Software Engineering 2

SafeStreets

Design Document

Fortina Valeria Maria
Galluccio Alessio

Monday, December 9, 2019

Contents

1 Introduction

1.1 Purpose

The following Design Document (DD) aims at providing a deepening, with more details, mainly technical ones, with respect to the RASD. In particular, the document will present a functional description of the SafeStreets system, showing above all the various components of the system and presenting a coherent architecture.

1.2 Scope

SafeStreets is a new crowd-sourced application whose goal is to ensure better road safety and improve compliance with the traffic rules, especially those related to parking. To achieve its goals, SafeStreets is aimed at different types of users, first all ordinary citizens. In fact, ordinary citizens can access SafeStreets services via the mobile application and, once registered, make a significant contribution. The application makes it possible to report possible violations by taking a photo and sending it along with general information such as the type of violation, the location, the date of the event and the license plates of the vehicles. These reports are taken over by the second type of user, those who can be defined as an authority, in particular, police officers. Once the report is received, the authority can assess its validity and if it deems necessary to directly generate a ticket to those who have committed the violation. Finally, to keep the system efficient over time, SafeStreets allows its employees, system managers, to make changes to the system, such as adding or deleting streets or types of violations, based on changes in the law or urban planning. To improve the user experience SafeStreets has also introduced additional features depending on the user, for example searching for a registered vehicle license plate, or violations committed on a specific street or even checking SafeStreets' efficiency data. SafeStreets is structured in a multitier architecture. More specifically, the Business logic layer has the task of providing the core features, through the use of interfaces, allowing users to generate reports or ticket. This layer is connected with the Data layer, in which are stored all the users' credentials but also all the reports and violations data .The Presentation layer, which provides presentation services, is build through the thin Client paradigm in which the client needs to perform close to no computation, allowing a more portable system.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Client: is a piece of computer hardware or software that accesses a service made available by another program (usually referred to as server).

- Server: a computer program or a device that provides functionality or services for other programs or devices, called "clients".

- End User: a common citizen registered to the system.
- Authority (or Authority User): a registered police officer.
- Report: a possible transgression of the traffic code.
- Violation: a verified report, which can be punished with a ticket.
- Ticket: sanction that an authority decides has to be paid as punishment for a violation of the traffic code.
- Efficiency: SafeStreets' efficiency, is calculated on the number of tickets issued before and after SafeStreets' arrival and on the the number of tickets on the total reports.
- Global Efficiency: SafeStreets' efficiency calculated on the whole territory covered by the system.
- Street Efficiency: SafeStreets' efficiency calculated in a specific street.
- Special User: An Authority user or a System Manager.

1.3.2 Acronyms

- DD: Design Document
- DB: Database
- API : Application Program Interface
- HTTPS : Hyper Text Transfer Protocol Secure
- HTTP : Hyper Text Transfer Protocol
- COTS : Commercial-Off-The-Shelf component

1.3.3 Abbreviations

- R.n : n-th functional requirement

1.4 Revision history

1.5 Reference Documents

- RASD document.
- Mandatory project assignment.

1.6 Document Structure

The rest of the DD is organized following a classical DD structure:

2 **Architectural Design:** It shows the main components of the systems and their relationships. Further details about the most meaningful interactions are given by means of sequence diagrams. This section will also focus on design choices, styles, patterns and paradigms.

3 **User Interface Design:** It is an improvement of the user interface given in the RASD document given through the use of UX modeling.

4 **Requirements Traceability:** It associates to each requirements one or more actual components of the system. Therefore, showing how design choices are able to satisfy requirements elicited in the RASD.

5 **Implementation, Integration and Test Plan:** It shows the order in which the implementation and integration of subcomponents will occur and how the integration will be tested.

2 Architectural Design

2.1 Overview

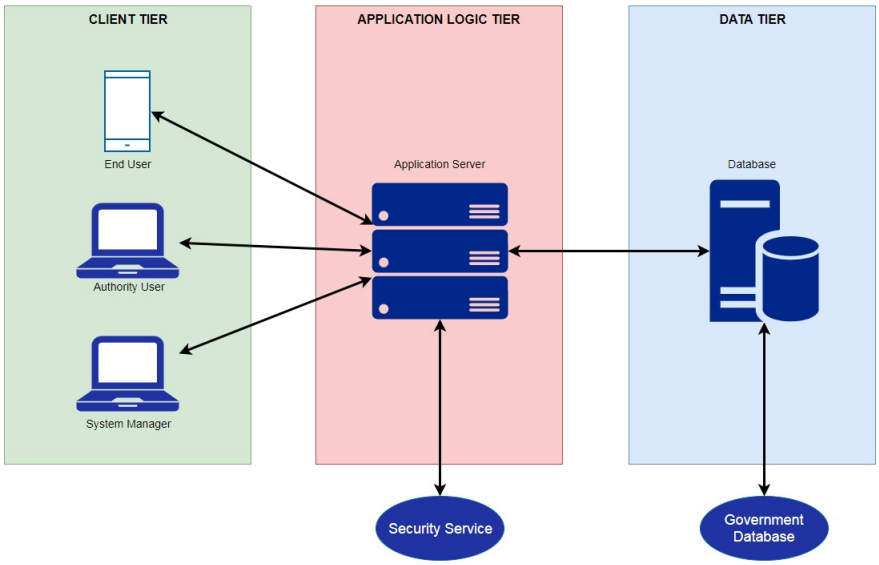

Figure 1: Overview diagram

The following image shows a high level abstraction of the system overview. The architecture is a 3-tier client-server with a thin client. The three layers Presentation, Application Logic and Data are divided in three different nodes. Therefore, the client node handles only the Presentation layer. The application server uses an external COTS security software. The database of the system is in communication with the Government Database, in order to synchronize the data between them. This diagram and the reasons behind the choice of this architecture are explained in the next sessions

2.2 Component view

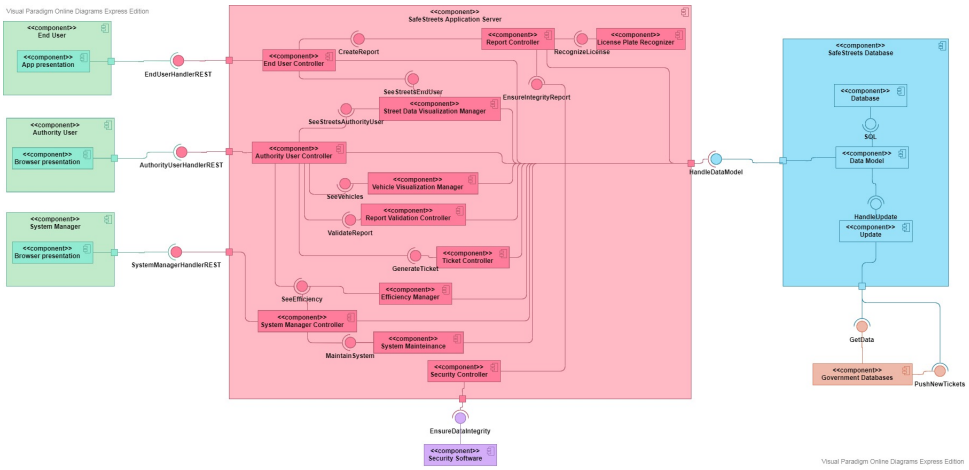

Figure 2: Component diagram

The UML component diagram aims at capturing the internal modular structure of components and their relationships.

Let's have a closer look to each component:

End User Controller
This component is responsible of all the interactions with the End User. It offers all the functionalities he/she needs. To achieve this task End User Controller uses the interfaces offered by the other components. The component handles the registration and login process. It also guarantee that the End User is authenticated, and that it can't access to other functionalities, like the System Manager's one, for example.

Authority User Controller
This component is responsible of all the interactions with the Authority user. It offers all the functionalities he/she needs. To achieve this task Authority User Controller uses the interfaces offered by the other components. The component handles the login process. It also guarantee that the Authority is authenticated, and that it can't access to other functionalities, like the System Manager's one, for example.

System Manager Controller
This component is responsible of all the interactions with the System Manager

user. It offers all the functionalities he/she needs. To achieve this task System Manager Controller uses the interfaces offered by the other components. The component handles the login process. It also guarantee that the System Manager is authenticated, and that it can't access to other functionalities, like the Authority's one, for example.

Report Controller
This component, exposes methods to generate a report. This component communicates with the component License Plate Recognizer, sending the images to recognize the license plate. After receiving the license plate from the component, it checks if the license plate received is present in the government database communicating with the Data Model. It also communicates with the Security Controller to verify that the image sent by the user related to the reports have not been modified.

License Plate Recognizer
This component verifies the all the licence plates highlighted in a report. To do this, after receiving a highlighted image, it uses an image recognition algorithm and returns the license plate. Since this algorithm must be updated regularly to follow the new technology, it is reasonably to use a COTS in its stead. In this case, it will be needed to create an adapter component between this COTS and the Report Controller.

Security Controller
This component is responsible for ensuring the integrity of data coming from end-user reports. After receiving the data from the component Report Controller, Security Controller verifies that the information has not been altered or modified before passing them back. Since the communication between the End User and the System is protected by an encrypted protocol, this component will mostly control that the time of the report is synchronized to the time of the server with a decided maximum error and that the photo of the report is not altered, for example by looking at the "print" that the camera leaves on the photo, and ensuring that it's the same in all the photo. Since this last control requires the State of Art technology, it is advised to use the interfaces of a COTS. If it is not available, the Security Software has to be implemented by us.

Report Validation Controller
This component takes care of providing Authority users with all the new reports sent by End Users, do achieve this task Report Validation Controller communicate with the Data Model. It also provides methods to delete a report in which the Authority User has not recognized a violation.

Ticket Controller
This component, exposes methods to generate ticket selecting one ore more license plates. It also communicates with the Data Model. In this way, tickets' data are saved both in the SafeStreets' Database and in the Government Database.

