AA 2019-2020

POLITECNICO
MILANO 1863

Computer Science and Engineering

Software Engineering 2

SAFESTREETS

DESIGN DOCUMENT (DD)

Version 1.0 - 09/12/2019

Authors:

Simone De Vita
Fabio Fontana

Professor:

Elisabetta Di Nitto

1 INTRODUCTION

1.1 Purpose

This document aims to provide a description of the design of SafeStreets clear enough to allow the development team to start building the system with the notion of what and how it should be done. In this document developers can find narrative and graphical representations of the software design, including Component diagram, Deployment diagram and others. Designers will find useful information in Chapter 3 that, together with section 3.1.1 of the RASD, give them enough material to start building the user interface. Obviously, the aim of the document is not to fully describe the system itself; in fact, some freedom is given to those who actually will implement it but here the guidelines and some decisions are pointed out.

1.2 Scope

We briefly recall what is the scope of the system, which is fully described in section 1.2 of the RASD:
SafeStreets arises from the need of reporting to municipalities traffic violations occurring in their territory, in order to reduce them and study their occurrence. We distinguish two kinds of users: the Municipality users, who have access to all of the segnalations and use them in different ways; moreover they receive solutions, computed by SafeStreets, whose aim is lowering the number of parking violations in some problematic zones. The “Normal” users who are citizens willing to help the Municipality by sending segnalations via a Mobile App. Both the users can learn more about the occurrence and the distribution of violations in their cities through specific maps and statistics. A non-registered municipality is notified by email when a new segnalation is received by SafeStreets while the registered ones can also share some information regarding past solutions and parking violations that help SafeStreets in computing statistics and find better solutions.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- System: This word is used to refer to SafeStreets and its components such as the server that allows communications with users, the database and all the others that allow data to be computed and solutions to be offered to its users.
- User: who benefits from the service offered by SafeStreets by sending photos of the violations or accessing the data saved in the system.

- Municipality: public authority who take advantage of the service offered by SafeStreets and eventually collaborate with it providing supplementary information and receiving possible solutions to reduce the number of violations.
- Segnalation: report sent by a user. It is composed by some elements like a photo in which the violation figures, the type of violation, date, time and the street where the violation occurred.
- Violation: corresponds to parking in an illegal way, there are many types of different parking violations: parking where it's not permitted (on a crosswalk, near intersections, on sidewalks, on bus stops, on a handicapped zone without permissions, parking in time slots where it is not allowed to stop on that specific street, double parking,).
- Solution: report sent by SafeStreets to the municipality after an accurate evaluation of violations occurred in a certain area. Its aim is to reduce the number of these types of events.
- Zone: the basic unit in which it is possible to divide a city.
- Statistic data: representations of the data that SafeSteets stores. Users can visualize them.
- Solution table: It corresponds to a map that assigns to each possible parking violation a solution to propose. It is regularly updated because the system evaluates the impact that certain solutions have had in specific zones of the city and whether these have actually reduced the number of violations.
- Component: a piece of a system whose implementation is hidden. It may require interfaces or offer them to other components. Their interfaces define its external behavior.
- Firewall: it is a component that controls the incoming and outgoing traffic in order to protect the system.

1.3.2 Acronyms

- RASD: Requirement Analysis and Specification Document
- DD: Design Document
- API: Application Programming Interface
- UX: User eXperience
- DB: DataBase
- GPS: Global Positioning System
- OCR: Optical Character Recognition
- HTTP: HyperText Transfer Protocol
- RMI: Remote Method Invocation
- JDBC: Java Database Connectivity
- JSON: JavaScript Object Notation
- REST: REpresentational State Transfer
- DBMS: DataBase Management System

1.3.3 Abbreviations

- Rn: n^{th} requirement

1.4 Revision history

- V1.0: the base document that eventually will be extended or modified in the future.

1.5 Reference Documents

- Specification document: “SafeStreets Mandatory Project Assignment”
- SafeStreets Requirement Analysis and Specification Document

1.6 Document Structure

The document is divided in six chapters:
Chapter 1 introduces the document giving the reader a general idea of what is the content of it and presents the scope.
Chapter 2 is the core part of the document. It provides the description of the system architecture, specifying the design of the system. There many diagrams are used to underline and show different aspects. This section also contains the justification of the main design decisions.
Chapter 3 completes the chapter 3 of the RASD with more mockups and provides the UX diagrams in order to give the reader an idea of how the user interfaces will be organised and how it can be explored.
Chapter 4 illustrates how the requirements specified in the RASD are fulfilled by the architectural choices and components defined in chapter 2 by providing the requirements traceability.
Chapter 5 suggest the implementation, testing and integration plan, giving an order that must be respected.
Chapter 6 shows schematically the effort spent for the realization of this document.

2 ARCHITECTURAL DESIGN

2.1 Overview

The system is composed by a three tier architecture that can be seen in the diagram below which in a very general way represents each of the three levels: the presentation level, the application level and finally the data level.

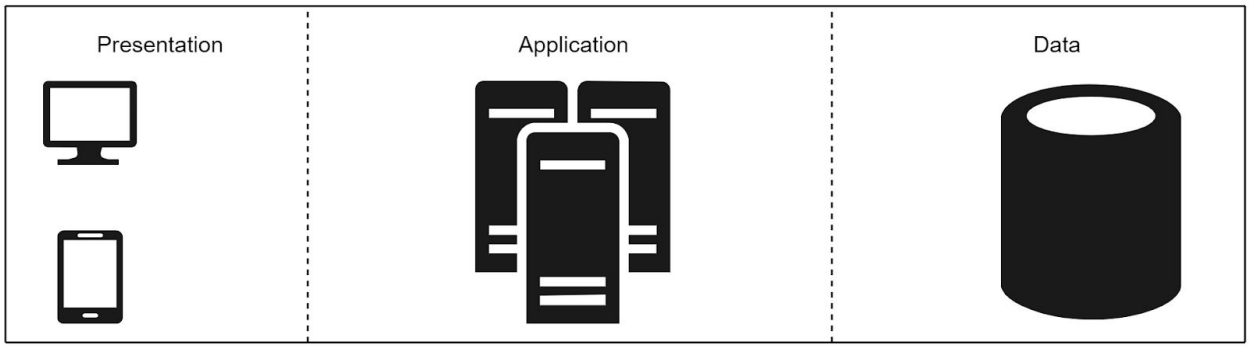

Figure 1 - 3-tier architecture

The first is dedicated to the user interface and it is the one which the user interacts with, in the second can be found the business logic and therefore it takes care of implementing all the functions of SafeStreets but to do that it is necessary the data tier that precisely keeps all the information that the system regularly uses and wants to store.
In the following diagram it is possible to better understand which parts compose each level: in fact the reader can see that the user interacts with the system through the application installed on his mobile device or the browser of a desktop system. These are the hardware devices on which the presentation level acts. The only difference is that their way to the application level is different: requests sent from web browsers are managed by a cluster of web servers that try to answer in a short time as it is expected that most requests from users connected via desktop are intended to consult statistics since through this interface it is not possible to send new reports for normal users. If what is required is already ready, the web server can easily answer but if the requested data need to be calculated or they are saved in the database, it will be necessary to contact one of the application servers. In fact, this component permits to access the business logic and communicates with the database; in this way it is able to fulfill any requests coming from mobile applications or web servers. Finally the DB server receives the requests coming from the application servers and directly accesses the databases of which it knows the organization: the segnalation DB contains all the segnalations (both those sent by normal users and municipality users) while the Private DB contains data that typically only the system uses (solution table, user data) or which municipality users can access (except for

segnalations that are stored in the segnalations DB). Before the web and application servers there are load balancers which distribute application traffic across a number of servers. Before accessing the application and web server there is also a firewall that allows the system to raise the security level, which is essential since there are data that must be carefully stored and protected.

