POLITECNICO
MILANO 1863

AA 2019/2020

DD – DESIGN DOCUMENT

Version 1.0 – 9-12-2019

Computer Science and Engineering – Software Engineering 2

Amedeo Carrioli – 866256, Andrea Ceruti – 945604

Professor Elisabetta Di Nitto

POLITECNICO DI MILANO

Table Of Contents

1. Introduction

1.1 Purpose

The DD purpose is to give an overall description of Safestreets. In particular, we analyze the architecture, in terms of computational components and interactions among those components.
Unlike the RASD, it will have a more detailed description of the backend of the application, underlining the aspects of the application such as the high-level architecture, the runtime behavior and the algorithm design.

1.2 Scope

Here we give an overview of the application based on the descriptions presented on the RASD.
SafeStreets is an application designed to report street violations committed by vehicles on the road.
The users are allowed to make reportings of those violations through the application. In order to do that, users must be signed in.
Two types of users can access the application and its functionalities: private customers and authorities. The first ones can use the basic functionalities, for example they can make reportings and interact with the map, on which the reportings made by other users are shown.
The second ones, authorities, besides the basic functionalities can also make an information cross with the application, and the application, in specific cases, can send suggestions to authorities, in order to prevent more violations.
The violations reported by the users are stored in a map obtained from GoogleMaps API, which has areas colored with different colors (green, yellow, red) depending on the frequency of the violations in that specific area.
When an area is red for a specific type of violations, for example parkings on the sidewalk, a suggestion is made by the S2B and sent to authorities.
Users, on the map, can filter date and type of the violation (parking, traffic lights, accident and speed violations) and the interval of time they want to see the map (today, last week…).
They can also type on a specific area, the map will zoom in, showing all the reportings in that area, in the position they were uploaded with, then, the users can type on a single reporting, and the information about it is shown, for example date, time, position and picture if present.
Authorities, as said earlier, can cross information with the application, sending accident reportings and receiving from the application parking violation reportings, because are the only reliable reportings that the application can guarantee.
The main purpose of SafeStreets is to make streets safer by allowing citizens to help each other throughout reportings of streets violations. SafeStreets helps to keep the streets clear also by

making suggestions to prevent more violations, these suggestions are sent to authorities, which have more power to take measures on the streets.

1.3 Definitions, acronyms, Abbreviations

Definitions:

- **Users**: a generic customer of the application.
- **Authority**: specific customers of the application. They are allowed to cross information with the application.
- **Private**: customers of the application that can't cross information with the application.
- **Reporting**: a description of any street infringement made by any user, uploaded on the application.
- **Application analyzing pictures algorithm**: the algorithm through which the license plate is recognized from the picture of the vehicle.

Acronyms:

- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document
- **CPU**: Central Processing Unit
- **DB**: Database
- **DBMS**: Database Management System
- **GPS**: Global Position System
- **API**: Application Programming Interface
- **TSL**: Transport Layer Security
- **SQL**: Structured Query Language

- **UI**: User interface

- **HTTP**: HyperText Transfer Protocol

Abbreviations:

- [Gn]: n-th goal

- [Rn]: n-th functional requirement

1.4 Reference documents

- Specification document: “SafeStreets Mandatory Project Assignment”.

- IEEE Std 830--1998 IEEE Recommended Practice for Software Requirements Specifications.

- Examples documents:

  · “DD from the car sharing project”.
  · “DD from Travelendar+”.
  · “DD to be analysed AY 2019-20”.

1.5 Document structure

- **Section 1**: An overview of the design document, giving and describing the scope and purpose of it. Definions, acronyms and abbreviation are listed as well.

- **Section 2**: It gives a description of the architecture of the entire system. This is the core of the document because it contains the most relevant architecture views and decisions:
  - Overview
  - High level components and their interaction
  - Component view
  - Deployment view
  - Runtime view
  - Component interaction
  - Selected architectural style and patterns
  - Other design decision

- **Section 3**: It presents the principal algorithms managed by the application. The algorithms are described in order to let the developers have the highest degree of freedom.

- **Section 4**: Here are described in details UI (user interface) already presented in the RASD document.

- **Section 5**: In this section is present the requirements traceability that links the requirements wrote in the RASD to the design element of the DD.

- **Section 6**: Description of the implementation, testing and integration of the system.

- **Section 7**: Shows the effort spent by each member of the group.

1.6 Revision History

- Version 1.0: First release

2. Architectural design

2.1 Overview

This chapter is the core of this design document. Here are illustrated both the physical and logical description of the system.

This chapter is divided in sections, for each of ones we properly provide specific diagrams and descriptions of the SafeStreets system.
The image below (*diagram 1*) explains the architecture of the system in layers. This decision has been made to guarantee scalability and flexibility.

- The **Presentation layer** is the logical layer that includes the UI and the hardware front-end side of the system, therefore the devices of the users. For example, a smartphone for the SafeStreets mobile app and a laptop for the web. On this layer is shown the GUI (graphical user interface), differently programmed based on the device used. Also, are managed the interactions from the front-end device with the systems (when the user makes a reporting, crosses data…).

- The **Application layer** is the part of the system that includes the user's messages from their device to the web server and the SafeStreets application server, through a DMZ (demilitarized zone). The corresponding layer focuses on the business logic and communication with the application server, it handles the functionalities of the application and works on data to show to the front-end side. The application server interacts with users sending notifications and questions when needed, asynchronously, and the users talk

to the server providing messages as sending information of a reporting or accessing the map.

- The **Data layer** comprehends the part of the system involving the database and the communication with it. SafeStreets also has a back-up server which only communicates and synchronizes with the database. The interaction between them stays in this layer. Also, the administration database and center are thought to be in this last layer. The application server communicates synchronously with the database to ask for information.

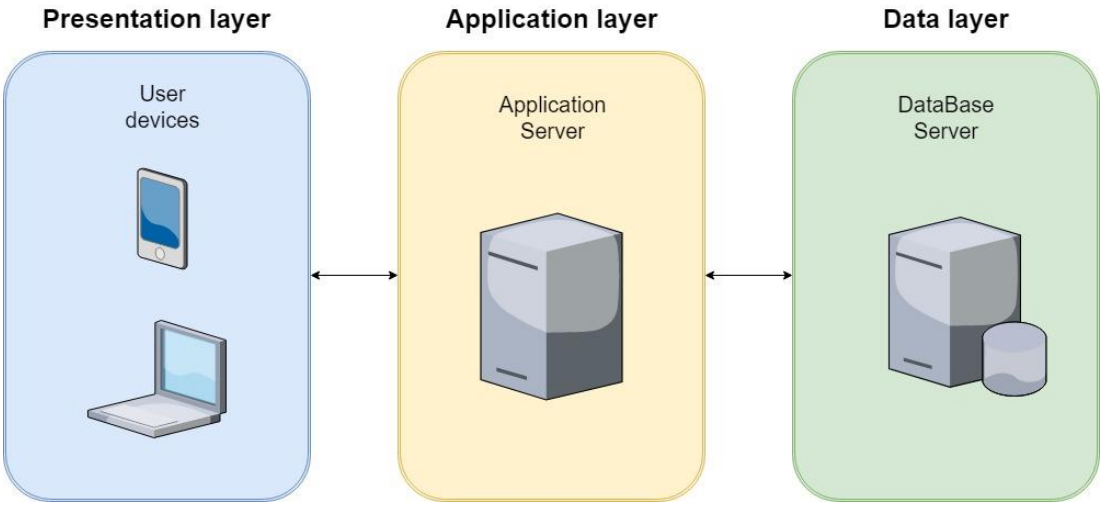

Figure 1 - Layers architecture

2.2 High level components and their interaction

The figure below provides a physical description of the system. The application server is put in the middle of the image to resume the idea of middle layer, in fact it provides access to the business logic tier. Connected to it are the installed firewalls to create a Demilitarized Zone (DMZ). The external networks communicate with the application server through the resources in the DMZ.
On the diagram is shown the presentation tier, which is represented by the users, that can use both the SafeStreets mobile app and the web app, exclusively for authorities. Both are connected with the web server, which constantly talks with the application server, through the DMZ.
The application server, in the middle, which communicate, on the right, with the database server. Information to the database are sent every time needed, for example when a reporting is made from a user and uploaded on the system, the database server will receive from the application server information about that reporting and will store it.
The database also talks with the backup server in order to have information cloning periodically.

Also, the administration center and database talks with the database without passing through the application server, but these components are implemented through a third-party ERP software, so, there won't be descriptions about them in this document since they are maintained from a third party.

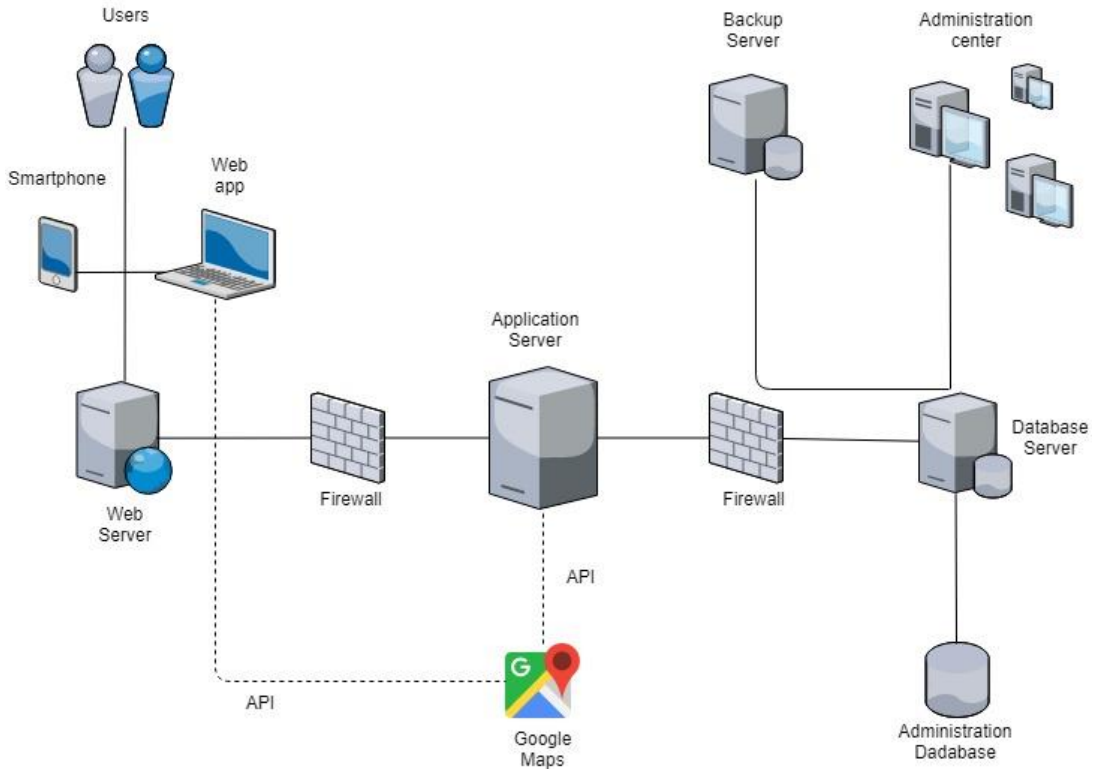

Figure 2 - Physical architectute diagram

2.3 Component view

This section contains diagrams showing the main components and the interfaces of the system. The figure below (figure 3) focuses on the front-end side of the system including the services close to it, each one of them is responsible of specific purposes. For example, the User web services component provide the connections and responses from the authorities through the web application. Remember that only authorities have access to the web application in order to upload and download data in an easy way.
The employee web services component supports the interactions between the system and the SafeStreets personal for small adjustments to the web application. This part won't be explained in detail because is not the focus of this document and doesn't involve any other functionalities or structural decisions for the system.

