POLITECNICO
MILANO 1863

AA 2019/2020

Computer Science and Engineering
Software Engineering 2 Project

SafeStreets

DD

Design Document

version 1.0 --- 09/12/2019

Authors:
Aida Gasanova
Alexandre Batistella Bellas
Ekaterina Efremova

Professor:
Elisabetta Di Nitto

Table of Contents

1 Introduction

1.1 Purpose

This document represents the Design Document (DD). The main purpose of this document is to fully describe the technical details of the SS application.
In comparison with the RASD, while it has the purpose to describe the functionalities, the requirements and the goals of the system, the DD specifies the design aspects of the system, defining the architecture, the main components and your interfaces, the runtime behavior, the design patterns, and the information about implementation, integration and testing. With these aspects, this document gives a wide knowledge about the description of the components forming the system to be developed.

1.2 Scope

Reviewing from the RASD, the SS service is a crowd-sourced application offered to common users and authorities that want to follow the violations occurred on the municipality territory.

Figure 1 *– SS as a sharing information system*

The system has various directions of action. First, the largest mass of users are ordinary people. Everyone can download the application to their phone and use it to the benefit of the welfare of their city. authorities

Every time when the citizens see the violation, they can take a photo and upload it to this application. Based on this information, a register of violations will be compiled for mapping threats on the streets of the city.

When the app receives a picture, it runs an algorithm to read the license plate, and stores the retrieved information with the violation, including also the type of the violation and the name of the street where the violation occurred. In addition, the application allows both end

users and authorities to mine the information that has been received, by highlighting the streets (or the areas) with the highest frequency of violations, and the vehicles that commit the most violations. In this case there are more user's levels lake the municipality and authorities, and different levels of visibility are offered to different roles.

Another, also a very important part of users is the municipality. Its role is divided into two parts. First, the municipality can upload accident information to the application, thereby complementing existing databases. The application mixes information from users and from the municipality. As a result, we get a new map with more relevant information about the city. Secondly, the municipality can receive information about offenses from the application. This application initially checks the accuracy of the information (including date, Photoshop using, etc.). Based on this information, the municipality, in cooperation with the police, may issue fines.

Police could also offer a service that takes the information about the violations coming from SS and generates traffic tickets from it. In this case, mechanisms should be put in place to ensure that the chain of custody of the information coming from the users it never broken, and the information is never altered.

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

- **User:** the "normal" customer of the application that send the information about the violations to authorities or extract the information that have been received (to use it for useful purposes);
- **Authorities:** the customer of the application that receive the information about violations that have been received from "normal" customers;
- **Customer:** general SS customer;
- **Municipality:** a town or district that has local government;
- **Violation:** general traffic violation, and in particular parking violation;

1.3.2 Acronyms

- **API:** Application Programming Interface
- **GPS:** Global Positioning System
- **HTTP:** HyperText Transfer Protocol
- **JSON:** JavaScript Object Notation
- **UI:** User Interface
- **DD:** Design Document
- **RASD:** Requirements Analysis and Specifications Document
- **SS:** SafeStreets

1.3.3 Abbreviations

- **Rn:** n-th functional requirement

1.4 Revision history

- 1.0.0 – Release version

1.5 Reference documents

- Specification document: "Mandatory Project Assignment AY 2019-2020"
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications
- UML diagrams: https://www.uml-diagrams.org/
- Alloy doc: http://alloy.lcs.mit.edu/alloy/documentation/quickguide/seq.html

1.6 Document structure

The DD is composed by five chapters, as outlined below.

Chapter 1 is an introduction to the design document. Its goal is to explain the purpose of the document and to highlight the differences with the RASD, whilst showing the link between them.

Chapter 2 aims to provide a description of the architecture design of the system, it is the core section of the document. More precisely, this section is divided in the following parts:

- Overview: High-level components and their interaction
- Component view
- Deployment view
- Runtime view
- Component interfaces
- Selected architectural styles and patterns
- Other design decisions

Chapter 3 specifies the user interface design. Actually, this part is already contained in the RASD in the mockups' section, but it is reviewed in this document.

Chapter 4 provides the requirements traceability, namely how the requirements identified in the RASD are linked to the design elements defined in this document.

Chapter 5 includes the description of the implementation plan, the integration plan and the testing plan, specifying how all these phases are thought to be executed.

Chapter 6 shows the effort which each member of the group spent working on the project.

2 Architectural design

2.1 Overview: High-level components and their interaction

The SS application will be based on the three-tier client server architecture, in such a way that will be separated in

- (C) Costumers layer, that handles the users and authority interaction with the system;
- (A) Application layer, that handles the business logic of the application and its functionalities;
- (D) Database layer, that contains the database itself and manages the access to it.

These separations are made aiming scalability and flexibility for the application, since that with the application layer, data can be easier processed with security and assertiveness, even more when dealing with sensible data. Besides this, the layer has one node dealing with the requests from users and authorities, and another node dealing with the interaction with the municipality's system. That's why there's no specification of layer about municipality: it is not included on the project development. By now, it is supposed to be already implemented by another development team and will only be used as a ready part of the complete project.

The image below shows a high-level architecture of the system for simple representation, with informal information about the functionality of the system.

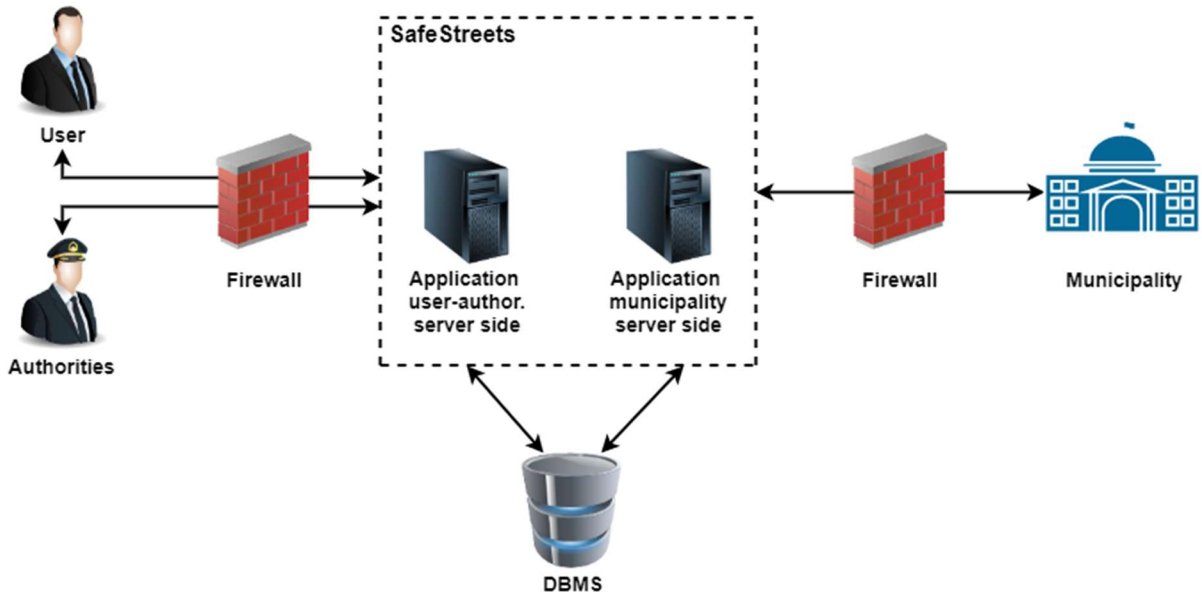

Figure 2 *– Simple architecture representation of the system*

In the Figure 2, the Costumers Layer is represented by the User, Authorities and Municipality agents, while the Application Layer is represented by the server-side agents, and, lastly, the Database layer is represented by the DBMS itself. For instance, the User and the authorities will interact with the Application Layer via mobile application, passing all the data through a Firewall to guarantee the data transfer safety between the agents. The same is applied for the Municipality, with the only difference that the communication is done by API

interaction. No Firewalls are necessary between the Application Layer and the Database Layer, since that all of them will be maintained together by the development team. If the DBMS would be from a cloud service, so it should have a Firewall between the Application Layer and it, but it is not the case.

About the functionality, both users and authorities can communicate through their devices with the Application Layer to make queries and retrieve data in a synchronous message flow, since they need the information to display in their mobile application after querying, while the answer given to the municipality needs to be asynchronous from the point of view of the Application Layer. As the municipality is not related to the system development, the retrieve information from the system is handled by the municipality developers through the existent API interaction. About the users, they can send photos about violations in an asynchronous way, and both users and authorities can update their profile also in an asynchronous way. The municipality, by the way, receives information asynchronously from the Application Layer, and has its information read synchronously. Lastly, the Application Layer communicates with the Database Layer synchronously for reading data and asynchronously for storing data.

Aiming the scalability of the application, two server instances inside the Application Layer were chosen to concentrate all the requests from each group of costumers. For the users and authorities, is destinated the user-authorities server-side, and for the municipality, the municipality server-side. The user-authorities server-side needs to be robust and more powerful than the municipality server-side, since that it has more requisitions handling and needs to perform more computation in comparison with the other one.

About the security of the application, to avoid any weakness on the interfaces with Customers Layer, there are Firewalls protecting any attacks or infiltrations from outside the Application Layer and the Database Layer. This type of protection is necessary since that the system deals with sensitive data from the costumers, so any leak of data can have rough consequences. Besides this, to maintain the recoverability of the data if there's a failure from the hard disk, there are redundant disks (defined by the RAID architecture approach) storing parity information about the data. Even that there are redundant disks, they are seemed as an only one by the Application Layer.

For instance, since that through all the RASD there wasn't specified which GPS technology would be used, the application will use the Google Maps API, because of its robustness on retrieving data of streets information and real time location, besides of the easy utilization of the tool.

