



POLITECNICO
MILANO 1863

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Design document (DD)

SAFESTREETS

- v1.0 -

Authors:

Morreale Federico

Maddes Evandro

Innocente Federico

Student Number:

945258

945642

870726

December 9th , 2019

Contents

1	Introduction	1
1.1	Purpose	1
1.2	Scope	1
1.3	Definitions, Acronyms, Abbreviations	2
1.3.1	Definitions	2
1.3.2	Acronyms	2
1.3.3	Abbreviations	3
1.4	Revision history	4
1.5	Document structure	4
2	Architectural Design	5
2.1	Overview	5
2.2	Component View	6
2.3	Deployment View	9
2.4	Run-Time View	11
2.4.1	Upload New Report	11
2.4.2	Daily Violation Map Request	12
2.4.3	Unsafe Area Request	13
2.4.4	Send Violation's picture Feedback	14
2.5	Component Interfaces	15
2.6	Selected Architectural Styles and Patterns	18
2.6.1	Three Tier Architecture	18
2.6.2	Serverless Approach	18
2.6.3	MVVM (Model-View-ViewModel)	18
2.6.4	RESTful Architecture	19
2.7	Other Design Decisions	20
3	User Interface Design	21
4	Requirements traceability	26

5	Implementation, integration and test plan	31
5.1	Component implementation	31
5.1.1	Provider choice	35
5.2	Component Integration	37
5.2.1	Integration of the Application logic component	37
5.2.2	Integration of the internal component of the application logic . . .	38
5.2.3	Integration of the external services	39
5.3	Algorithms	41
5.3.1	Unsafe Area Search	41
6	Effort spent	42
7	References	43

List of Tables

4.1	<i>Traceability matrix</i>	30
5.1	<i>Features importance</i>	32
6.1	<i>Time spent</i> by all team members	42
6.2	<i>Time spent</i> by each team member	42

List of Figures

2.1	<i>OverView</i> Diagram	5
2.2	<i>Component</i> Diagram	6
2.3	<i>Deployment</i> Diagram	9
2.4	<i>Sequence</i> Diagram (<i>Upload a report</i>)	11
2.5	<i>Sequence</i> Diagram (<i>Show reports on map</i>)	12
2.6	<i>Sequence</i> Diagram (<i>Show unsafe areas</i>)	13
2.7	<i>Sequence</i> Diagram (<i>Send feedback to picture</i>)	14
2.8	<i>UML Interfaces</i> Diagram	15
3.1	<i>UX</i> Diagram	21
3.2	<i>Sign Up</i> Screenshot	22
3.3	<i>Login</i> Screenshot	22
3.4	<i>Home Page</i> Screenshot	23
3.5	<i>Menu</i> Screenshot	23
3.6	<i>My Reports</i> Screenshot	24
3.7	<i>Create Report</i> Screenshot	24
3.8	<i>View Report</i> Screenshots	25
5.1	<i>Integration</i> Diagram of the application logic	37
5.2	<i>Integration</i> Diagram of the internal component	38
5.3	<i>Integration</i> Diagram of the database provider	39
5.4	<i>Integration</i> Diagram of the authentication provider	39
5.5	<i>Integration</i> Diagram of the map provider	40
5.6	<i>Integration</i> Diagram of the plate recognizer provider	40
5.7	<i>Integration</i> Diagram of the authority database	40

INTRODUCTION

1.1 Purpose

The Design Document (DD) aims to specify in more details the technical, functional description on the system-to-be. In particular, it provides an overall guidance to the architecture of the system, it describes the main components, their interface and deployment, it also focuses on run-time behavior, design pattern, implementation, integration and test planning. In this way, this document is mainly aimed to the developers in fact it represents a guide during the development process.

1.2 Scope

With reference to what has been stated in the RASD document, SafeStreets intends to provide users with the possibility to notify authorities when traffic violations occur, and in particular parking violations. The application allows users to send pictures of violations, including their date, time, and position, to authorities. In more details, SafeStreets stores the information provided by users, completing it with suitable meta-data. In addition, the application allows both end users and authorities to mine the information that has been received. Of course, different levels of visibility could be offered to different roles. If the municipality offers a service that allows users to retrieve the information about the accidents that occur on the territory of the municipality, SafeStreets can cross this information with its own data to identify potentially unsafe areas, and suggest possible interventions. In addition, the municipality could offer a service that takes the information about the violations coming from SafeStreets, and generates traffic tickets from it. This addition information about issued tickets can be used by SafeStreets to build statistics.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **User:** is a general customer that use the application. It is used to refer to both an authority and a citizen.
- **Citizen:** is the basic customer of the application. He can upload violations and query the system to get statistics on a selected area.
- **Authority:** advanced customer of the application, is a registered user that is qualified to make fines and view sensitive information. Must verify himself during the registration.
- **Municipality:** is always intended as the local municipality. Every municipality provides and receives information exactly and only about its own jurisdiction.
- **Violation:** is a general infringement reported by a user.
- **Dossier:** is the instance of a violation on the system, that includes the pictures, position, classification, textual specification and the mark.
- **Authority database:** is the database in which are stored all data about incidents. This data are uploaded by authority while the interface to access the database is offered by the municipality. SafeStreets uses this interface to retrieve data from the database.
-

1.3.2 Acronyms

- RASD: Requirement Analysis and Specification Document
- LP: License Plate
- GPS: Global Positioning System
- API: Application Programming Interface
- UML: Unified Modeling Language

1.3.3 Abbreviations

- $[Gn]$: n-goal.
- $[Dn]$: n-domain assumption.
- $[Rn]$: n-functional requirement.
- $[UCn]$: n-use case.

1.4 Revision history

- V1.0, December 9th 2019: First release
- V1.1, December 15th 2019: Minor changes, lexical corrections

1.5 Document structure

Chapter 1: Introduction. A general introduction and overview of the Design Document. It aims giving general but exhaustive information about what this document is going to explain.

Chapter 2: Architectural design. This section contains an overview of the high level components of the system-to-be and then a more detailed description of three architecture views: component view, deployment view and runtime view. Finally it shows the chosen architecture styles and patterns.

Chapter 3: User interface design. This section contains the UX Diagrams of the system to be according to the mockups given in the RASD.

Chapter 4: Requirements Traceability. This section explains how the requirements defined in the RASD map to the design elements defined in this document.

Chapter 5: Implementation, integration and test plan. This section identifies the order in which it is planned to implement the sub-components of the system, the integration of such sub-components and test the integration.

Chapter 6: Effort Spent. A summary of the worked time by each member of the group. At the end there are an Appendix and a Bibliography.

ARCHITECTURAL DESIGN

2.1 Overview

From an external point of view, users using their smartphones, exploit the different services offered by SafeStreets that are different for citizens and authorities. The application to be developed is based on one application component (application logic) which manages the interactions between different components of the system. Application logic is present both in the mobile view and in the back-end logic. In the mobile view there is the management of the GUI and the connection with other useful components present in the smartphone: GPS and camera; while in the back-end logic there is the most of the logic and also the interfaces to external services: Database provider, Plate recognition provider, map provider and authentication provider.

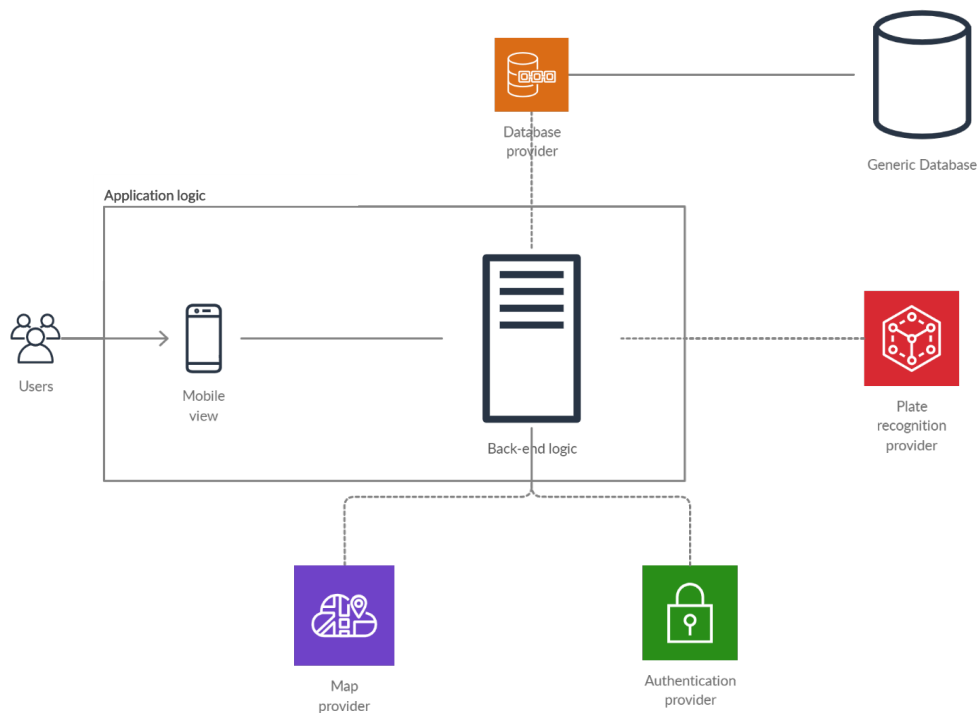


Figure 2.1: Overview diagram of the application.