Street Data Visualization Manager
This component provides the End User and the Authority with the information on violations occurring in a specific street, for example the number of violations. To do this it has to communicate with the Data Model to access the data contained in the Database.

Vehicle Visualization Manager
This component has two main features. The first is to provide the data for a single vehicle, after the Authority has searched for the corresponding license plate. The second is to provide a complete list of all information on all vehicles registered in the SafeStreets' Database. To do this it has to communicate with the Data Model to access the data contained in the Database.

Efficiency Manager
This component is responsible for implementing the most relevant algorithm for calculating the Global and Street efficiency of SafeStreets. To do this it has to communicate with the Data Model to access the data contained in the Database. It also provides the Authority and System Manager users with the information obtained.

System Maintenance
This component, exposes methods to allows the System Manager to make changes for future reports, by adding or deleting a street or a type of violation, and to add or remove Special Users.

Data Model
It handles all the interactions with the Database. It's used to insert new data in the database and to extract data from it, converting them in compatible objects usable by the application. It has also a crucial role in the handle of reports. Since more End User can create more reports for the same violation, when the Report Validation Controller requests the list of the reports, Data Model will not give the entire list of reports, but it will aggregate them and give only one for each "case". This could be implemented with various algorithms.

One is proposed here. Data Model creates sets of reports that shares the type of violation and location in a period of time (a day, for example) and sends the minimum number of reports needed to cover all the License Plates reported. If two or more of these reports share a License Plate, only one of them will maintain it, in order to not create two or more tickets to the same License Plate if it is found to have committed a violation. It should act in the same way if a report is deleted by Report Validation Controller or Ticket Manager: it should remove all the reports that share the same type of violation, time and location and that have the same or a subset of the license plates. If a report has some license plates of the report deleted, but has some other too, it should maintain only the license plates not affected. The others should be deleted.

Update
Update is a component that observes the Data Model component and the Government Database. It uses the Observer pattern in both ways. It ensures that the Database of SafeStreets and the Government one are consistent. Moreover, It handles all the interactions with the Government Database using the interface offered by it.

2.3 Deployment view

In this section we analyse how the system will be physically deployed. In the deployment diagram we decided to represent only the the section of the system which depends on our choices. Besides, the nodes that contain the security software and the government database are not showed, with the exception of Government Database, because it was necessary to show the firewall between the two databases. It was decided to put firewalls between the application server and the Database and between the Safestreets Database and the Government Database because security is a great concern in this System, since it must guarantee that the reports of the End Users and other information are not altered. Besides, the communication to the Application Server use HTTPS protocol, in order to guarantee the security of the communication.
Since the Application Server must be capable of load balancing, it is recommended to use an event-driven architecture for it, since it is very conservative memory-wise and, by using one worker per core, it achieves a near-optimal CPU utilization. This would increase the parallelism. It would be possible to add more Application servers in order to augment the parallelism, the availability and the reliability of the application. This would increase the costs, so it is recommended in case of huge traffic server.
The Database is in a separate node from the Application server. It was decided to use this architecture because it permits to clearly separate the layers of the

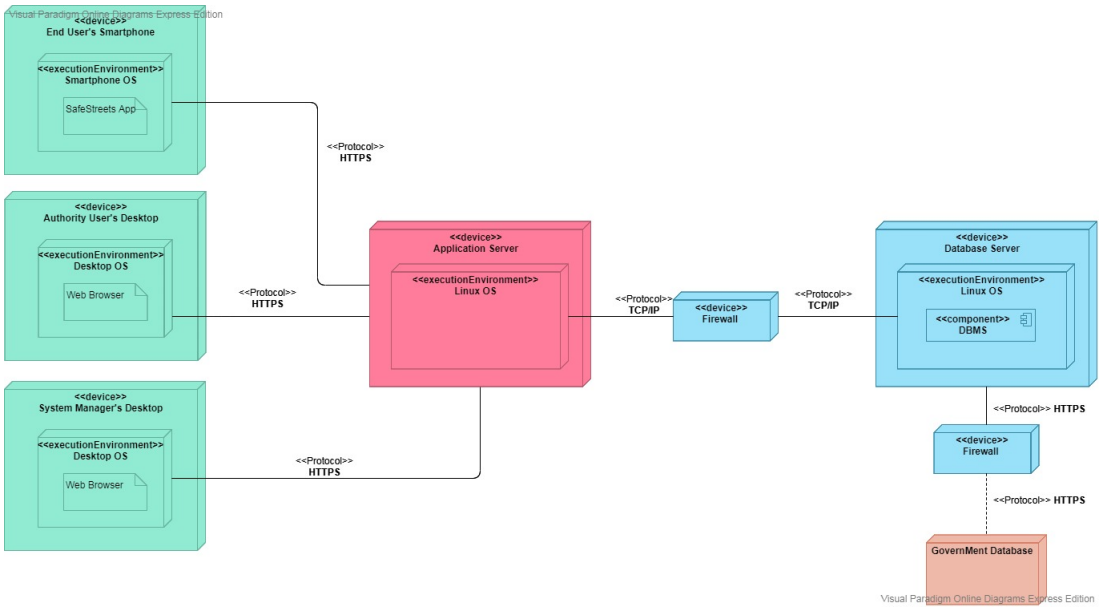

Figure 3: Deployment diagram

system. This would help the maintenance of the system. Moreover, it permits to add new nodes of databases to increase the availability and availability and reliability of the data storage only. This would be optimal if the systems needs a huge data storage capability, but a low server traffic. It would be possible to add more nodes of databases without adding new and useless application servers. The same idea is valid even in huge server traffic and low data storage need.
Having the database separated from the application server has another advantage for our architecture. The SafeStreets' database must synchronize with the Government database, in order to be able to push new generated tickets and to get data of violations reported not through SafeStreets' app. Having this update algorithm on a different machine of the application server simplifies the maintenance, since it shouldn't be transparent to the application layer.

2.4 Runtime view

This section presents some of the most important Use cases previously seen in the RASD. To give a more complete view, some sequence diagrams are seen from different perspectives:
The most relevant use cases chosen are subsumed below:

End user prospective:

- End User Registration
- Get Street Information
- Add License Plates for report
- Report Generation

Authority prospective:

- Authority Login
- Ticket Generation

System Manager prospective:

- Authentication Process
- Delete a Violation Type

2.4.1 End User Registration

In this sequence diagram it is shown how an End User can register to SafeStreets. The guest must register in order to access to all the functionalities. To do this, the guest fills a form with his / her username, a password and the fiscal code. This information arrives at the End User Controller through an HTTP POST request. The End User Controller handles the request, and, through the Data Model's Interface, verifies that the fiscal code is not already used. Any problem (like invalid information or fiscal code already in use) are handled by the system and it will return an error code to the user. The correct data is then saved in the database.

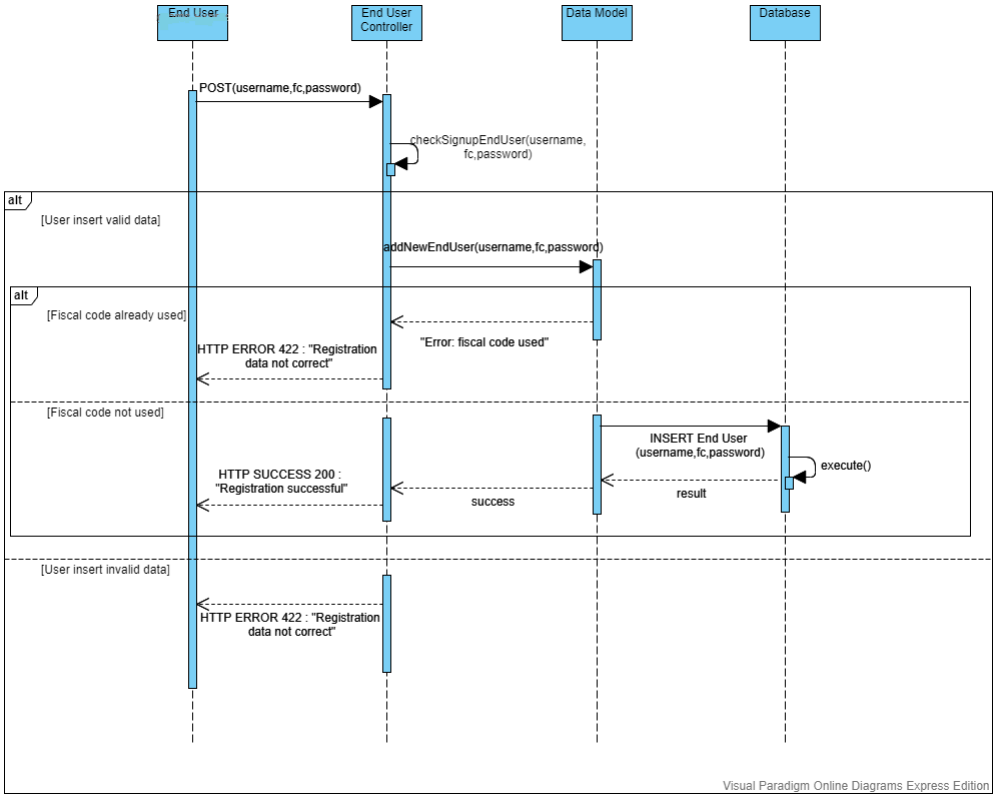

Figure 4: Sequence Diagram: End User Registration

2.4.2 Authority Login

In this sequence diagram it is shown how an Authority can login to SafeStreets. The Authority must fill a form with his / her code and a password. This information arrives at the Authority User Controller through an HTTP POST request. The Authority User Controller handles the request and it validates this info by comparing them to the info saved into the DB. If they are valid, it returns an access token to the user; if they are not, an error code is sent instead. This process is similar for the End User and the System Manager.