As all the users and all the authorities will always check the new information about violations, the first ones will always submit new photos and the second ones will always check the ranking of vehicles with most violations, no cache structure is necessary. This decision is taken because every interaction with the system, they'll need to download new information with updated data, always updating their screen with this.

Lastly, for the traffic tickets statistics functionality, data mining techniques are used for analytics approaches for the SS team, retrieving this information from the municipality once a day, updating with the new generated tickets and inputting on the script to analyze and to output good visualization of the data. With this data, at first would be a relevant information

for the development team, but later can be totally used for public known about the effectiveness of the system.

2.2 Component view

In the Figure 3, is represented the Component diagram of the system, with all the logical and physic components showing their interactions. It's clear that only the Application Layer is explored deeper because it contains the business logic of the system, while all the other layers are represented briefly just to show how they interact with the first one.

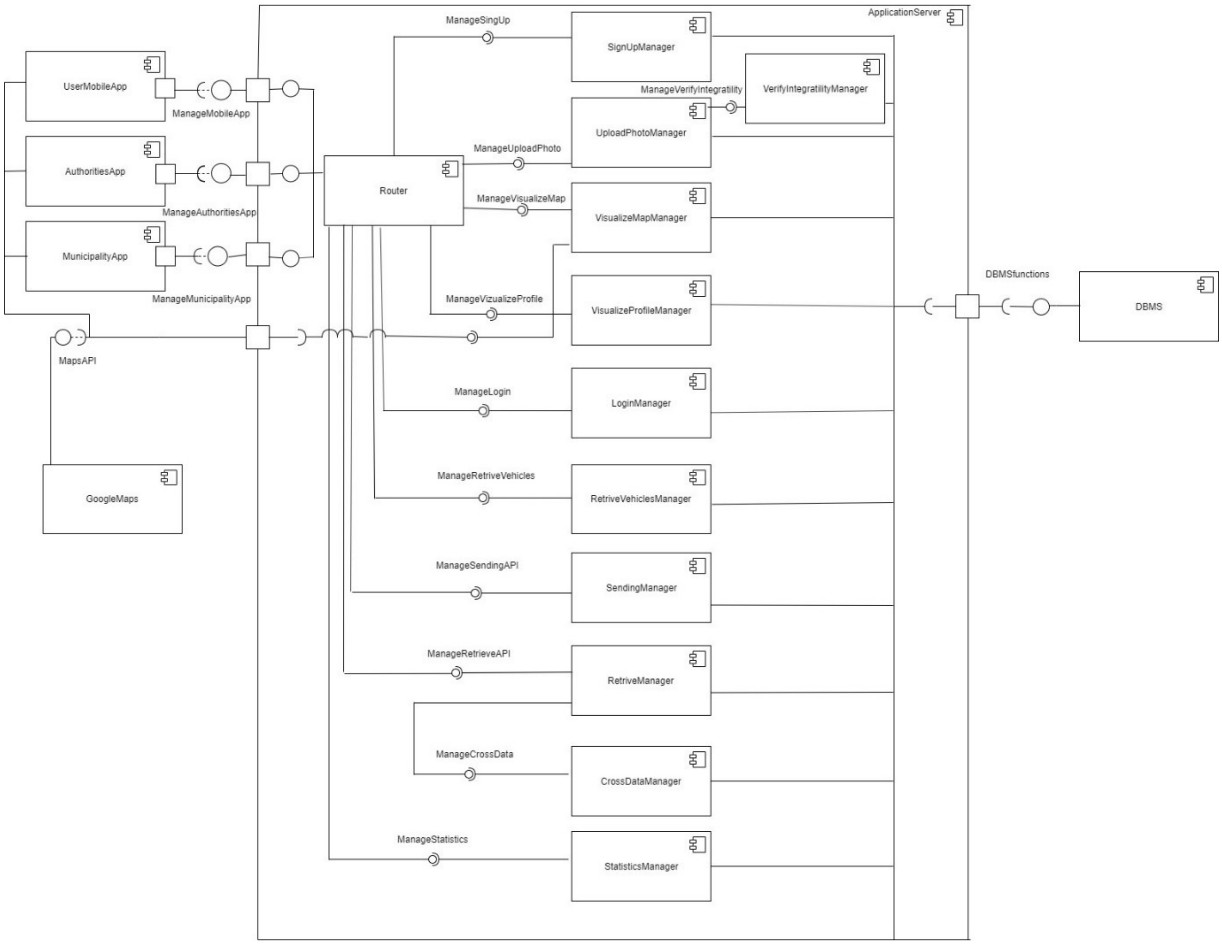

Figure 3 – *Compoment diagram of the system*

Below are the descriptions about the components' functions contained in the Application Layer, represented by the 'ApplicationServer' component.

- **Router:** manages all the "messages" and all function calls coming from other subsystems to transfer data to the correct component and, ultimately, call the correct method / function on it. The router is divided into parts according to the type of component with which it should interact, since the proposed functions are quite different. UserMobileAppRouter manages the interaction with users's mobile application: it allows them to register, log in, view their

own data and statistics, and upload photos of violations they noticed in the city. Keep track of the city map and highlight map. AuthoritieAppRouter works mainly with authorities. They have access to all the same functions as ordinary users, but in addition they can track violations of other users. Access to this organization requires special access, as security policies must be followed. MunicipalityAppRouter is designed for more advanced work directly with the municipality. Since this type of user has wider rights, new managers are provided for it. The municipality has the ability to download accident information. The police also can use user information to create penalty tickets. A more detailed view of the Router component to show the described partitioning is provided below.