2.2 Component View

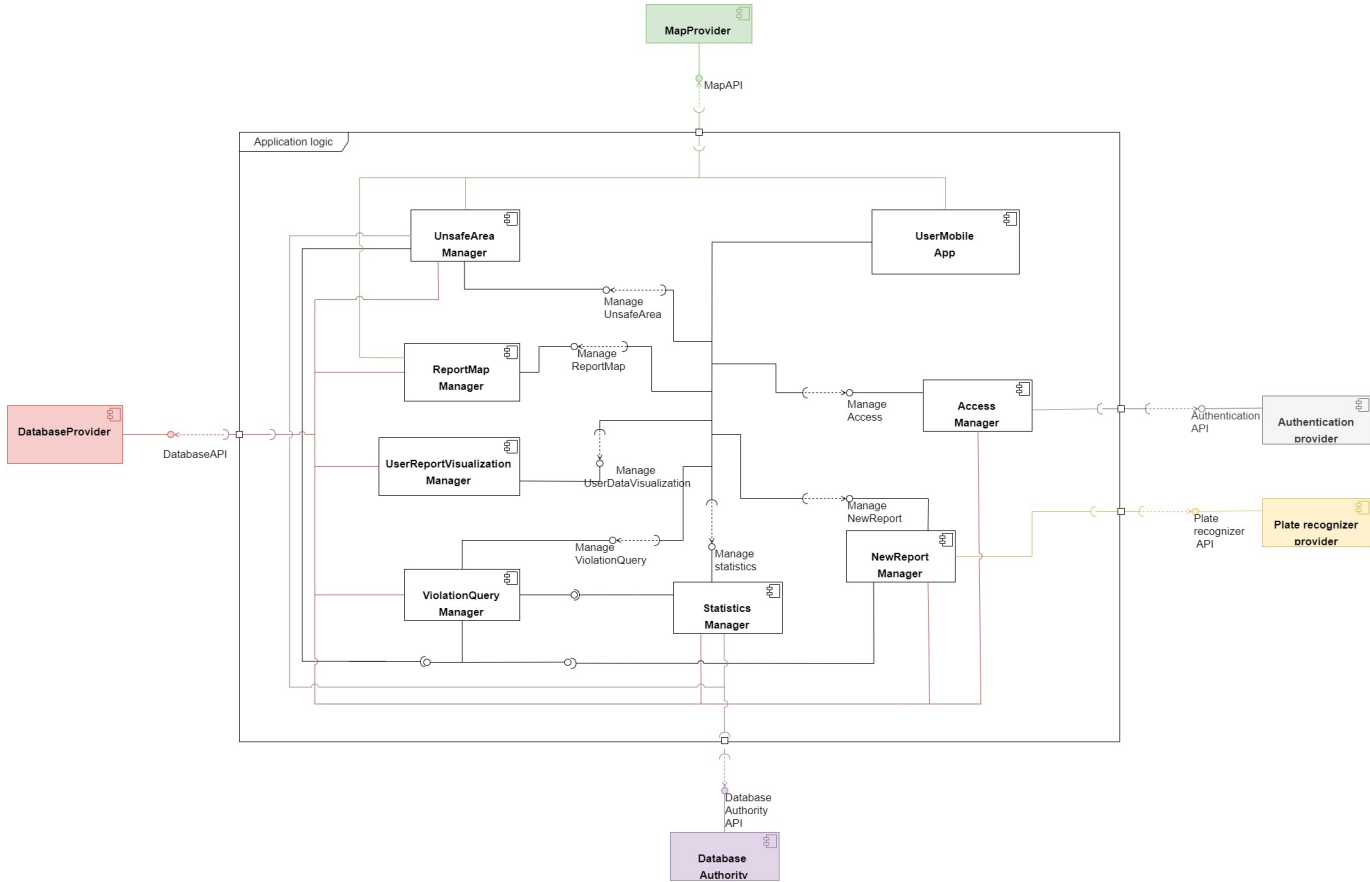


Figure 2.2: Component diagram of the application.

The components' functions contained in the application are described in the following list:

- **ReportMap Manager:** it takes care of showing on the map all the reports with the correct color of the marker. It does that using the position of the report retrieved by gps on device. It interacts with map provider to shows all the reports in the correct position and also with database provider to retrieve the report position. Furthermore if a user clicks on a marker of a report, this manager shows all the information of selected report concerning the type of user: if a citizen the plate are obscured otherwise they are not.
- **NewReport Manager:** it manages the insertion of a new report: procedures to validate the correctness and completeness of the report and to retrieve the position of the user using the gps on the device. It interacts with plate recognizer provider

to recognize the plate in the image so this manager can obscure the position of the plate in the picture. It checks if there is a report of the same type, in the same position and at the same date (using violationQuery manager) and the manager sends the report to the database provider which stores it.

- **Access Manager:** it manages the sign up and login of the users. In particular, it interacts with authentication provider (which contains all the procedures to register a new customer and to authenticate a registered one). For the sign up of an authority is mandatory to insert not only mail address and a password but also the ID. Access manager also interacts with database provider on which there are stored the mails of the users. This is useful for keep an association between the user and all his reports.
- **UserReportVisualization Manager:** it manages the visualization of all reports of one user. In particular interacts with database provider from which it takes reports stored. It shows all the reports of a user and also the possible actions on it: delete, modify, feedback areas and only for authority it displays the field to check if a report is used to generate a traffic tickets.
- **ViolationQuery Manager:** it manages all the procedures to make a query on the database (through the database provider) to retrieve same specific data. User, UnsafeArea manager, Statistics manager and also NewReport manager use ViolationQuery manager to obtain data. Users have a dedicated area in the application where they can make a query by setting different fields (data, type, zone). UnsafeArea manager uses it to retrieve information so that it can find unsafe areas. Statistics manager uses ViolationQuery manager to build statistics on the reports and the NewReport manager checks correctness of a report that is going to be added by checking position and data of report already stored in the database.
- **Statistics Manager:** it takes care of building statistics on the app through specific algorithms. It retrieves data using both database provider in which are stored data about reports and database of authorities in which are stored data about accidents. In particular it builds statistics on effectiveness of the application analyzing the feedback by authorities (reports used for generate traffic tickets) and on the accidents. For all type of statistics, this manager checks if there is at least one user from which it takes data.
- **UnsafeAreaManager:** it takes care of showing on the map the unsafe areas. It finds unsafe areas and displays on the map the information retrieved by using services provided by ViolationQuery manager and also data from the authority

database. Furthermore, only for authorities, it shows suggestions for possible interventions in that specific area.

- **UserMobile App:** it's the application installed on users devices, it interacts with all the services provided by the different managers, it also exploits the map provider on which there are all reports with their own position. It takes the user interaction and feedback. It manages GUI and also the permission of the user to use the GPS and the camera.

Providers:

- **Plate Recognition Provider:** it is used to recognize the plate of a car from a picture loaded by a user.
- **Map Provider:** it provides a real time map of the globe.
- **Authentication Provider:** it manages the access of the registered users and the registration of the new ones.
- **Database Provider:** it is used to store all the information regarded the reports and the users information.

Other Components:

- **Database Authority:** it is used by SafeStreets in particular w.r.t component diagram, UnsafeArea manager and StatisticsManager exploit it to use data about accidents.

2.3 Deployment View

In the following image the SafeStreets deployment diagram is represented: it shows the execution architecture of the system and represents the distribution (deployment) of software artifacts to deployment targets (nodes). Artifacts, in general, represent pieces of information that are used or are produced by a software development process and are deployed on nodes that can represent either hardware devices or software execution environments.

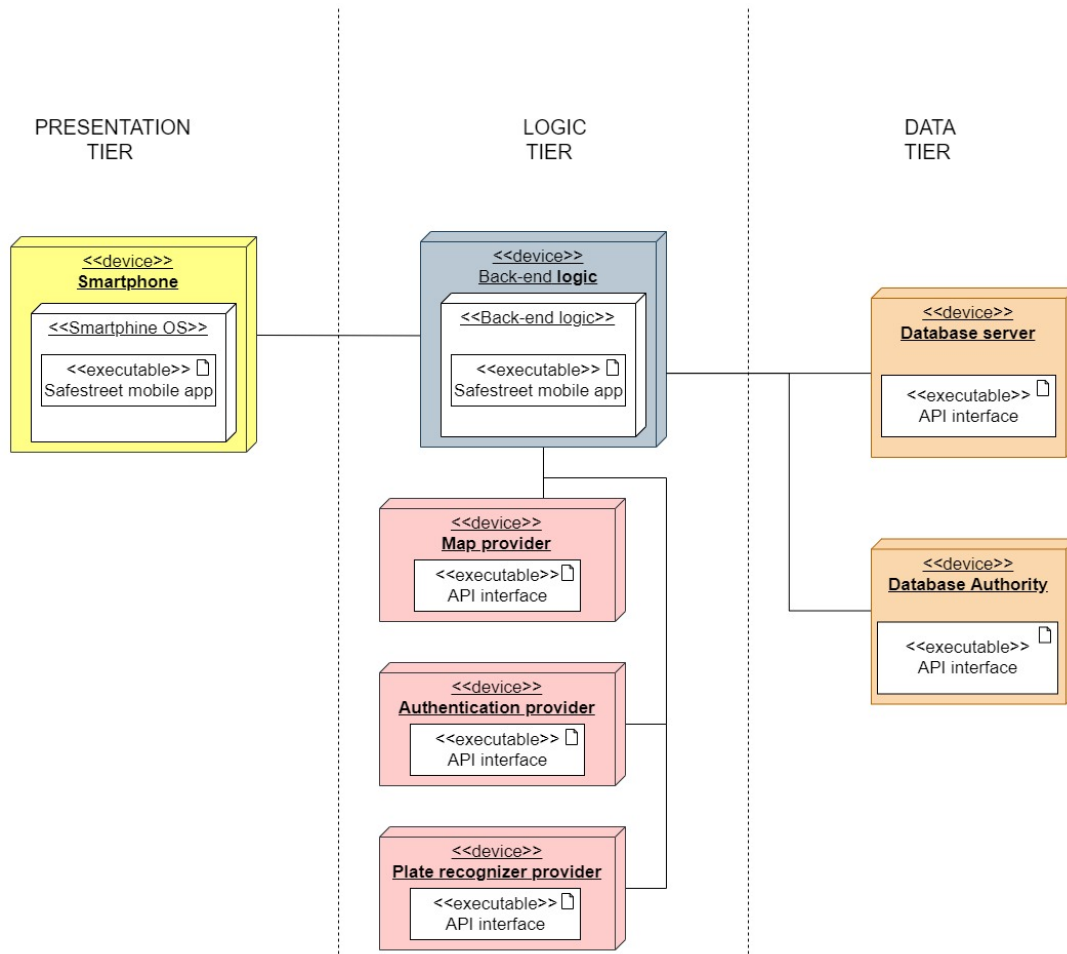


Figure 2.3: Deployment diagram of the application.

