POLITECNICO
MILANO 1863

Dipartimento di Elettronica, Informazione e Bioingegneria

Design Document (DD)

SafeStreets

- v1.0 -

Authors:

Quacquarelli Sebastiano	945071
Ricchiuti Simone	945613
Sala Nicolò	945898

December 9th , 2019

Contents

1 Introduction

1.1 Purpose

The main aspects of SafeStreets system are presented in the RASD document (see References section). The purpose of this document is to describe in a more detailed way the technical aspects regarding architectural and design choices for SafeStreets components.

More precisely, the document presents:

- Overview of the high level architecture;
- The main components, their interfaces;
- The runtime behaviour;
- The design patterns;
- Additional details about user interface;
- A mapping of the requirements on the architecture's components;
- Implementation, integration and testing plan;

1.2 Scope

In this section is shortly summarised what is already defined in a more detailed way in the *Scope* section of the RASD document.
SafeStreets scope is to give the opportunity to report traffic violations by users and eventually to let them be managed by a competent authority in an easy way. Basically users can simply open the SafeStreets Application on their smartphones and immediately take a picture of the traffic violation to report. Including a few more information, the report is done and is ready to be sent to SafeStreets system.
On the other hand, authorities in collaboration with SafeStreets can check violation reports on their SafeStreets Authority Edition (desktop dedicated application).
Evaluating reports, authorities can eventually define them consistent or not and, if consistent evaluated, they fine the offender reported.
These ones are the basic functionalities offered by SafeStreets but also other ones are offered to individuals:

- Users can browse the map to know statistics about traffic violations sent to SafeStreets and their status (confirmed or not by authorities);
- Authorities can browse map such as users but including access also to sensitive information such as license plates included in reports;

- Authorities can access to advanced statistics such as filtering results to return the most egregious offender;
- Authorities can also send to SafeStreets some information about accidents in order to enrich SafeStreets database. It is important because authorities can receive suggestions from SafeStreets system to improve the security of streets according to violations and accidents occurred in a defined area.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Operative area**: the geographic area in which an authority retains control and actually works.
- **Registered authority**: an authority that decided to participate in SafeStreets initiative and that installed in its head office SafeStreets AE, a particular version of the application that provides some features not available to common users (for example, the possibility to rate the traffic violation reports collected by SafeStreets).
- **Report image**: it is the image acquired through the SafeStreets app. It is obligatorily associated with a valid report.
- **Report timeout**: it is the timeout started when SafeStreets app acquires the report image. If the user doesn't complete the completion of the report by the deadline of this timeout, the report is considered invalid.
- **SafeStreets**: it is the crowdfunding app subject of this document.
- **SafeStreets application**: it is the application used by users, installed on their smartphone.
- **SafeStreets Authority Edition**: it is the application used by authorities, installed on computers in their station.
- **SafeStreets Client**: a generic way to point out the SafeStreets App or the SafeStreets Authority Edition.
- **Traffic violation report**: it is the message that SafeStreets collects through its app from users who want to report an alleged violation. It is often abbreviated as "report".
- **Individual**: A generic user or authority.

1.3.2 Acronyms

- **AE** : SafeStreets Authority Edition.
- **GUI** : Graphic User Interface.

- **HTTP** : Hypertext Transfer Protocol.
- **MVC** : Model-View-Controller design pattern.
- **OCR** : Optical Character Recognition.
- **RASD** : Requirement Analysis and Specification Document.
- **REST** : REpresentational State Transfer.
- **URI** : Universal Resource Identifier.

1.3.3 Abbreviations

- [DB]: Database.
- [Rn]: n^{th} functional requirement.

1.4 Revision history

Version	Last update	Comments
1.0	9^{th} December, 2019	

Table 1: Revision history

1.5 Document structure

In this part it is shown how the document has been divided. For each chapter, is given a short description:

- *Chapter 1* gives an introduction of the design document. It contains the purpose of the document and the scope of the SafeStreets system, as well as some abbreviations in order to provide a better understanding of the document to the reader.
- *Chapter 2* is the core section of the design document and it deals with the architectural design of the application. It gives an overview of the architecture and it also contains the most relevant architecture views:
  - High-level components and their interactions;
  - Component view;
  - Deployment view;
  - Component interfaces;
  - Runtime view.

Some of the used architectural designs and designs patterns are also presented here, with an explanation of each one and the purpose of their usage.

- *Chapter 3* refers to the mockups already presented in the RASD document including some more detailed versions of them such as interaction diagrams.
- *Chapter 4* explains how the requirements that were defined in the RASD are now assigned to the design elements defined in this document.
- *Chapter 5* presents the implementation, integration and test plan. It specifies how the different components of the application are integrated with each other and the testing strategy.
- *Chapter 6* shows the effort spent by each group member while working on this document.
- *Chapter 7* simply includes the references.

2 Architectural design

2.1 Overview

In this section is defined the high-level architecture of SafeStreets system.
Figure 2.1 represents the high level interaction between the main parts of SafeStreets' structure.

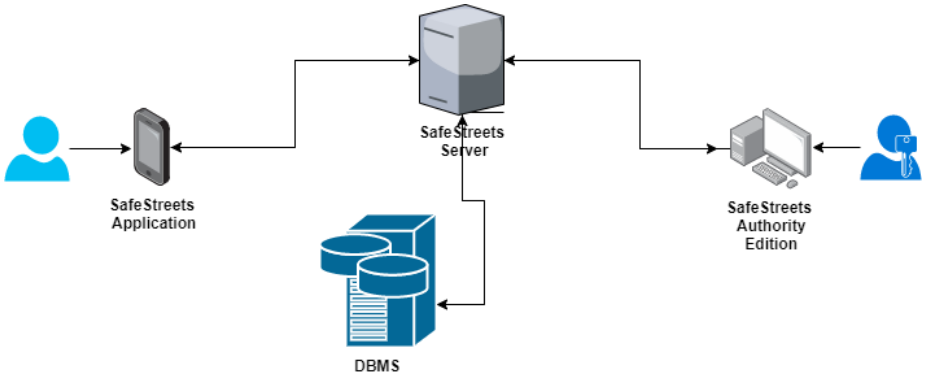

Figure 2.1: High-Level Interaction diagram

The figure describes how individuals interact with SafeStreets. Only one user is represented on the left and only one authority on the right, but this is only made to define clearly interactions. The real system is actually composed by multiple individuals.
Basically everyone can download the SafeStreets application on his smartphone and can be an *user*. On the other hand the Authority Edition is distributed by SafeStreets only to certificated *authorities* that wants to collaborate with it.

