POLITECNICO DI MILANO

SOFTWARE ENGINEERING II

SafeStreets - Design Document

Version 1.0

Authors
Iacopo Marri
Manuel Salamino
Steven Alexander Salazar Molina

December 15, 2019

Contents

1 Introduction

1.1 Purpose

This document provide an overview of *SafeStreets* application, explaining how to satisfy the project requirements stated in the RASD.
It's mainly intended for developers and testers and they can find a functional description of the components of the System, their interactions and their interfaces, and also how they will be implemented.

Finally, all the requirements expressed in the RASD will be dealt with what is expose in this one, describing how the components presented can satisfy them.

1.2 Scope

As explained in the RASD, *SafeStreets* is an application that was created with the intention of monitoring the compliance with traffic regulations.
Its main goal is to allow Users to notify traffic violations.
Then the application must collect the reports sent by the Users and allows Authorities to verify if each violation is sanctionable and, if it is, provide them the possibility of generate Traffic Ticket.

Moreover, by crossing its collected data with the Municipalities ones, is also possible to provide to Users and Authorities statistics about the effectiveness of *SafeSteets* and suggestions about how to avoid recurrent accidents.

The application provides Users a way to send data about a violation (most common are parking violations ones, e.g. vehicles parked in reserved places), like the kind of violation, pictures of the involved vehicles, and the date and the position in which the violation occurred. The User can also specify the license plate of the vehicle and the name of the street, but in case he/she doesn't, the application is equipped with algorithms capable to obtain those and other metadata by pictures and position.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Guest**: a person who has not logged in or registered yet and cannot use the functionalities of the System.
- **User**: a person that uses the application to send notifications of traffic violations.
- **Authority**: a municipality worker that is able to create traffic tickets depending on the violation that a person has committed.

1.3.2 Acronyms

- DD: Design Document
- RASD: Requirement Analysis and Specification Document
- DBMS: Data Base Management System
- UI: User Interface
- API: Application User Interface
- OS: Operating System
- REST: REpresentational State Transfer
- HTTP: HyperText Transfer Protocol
- UX: User eXperience

1.3.3 Abbreviations

- [**R.i**]: i-th functional requirement.

1.4 Reference Documents

- RASD document
- Project assignment specifications

1.5 Document Structure

Architectural Design: shows the main components of the System and their relationships. This section also focuses on design choices and architectural styles, patterns and paradigms.

User Interface Design: following what has already been included in the RASD, this section defines the UX by means of view flow modeling.

Requirements Traceability: shows how the requirements in the RASD are satisfied by the design choices of the DD.

Implementation, Integration and Test plan: explain the order in which the implementation and integration of components will occur and how the integration will be tested.

2 Architectural Design

2.1 Overview

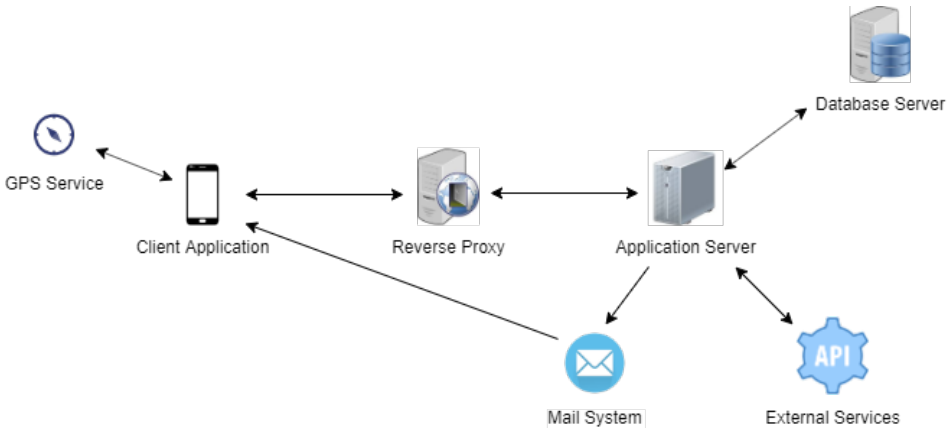

Figure 1: System overview

The above figure shows an high level overview of the System's architecture. We can immediately identify that is divided in two parts: the Client Application part and the Application Server part.
The former shows the service used by the Client Application and its communication with the Application Server.
The other one indicates how the System works with the external functionalities in order to accomplish the required Goals.
Further details on the System components and their interactions will be explained in detail in the following sections.

2.2 Component view

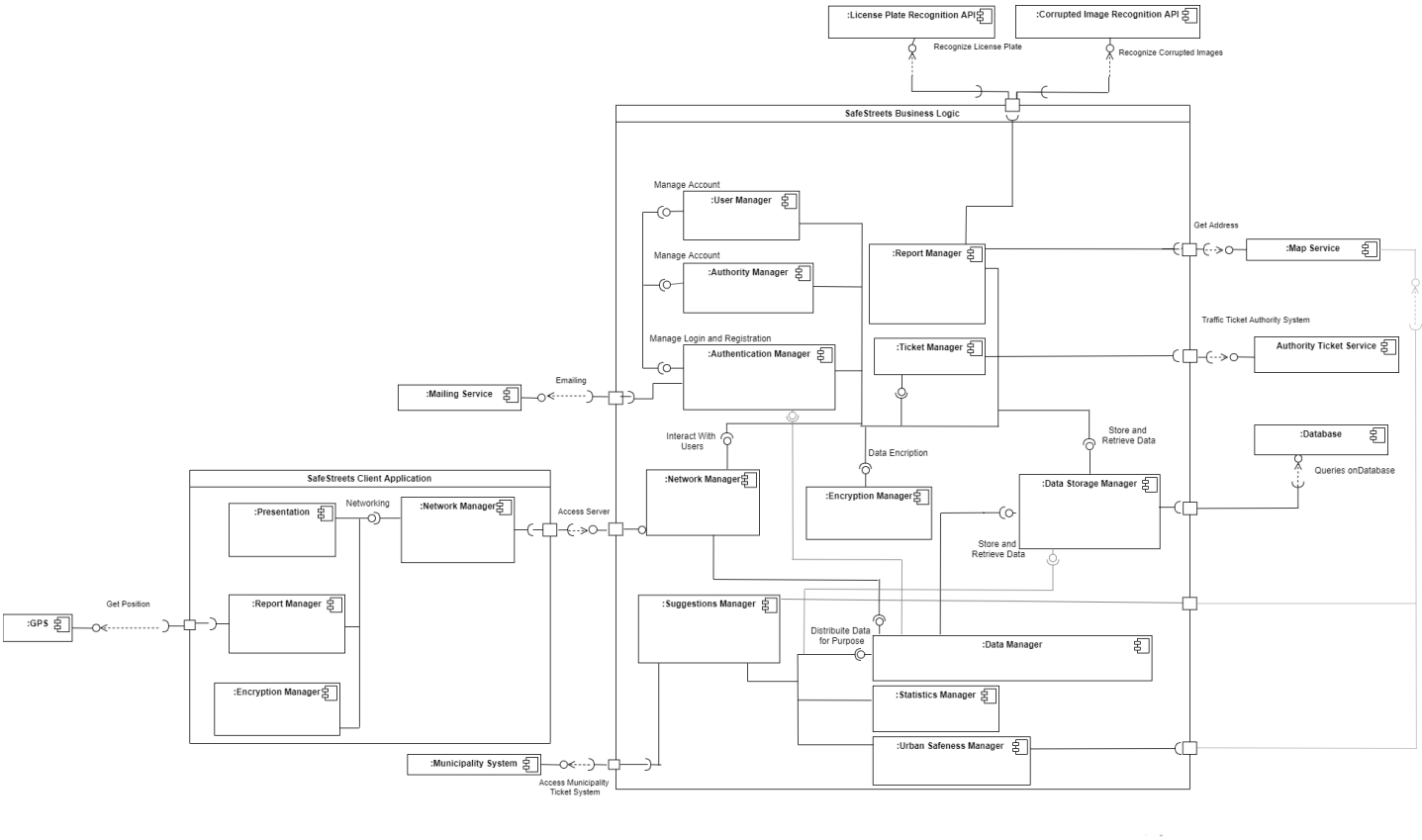

Figure 2: Component Diagram

In this section, an UML diagram is used to show the internal structure of the *SafeStreets* system.
It is divided into one application client layer and one Business Logic layers, which reflect the physical structure showed in the Overview. The client layer communicates with the Business layer for requesting the services and accomplishing its scopes. Business Logic Layer communicates with the Database, from which it stores and retrieves data.
The relationships and interfaces between the components are represented in our diagram through assembly connectors and delegation connectors (externally). Let's now take into account each single component:

SafeStreets Business Logic
This component is delegated to solve all the tasks relative to the processing of new reports, the managing of reports in general, and to provide all the functionalities about Traffic Ticket generation, Statistics, Suggestions and

Urban Safeness.
Furthermore, also the responsibilities of registration and authentication tasks lie here.

- **Authentication Manager**
  This component is in charge of managing the registration tasks, so it will check the data of the users, add them to the Database, send confirmation emails. It is also up to this component the task of logging users in, giving them the right authorizations.

- **Network Manager**
  Every communication among the Server and the Clients go through this component; it routes incoming messages toward the right component, and sends outgoing messages at the recipient application client.

- **User Manager**
  It handles the functionalities of User accounts. Through the presentation level it provides the right interface to the User. Furthermore, this component handles the access and modification to the User's personal data, username and password.

- **Authority Manager**
  It handles the functionalities of Authority accounts. Through the presentation level it provides the right interface to the Authority. Furthermore, this component handles the access and modification to the Authority's personal data and password.

- **Report Manager**
  This component takes care of all the aspects of the reports management. It takes incoming reports from Users, and check if they correspond to already notified violations. It uses external service API to check if the received picture has been corrupted, and API to recognize the license plate from the picture. An external Map Service is used to obtain the address from the position of the report. This component handles the modification or deletion operations of the Users.
  Report Manager is also in charge of computing the trustness of each report, and deciding if to discard it or no, and in case, it must store it correctly in the Database.
  Operations of viewing, accepting or rejecting a report, carried out by Authorities, are also under the responsibility of the Report Manager. Report Manager also ensures that no more that one Authority at a time can access a report.

- **Ticket Manager**
  After that a report has been accepted by an authority, the Application Server generates a traffic ticket related to that violation. All that

concerns the managing of traffic tickets is handled by the Ticket Manager. It allows to generate tickets, by accessing the external Authority Ticket Service.

- **Data Store Manager**
  This component realizes a intermediate layer between the Database and each component that accesses it; it provides to these components an easier way to store and retrieve data from the Database.
  Data Store Manager create queries and directly interacts with the Database.
  It is linked to every other component who needs to access the Database.

- **Encryption Manager**
  Components who manage important data interfaces with the Encryption Manager, who provides encryption methods.

- **Data Manager**
  The Data Manager stands between the Suggestions, Statistics and Urban Safeness managers, and the rest of the Logic Layer; these three components only use stored data to provide information based on the kind of request from the client. Data Manager gets the request, and chooses to which of these components turn the request to, depending on the type of information requested.
  But the main scope of this manager is to keep updated the three components under it; it implements an Observer Pattern which receives a pull notification every time reports or tickets are added to the Database, then it brought up to date the components information that are interested in the new data.