In the diagram, external systems are represented through the interface that they offer. The system is divided into three different tiers and represents the common architectural style:

- **Presentation Tier:** The top-most level of the application is the user interface. The main function of the interface is to translate tasks and result to something

that the user can understand.

- **Logic Tier:** This layer coordinates the application, processes commands, makes logical decision and evaluations, and performs calculations. It moves and processes data between the two surrounding layers and this happens for all the managers of back-end logic. It also interacts with the different interfaces through which the providers offer their services.
- **Data Tier:** Here information is stored and retrieved from two databases: authority database for the accidents and the other one for the reports and users information. The information is then passed back to the logic tier for processing and then eventually back to the user.

2.4 Run-Time View

2.4.1 Upload New Report

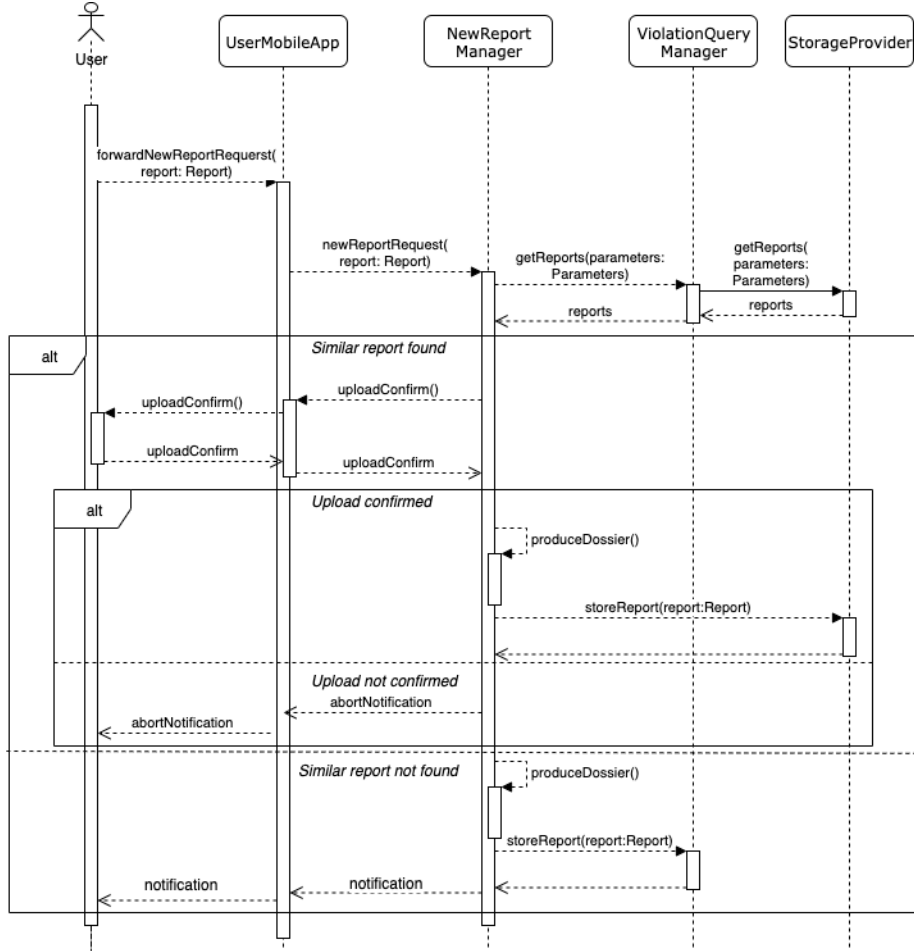


Figure 2.4: Sequence diagram of uploading a report.

This sequence diagram shows the process that allows a user to upload a new violation report. The information about the violation are sent to the UserMobileApp, except for the position that is obtained by UserMobileApp itself through the user device GPS. The request is forwarded to the NewReportManager, that calls the ViolationQueryManager to check if a similar report has already been uploaded. In that case, the NewReportManager sends a request to the user to confirm the upload. If the answer is positive, or if a similar report has not been found at all, it uses all the collected information to create a dossier and stores it in the database. Otherwise, if the user does not confirm the upload, the operation is aborted.

2.4.2 Daily Violation Map Request

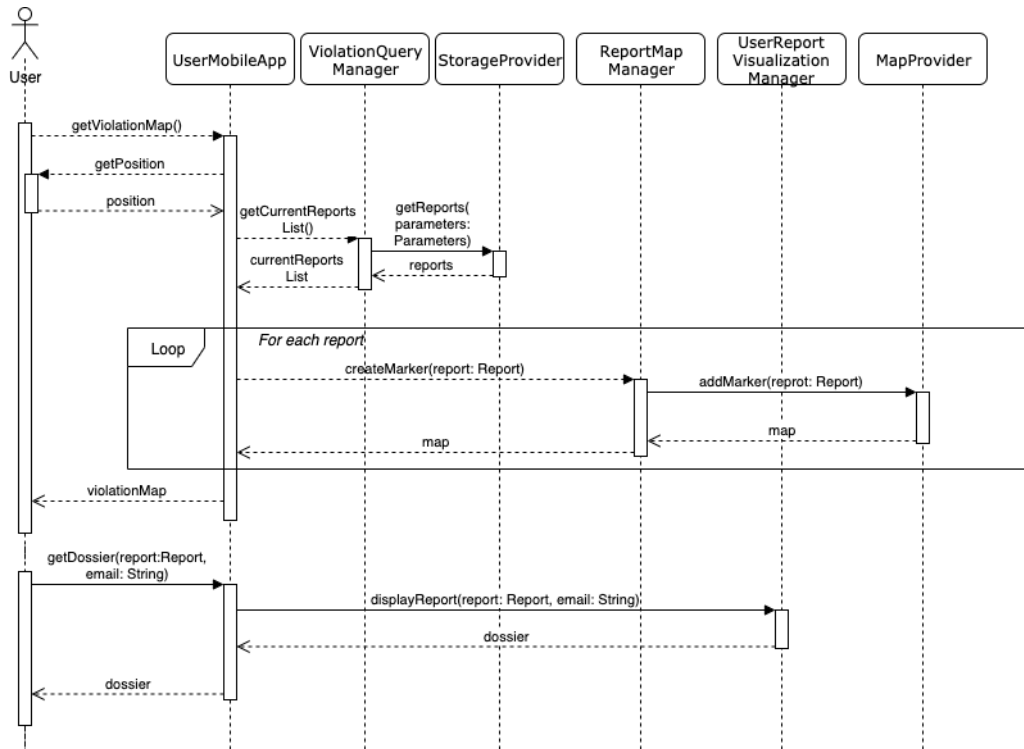


Figure 2.5: Sequence diagram of showing the map with all the reports.

In this sequence diagram is shown the process that provides the user with the daily violation map. It starts with a request from the user, that is followed with the collection of the information about the user position: this information is useful in order to set the map at the user's location as default. Then the database is queried to collect all the report stored in the last 24 hours. Every report is than passed to the ReportMapManager, that marks the position of the report. The map thus obtained is than shown to the user. After that, the sequence diagram shows what happens in case that the user requires the dossier by clicking on the report marker: a request is sent from the UserMobileApp to the UserReportVisualizationManager, that uses the email to get the access level of the user and provides a dossier that contains only the accessible information.

2.4.3 Unsafe Area Request

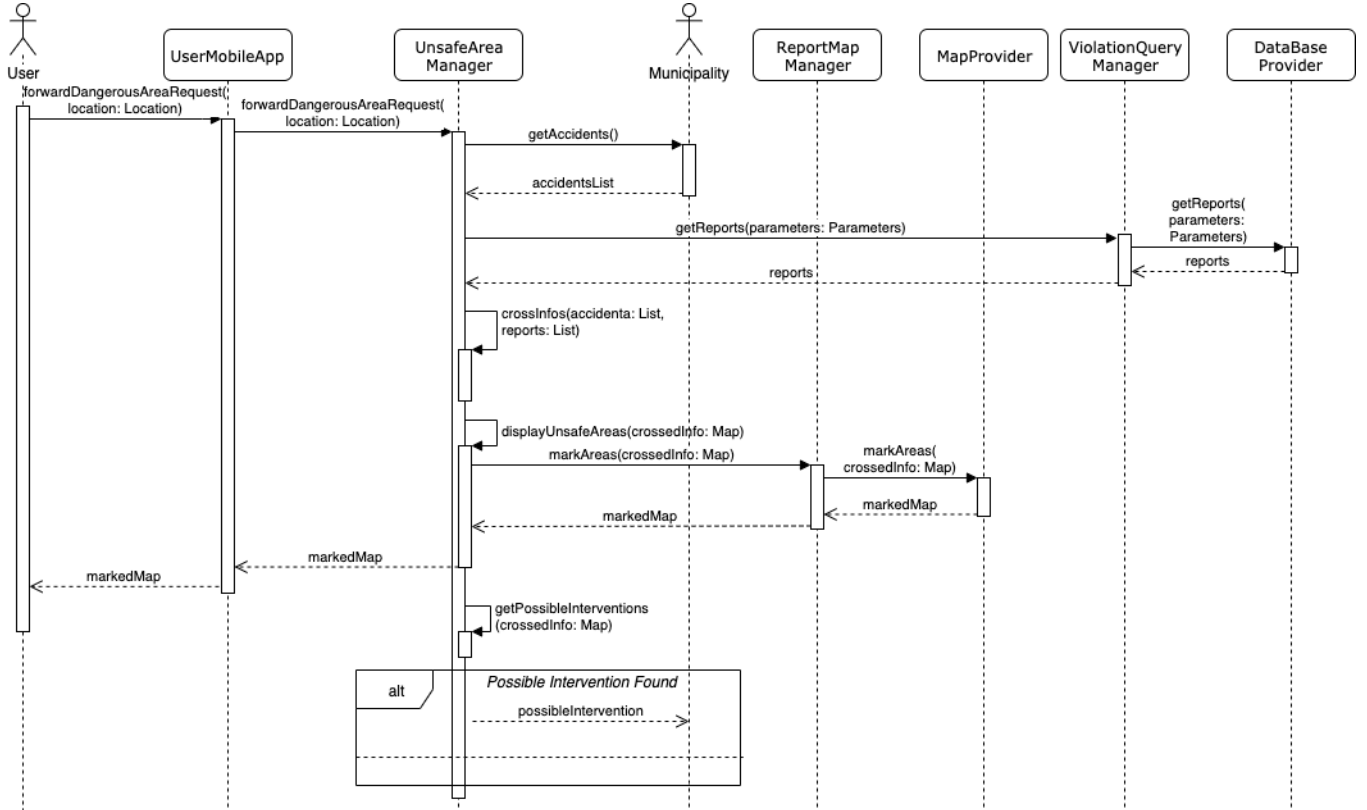


Figure 2.6: Sequence diagram of requesting unsafe areas.