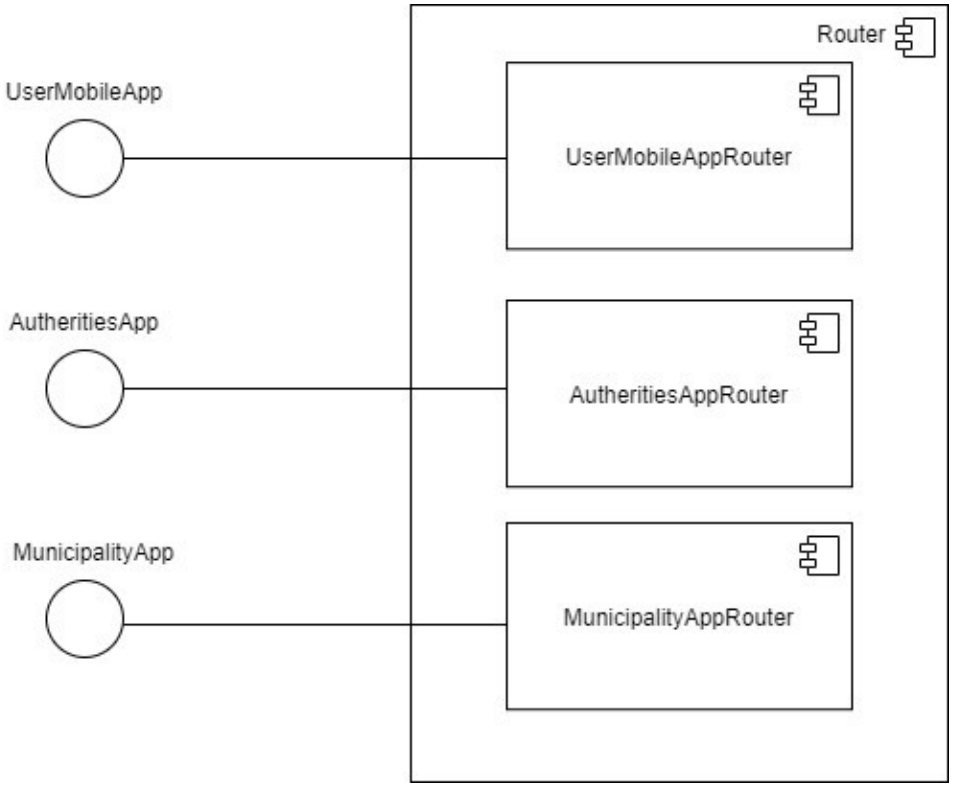

Figure 4 *– Detailed view of the Router component and its provided interfaces*

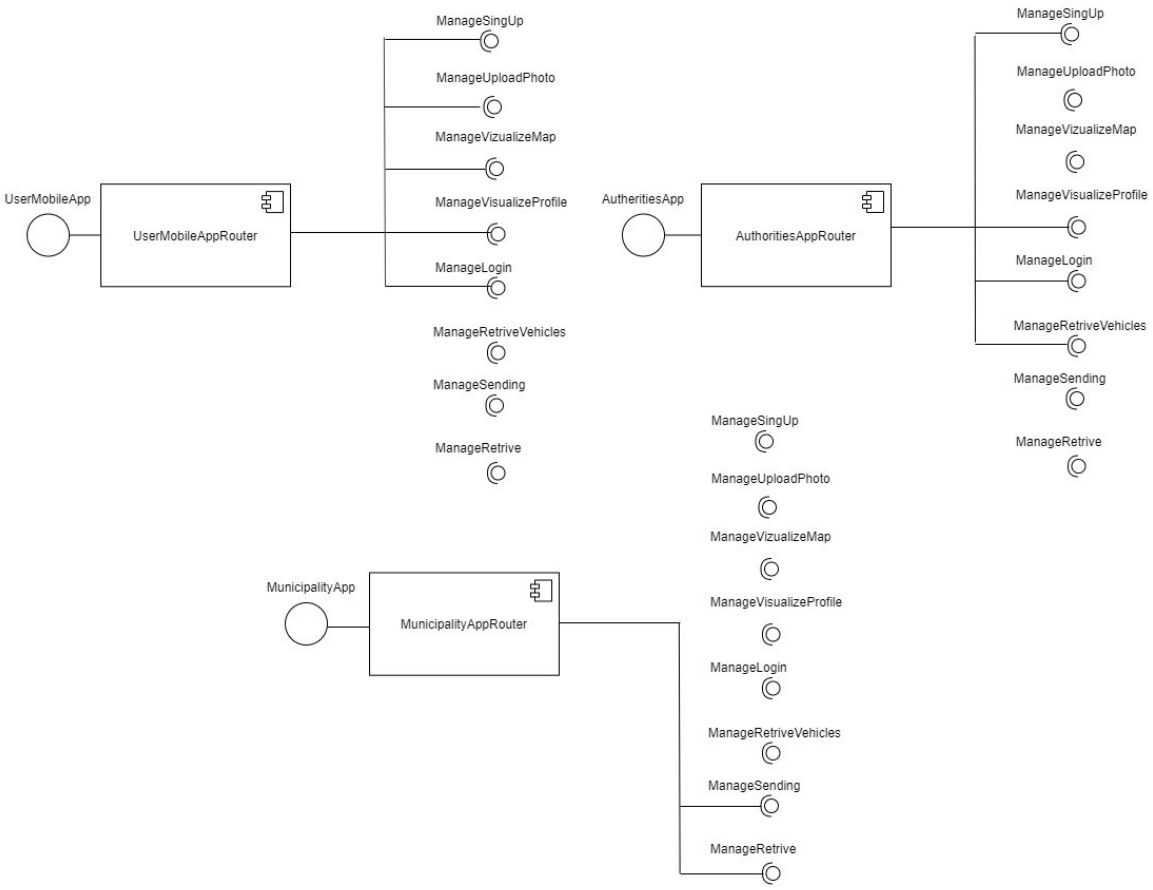

***Figure 5** – Detailed view of the interfaces exploited by the Router components*

- **SignUpManager**: After downloading the application, the user will use it. To do this each user of this application must go through the registration phase. He fills out information about himself. In the future, this information will be stored in the database. Further after the moment of registration, the user can fully use the applications.
- **UploadPhotoManager:** If the user sees a traffic violation, he can upload the photo to the SS application. This component is responsible for allowing users to upload information to the application.
- **VerifyIntegratilityManager:** The application should initially verify the credibility of the photo uploaded by the user. This component is responsible for checking the photo. It checks the date, the presence of changes in the photo using Photoshop. Only after this check the information will be uploaded to the system. This component expires directly from the UploadPhotoManager**.**
- **VizualizeMapManager:** This component is responsible for rendering the map to users. In addition to Router, it directly connects to Google maps via API. This component allows users to monitor up-to-date information on violations, streets with the greatest number of violations highlighted, etc.
- **VizualizeProfileManager**: Using this component, the user can conveniently track information in his profile. After registration and authorization, the user

is given access to his personal profile to view, add information and track events.

- **LoginManager**: This component serves as a key to getting into your personal profile for each user. If the user has already passed the moment of registration, he can get into his personal account using the login procedure and password, which is monitored by this component. The manager verifies that the data entered by the user matches the data in the database. After successful confirmation, the user can use his account for further actions.
- **RetrieveVehiclesManager**: This component is available for viewing only by authorities. It creates a list of users with the most violations. Authorities have the ability to view user information for further analysis. Ordinary users do not have access to this information.
- **CrossDataManager**: The application database is built from different sources of information. Users upload photos of violations. The municipality sends accidents. This manager is looking for patterns and intersections in those data. After this crossing, the manager is able to create interventions to be suggested to the municipality.
- **StatisticManager**: This manager compiles personal statistics of the application. This information allows the SS team to analyze the performance of the application. It also helps to explore areas of potential improvements. In general, this function is internal.
- **RetrieveManager**: This component only works with the municipality. The SS application has information from users about violations. The app also has accident information from the municipality. After analyzing those data, the RetrieveManager sends information to the municipality for the possibility of improving the structure of the city from the response of the CrossDataManager. RetrieveManager is also responsible for sending violations uploaded by users to the municipality for further work with them.
- **SendingManager**: This component only works with the municipality. It is used to load accidents and traffic tickets information into the application. The municipality uploads those data to the system through the API interface. After this, the manager is responsible to literally manage the data and store inside the DBMS.

2.3 Deployment view

Figure 6 shows the deployment diagram, whose makes clear the execution architecture of the system, representing the distribution (deployment) of software artifacts to the deployment targets (nodes).

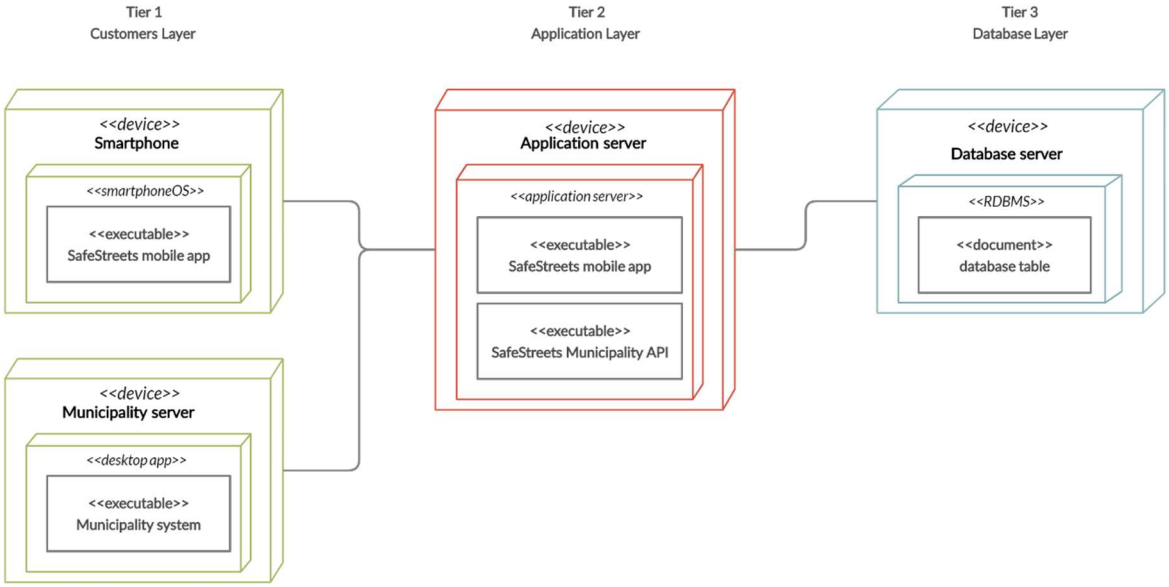

Figure 6 *– Deployment diagram*

In the diagram, the tags *<<device>>* and *<<document>>* are standard UML stereotypes that represents the type of the artifact. The first one represents an executable program file and the second one represents data to be stored (in other words, the own database). Besides, this diagram doesn't include external systems and interactions, as the Google Maps API and firewalls. In that way, it's possible to focus only on the core functionalities of the system. For instance, the "Municipality server" part is the only part that will not be implemented by the development team, since that, as already said, this is assumed as a already developed system by the municipality development team, and will only be handled the interface with it.

Analyzing the diagram, we have the three tiers architecture.

2.3.1 Tier 1

Tier 1 is related to the Customers Layer, meaning that includes all the view and interactive part of the system, as the mobile application and the API available for use. From what concerns the user and authorities, they interact with the system via the mobile application that must be developed for both Android and iOS, aiming the largest number of devices for availability. Moreover, the municipality interacts with the system via API interface, retrieving and sending data through this.

Normal users connect to the system to upload photos of violations. Authorities interacts with the system to see the list of vehicles with most violations. Both of them also can retrieve personal data and see a map with highlighted streets which have more frequent violations. Municipality interacts with the system to send information about accidents and traffic tickets, and to retrieve information about violations and suggested interventions for streets with high quantity of violations and accidents.

2.3.2 Tier 2

Tier 2 is related to the Application Layer, in which is implemented all the business logic, handles all the requests and give all the destinated answers to all the offered services. Also, it's connected to all the other tiers (1 and 3), being the mid-term between them, controlling the stored data, managing sensible and common data and dealing with all the requests made by the Customers Layer.

2.3.3 Tier 3

Lastly, the Tier 3 refers to the Database Layer, in which the data access must be deployed. Since that there are not so much queries to be done within the system, the database will be a non-relational one, since that it makes the retrieving much faster and easier to do. It also could be relational, but the non-relational approach has more advantages as already said. The disadvantages of this approach are that, for example, crossing data from violations and accidents will be very longer than in a relational approach, but, as the system will perform much more retrieving stored data than complex queries, it'll not influence the effectiveness of the functionality.