- **Suggestion Manager**
  This component accesses the external Municipality data and crosses them with the data of SafeStreets. In order to provide physical intervention at the urban mobility this component also exploits the external Map Service. It stores the suggestions in the Database. Through the Data Manager, the Suggestion Manager keeps its suggestions updated to every new incoming data.
  Furthermore, through the Presentation layer it allows Authorities to view the suggestions list.

- **Statistics Manager**
  Handles the statistics information that Users and Authorities view in the Statistic Section of the client application.
  It accesses the Database, retrieves information, computes statistics, and keep them up to date thanks to the Data Manager.

- **Urban Safeness Manager**
  It elaborates data from the Database to identifies the most safe or unsafe areas, and then present them to Users and Authorities by using the external Map Service.
  As other components, it leans to the Data Manager to keep its information updated to every new data.

External Interfaces
Both Client Application and Business Logic Layer use functions form external component with which they communicate. We are introducing these components with some more particulars:

- **GPS**
  Global Positioning System, is used by the Client Application to send, along with a report notification, the global position where the violation occurred.
  Obviously the device on which SafeStreets is running has to have a built-in GPS module. GPS position is mandatory in a report notification.

- **Map Service**
  It's used by SafeStreets to retrieve the address from the GPS position, or to graphically show on a map some information requested by Users or Authorities.

- **Mailing Service**
  Authentication Manager uses this service to send confirmation e-mail to new registered Users, and to carry on any other via mail needed communication.

- **Database**
  Where SafeStreets store all its data. It is external to the Business Logic Layer, and the Data Store Manager is in charge of making this layer communicate with the DBMS running on the Database.

- **Authority Ticket Service**
  Is the hardware and software system owned by Authorities through which Authorities generates traffic tickets. SafeStreets accesses this system and its functionalities to correctly generate tickets.

- **Municipality System**
  For providing suggestions, SafeStreets need to cross its own data with urban data owned and provided by the municipality.
  This data are obtained by making Suggestions Manager access the Municipality System that contains the data.

- **License Plate Recognition API**
  Report Manager has to add metadata to the received report notification. In order to get the License Plate of the vehicle involved in a violation, SafeStreets must use an algorithm that recognize it from the picture received.
  This algorithm is provided from an external API.

- **Corrupted Image Recognition API**
  Corrupted data that is received must be discarded. SafeStreets (and in particular, the Report Manager) uses ad external API service that implements an algorithm to identify modifications carried out on a picture.

2.3 Deployment view

The distribution of components capturing the topology of the system is illustrated below by using a deployment diagram.
The system is structured in a multitier architecture.

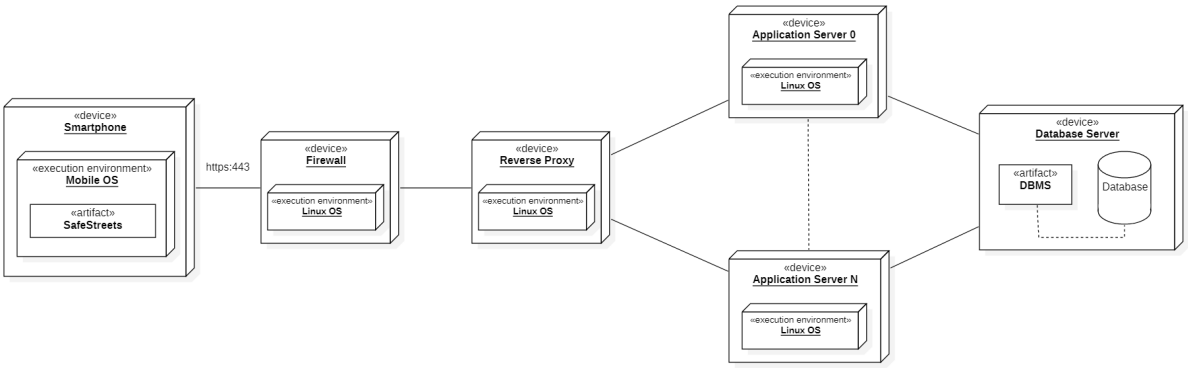

Figure 3: Deployment Diagram

Mobile Device
This node represents the mobile device of the User or Authority, both can access to the functionalities of SafeStreets by using their own device.

Firewall
This component allows to protect the Reverse Proxy by filtering the packets received from *Internet*.

Reverse Proxy
We chose to deploy a reverse proxy in order to increase parallelism and

scalability. Moreover, the reverse proxy is in charge of the load balancing since it is the component receiving all the request from internet (filtered by the Firewall). Another reason for the deployment of the Reverse Proxies is because it increases security and anonymity by protecting the identity of our backend servers.

Application Server
This component contains all the logic used to accept reports from Users, allows the corresponding Authorities to view them, computes statistics by accessing to local and public information about traffic violations, generates suggestions and allows to visualize safe/unsafe areas. In order to balance the workload this component may be replicated.

Database Server
This part of the System is equipped with a relational DBMS, it allows to store and retrieve all the data needed by the Application Servers.

2.4 Runtime view

2.4.1 Proxy Runtime View

The sequence diagram in *Figure 3* illustrates the interaction between the User, the Reverse Proxy and a component of the Application Server. Everytime a User sends a request through the client mobile application, the Reverse Proxy verifies if the request follows the proper scheme that defines a SafeStreet's client request, if the verification fails it returns an ERROR message to the client, otherwise it selects the correct server component to forward it. The Reverse Proxy could have been used to perform a verification of the credentials of the User but it was preferred to save a token in the Database and perform a query every time through the Data Storage Manager. In all the following diagrams the interaction between the User and the Reverse proxy won't be included for readability purposes, but it should be kept in mind that all the external requests will pass through the Reverse Proxy and only the ones with a correct schema will be forwarded as HTTP requests.

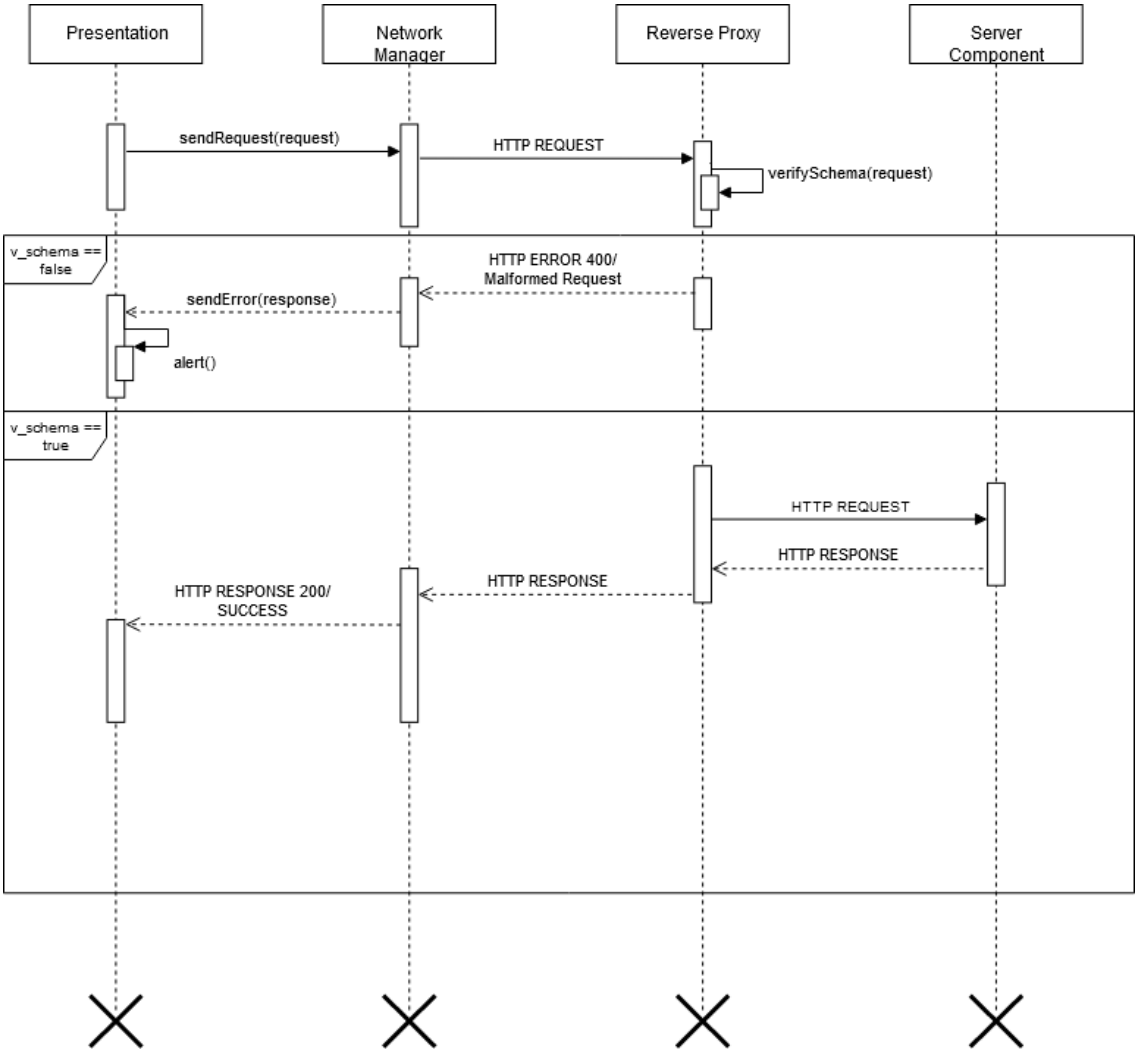

Figure 4: Proxy Runtime View

2.4.2 Registration Runtime View

A Guest to use *Safestreets* functionalities need to be authenticated, so he/she must compile a registration form.
The Client Application provide to the Guest a form that he/she must fills with the necessary information.
The information are serialized and then sent to the Application Server through an HTTP POST request. The request passes through the Network Manager of the Client and of the Server, but these steps are implicit and not shown in the sequence diagram also for readability reasons.
The Authentication Manager handles the request, verifies if the information are correct and complete, then, through the Data Storage, invokes a Query

and creates a new entry in the User table on the Database.
At this point the User Account has been created but it has to be confirmed.
An email with a confirmation link is sent by the external Mailing Service.
When the Guest clicks on the provided link, a Query on the database is executed and the new credentials become active.