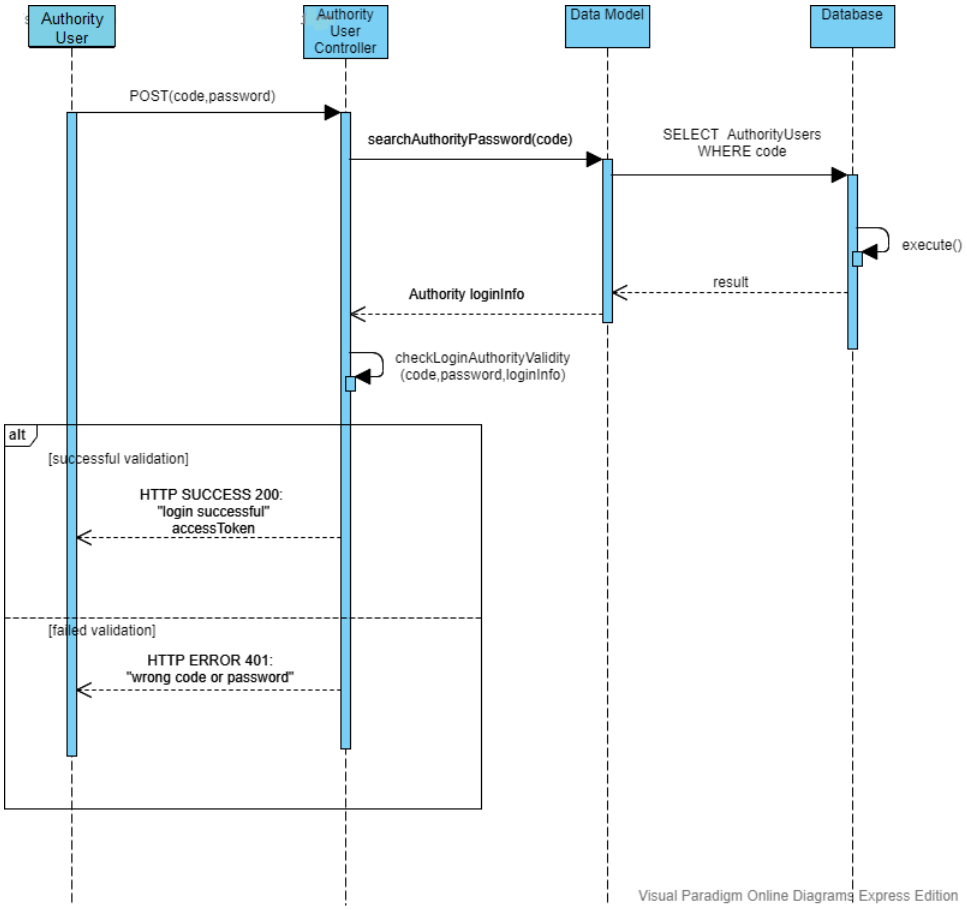

Figure 5: Sequence Diagram: Authority Login

2.4.3 Authentication Process

In this sequence diagram it shows the authentication procedure. The authentication procedure lets the system know if the System Manager that is trying to make a request is logged in or not. First, the user makes an HTTP POST to the System Manager Controller, sending its access token (previously received with the login). The System Manager Controller searches for this token in the DB. If it is found, then the System Manager is authenticated and the request is elaborated; if not, the user receives an error. The same process is valid for End Users and Authority users, with a small difference in the first component used (End User Controller and Authority User Controller).

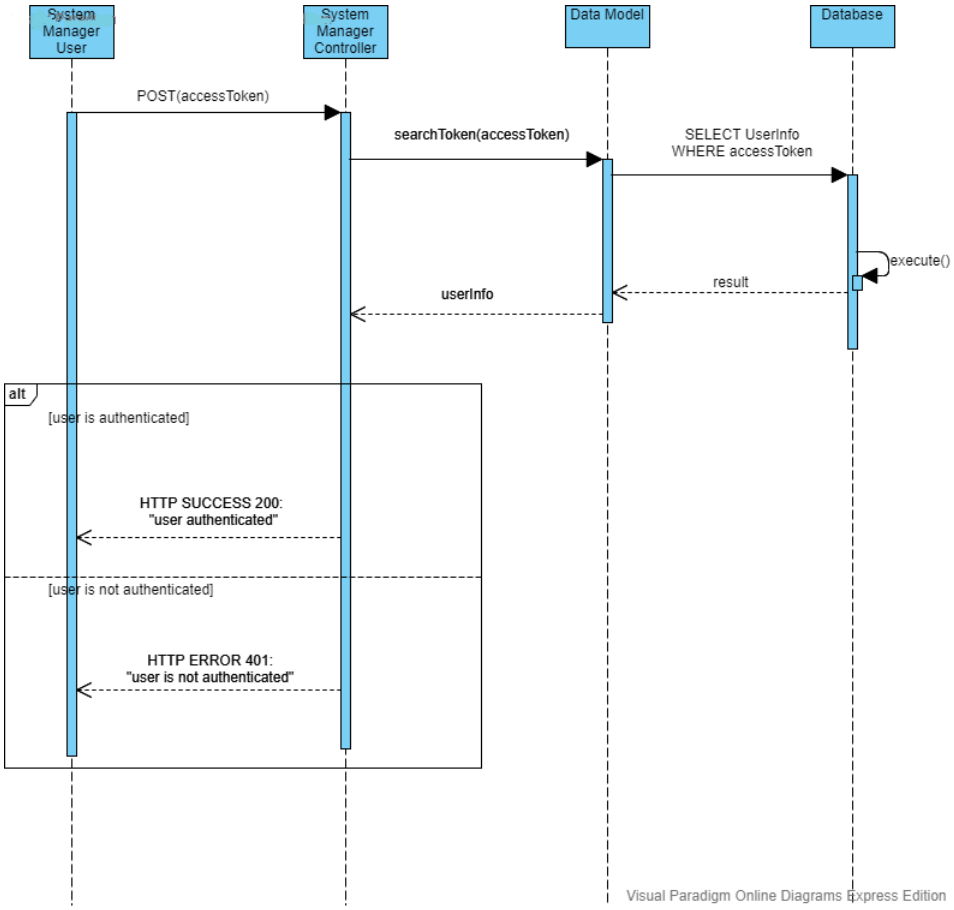

Figure 6: Sequence Diagram: Authentication Process

2.4.4 Get Street Information

In this sequence diagram it is shown how a End User can access to information about a specific street. The End User send the access token for the authentication process, which was reported in the previous sequence, and the address of the street. This information arrives at the End User Controller through an HTTP GET request. The End User Controller handles the request, and use the Street Data Visualization Manager's Interface to get the data. If the street address was spelled wrongly the problem is handled by the system and will return an error code to the user. Otherwise the End User receives the requested information. A similar procedure is valid for the Authority User.

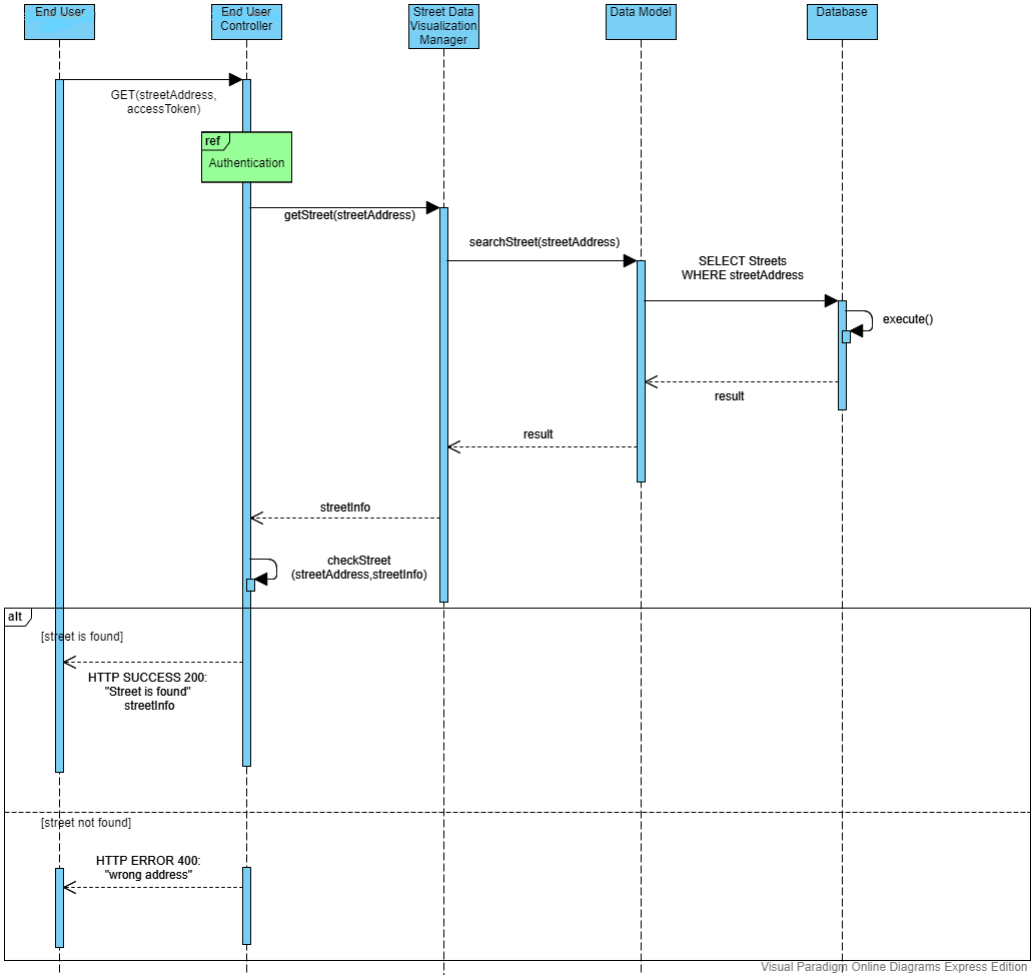

Figure 7: Sequence Diagram: Get Street Information

2.4.5 Add License Plates for report

In this sequence diagram it is shown how an End User can add one or more license plates to a report. The End User sends the access token for the authentication process, the image of the possible violation and an highlighted zone of the image. This information arrives at the End User Controller through an HTTP GET request. The Report Controller use the License Plate Recognizer to get a license plate. If the license plate is found the Report Controller verify, thanks to the Data Model and Update, if the license is registered in the Government Database. If the License Plate Recognizer doesn't found a licence plate or the license plate is not registered in the Gov DB, the problems are handled by the system and will return an error code to the user. Otherwise the End User receives the requested information.