This sequence diagram shows the process that provides the map of the unsafe area in a city, crossing the report collected by SafeStreets and the accidents data provided by the local municipality. The user sends a request, providing the location that would like to interrogate. The UserMobileApp forwards the request to the UnsafeAreaManager, which has the task of managing this particular kind of feature. First, it gets the accident list from the local municipality, then calls the ViolationQueryManager to get all the reported violations in the selected area from the database. This information is now crossed and processed in order to define the possible unsafe areas, by giving different weights to different type of violations. The data produced is then passed to the ReportMapManager, that interacts with the map provider to mark the areas denoted as dangerous. The map is then showed to the user. After that, the UnsafeAreaManager processes the information on the violations and the accidents to find possible solutions, considering the type of the violations reported and the nearness of the accidents to them. If it is able to define possible solutions, these are sent to the municipality.

2.4.4 Send Violation's picture Feedback

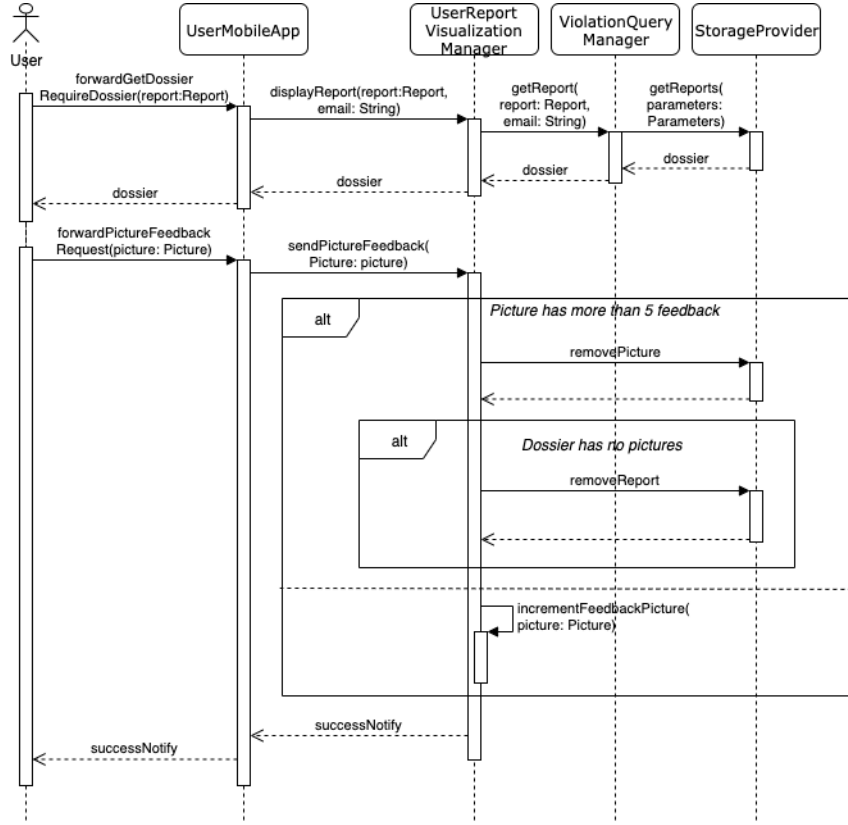


Figure 2.7: Sequence diagram of sending a picture feedback.

This last sequence diagram shows a situation in which a user gives a feedback on a picture of a report. Firstly, the user requires the dossier of a report, that is got from the database by the ViolationQueryManager and showed by the UserReportVisualizationManager, that has the task to show him only the data that are allowed by his authorization level. Secondly the user can decide to send a feedback of a picture: in that case, the information on the picture itself and the email of the user are sent to the UserReportVisualizationManager, that first checks whether the picture has already a feedback associated to the user's email. If it already exists, the operation is aborted; otherwise, the reporting is accepted, and different situations can occur. In the standard scenario, if the picture does not already have 5 feedback, a counter is incremented. Otherwise, if the picture has already collected 5 feedback, it is deleted from the dossier of the violation and from the database; now, if the violation has no more associated pictures, it is entirely deleted from the database, since it has no more photos to validate the report.

2.5 Component Interfaces

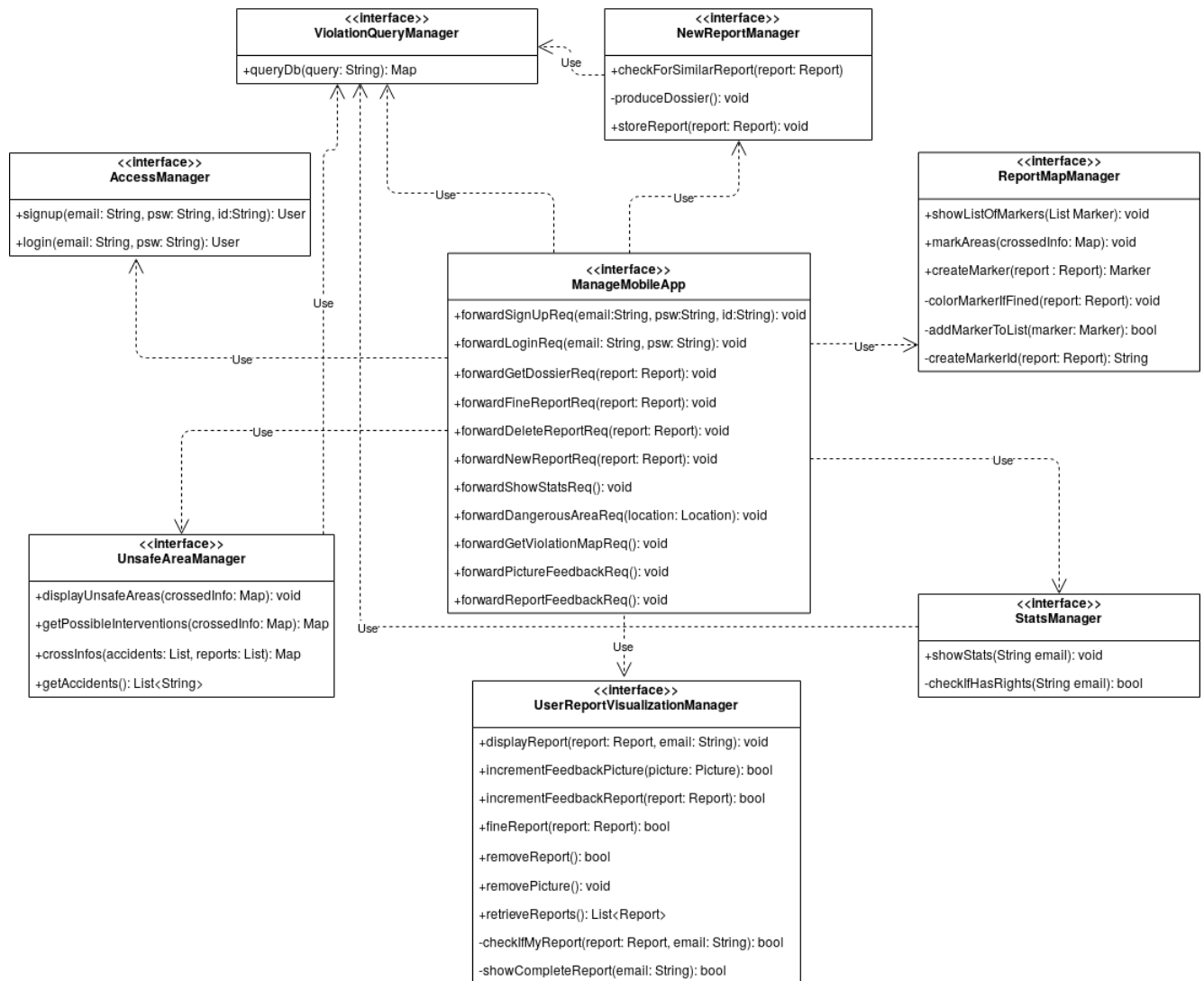


Figure 2.8: UML Interfaces diagram of the application.

All the methods below are not intended to be implemented as they are described, but their aim is to give a general idea of how the component interface will look like. Only the public methods of the managers are described.

AccessManager:

- *Function* signup: given email, password and id(mandatory only if authority), a new instance of User is return by the authentication provider, and the new user is now stored in the database with its email and level(standard or complete, according to

the id provided).

- *Function* login: given the email and password, will return a User object given by the authentication provider.

NewReportManager:

- *Function* checkConstraintsOnReport: this function is used to check for example if at least one image is uploaded, will return true or false.
- *Function* recognizePlates: given a list of images, the application will use an API in order to recognize plates text if present and will be put it in a list, one string for each image.
- *Function* getPosition: this function will be called as soon as the first image is taken, will retrieve the user position using GPS.
- *Function* checkForSimilarReports: this function will make sure that there are no other reports of the same violation in the same spot.
- *Function* storeReport: the report will be then stored in the database, ready to be fetched by the other users.
- *Function* getTime: this function will set the time of the report and will be taken from the device.