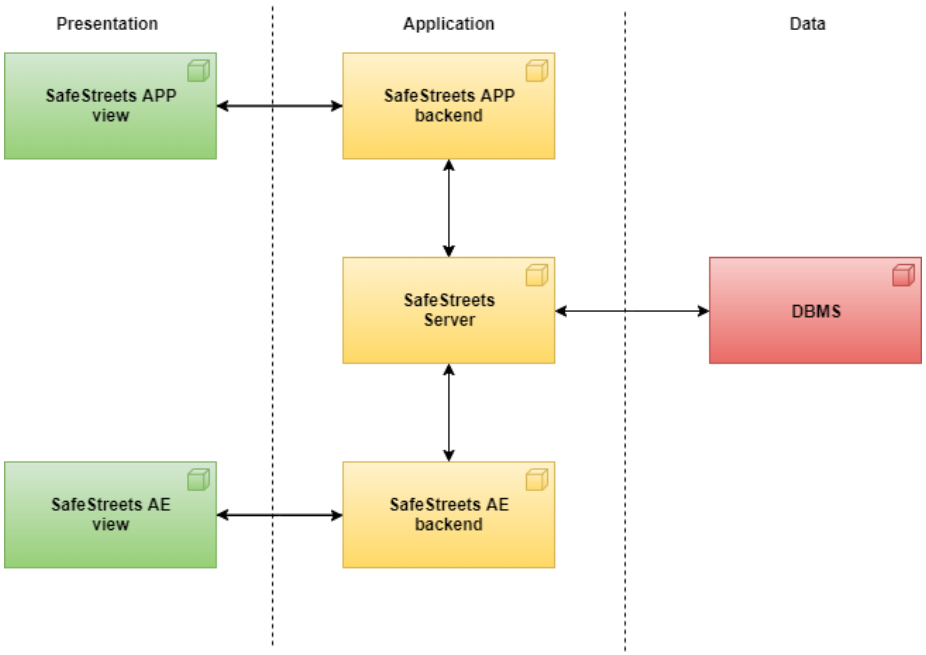

Figure 2.2: High-Level layers

In figure 2.2 are shown the three software logic layers by which the system is defined.

The *Presentation layer* contains the presentation logic of the system. It includes the two views addressed to different kind of SafeStreets clients.
The SafeStreets App is addressed to everyone that is interested in SafeStreets and uses its services through the smartphone application.
The SafeStreets Authority Edition (AE) is addressed to certified authorities collaborating with SafeStreets that have received from SafeStreets the desktop application.

The *Application layer* contains the application logic of the system. Basically it is the layer in which all functionalities are implemented.
This layer contains the SafeStreets App back-end, the SafeStreets AE back-end and the SafeStreets Server. The two back-ends communicate only with the server. This is important because only the server can exchange messages with the database system of the data layer, ensuring an highest level of security not allowing individuals to access directly to sensitive data.

The *Data layer* simply contains the Data Management System, that collects and execute queries from relational databases. All the queries results are sent only to the SafeStreets server.

2.2 Component view

Figure 2.3 represents all the components of SafeStreets ecosystem.
The diagram focuses on the applicative layer: the presentation layer is described in the section "User interfaces" of the RASD and deepened in section 3 of this document.
The diagram underlines that the clients, SafeStreets app and SafeStreets AE, are not thin: they include part of the application logic.

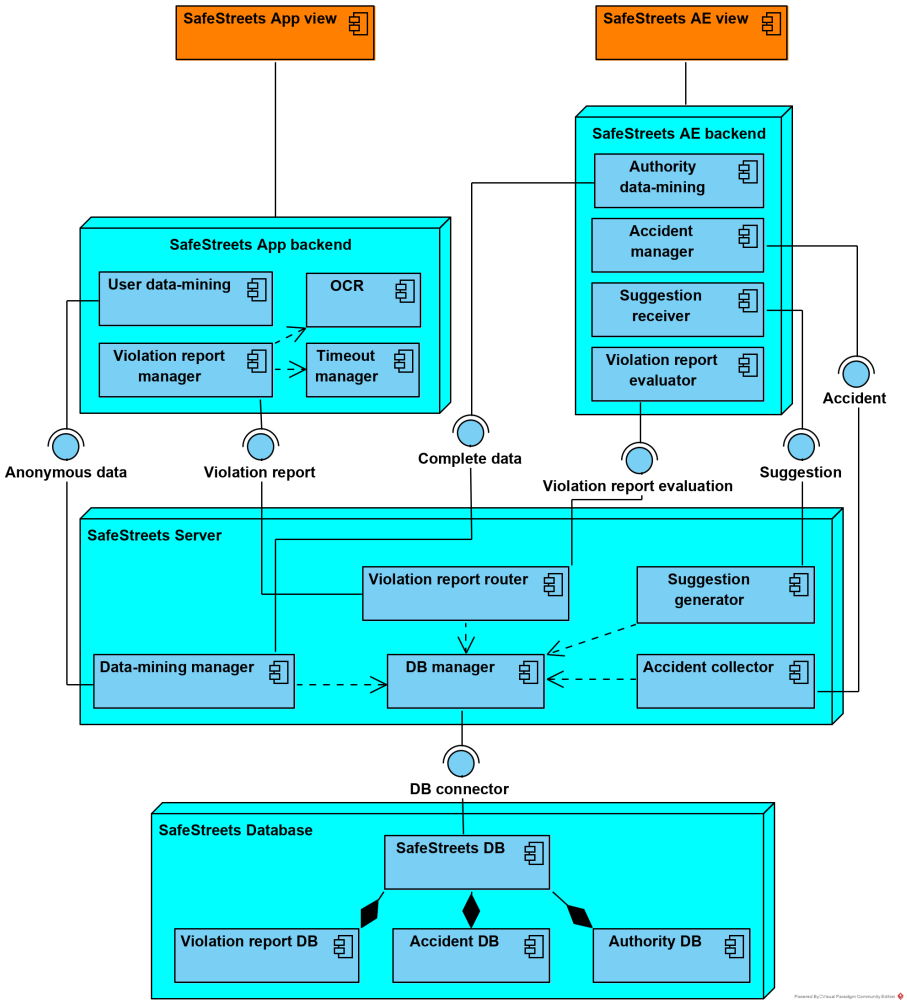

Figure 2.3: Component diagram

For each component of each node, a description of its role is given.

2.2.1 SafeStreets App components

- **Violation report manager**
  It deals with the management of a request for a violation report. First of all, it manages the acquisition of the reporting image: it automatically retrieves information about date, time and position. Afterwards, it forwards the image to the license plate recognition algorithm (OCR). If the algorithm returns a valid license plate and the user confirms the request by the deadline of the report timeout, then the violation report manager sends such report to SafeStreets Server through the "violation report" interface.

- **User data-mining**
  This component provides data-mining and statistical survey capabilities to SafeStreets App. Remember that SafeStreets App doesn't receive information considered private from the server, in particular it doesn't receive the license plates associated to violations or accidents. The component retrieves information from the server through the "anonymous data" interface.