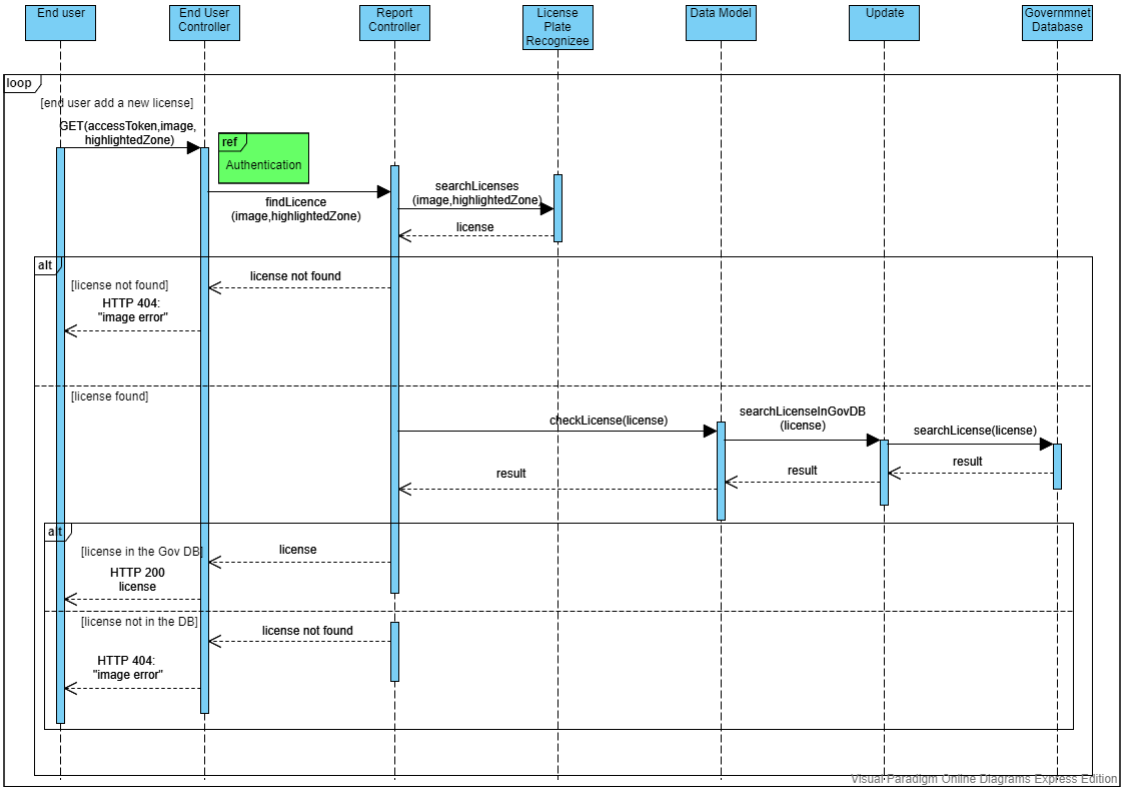

Figure 8: Sequence Diagram: Add License Plates for report

2.4.6 Report Generation

In this sequence diagram it is shown how an End User can generate a report. The End User sends the access token for the authentication process, the image of the possible violation and an highlighted zone of the image, the time, the position and the type of violation. This information arrives at the End User Controller through an HTTP POST request. First the Security Controller verify that the image and the time are not altered. The Report Controller use the License Plate Recognizer to get all the license plates. If the license plates are all found the Report Controller verify, thanks to the Data Model and Update, if all the plates are registered in the Government Database. If the image or time are modified or if the License Plate Recognizer doesn't found a licence plate or the license is not registered in the Gov DB, the problems are handled by the system and will return an error code to the user. Otherwise the End User receives a confirmation message.

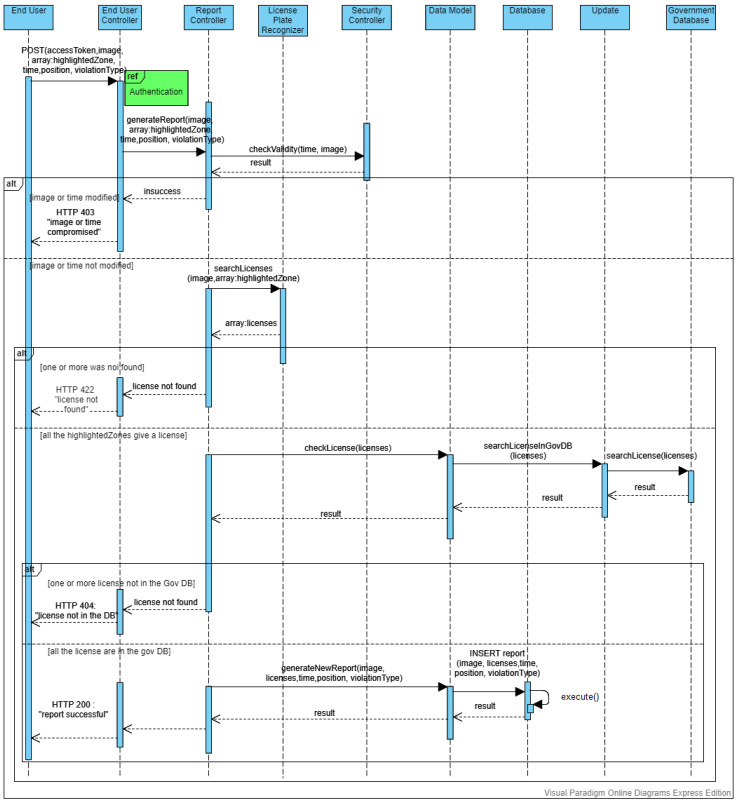

Figure 9: Sequence Diagram: Report Generation

2.4.7 Ticket Generation

In this sequence diagram it is shown how an Authority can generate a ticket. The Authority sends the access token for the authentication process. The Authority ask to see all the reports that don't have a ticket yet. This information arrives at the Authority User Controller through an HTTP GET request. Report Validation Controller provides all the new reports using the Data Model methods. Data Model, in fact, as explained in the component section, returns only an aggregation of the reports, in order to let the Authority see all the possible violations, but without the risk to assign to the same License Plate more tickets for the same violation. Then the Authority ask for the information about a specific violation and send a report id. This information arrives at the Authority User Controller through an HTTP GET request. Report Validation Controller provides the information about the specific report. If the Authority User think that the reports corresponds to an actual violation, Authority Frontend sends all the information about the violation and the license plates that should receive the tickets. The Authority User Controller receives through an HTTP POST request, and Ticket Controller generates the tickets. Instead if there isn't a violation the Authority sends the report ID to delete the report and the Authority User Controller receives through an HTTP DELETE request. In both these precedent cases, the report must be deleted. However, since the reports showed to the Authority are only an aggregation, Data Model has to delete all the reports that have the same license plates or a subset of them for that violation, and delete the license plates which appeared in the selected report in the other reports of the same time, place and violation, as explained in the component section. If there aren't new reports, the Authority User Controller provides the error. In this case the other possible errors are omitted for simplicity, since all the problems are handled by the system and it will return an error code to the user.