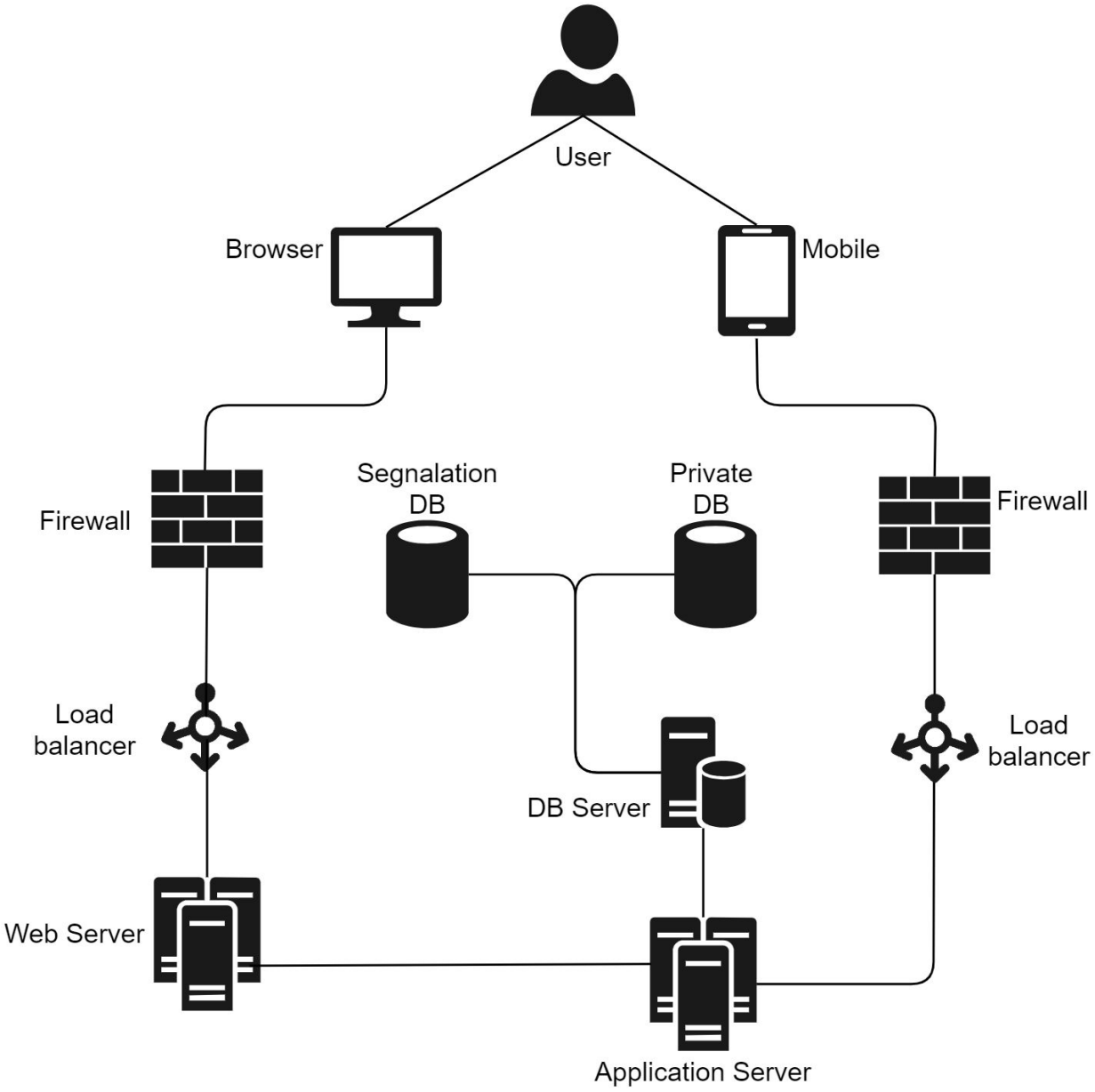

Figure 2 - Overview architecture

2.2 Component view

The following diagram shows the components in which the system is structured and their interactions. Only the Application layer is analysed in detail because it is the core of the system since it implements the business logic. The Presentation layer and the Data layer are represented through their (software) components in such a way that the 3-tier architecture can be recognized also here.

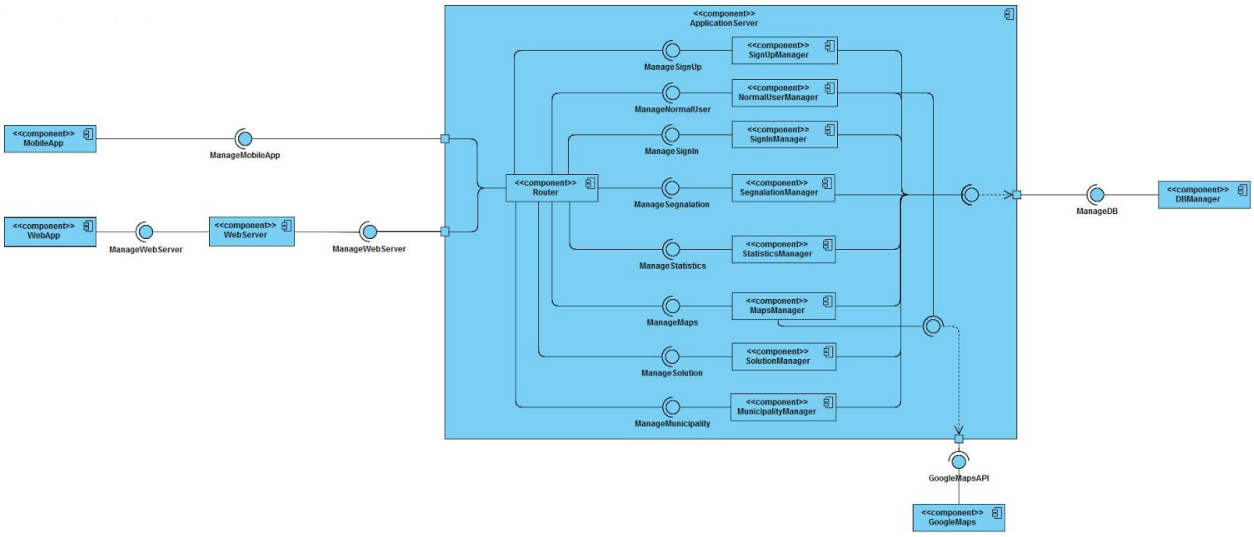

Figure 3 - Component diagram

A brief description of the Application Server's components is given below:

- **Router**: it is in charge of accepting all the requests coming from the Presentation layer and redirecting them to the correct component that is capable of handling it. This component is partitioned into two subcomponents, the **MobileAppRouter** and the **WebAppRouter**, since the two platforms slightly differ on which services they can offer. Infact, while both platforms offer the possibility to visualize statistics, segnalations, maps, signup and signin, only via the MobileApp a normal user can send a segnalation, and a Municipality user can send its own data only via the WebApp.
  Once a user has logged in, the Router knows which type of user it is together with which device he is using, and can offer the available services through its interfaces.

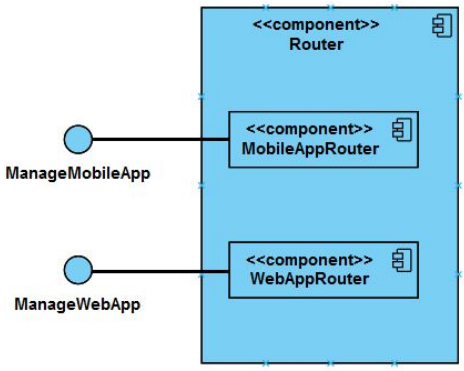

Figure 4 - Router partitioning

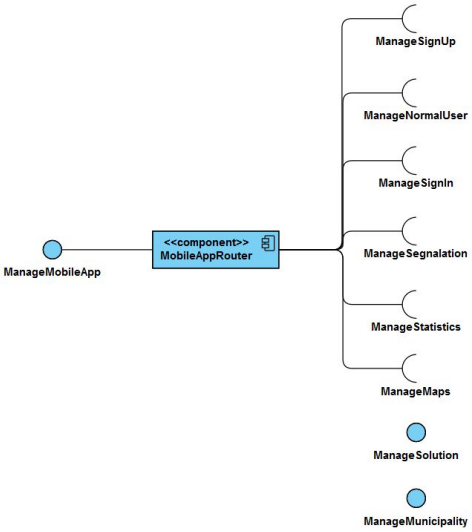

Figure 5 - MobileAppRouter

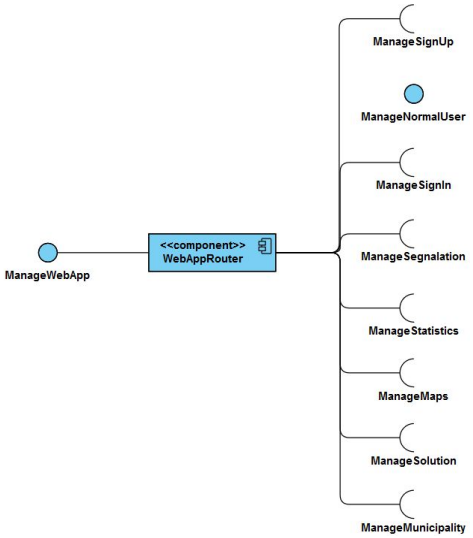

Figure 6 - WebAppRouter

- **SignUpManager**: it lets users sign up, specifying which type of user is registering to the system. It exploits the DBManager in order to save the user's data in the PrivateDB.
- **NormalUserManager**: this component covers the crucial aspect of SafeStreets, it takes care of creating and storing the segnalations sent by a normal user via the MobileApp. It's important to notice that the OCR software runs here in the Application Layer, this is mainly due to security reasons. When the user sends the segnalation with the photo, the algorithm will try to recognize the licence plate. If it succeeds, the segnalation is stored in the Segnalation DB, otherwise an error message is delivered to the user. Once the segnalation is inserted into the DB, the component checks whether the specific municipality is registered in the system and if it is not then it sends an email to the right address.
- **SignInManager**: it concerns the authentication of users, involving the access to the Private DB in order to check the authentication credentials.
- **SegnalationManager**: this component retrieves the requested segnalations from the segnalation DB for each user. In the case of a Normal User it sends back all