ReportMapManager:

- *Function* showListOfMarkers: display all markers created on the map
- *Function* addMarkerToList: given a report, the associated marker is created and return true if no errors occur.
- *Function* markArea: once the crossed info between SafeStreets and the municipality are calculated, the most dangerous areas are marked.

StatisticsManager:

- *Function* showStats: given the current user's email that is logged, the right stats are loaded depending on the level of the user.

UserReportVisualizationManager:

- *Function* displayReport: the report selected on the map will be shown, displaying partial or complete data (visible plates for example) depending on the level of the user.

- *Function* incrementFeedbackReport: given a report, increment the counter of the negative feedback on that report.
- *Function* incrementFeedbackPicture: given a picture inside a report, increment the counter of the negative feedback on that image.
- *Function* sendFeedback: given a report, increment the counter of all feedback on that report.
- *Function* fineReport: this method is available only for authorities, given a report will be marked as fined on the map.
- *Function* removeReport: given a report, if that report belongs to the user who wants to perform that action, it will be deleted. Moreover, this function will also be called when the reports reaches the maximum number of negative feedback.
- *Function* removePicture: when the maximum number of negative feedback on the single image is reached, the picture will be deleted.
- *Function* retrieveReports: this method is called either once the map is loaded or whenever the user taps on the refresh button.

UnsafeAreaManager:

- *Function* displayUnsafeAreas: given a map of information, all the unsafe areas will be displayed on the map
- *Function* getPossibleInterventions: from the map of all crossed information received by the API of the municipality, the possible interventions are created according to a precise algorithm that compares the places of the accidents with the violation's report in the same places.

ViolationQueryManager:

- *Function* queryDb: this method is used both by UnsafeAreaManager and StatisticsManager in order to cross all the information necessary to do their job. In addition to that, it is used by the user to query reports by date or violation type for example.

2.6 Selected Architectural Styles and Patterns

2.6.1 Three Tier Architecture

As already mentioned in section 2.3, we chosen a multi-tiered architecture composed by three tiers: presentation, logic and data. The advantages of this choice are in terms of decoupling of the complexity of the system making him more flexible to future modification and in this way also more reusable. This division also provides more security to the system in fat it separates the access to data from the layer where then logic for presentation and for the interaction with the customer is located.

2.6.2 Serverless Approach

- **Performance:** we want the database access to be fast and in order to do that could be useful to directly query it and get all the tuples locally. Moreover, using this approach the database has a local cache that stores query results for future use. When a client app issues a query, if the SDK determines that the cache contains up-to-date results for that query, the results can come directly from the cache. The obvious benefit here is that network bandwidth and latency are reduced. The results appear faster, even while offline, and the end user pays less data costs to see those results. If you make the request indirectly through a server, there is absolutely no client-side caching done by default. If you want to cache the results, you'll have to do that on the client, using some mechanism you choose.
- **Price:** is tied (partly) to how much data the app reads. As mentioned previously, the local cache can prevent many data reads from happening on the server. Data coming from cache, unchanged on the server, prevents the cost of the query and the cost of the bandwidth to bring that data to the app. If you query indirectly through a server in the middle, you will pay for the cost of the query in addition to the cost of the execution of the function. Usually the server SDKs you use do not cache data, so each execution pays the full cost of the query, and its data transfer.

2.6.3 MVVM (Model-View-ViewModel)

MVVM facilitates a separation of development of the graphical user interface - be it via a markup language or GUI code - from development of the business logic or back-end logic (the data model). The view model of MVVM is a value converter, meaning the view model is responsible for exposing (converting) the data objects from the model in such a way that objects are easily managed and presented. In this respect, the view model is

more model than view, and handles most if not all of the view's display logic. Unlike the controller method (MVC pattern), the ViewModel method (MVVM) relies heavily on the frontend of the application. ViewModel acts as a binder that binds data between the view and model. Whereas the MVC format is specifically designed to create a separation of concerns between the model and view, the MVVM format with data-binding is designed specifically to allow the view and model to communicate directly with each other. This is why single page applications work very well with ViewModels. It's simple and allows the view to directly communicate with the backend. Because of this, MVVM single page applications can move quickly and fluidly and save information to the database continuously (Google Docs would be a perfect example).

2.6.4 RESTful Architecture

The communication between SafeStreets and their users is done via HTTP requests following REST principles. REST (Representation State Transfer) is an architectural style for communication based on strict use of HTTP request types. One of the most important REST principles is that the interaction between the client and server is stateless between requests. Each request from the client to the server must contain all of the information necessary to understand the request. The client would not notice if the server were to be restarted at any point between the requests. The RESTful HTTP requests are categorized according to method types as the following:

- GET: used to retrieve resource representation/information only - and not to modify it in any way.
- POST: used to create a new resource into the collection of resources.
- PUT: used primarily to update existing resource (if the resource does not exist then API may decide to create a new resource or not).
- DELETE: used to delete resources (identified by the Request-URI).

2.7 Other Design Decisions

Choice of Providers

Starting from security and scalability reason, we have decided to lean on different providers for the more complexes and sensitives functionalities of SafeStreets.

- **Authentication Provider:** Regarding the authentication provider we want a service that can:

1. Register new users
2. Authenticate the registered ones

The best effort is the encryption of the password that is useful in terms of security.

- **Map Provider:** It's fundamental in SafeStreets app for display report on a map. Furthermore the map provider must be able to displays the position of the user and of the reports on the map.
- **Plate Recognizer Provider:** For recognize a plate from a picture, we decide to choose a provider that it's be able from a photo to keep the plate (text format).
- **Database Provider:** Database system are complex, difficult and time-consuming to design also for the developers requires an initial training so from this point of view is more convenient to take one ready-made. We want a database that stores reports with all their information and also be able to retrieve information about one user from his mail.

USER INTERFACE DESIGN

In this chapter is given a general view about the structure of the application. A user flow diagram is presented to clarify the possible actions that a user can perform while using the application. This section is built according with the mockups presented in the RASD; regarding that, some windows has been added to better clarify the flow and to give a more complete idea of the user experience inside the application.

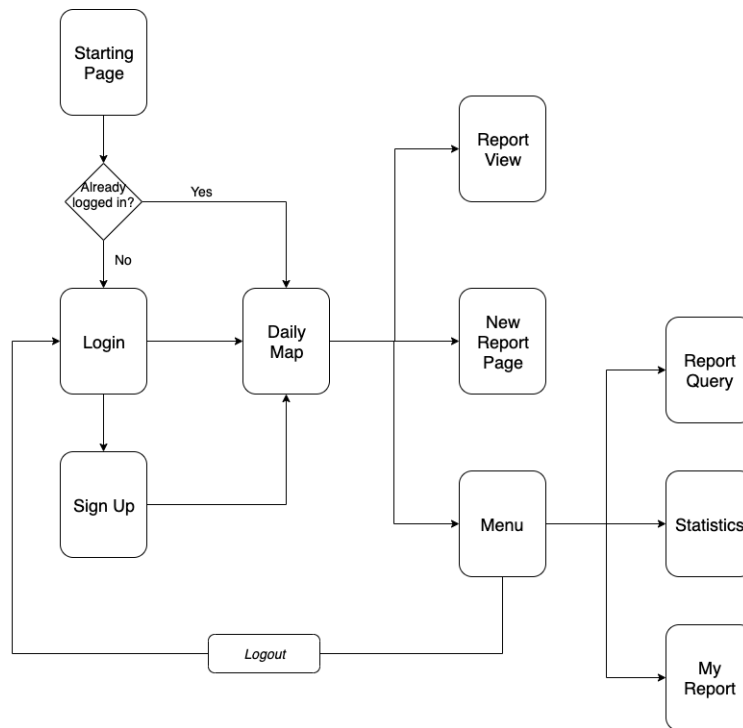


Figure 3.1: UX diagram of the application.

It can be useful to analyze some aspects of the diagram. First of all, the starting page is not a real window of the application, but just denote the first page shown to the user after the application launch: in particular, if the user already used the application without logging out, he will directly see the daily map, otherwise if he logs out or if it is the first time that he uses the application on that device, the login page will be shown. Another important thing to notice is that the main page of the application is not the menu, but daily map, which is considered the core of the application itself. From this page will be than possible to switch to some related functionalities or to the main menu. It is important to clarify that this diagram wants to show the structure of SafeStreets,

that will find a corresponding in the real application, but some more minor features not showed could be added later.

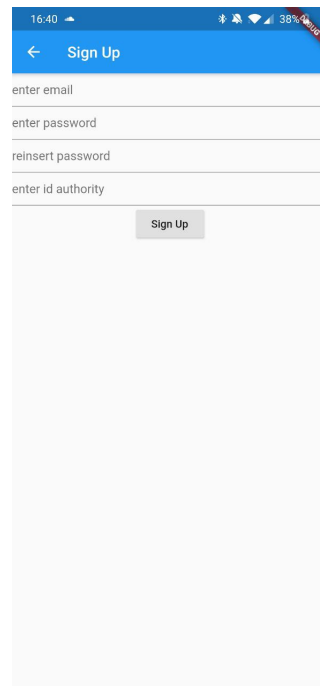
A screenshot of a mobile application's registration page. The status bar at the top shows the time 16:40, signal strength, Wi-Fi, and 38% battery. The app's header is blue with a back arrow and the text "Sign Up". Below the header are four text input fields with labels: "enter email", "enter password", "reinsert password", and "enter id authority". A grey "Sign Up" button is positioned below the "enter id authority" field.

Figure 3.2: The registration page of SafeStreets.

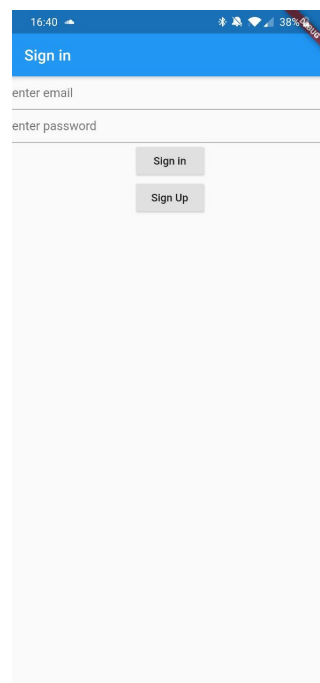
A screenshot of a mobile application's login page. The status bar at the top shows the time 16:40, signal strength, Wi-Fi, and 38% battery. The app's header is blue with the text "Sign in". Below the header are two text input fields with labels: "enter email" and "enter password". Below the "enter password" field are two grey buttons: "Sign In" and "Sign Up".