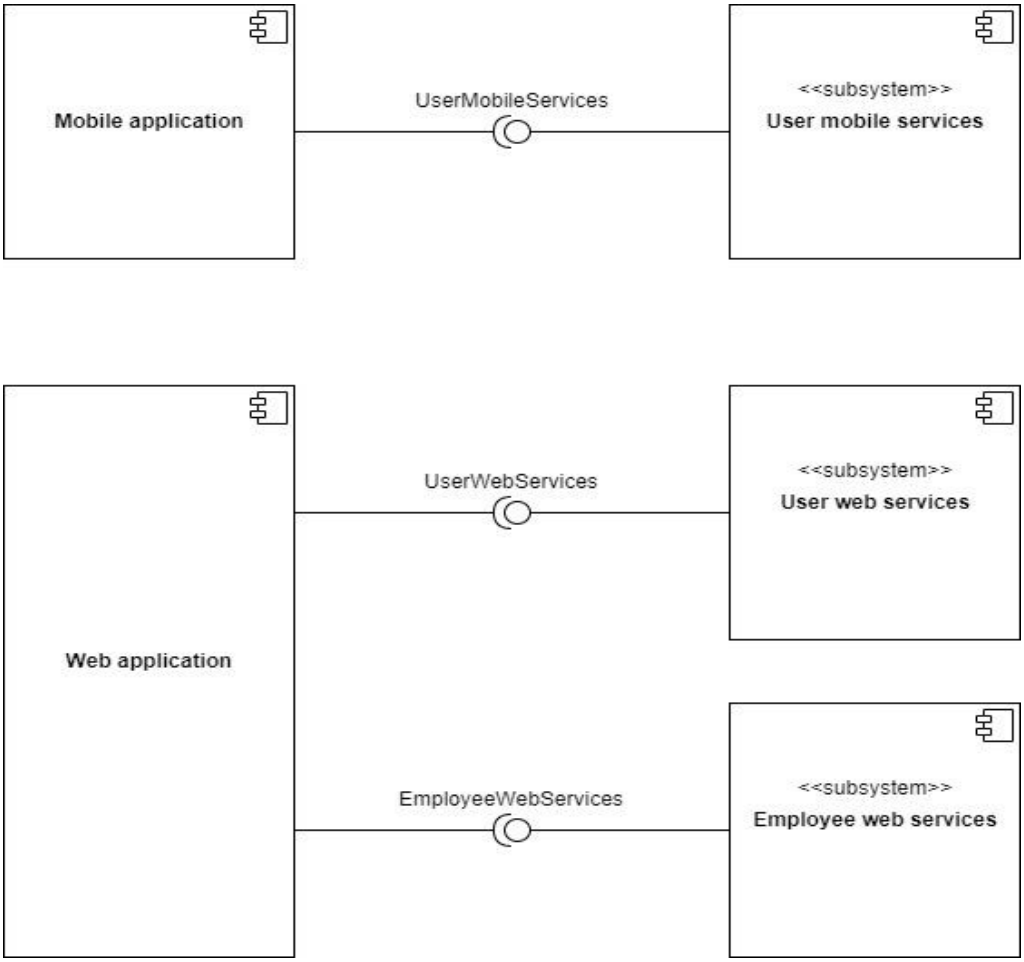

Figure 3 - Front-end services

In figure 4, is shown the user mobile services subsystem, which contains three components:

- Map visualizer module
- Make reportings module
- Account manager

These components provide the user mobile services the following interfaces:

- Visualize map
- Make a reporting
- Reportings history
- Profile manager

The components need to communicate with the GoogleMaps API and the DBMS in order to be able to give these functionalities to the system.

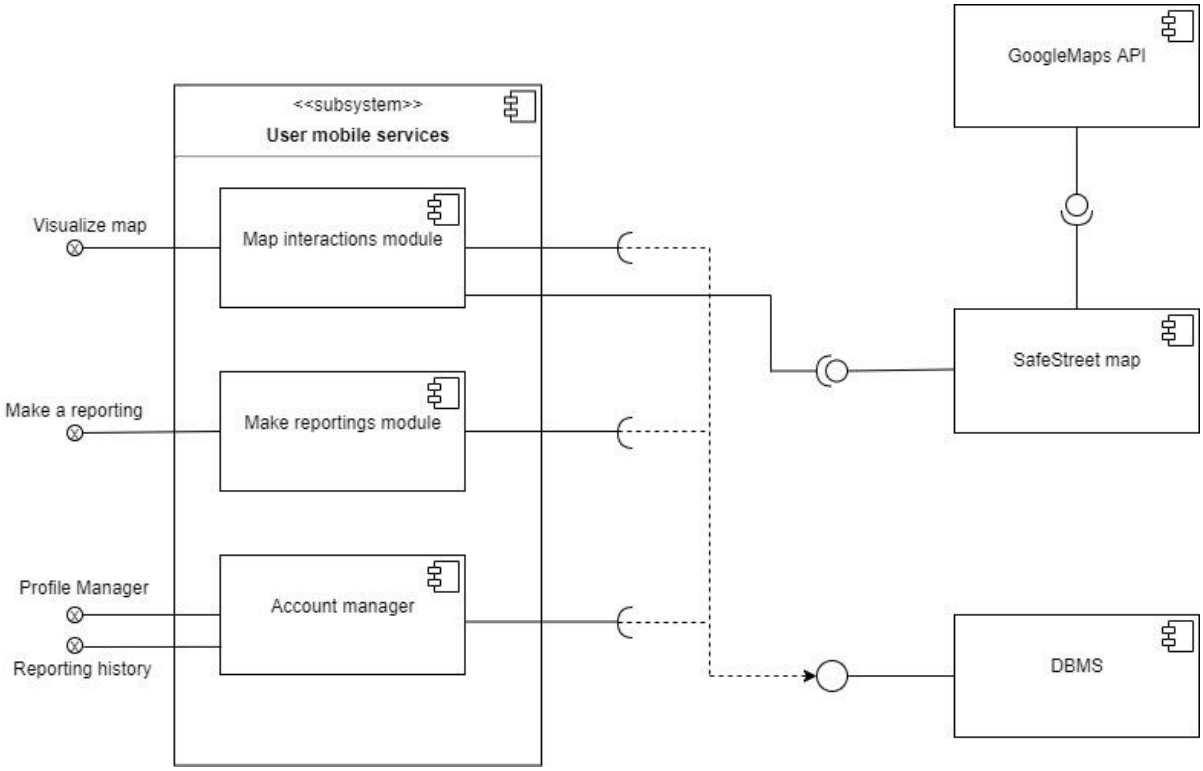

Figure 4 - User mobile services

Figure 5 shows the User web services subsystem, with four components. Three are equals to figure 4, for the mobile services, but, since the web services are accessible only for authorities, a Cross data manager is needed, which allows the users to be able to download and upload data.

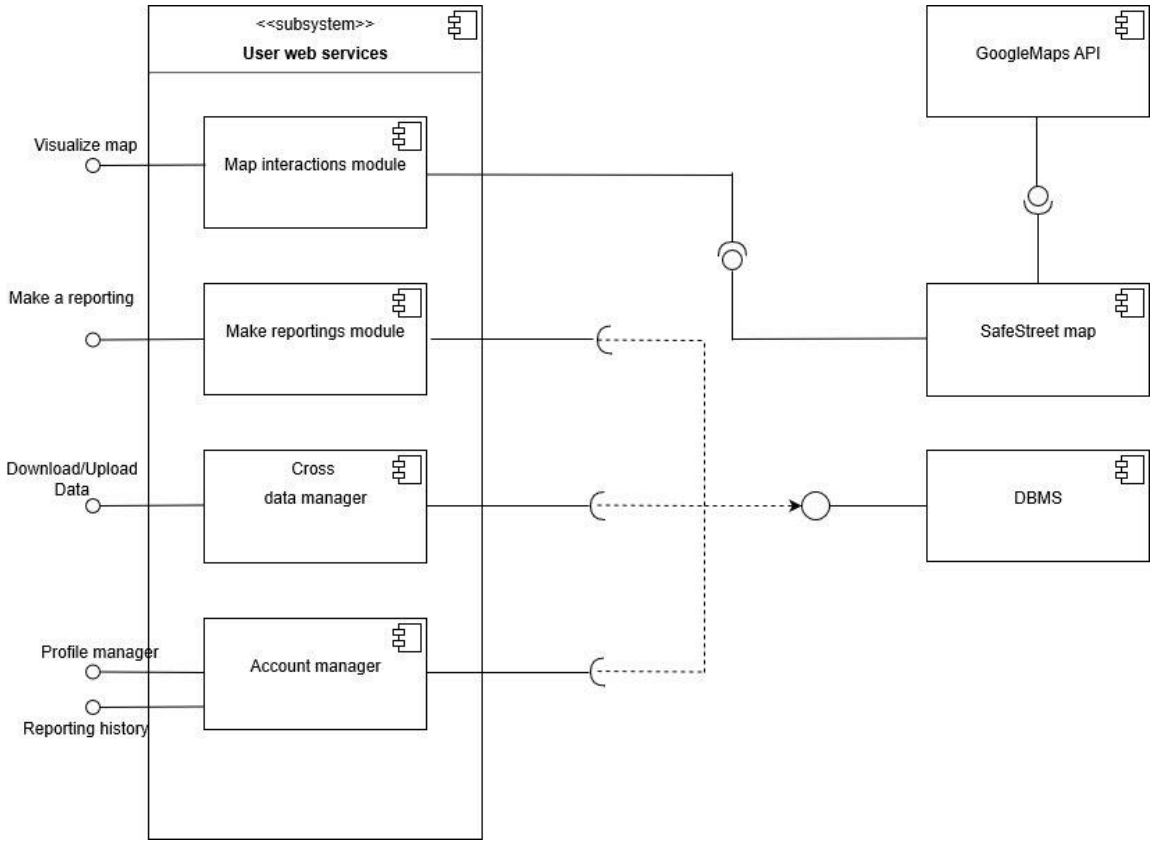

Figure 5 - User web services

Figure 6 is the entity relationship diagram of the system; it provides a conceptual representation of the system.
An area is on a map, is present one map for each city. The map is seen and interacts through the application with the users, identified by a userID. They can be either privates or authorities. Authorities can cross data with SafeStreets, either uploading or downloading them. All the data are saved in the SafeStreets database.
The data that authorities upload are only accidents reportings, while the downloaded ones are only parking reportings, because are the only ones that SafeStreets, through the license plate algorithm, can consider reliable.
Every reporting made is shown on the map, and it is associated with at least one vehicle (in accidents can be involved more than one).
Parking reportings only can have a picture.

