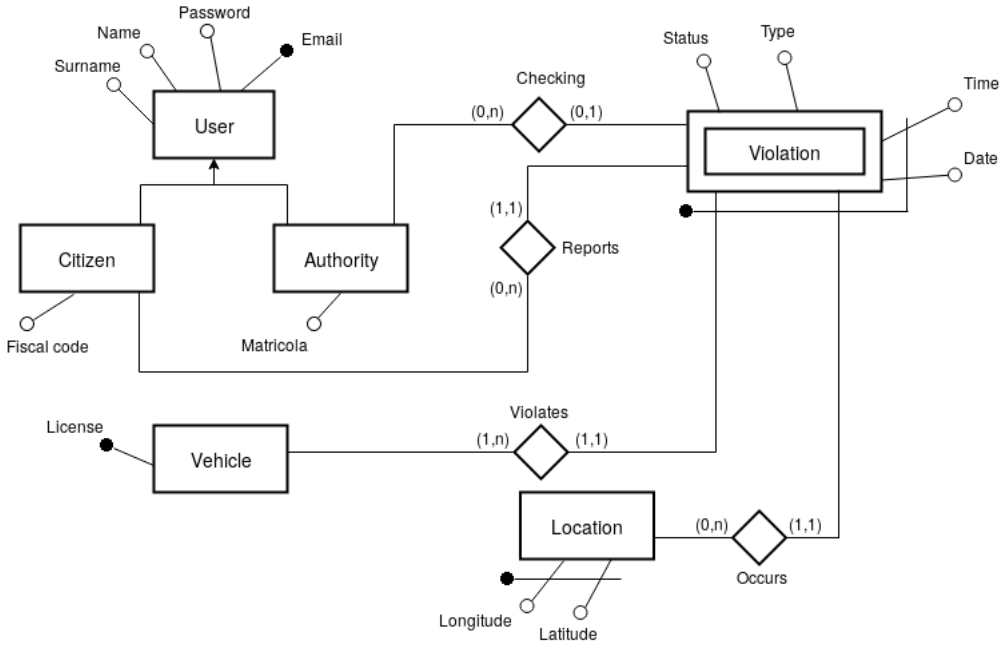

Figure 14: ER Diagram

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.

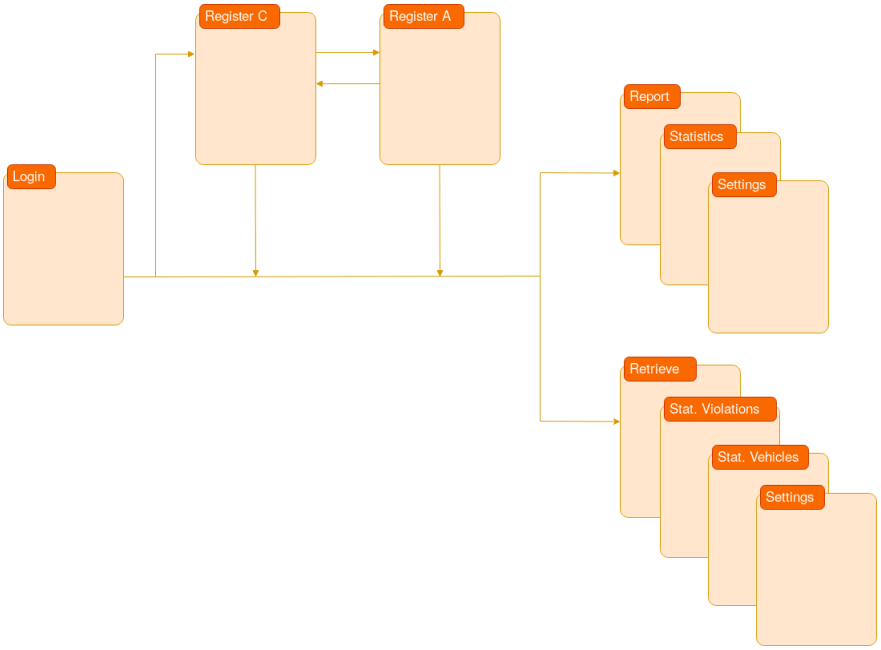

Figure 15: UX flow

4 Requirements Traceability

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