the segnalations sent by that particular user. In the case of a Municipality User it looks for all segnalations concerning the zones under the control of that specific Municipality and allows them to modify the state of each segnalation.

- **StatisticsManager**: this component deals with the data analysis of the segnalations sent by user together with the Municipality data. It is in charge of computing all the statistics that are shown to all the users when they access the corresponding section. It interacts with the DB since it retrieves the date it uses as input from there.
- **MapsManager**: it offers to the user the requested maps. The map of a specific zone is given by GoogleMaps API and it is fulfilled with a StatisticData of a particular type stored in the DB that is updated daily and represents the occurrence of violations in a specific zone till that day. This component is responsible for updating the StatisticData of each zone adding to it data coming from segnalations received after the last update or segnalations that changed their state.
- **SolutionManager**: Municipality can receive solutions thanks to this component. When a certain number of violations in a given zone exceeds a threshold, it checks the solution table and sends the most appropriate solution to the Municipality. So this is the component that triggers the process that leads to the creation of a solution that is finally saved in the private database. It is also used by municipality users to retrieve from the database all the solutions for their municipality and eventually change the state of a solution if the municipality chooses to adopt the suggested solution. In this case this component checks if the number of segnalation for that specific zone decreases and eventually updates the solution table in order to improve the quality of proposed solutions.
- **MunicipalityManager**: here is implemented the tool that allow Municipality to send its own data to SafeStreets. It accepts in input a JSON file divided into two sections: the first describes the violations that the municipality wants to share with SafeStreets, while the second reports old solutions adopted in the past to face a high occurrence of violations in a specific area. It is accessible only via the WebApp.

The following Class diagram wants to give the reader a general idea of what the DB will store and how data are related. This diagram doesn't want to be prescriptive description of the structure of the database because who will actually create it must be free of choosing some aspects but the important thing is that these kind of data can be effectively stored.

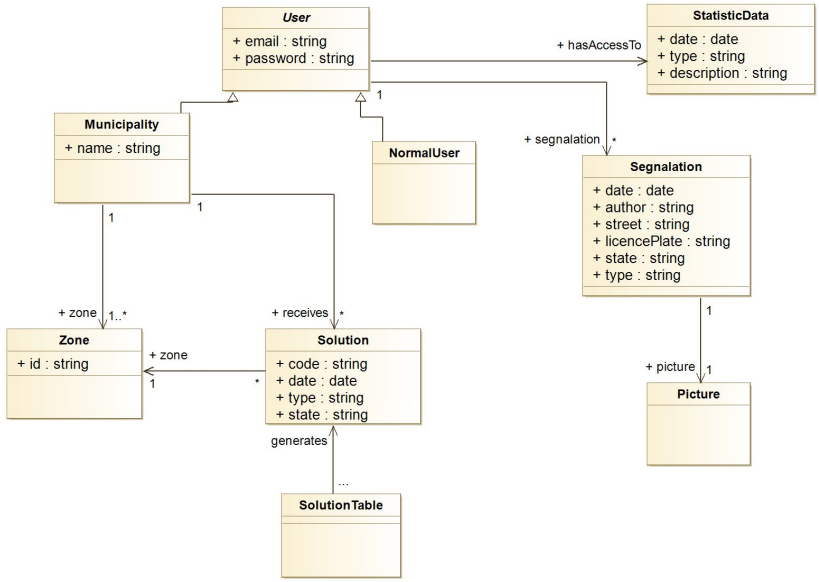

Figure 7 - Class diagram

2.3 Deployment view

The following image represents the Deployment diagram of the system. The distribution of artifacts across the devices is shown; load balancers and firewalls are not represented since here the aim is to highlight just the relevant parts.

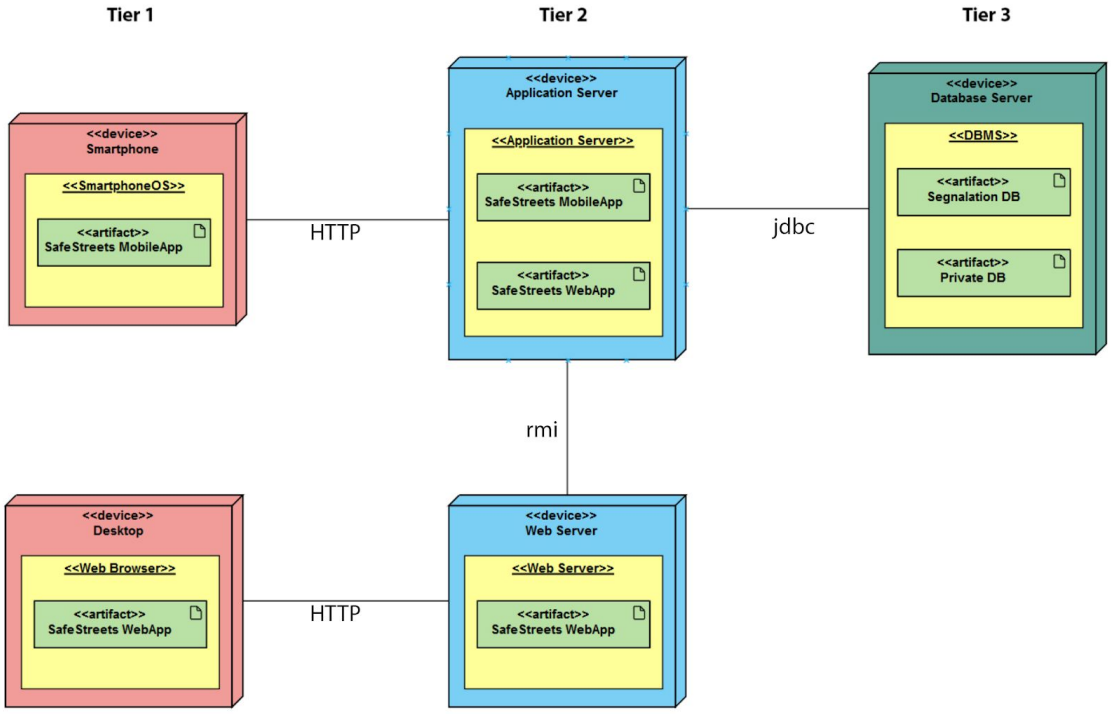

Figure 8 - Deployment diagram

1. **Tier 1**: it is the Presentation layer. User interact with SafeStreets either via a MobileApp, for Android and iOS, or WebApp that can be accessed by any browser.
   The MobileApp is conceived especially for normal users to let them send new segnalations and view the sent ones as well as visualizing statistics but it can be used also by municipality users.
   The WebApp is mostly conceived for municipality users since here they can read solutions provided by SafeStreets and send municipality data to SafeStreets. It is of course possible to visualize statistics and maps. Both the mobile app and the WebApp interact with the second tier via HTTP protocol.

2. **Tier 2**: the business logic is implemented in this tier.
   The Application Server is responsible for managing all operations. It receives requests from the Presentation layer and trough its main component (the Router) it redirects them to the appropriate component. It directly interacts with the third tier and the MobileApp.
   The Web Server is an intermediate node between the WebApp and the Application Layer. It is in charge of receiving requests coming from Desktop users and forwarding their them to the Application Server. It also acts as a caching system in the case that the requested content is already available. The communication with the application server exploits the rmi system.