2.4 Runtime view

2.4.1 Upload a photo

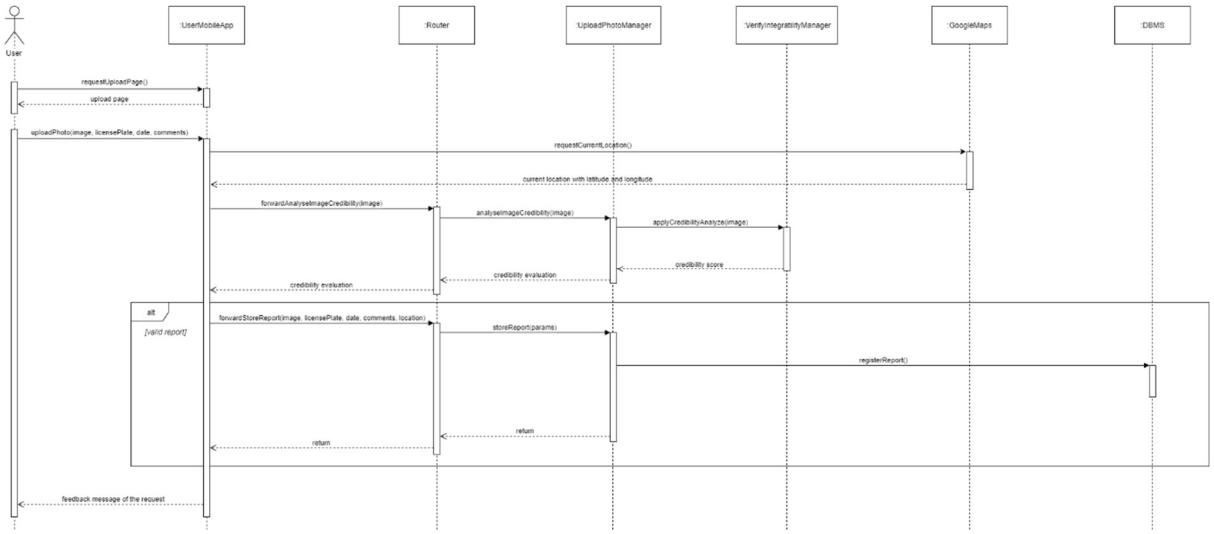

***Figure 7** – Sequence diagram for 'Upload a photo'*

For this sequence diagram, it's shown the process of uploading a photo from the user, when he is able to register a violation. Firstly, the upload photo page is opened, and then, when the user takes the photo and put all the requested information, first his location is taken, binding with the data already provided, and then the request of uploading a photo is directed to the 'Router', which will communicate the 'UploadPhotoManager' to handle the analyze of credibility of the image with 'VerifyIntegratilityManager' and later to register the information on the 'DBMS'. If the image has a credibility above 80%, the final feedback to the use will be positive. Otherwise, it'll be negative.

2.4.2 Retrieve the vehicles that commit the most violations

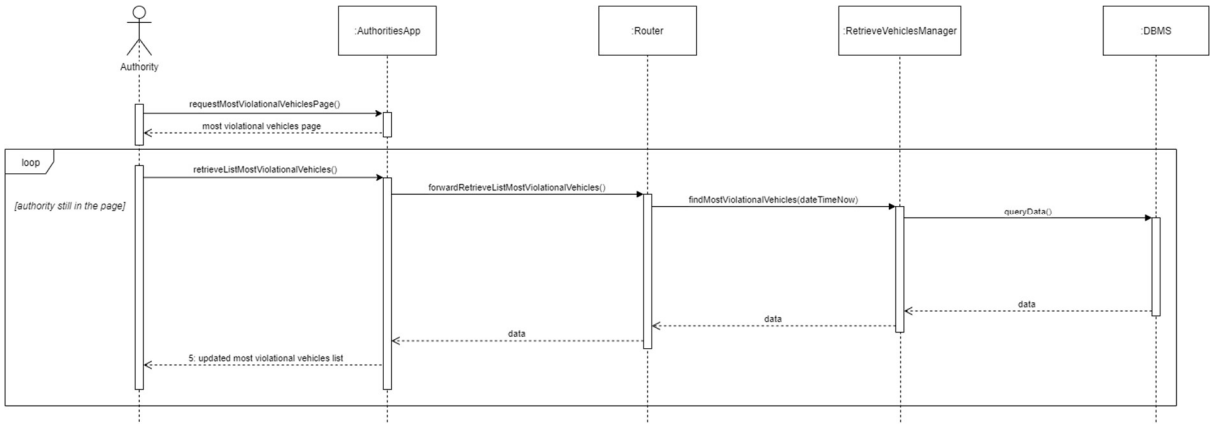

Figure 8 – *Sequence diagram for 'Retrieve the vehicles that commit the most violations'*

This sequence diagram represents the flow of the authorities when they want to see the ranking of vehicles that commit the most violations. First of all, the page is shown to the authorities. Later on, the list of vehicles will be requested for the app, and for this, 'Router' will handle this request routing it to 'RetrieveVehiclesManager', and it, lastly, will query on the DBMS to retrieve all the requested list. The returning data will reach the authority, which will be able to see the information. While the authority is still inside the page, the list will be updated with new information with a cooldown between each update.

2.4.3 Visualize map of streets with highest frequency of violations

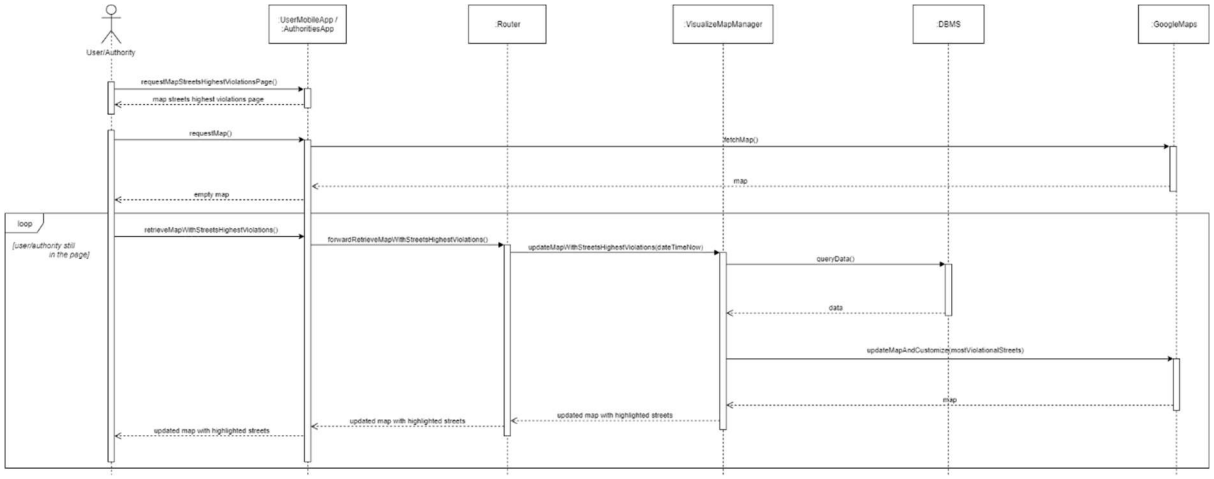

Figure 9 – *Sequence diagram for 'Visualize map of streets with highest frequency of violations'*

This sequence diagram, related to visualize the map of highlighted streets which has highest frequency of violations, is applicable for both users and authorities, and here both will be threaten as 'user'.

First of all, the user requests the map page, and it is displayed for him. Automatically, is shown an empty map for the user, without any information, only the streets nearby him. This first map should be with enough zoom and with the location of the user in the middle of it. In

a loop with a cooldown between each iteration, the map will be updated with the streets with most violations highlighted, in a way that in every loop, ‘Router’ will be triggered to send the updated information, and it will ‘ask’ for the ‘VisualizeMapManager’ to query in the DBMS and get the necessary data to request to Google Maps a custom map with the data queried, returning it to the app.

2.4.4 Send information

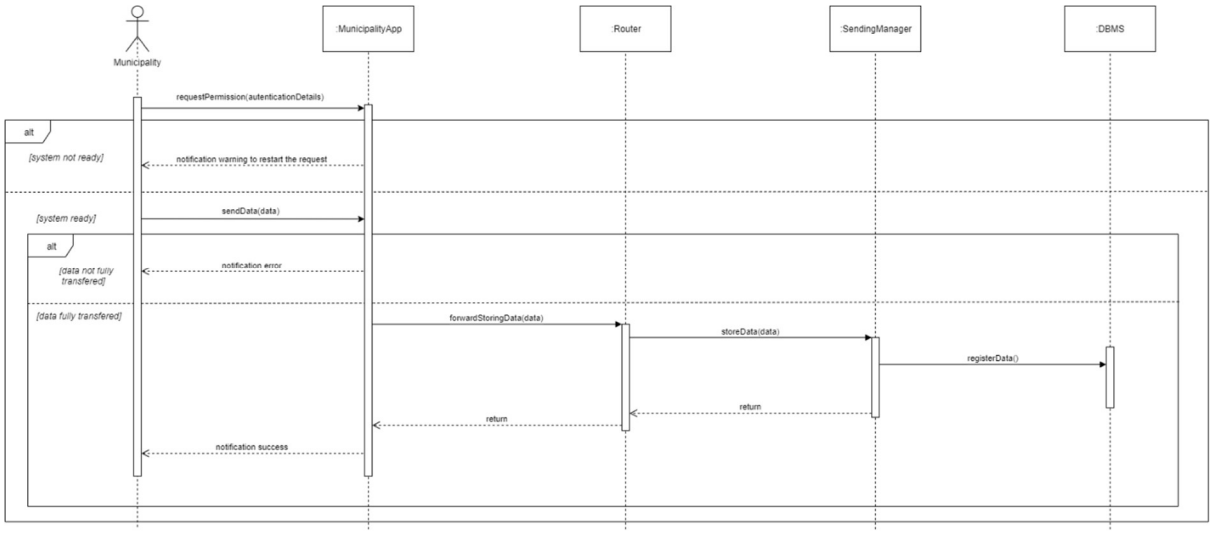

Figure 10 *– Sequence diagram for ‘Send information’*

The ‘Send information’ sequence diagram is related to the municipality, in which will handle the accidents and traffic tickets information sent to the system. So, for both situations, the flow shown by the diagram will be applied.