- **OCR**
  This is the software used in order to retrieve a valid license plate from the reporting image. When it receives an image from the violation report manager, it looks for a valid license plate into the image, then returns an error message or the license plate found.

- **Timeout manager**
  It manages the report timeout. When the user starts a new violation report request, a timeout starts: if the timeout ends, the timeout manager sends a message to the violation report manager in order to cancel the current violation report request.

2.2.2 SafeStreets AE components

- **Violation report evaluator**
  This component receives violation reports from the server, via the "violation report" interface. After that, it manages the evaluation of the same by the authority. Once the evaluation has taken place, the violation report evaluator sends the result to the server through the aforementioned interface.

- **Authority data-mining**
  This component provides data mining and statistical survey capabilities to SafeStreets AE. Remember that SafeStreets AE can also retrieve information about the license plates linked to accidents and violations. The

component retrieves information from the server through the "complete data" interface.

- **Suggestion receiver**
  It waits for suggestions from the SafeStreets Server, listening the "suggestion" interface. Then it forwards the information to the view.

- **Accident manager**
  It manages the forwarding of incident-related information from an authority to SafeStreets Server. It ensures that the reporting of the accident is accompanied by all the data deemed necessary (see RASD, 3.2.1 for more details), after which it forwards the same to the server through the "accident" interface.

2.2.3 SafeStreets Server components

- **Violation report router**
  It passes violation reports received from users to the database. Also, it routes them to the authorities through the "violation report evaluation" interface, so they can be evaluated. The router follows a precise routing policy: it reads the position associated with the violation report, after which it consults the authority database: if there is an authority with has "municipal" jurisdiction for that area, the report is forwarded to it. Otherwise, an authority with "provincial" competence is sought for that area. If this is not the case, an authority with "state" jurisdiction is sought. If no authority can handle the validation of the report, it is not forwarded to any authority.

- **Data-mining manager**
  This component handles data requests from SafeStreets clients. In particular, it deals with extracting the data requested from the SafeStreets database, through the DB manager, filtering such data according to the authorizations of visibility available to the applicant. If the applicant is an authority, it can view all the data; if you are a user, the license and image information is deleted before sending it through the "anonymous data" interface.

- **Suggestion generator**
  When SafeStreets Server receives a new confirmation of violation or a new incident report, it is also received by this component, which compares the newly obtained data with those already saved in the database. Depending on a table of which it is available, it can generate advice to be sent to the authorities in order to avoid the repetition of certain dangerous situations, through the "suggestions" interface.

- **Accident collector**
  This component listens for new incident reports through the "accident" interface. The correctness of these reports has already been confirmed by

SafeStreets AE, so it is only necessary to send the new data to the DB manager, so that they are saved in the database.

- **DB manager**
  It manages the interaction between SafeStreets Server and SafeStreets Database. All the components of the server that wants to interact with the database has to pass through this component. This means that in the event of updates to the structure or operation of the database, only the DB manager may need to be updated, guaranteeing better independence between the server and the database.

2.2.4 SafeStreets Database components

The database stores all the information used by SafeStreets ecosystem. It is composed by three sub-databases, whose role can be easily understood from their name: **Violation report DB**, **Accident DB** and **Authority DB**.

2.3 Deployment view

In this section are defined the devices and the environments in which the three layers defined in the chapter 2.1 are concretely realized.

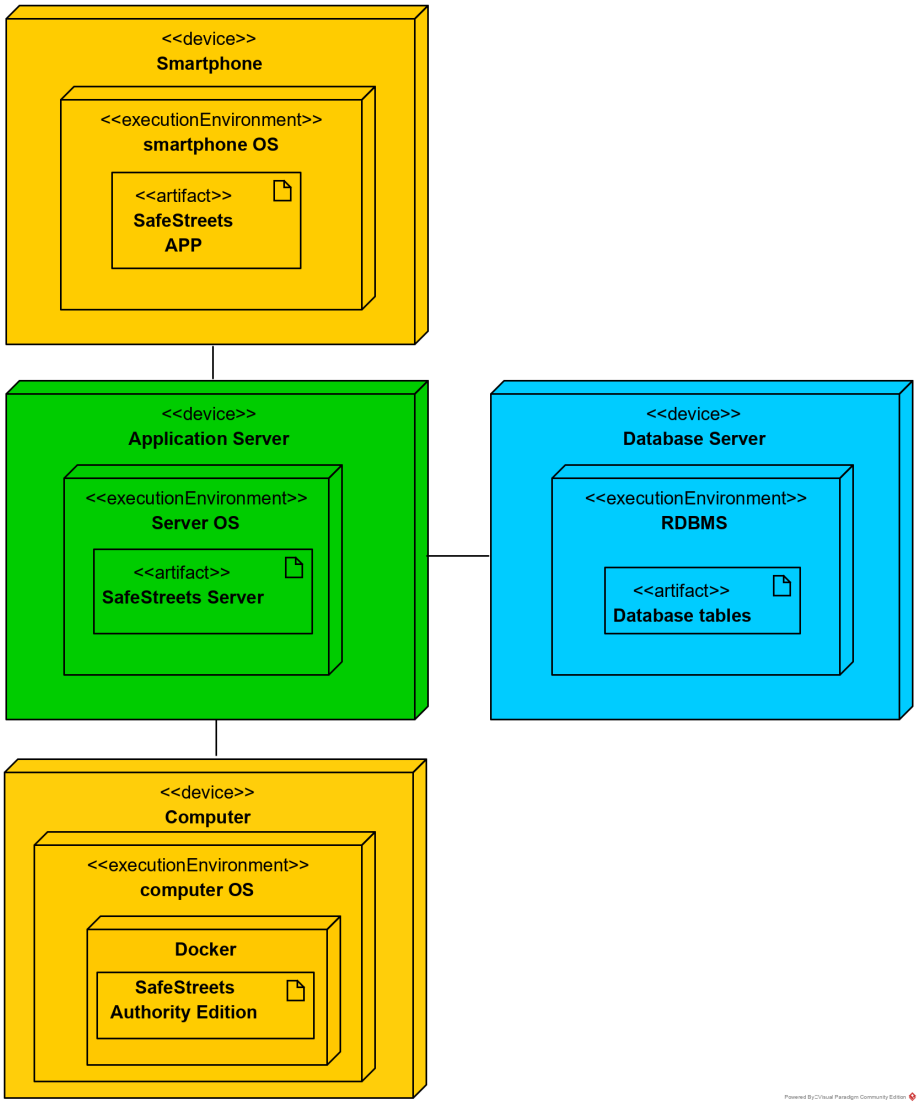

Figure 2.4: Deployment view

SafeStreets App is executed on smartphones and the application is available for the two most popular mobile operative systems: iOS and Android. The smartphone is the device in which both the SafeStreets App view and the SafeStreets App back-end are placed, so it contains the first two logic layers

that are the presentation and the application one.
The smartphone can communicate with SafeStreets Server to execute all the SafeStreets services such as sending a report or browsing through statistics.
SafeStreets AE is distributed only to authorities that collaborate with SafeStreets. In this context, the application is executable only on computers located in authorities' stations. This type of application is distributed into a (Docker container). In this way, the app can be installed on any operating system (more precisely, on any OS for which Docker is available) and the developers of SafeStreets doesn't have to care about the OS on which the app will be executed.
The reason why the AE is different from the simpler App is that authorities are allowed to make additional and more complete requests to the system and so, a desktop interface and a powerful hardware is better to manage all these requests and to make longer working sessions.