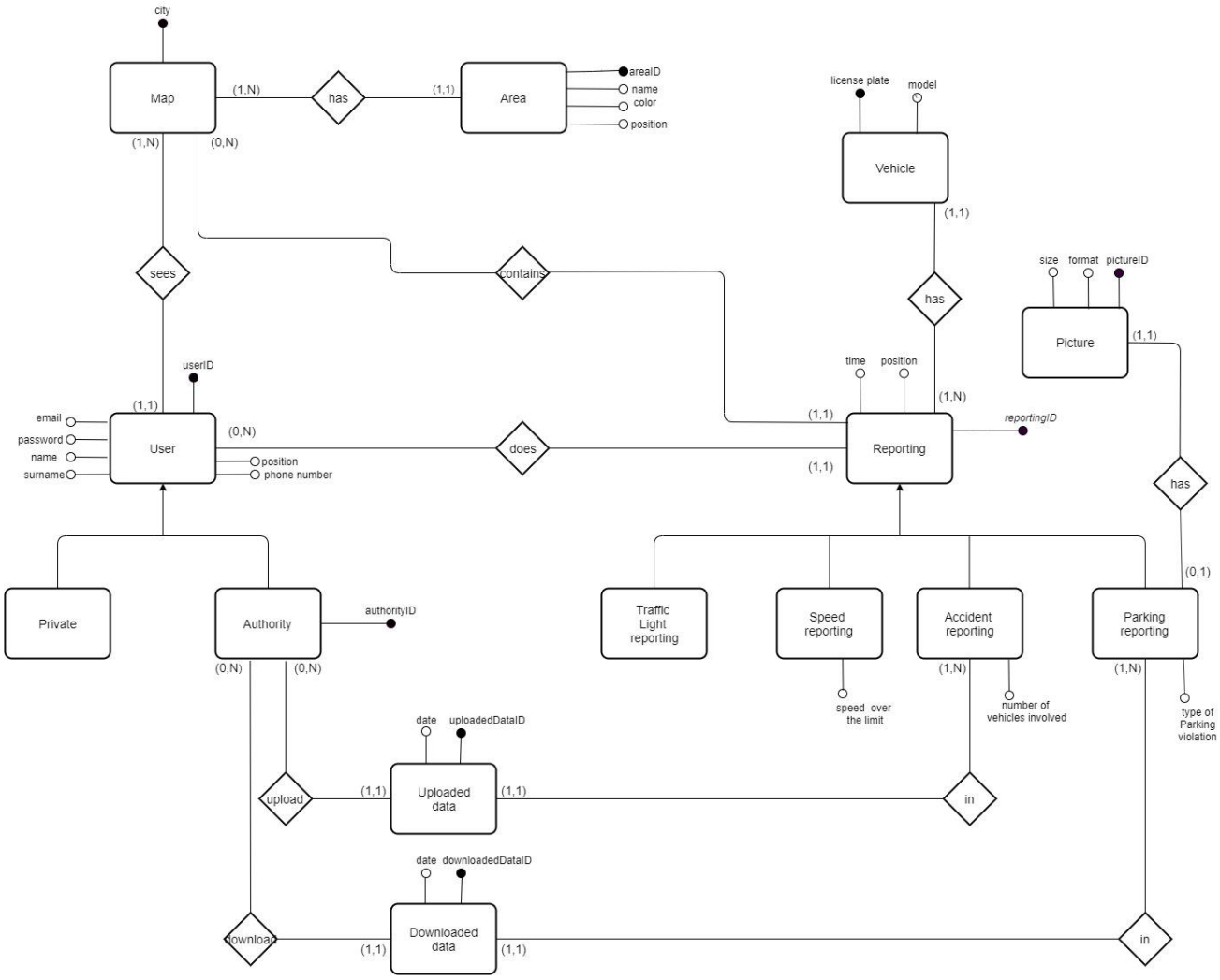

Figure 6 - Entity-relationship diagram

2.4 Deployment view

The image below, figure 7, shows the deployment view of the system represented with artifacts. Artifacts are considered as pieces of information produced or used by a software development process, which are deployed on nodes that can represent hardware of software environments.
In this diagram are not shown minor components of the system as well as firewalls, due for a willing to represent only the cores of the application.
As said earlier, it is a three tiers architecture:

- In **tier 1** are put the smartphone and personal computer as hardware interfaces, because it represents the presentation logic layer, on which the users interact with the

system through the SafeStreets mobile or web app. This tier relates to the application and web servers, the users ask information to them in case they want, for example, upload a reporting or watch a specific reporting on the map.

The mobile app must be compatible with Android and iOS, to make SafeStreets available for almost everyone who has a smartphone. The web app must be compatible with the most common browsers:

  - Google Chrome
  - Firefox
  - Internet Explorer
  - Safari

- **Tier 2** is related with the business logic. It comprehends the application and web servers and it communicates constantly with the previous tier in order to give services to the users. Every time a user asks for information, the request is sent to the application server, which, based on the type of request, either responds to the user or forwards the request to the database server. The principal application's algorithms are in the application server.

- In **tier 3** the SafeStreets database server is present. It works with a RDBMS (relational DBMS) in order to make the system faster and easier to query. This type of DBMS organizes data into one or more tables or "relations" of columns and rows with a unique key identifying each row. Most of the programming within a RDBMS is accomplished using stored procedures (SPs). These are used to greatly reduce the amount of information transferred within and outside of a system. For increased security, the system design may grant access to only the stores procedures and not directly to the tables.

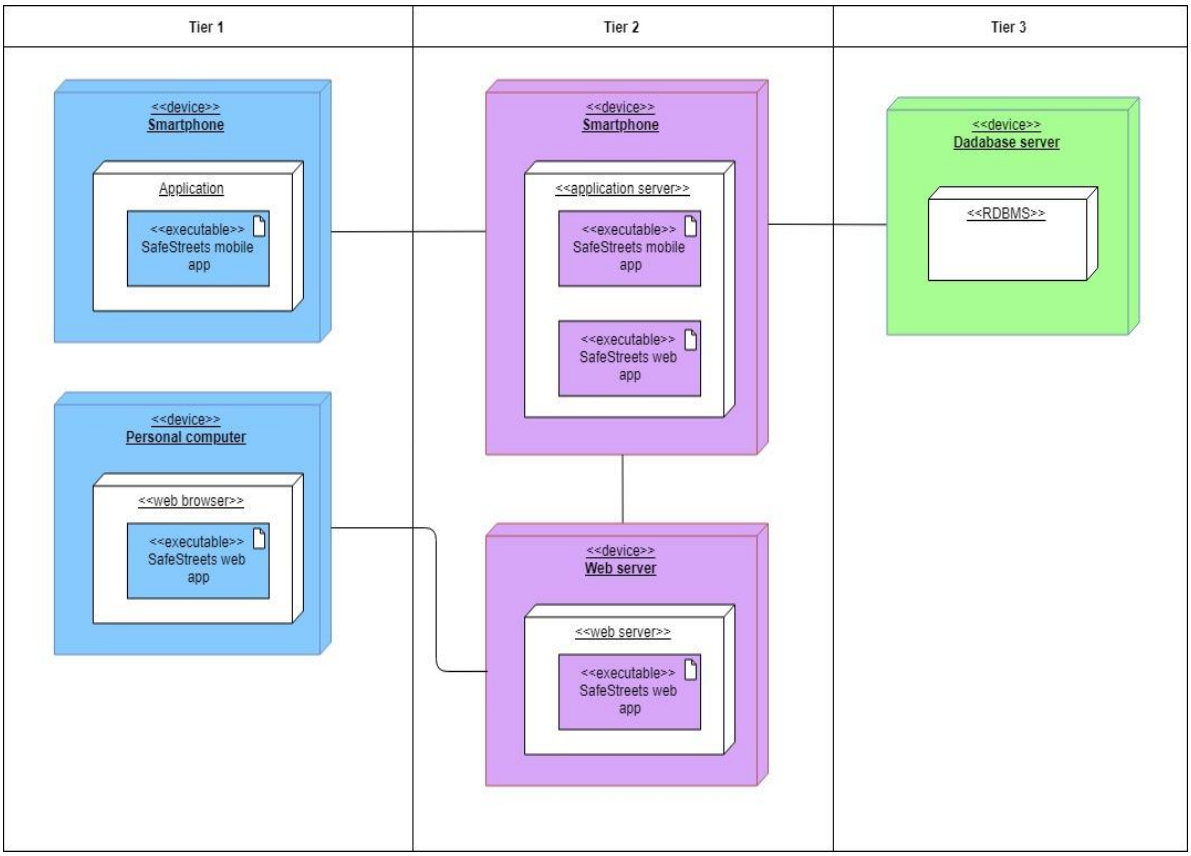

Figure 7 - Deployment diagram

2.5 Runtime view

In this section will be shown sequence diagrams of the main functionalities of the system. It is a high-level description of what really happens, due for simplicity and clearness. For example, the name of the functions in these diagrams might change their name during the development process.
However, the purpose of these functions is and will remain exactly what is show in the following diagrams.

2.5.1 Login

This first sequence diagram, figure 8, shows the Login action that every user must make at least one time.

Once the login page is shown on the user's device, the user inserts email and password and sends it to the system only by clicking on the button "Login".

Now, the system starts to send information up to the DBMS in order to check if those credentials are allowed to enter the system. Then, if the answer from the DBMS is negative, the system blocks the login of the user, otherwise the user logs in in the system and he's ready to use all the functionalities offered by the application.

The request is forwarded to the DBMS because the system stores the majority of information in the database, for example the credentials of the users.

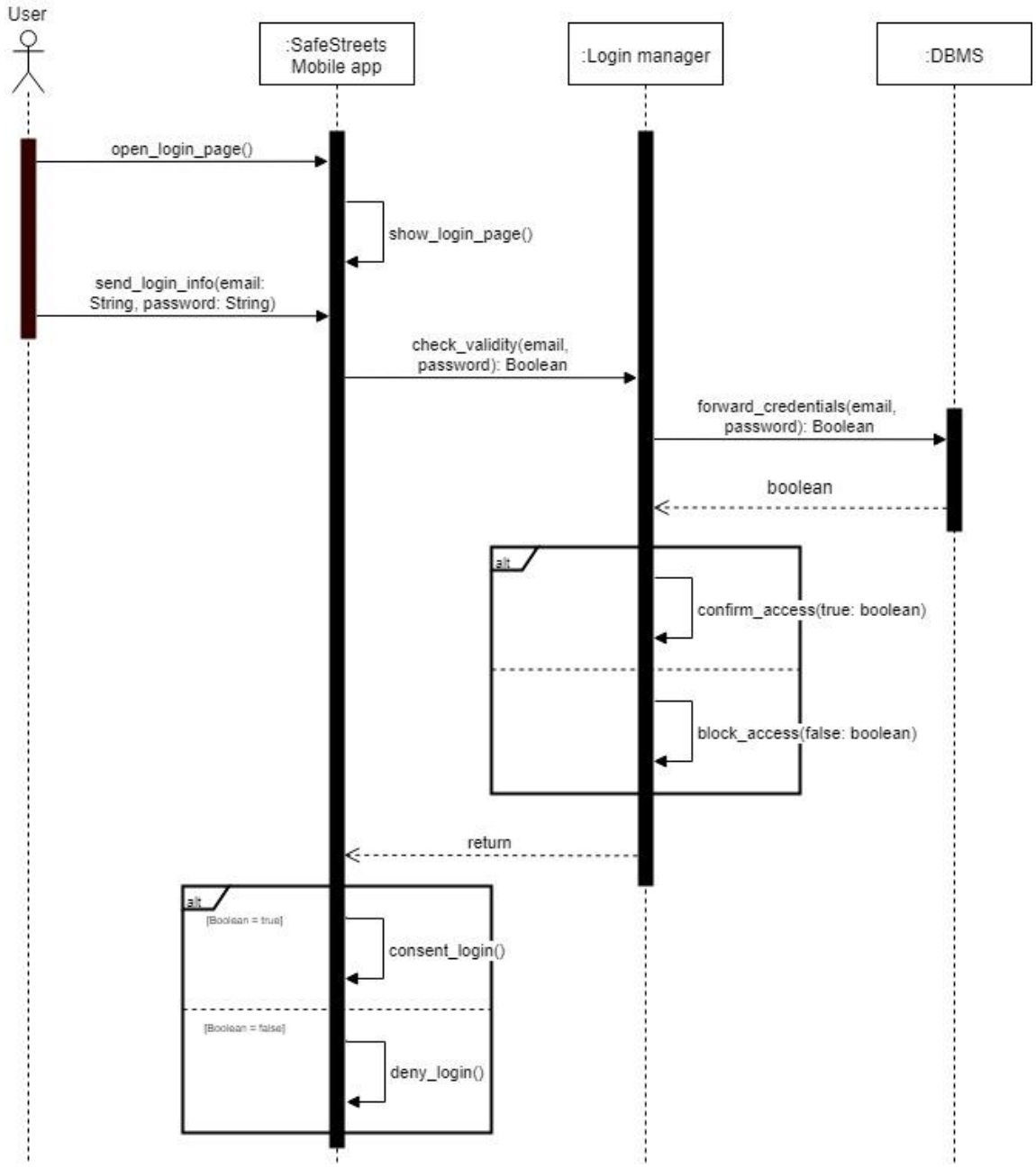

Figure 8 - Login

2.5.2 Reporting making

The sequence diagram below represents the core of the application: reporting making.