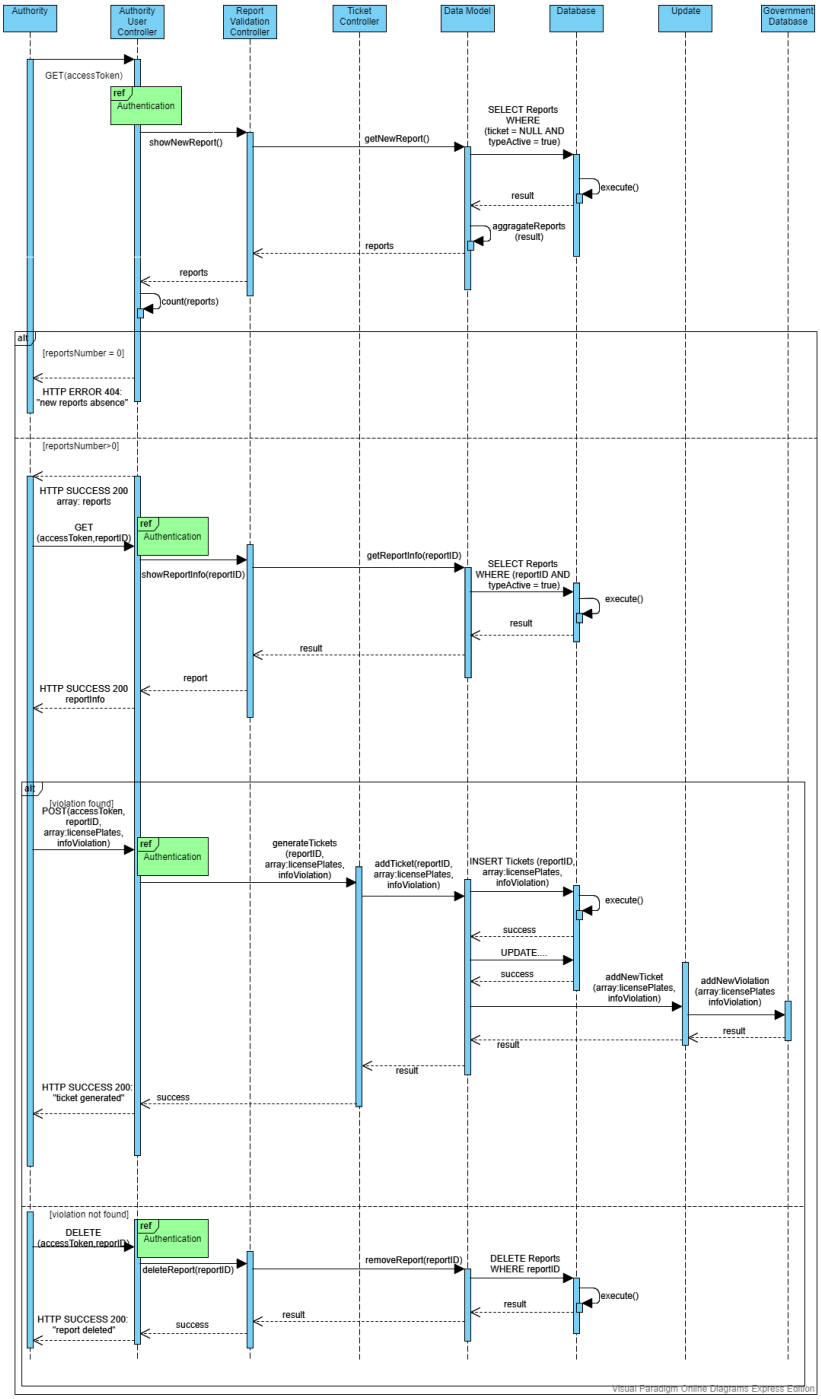

Figure 10: Sequence Diagram: Ticket Generation

2.4.8 Delete a Violation Type

In this sequence diagram it is shown how a System Manager can delete a Violation Type. The System Manager sends the access token for the authentication process. This information arrives at the End System Manager Controller through an HTTP GET request.System Manager Controller show to the System Manager all the violation types saved in the Database. The System Manager proceed by sending the violation type. This violation type is searched in the Database and the flag typeActive is set to false. For simplicity, the case in which there are still no types of violations registered in the database and the case in which the type searched for deleting is not registered are both omitted. Nevertheless this problems are handled by the system and it will return an error code to the user.

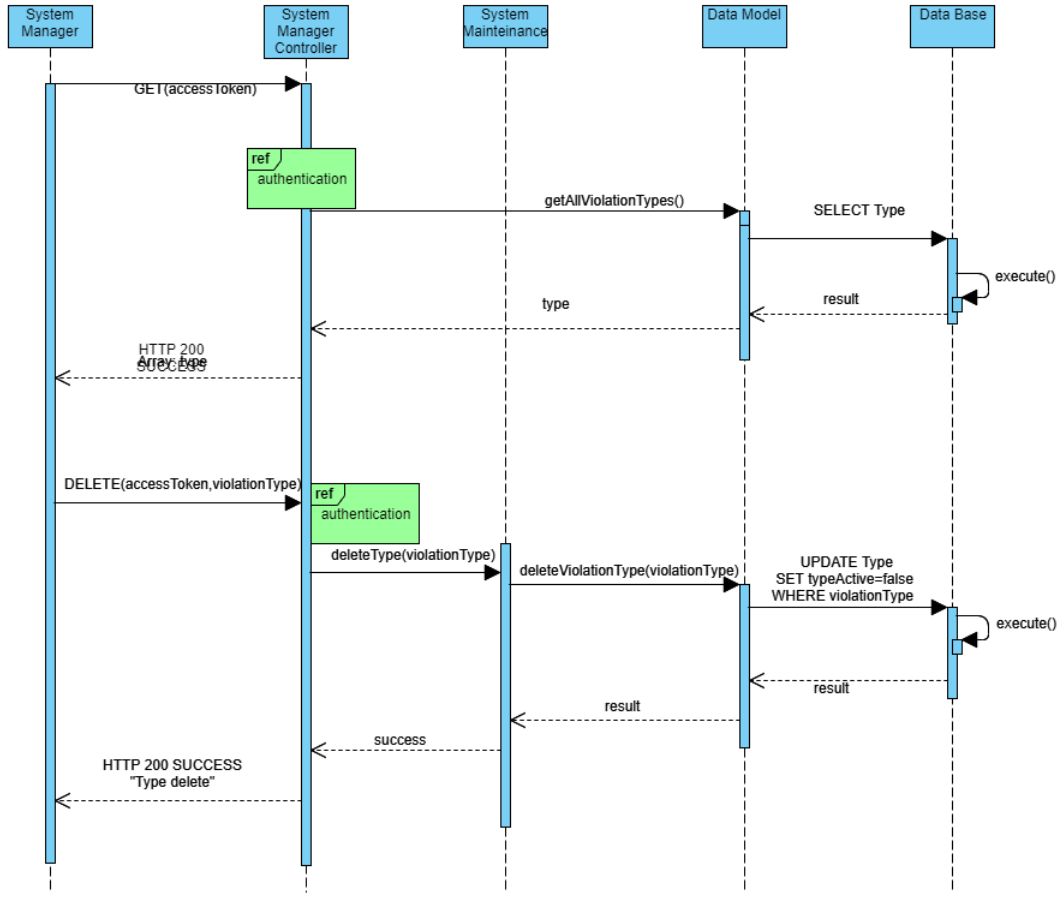

Figure 11: Sequence Diagram: Delete a Violation Type

2.5 Component Interfaces

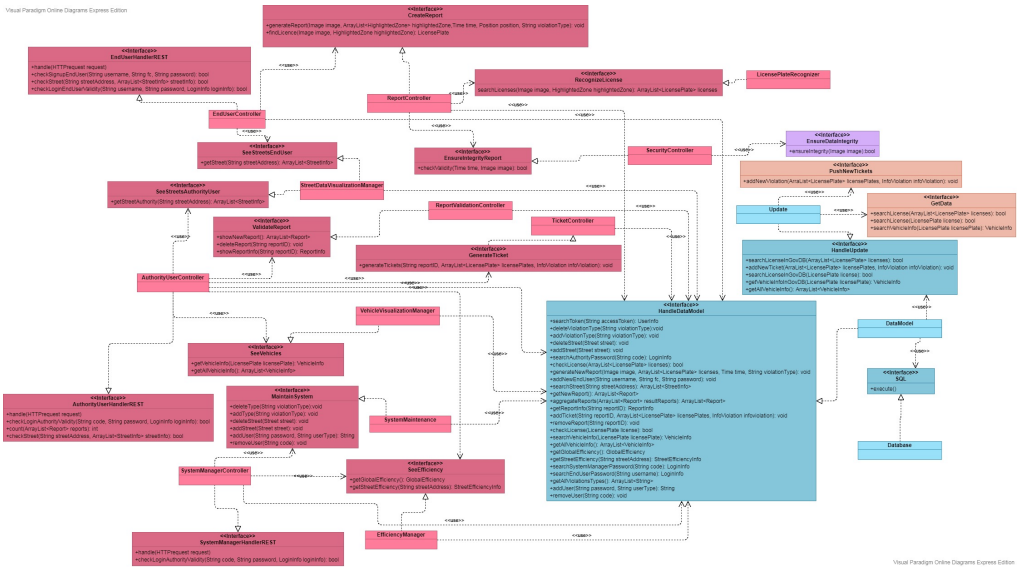

Figure 12: Interface diagram

In this section, the components interfaces are showed with the most important methods. The Interface Diagram gives a briefly overview. The most important methods of the interfaces are showed, but developers could and add more as needed, without disrupting the idea of the architecture, however. The Observer methods of the interface of Update are not showed, since they depends on the interfaces offered by the Government Database, which could vary. The following tables give a more detailed structure of the most critical interfaces of the system.

2.5.1 End User's REST APIs

API1.1

Name ID	**End User Registration**
Endpoint	*/enduser/register
Method	POST
URL parameters	
Data parameters	password: [alphanumeric] fiscal code: [alphanumeric] username: [alphanumeric]
Success response	Code: 200 Content: message: "Registration successful"
Error response	Code: 422 UNPROCESSABLE ENTRY Content: message: "Registration data not correct"
Description	It allows a Client to register to the SafeStreets platform as an End User.

API1.2

Name ID	**End User Login**
Endpoint	*/enduser/login
Method	POST
URL parameters	
Data parameters	password: [alphanumeric] username: [alphanumeric]
Success response	Code: 200 Content: - message: "login successful" - accessToken: [alphanumeric]
Error response	Code: 401 UNAUTHORIZED Content: message: "wrong username or password"
Description	It allows a Client to login to the SafeStreets platform as an End User.

Once the user is logged in and got his token, we assume that it will be used to authenticate the user for all the subsequent requests.

API 1.3

Name ID	**Search Street's Violations**
Endpoint	*/enduser/streets/number
Method	GET
URL parameters	accessToken: [alphanumeric] streetAddress: [alphanumeric]
Data parameters	
Success response	Code: 200 Content: - message: "Street is found" - streetInfo: array <StreetInfo>
Error response	Code: 400 BAD REQUEST Content: message: "wrong address" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows an End User to see number and type of violations in a specific street

API1.4

Name ID	**Generate a report**
Endpoint	*/enduser/report
Method	POST
URL parameters	
Data parameters	accessToken: [alphanumeric] image:Image highlightedZone: array <HighlightedZone> time: Time position: Position violationType: [alphanumeric]
Success response	Code: 200 Content: message: "Report successful"
Error response	Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action" Code: 403 FORBITTEN Content: message: "image or time compromised" Code: 422 UNPROCESSABLE ENTITY Content: message: "license not found" Code: 404 NOT FOUND Content: message "license not in the DB"
Description	It allows an End User to generate a report

API1.5

Name ID	**Add licence plate**
Endpoint	*/enduser/report/license
Method	GET
URL parameters	accessToken: [alphanumeric] image:Image highlightedZone: array <HighlightedZone>
Data parameters	
Success response	Code: 200 Content: • message: "Report successful" • license: [LicensePlate]
Error response	Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action" Code: 404 NOT FOUND Content: message "image error"
Description	It allows an End User to send an highlighted image, and recognise if the image highlighted zone show a valid licensePlate

2.5.2 Authority User's REST APIs

API2.1

Name ID	**Authority Login**
Endpoint	*/authority/login
Method	POST
URL parameters	
Data parameters	password: [alphanumeric] code: [alphanumeric]
Success response	Code: 200 Content: - message: "login successful" - accessToken: [alphanumeric]
Error response	Code: 401 UNAUTHORIZED Content: message: "wrong code or password"
Description	It allows a Client to login to the SafeStreets platform as an Authority.