First of all, the municipality, through the API configured in SS system, will request a permission to start a request of send. If the system is not ready or the authentication details are wrong, only a notification warning is returned. Otherwise, the municipality is able to send the data for the system. If the data is not fully transferred to the ‘MunicipalityApp’ component, a notification error is sent to the municipality. Otherwise, the data is forwarded to the ‘Router’, and then it will route the data to the ‘SendingManager’, in which will handle the type of the data (if it is accidents or if it is traffic tickets) and store in the ‘DBMS’ in a proper way. Finally, after storing the information, a notification success is sent to the municipality finalizing the operation.

2.4.5 Retrieve information

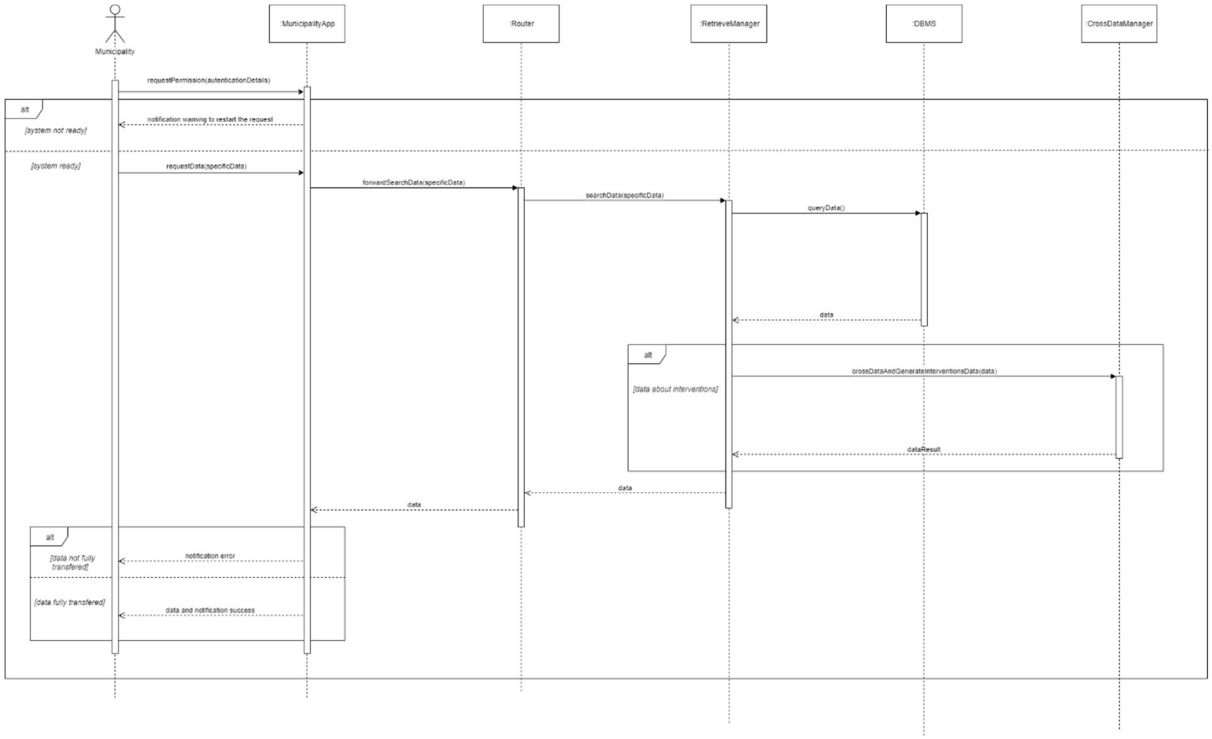

Figure 11 *– Sequence diagram for 'Retrieve information'*

The sequence diagram for 'Retrieve information' is analogous to the 'Send information' sequence diagram, in the context of it is a general representation of the municipality receiving data from the system, with regards to the suggested interventions (after crossing the data of accidents with violations) and to the violations list itself (for generating traffic tickets).

The treatment of the interface between the municipality and the system is the same from the 'Send information' diagram. After the request is sent, the app will connect to the 'Router' and it will route the request to the 'RetrieveManager', in which will query on the DBMS the requested data and will return the specific data requested. If the data request is about interventions, so the data queried from the DBMS needs to be crossed and the interventions need to be created, those being the data result. Finally, the data is sent to the municipality through the API. If the data is fully transferred, there's a notification of success. Otherwise, a notification error is sent.

2.5 Component interfaces

The Figure 12 shows the component interfaces in the application server, with source on the Component diagram represented in the Figure 3. The arrows represent a dependency usage of an interface from another one.

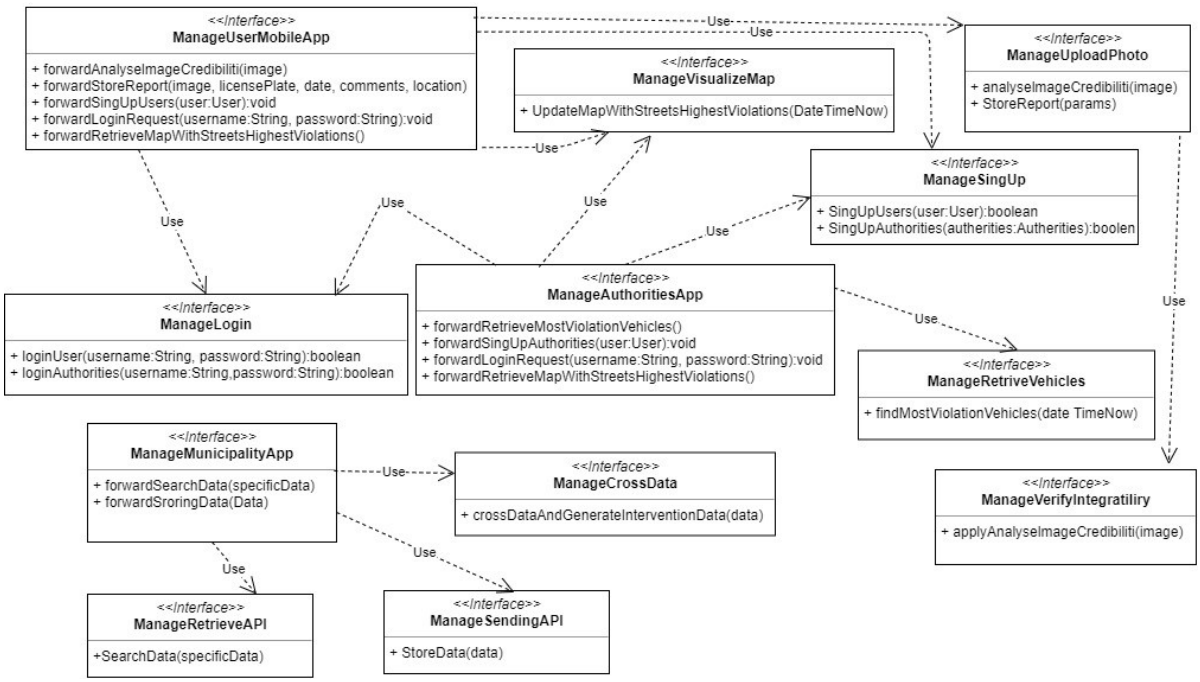

Figure 12 *– Component interfaces of the system*

Since that all the interfaces shown in the Figure 12 are about the application server, there are three main interfaces that use all the other interfaces: 'ManageUserMobileApp', 'ManageAuthoritiesApp' and 'ManageMunicipalityApp'.

- 'ManageUserMobileApp' and 'ManageAuthoritiesApp' use almost all the same interfaces, with the only difference that the first also uses 'ManageUploadPhoto' and the second uses 'ManageRetrieveVehicles'. This choice is taken due to the almost the same requirements for both costumers, with only few different features between them. These two interfaces are related to the mobile app to be developed, in which will exist all the interaction of inputting data on the system;
- 'ManageMunicipalityApp' is related to the interface of municipality and system through API. All the sending and retrieving data from and to the system is handled by this interface, besides the crossing data for the interventions creation.

In Figure 13, is shown the class diagram of the system, being the model for the application.