The diagram shows how a parking violation is reported, but, as already said, there are also speed and traffic light violations and accidents that can be reported through the application. The processes are similar to the one described below, but with some differences. For example, when a user reports an accident, can insert the number of vehicles involved but can't send a picture of it. Also, for speed and traffic light violations is not able to send a picture of the infringement. This is due to prevent the use of the phone while driving.

This process includes, of course, the insertion of information such as the license plate or the geographical position from the user. As soon as a user chooses to make a reporting, the GoogleMaps API is requested, this is because the user often selects the position of the street violation choosing directly from the map.

The information about the violation are asked in two different moments (see mock-ups in RASD). Firstly, is asked position, time and type of violation, depending from these, the application will ask different information to the user. For example, as below, when the user selects "parking violation" then the system will ask for a picture, a license plate and the specific type of parking violation (zebra crossing, bus stop, etc).

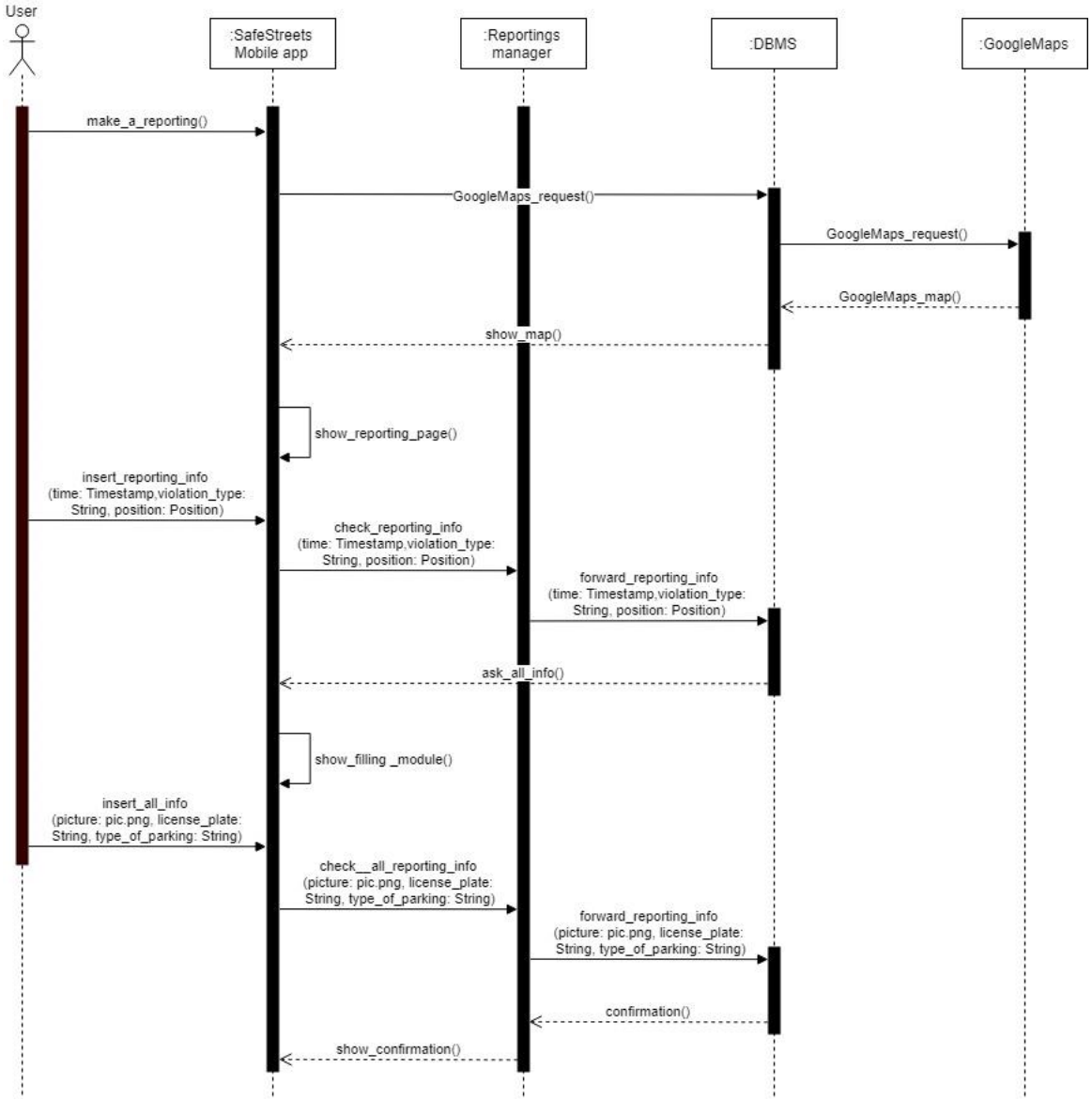

Figure 9 - Make a reporting

2.5.3 Functionalities through the map

In the following diagrams are presented the functionalities offered by the system when a user opens the application and looks and interacts with the map. We remember that, by default, when a user opens the application, it is shown the map of all the types of infractions occurred in the last month, grouped by areas of different colors based on the frequency of violations.

Again, the GoogleMaps API is required. In this specific example is shown the user selection of specific interval of time and type of violations that he wants to watch the map with. Then a specific colored area is selected, we remember that when this is done, the map zooms in and all the reporting of that area are shown. Now the user can

select one of those reportings, and the system shows all the information about it, for example the time and picture, if present.

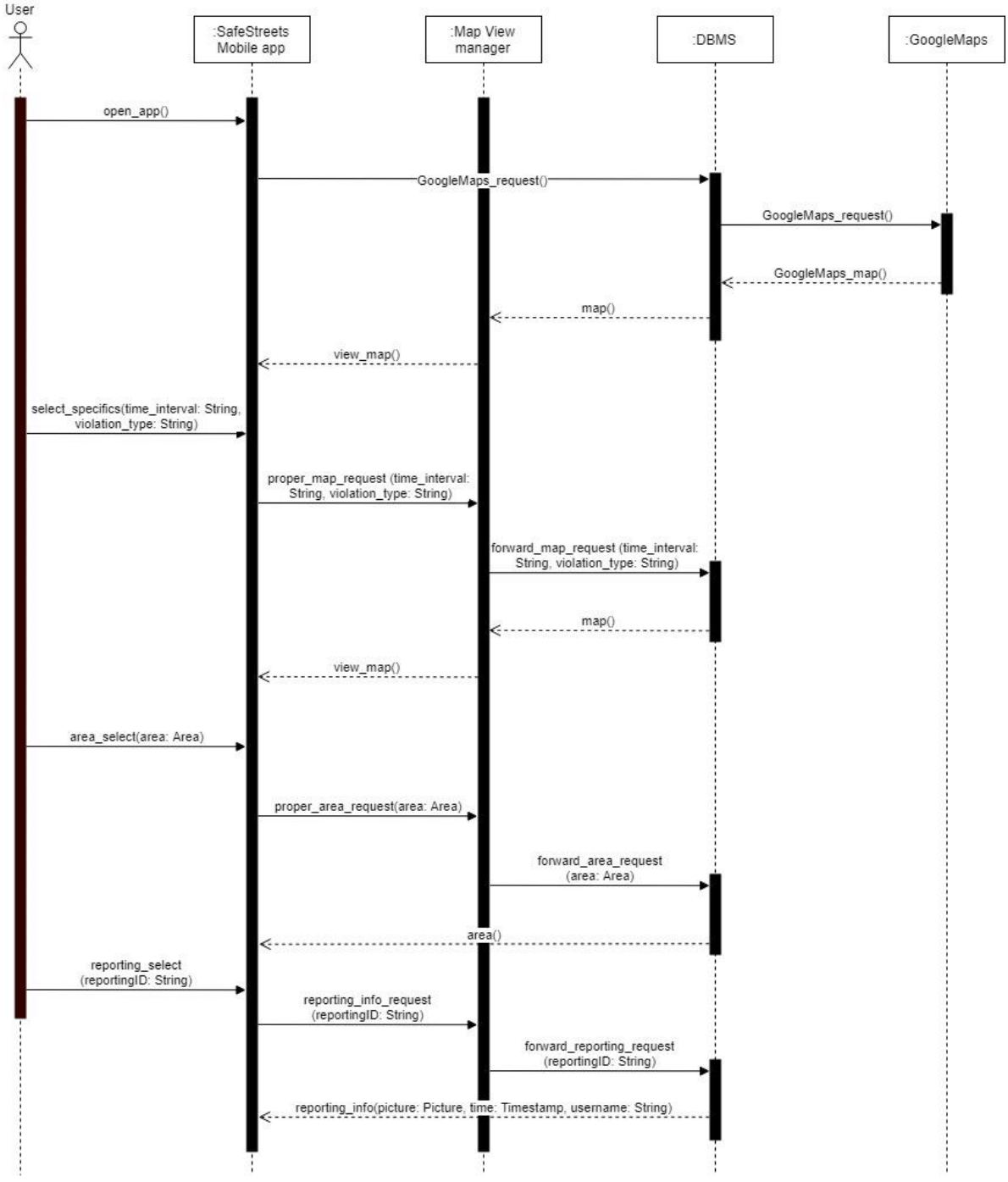

Figure 10 - Map functionalities

2.5.4 Data crossing

Figure 11, the following diagram, shows a specific functionality reserved for authorities: crossing information. In this example the authority uses the SafeStreets web app and both the actions of download and upload data are described.

When the authority wants to send to SafeStreets its information, which, as said previously, are only accidents violations, needs to upload a file .sfst, the format given and made specifically by SafeStreets for data crossing with authorities, in order to send and store data in an easy way for both sides. When the file is uploaded, it is analyzed by an algorithm which puts the reportings on the map.

When authorities download a file from SafeStreets, they download it in the same format .sfst, and the file will have both reliable parking violations and suggestions, if present.