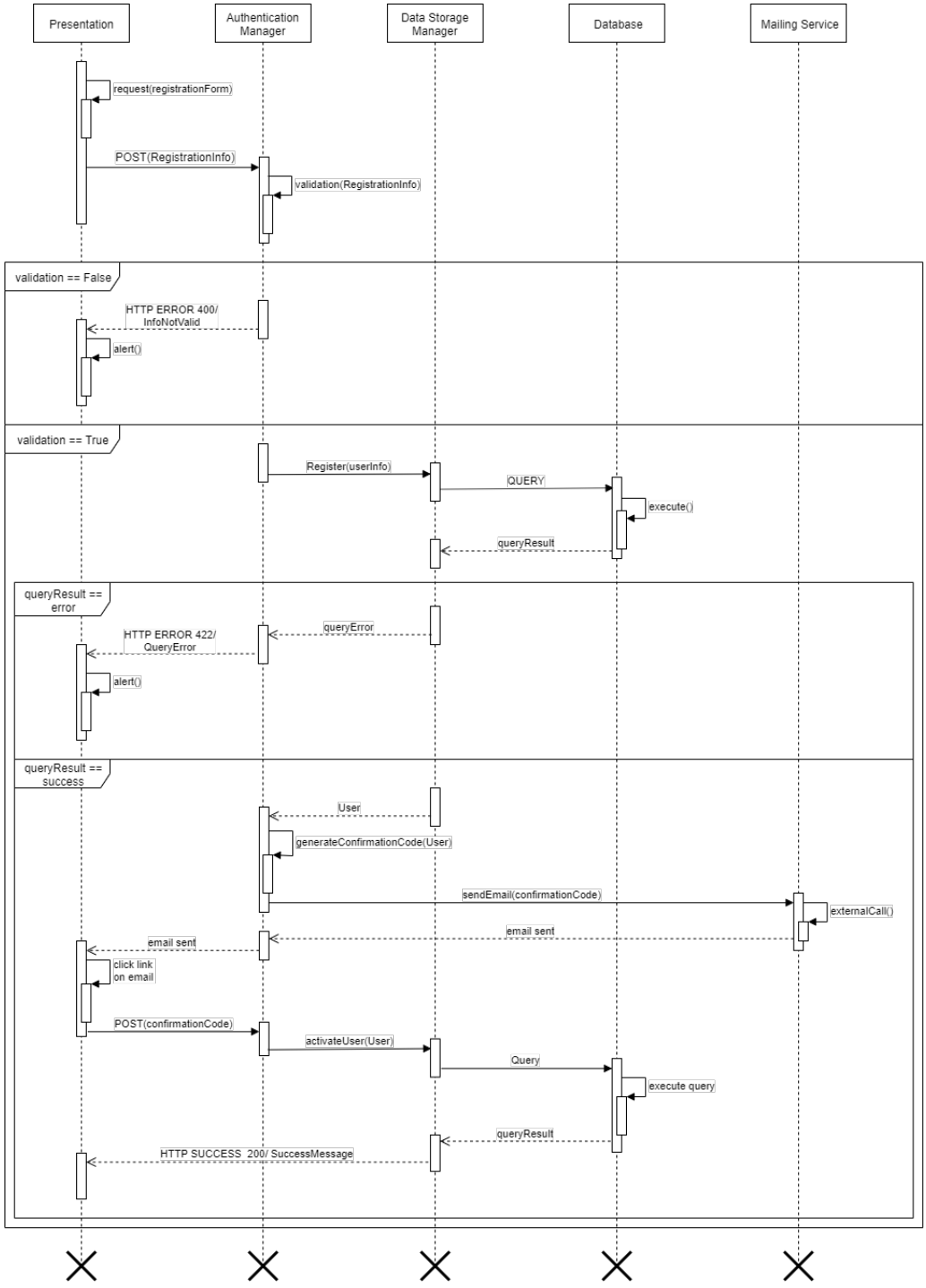

Figure 5: Registration of a User Runtime View

2.4.3 Login Runtime View

The following sequence diagram shows the interaction between the User when he/she tries to login.
As a first step, the User (through the mobile application) submits an HTTP POST request containing his/her credentials. Then like the first step in the Registration sequence diagram, the Authentication Manager receives the request and checks if the information given are correct (in terms of data types) and complete.
If the validation fails, the Authentication Manager replies to the User with an ERROR response allowing the mobile application to show an alert.
Otherwise, if the validation succeeds, the Authentication Manager, through the Data Storage Manager, performs a query to the Database. Then, if the number of tuples received is not equal to one, it returns a error message allowing the user to know that the username or password were incorrect, if the number of tuples is equal to one then the Authentication Manager generates a token that will allow the Server components to know that the user is logged in, this token is saved in the Database in order to validate it every time a request arrives.

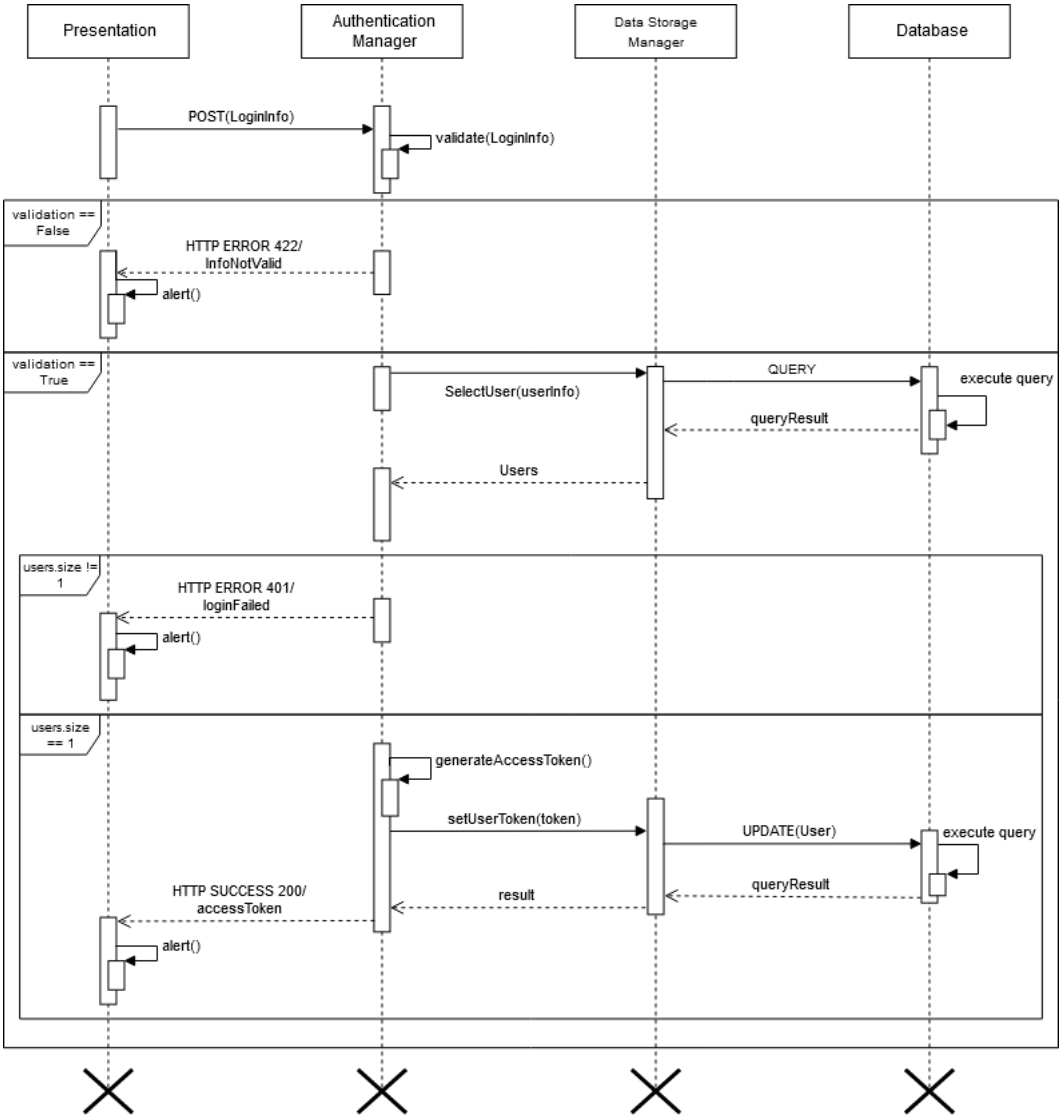

Figure 6: Login of a User Runtime View

2.4.4 Send Report Runtime View

The following sequence diagram describes the operations done by the components of the System to create and store a Report sent by a User.
The first step the System does is to check if the one who send a Report has the necessary privileges to send it (it need to be registered as User). After that, the Report Manager will receive all the submitted data and, after a consistency check on them is done at client level, it will add more data using external services. These services add data about the License Plate of the vehicle, the address of where the violation has occurred and the validity of

the submitted photo. If some of these data has already been inserted by the User who sent the Report, the System computes its results anyway and checks if are the same, if not then the System will add some notes to the Report in which it says to Authority that some data may not be correct at all.

The appointed services are:

- *Map Service*, to extract the address from the position
- *License Plate Recognizer API*, to read the License Plate of the vehicle from the provided photos
- *Corrupted Image Recognition API*, to be sure that the received images have not been modified

Finally the complete Report is add to the Database and a confirmation of success is sent to the client.