API 2.2

Name ID	**Search Street's Violations**
Endpoint	*/authority/streets/info
Method	GET
URL parameters	accessToken: [alphanumeric] streetAddress[alphanumeric]
Data parameters	
Success response	Code: 200 Content: - message: "Street is found" - streetInfo: array <StreetInfo>
Error response	Code: 400 BAD REQUEST Content: message: "wrong address" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows an Authority to see all the information about violations in a specific street

API 2.3

Name ID	**Get a vehicle's information**
Endpoint	*/authority/vehicle/info
Method	GET
URL parameters	accessToken: [alphanumeric] licensePlate: LicensePlate
Data parameters	
Success response	Code: 200 Content: - message: "Vehicle is found" - vehicleInfo: VehicleInfo
Error response	Code: 400 BAD REQUEST Content: message: "wrong licence plate" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows an Authority to see all the information about a vehicle registered in the system

API 2.4

Name ID	**Get all the vehicles' information**
Endpoint	*/authority/vehicle/all
Method	GET
URL parameters	accessToken: [alphanumeric]
Data parameters	
Success response	Code: 200 Content: - vehicles: array <VehicleInfo>
Error response	Code: 404 NOT FOUND Content: "no vehicles yet" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows an Authority to see all the information about all the vehicles registered in the system

API2.5

Name ID	**Get the new reports**
Endpoint	*/authority/newreports
Method	GET
URL parameters	accessToken: [alphanumeric]
Data parameters	
Success response	Code: 200 Content: reports: array <Report>
Error response	Code: 404 NOT FOUND Content: message: "new reports absence" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows and Authority User to see all the new reports

API2.6

Name ID	**Get report's information**
Endpoint	*/authority/newreports/info
Method	GET
URL parameters	accessToken: [alphanumeric], reportID[alphanumeric]
Data parameters	
Success response	Code: 200 Content: reportInfo: ReportInfo
Error response	Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows an Authority user to get all the information about a report

API2.7

Name ID	**Delete a report**
Endpoint	*/authority/newreports/delete
Method	DELETE
URL parameters	accessToken: [alphanumeric], reportID[alphanumeric]
Data parameters	
Success response	Code: 200 Content: message: "report deleted"
Error response	Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action" Code: 404 NOT FOUND Content: "report not found"
Description	It allows an Authority user to delete a report

API2.8

Name ID	**Generate a ticket**
Endpoint	*/authority/ticket
Method	POST
URL parameters	
Data parameters	accessToken: [alphanumeric] reportID: [alphanumeric] licensePlates: array <LicensePlate> infoViolation: InfoViolation
Success response	Code: 200 Content: message: "ticket generated"
Error response	Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action" Code: 404 NOT FOUND Content: "report not found"
Description	It allows an Authority user to generate a ticket

API 2.9

Name ID	**Get Global Efficiency, Authority**
Endpoint	*/authority/efficiency/global
Method	GET
URL parameters	accessToken: [alphanumeric]
Data parameters	
Success response	Code: 200 Content: - globalEfficiency: GlobalEfficiency
Error response	Code: 404 NOT FOUND Content: "Global Efficiency empty" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows an Authority to see all the information about Global Efficiency

API 2.10

Name ID	**Get Street Efficiency, Authority**
Endpoint	*/authority/efficiency/street
Method	GET
URL parameters	accessToken: [alphanumeric] streetAddress[alphanumeric]
Data parameters	
Success response	Code: 200 Content: - message: "Street is found" - streetEfficiencyInfo: StreetEfficiencyInfo
Error response	Code: 400 BAD REQUEST Content: message: "wrong address" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows an Authority to see all the information about Street Efficiency in a specific street

2.5.3 System Manager's REST APIs

API3.1

Name ID	**System Manager Login**
Endpoint	*/systmanager/login
Method	POST
URL parameters	
Data parameters	password: [alphanumeric] code: [alphanumeric]
Success response	Code: 200 Content: - message: "login successful" - accessToken: [alphanumeric]
Error response	Code: 401 UNAUTHORIZED Content: message: "wrong code or password"
Description	It allows a Client to login to the SafeStreets platform as a System Manager.

API 3.2

Name ID	**Add a new street in SafeStreets**
Endpoint	*/systmanager/maintenance/addstreet
Method	POST
URL parameters	
Data parameters	accessToken: [alphanumeric] street: Street
Success response	Code: 200 Content: message: "Street added"
Error response	Code: 400 BAD REQUEST Content: message: "Street already in the system" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows a System Manager to add a new street in the system

API 3.3

Name ID	**Add a new violation type in SafeStreets**
Endpoint	*/systmanager/maintenance/addtype
Method	POST
URL parameters	
Data parameters	accessToken: [alphanumeric] violationType[alphanumeric]
Success response	Code: 200 Content: message: "Type added"
Error response	Code: 400 BAD REQUEST Content: message: "Type already in the system" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows a System Manager to add a new violation type in the system

API 3.4

Name ID	**Delete a street in SafeStreets**
Endpoint	*/systmanager/maintenance/delstreet
Method	DELETE
URL parameters	accessToken: [alphanumeric] street: [Street]
Data parameters	
Success response	Code: 200 Content: message: "Street deleted"
Error response	Code: 400 BAD REQUEST Content: message: "Street not in the system" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows a System Manager to delete a street in the system

API 3.5

Name ID	**Get all the violation types in SafeStreets**
Endpoint	*/systmanager/types
Method	GET
URL parameters	accessToken: [alphanumeric]
Data parameters	
Success response	Code: 200 Content: type: array <Alphanumeric>
Error response	Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows a System Manager to delete a violation type in the system

API 3.6

Name ID	**Delete a violation type in SafeStreets**
Endpoint	*/systmanager/maintenance/deltype
Method	DELETE
URL parameters	accessToken: [alphanumeric] violationType:[alphanumeric]
Data parameters	
Success response	Code: 200 Content: message: "Type delete"
Error response	Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action" Code: 404 NOT FOUND Content: "type not found"
Description	It allows a System Manager to delete a violation type in the system

API 3.7

Name ID	**Get Global Efficiency, System Manager**
Endpoint	*/systmanager/efficiency/global
Method	GET
URL parameters	accessToken: [alphanumeric]
Data parameters	
Success response	Code: 200 Content: - globalEfficiency: GlobalEfficiency
Error response	Code: 404 NOT FOUND Content: "Global Efficiency empty" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows a System Manager to see all the information about Global Efficiency

API 3.8

Name ID	**Get Street Efficiency, System Manager**
Endpoint	*/systmanager/efficiency/street
Method	GET
URL parameters	accessToken: [alphanumeric] streetAddress[alphanumeric]
Data parameters	
Success response	Code: 200 Content: - message: "Street is found" - streetEfficiencyInfo: StreetEfficiencyInfo
Error response	Code: 400 BAD REQUEST Content: message: "wrong address" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows a System Manager to see all the information about Street Efficiency in a specific street

API 3.9

Name ID	**Add new special user**
Endpoint	*/systmanager/maintenance/adduser
Method	POST
URL parameters	
Data parameters	accessToken: [alphanumeric] password: [alphanumeric] typeUser:[alphanumeric]
Success response	Code: 200 Content: - message: "User added" - code: [alphanumeric]
Error response	Code: UNPROCESSABLE ENTRY Content: message: "Registration data not correct" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows a System Manager to add a new special user: a new System Manager or a new Authority

API 3.10

Name ID	**Delete a special user**
Endpoint	*/systmanager/maintenance/deluser
Method	DELETE
URL parameters	accessToken: [alphanumeric] cose: [alphanumeric]
Data parameters	
Success response	Code: 200 Content: message: "User delete"
Error response	Code: UNPROCESSABLE ENTRY Content: message: "Registration data not correct" Code: 401 UNAUTHORIZED Content: error: "User is not allowed to perform this action"
Description	It allows a System Manager to delete a special user: a System Manager or an Authority

2.6 Selected architectural styles and patterns:

2.6.1 Three-tier Client-Server architecture

Since the document presented by the stakeholders required a system where multiple users could interact, it was decided to use a Client-Server architecture, since it's a common used architecture and it's very efficient for this scope. It was decided to use a three-tier architecture because it would simplify the maintenance, the scalability and the availability of the system. Moreover, the Safestreets Database has to be synchronized with the Government Database. Separating the Application Logic and the Data in different nodes permits to separate the traffic. It would help the scalability because Application Logic servers or the Database servers can be duplicated as needed independently. Finally, maintenance is helped because servers can be upgraded independently, and the separation of duties and layers between the machines is clear.

2.6.2 Thin client

Another architectural style selected is thin client. Since many user of this system will be ordinary citizen, we can't assume the computing power and the disk space of their devices. A thin client is the solution to this problem. Having all the Application Logic in the Server leaves to handle only the Presentation to the User's devices. Moreover, this helps the security of the system, since all the information is processed only in the Server.

2.6.3 API REST

The communication between the Client and the Server follows the RESTful principles by using API REST. This permits to have a stateless communication between them. It greatly helps the Server in handling multiple Clients, because it doesn't need to hold memory of the past interaction with them. This, combined with the Event-Driven architecture style, improve the scalability of the system, and it reduces the resources needed.