Figure 3.3: The login page of SafeStreets.

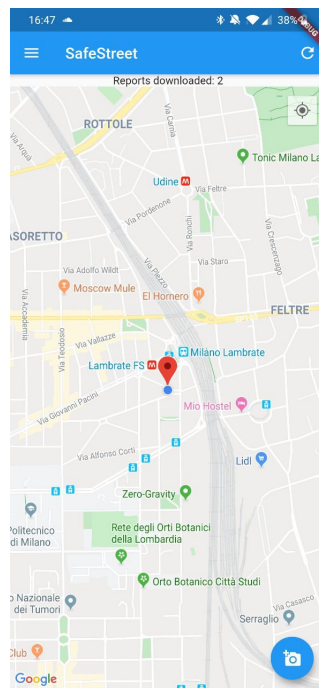


Figure 3.4: The homepage page of SafeStreets.

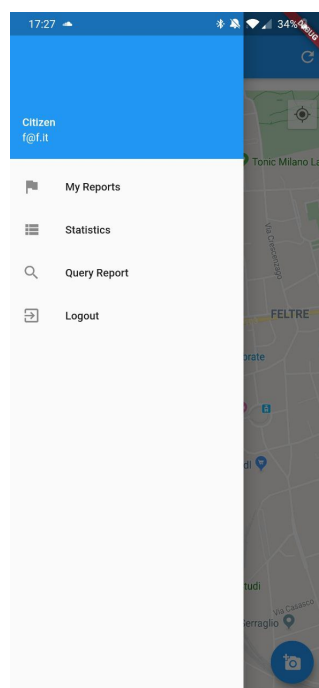


Figure 3.5: Menu drawer in SafeStreets.

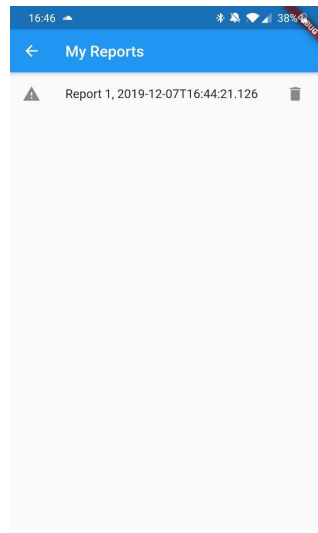


Figure 3.6: My reports page of SafeStreets.

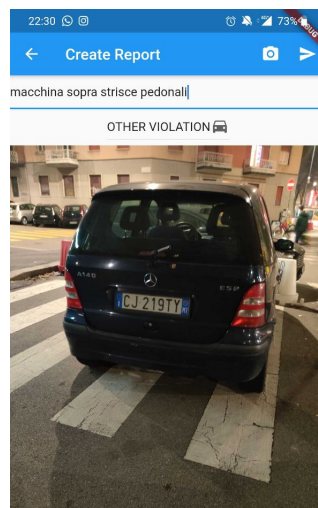


Figure 3.7: The create report page of SafeStreets.



(a) Report View for a citizen in SafeStreets.



(b) Report View for an authority in SafeStreets.

Figure 3.8: Report View

Note that these screenshots represent only the prototype of the application that will be developed afterwards, they are not intended to give a strict user interface design.

REQUIREMENTS TRACEABILITY

Below is shown a mapping between requirements defined in the RASD and the design components in the logic application that will ensure their fulfillment in a direct way. Other components that are indirectly needed to enforce some requirements are not mentioned in the list, but their role has been made clear through some comments:

[R1] Unregistered users cannot use SafeStreets.

- **Access manager** and authentication provider : with the services offered by authentication provider, Access manager checks if a user is registered or not.

[R2] In signing up, users must provide an email and a password.

- **Access manager**: it checks if all the fields of the registration are filled in the correct way.

[R3] In signing up, an authority must also provide an ID.

- **Access manager**: it checks if all the fields of the registration are filled in the correct way.

[R4] In signing up, users must accept “terms & conditions”.

- **Access manger** and authentication provider: in sign up access manager shows “terms & conditions” to the user.

[R5] SafeStreets identifies a user by its email.

- **Access Manager**: each user has an identifier which is the user mail.

[R6] SafeStreets collects user data in its database.

- **Access manager** and database provider: for each user access manager stores user mail in the database provider to keep an association between the user and his reports.

[R7] SafeStreets stores all reports in its database.

- **NewReport manager** and database provider: when user adds a new report, it is stored through the database provider.

[R8] SafeStreets retrieves reports from its database.

- **ViolationQuery manager** and database provider: this manager contains all the procedures to make a query on the database to retrieve some specific data.

[R9] SafeStreets authenticates a user when tries to log in.

- **Access manager** and authentication provider: through the services offered by the authentication provider, access manager logs in a user.

[R10] Users can view the reports in the map.

- **ReportMap manager** and map provider: this requirements is the main functionality of this manager. It displays all the daily reports on the map.

[R11] Authorities can see all the sensitive information contained in a report.

- **UserReportVisualization manager**: this manager, based on the type of user, can show or not the sensible information of one report. For the citizens shows the images with the plate blurred.

[R12] Citizens cannot see sensitive information in a report.

- **UserReportVisualization manager**: this manager, based on the type of user, can show or not the sensible information of one report. For the citizens shows the images with the plate blurred.

[R13] Users can view unsafe areas in the map.

- **Unsafe area manager** and map provider: this manager shows unsafe areas on the map archived from the map provider.

[R14] Users can delete reports they submitted within a day.

- **UserReportVisualization manager**: this manager manages all the operation on one report, in this case the deletion of a report.

[R15] Users can edit reports they submitted within a day.

- **UserReportVisualization manager**: this manager manages all the operation on one report, in this case the editing of a report.

[R16] Users can upload *valid* reports.

- **NewReport manager:** this manager allows one user to upload a report.

[R17] A report must contain at least one image.

- **NewReport manager:** this manager checks the completeness and the correctness of one report that it's going to be added.

[R18] A report must indicate the type of violation.

- **NewReport manager:** this manager checks the completeness and the correctness of one report that it's going to be added.

[R19] A report is *valid* if and only if R17 and R18 are satisfied.

- **NewReport manager:** this manager checks the completeness and the correctness of one report that it's going to be added.

[R20] Images containing sensitive information (like license plates) must be blurred by the system.

- **NewReport manager** and plate recognizer provider: this manager, using the plate recognizer provider, is able to get the pixels of the plates on the images uploaded with a report.

[R21] The system must notify the user if the report submitted is not valid.

- **NewReport manager:** this manager checks the completeness and the correctness of one report that it's going to be added. If it does not respect all the constraints, the user will be notified.

[R22] For each type of statistics there must be at least one user from which SafeStreets takes data.

- **Statistics manager:** this manager checks all the conditions to build one type of statistic.

[R23] A user can access to feedback area for each report.

- **UserReportVisualization manager:** this manager manages all the operation on one report, in this case the feedback area of a report.

[R24] A user can select negative feedback for each report.

- **UserReportVisualization manager**: this manager manages all the operations on one report, in this case the feedback area of a report.

[R25] Authorities can see suggestions for possible interventions.

- **UnsafeArea manager**: for each unsafe area, this manager suggests possible interventions to authorities. The suggestions are based on the type of unsafe area (type of most accidents committed).

[R26] During adding a new report, if there is a report of same type, in the same position and at the same date, SafeStreets shows the report stored to the user if the violation is the same.

- **NewReport manager** and database provider: this manager checks the correctness of a new report.
- **ViolationQuery manager** and database provider: this manager checks if a report that is going to be added has the same position and data of at least one report already stored in the database.

[R27] Authorities can mark a report as “fined”.

- **UserReportVisualization manager**: this manager manages all the operation on one report, in this case the specific “fined field” of a report that is available only for the authorities.

[R28] User can query SafeStreets to achieve a specific report.

- **ViolationQuery manager** and database provider: Users have a dedicated area in the application where they can make a query by setting different fields (data, type, zone).

[R29] SafeStreets retrieves incidents from municipality database.

- **Statistics manager** and database authority: It retrieves data using database of the authority in which are stored data about accidents.

R	Access manager	New Report manager	Report Map manager	Statistics manager	Unsafe Area manager	UserReport Visualization manager	Violation Query manager
1	Yes						
2	Yes						
3	Yes						
4	Yes						
5	Yes						
6	Yes						
7		Yes					
8							Yes
9	Yes						
10			Yes				
11						Yes	
12						Yes	
13					Yes		
14						Yes	
15						Yes	
16		Yes					
17		Yes					
18		Yes					
19		Yes					
20		Yes					
21		Yes					
22				Yes			
23						Yes	
24						Yes	
25					Yes		
26		Yes					Yes
27						Yes	
28							Yes
29				Yes			

Table 4.1: Traceability matrix

IMPLEMENTATION, INTEGRATION AND TEST PLAN

In this section there is the order in which we plan to implement the sub-components of our system and the order in which we plan to integrate such sub-components and test the integration.

5.1 Component implementation

The system development will only concern the UserMobileApp, since the server, the map and the storage will be provided by external companies. The system will be implemented, but also tested and integrated, following a bottom-up strategy. The external system's components will not need to be implemented and tested, since they can be considered reliable. Nonetheless, the interaction with these systems will need to be tested as well, to check that it has been properly interfaced with SafeStreets. Since the features have different importance in the context of the application, the following table will resume some consideration on each functionality in the context of the application, the user and the development. In particular, the importance for the customer will consider the impact on the customer itself in his experience inside the application and the reasons that could bring him to join and use SafeStreets; the importance for SafeStreets will consider the possible impact of SafeStreets in the market, the use that it could do to improve the system itself and the possible legal aspects; the implementation difficulty will provide a qualitative estimation of the amount of work and time that will be needed to develop that particular feature.