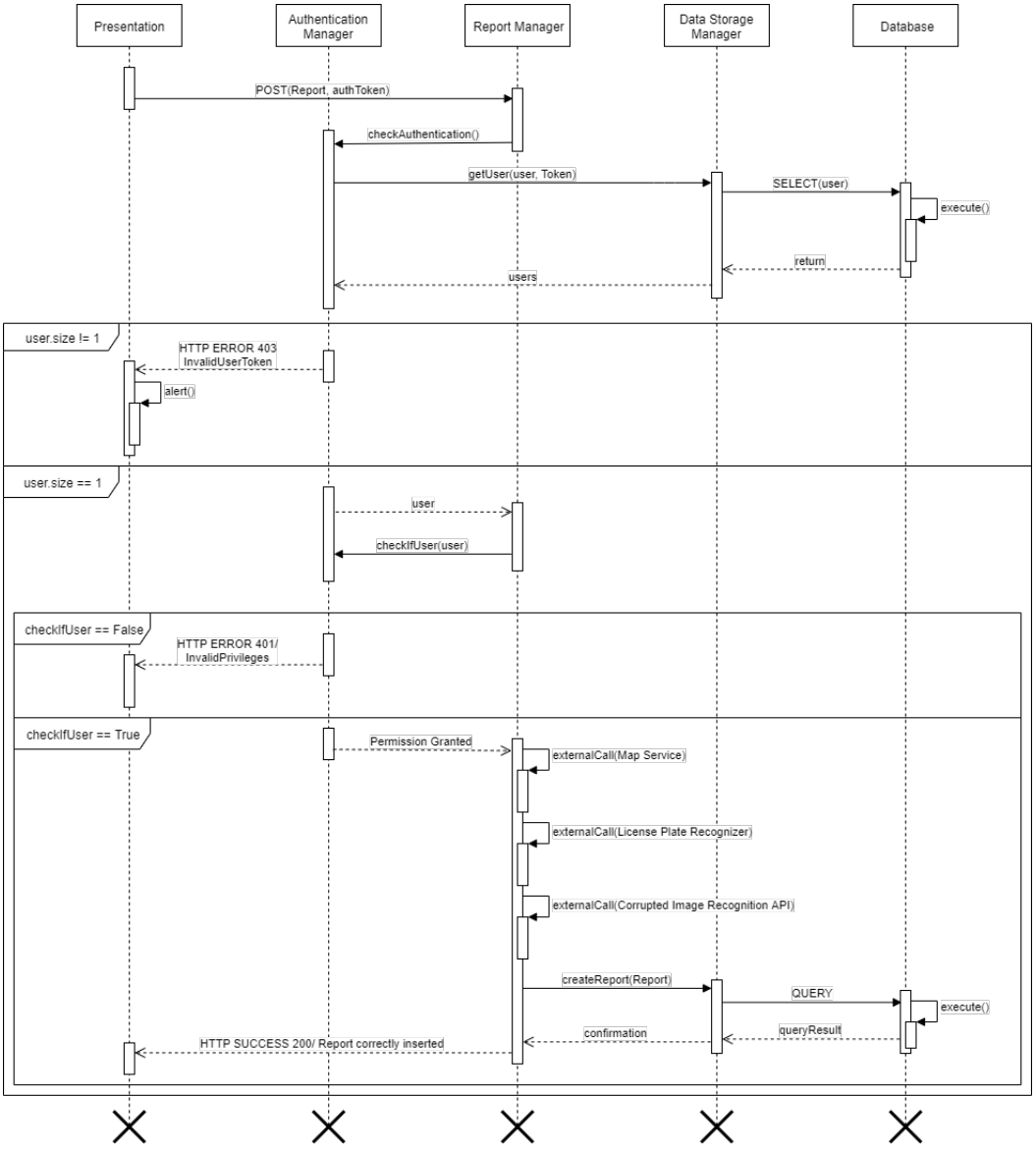

Figure 7: User send a violation Report Runtime View

2.4.5 Generate Traffic Ticket Runtime View

The sequence diagram below shows the RunTime view of the generation of a Traffic Ticket, like in the previous RunTime View the first steps are used to verify if the user exists and has a valid authentication token. The difference here is that the verification steps are started by the Report Manager since the User requests the Reports Lists, Once the Report Manager gets the User instance it asks to the Authentication Manager if it is an Authority. If the verification results *False* it returns a ERROR message to the User who sent

the initial request. If the verification is *True* then it invokes a query, through the Data Storage Manager, that allows to get the Reports corresponding to Authority's municipality.
The Authority that wants confirm/reject a real violation, sends a request to get the details of that report. Once he/she has seen the details as photo, license plate, type of violation and, eventually, notes added by the Report Manager, he/she can choose to confirm or reject the report through a new POST request containing the decision and the reportID that will be send to the Ticket Manager.
If the decision is to *reject* the report, then the Ticket Manager will invoke a query, through the Data Storage Manager, that allows to update the status of the Report as *rejected*. If the decision is to confirm the report then the Ticket Manager besides updating the status it will also perform an externalQuery (through the Municipality Traffic Ticket API) that will generate the real Traffic Ticket that will be send to the Person who committed the violation.
For simplicity the interactions that verifies that a Report is confirmed by only one Authority (and only one time) has been omitted but it should be kept in mind that, before modifying the status of the report, a locking mechanism is performed in order to do not send (to the Authority Ticket Service) the information of the report more than one time.

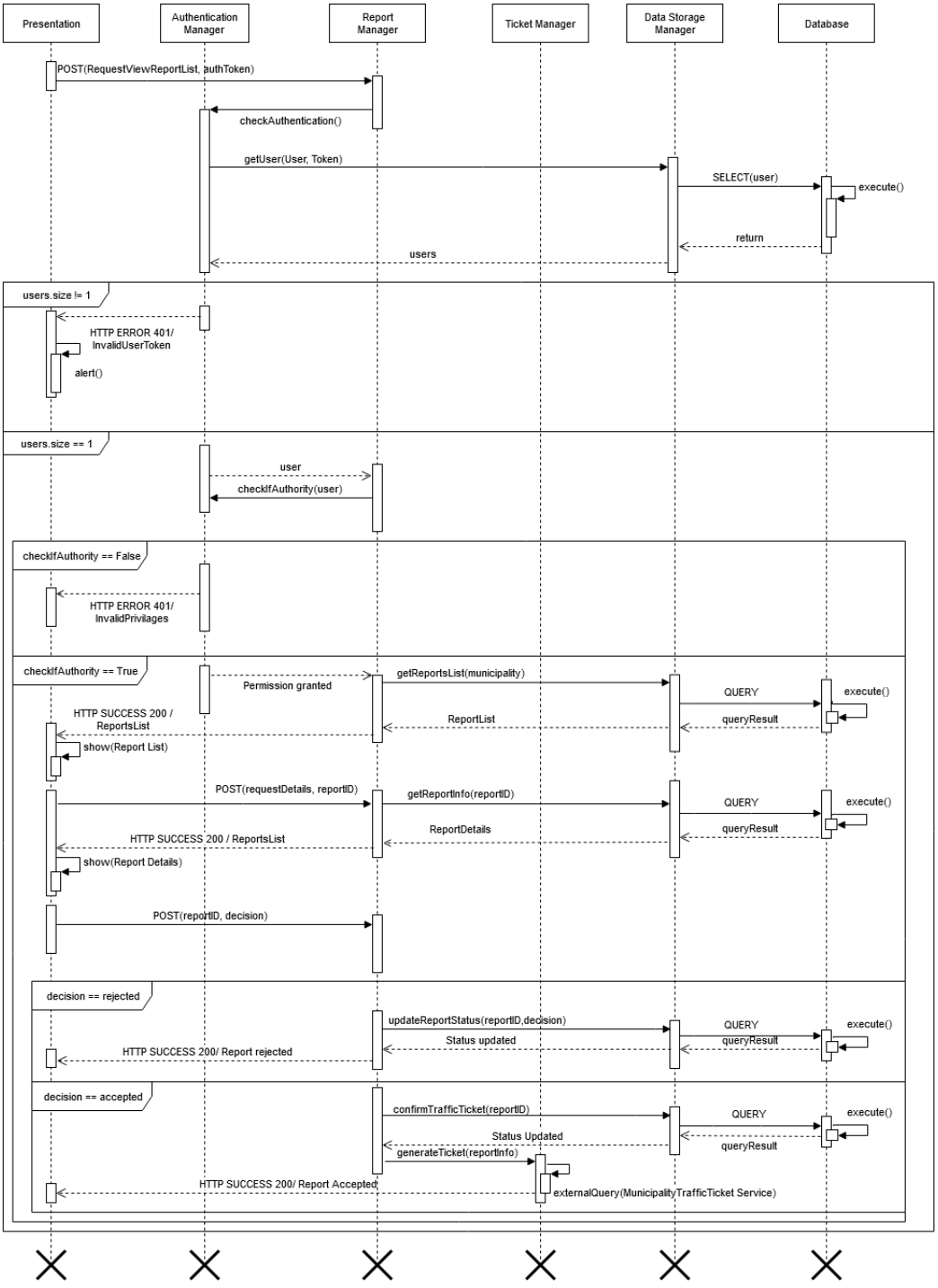

Figure 8: Creation of a Traffic Ticket Runtime View

2.4.6 Get Suggestions Runtime View

In this sequence diagram is shown the Runtime when an Authority want to see new suggestions.
As usual the first step is the verification of the authentication and of the necessary privileges. After that, the Suggestions Manager cross the data of the reports done on Safestreets with the Municipality data, so it needs to view the Report List and to invoke the Municipality System API. After the cross operation has been performed, it obtains one o more new suggestions that will be stored in the Database and then returned to the Application Client to be shown to the Authority.

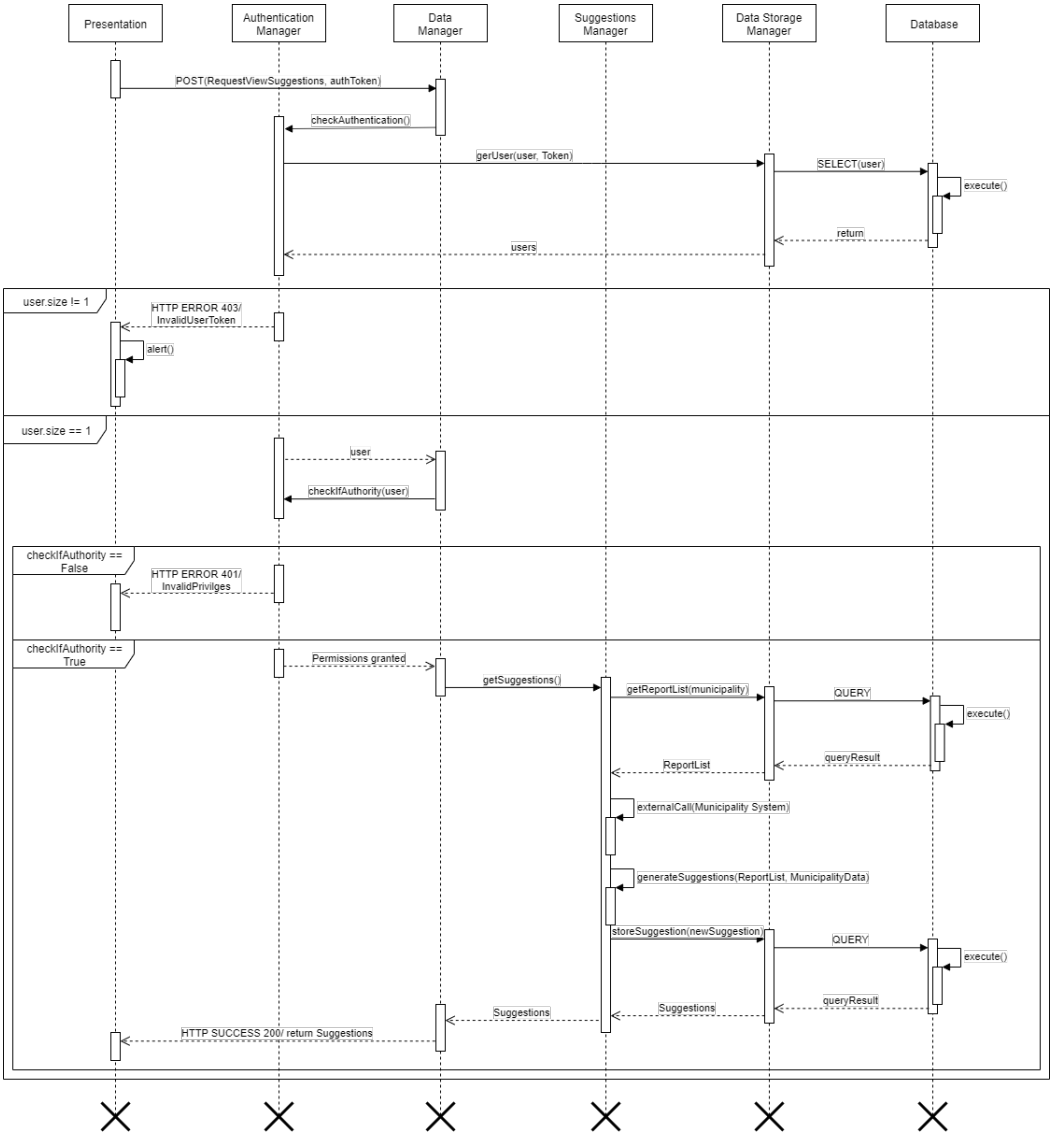

Figure 9: Request of Suggestions Runtime View

2.4.7 Get Statistics Runtime View

In this last sequence diagram is shown the Runtime behaviour of the Application when an Authority or a User want to see SafeStreets Statistics.
After the check of authentication (both User and Authority can see Statistics, but they must be logged in) the Data Manager sends the request to the Statitics Manager which take from the Database all the Reports and then extract information from them in order to generate statistics.
The final step is to send the generated statistics to the Application Client.