3. **Tier 3**: in the third level the DB is accessed thanks to jdbc which is used for this purpose. The DBMS that has been chosen is SQL Server and there are two different databases:
   a. **Segnalation DB**: this DB stores data about segnalations and violations, both the ones provided by users and those provided by the Municipality. This DB offers the raw data that can be mined and that is used to build up statistics.
   b. **Private DB**: this DB stores user's data like email, password etc. together with the solution table, solutions offered to the municipality and those applied by municipality in the past and reported to SafeSteets.

2.4 Runtime View

Below the sequence diagrams for the most relevant use cases are shown and commented.
It's important to notice that since the user's sign in is a prerequisite to all of the use cases, it's omitted from these and it's shown only once in the dedicated use case (2.4.3) so we can assume that for all other use cases the user is already logged in.

2.4.1 Send new segnalation (by Normal User)

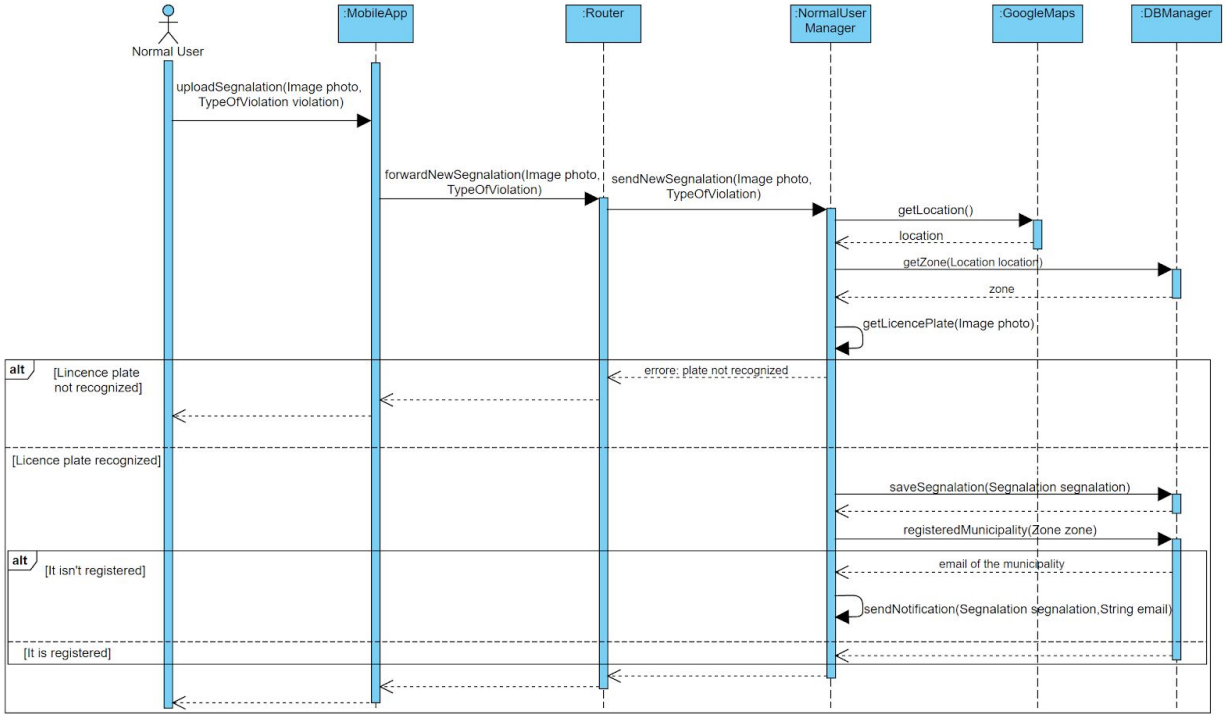

Figure 9 - Sequence Diagram of "Send new segnalation" use case

The first use case we examine is the most important one: it shows how a segnalation whose author is a normal user is generated and stored. Once the user takes the picture and specify the type of violation, the MobileApp contacts the Router component of the Application Server. The latter one, considered the type of the request and the users who sent it, forwards the message to the NormalUserManager component which integrate the segnalation with the location where the photo was taken exploiting the GoogleMaps API. Then it runs the OCR algorithm on the photo to get the licence plate, if this fails a message is provided to the user who is asked to take the photo again. Otherwise the NormalUserManager contacts the DBManager to save the segnalation. It also asks the DBManager for the registration status of the Municipality who has control on the zone where the location belongs. If it isn't registered, an email is sent to notify the violation.

2.4.2 Send Municipality's data

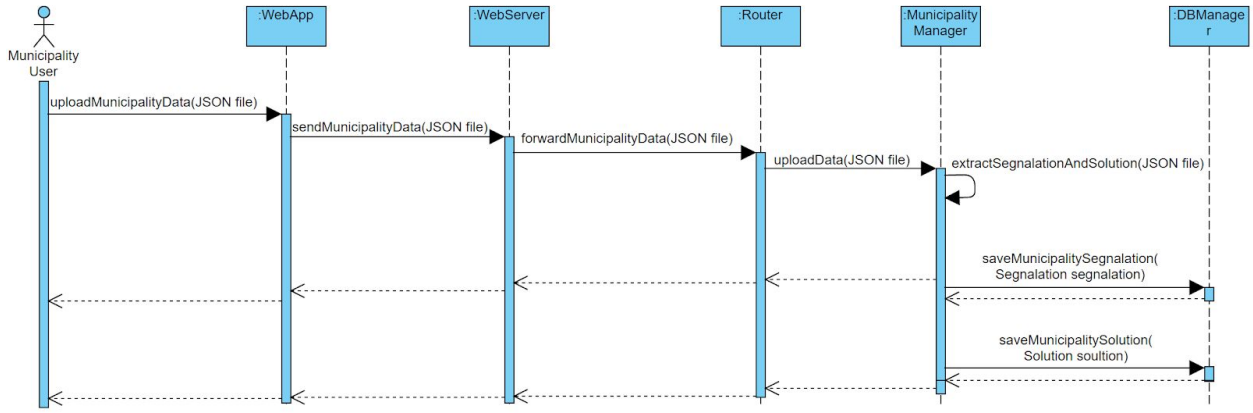

Figure 10 - Sequence Diagram of "Send Municipality's data" use case

In the previous figure it's shown how the Municipality interacts with the system when it wants to cooperate with it by providing its own violation data together with the solutions adopted to face the violations. As mentioned before, the Municipality can achieve this task only by means of the WebApp. It provides a JSON file that encodes the data about violations and solutions, which is forwarded to the WebServer. The latter one sends the request to the ApplicationServer, where the Router recognizes the request and passes the data to the MunicipalityManager. Here the JSON file is parsed and the two types of data get encapsulated into the classes Segnalation and Solution, which the DBManager is able to work with.

2.4.3 User Sign In

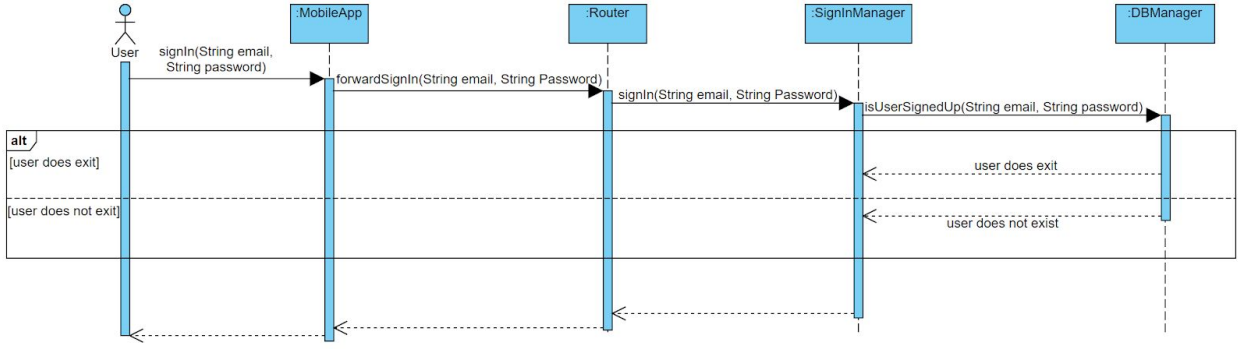

Figure 11 - Sequence Diagram of "User Sign In" use case

This sequence diagram shows the activity that a user does in order to use the services offered by SafeStreets. It's shown the Sign In process only via the MobileApp, which is similar to that via the WebApp with the only difference that the WebServer forwards the request to the ApplicationServer. The user provides its Email address and its password. Once the request reaches the Router component, the SignInManager is called in which queries the DBManager

for an Email/Password match. If it does exist, the user is correctly signed in, otherwise the user is asked to provide credentials again.