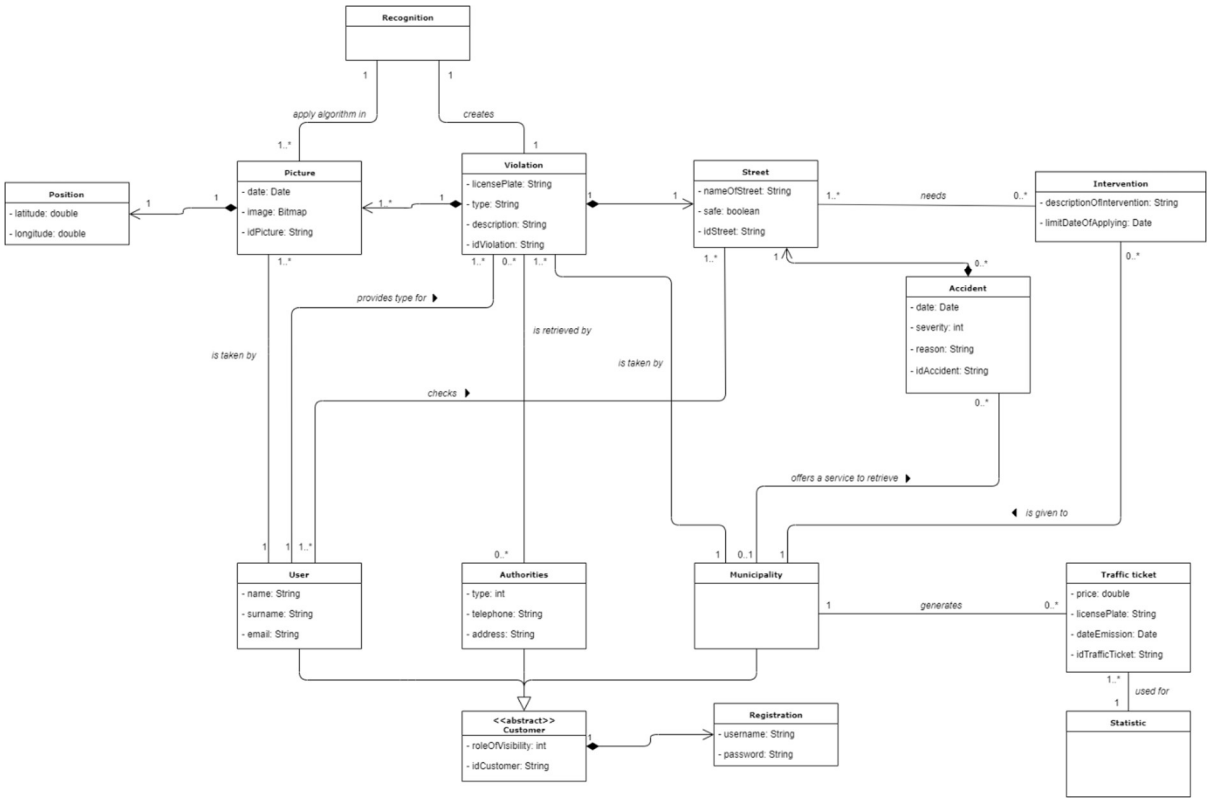

Figure 13 – *Class diagram of the system*

The class diagram is basically the same from the Conceptual diagram, with the difference that the main classes need to have an ID to be indexed on the database, since that the non-relational usage oblige the usage of this approach.

2.6 Selected architectural styles and patterns

RESTful architecture

We decided to use a RESTful architecture, because the key goals of REST include component scalability, interface commonness, independent component deployment, intermediate components that reduce latency, enhance security, and encapsulate legacy systems. An important concept in REST is the availability of resources, each of which is determined by a link with a global identifier. In order to manipulate these resources, network components (user agents and origin servers) communicate via a standardized interface (for example, HTTP) and exchange representations of these resources (actual documents for transmitting information).

The application can interact with resources, knowing two things: the resource identifier and the required action - it does not need to know if there is a cache, proxy server, gateway, firewall, tunnel, or anything else between it and the server that owns the real information. The application, however, must understand the returned data format (presentation), which will be a JSON document.

The REST architectural style describes the following five constraints on the architecture, leaving the implementation of the individual components free:

- Client-server technology – Clients are separated from the server by a single interface. This division of responsibility means, for example, that clients are not responsiblethe data store that is internal to each server, so portability of client code is improved. Servers are not responsible for the user interface or user state, so servers can be simpler and more scalable. Servers and clients can be developed and replaced independently until the interface changes.
- Stateless – Client-server interaction is further limited by the lack of saving the client context on the server between requests. Each request from any client contains all the information necessary for its servicing, and any session state is stored in the client. The server may be in state; this restriction requires server-side state to be addressable via a URL as a resource. This not only makes the servers more visible for monitoring, but also makes them more reliable in the event of a partial network failure, and also further improves their scalability.
- Cacheable – Clients can cache responses. Therefore, responses must, explicitly or implicitly, identify themselves as cacheable or not, to prevent the client from using old or inappropriate data when responding to the following requests. Well-managed caching partially or completely eliminates some client-server interactions, further enhancing scalability and performance.
- Layered system – The client cannot unambiguously determine whether it connects directly to the server or to an intermediary along the connection path. Server broker can improve system scalability by providing load balancing and providing a common cache. It may also require compliance with security policies.
- Uniform Interface – The unified interface between clients and servers simplifies and shares the architecture, allowing each part to develop independently.

Information resource management is entirely based on the data transfer protocol that will be the HTTP. For HTTP, the action on the data is set using the methods: GET (get), PUT (add, replace), POST (add, change, delete), DELETE (delete). Thus, CRUD actions (create-read-update-delete) can be performed with all 4 methods as well as with GET and POST.

We will also exploit a TLS protocol which is designed to provide three services: encryption, authentication and integrity. By this, we can manage sensible data with safety and deal easier with security breaches.

Three Tier Client-Server

We use the three-tier client-server architecture, because it has following advantages:

- Less load on the client application (Thin Client).
- Between the client program and the application server, only the minimum necessary data stream is transferred – the arguments of the called functions and the values returned from them.
- The IP application server can be run in one or several instances on one or several computers.
- Cheap traffic between the application server and the DBMS.
- Reducing the load on the data server compared to the 2.5-layer scheme, and therefore increasing the speed of the whole system.

- It is cheaper to build functionality and update software.

The scalability and the expansion of functionality in a three-level system do not cause special problems as well – installing an additional application server solves the problem of satisfying new functional requirements for the system.

Model View Controller (MVC)

MVC is a web application design pattern that includes several smaller templates. When using MVC, the application data model, user interface, and user interaction are divided into three separate components, so modifying one of them has minimal effect on the others or does not have it at all.

We are going to use the MVC concept, because we get the most obvious benefit – a clear separation of presentation logic (user interface) and application logic. The MVC concept divides data, presentation and processing of user actions into components:

- Model – provides an object model of a certain subject area, includes data and methods of working with this data, responds to requests from the controller, returning data and / or changing its state, while the model does not contain information like data can be visualized, and also does not "communicate" with the user directly. It is represented by the Database Layer.
- View – is responsible for displaying information (visualization), the same data can be represented in various ways, for example, a collection of objects using different "views" can be presented in a tabular form or in a list. It is represented by the Costumers Layer.
- Controller – provides communication between the user and the system, uses the model and view to implement the necessary response to user actions, as a rule, the data is filtered and authorized at the controller level (the user's rights to perform actions or receive information are checked). It is represented by the Application Layer.

2.7 Other design decisions

Non-relational database

As SS will not have complex queries so often on the database, but more retrieving and storing data queries, a non-relational database approach makes implementation easier to be done and data much faster to be retrieved, since that the structure already has the data ready to be retrieved in an intuitive stored way. As said before, it also could be relational, but the non-relational approach has more advantages as already said. The disadvantages of this approach are that, for example, crossing data from violations and accidents will be very longer than in a relational approach, but, as the system will perform much more retrieving stored data than complex queries, it’ll not influence the effectiveness of the functionality.

3 User interface design

The application mockups have already been exposed in the RASD document in the corresponding section 3.1.1.

Here is presented some UX diagrams to show the flow which the Customer will follow to navigate inside the application, in accordance with the mockups contained in the RASD.