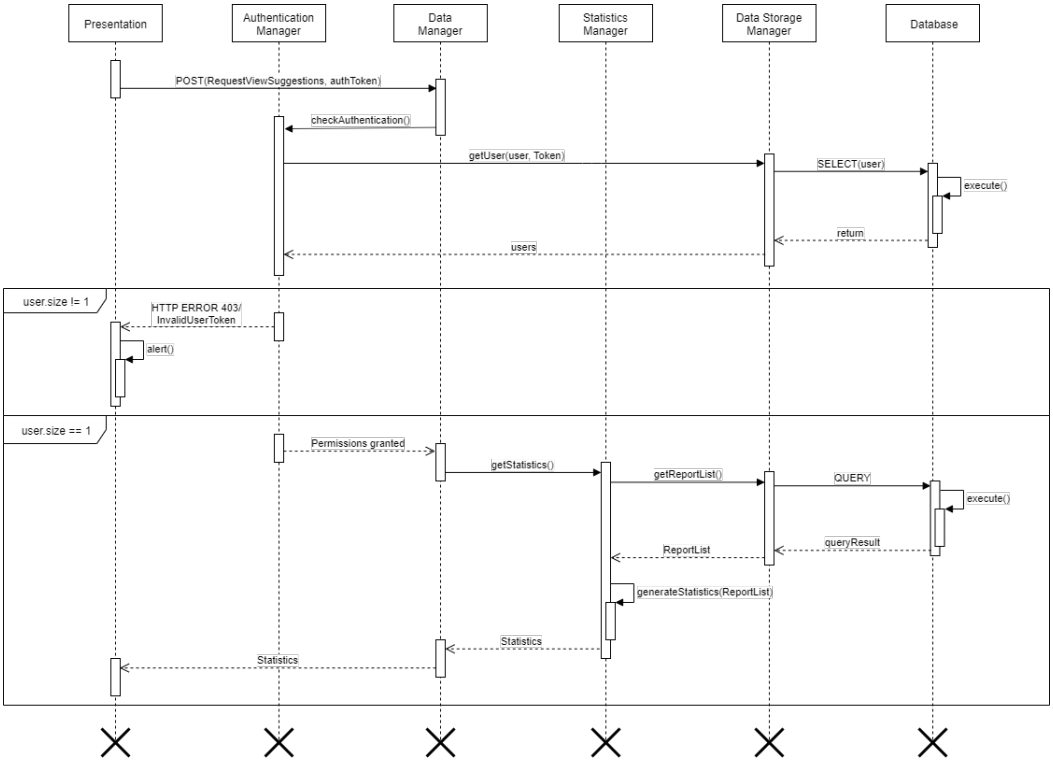

Figure 10: Request of Statistics Runtime View

2.5 Component interfaces

Interface Diagram

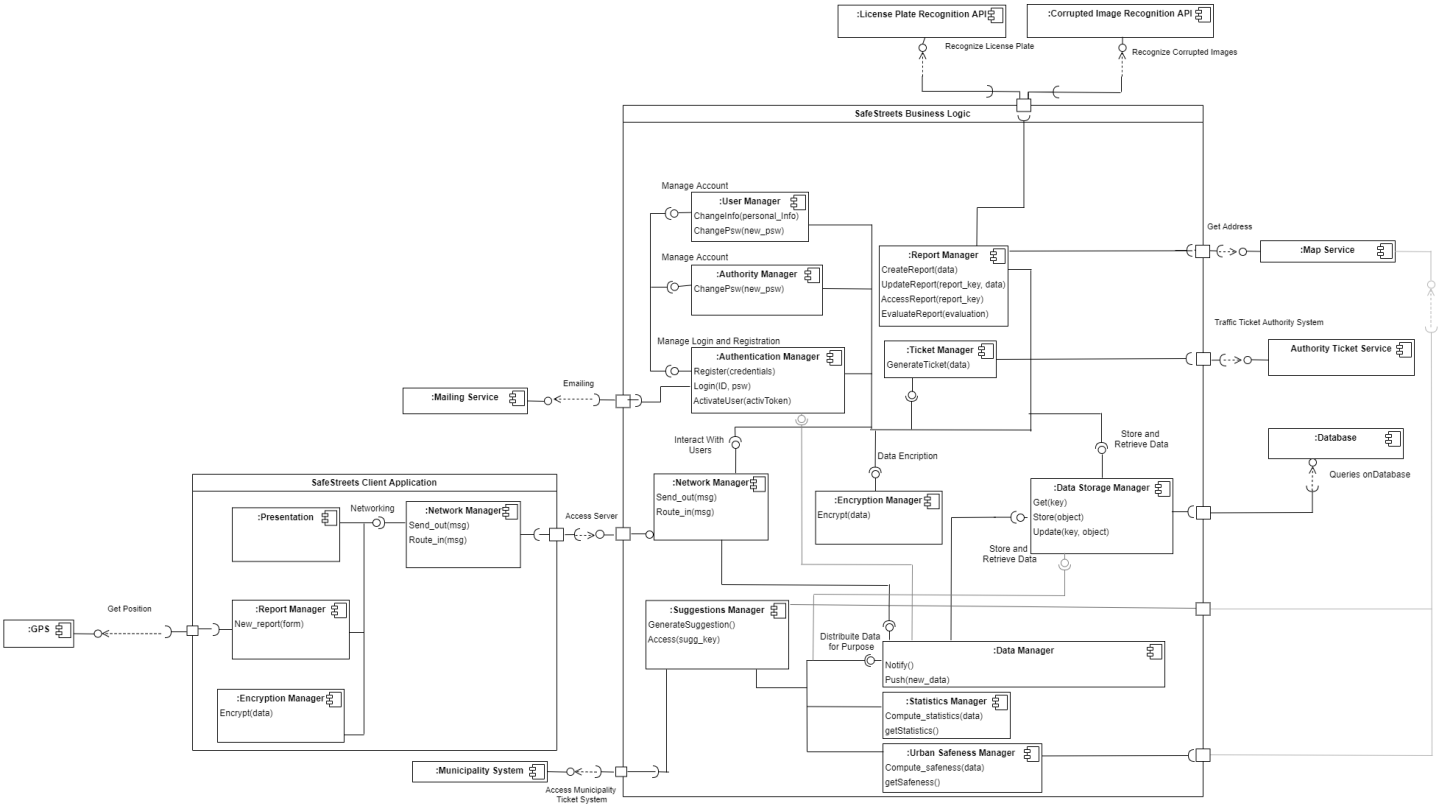

Figure 11: Component Interface diagram

REST API

Info	User register to the service
Endpoint	/*auth/reg/User
Method	**POST**
Body	email:[string] username:[string] password:[string] full_name:[string]

Success	Code: 200 OK
	Content:{
	message: "successfully registered"
	}

Error	Code: 400
	Content:{
	error:"data are not valid"
	}
	
	Code: 422
	Content:{
	error:"this user is already registered"
	}

Info	User activates his/her account
Endpoint	/*auth/actv?activToken=[string]
Method	**GET**
Body	

Success	Code: 200 OK
	Content:{
	message: "account successfully activated"
	}

Error	Code: 401
	Content:{
	error:"account not authorized. Token not valid"
	}
	
	Code: 422
	Content:{
	error:"this account is already activated"
	}

Info	User and Authority log in
Endpoint	/*auth/login
Method	**POST**
Body	username:[string] password:[string]

Success	Code: 200 OK
	Content:{
	message: "successfully logged in"
	}

Error	Code: 400
	Content:{
	error:"data are not valid"
	}
	
	Code: 401
	Content:{
	error:"account not activated"
	}
	
	Code: 401
	Content:{
	error:"incorrect username or password "
	}

Info	User modify his/her account info
Endpoint	/*User/Settings/info
Method	**POST**
Body	activToken:[string] name:[string] surname:[string]

Success	Code: 200 OK
	Content:{
	message: "changes applied"
	}

Error	Code: 400
	Content:{
	error:"data are not valid"
	}
	
	Code: 403
	Content:{
	error:"User username and token don't match"
	}
	
	Code: 404
	Content:{
	error:"User not found "
	}

Info	User accesses his/her account info
Endpoint	/*User/Settings/info
Method	**GET**
Body	

Success	Code: 200 OK
	Content:{
	name:[string]
	surname:[string]
	username:[string]
	}

Error	Code: 403
	Content:{
	error:"User username and token don't match
	}
	
	Code: 404
	Content:{
	error:"User not found "
	}

Info	Authority accesses his/her account info
Endpoint	/*Autho/Settings/info
Method	**GET**
Body	

Success	Code: 200 OK
	Content:{
	name:[string]
	surname:[string]
	username:[string]
	municipality:[string]
	}

Error	Code: 403
	Content:{
	error:"Authority username and token don't match
	}
	
	Code: 404
	Content:{
	error:"User not found "
	}

Info	User or Authority changes password
Endpoint	/*User/Settings/credentials or /*Autho/Settings/credentials
Method	**POST**
Body	activToken:[string] oldPassword:[string] newPassword:[string]

Success	Code: 200 OK
	Content:{
	message:"password changed successfully"
	}

Error	Code: 400
	Content:{
	error:"new password not valid"
	}
	
	Code: 400
	Content:{
	error:"old password not correct"
	}
	
	Code: 403
	Content:{
	error:"username and token don't match"
	}
	
	Code: 404
	Content:{
	error:"User not found "
	}

Info	A User sends a report
Endpoint	/*report/create
Method	**POST**
Body	picture:[object] kindOfviolation:[int] GPS:[string] dateTime:[date] (optional) plate:[string] (optional) model:[string] (optional) brand:[string] (optional) color:[string]

Success	Code: 200 OK
	Content:{
	message:report created successfully”
	}

Error	Code: 400
	Content:{
	error:”inserted data not valid”
	}
	
	Code: 400
	Content:{
	error:”missing some data”
	}
	
	Code: 401
	Content:{
	error:”Not authorized.”
	}

Info	User views a list of his/her reports
Endpoint	/*report/access?authToken=[string]
Method	**GET**
Body	

Success	Code: 200 OK
	Content:{
	listOf:{
	reportPreview:[object]
	}
	}

Error	

Info	A User views one of his/her reports
Endpoint	/*report/access?reportKey[string]&authToken=[string]
Method	**GET**
Body	

Success	Code: 200 OK
	Content:{
	picture:[file]
	typeOfViolation:[string]
	GPS:[string]
	dateTime:[date]
	address:[string]
	plate:[string]
	(optional) model:[string]
	(optional) brand:[string]
	(optional) color:[string]
	}