2.6.4 Event-driven architecture

The Application server of the system uses an event-driven approach when handling the communication with the End Users. This is a very conservative memory-wise method. Using only one worker per CPU core, it permits a near optimal use of the CPU. This solution helps the parallelism and reduce the resource cost.

2.6.5 Observer Pattern

The Update component of the system located in the Data tier uses an Observer pattern in order to maintain the Safestreets Database and the Government one consistent. It observes both databases, and it uses their interfaces to update them as needed. For example, if a new ticket is generated in Safestreets, it push it in the Government Database. If the Government database adds new violations reported outside Safestreets, the Update component will update the Safestreets database conveniently.
Observer pattern was chosen because it's a very common pattern and many software libraries implement it.

2.7 Other Design Decision

2.7.1 Communication between Safestreets Database and Government Database

As seens in the deployment view and in the component view, the Safestreets database communicates directly to the Government database. This communication doesn't pass through the Application Logic tier. This was decided because it would lessen the traffic between database and application server, and it would maintain the separation between tiers. The Application Logic tier, in fact, shouldn't concern about the handle of Data.

2.7.2 Security Software as COTS

As it can be seen in the Component diagram, the Security Software is a COTS. This decision was made because it would be better to use State of Art technology to ensure that images are not altered. Using a COTS instead of creating it ourselves would eliminate the cost of upgrading the software to new technologies. However, it will need a deal with an external software house. If a COTS that

satisfy the needs of the system in terms of functionalities and cost can't be found, this component will need to be implement by ourselves.
A similar discourse could be done with License Plate Recognizer, but it was decided to realise it instead of buying a COTS because it is not that crucial that the image recognition algorithm is at State of Art. There is no risk of security in this case, in fact. However, it is a possible option to keep in mind.

3 User Interface Design

The User Interface is already showed in the RASD document. In this section the UX diagrams of the End User, Authority User and System Managers are showed in order to give a clear representation to the developer on the flow of the Interface during the interaction with the different users. Some mockups from the RASD are showed also here in order to give an idea of a possible implementation of the User Interface.

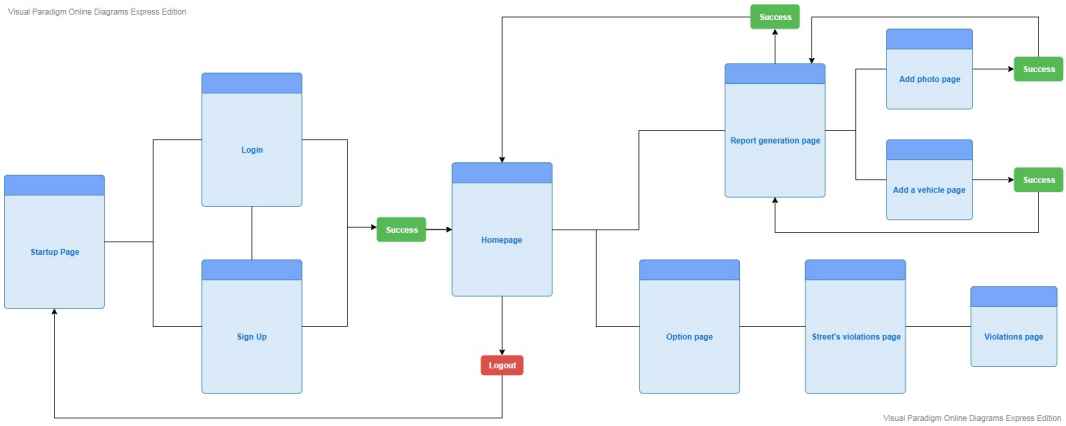

Figure 13: End User UX diagram

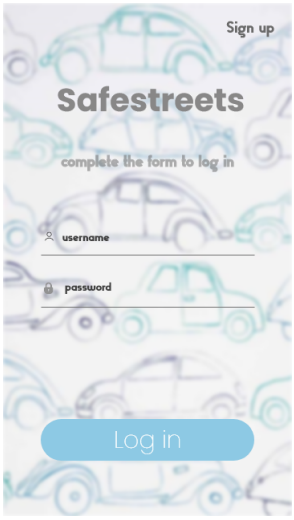

Figure 14: End User mockup

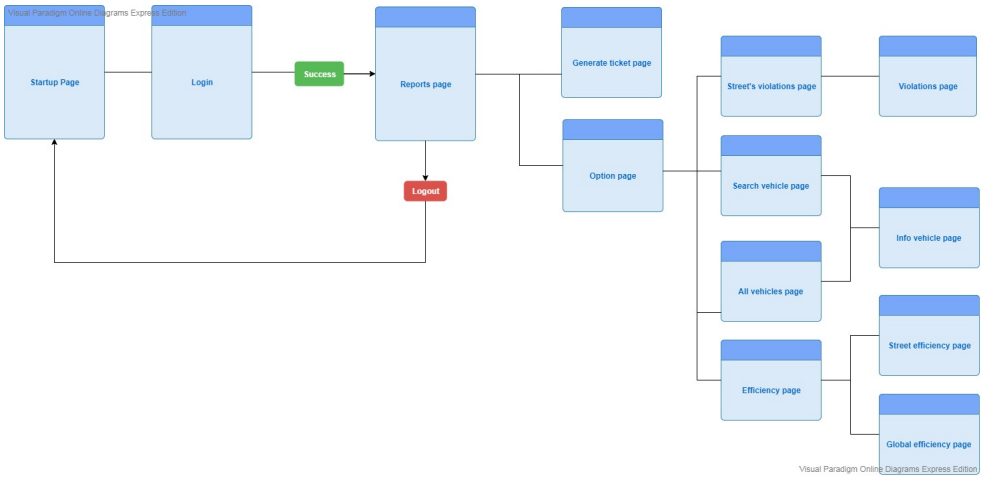

Figure 15: Authority User UX diagram

Figure 16: Authority User mockup

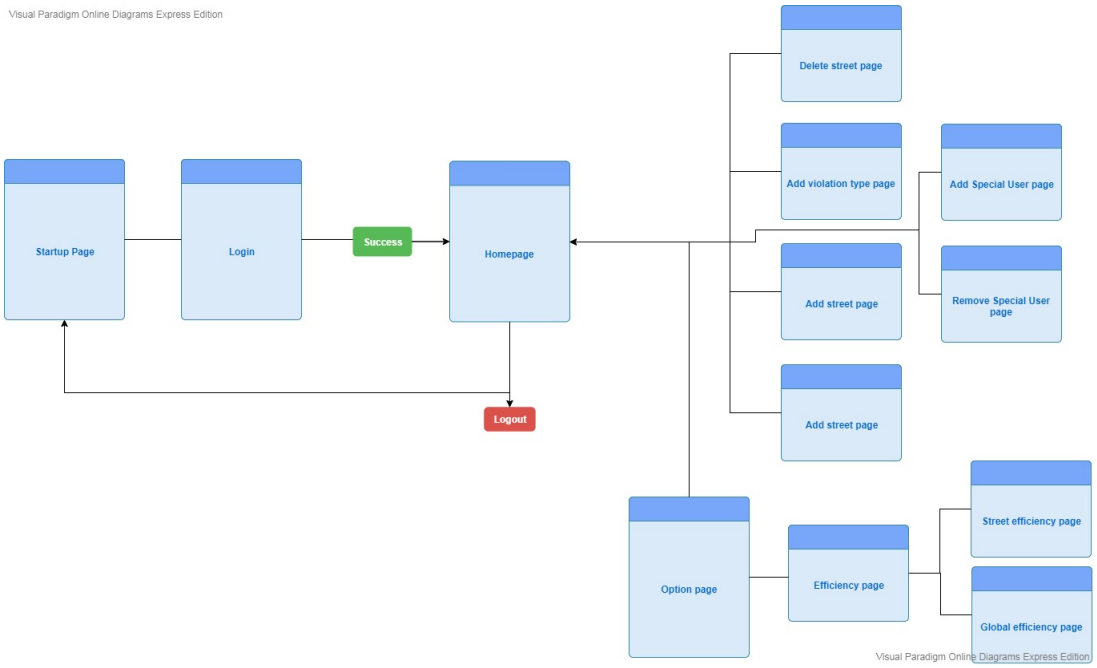

Figure 17: System Manager UX diagram

4 Requirements Traceability

End User Controller

- R.1 The system must provide a sign up mechanism to end users. During the registration the System will ask to provide credentials.
- R.2 The System must check in the sign up process if the Guest credentials are valid: username not already taken by another registered User, fiscal code in the right format and password with a minimum length for the end user.
- R.3 The system must be able to keep track of each user's data.
- R.4 The system must provide a login mechanism to users.

Authority User Controller

- R.3 The system must be able to keep track of each user's data.
- R.4 The system must provide a login mechanism to users.

System Manager Controller

- R.3 The system must be able to keep track of each user's data.
- R.4 The system must provide a login mechanism to users.

Report Controller

- R.5 The system must provide the end user with a way of sending a report.
- R.6 The system must be able to collect data from the user's devices.
- R.8 The system must provide the end user with a way of selecting the report's type.
- R.9 The system must provide the end user with a way of sending the license plate of a vehicle.

Security Controller

- R.7 The system must be able to guarantee that the information of the reports is not altered.

License Plate Recognizer

- R.9 The system must provide the end user with a way of sending the license plate of a vehicle.

Street Data Visualization Manager

- R.10 The system must be able to publish new data.

- R.11 The system must provide the user with a way of searching a street's information.

Vehicle Visualization Manager

- R.10 The system must be able to publish new data.
- R.12 The system must provide the Authority with a way of searching a vehicle's information.
- R.13 The system must provide the Authority with a way of seeing all the vehicles' information.

Report Validation Controller

- R.10 The system must be able to publish new data.
- R.14 The system must provide the Authority with a way of seeing a report's information.
- R.15 The system must provide the Authority with a way of delete a report.