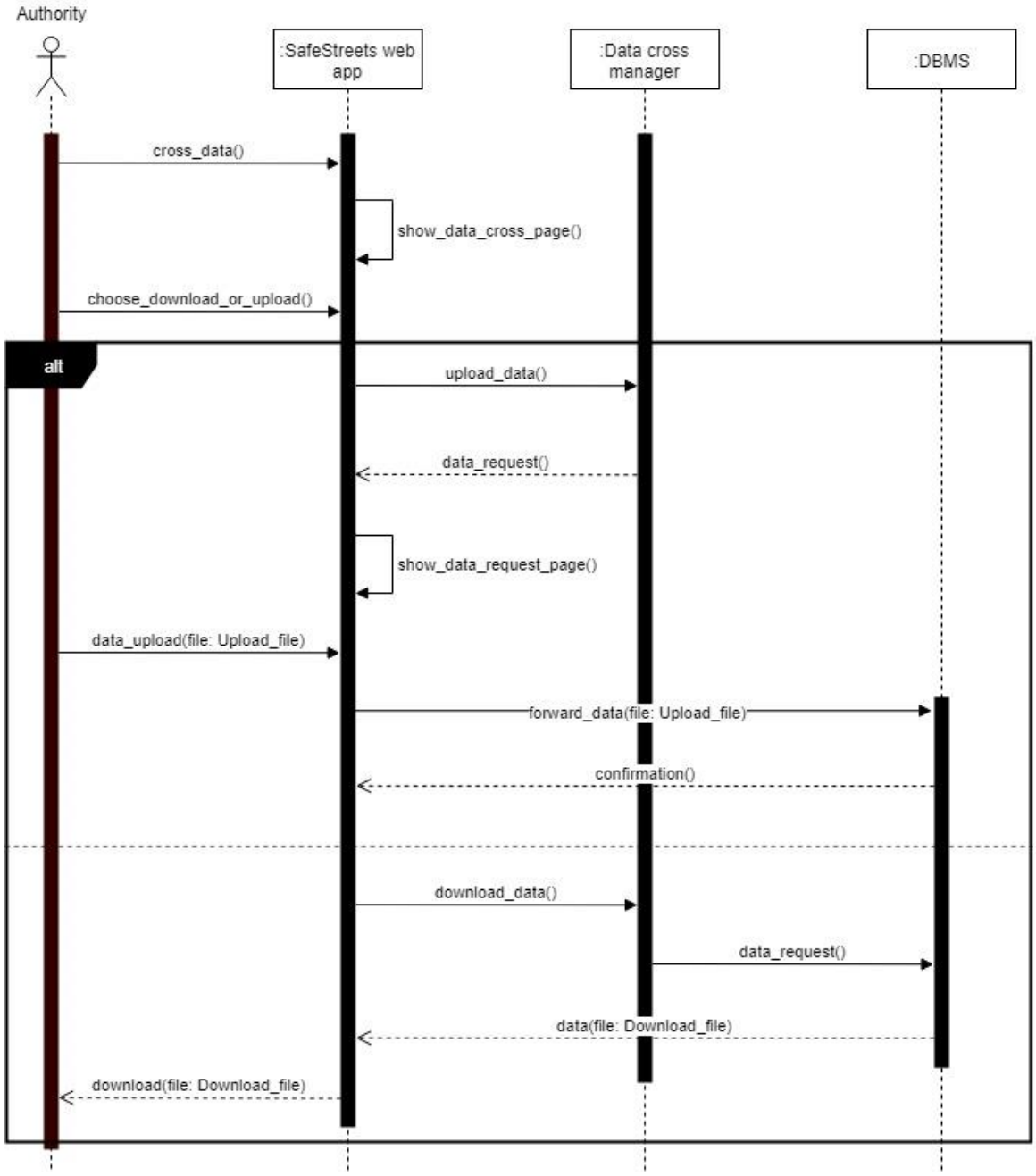

Figure 11 - Data crossing

2.5.5 Reporting history

This diagram shows how a user accesses his personal reportings that he has made. The reportings are stored in the DBMS so the request is forwarded to it.

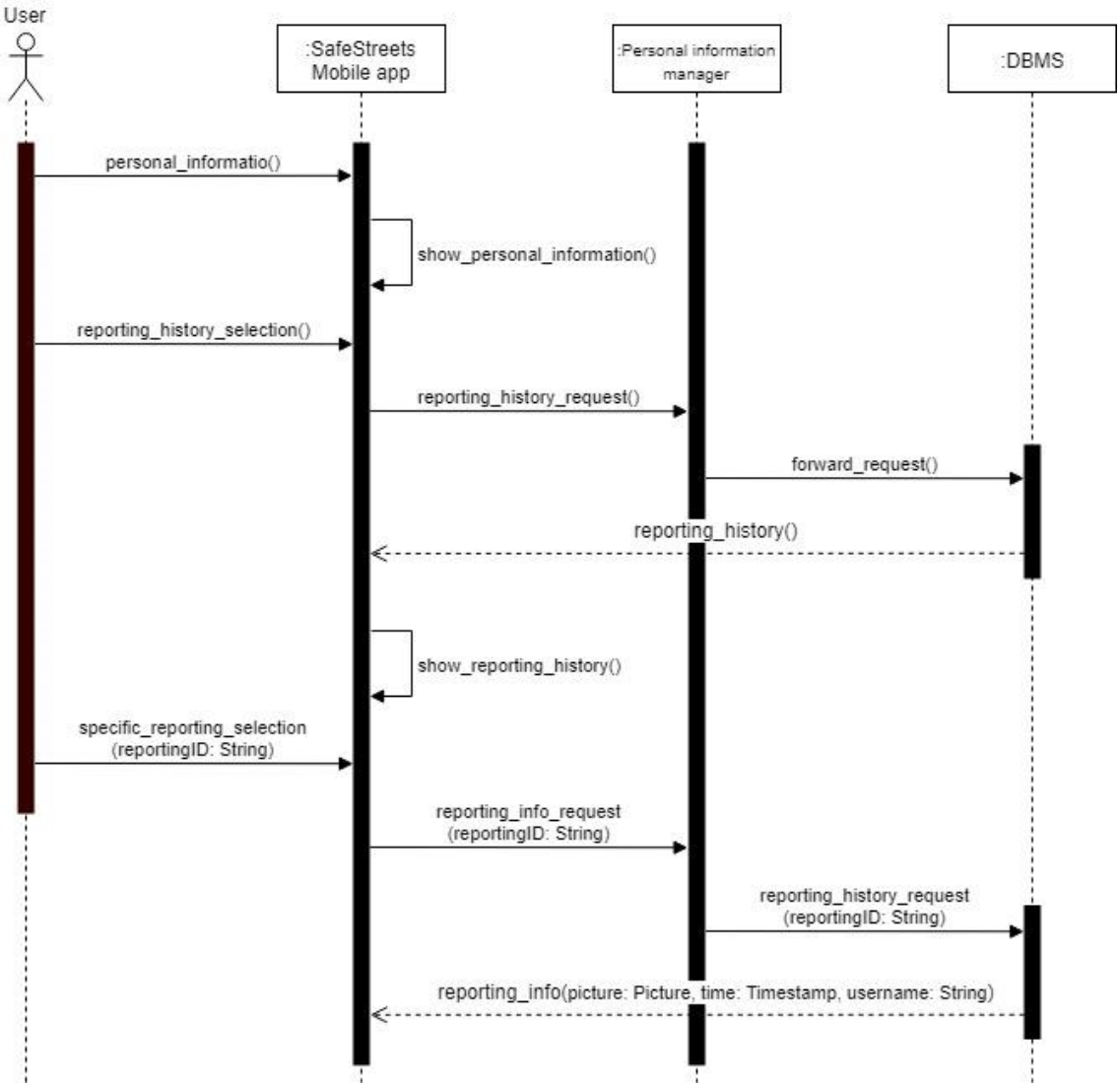

Figure 12 - Reporting history

2.6 Component interfaces

In this section, we delve into the main components of section 2.3 Component view. In particular, we show the main methods belonging to the interfaces of the two principal components of the application, the user mobile services and the user web services.
Regarding the component of user mobile services, it represents all the features offered to the user. We have an interface for the map, “Visualize Map” that permits the user to explore the map object saved in the model application, to filter it according with the parameters he has chosen and to click on specific reportings on map for seeing all the reporting infomation.
Another important interface is the “Make a reporting” interface through which the user can fill out a reporting and upload a photo of it (only parkings).
Then we have two minor interfaces that allow the user to manage his profile and to see all his reportings done since his registration.

The web mobile services component uses the same interfaces as the ones used by the user mobile services because the main features are the same. In addition, it has another interface called Download/Upload data that permits to registered authorities to exchange information with reportings received from Safestreets (this exchange of information will be delved into the algorithm design, section 3).

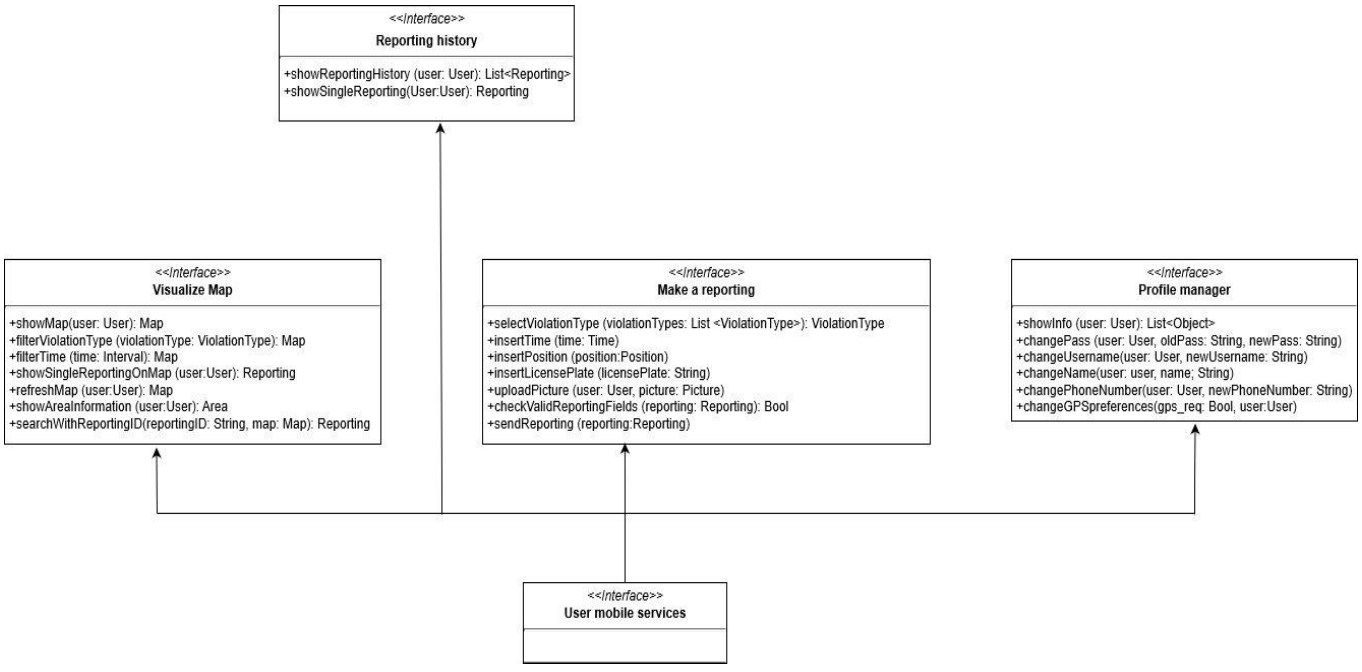

Figure 13 - Component interfaces diagram 1

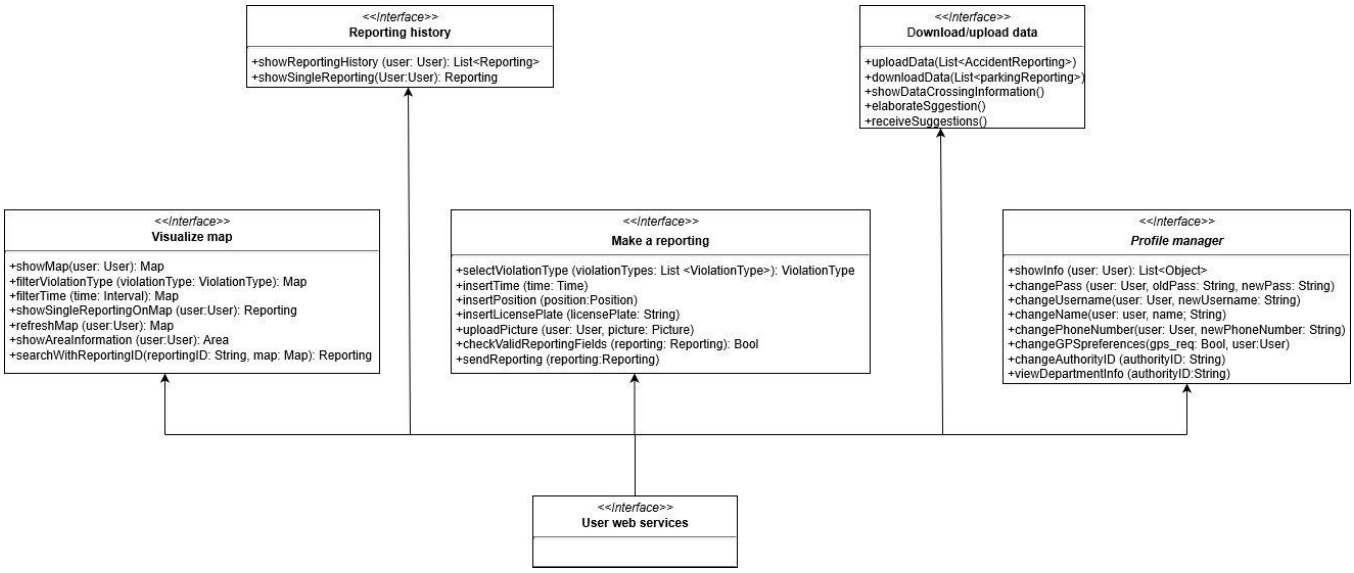

Figure 14 - Component interfaces diagram 2

2.7 Selected architectural styles and patterns

2.7.1 Layer architecture

A good application identified to develop our software is the 3-level architecture.
The advantage of the layered architecture is the separation of concerns: changes made in one layer of the architecture generally don't impact or affect components in other layers: the change is isolated to the components within that layer. The layers of isolation concept also means that each layer is independent of the other layers.
We can say that one of the first advantages that emerges right now is scalability, by separating out the different layers we can scale each independently depending on the need at any given time, also by having disparate layers we can increase reliability and availability by hosting different parts of our application on different servers.
Two other strengthness of the approach used that are worth mentioning:

- HIGH TESTABILITY: because components belong to specific layers in the architecture, other layers can be mocked or stubbed, making this pattern is relatively easy to test. A developer can mock a presentation component or screen to isolate testing within a business component, as well as mock the business layer to test certain screen functionality.