Error	Code: 400
	Content:{
	error:"reportKey not valid"
	}
	
	Code: 401
	Content:{
	error:"Not authorized. Report owner doesn't correspond to token."
	}

Info	Authority views a list of reports of his/her municipality
Endpoint	/*report?authToken=[string]
Method	**GET**
Body	

Success	Code: 200 OK
	Content:{
	listOf:{
	reportPreview:[object]
	}
	}

Error	

Info	An Authority evaluates a report
Endpoint	/*report/evaluate
Method	**POST**
Body	reportKey:[string] authToken:[string] evaluation:[boolean]

Success	Code: 200 OK
	Content:{
	message:"report evaluated."
	}

Error	Code: 400
	Content:{
	error:"reportKey not valid"
	}
	
	Code: 401
	Content:{
	error:"Not authorized."
	}
	
	Code: 422
	Content:{
	error:"report already evaluated."
	}

Info	Authority views a list of suggestions for his/her municipality
Endpoint	/*suggestions?authToken=[string]
Method	**GET**
Body	

Success	Code: 200 OK
	Content:{
	listOf:{
	suggestionPreview:[object]
	}
	}

Error	Code: 401
	Content:{
	error:"Not authorized."
	}

Info	An Authority asks for a new suggestion
Endpoint	/*suggestion/create?authToken=[string]
Method	**GET**
Body	

Success	Code: 200 OK
	Content:{
	suggestion:[string]
	(optional) GPS:[string]
	(optional) picture:[file]
	}

Error	Code: 401
	Content:{
	error:"Not authorized."
	}

Info	User or Authority asks for statistics
Endpoint	/*statistics/showStatistics?type=[int]&startDate=[date] &endDate=[date]&municipality=[string]
Method	**GET**
Body	

Success	Code: 200 OK
	Content:{
	x:[setOfInt]
	y:[setOfInt]
	}

Error	Code: 400
	Content:{
	error:"date is not valid"
	}
	
	Code: 400
	Content:{
	error:"municipalitiy is not valid"
	}

Info	User or Authority asks for safe/unsafe areas.
Endpoint	/*Safenessmanager/showSafeness?municipality=[string]
Method	**GET**
Body	

Success	Code: 200 OK
	Content:{
	requestedAreaMap:[object]
	}

Error	Code: 400
	Content:{
	error:"municipality is not valid"
	}

2.6 Selected architectural styles and patterns

In this section are shown the most relevant architectural design choices, with also the explanation of their role and their advantages.

Multi-tier Architecture
The architecture chosen for the System is a *multi-tier architecture*, composed by three tiers: *Presentation tier*, *Application tier* and *Data Storage tier*.

These division guarantees that each tier only deals with a specific task, creating a flexible and reusable application. Furthermore this very slight dependence allows modification to a single layer without affect the others, so the overall maintenance of the System is much easier.
Going more in details, these are the functionalities of each single tier:

- *Presentation tier:* is the only tier directly accessible by the user and it display to him/her all the information received by the Application tier. It also allows the user to interact with the application by requiring the execution of specific actions.
- *Application tier:* is the core of the application, because it controls the application functionalities. In our architecture it includes both the application servers and the reverse proxy, which is used to handle client requests and to balance the workload.
- *Data Storage tier:* in this last tier is included the Database of the System, so it includes all the mechanism about data storage. Also this layer is the core of the System, not in terms of functionalities but in terms of importance, because it contains all the data of the application and so it's always invoked by the other tiers.

REST
The communication between the Client and the Application Server is done according to the REpresentational State Transfer (REST).
Data exchange is also made through the HTTP Protocol and using SSL to provide encryption of sensible data.
Using HTTP and complying with REST guidelines allows to provide clear and complete APIs to the Clients.

Relational DBMS
The Data Storage layer consists of a relational DBMS. The relational model is a widely used design and there are several valid options, both open source and not, for what concerns the choice of the Management System.

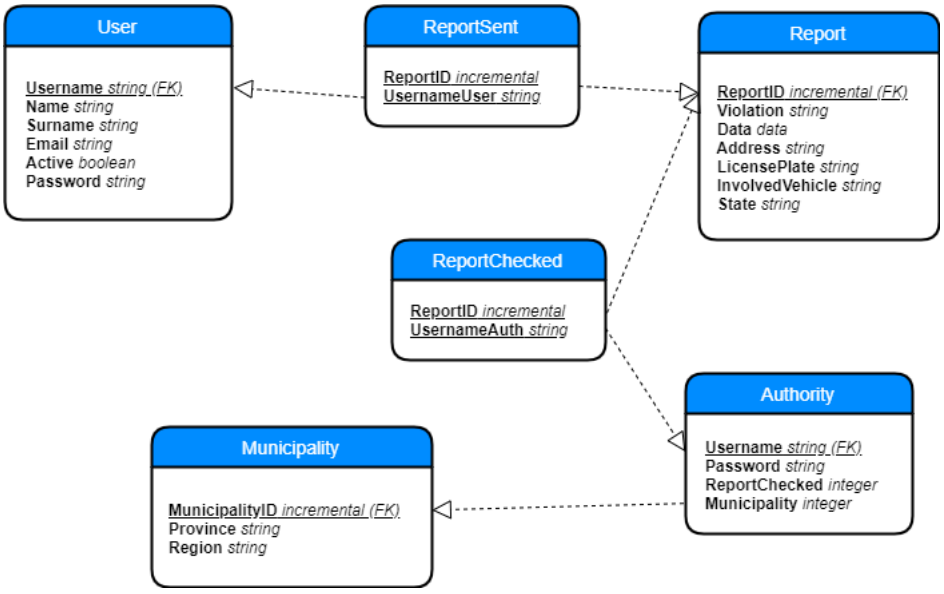

Figure 12: Database Schema Example

2.7 Other design decisions

Firewall
Between the Reverse Proxy of the Application Server and the Internet, there is a Firewall with a default-deny policy, to apply a selective package filtering. There are a few kind of request that can be sent to the server, so it is an easy job to configure all the possible incoming requests in the default-deny Firewall, getting better results in term of security.

Observer
An Observer Pattern has been implemented between the Data Manager and the Data Store Manager. Data Manager observes every new data that is added to the to Database. Then it notifies to Suggestion, Statistics and Urban Safeness Manager, depending on which of those is interested in the new data for their aims, pushing data to them.
This mechanism is used to make the Suggestions, Statistics and Urban Safeness Manager aware of each new data, so they can keep their information up to date.

Replication for Availability As mentioned in the RASD document, SafeStreets has to keep an availability of at least 90%, and to get this, the application server is replicated at least two times, to ensure that in case the main one stops working, it could be temporarily replaced. This mechanism will also help to improve the load balancing and the reliability of the system.

3 User Interface Design

The mockups of the application were already presented in the RASD. This section illustrates the User Experience flow by using two User Experience diagrams. Both diagrams are pretty similar since some interfaces are the same for both common User and Authority. As we can see from the following diagrams the leftmost part is merely devoted to allow users to log into the application, and is composed by these elements:

- Splash Screen
- Login
- Register
- Recovery Password

The Main Menu is different for User and Authority since an Authority is able to:

- View the violations notified by Users and generate traffic tickets
- View the own traffic tickets generated
- Get Suggestions

And the User is able to:

- Report a traffic violation
- View the own reports list

However, both share also some other interfaces in addition to the ones related to the login:

- Statistics
- Safe/Unsafe Areas
- Settings
- About SafeStreets
- Contact SafeStreets

Settings, About and *Contact SafeStreets* are accessible through the button in the top-left corner from the Main Menu. Those 3 activities were not shown on the mock-ups because aren't relevant for the Application functionality.