SafeStreets Server is installed on an application Server using preferably a Linux operative system. The Docker middleware is not exploited here, so to have better performance. Moreover, for the server, the possibility of supporting multiple OS is not as relevant as for SafeStreets AE. The server has to manage and forward all the requests in the system and it represent the only way for individuals to retrieve information from the SafeStreets database.

The Database Management System is installed on a non-specific type of server. One possible suggestion is to define it in MongoDB that provides a several number of advantage in terms of security, performance and scalability.
Other kind of DBMS are also allowed. The key aspect is that the database structure must be relational to satisfy the functionalities required by the system that need a fixed data structure to return specific kind of information or it needs to aggregate related information to make the server able to generate suggestions.

2.4 Component interfaces

This section describes the methods exposed by each interface that allows the communication between SafeStreets Server and the clients. These interfaces were already briefly shown in the component diagram of figure 2.3.
Information exchanged through these interfaces is standardized with respect to the following class diagram.

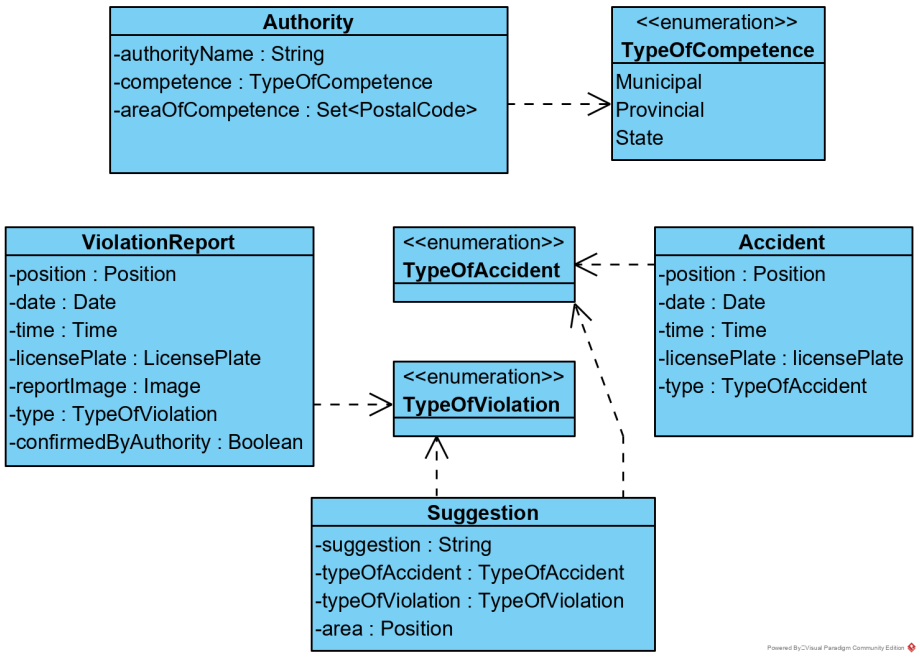

Figure 2.5: Class diagram

The class diagram represents only the classes considered fundamental to understand the operation of SafeStreets. In any case, the classes referred to without providing an explicit declaration - for example the "LicensePlate" class - have a name that effectively expresses their meaning.

A list of the interfaces follows. For each interface, all the methods it displays are pointed out. For each method, the necessary input parameters and outputs are indicated.
Every interface is exposed by the server to the clients. Therefore, the names of the methods have been chosen from the point of view of the server.

- **Anonymous data**
  Opportunely combining calls to the following two methods, SafeStreets

App can retrieve all the information then used to do data-mining or to show statistics. Notice that SafeStreets App will receive information that doesn't include images or license plates, so it is not possible, being useless, to specify these parameters in the method calls.

- getCompleteViolationReports

Input	Output
positions: Set(Position) dates: Set(Date) times: Set(Time) types: Set(TypeOfViolation) onlyConfirmed: Boolean	requestedViolationReports: Set(ViolationReport)

- getCompleteAccidents

Input	Output
positions: set(Position) dates: set(Date) times: set(Time) types: Set(TypeOfAccident)	requestedAccidents: Set(Accident)

- **Violation report**
  This interface exposes only one method that SafeStreets App uses to send a new violation report to the server. The correctness and the completeness of the violation report has already been checked by the app, so the server doesn't have to check it again and to return something as output.

  - receiveViolationReport

Input	Output
newViolationReport: ViolationReport	void

- **Complete data**
  Combining calls to the following methods, SafeStreets AE can retrieve all the information used to do data-mining or show statistics to authorities. The information received also contains license plates and images.

  - getCompleteViolationReports

Input	Output
positions: Set(Position) dates: Set(Date) times: Set(Time) types: Set(TypeOfViolation) licencePlates: Set(LicensePlate) onlyConfirmed: Boolean	requestedViolationReports: Set(ViolationReport)

– getCompleteAccidents

Input	Output
positions: set(Position) dates: set(Date) times: set(Time) licensePlates: set(LicensePlate) types: Set(TypeOfAccident)	requestedAccidents: Set(Accident)

- **Violation report evaluation**
  The interface is used to send violation reports to evaluate to an authority, and also by the authority to send the result of an evaluation.

  – sendViolationReportsToEvaluate

Input	Output
destinationAuthority: Authority ViolationReportTOEvaluate: ViolationReport	void

  – receiveEvaluation

Input	Output
evaluatedViolationReport: ViolationReport evaluation: Boolean	void

- **Suggestion**
  This interface is used by SafeStreets Server to send suggestions to authorities.

  – sendSuggestions

Input	Output
destinationAuthority: Authority suggestions: Set(Suggestion)	void

- **Accident**
  This interface is used by an authority to send information about an accident to SafeStreets Server.

  – receiveAccidents

Input	Output
newAccident: Accident	void

2.5 Runtime view

In the following section, sequence diagrams for SafeStreets are shown.

Figure 2.6 shows a detailed interaction between a user and SafeStreets system, through SafeStreets App, using the violation report functionality.