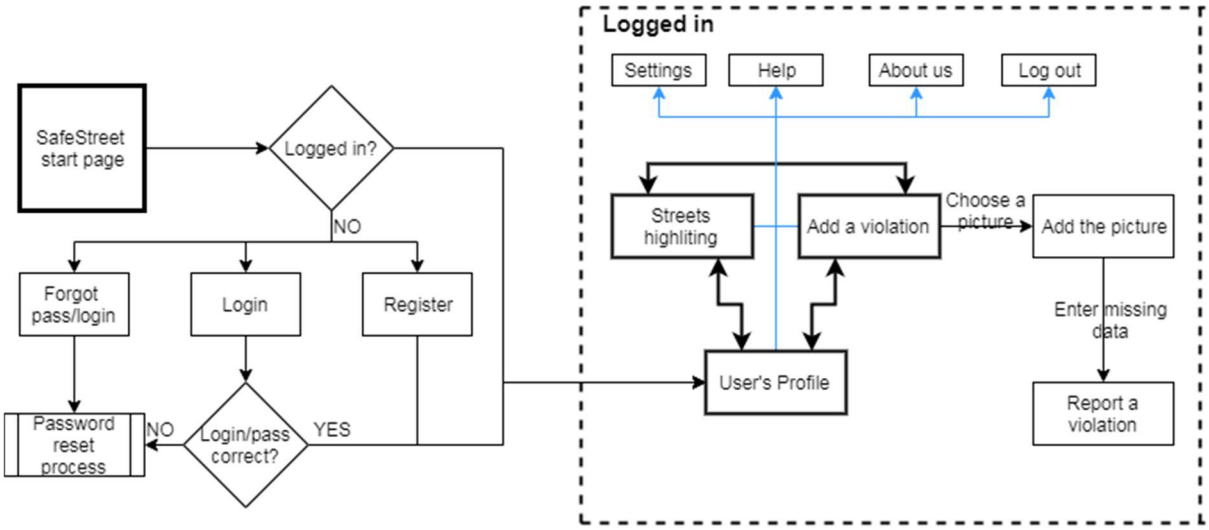

Figure 14 *– UX Diagram (User)*

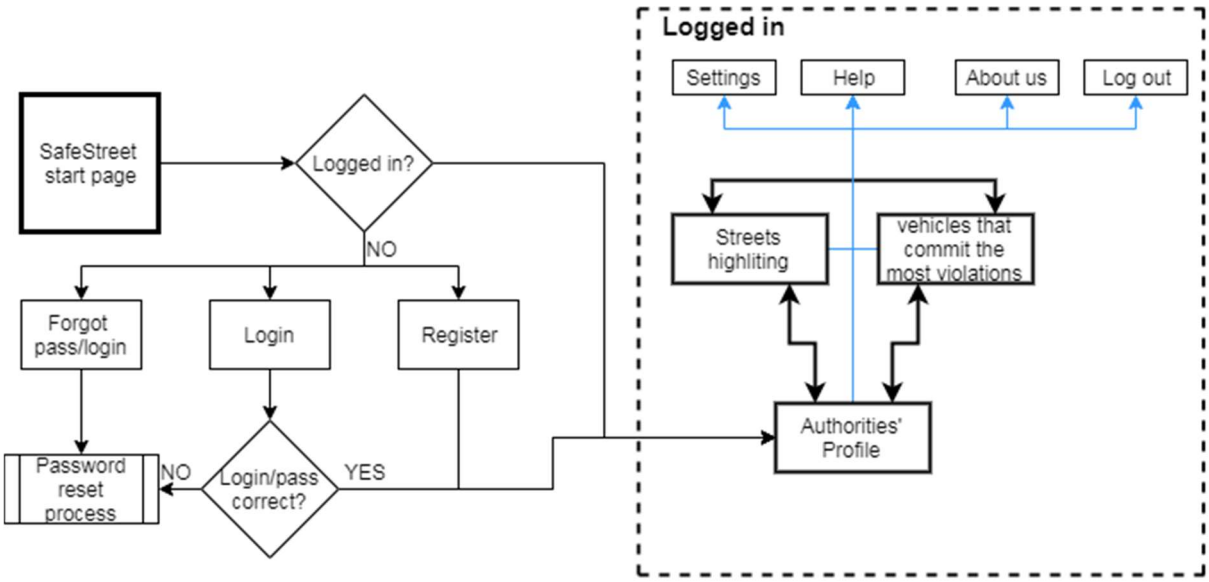

Figure 15 *– UX Diagram (Authorties)*

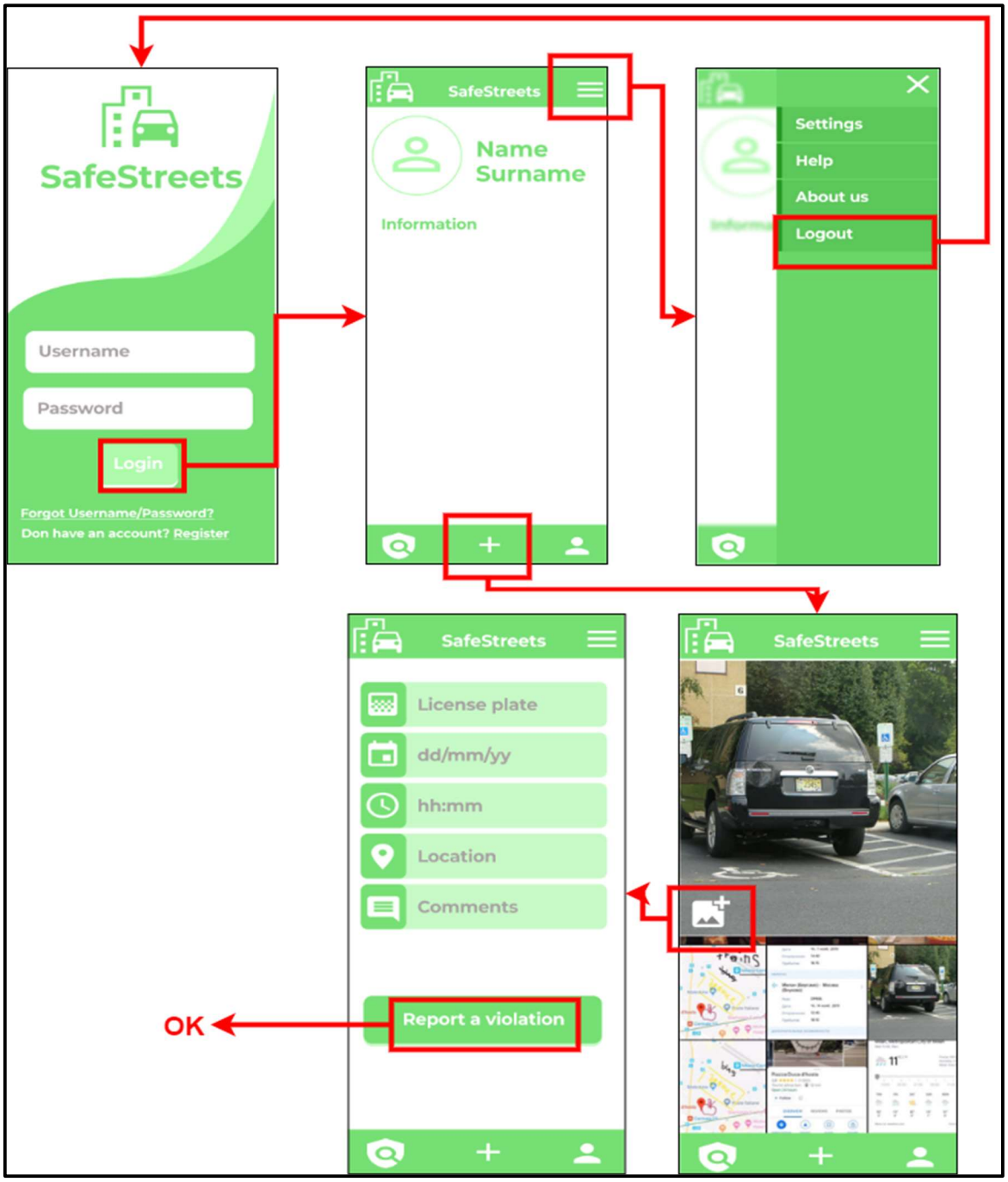

Figure 16 *– Application Flow (User)*

The figure 16 shows the application flow for user. It should be noted that we didn't show user's steps in case if he forgot password or login, etc. We decided to skip it so as not to complicate the diagram. You can see simple steps for the user to achieve the main goal – to report a violation. The user just should choose a picture with violation, enter the missing data and press the button.

4 Requirements traceability

The design of our system must comply with all requirements. The correspondence between requirements and design components in the application is presented below.

- **R1**: The system must save the collected data of users registered to SS in real time
  - **UploadPhotoManager:** this component is responsible for saving the received data in the database
- **R2**: The system must automatically check photos from manual editions
  - **VerifyIntegratilityManager:** only in case the received picture by user is authentic this component will allow upload the information to the system
- **R3**: The photo data (date, time) needs to correspondence with the time of sending
  - **VerifyIntegratilityManager:** this component allows users to upload the information to the system only after checking the picture. It checks the photo data as well
- **R4**: The authorities can see all the information about violations
  - **RetriveVehiclesManager:** this component creates a list of all the users with violations. The users with the most violations are at the top of the list.
- **R5**: Users and authorities have the right to view the map of highlights.
  - **VizualizeMapManager:** this component connects to Google maps and renders the map to show any customer of the application the map of highlights
- **R6**: The municipality sends information about current accidents to the application
  - **SendingManager:** if the service of municipality sends the information about current accidents to the SS, this manager loads the information received from the service
- **R7**: The system combines data and issues a highlight map.
  - **VizualizeMapManager:** this component allows users to monitor up-to-date information on violations, streets with the greatest number of violations highlighted
- **R8**: The system must process the accidents information provided by the municipality
  - **SendingManager:** After the municipality sends the data to the system, this component is responsible to manage the data and store inside the DBMS
- **R9**: The system must cross the information from the violations to the information from the accidents

    - **CrossDataManager**: This manager is looking for patterns and intersections in data about accidents received from municipality and data about violations in the SS
- **R10**: The system must send to the municipality the intervention generated from the crossing information
    - **RetrieveManager:** after creating the interventions to be suggested to the municipality by CrossDataManager this component sends the information to the municipality
- **R11**: The application must build statistics from the traffic tickets generated by the municipality.
    - **StatisticManager:** this manager builds the statistics of the applicatioin
- **R12**: The system must retrieve the information of traffic tickets from the municipality.
    - **SendingManager:** after the service of municipality sends the generated traffic tickets to the SS, this component loads the information

5 Implementation, integration and test plan

The system is divided in the following subsystems.

- UserMobileApp;
- AuthoritiesApp;
- ApplicationServer;
- DBMS;
- External systems: MunicipalityApp, GoogleMaps.

All the subsystems, excluding the external ones, will be implemented in a bottom-up strategy, in a way that can be applied properly in an agile method for showing time to time the result to our client. That means the software needs to be usable in each step done, with a new feature implemented for every step.

It should be noted that the components of external systems do not need to be implemented and tested only because they are external and can be considered reliable. The table below summarizes the main features available to the SS client, their importance and complexity of implementation in order to better understand the implementation, testing and integration decisions that will be made in the remainder of this section.

Feature	Importance for the customer	Difficulty of implementation
Sign up and login	Low	Low
Visualize a personal profile	Medium	Low
Upload a photo	High	Low
Calculate the credibility of the photo	High	High
Visualize map of streets with highest frequency of violations	High	High
Retrieve the vehicles that commit the most violations	High	Medium
Cross data and generate interventions	Medium	High

Build statistics from the traffic tickets generated by the municipality	Medium	Low
Send the information to the municipality through API	Medium	Low
Receive the information from the municipality through API	Medium	Low

An application should perform all its functions at launch time, and therefore it is impossible to divide features into more or less important ones. In the further part, all functions will be described in more detail.

- **SignUp and login**: register and login functions, obviously, are the initial condition for the correct functioning of the system, but they are not the main functions and they are not very complex, therefore, the SignUpManager and LoginManger components must be implemented and tested in any order (they are independent components).
- **Visualize a personal profile**: This feature allows users to conveniently use the SS application. It is not one of the most important functions, it is also not difficult to implement, but it should be convenient to use for users. It allows the customers to see the personal details of them.
- **Upload a photo**: The function allows users to upload photographs of violations. This function is one of the main features of this application. It builds the main essence of the application - to give users the opportunity to influence the structure of the city.
- **Calculate the credibility of the photo**: The function comes from the previous one, uploading photos. Since we want to be sure of the veracity of the photograph, we must verify them. Thus, it turns out that this function is hidden from users, but without it the application will not be faithful enough.
- **Visualize map of streets with highest frequency of violations**: Function is key for many users. It takes information from users and the municipality to show an up-to-date map of the incidents of the city. It also takes Google maps at the beginning and uses current maps to download information from the application.
- **Retrieve the vehicles that commit the most violations**: Authorities have the right to receive a list of information about users with the most violations. The application analyzes all violations that users of the application have sent and perform their own analyzes.
- **Cross data and generate interventions**: This function is difficult to implement, but very important. The application receives information from

users. Also receives information from the municipality. The application has the ability to analyze the received data. Also finds patterns and matches. Lastly, finds possible solutions to the problems encountered. The function allows you to find solutions to problems that may not have previously been noticed. Or the connection between violations and accidents was not clear.