2.4.4 Visualize maps

In the next diagram the reader can understand how components interact in order to show to the user the map of a zone requested by him. The user inserts as input the zone he is interested in, the request arrives directly to the application server if he is using the mobile app or through the web server in the other case. In the application server the router contacts the MapsManager which first retrieves the map of the requested zone through an HTTP request to Google Maps, then it retrieves the Statistic data that represents the current state of that particular zone and applies it to the map in order to highlight the areas based on the occurrence of violations as described in the RASD. Finally, the result is returned to the user.

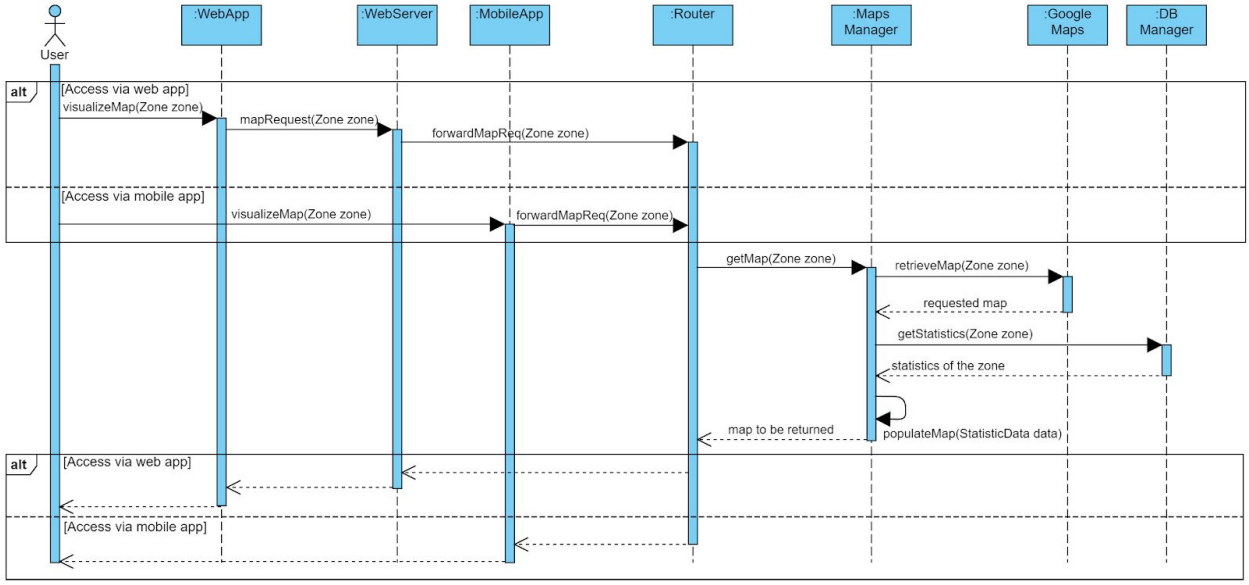

Figure 12 - Sequence Diagram of "Visualize maps" use case

2.4.5 Find a solution and access to it

The following diagram shows the components which participate in the creation and identification of a solution. The SolutionsManager retrieve from the database all the segnalations of a zone and checks if the situation requires the computation of a solution to be presented to the municipality and if the occurrence of violations of the same type exceeds a specific threshold then the solution table is accessed, the corresponding solution is read and inserted in the final solution that will be offered together with the type of violation and the zone. The process that has been just described must be repeated for each zone of every municipality once a week since this is a very heavy operation and for this reason it is not possible to repeat it daily.

In the last diagram it is represented how the access to solutions by a municipality user is managed by the system and its components.

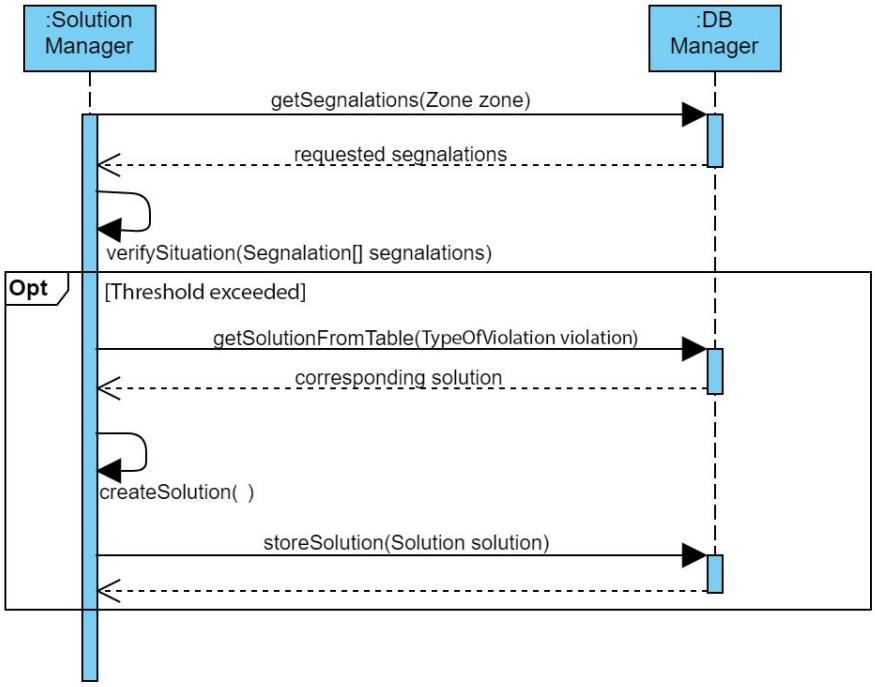

Figure 13 - Sequence Diagram of of the computation of a solution

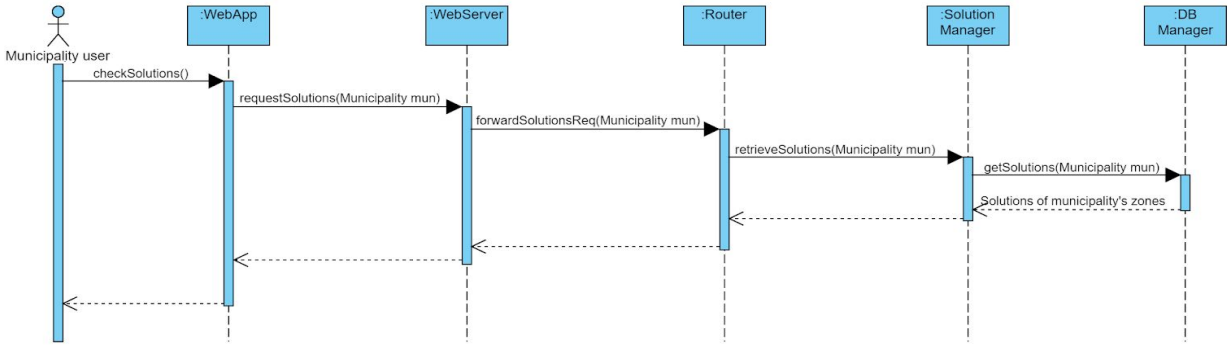

Figure 14 - Sequence Diagram of the access to solutions by municipality users

2.5 Component interfaces

In the following diagram the component interfaces and their dependencies are shown in detail. Obviously they are the same that can be found in the component diagram but here we have a better view over them.
The reader should consider that the methods reported in the diagram are not all the methods that can be invoked but they are the main ones.