- EASE OF DEVELOPMENT**:** Ease of development is a quality of layered architecture, mostly because this pattern is so well known and is not overly complex to implement. Because most companies develop applications by

separating skill sets by layers (presentation, application, database), this pattern becomes a natural choice for most business-application development.

The Model-View-Controller (MVC) structure, which is the standard software development approach offered by most of the popular web frameworks, is clearly a layered architecture. Just above the database is the model layer, which often contains business logic and information about the types of data in the database. At the top is the view layer, which is often CSS, JavaScript, and HTML with dynamic embedded code. In the middle, we have the controller, which has various rules and methods for transforming the data moving between the view and the model.

2.8 Other design decisions

2.8.1 GoogleMaps API usage

Since our application is focused on the vision of the customized map by our software, it is necessary to use GoogleMaps APIs to render the map on each dispositive that will use SafeStreets.
We could use both the GoogleMaps API which is optimized for smartphones (more on IOS than Android) and the JavascriptAPI designed for web pages, as we need to provide our services on the various registered authority departments.

2.8.2 Relational database management system

RBDMS is a product that showcases data arranged as a collection of rows and column. With an RDBMS, it is possible to embed a collection of programs or capabilities, enabling IT and other related teams to create, edit, update, manage and interrelate with a relational database.
SQL is implemented in our RDBMS systems for accessing the database.
The main function of a RDBMS is to manage and to deal with all relational database, approach taken to develop our application, as mentioned in section 2.4 Deployment view.
The principal strengthness are:

- Easily to understand data structure for data access.
- Facilititating the communication between database clients and the database.
- An RDBMS supports, as mentioned before, a standard language as SQL.
- Maintenance is easier as it helps the database admins or technicians to maintain, repair, control, test and even back up the databases that reside within their main system.

- User access to more than one person. As the data is being updated or changed, users can utilize the built-in locking and transactions management functionality to access the data. This mainly helps in overcoming any possible crashes between more than one users working on the data.
- With the authorization and privilege control features in an RDBMS, it is possible for the database administrator to stop any access requested by authorized users.

3. Algorithm design

3.1 License plate recognizing algorithm

This algorithm is activated when the system receives a parking reporting with a photo attacked. The algorithm takes care of reading the license plate of the car in the photo. If it coincides with the license plate manually written by the user, the reporting is sent to the authorities. Otherwise, either if the user only types the license plate without sending a picture of it or the application analyzing pictures algorithm can't recognize the plate, it is considered not reliable, so the reporting won't be shared with authorities, but It will only be added on the map. Here follows a more technical explanation about the algorithm flow-chart, without constraining it to particular platforms or programming languages.

The algorithm is divided in these macrophases: firstly, it converts true color image into binary image. Secondly, by using filtering it converts the binary image into binary filtered image to remove the unnecessary information and then by using image processing toolbox, it figures it out the connected components in the binary filtered image. Lastly, using the correlation coefficient, the function identifies each letter from the license plate image. Let's take a closer look at each stage.

1. **Load image**: in the first stage, the user loads the image of the car with visible license plate in the GUI; it goes to the second stage.
2. **Filtering**: This stage converts the true color image into a binary one and it processes and filters it out unnecessary information and noises. Once we have a filtered binary image, it moves to a third stage.
3. **Identifying zone of interest**: This stage uses an image processing Toolbox to figure it out the connected components in the binary filtered image. The alignment, height, position of this components are taken into considerations and then the program figures it out the set of components which actually holds the required information and select it.
4. **Recognizing letter**: on the fourth stage the program sends the snaps of these selected components to our read letter function. This function matches the snaps against our pre-recorded database which is analogous to the alphabet in our memory. Using the correlation coefficient, the function identifies each letter and returns it. Thus, the program

saves the letter in the string. To identify the space, the program simply judges the distance between the significantly enlarge than the average; it inserts a white space into the string.

Finally, the string obtained is compared to the license plate manually written by the user. If the two license plates are the same, the reporting is saved in a staging area and once a day a notification is sent to the authorities, who access the web app and download all reporting in the staging area with a .sfst format (also their reportings to SafeStreets are sent with this format, this is the format made by SafeStreets in order to share compressed reportings in an easy way). Otherwise if license plates are not equal or license plate is unreadable, the reporting remains only saved in SafeStreets database.

3.2 Map update algorithm

In order to use of this algorithm, which as the name shows, is responsible of updating the map with the reportings received by the users, SafeStreets deals with the problem of associating each reporting with an area of interest. The following lines describe a possible solution.

Once you receive the city map from GoogleMaps API, the software divides the entire area where the application operates in “neighborhoods”. This subdivision is an implementation issue and will be disregarded later in the algorithm description, assuming that developers have already passed this part and provided us with a class of objects called “Neighborhoods” to work with inside the application model.

Each time a user submits a report, the algorithm takes care of saving it in the neighborhood object and consequently on the Map object, which consists of multiple neighborhoods (organized, for example, in an oriented graph). Neighborhood objects will then have saved lists of reportings within them, with their attributes, which is important to mention two of them: the date and the violation type.

Another useful feature of the algorithm is to allow the user to interactively search the map, through another query application that parameterizes user requests, he inserts parameters such as time interval and type of Violation. The algorithm thus returns for each neighborhood the list of reportings that meet the parameters entered by the user, which will then see the results thanks to a method in the client-side view (Map Color).

The implementation of this algorithm depends on the platforms on which it will be operated, so we simply described the principle behind it, describing the design part at a high level.

3.3 Making suggestions algorithm

This algorithm takes advantage of a Map Update feature, in fact it creates once a day, a reduced Map object with these parameters: Type of violation = parkingViolation, interval of time = last week, thus obtaining from Map Update the map of violations concerning parking of last week. Subsequently for each Neighborhood object in the city performs these operations:

- Separate the reporting list into different lists for each subtype (sidewalk, bus line, ...).
- Count the length of each subtype list.
- If the list number exceeds a certain threshold (we can say a number but we prefer to leave it to the design part of the software) a specific suggestion is generated for that neighborhood that is saved in a list of suggestions.
- Create a new data structure where we find the association neighborhood-list of suggestions

This algorithm then writes the neighborhood/suggestion association to a document in word format, which is compressed together with the list of all the reportings (.sfst format) in the staging area where the authorities interface. It is good to mention that once the word file is created and uploaded, the authorities will get a notification on their web app to remind them to download the staging area.

3.4 Map coloring algorithm

Map color is the application that interfaces with the user's map. Map color sees all neighborhood objects saved by the Map Update software. For each neighborhood represented on the map, Map color by default looks at the entire list of reportings present and calculates a value k = Total number of reports on total reportings types, so we get the average of reportings per type.
Next, based on some tables that measure the degree of danger of the zone (for example, for a k value between 0 and 10 means that the area is green, and so on) it colors the neighborhood on the given map so that the user can have a visual feedback.

When a user wants to see the map, it is displayed as the default with all reports of any type in the last month. Once the user wants to filter it, the user must enter the time interval and type of the violation to see the custom map. These parameters are sent to the Map Update application that lightens the model by returning to Map Color the Neighborhood objects with the requested parameters.
The algorithm then reloads the map from the GoogleMaps API and colors the neighborhoods according to the reporting lists received. If neighborhood don't have reportings will be the default color, otherwise the danger scale is green, yellow, red in ascending order. Finally, it allows the user to see the coloring from its app on the phone.

Once again, we limited ourselves to describing the functionality of the algorithm by exulting from implementation to leave complete freedom to subsequent software developers to choose language and platform.

Here follows a simple diagram to get an idea of the algorithms flow between each other.

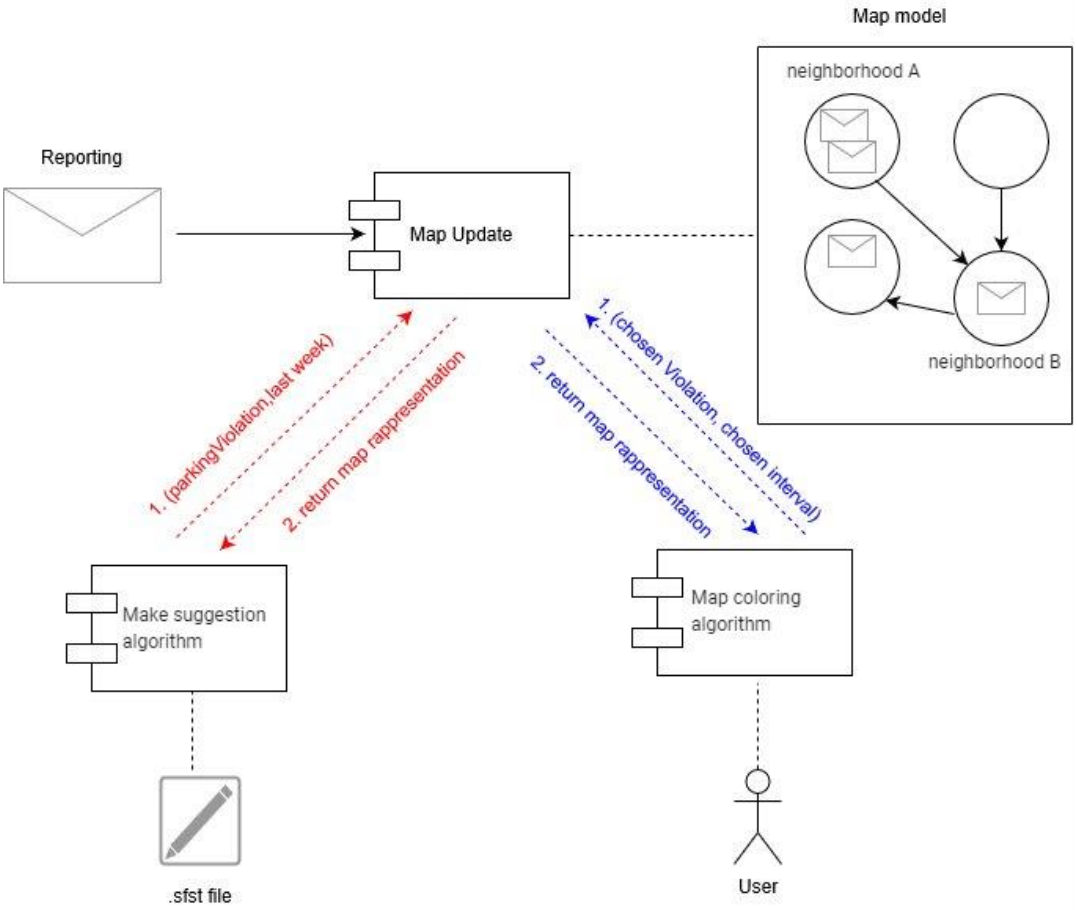

Figure 15 - Algorithms diagram

4. User Interface design

The application UI mockups were presented in the RASD document, section 3.1.1 User Interfaces. Below are presented two diagrams related to both privates and authorities front end application, in particular they expose the flow of the main menu that the user can navigate from their devices. The application windows are represented as colored rectangles half blue and half white, while the available actions in the windows and the credential recovery as simple rectangles. The action to return to the previous menu is omitted for clarity.
Finally, the small green rectangle (success) represents the access to the application servers and the recognition of the user's activity, the red one (logout) represents the logout from the application.