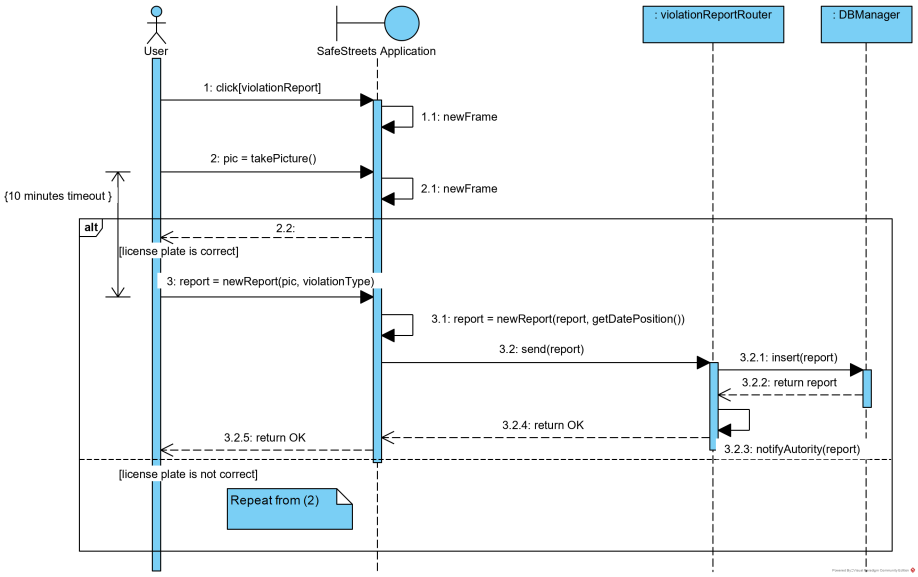

Figure 2.6: Sequence diagram for violation reports in SafeStreets AE

After clicking the apposite menu, the application change its frame and let the user take a photo of traffic violation. Once completed, a countdown of 10 minutes starts: this guarantees that a report could represent an actual violation, so it's not possible neither taking a photo and compile report after a long time, nor upload any photo. SafeStreets Application forces the user to take a photo of violation only through the user interface and in less than 10 minutes.
Now, two scenarios can occur: the OCR component of SafeStreets App (described in Figure 2.3 and in 2.2.1) recognizes the right license plate or not.
If the license plate read is correct, then the application let the user compile the violation report with remaining information and, before sending it, retrieves data as position and date. The **:violationReportRouter** forward the report coming from the application to **:DBManager**. This last component stores the report (in pending status) and let the **:violationReportRouter** notify the authority which operates in the area where the violation occurred. If license plate is either not correct or not detected, then user can repeat the procedure and the report timeout is reset.

Authorities evaluate reports as follows in Figure 2.7.
:violationReportRouter forwards (asynchronously) each report to the right authority. If any reports are retrieved, an authority for each of them can do evaluation, based on photo and information.
Two scenarios have to be managed: if the authority establish that a report is valid ("confirmed"), then **:violationReportRouter** queries an update of report status to **:DBManager**; otherwise, if not valid, report should be discarded, so the query asks to delete that report.

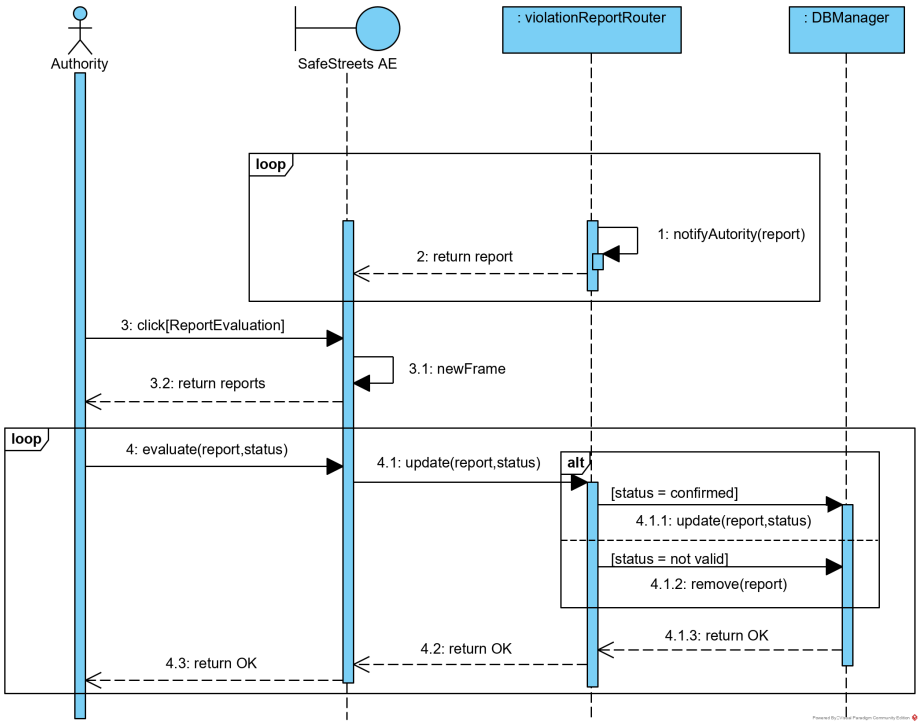

Figure 2.7: Sequence diagram for report evaluation in SafeStreets AE

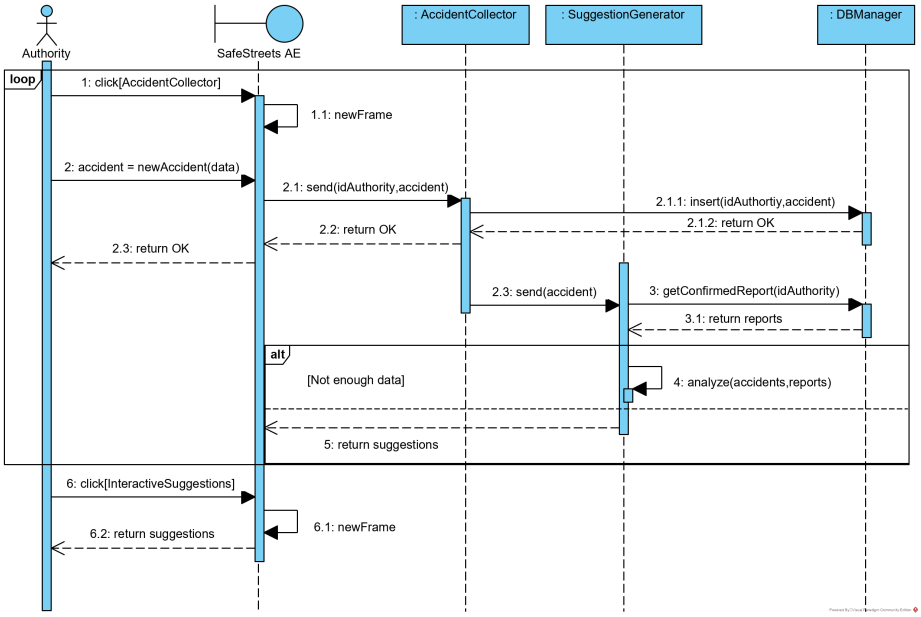

Figure 2.8: Sequence diagram for interactive suggestions in SafeStreets AE

A more complex interaction between authorities and SafeStreets system is given in Figure 2.8. An authority can share with SafeStreets system its data about accidents occurred in its operative area, using the apposite menu. The **:AccidentCollector** component forwards the accident data from application to **:DBManager**, which will insert them onto database.
When new suggestions are generated from **:SuggestionGenerator**, after accidents and reports analysis, that one component forwards (asynchronously) suggestions to the right authority.
Once authorities get a notify about new suggestions, they can select the appropriate menu and read them.