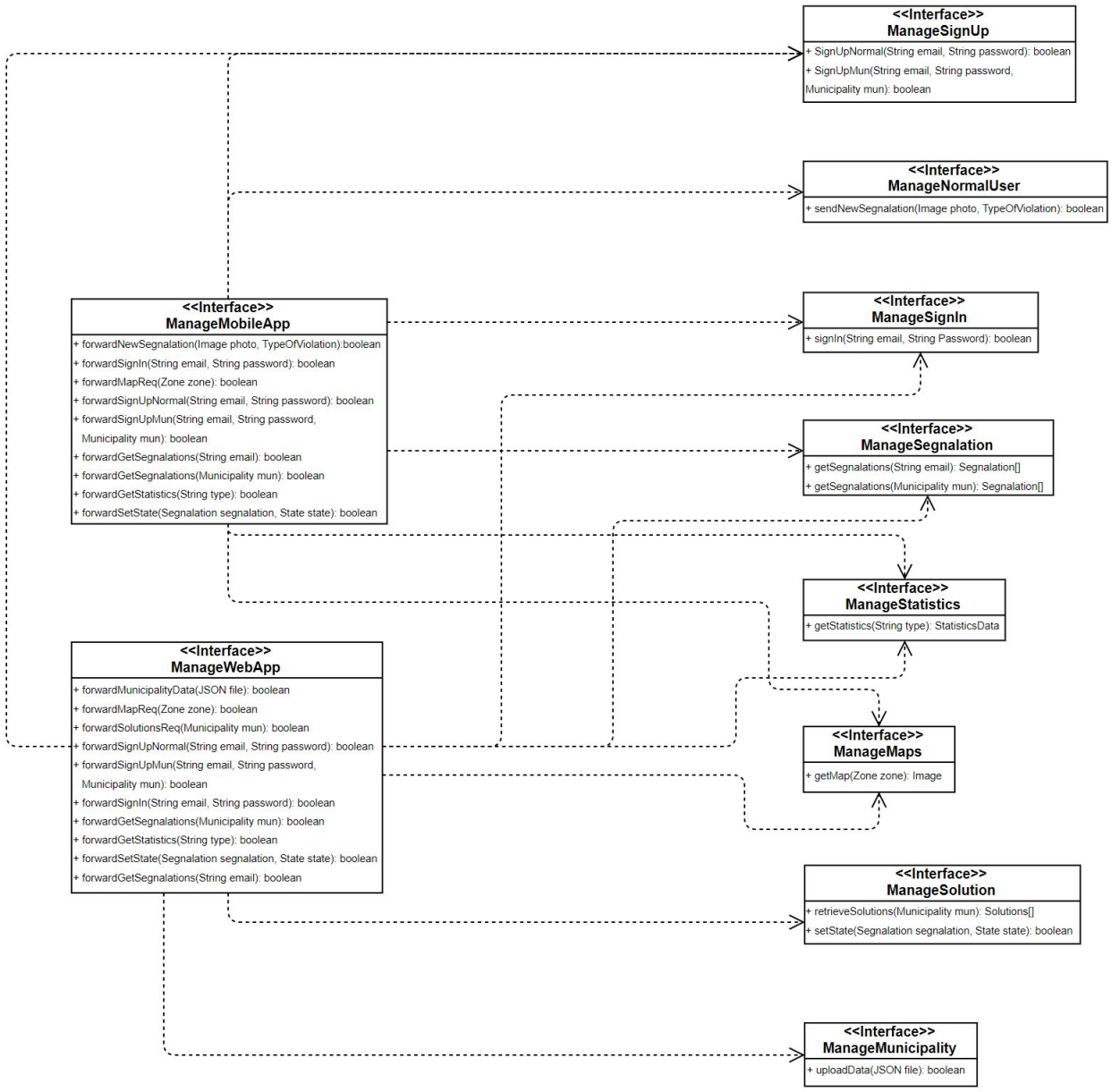

Figure 15 - Component interfaces diagram

2.6 Selected architectural styles and patterns

2.6.1 Client-server architecture

As it has been described in this chapter, SafeStreets has a client-server architecture with three tiers and it has been examined in all its components but now the reader will be able to find the justification that explains this choice: this architecture allows to decouple the three fundamental parts for the system, that are the interface with the user, the business logic and the management of the stored data. These three parts offer services that differ greatly from each

other and for the proper functioning of the whole it is necessary that they communicate through appropriate interfaces which are the only shared border information while the internal organization of each level is unknown to the others. This modularization makes it possible to make changes, updates, improvements because of changes of requirements or technologies without compromising the functionalities of the other levels since the communication interfaces will be respected. This division into levels also guarantees the freedom to customize the configuration of each level according to the needs of the system, especially this is visible in tier 2 and 3. The client-server approach allows to remove from the client side the business logic and the direct access to data; its only function is to send requests coming from the user, who interacts through it, and understand the answer, which has a known structure, coming from the second tier, read it and offer it to the final user. This architecture results to be an optimal choice in this case since it permits to manage the access to data in a tier different from the one which the user interact with. Moreover, since SafeStreets offers its services both through a web app and a mobile app, the client server approach assigns to the presentation level the task of processing the response of the second level, which is independent of the device used by the user, and of formatting it according to the hardware. Finally, availability and reliability are ensured as the system is divided into multiple servers that also use temporary data to speed up response times and allow the system to have better performances when failures on a specific server occur.

2.6.2 Representational state transfer

A further step towards the decoupling of clients and servers is achieved through the REST architectural style which imposes some constraints:

- The interaction must be stateless, there is no session. For the server each request of the client is new, it doesn't store any information about the client. The interactions that must be stateful according to the needs of the client must contain all the information, like authentication details, inside of them.
- The client must be able to cache the results obtained from the server in order to avoid useless interactions. By the way responses must be declared cacheable or not.
- The system must have a layered architecture but this constraint was already satisfied as it has been discussed before.
- Having a uniform interface for sharing resources and in our case a json format of data.
- The system must have a client-server architecture but this is easily satisfied by SafeSteets as it has been previously shown. This permits to both of them to change their inner structure but they must respect the communication interface.

The advantages that REST architecture introduces are several: in fact it improves the separation between the client and the server making them scalable, reliable and able to develop independently. Moreover, it makes communications with the user independent from the hardware he is using since the messages have a specific pattern and a specific format well

known. The REST communication is used to let the first and second tier interact, in fact the applications running on the devices used by users send HTTP requests to the web and application server.

2.7 Other design decisions

In this paragraph some of the aspects represented in figure 2 and not well discussed will be now clarified in order to have a better vision of the system.

2.7.1 Load balancer

In the referred figure the reader can find three nodes that represent load balancers, they have been introduced since the system is expected to receive a large amount of requests from users from different types of hardware and in this way they can be easily managed in order to offer a better experience to the final user. In this case the costs introduced by these nodes are justified because they contribute to respect the non-functional requirements. In fact, they run the Least Connections balancing algorithm in order to fairly distribute requests from clients; in this way no server will be overloaded and will be able to respond as quickly as possible. These components also increase the availability of the system since in case of failure of a server they will redirect traffic to the remaining ones.

2.7.2 Firewall

All the requests coming to the web and application servers first are analyzed by a next-generation firewall that acts as an intermediary managing and controlling network traffic in order to defend resources and validate the access without lowering network performance too much. Doing this it blocks malicious traffic requests packets while allows legitimate traffic through and it also employees TLS for critical requests for encrypted traffic inspection. These components result to be extremely important for SafeStreets because the data stored must be secure since there are also private information that the system shares with municipalities. Obviously they impact on system performances but these components perform such an important function that they are necessary.

2.7.3 Relational database

In the system represented in figure 2 the reader can see two relational databases: it was thought that this could be the best solution since one contains the reports that are the protagonists of SafeStreets and for this reason it was preferred to divide them from the rest of the data. They are managed by the DB server that knows their internal organization. The choice of using relational databases is due to the fact that knowledge of the SQL language is

widespread and can be exploited to its full potential. Moreover, these DBs allow to create relationships between tables that are very useful in the case of SafeStreets. Finally, they have a reliable structure since they follow the ACID properties that permit to have valid transactions in cases of failure.

2.7.4 Google Maps

The tool that SafeStreets use in order to obtain the maps of the zones requested by users is an API of the Google Maps platform called "Maps Static API" that allows you to receive, following a specific HTTP request, the requested map. This service has been chosen since it offers static maps that can be easily edited by the system which adds colors to underline different grades of occurence of parking violations.

2.7.5 OCR service

As it has been described in this document and in the RASD, when a normal user sends a new segnalation SafeStreets must retrieve the plate of the car interested in the accident. In order to do so, it exploits a free library developed by Google called “Tesseract OCR” which can offer this kind of service efficiently. This library has been chosen since it offers a free solutions to the problem and it is widely documented so integrating it should be immediate. In the following image the reader can understand the workflow that Tesseract OCR follows when it processes an image sent by the user.