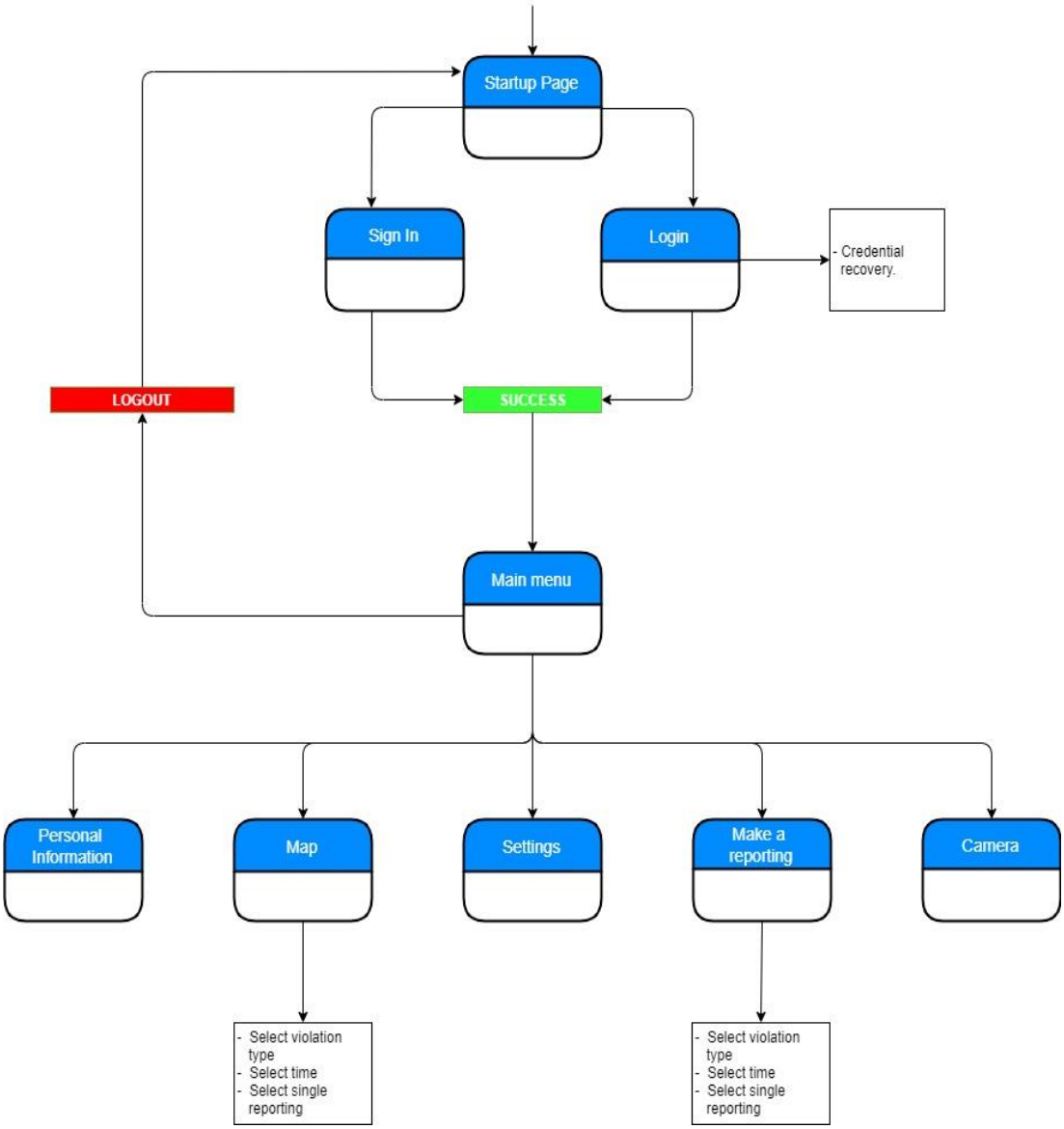

Figure 16 - UX user diagram

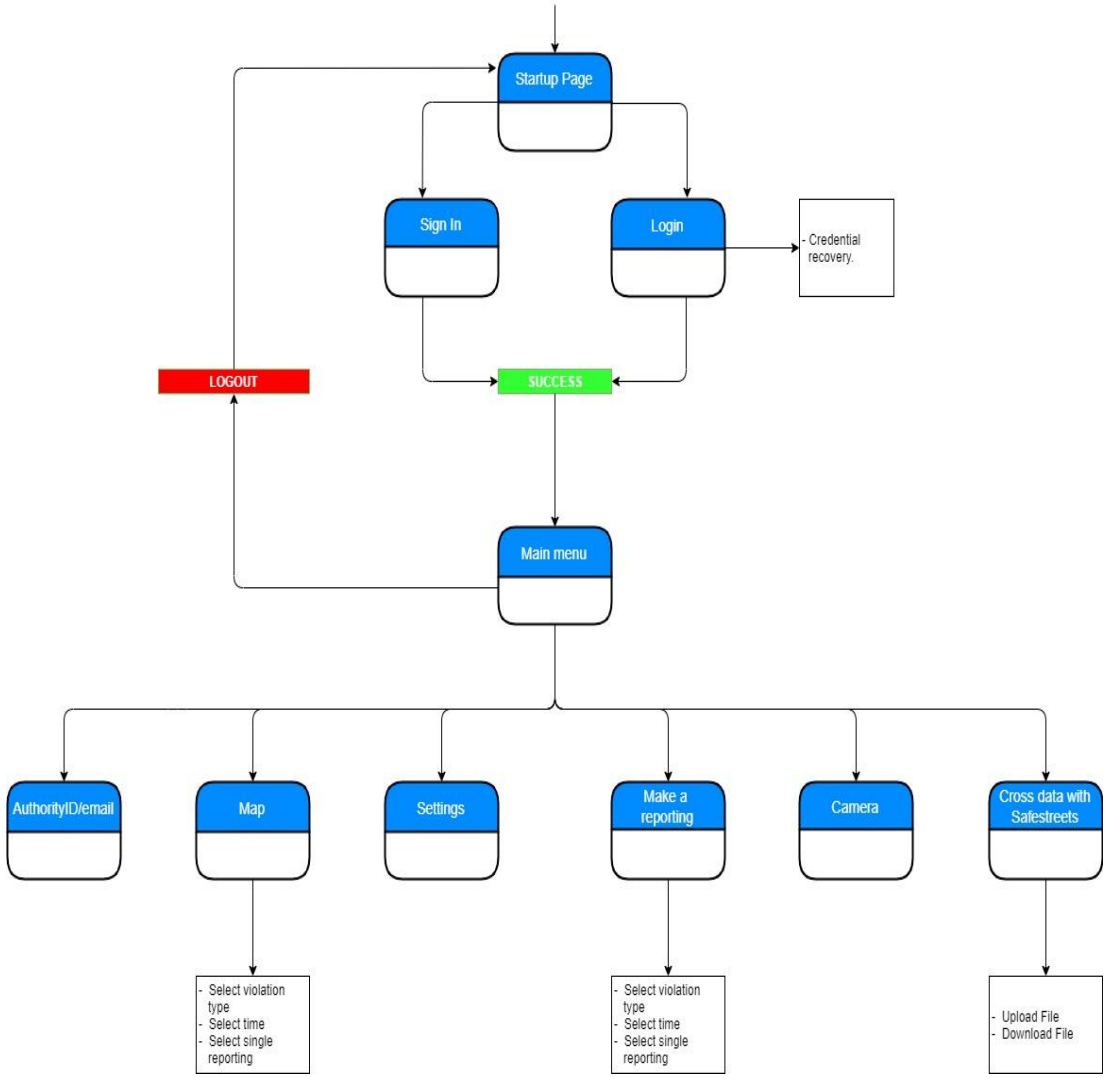

Figure 17 - UX authority diagram

5. Requirements traceability

In this section we show how the designed system guarantees the requirements that have been explicated in the Requirement Analysis and Specification Document (RASD). For this purpose, a mapping between the RASD requirements and our DD components is shown. Note that only components at a high level of abstraction are described, because we have previously decided to keep component diagrams in section 2.3 as simple as possible to give the most freedom to the future designers.

- **R1**: a visitor must be able to begin the registration process. During the process the system will ask him/her to provide credentials.
    - **Account manager**: there is a subcomponent of the manager which is called SignUp activity that allow new users to register into the system.

- **R2**: the user can log in to the application by providing the combination of a username and a password that match an account.
    - **Account manager**: another subcomponent of the manager is the Login activity that allow registered to login into the system in order to make use of the platform features.

- **R3**: the systems must allow the user to compile the reporting form with all the information.
    - **Make reportings module**: the module permits to the user that wants to make a reporting to fill all the information regarding the reporting form.

- **R4**: the system must allow the user to see his violation reporting immediately after he confirms to send the reporting to SafeStreets.
    - **Account manager**: the user through the reporting history interface can see all the past reportings done by him since his registration.
    - **Map interactions module**: the module after it receives a reporting, immediately stores it in the model used by the application, allowing user to see it on map view.

- **R5**: the systems must allow user to remain anonymous after the reporting.
    - **Map interactions module**: after the reporting is complete, the system asks to the user if he wants to remain anonymous or not. If user chooses to not remain anonymous his name will be displayed together with the reporting on the map.

- **R6**: the system must allow a user who send a reporting both to share his position through GPS and to select his position on the map manually.
    - **Make reportings module**: the module at the end of the reporting will also ask for the position sharing. The user is allowed to choose between his GPS position or to insert the position manually on the map.

- **R7**: the application will have to store the information about violations and complete them with suitable metadata.
    - **Account manager**: an important functionality of the account manager is to store all the information retrieved from the user to SafeStreets DBMS. This is done by a submodule called Data collection manager that directly interacts with a lower level of our relational database.

- **R8**: the system must be able to color the map based on the number of violations occurred in each zone.
    - **Map interactions module**: the module uses an algorithm to give this functionality to the user. It is the Map coloring algorithm described above in the section 3.

- **R9**: the system must update the map after every reporting.
    - **Map interactions module**: the module uses an algorithm to perform this functionality. It is the Map Update algorithm, that every time a report is received, it sends it in the application model.

- **R10**: the system must be able to cross information received from municipality with its own data.
    - **Cross data manager**: the cross-data manager stores all the reportings that each part wants to cross in a .sfst file, the file used for compressing the reportings. Thanks to him we can cross the information.

- **R11**: the system must be able to generate suggestions for each type of violation parking.
    - **Cross data manager**: another functionality of the manager is to use an algorithm called Making suggestion algorithm that generates suggestions based on the parking reportings stored in SafeStreets database.

6. Implementation, integration and test plan

The in-scope components are the components of the system that need to be tested. The principal objective is to find as many software defects as possible. We should ensure that our product is bug free before its release. Here we focus on the functions and the external interfaces of SafeStreets. Remember that the principal components of the application are:

- userMobileApp
- authorityMobileApp
- authorityWebApp
- webServerWebApp
- application Server
- external systems: GoogleMaps

The testing is done with a bottom-up approach, it begins with the integration of the lowest level modules, so the model is the first component that will be tested and integrated after the implementation, because It contains all the logic of the application and it will be necessarily used by other components. After the lowest level it continues till all the modules of the software are integrated and the entire application is tested as a single unit, including the integration of GoogleMaps APIs into our software. We choose this approach because it provides high deployment coverage in early process, earlier return on investment, a high visibility of organizational change and principally because all the defects and errors detected in the later stages of the software development lifecycle are resolved and removed before the final product is released for the clients use. It is important that the test and validation phase starts as soon as the development starts in order to detects the bugs as quickly as possible.