2.6 Selected architectural styles and patterns

2.6.1 Model-View-Controller (MVC)

In order to obtain a clear division between data, user interfaces and software logics, a Model–View–Controller (**MVC**) pattern has been used. MVC is a software design pattern commonly used for developing user interfaces which divides the related program logic into three interconnected elements:

- *Model*: the central component of the pattern. It represents the application's data structure, independent from the user interface. It directly manages the data of the application.
- *View*: any form of data representation.
- *Controller*: accepts input and converts it to commands for the the model or the view.

One of the biggest advantages of MVC is the possibility to have multiple views of the same information. Thinking about SafeStreets: both users and authorities have their own GUI and different level of details about data. This division of concerns makes the application easier to test and maintain: requests first arrive at the controller (or some of them) which binds the model with the corresponding view. In fact, another advantage is the possibility of parallelize application development among workers, such that groups can focus on a fixed part, i.e. model, view and controller.

Focusing more on SafeStreets, the MVC division is also observable in Figure 2.3. The back-end of each application, both client and server side, is part of the *controller*; SafeStreets user interfaces represent the *view* and they are just a mean to let users and authorities to communicate with the *model*, information stored in SafeStreets databases.

Follows an high-level representation of SafeStreets architecture.

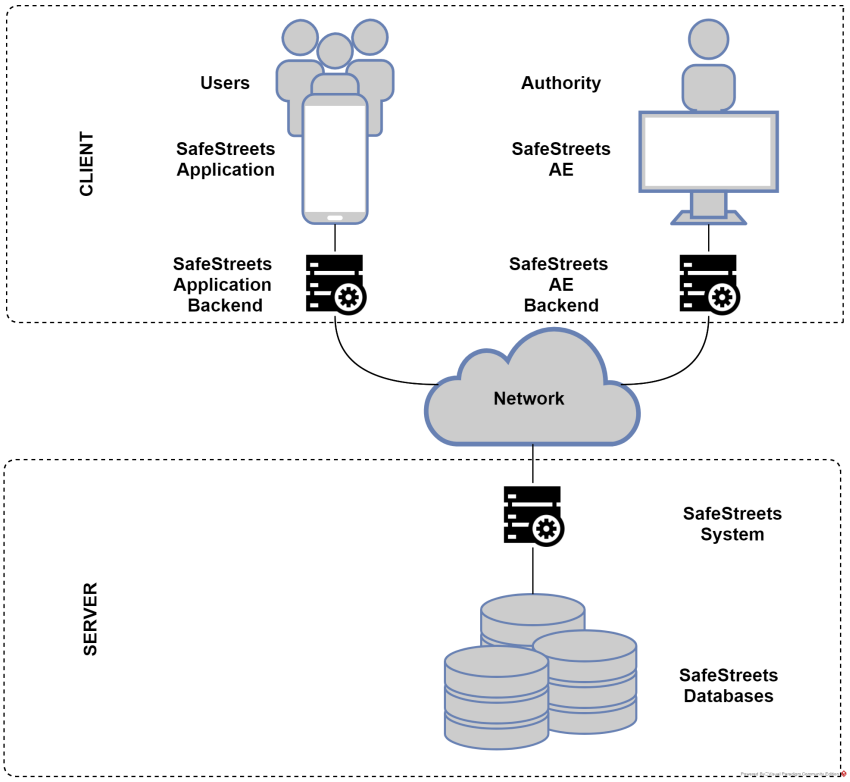

Figure 2.9: High-level representation of SafeStreets architecture

Figure 2.9 gives the idea of two-tiered architecture on which SafeStreets is based: clearly, clients are not *thin*, because they don't have only a presentation layer in their application. In fact, with *fat* or *thick* clients there is no need for continuous server communications (e.g. violation reporting for users involve locally an OCR algorithm to verify a correct reading of license plate, without contacting the server generating more workload).

2.6.2 RESTful web services

SafeStreets system doesn't need to exchange, with individuals, complex objects or computations. Basically, there's an exchange of simple objects, easy to format (e.g. in strings, text, file).
In this way, SafeStreets retrieves requests from individuals and returns responses via HTTP, following **REST** principles.
REST is a software architectural style that defines a set of constraints to be used for creating Web services, in order to provide interoperability between systems on the Internet.

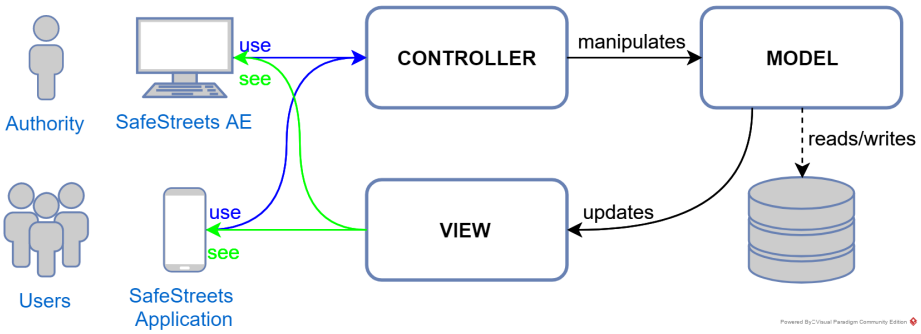

Figure 2.10: RESTful web services

RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of *stateless* operations: the client-server communication is constrained by no client context being stored on the server between requests. Each request from any client contains all the information necessary to service the request and the session state is held in the client.

A resource is an item of interest, exposed to the clients. In SafeStreets context, examples of resources can be violation reports, accidents or suggestions for authorities.
Every resource is referred in requests using a *URI*, a string of characters that unambiguously identifies the resource.

3 User Interface Design

To give an approximate idea of how the interfaces of the application should appear, some mockups, both for SafeStreets Application and SafeStreets AE, have been given in RASD [Section 3.1.1].
In order to express the relations among pictures in RASD [3.1.1], here it will be illustrated a pair of UX Diagrams.