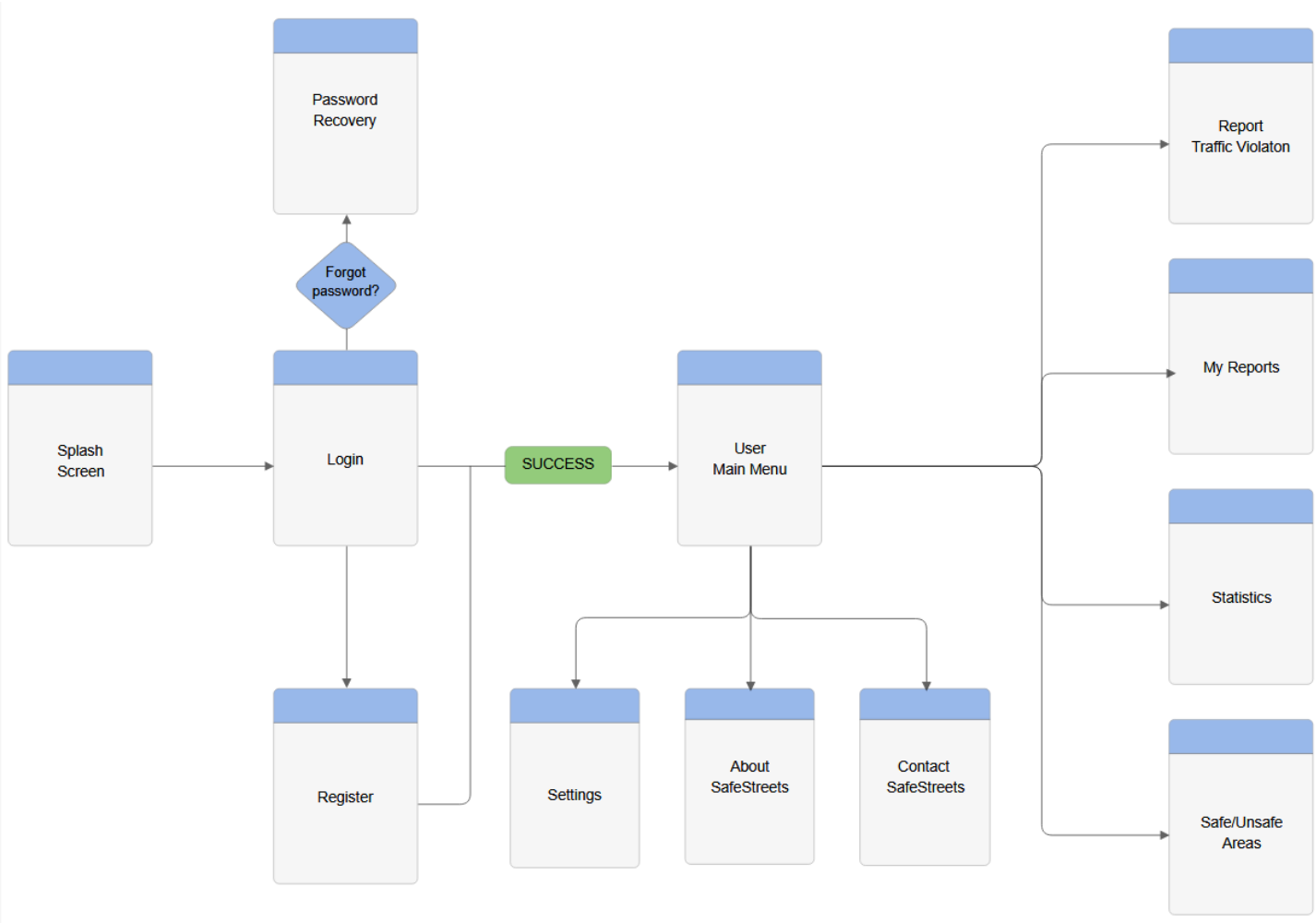

Figure 13: User UX Diagram

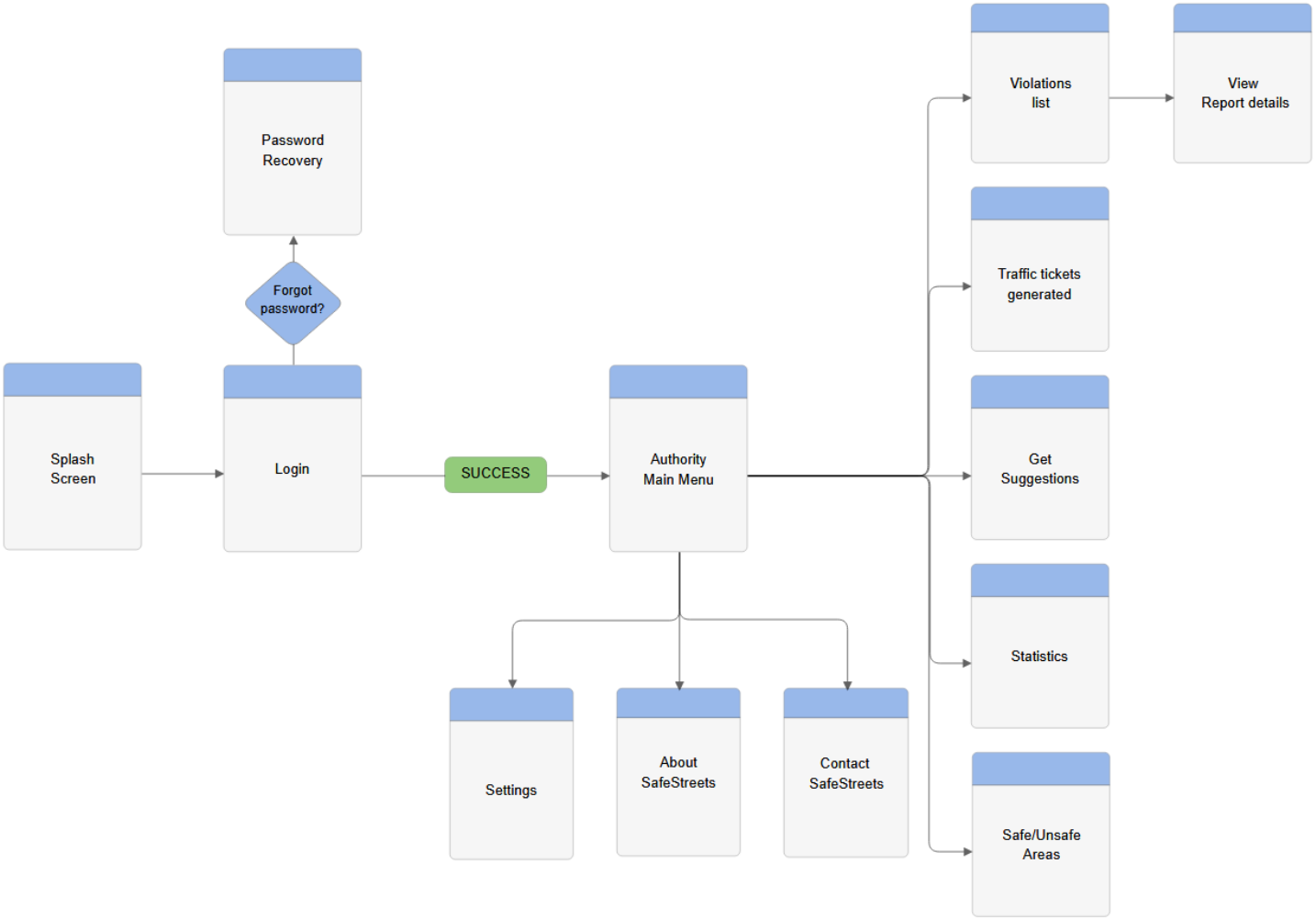

Figure 14: Authority UX Diagram

4 Requirement Traceability

Component (DD)	Requirements (RASD)
Authentication Manager	• **[R.1]**A Guest must be able to register and become a User. • **[R.2]**The S2B must provide an already created account to the Authorities. • **[R.3]**The S2B must check that credentials are correct, then send a confirmation e-mail. • **[R.4]**The S2B must allow Authority to log in with his/her given credentials. • **[R.5]**The S2B must allow User to log in with his/her registration credentials.
Report Manager (Server)	• **[R.6]**The S2B must provide an algorithm for compute the trustness of reports. • **[R.7]**User can see the list of his/her reports. • **[R.8]**User can edit or cancel his/her reports. • **[R.10]**The S2B must submit the received reports to an Authority for checking its validity. • **[R.11]**The S2B must recognize if a report that is about to be made, may involve a violation already notified, and alert the User. • **[R.12]**The S2B must alert the Authorities if more done reports refer to the same violation • **[R.17]**The S2B must allow only one Authority per time to change the status of a report. • **[R.18]**The S2B must submit to the Authority only the reports with a computed trust value higher than 20%. • **[R.26]**The S2B must not submit to Authorities the reports that have been valuated with a low trust level.

Component (DD)	Requirements (RASD)
Ticket Manager	• **[R.19]**The S2B must not allow Authority to generate more tickets for the same violation. • **[R.20]**The S2B must be allowed to access the Authority Traffic Ticket Service.
Report Manager (Client)	• **[R.9]**The User must send the report only within 5 minutes from when he/she starts to fill it in. • **[R.11]**The S2B must recognize if a report that is about to be made, may involve a violation already notified, and alert the User. • **[R.13]** The S2B must allow the User to send pictures taken by accessing the camera by the SafeStreets application. • **[R.14]**The S2B must allows the User to chose from a predefined list of "type of violation". • **[R.15]**The S2B sends date and time of the report taking them automatically from the operative system timer. • **[R.25]**The S2B must compare received data with the ones computed with algorithms in order to find discrepancies.
Data Manager	• **[R.21]**The S2B must keep its information about safe/unsafe areas up-to-date with all the verified reports it received. • **[R.22]**The S2B must keep its information about statistics up-to-date with all the reports it received. • **[R.23]**The S2B must generate suggestions based on the actual information and the actual urban situation.
Suggestions Manager	• **[R.23]**The S2B must generate suggestions based on the actual information and the actual urban situation.

Component (DD)	Requirements (RASD)
Statistics Manager	• [**R.21**]The S2B must keep its information about statistics up-to-date with all the reports it received.
Urban Safeness Manager	• [**R.20**]The S2B must keep its information about safe/unsafe areas up-to-date with all the verified reports it received.
Mailing Service	• [**R.3**]The S2B must check that credentials are correct, then send a confirmation e-mail.
License Plate Rec API	• [**R.18**]The S2B must submit to the Authority only the reports with a computed trust value higher than 20%.
Corrupted Image Rec API	• [**R.18**]The S2B must submit to the Authority only the reports with a computed trust value higher than 20%. • [**R.23**]The S2B must use an algorithm to recognize picture that have been physically or digitally modified.
Map Service	• [**R.16**]The S2B must use some out coming map service.

5 Implementation, Integration and Test Plan

As it was already presented in the previous Sections, the System can be seen in terms of three main subsystems:

- **Front-end subsystem**: Report Manager, Encryption Manager, Network Manager, Presentation.
- **Back-end subsystem**: Authentication Manager, Authority Manager, User Manager, Report Manager, Ticket Manager, Network Manager, Encryption Manager, Data Store Manager, Data Manager, Suggestion Manager, Statistics Manager, Urban Safeness Manager, Database.
- **External subsystem**: Mailing Service, GPS, Map Service, Municipality System, Authority Ticket Service, License Plate Recognition API, Corrupted Image Recognition API.

The implementation, integration and testing of the System will follow a *bottom-up* approach, without leaving aside the dependencies between components in the same subsystem. Thus, the implementation and integration process will begin considering different components in the same subsystem as first step and then it will continue with the integration of different subsystems.
It is important to notice that components in the external subsystem do not need to be implemented and tested since it is assumed that they are reliable. Another crucial fact to be considered is that, besides following a *bottom-up* approach, it is possible to implement components in parallel by using an incremental approach for the integration and testing processes. Once two components (within the same subsystem) are implemented, the integration and testing of those can be performed.

5.1 Implementation Plan

The implementation order each component in the Server side shall be:

1. Database Services (Database, Data Storage Manager)
2. Encryption Manager and Network Manager
3. Account Services (Authentication Manager, User Manager, Authority Manager)
4. Traffic Ticket Services (Report Manager, Ticket Manager)
5. Supplementary Services (Data Manager, Suggestions Manager, Statistics Manager, Urban Safeness Manager)

In the Database Services, since Data Storage Manager uses the Database, the idea of the *bottom-up* approach is to implement first the basic structure of the Database (and to chose the DBMS). Once the Database has been implemented, the Data Storage Manager can be implemented.
Network Manager and Encryption Manager are used to allow a secure communication between external services and internal services, and since it is important for the integration of the subsystems and components, it should be implemented as soon as the Database Services are working and we are able to receive/send messages.
Account Services allows to manage all the information about Users and what functionalities they can access, thus, this set of components should be implemented before Traffic Ticket Services.
Traffic Ticket Services are crucial for the basic functionalities of SafeStreets, thus, these components should be implemented before the Supplementary components.
The implementation order of each component in the client application side shall be:

1. Encryption Manager and Network Manager
2. Report Manager
3. Presentation

Following the *bottom-up* approach the Presentation component will be implemented as the last component in the client application side since it will need to use Report Manager, Ticket Manager, Encryption Manager and Network Manager.

5.2 Integration and Test Plan

In the following diagrams it will be shown how the integration and testing are performed. Network Manager and Encryption Manager will not be included for readability purposes, but it should be kept in mind that their integration and testing are performed after the components of the Database Services have been integrated and tested. Before starting with the integration process, all the unit test for each component already implemented will be performed. Once the unit test phase have been completed, the integration process can start.

5.2.1 Integration Process

Following what have been said in the previous section, the integration process follows an incremental approach, when a set of components (in the same Service) have been implemented, their integration will be performed.

Integration of components inside the Database Services
The first set of components to be integrated and tested are the ones that interact with the Database, thus, once the implementation of the Database have been completed its integration and testing with the Data Store Manager should begin.