Here follows a representation about the principal functionalities that SafeStreets offers to costumer, including a column for the importance of the costumer and the difficulty of implementation, then we will discuss about testing and integration.

Feature	Importance for the costumer	Difficulty of implementation
Sign up and login	low	low
Interaction with SafeStreets map	high	medium
Make reportings functionality	high	medium
Manage account information	low	low
Cross data	high	high

All these features except Sign up and login need to be tested after the model is implemented and tested, because they are all built on top of it.

- **Sign up and login:** this feature is used to ensure that the user of the application is a registered user in our databases**.** It doesn't present a great difficulty in implementation and testing because it is a small module. Some important functionalities that we want to achieve and ensure that they work are the non-JavaScript compatibility, the "forgot your password?" script, "remember me" cookie option.

- **Interaction with SafeStreets map:** this is a core function of the system which allows users to interact with the map of their city. It needs the implementation, the unit-testing and the integration of the "Map interactions module" which stands at the highest level of abstraction. At the lowest level, in fact, we have in the model the algorithm responsible for showing the colored map to users, which is the "Map coloring algorithm" that trades

objects with the “Map update algorithm”. So even if we have told this in the premises before the list of functionalities, we should remember that first the “Map update algorithm” should be implemented and unit tested. Then it should be integrated with the “Map coloring” and unit-test the two algorithms together, to see if the algorithm works as expected. When we are sure of this, we can integrate the algorithms within the “Map interactions module” and test the functionality through devices that differ for the operating system installed.

- **Make reportings functionality:** this is another core function of our system, it allows users to send reportings that the system will put on the map, so it is necessary to implement and unit test the “Make reportings module”. Because we use the bottom-up approach we will firstly test the algorithms that let users compile the reporting form, then integrate them with the “Map update algorithm” that it is responsible of saving the reporting coming from the user into the model structure, and unit-test the parts together. Once we know that all work, we can integrate the parts into the “Make reportings module” and test it through different devices, like smartphone and personal computer (for authority).

- **Manage account information:** this is a minor functionality, to achieve it the “Account manager” needs to be implemented and tested, in particular we need to test if the system recognizes all the reportings done by a user in order to show him the reportings history since his registration.

- **Cross reportings:** this is an exclusive feature for authorities, so the “Cross data manager”, which is present in the web and mobile app, needs to be implemented and unit tested. To achieve it we firstly need to make sure that “Map update algorithm” is implemented and unit-tested, after that we have other two things to check. First, we need to implement and test the .sfst file compression, otherwise we aren’t able to compress reportings and cross them with authorities. The second part is the “Making suggestion algorithm” that produces suggestions observing the parking reportings stored in the database. The suggestions then are compressed in the .sfst file as well. After we have implemented, unit-tested and integrated the algorithms in a single unit, we can pack them within the “Cross data manager” and test it from different personal computers.

Finally, the application will be tested in its entirety to make sure that the front end is well integrated with the back end. There are other types of testing that can be done in order to find defects and break the software: functional testing like the ad-hoc testing can be performed by any stakeholder with no reference to any test case or test design document by a person that has a good understanding of the domain. Moreover, we can do some non-functional testing like the load testing that checks the behavior of the software under normal and over peak load conditions, or the performance testing that checks some quality attributes of the system like stability, reliability and availability.

6.1 Component integration

6.1.1 Integration of internal components of the Application Server

All the components in the diagrams are implemented and unit tested.
The following diagrams show the main managers components of the system and how they work together. There are many other managers under those explicated in the following diagrams, but we focus on the main ones in order to give a clear idea of the most important functions of the system.

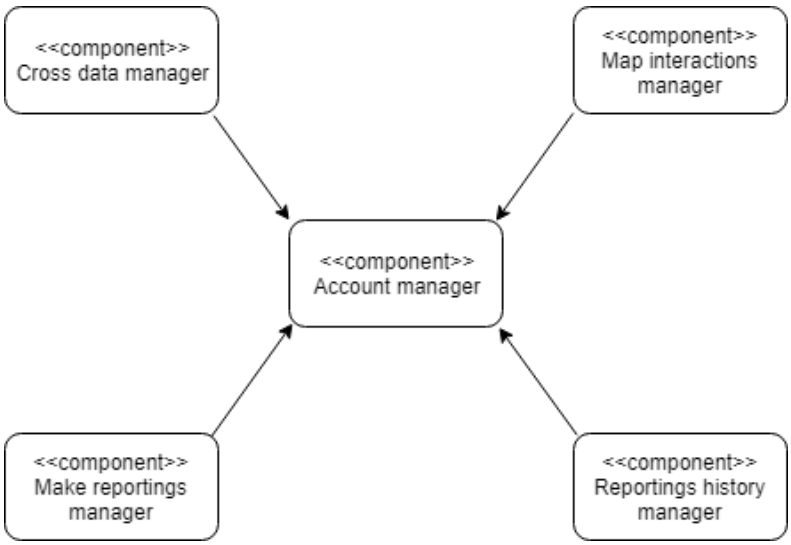

Figure 18 - Account manager integration

Figure 19 - Make reporting manager integration

6.1.2 Integration of the frontend with the backend

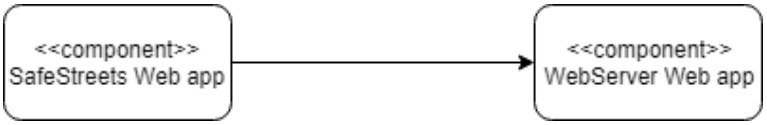

Figure 20 - Web app - Web Server app integration

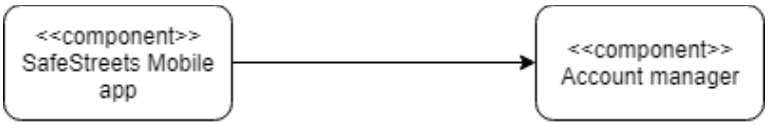

Figure 21 - Mobile app - Account manager integration

6.1.3 Integration of external services

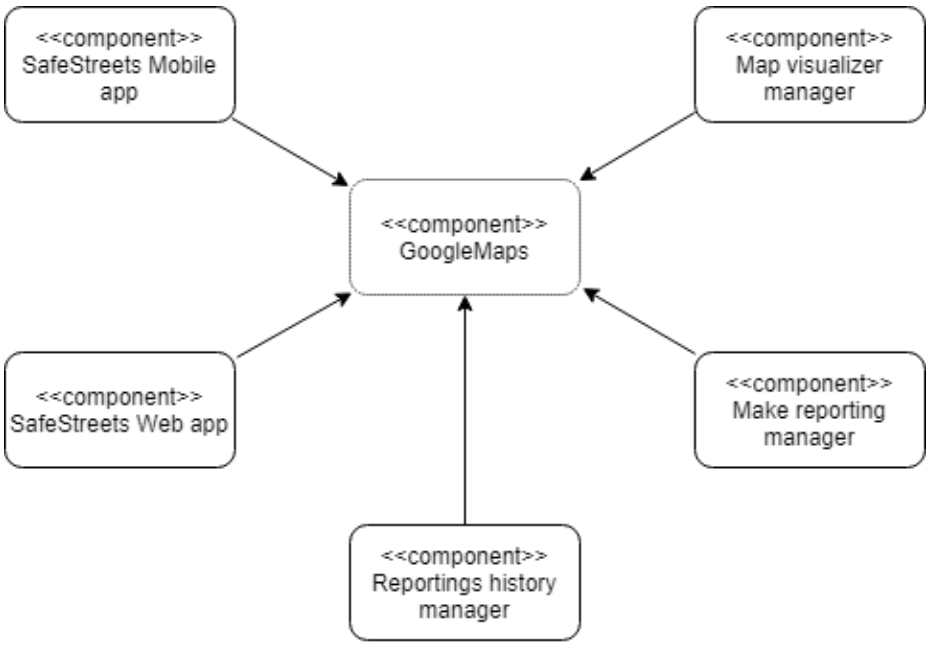

Figure 22 - GoogleMaps integration

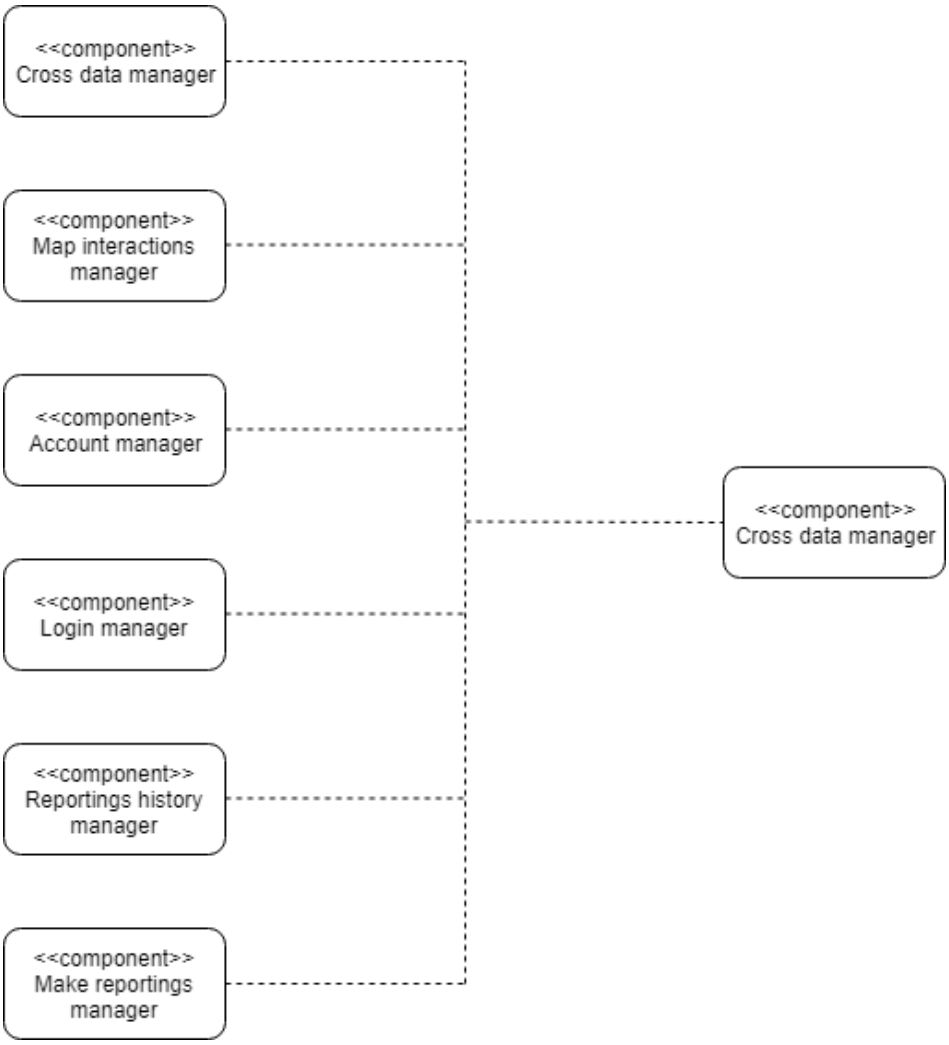

Figure 23 - DBMS integration

7. Effort spent

7.1 Carrioli

Section 1	5
Section 2	45
Section 3	3
Section 4	3
Section 5	2
Section 6	4

7.2 Ceruti

Section 1	2
Section 2	30
Section 3	10
Section 4	8
Section 5	5
Section 6	4