Functionalities have been represented with different colours, while "Welcome Screen" and "Main Menu" (in purple) are basically the first graphical approach to the app, in particular "Main Menu" contains the menu voices at its right. Each functionality has an own sub-flow: black dotted lines represent optional navigation from a screen to another one; blue dotted lines simply refers to the possibility to go back through screens until "Main Menu" is reached.

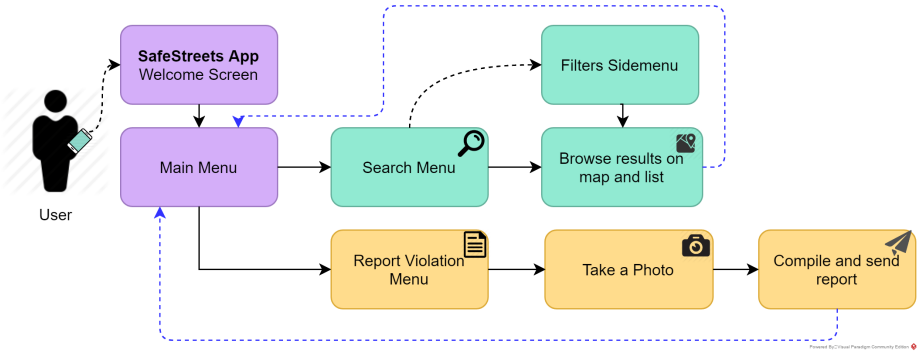

Figure 3.11: User Experience diagram for SafeStreets App

Figure 3.11 shows the possibilities given to users, in order to navigate in SafeStreets application. A user can, for instance, select the "Search Menu" to browse all traffic violations reported in an area, showed in a map and listed together in a list. Moreover, optionally, a user can improve its research swiping right a side menu in which he can select some filters: the result of research filtered will be shown as the previous modalities, i.e. in the map with a list.

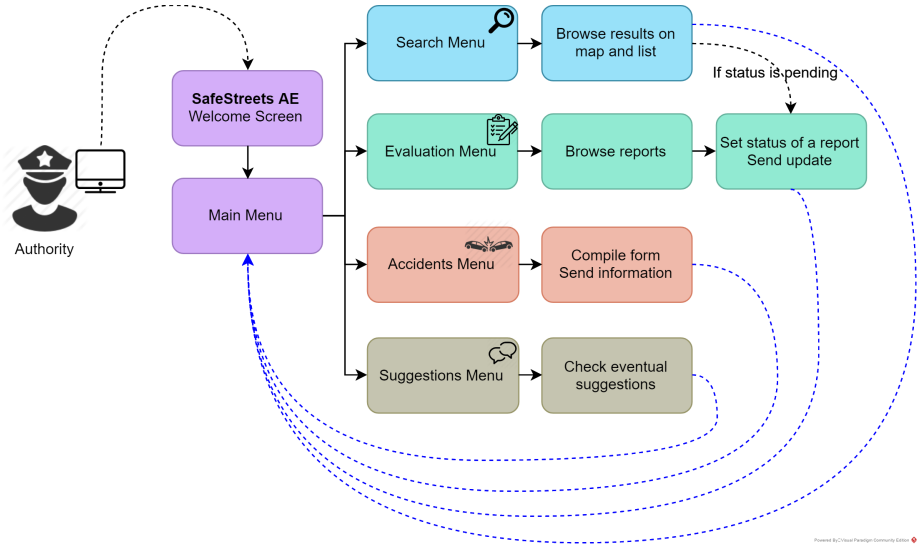

Figure 3.12: User Experience diagram for SafeStreets AE

Figure 3.12 is the user interface flow, designed for authorities who use SafeStreets AE.
Similarly as in figure 3.11, here is shown how authorities can explore the offered features. In particular, there's an optional flow which jumps from a result of a research to the "Evaluation" functionality: when an authority browse in a map (using "Search Menu") all traffic violations reported, it can both read information about a particular report and go to evaluation of the same report if and only if its state is pending. In this case, authorities can immediately decide if a report is correct or not valid.

4 Requirements Traceability

In this section is shown how the requirements specified in RASD are mapped to the design components defined in this document.

[R1] The user report must be of one of the type of violation defined by SafeStreets.

- **Violation Report Manager**

[R2] The report image must show the license plate of the offender.

- **Violation Report Manager**

[R3] The user must insert the picture of the violation.

- **Violation Report Manager**

[R4] The user must complete the report before the expiration of the report timeout.

- **Violation Report Manager**
- **Timeout Manager**

[R5] The OCR algorithm must detect one license plate in the report image.

- **Violation Report Manager**
- **OCR**

[R6] The APP must detect automatically position and date.

- **Violation Report Manager**

[R7] Every violation report sent to server must be filled in all its fields.

- **Violation Report Manager**
- **Violation Report Router**

[R8] The SafeStreets client must show all and only the information required by the individual.

- **User data-mining**
- **Authority data-mining**
- **Data-mining Manager**

[R9] Every individual can browse the map.

- **User data-mining**
- **Authority data-mining**

[R10] Every individual can see statistics in map.

- **User data-mining**
- **Authority data-mining**

[R11] Every individual can define filters for the research.

- **User data-mining**
- **Authority data-mining**
- **Data-mining Manager**

[R12] Every authority can see advanced statistics in map (including sensitive information).

- **Authority data-mining**
- **Data-mining Manager**

[R13] Every authority can define advanced filters for the research (including sensitive information).

- **Authority data-mining**
- **Data-mining Manager**

[R14] The information about accidents must be of one of the type defined by SafeStreets.

- **Accident Manager**

[R15] Information about accidents occurred must include all the information required by SafeStreets.

- **Accident Manager**
- **Accident Collector**

[R16] Every accident reported by a registered authority is consistently saved into the SafeStreets database.

- **Accident Collector**
- **DB Manager**

[R17] Every violation reported to SafeStreets is consistently saved into SafeStreets database.

- **Violation Report Router**
- **DB Manager**

[R18] SafeStreets must be able to generate suggestions, based on accidents and confirmed violation reports.

- **DB Manager**
- **Suggestion Generator**

[R19] Every authority must receive suggestions only referred to its operative area.

- **Suggestion Generator**
- **Suggestion Receiver**

[R20] An authority receives a report if and only if that report is related to a position covered by that authority (i.e. the postal code of the position of the violation report is assigned to that authority)

- **Violation Report Router**
- **Violation Report Evaluator**

[R21] When a registered authority confirms a violation report, that confirmation is consistently stored into the server.

- **Violation Report Router**
- **Violation Report Evaluator**
- **DB Manager**

[R22] A violation report rejected by a registered authority is removed from SafeStreets database.