Feature	Importance for the customer	Importance for SafeStreets	Implementation difficulty
Sign up and login	Low	Medium	Low
Violation reporting	Very High	Very High	Medium
Visualization of customer's own reports	Medium	Low	Low
Daily map visualization	High	High	High
Query all the violations	High	Medium	High
Violation Visualization	High	High	Low
Show/hide sensitive information	High	High	High
Give a feedback to a violation	Medium	High	High
Build statistics on the violations	Low	Medium	Medium
Mark dangerous area	High	Medium	High
Provide suggestions about unsafe areas	Low, for the users High, for the municipality	Low	Medium
Mark violations as fined	Medium	Medium	Low

Table 5.1: *Features importance*

It is important to mark again that the core functionality of SafeStreets will be the violation reporting, and since almost all the other functionalities are based on the reported violations it is important to develop it as first. According with the considerations done, the features will need to be implemented and unit tested with the following order:

- **Violation reporting:** as previously said, this is the core functionality of the application. To develop this feature, the NewReport Manager will need to be partially developed and tested. Moreover, the correct integration in the system of the server and database must be ensured, at least in the upload task.

- **Sing up and login:** even if this feature is not considered to have an high impact neither for the user nor for safe street, it is important to implement it as soon as possible to match reports with the people who upload them. At this point, the NewReport Manager must be partially implemented and tested even in the functionality that concern the user matching.
- **Violation visualization:** this is one of the most important features of the system, and so it needs to be developed as soon as possible. To ensure that it work correctly, the ViewReport Manager must be partially implemented and tested, and the correct integration with the database and the server must be ensured even for what concerns the download functionalities.
- **Query all the violations:** This is another core functionality of the system, and so must have a high priority in developing it. Even if the main part of this functionality is done by the database provider, it is important to implement and meticulously test the ViolationQuery Manager, since it is used by other managers.
- **Daily map visualization:** to provide this functionality, the ViolationMap Manager must be almost fully implemented and tested, and its integration with the ViolationQuery Manger (already implemented for the violation query feature) must be tested as well to ensure that they interact correctly. The integration with the map provider must also be checked to work correctly.
- **Visualization of customer's own reports:** this is not a core functionality of the system; however, since it could be considered important for some of the users and most of its components should be already have been implemented at this point, it could be useful to add it to the application. The ViolationQuery Manager is the only component on which this feature is based, so the last feature functionality of this manager must be implemented if needed and eventually tested.
- **Show/hide sensitive information:** this is the most critical feature of the whole system, since if it does not work correctly the sensitive information will risk being shown to all the users. Because of that, the NewReport Manager must be fully implemented at this point, as well as the integration of the plate recognizer, and they must be accurately tested.
- **Give a feedback about a violation:** this feature is not crucial but is important to let the community limit the wrong reporting. To do that, the UserReportVisualization Manager must be implemented in all the functionalities that concern the feedback upload, and they must be tested as well.

- **Mark dangerous area:** this feature is one of the most difficult to implement but is at the same time one of the most useful for the users. An algorithm must be designed to identify the potential unsafe area, and the UnsafeArea Manager must be implemented in all the functionality that are used to define them. Because of the possible difficulties that could rise in the development phase, the testing of this feature must be very accurate.
- **Provide suggestions about unsafe area:** to provide this feature, the UnsafeAreaManager must be implemented with the last functionalities that are missing from the previous point. The algorithm to identify the possible interventions must be designed as well, and everything must be finally tested.
- **Mark reports as fined:** even if this is one of the easiest features to develop, it is less important than the others for both the users and SafeStreets, and so is one of the last to be developed. The UserReportVisualization Manager must implement the last functionality to enable this feature, and they must be tested as well.
- **Build statistics on the violations:** this is the last feature to be implemented, because it will use the data collected by all the other functionalities and would be less important without them. The Statistics Manager must be fully implemented and tested.

It is important that the testing run in parallel with the implementation of the system whenever is possible, in order to spot possible errors as soon as possible and avoid propagating them on the following features. Whenever the integration of two or more components have to be integrated, it is important that will be unit-tested first, to limit as much as possible the presence of errors and bugs. Once the system will be fully integrated, it will have to be tested again to test that all the functional and non-functional requirements hold. Moreover, since the system is thought to work with a big amount of data, some performance tests will have to be done. Even if most of them will be possible only once that the system is fully integrated, these tests must be introduced as soon as possible, to detect possible data structure or algorithm that may cause a bad computation of data.

5.1.1 Provider choice

- *Requirements:*

- **Authentication provider:**

Regarding the authentication provider we want a service that can:

1. Register new users
2. Authenticate the registered one

The best effort is the encryption of the password that is useful in terms of security.

- **Map provider:**

It's fundamental in SafeStreets app to display reports on a map. Furthermore, the map provider must be able to display the position of the user and the reports on the map.

- **Plate recognizer provider:**

In order to recognize a plate from a picture, we decided to choose a provider that it's able from a photo to return a readable file with plate and pixels of it as strings (if present).

- **Database provider:**

Database systems are complex, difficult and time-consuming to design and for the developers require an initial training, so from this point of view is more convenient to take one ready-made. We want a database that stores reports with all their information and also be able to retrieve information about one user from his mail.

- *Implementation choice:*

To satisfy all these requirements we choose to lean on *Firebase* for the authentication and database functionality. *Firebase* is a Backend-as-a-Service (BaaS) app development platform that provides hosted backend services. One of them is *Firebase Auth* that provides backend services to authenticate users to our app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more. Another services that comes from *Firebase* is *Cloud Firestore*. It is a flexible, scalable NoSQL cloud database to store and sync data for client- and server-side development. In *Cloud Firestore*, SafeStreets can use queries to retrieve individual, specific documents or to retrieve all the documents in a collection that match our query parameters. In this way SafeStreets queries can include multiple, chained filters and combine filtering and sorting. They're also

indexed by default, so query performance is proportional to the size of the result set, not of the data set. *Firebase Storage* provides secure file uploads and downloads, regardless of network quality. The developer can use it to store images, audio, video, or other user-generated content. For the map provider we choose *Google Maps API*. It provides detailed and accurate maps of all the world. It also allows to customize the map for instance by adding markers. For the plate recognizer service we choose *Plate Recognizer*. It takes an image as input and produces an easy to use JSON response with the number plate value of the vehicle and the pixels's location of that plate.

5.2 Component Integration

Each component should be integrated with the others only after it's almost totally completed and with a satisfying unit testing's result in order to avoid and contain the impact of each possible error, fault and failure on the system. For the same aim, right after a component has been integrated with the system, the relative integration testing should be made. The diagrams in the next sections show which components will go through the process of integration for a further clarification. The arrows start from the component which "uses" the other one.

5.2.1 Integration of the Application logic component

Managers will be integrated with User mobile app and this will be the complete logic application.

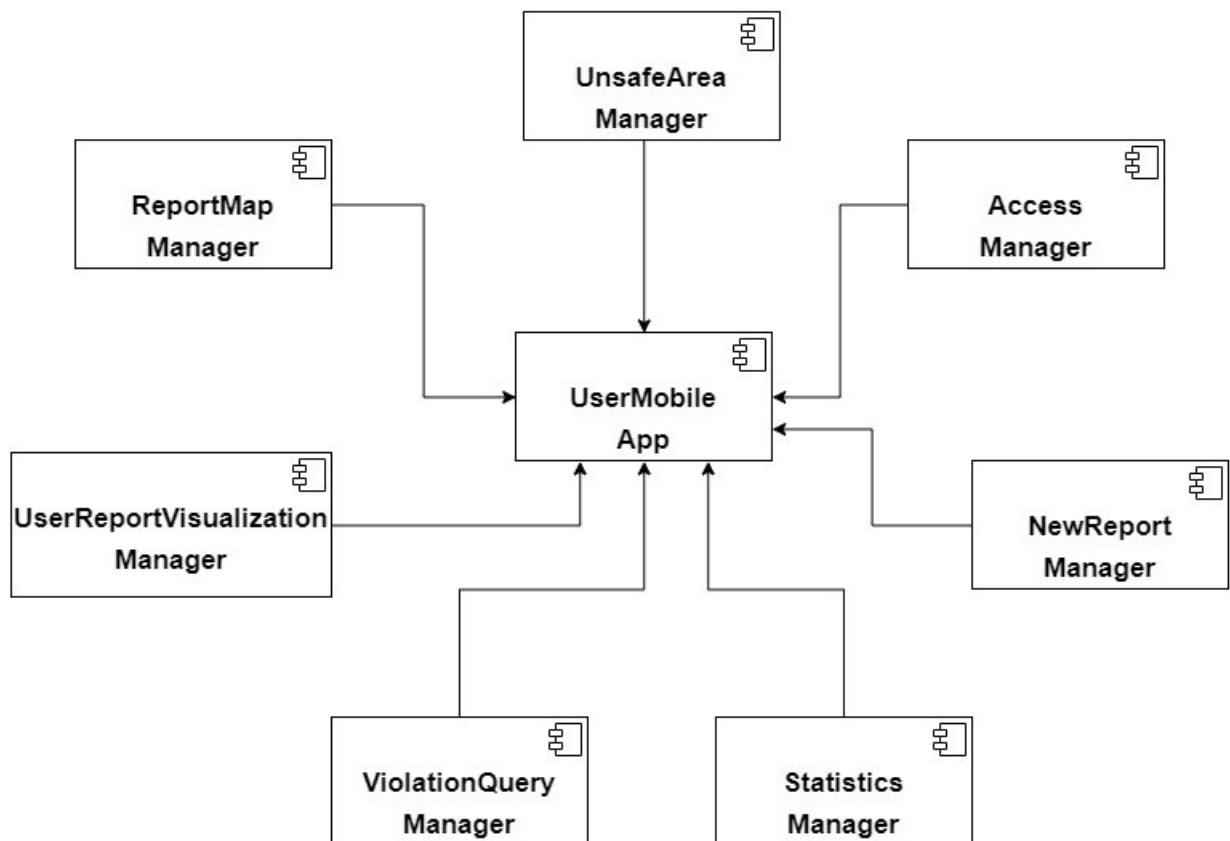


Figure 5.1: Integration diagram of the application login.