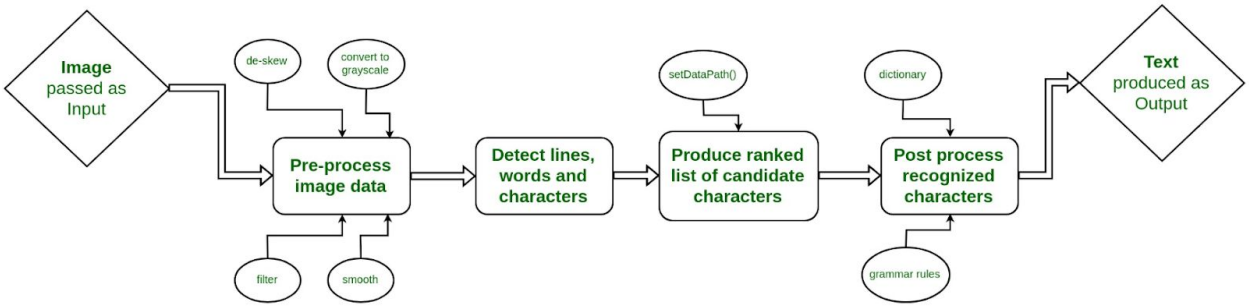

Figure 16 - Workflow of Tesseract OCR library

Safestreets uses this library running on the application server instead of an API since it wants to avoid to send over the net the images sent by its users so this decision has a motivation that deals with privacy and security of data.

3 USER INTERFACE DESIGN

In this section, we introduce UX diagrams to show how the user's flow inside the system should be. The mockups of the UI can be found in the paragraph 3.1.1 of the RASD.

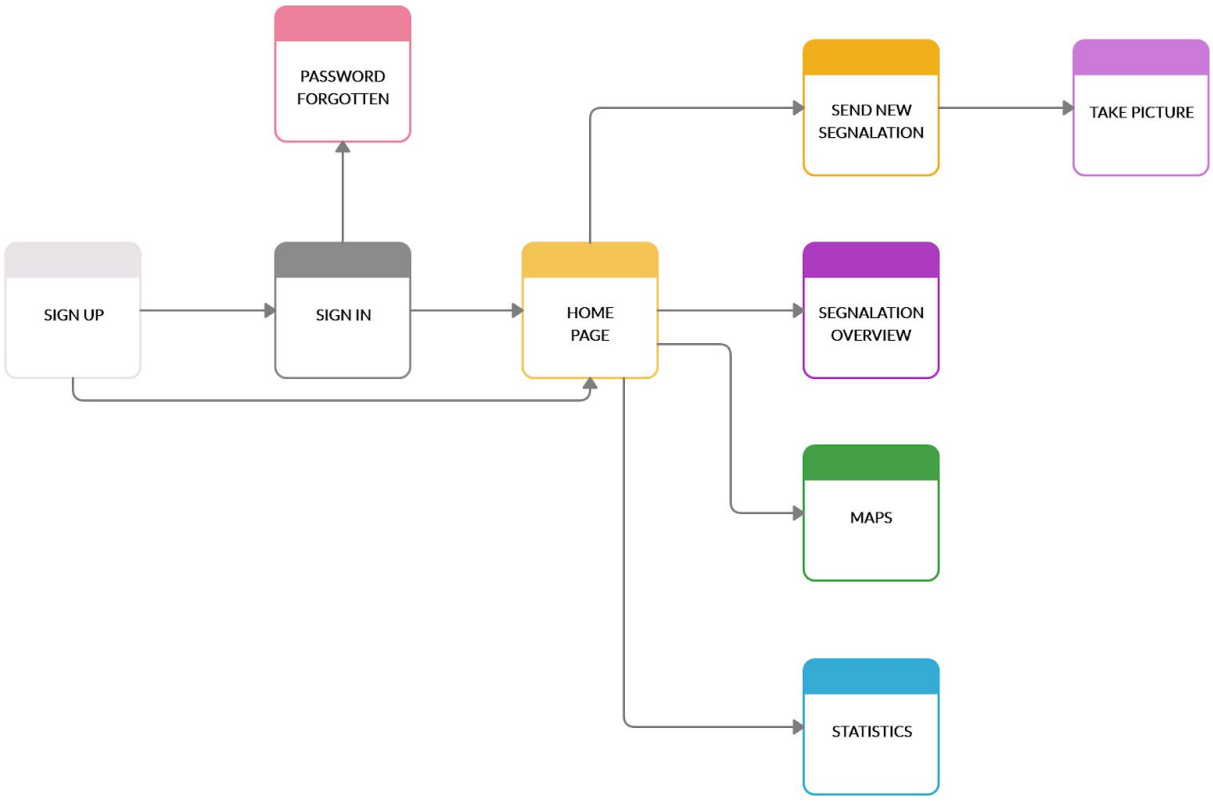

Figure 17 - MobileApp UX

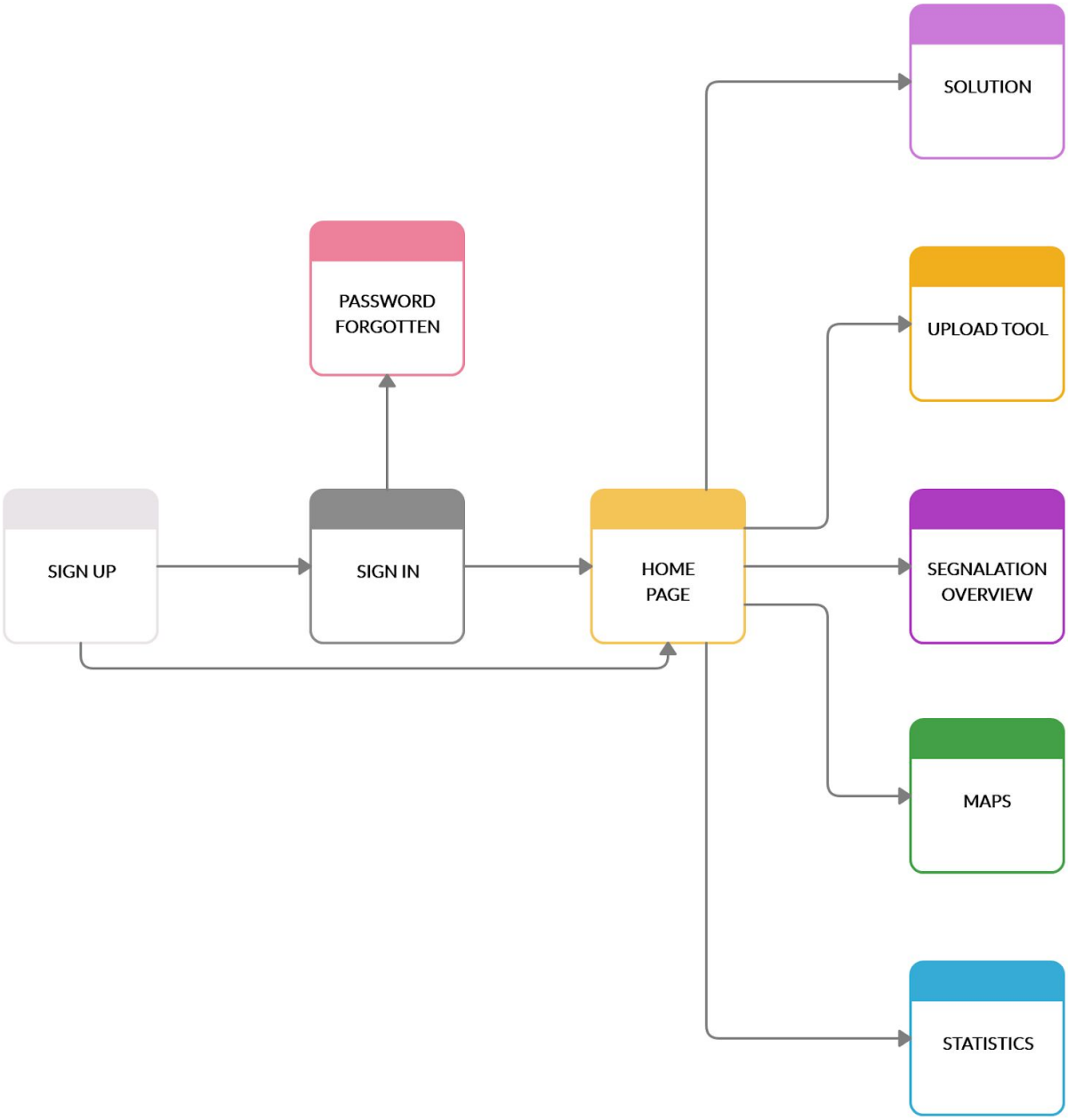

Figure 18 - WebApp UX

4 REQUIREMENTS TRACEABILITY

4.1 Functional requirements

The design document has been designed following the goals presented in the RASD. In order to achieve them, the relative requirements need to be guaranteed. Here we specify how each requirement is fulfilled by one (or more) of the components defined.
It's important to notice that (as can be seen in the Component diagram) since all the components of the application server, except the Router, directly interact with the DB through the DBManager, this latter component is omitted from the following lists, but it must be clear that it belongs to all of the cases where a "Manager" component of the application server takes part. For the same reason, since the Router component only performs a redirect action, it is also omitted.