- **Violation Report Router**
- **Violation Report Evaluator**
- **DB Manager**

5 Implementation, integration and test plan

5.1 Order of implementation

We define a precise order of implementation for the various components of SafeStreets ecosystem. To have a complete view of the components, please see the diagram of figure 2.3.
First of all, "DB manager" must be implemented: it manages the interaction between the server and the database, providing query and definition functionalities that are fundamental for the other components.
After that, the components are grouped by the functionality they implement. These groups are sequentially implemented with respect to the order with which their associated functionality is presented into the project assignment. Inside every group, the order of implementation depends on the physical node in which the component is: first the Server, than the App, AE in the end.
Therefore, the first components to be implemented are the ones related to violation reports. In this group, the component of the server closer to the database is "Violation report router". Then we implement the "violation report manager" on SafeStreets App, with its sub-components "OCR" and "Timeout manager". Finally, "Violation report evaluator" is implemented on SafeStreets AE.

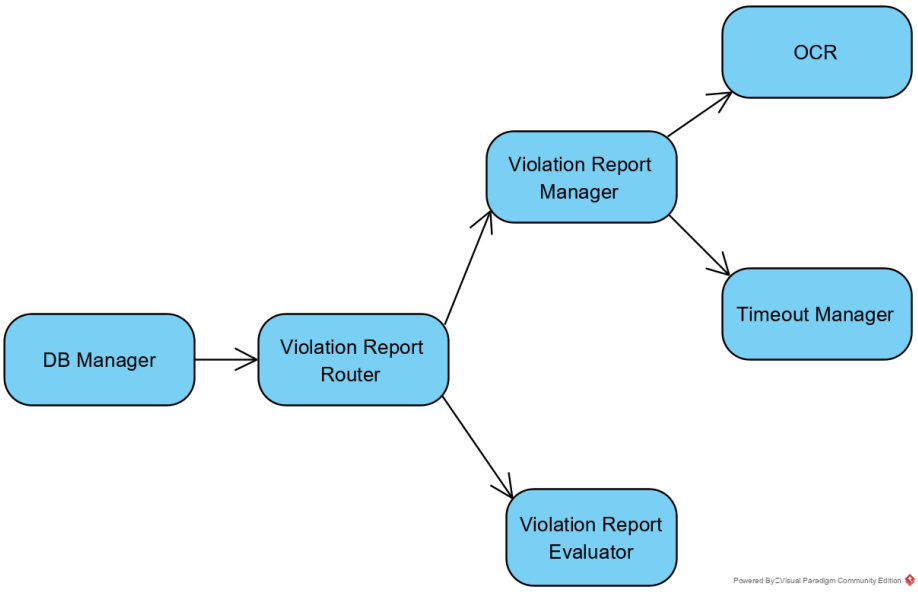

Figure 5.13: Report Implementation order

The second functionality is about data-mining and statistics. Therefore, in this order, we implement: "data-mining manager" in the server, "user data-mining" in the App, "authority data-mining" in SafeStreets AE.

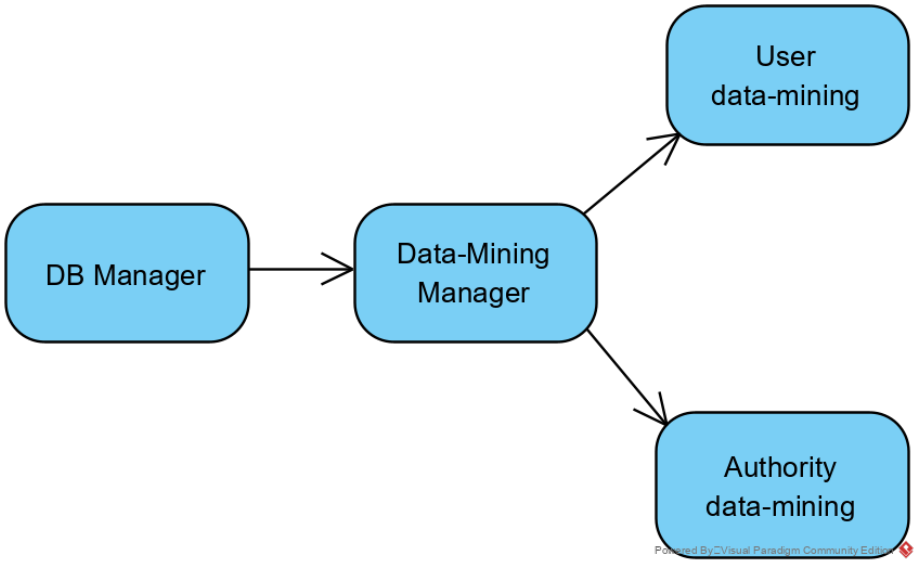

Figure 5.14: Data Mining Implementation order

For "suggestion" functionality, we have "suggestion generator" in the server and "suggestion receiver" in AE.

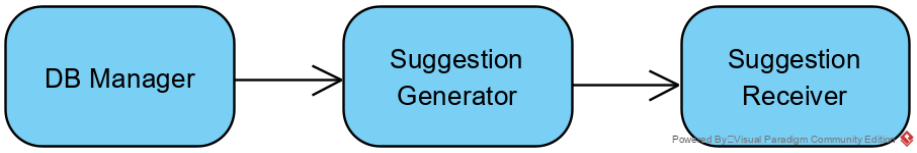

Figure 5.15: Suggestions Implementation order

In the end, for the "accident" functionality, the "suggestion generator" component is implemented, then the "accident manager" on AE.

Figure 5.16: Accident Implementation order

5.2 Integration and testing

The implementation of each component must proceed in parallel with its testing phase. As soon as all the components of a given functionality are implemented, an integration test must be performed. Therefore, the order of testing and integration is strictly related to the order of implementation.
Thanks to the MVC pattern, the view and the controller of each functionality can be implemented in parallel. In fact, as soon as the interfaces between the view and the controller have been defined, these components interact only using such interfaces.
When the view and the controller of the same functionality have been implemented, a test of the entire subsystem that manages that functionality has to be done.
When all the functionalities have been implemented, they must be integrated and tested with respect to the node to which their components belong.
In the end, a global test of SafeStreets must be performed.

6 Effort Spent

Team Work	
Task	**Hours**
Architectural design	4
Discussion about Component and Deployment view	4
Total	8

Table 2: Time spent by all team members

Individual Work					
Nicolò Sala		**Sebastiano Quacquarelli**		**Simone Ricchiuti**	
Task	**Hours**	**Task**	**Hours**	**Task**	**Hours**
Component view	4	User Interface Design	3	Introduction	2
Component interfaces	4	Runtime view	3	Architectural Design	4
Test plan	1	Architectural style/patterns	4	Scheduling plans	2
Requirements	2	Requirements	2	Requirements	2
Total	11	**Total**	12	**Total**	10

Table 3: Time spent by each team member

7 References

- "2019-2020 Software Engineering 2 mandatory project: goal, schedules and rules";
- TeXstudio (`https://www.texstudio.org`) to edit the LaTeX document;
- Visual Paradigm CE (`https://www.visual-paradigm.com/`) to create UML and ArchiMate diagrams.