5.2.2 Integration of the internal component of the application logic

Components that use ViolationQuery functionalities are going to be completed only after they will be integrated with ViolationQuery manager.

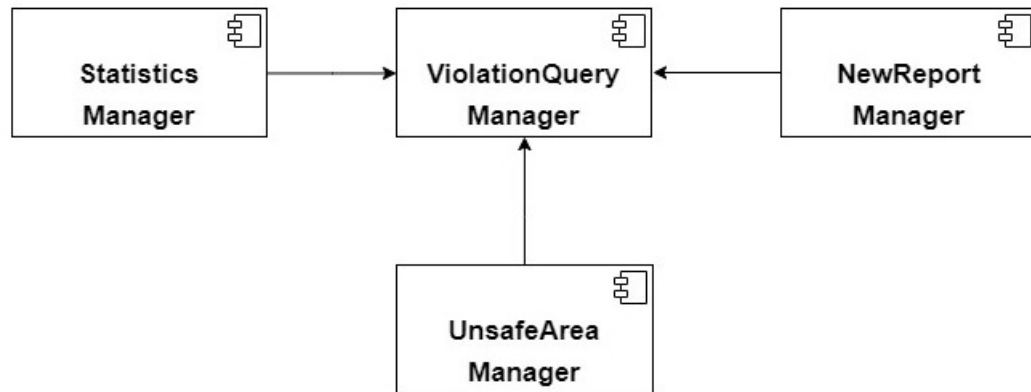


Figure 5.2: Integration diagram of the internal component.

5.2.3 Integration of the external services

The integration of the external services is done only after each internal component is developed and tested sufficiently.

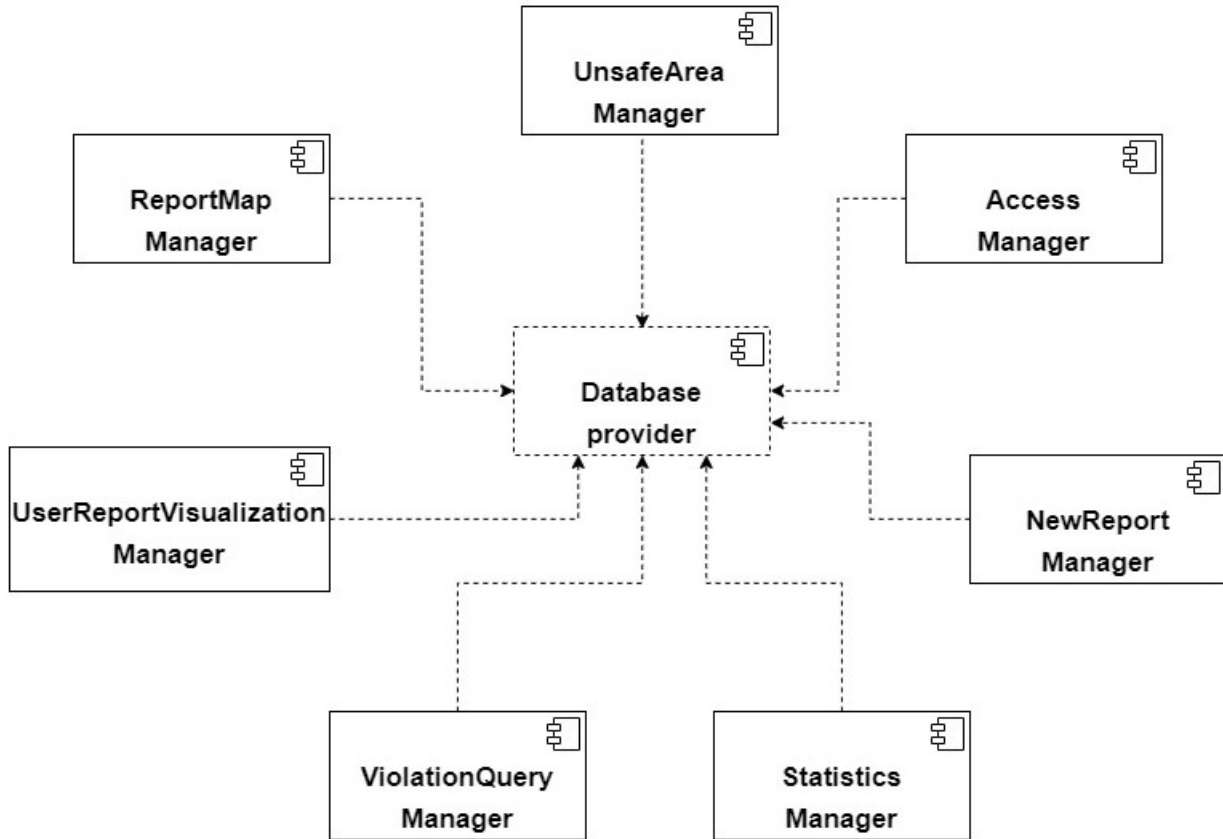


Figure 5.3: Integration diagram of the database provider.



Figure 5.4: Integration diagram of the authentication provider.

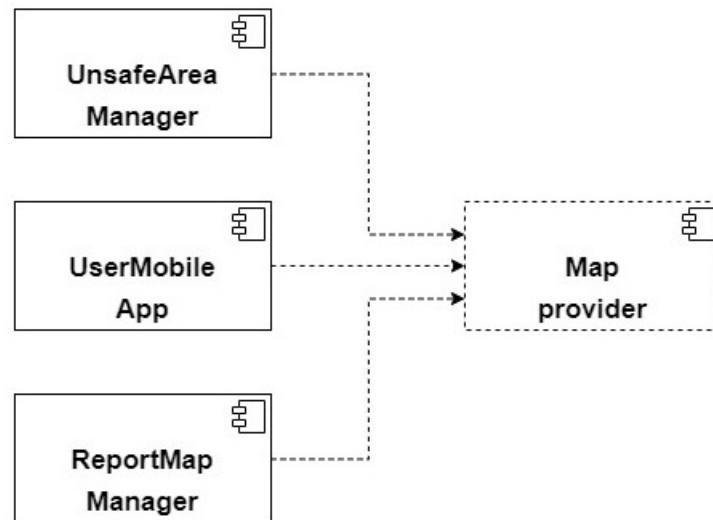


Figure 5.5: Integration diagram of the map provider.



Figure 5.6: Integration diagram of the plate recognizer provider.

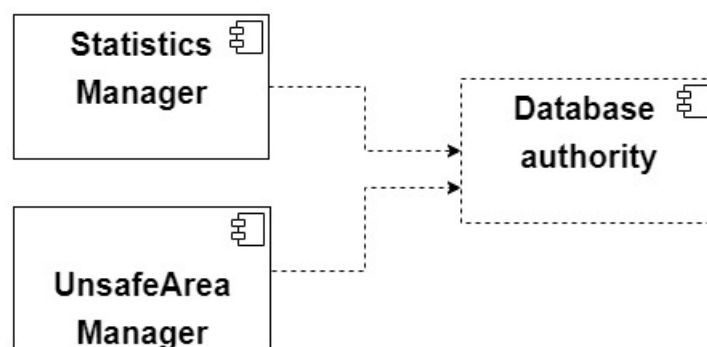


Figure 5.7: Integration diagram of the authority database.

5.3 Algorithms

This section will provide some general indication about the significant algorithms that will be developed within the system.

5.3.1 Unsafe Area Search

Determine the unsafe areas is one of the most critical aspect of the system, because of the lack of a objective criteria to determinate when an area is more dangerous than another. The data that will be used for this algorithm are the reports collected by SafeStreets and the accidents provided by the municipality. Each of them will be associated to some parameters that will be used to evaluate the impact that they have on the danger level of the area; for simplicity they will both be referred as reports from now on.

- **dangerLevel:** every report category will be associated with a value that will set the basic danger level of the area.
- **timePassed:** is a coefficient that will be decreased with the time passed from the reporting. The idea is that a report will be less relevant as the time from its occurrence passes.
- **influenceDistance:** is a parameter that sets the radius of the circle in which the report has a relevant influence. It can be different depending on the report category.

Since the accidents will be provided by the municipality, it will be possible that some information will miss: in that case, a default value will be assigned. The algorithm will work as following: every accidents will mark an unsafe area defined by its influenceDistance. To these areas will be assigned a value equal to their dangerLevel, eventually corrected with the timePassed parameter, and if some areas will overlap their danger level will be summed. Finally, the violations reported in the marked zone will be used to increment the unsafe level, according to their parameters. After processing the data, the system will check for the violations in the previously determined area that have a danger value beyond a certain threshold and will provide a default solution for the most reported category.

EFFORT SPENT

Team Work	
Task	Hours
Brainstorming	2
Planning Architecture	5
Choosing Patterns	2
Software Development Process	4
External interfaces choice	2
Total	15

Table 6.1: *Time spent* by all team members

Individual Work					
Morreale Federico		Maddes Evandro		Innocente Federico	
Task	Hours	Task	Hours	Task	Hours
Overview	1	Introduction	1	Run-Time View	8
Component Interfaces	7	Component View	8	User Interface design	3
Architectural Styles	7	Deployment View	5	Component Implementation	6
User Interface Design	5	Requirements Traceability	5	Algorithm	4
Spelling check	2	Provider Description	3	Spelling check	2
Patterns	3	Integration Plan	2		
Total	25	Total	24	Total	23

Table 6.2: *Time spent* by each team member

REFERENCES

- This document follows ISO/IEC/IEEE 29148:2011 and IEEE 830:1998 standard for software product specifications. All the specifications of this project have been given by Rossi and Di Nitto for the Software Engineering 2 Mandatory Project 2019-2020.
- MVVM
- Firebase: Firebase Auth, Firebase Storage and Cloud Firestore
- Google Maps API
- Plate Recognizer API