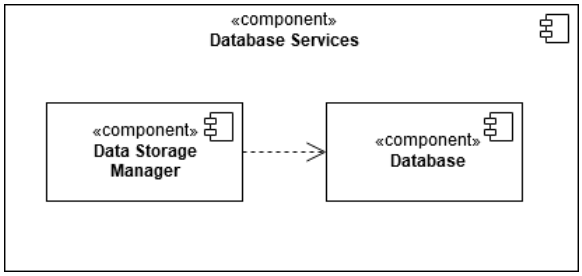

Figure 15: Integration of Database Services

Integration of components inside the Account Services
This set contains the components that allows to identify and authenticate a user. Once the User Manager and Authority Manager have been implemented and the unit testing of each have been performed, they can be integrated with the Authentication Manager.

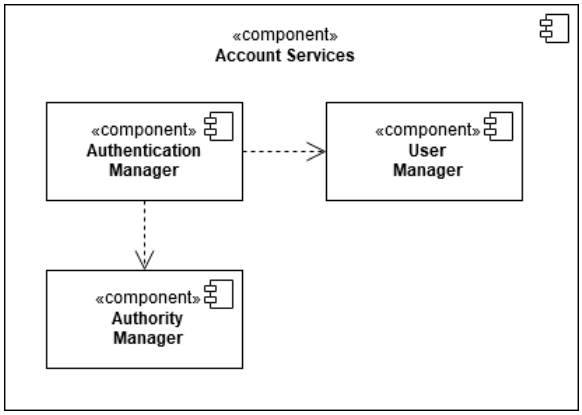

Figure 16: Integration of Account Services

Integration of components inside the Traffic Ticket Services
This set of components contains the Report Manager and the Ticket Manager, the first component that has been implemented is Ticket Manager since it is used by the Report Manager, once the implementation of Ticket Manager and Report Manager have been completed their integration and testing will start.

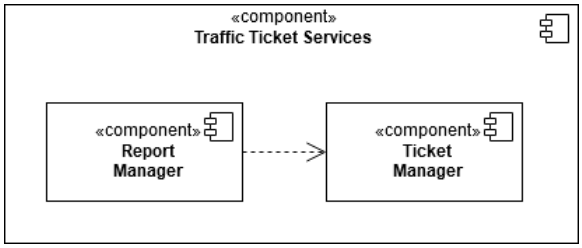

Figure 17: Integration of Traffic Ticket Services

Integration of components inside the Supplementary Services
Once the components that allows to perform the basic functionalities have been integrated and tested, the same process should be performed for the components that contains the logic behind the supplementary services. Following the *bottom-up* approach, once the leaves in the use hierarchy are ready to be integrated, the integration with Data Manager can start.

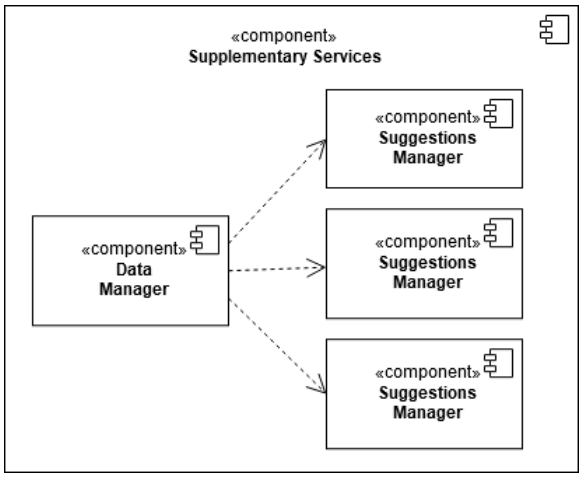

Figure 18: Integration of Supplementary Services

Integration of services inside the Back-end Subsystem
Once all the components inside the application server have been implemented, integrated and tested, the last integration and test can be performed and, thus, the back-end part will be ready to interact with the the client application.

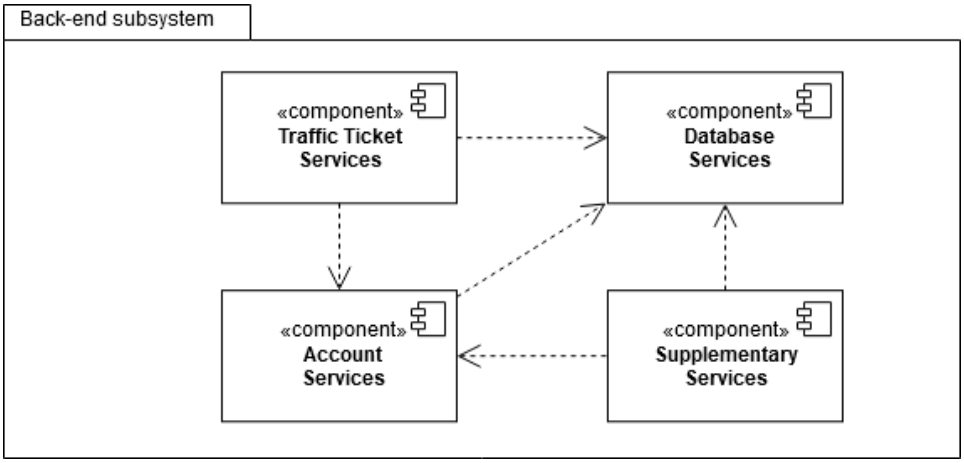

Figure 19: Integration of the Back-end Subsystem

Integration of Back-end components with external components
In the following diagrams, it will be shown how internal components of our system will be integrated with external components that each component use (taking into account the *bottom-up* approach). For simplicity the diagrams show only the components using the external *Services or API* and not the relation between internal components in our system.

Figure 20: Integration of Mailing Service

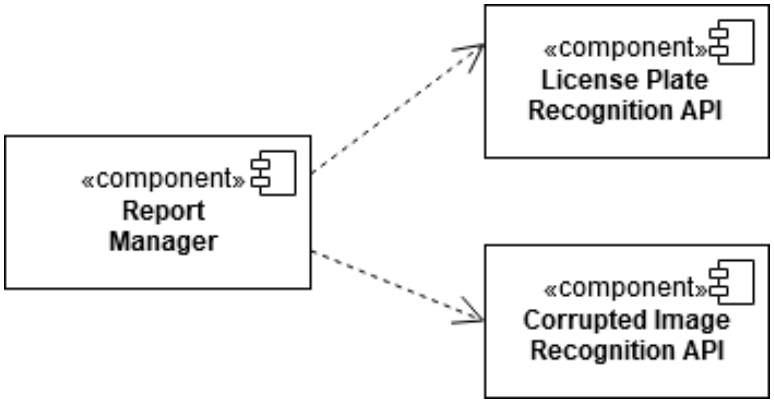

Figure 21: Integration of License Plate Recognition API and Corrupted Image Recognition API

Figure 22: Integration of Authority Ticket Service

Figure 23: Integration of Municipality System

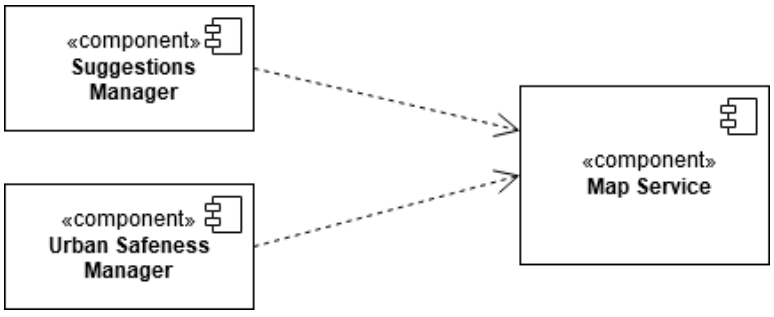

Figure 24: Integration of Map Service

Integration of components inside the Front-end Subsystem
Since the Front-end subsystem contains only the Encryption Manager, Network Manager, Report Manager and Presentation components, and three of them were already present in the back-end subsystem (for Report Manager different logic but same purpose), Their integration and test process will be the same with the only difference that the Presentation component will be the last component to be integrated and tested. In the following diagrams, the front-end subsystem will be represented as the *Client Application* component.

The following two diagrams represents how the integration between the client application and the application server is performed. First the client application is integrated and tested with the basic functionalities (Traffic Ticket Services), and then with the additional functionalities (Supplementary Services). At the end of each integration, an integration test is performed.

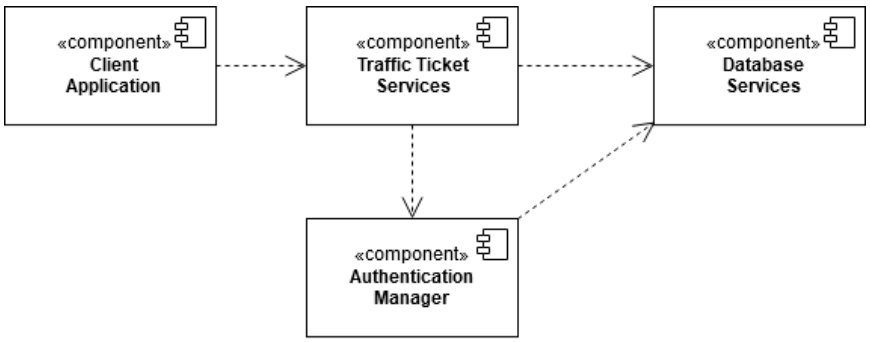

Figure 25: Client Application with Traffic Ticket Services

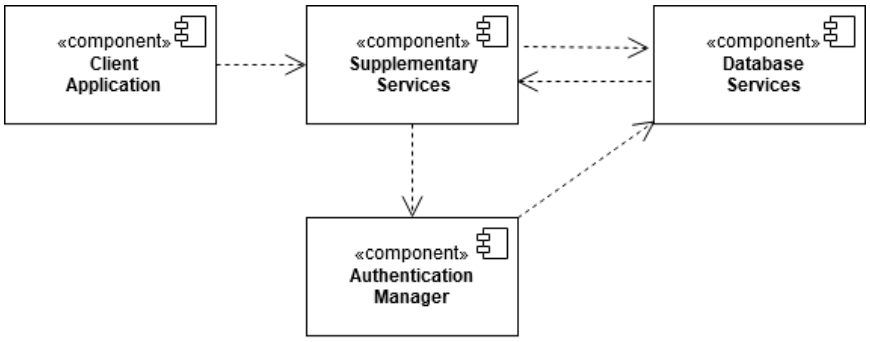

Figure 26: Client Application with Supplementary Services

6 Effort Spent

The following tables summarize the effort spent by each member of the team to create the Design Document.

6.1 Marri Iacopo

Description of the task	**Hours**
First meeting	2
Overview and Deployment	4
Component View	8
Component Interface	5
Architectural Styles and Patterns	2
UX Diagram	1
Requirement Traceability	3.5
Runtime View	2
Implementation	1.5
Final Review	4

6.2 Salamino Manuel

Description of the task	**Hours**
First meeting	2
Overview and Deployment	3
Component View	2
Component Interface	2
Architectural Styles and Patterns	4
UX Diagram	1.5
Requirement Traceability	2
Runtime View	8
Implementation	1.5
Final Review	4

6.3 Salazar Molina Steven Alexander

Description of the task	**Hours**
First meeting	2
Overview and Deployment	3.5
Component View	2
Component Interface	2.5
Architectural Styles and Patterns	3
UX Diagram	4.5
Requirement Traceability	2
Runtime View	8
Implementation	5
Final Review	4