Ticket Controller

- R.16 The system must provide the Authority with a way of selecting one or multiple license plate.
- R.17 The system must provide the Authority with a way of generate traffic tickets.

Efficiency Manager

- R.10 The system must be able to publish new data.
- R.18 The system must provide the Authority and System Manager with a way of see Global Efficiency.
- R.19 The system must provide the Authority and System Manager with a way of see Street Efficiency.

System Maintenance

- R.10 The system must be able to publish new data.
- R.20 The system must provide System Manager with a way of add a new street.
- R.21 The system must provide System Manager with a way of add a new violation type.
- R.22 The system must provide System Manager with a way of delete a street.

- R.23 The system must provide System Manager with a way of delete a violation type.
- R.24 The system must be able to restore violations' type data.
- R.25 The system must be able to restore streets' data .
- R.26 The system must provide System Manager with a way of add a new System Manager.
- R.27 The system must provide System Manager with a way of add a new Authority.
- R.28 The system must provide System Manager with a way of remove a System Manager.
- R.29 The system must provide System Manager with a way of remove an Authority.

Data Model

- R.3 The system must be able to keep track of each user's data.
- R.10 The system must be able to publish new data.

Update

- R.10 The system must be able to publish new data.

5 Implementation, Integration and test plan

In this section we propose a plan to implement and test the system. The design process explained below is **bottom-up**. The flow diagram shows in which order the components of the system should be implemented and tested in a graphical way. It is assumed that each component is unit tested during and after its implementation. Some conventions have been made in the diagram:

- **Frontend components** are the ones that implement the Presentation layer. They are colored in **green**. **Backend components** are the ones of the Application Logic and Data layers. They are colored respectively in **red** and **blue**.
- **End User components** are components that offers interfaces to implement the functionalities needed by End Users. They can be Frontend components or Backend components. For example, App presentation and Report Controller are End User components. The same is true for **Authority User components** and **System Manager components**. A component can be needed by more than one user. In that case, it is part of both. For example, and Efficiency manager is both an Authority User and System Manager component.
- **Yellow** rectangles in the diagram, with the exception of "RELEASE", are the **integration tests**.
- **Grey** rectangles represents **possible alpha version** to be validated by the nominated users. They are great opportunities to get a feedback by the users and to fix the project as needed before it is completed. Doing them is highly recommended, since it can be very expensive to fix the system in terms of time and money after the release.

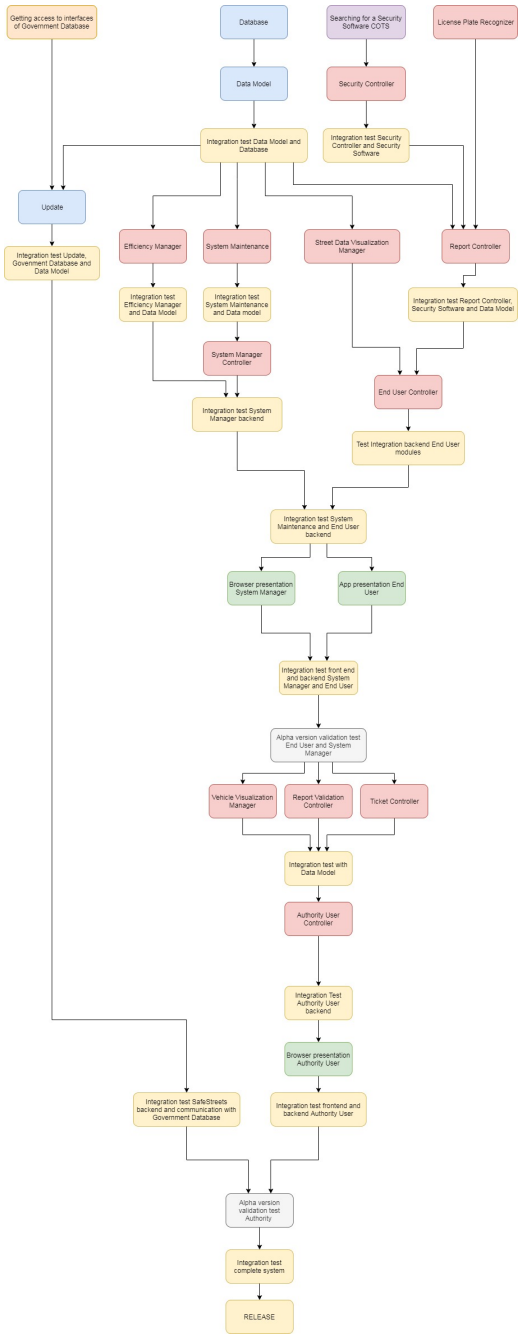

Figure 18: Implementation plan diagram

The dependencies of the various components are:

- **App presentation:** this component is the frontend of the End User, and it depends on all the End User backend. It should be implemented after the backend is completed
- **End User Controller:** this component depends on Data Model, Street Data Visualization Manager, Report Controller and Data Model to complete its task, and it offers methods to App presentation.
- **Street Data Visualization Manager:** this component offers methods to both End User Controller and Authority User Controller. This is a component where the **thread** technique could be applied. In fact, when we implement the End User backend, we could implement only the interface towards the End User Controller in this component, and later, when the Authority User backend is implemented, it is possible to implement the interface towards the Authority User Controller. This decision is left to who will implement the system, in order to adapt to delays or bottlenecks of the project. The component is dependent only on Data Model.
- **Report Controller:** this component depends on Data Model, License Plate Recognizer and Security Controller, and it's required by End User Controller.
- **License Plate Recognizer:** this component is independent, but it is required by Report Controller. It can also be acquired as a COTS, since it must implement a single algorithm of image recognition.
- **Security Controller:** this component depends on a Security Software, a COTS, and it is required by Report Controller.
- **Data Model:** this is the most critical and important component, since almost all of components depends on it. It depends on the database. It also communicate with Update, which is an adapter. Even if Data Model uses the interfaces of Update, Data Model should be implemented before.
- **Database:** this component is independent, but it should be implemented in respect of the needs of Data Model in matter of optimization of queries.
- **Update:** this components is an adapter between Data Model and the Government Database, so it should be implemented after Data Model is realised and the interfaces of the Government Database are obtained.
- **Browser Presentation - Authority Controller:** this component depends on the the Authority User backend components.
- **Authority User Controller:** this component depends on Data Model, Street Data Visualization Manager, Vehicle Visualization Manager, Report Validation Controller, Ticket Controller and Efficiency Manager. It is required by Browser Presentation - Authority Controller.

- **Vehicle Visualization Manager:** this component depends on the Data Model and it's required by Authority User Controller.
- **Report Validation Controller:** this component depends on the Data Model, and it's required by Authority User Controller.
- **Ticket Controller:** this component depends on the Data Model and it's required by Authority User Controller.
- **Efficiency Manager:** this component depends on the Data Model and it's required by Authority User Controller and System Manager Controller. It's not a critical component, since the system can work without it. It has a low priority in implementation in respect to other components.
- **Browser Presentation - System Manager:** this component depends on the the System Manager backend components.
- **System Manager Controller:** this component depends on Data Model, Efficiency Manager and System Maintenance. It is required by Browser Presentation - System Manager.
- **System Maintenance:** this components depends on Data Model and it is required by System Manager Controller. It's a critical component and the system can't function without it, so it has a high priority.

The first component that should be implemented and unit tested is Data Model, since many components depends on it. Its implementation define how the data is represented and how other components can add or retrieve it. However, its implementation must happen after the one of the database, since Data Model depends on it.
At the same time, other three activities can be done. The research for a COTS Security Software and the process to get the interfaces and permissions from the Government Database should be done as soon as possible. Not just because they are critical aspects of our system, but because they require communications and deals with external entities. Those process can be long and not predictable. It would be better to be safe in case of delays. Obviously, the interaction with these two components must be unit tested. If a COTS Security Software can't be found, it has to be implemented and unit tested by ourselves.
License Plate Recognizer is an independent component, instead, so it doesn't need to wait for Data Model. This software can be also a COTS component. If it is decided like that, it should be searched as soon as possible for the same reasons explained before for the Security Software.
After that, The first components to be implemented are the ones of End User and System Manager. System Manager is a type of user necessary in the system, otherwise it couldn't be possible to add more streets, street laws, Authority users and other System Managers. Obviously, the first System Manager must be implemented in the code, in order to be able to add other System Managers. Without End Users, there can't be reports. So, they are fundamental too.

However, the system can work in a simplified way without the Authority User, if we assume that the system can retrieve data from the Government database. This permits to validate the system with End Users and System Managers before implementing the Authority components.
After that, the components of Authority User must be implemented, tested and integrated in the system. It would be better to validate the system with Authority user at the end of this phase. After the integration test of the entire system, and a validation of the entire system if desired, the system can be released.

6 Effort Spent

Valeria Maria Fortina

Date	Section	Hours
28/11	Initial Brainstorming of the project, Components, Requirements Traceability	4h
01/12	Components	1h
02/12	Sequence Diagrams	6h
03/12	Api, Sequence Diagrams Runtime view	6h
04/12	Api, Sequence Diagrams Runtime view, RASD corrections, Introduction	7h
05/12	Requirements Traceability	1h
07/12	Sequence Diagrams corrections, Api corrections, components	5h
08/12	Corrections, revision	6h
09/12	RASD update, DD complete revision, corrections	7h
		Total: 43h

Alessio Galluccio

Date	Section	Hours
28/11	Initial Brainstorming of the project, Components, Overview	2h
30/11	Component diagram	2,5h
01/12	Component, deployment and interface diagrams	4,5h
02/12	Deployment and sequence diagrams	3,5h
04/12	Implementation plan diagram, correction sequence diagrams and component diagram	4,5h
06/12	Part 5 and corrections	5h
07/12	Part 5, architectural styles and other design decision, corrections	8h
08/12	Interface diagram, part 3, corrections	10h
09/12	RASD update, DD complete revision, corrections	7h
		Total: 47h

7 References