- **Build statistics from the traffic tickets generated by the municipality**: The municipality have the opportunity to receive information from the application about violations to create penalty tickets. The application should show reliable information to the municipality for further processing. Including for the creation and distribution of penalty tickets to residents of the city.
- **Send the information to the municipality through API**: One of the simplest functions, but nevertheless important for the application to work with all structures. The application sends data about violations and traffic tickets to the municipality for further processing.
- **Receive the information from the municipality through API**: An additional function by which the application can receive information from the municipality about accidents that occurred on the territory of municipality. Thus, the application becomes more useful for the users which can see what exactly is happening on the streets of the city.

Finally, the Router component should be implemented and tested modularly, and then it should be integrated with other components on the application server: this component should only redirect requests and messages from clients and is not a very interesting business logic, although it is obviously fundamental component for the correct behavior of the system: that is why it is supplemented and tested only in the end (moreover, it is on top of all other components, because it "uses" everything presented interfaces).

It is important that the verification and confirmation steps begin immediately after the start of the development of the system in order to find errors as quickly as possible. As already mentioned, testing the program for finding errors should be carried out in parallel with the implementation: unit testing should be performed for individual components and, as soon as the first (partial) versions of the two components to be integrated are implemented, the integration is completed and tested: we perform the integration step by step a way to facilitate error tracking. In addition, scaffolding methods should be used if necessary.

5.1 Component integration

The diagrams below represent the integration of the application components after implementation. For instance, the components that can be integrated with other ones are represented with a connection between an arrow, in which the point shows the target component to have the integration.

First of all, will be implemented the internal components of the system with the features evaluated above, as well tested and integrated between each other. Later on, the back-end will be integrated with the front-end. Lastly, the entire application will be integrated with

the external services used by the system. Of course, during every integration, all the components need to be tested to ensure that all the functionalities are working properly.

Integration of the internal components of the Application Server

All the components will be implemented and tested individually and, later, integrated. After integration, everything needs to be tested as well.

Figure 17 – *Verify credibility of photo integration*

Figure 18 – *Cross data manager to generate interventions integration*

Integration of the front-end with the back-end

After implementing and testing the front-end and the back-end separately, they will be integrated at a once.

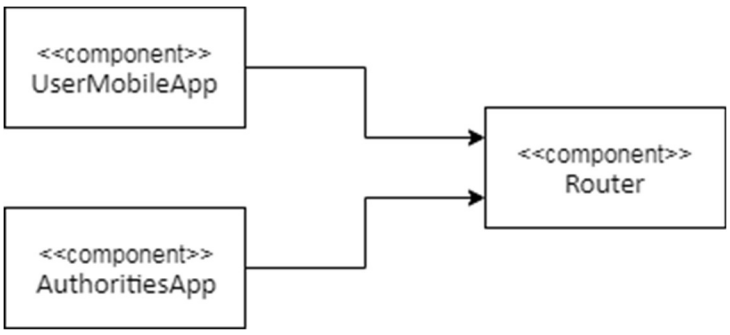

Figure 19 – *Router integration*

Integration with the external service

After implementing and testing all the integrations before, the integration with the external service can be done, and, later on, the testing with all the components together.

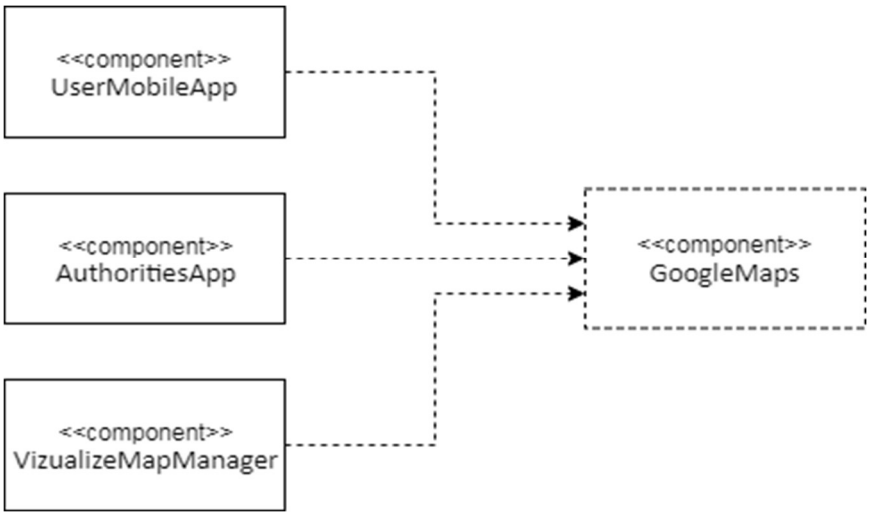

Figure 20 – *Google Maps integration*

Figure 21 – *Municipality integration*

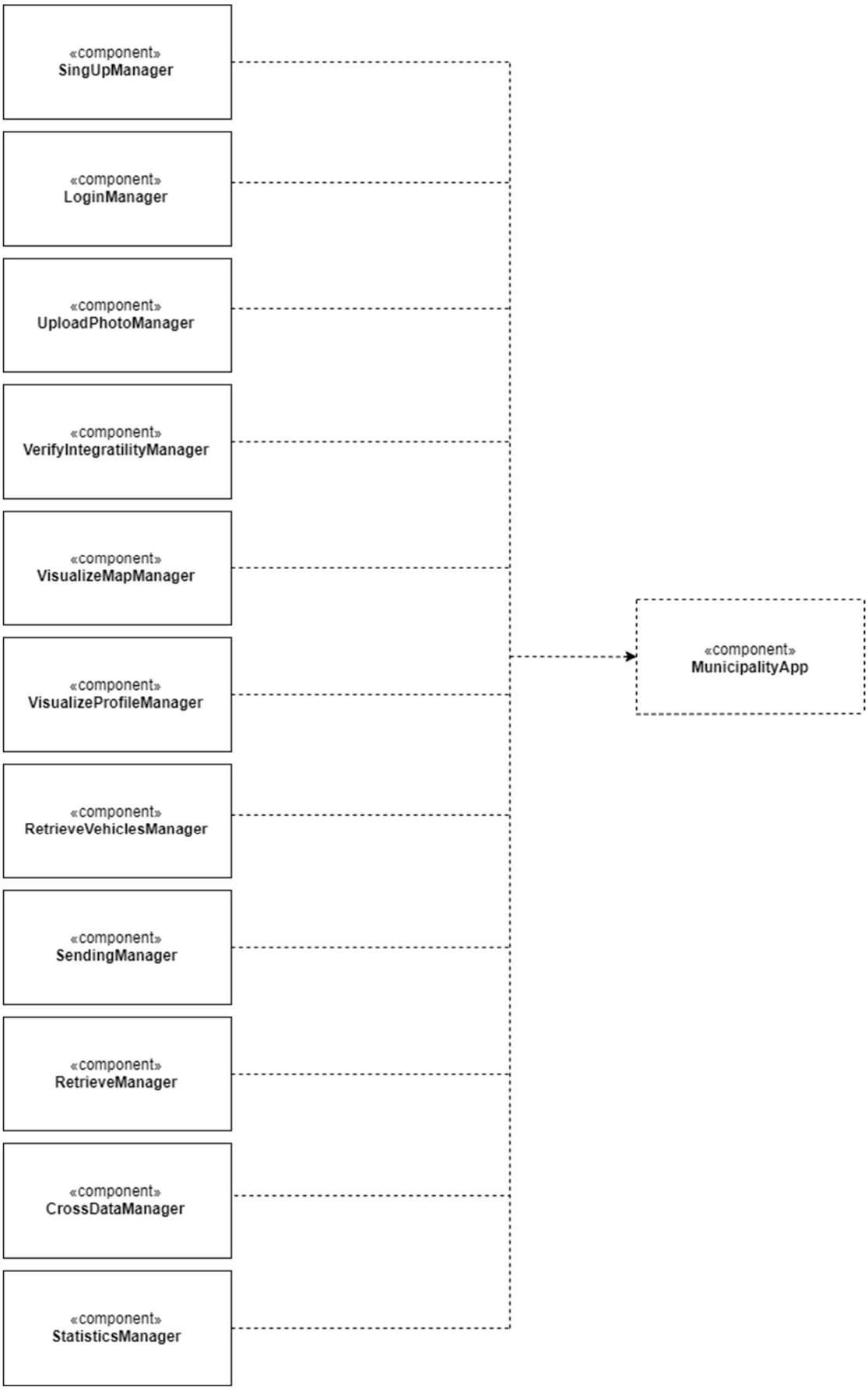

Figure 22 *– DBMS integration*

6 Appendices

6.1 Used tools

The tools used for the development of this document were those ones listed below.

- Microsoft Office Word Professional Plus 2016
- draw.io
- GitHub

6.2 Hours of effort spent

The hours spent by the group are listed below, differentiating for each participant.

Task	Hours spent		
	Aida Gasanova	Alexandre Batistella Bellas	Ekaterina Efremova
Introduction	0.5	1	0.5
Architectural design	15	20	17
User interfaces design	5	0.5	2
Requirements traceability	6	2	3
Implementation, integration and test plan	5	6	6