1. **R1:** The system must allow the user to sign-up.
    - **SignUpManager**: this component let the user register to SafeStreets. It stores the user's email and password in the DB. It also stores the name of the municipality in case of municipality users.
2. **R2:** The system must allow the user to sign-in.
    - **SignInManager**: this component let the user to sign in. It queries the DB for a match email/password, in the case this is found, it grants access to the system.
3. **R3:** The system must store the segnalations sent by users in order to collect data of parking violations and where they occur.
    - **NormalUserManager**: this component let normal users to send segnalations to SafeStreets via the MobileApp. These segnalations are stored in the SegnalationDB.
    - **MunicipalityManager**: this component let municipality users to provide their own data about violations to SafeStreets, and they do so via the WebApp. These segnalations are stored as well in the SegnalationDB.
4. **R4:** The system must use only the segnalations whose state is "verified" to update its maps and statistics presented to users if the municipality which has control over that specific zone, on the other hand it will use the non verified segnalations if the municipality is not registered.
    - **SegnalationManager**: this component also let the municipality users to mark a segnalation as verified or not.
    - **StatisticsManager**: performs data analysis and computes all relevant statistics regarding segnalations.

- **MapsManager**: this component is responsible for returning the map associated to a given zone, decorated with highlights indicating which zones are most prone to violations.
- **GoogleMapsAPI**: exploited by the MapsManager to retrieve the static map of the zone requested by the user.

5. **R5:** The system must allow the user to choose which statistics or maps visualize between those available.
    - **StatisticsManager**: let the user access the desired statistics.
    - **MapsManager**: returns the map requested by the user.
    - **GoogleMapsAPI**: exploited by the MapsManager to retrieve the static map of the zone requested by the user.
6. **R6:** SafeStreets should allow municipality users to use a tool that allow them to send data to the system.
    - **MunicipalityManager**: provides a tool for municipality users to send their own data to the system.
7. **R7:** When a municipality user visualize a specific segnalation the system must allow him to modify, using a tool, the current state of the report.
    - **SegnalationManager**: despite providing the requested segnalations to users, it also allows a municipality user to change the state of a segnalation concerning a violation in the territory of that specific municipality.
8. **R8:** The system must provide the requested data if and only if the user has the permission to visualize them.
    - **SegnalationManager**: in the case of a normal user, it sends back only the segnalations sent by him. A municipality has access to all the segnalations concerning their territory.
9. **R9:** The system must allow municipality users to visualize the current state of a specific segnalation.
    - **SegnalationManager**: despite providing the requested segnalations to users, it also allows a municipality user to view the state of a segnalation concerning a violation in the territory of that specific Municipality.
10. **R10:** SafeStreets must use the data sent by municipalities to update the solution table.
    R14: The system must update its solution table, evaluating if the occurrence of violations has decreased, whenever the municipality reports that a solution has been adopted.
    - **SolutionManager**: when SafeStreets provides a solution to a municipality, it monitors the zone interested by it, if the municipality has confirmed that it had applied the solution, and the segnalations regarding that zone (coming both from normal users and from the Municipality itself) in order to decide if the proposed solution was effective or not.
    - **MunicipalityManager**: provides a tool for municipality users to send their own data to the system.
11. **R11:** Given a problem, SafeStreets must be able to find the most suitable solution using the solution table in order to send it to the municipality.
    - **SolutionManager**: it accomplishes exactly this task exploiting the solution table.

12. **R12:** SafeStreets must send suggestions to the municipality users who have the problematic zone under their control.
    R13: The system should recognize zones where the occurrence of parking violations can be reduced and find a solution to propose.
    - **SolutionManager**: when a certain threshold of violations occurred in a zone, this component search for a solution in the solution table concerning the particular type of violation and then sends this solution to the Municipality whose territory includes the interested zone.
13. **R15:** The system must allow the user to attach to the segnalation a photo just taken through the app.
    R16: The system must allow the user to specify the type of parking violation that just occurred.
    - **NormalUserManager**: these tasks can be accomplished through the form normal users must fulfill when they want to upload a new segnalation. A home-made camera app is provided by the system, in order to avoid fake photos and a choicebox allow to choose the type of violation.
    - **MunicipalityManager**: provides a tool for municipality users to send their own data to the system.
14. **R17:** The system must complete the segnalation with the correct date, time, the licence plate, the name of the street and the username of the author of the segnalation.
    - **NormalUserManager**: this happens in the NormalUserManager, where the OCR algorithm is run. GoogleMapsAPI are exploited to infer the name of the street. The other information are taken from the request of upload of the user.
    - **GoogleMapsAPI**: exploited by the NormalUserManager.
15. **R18:** The system, when it shows to municipality users the list of all the segnalation of the zones under their control, must take care of removing the email of the author from the reports.
    R19: When a municipality user chooses to visualize the list of all the violations, SafeStreets must provide him only those which occurred in the zones under the control of the specific user.
    - **SegnalationManager**: this is done by the SegnalationManager before presenting the segnalations to the Municipality.
16. **R20:** SafeStreets must send an email to the correct municipality to notify the violation each time it stores a segnalations sent from a zone under the control of that specific municipality.
    - **NormalUserManager**: when a segnalation is correctly received and stored in DB, the NormalUserManager queries the DBManager in order to get the email of the Municipality in whose territory the violation occurred and sends an email to it.
17. **R21:** The system must present in the homepage of each user the list of segnalations whose author matches the email of the user who logged in.
    - **SegnalationManager**: once the user is logged in, it retrieves the segnalations sent by a specific user using its email.

4.2 Non-functional requirements

- **Performance:** SafeStreets tries to offer the best possible experience to the user that interact with it and it achieves this aim through many ways that have been described in the previous paragraphs and that will be shortly recalled in the following: the system has an architecture that replicates fundamental modules in order to answer requests coming from the presentation level; this has a positive impact on performance when combined with a load balancer that prevents these machines from being overloaded. For this reason they are able to work at full capacity when they are solicited.
  In order to provide a rapid and satisfactory answer to the user, it should also be remembered that the system uses pre-computed data (for example for the realization of maps) which are periodically updated in time slots in which the application server is less stressed.

- **Reliability:** as it has been stated in the RASD some tools of Statistical Modeling and Estimation of Reliability Functions are used. Moreover, the replication of some nodes in the architecture of the system causes a higher reliability.

- **Availability:** in the RASD it has been stated that SafeStreets has the aim to obtain an availability of at least 99.9% and it should be obtained through the replication just discussed; in fact, even if a machine in a node has a failure, the load balancers should be able to avoid a complete failure of the system redirecting the request to other available machines.

- **Security:** the data stored in the database are encrypted and the DB server is responsible for this operation: it acts as a layer between the DB and the application server that encrypts data when put it in the database and decrypt them when it must return them to the application server. Moreover the architecture exploits firewalls in order to check the requests coming from the users and discard those that can damage the system.

- **Maintainability:** this requirement can be easily obtained through the adopted three-tier architecture which decouples each layer from the others giving to it some independence in case of changes of its inner part as discussed in 2.6.1. Moreover, those who actually will implement the system are required to use design patterns that permit to achieve a higher level of maintainability.

- **Portability:** this aspect has been widely discussed in the document but here it is just recalled that the first and second tier have a RESTful interactions that implies that the format of requests and answers are the same, it does not depend on the device the user is using.

5 IMPLEMENTATION, INTEGRATION AND TEST PLAN

5.1 System Overview

In this chapter we present the plan to implement, integrate and test the SafeStreets system. Notice that the system is composed by the following subsystems:

- MobileApp
- WebApp
- WebServer
- ApplicationServer
- DBManager

Notice further that both the DBMS and GoogleMaps are already implemented and deployed, so we rely on them as being fully operative. The general approach adopted for the implementation and integration of the various parts of the system is bottom-up. For what concern this macro-view of the system, we will start the implementation from the components of the ApplicationServer, because that is the core of the system that requires more work. Specifically, we first implement and test the components piloted by the MobileAppRouter, since we want to have the MobileApp running first in order to have a solid base, with some important functions, on which the other aspects of the system (especially those thought for municipalities) can be added. In doing so we will write a driver for the MobileApp, a stub for the DBManager and ignore temporarily the Web side. Then the DBManager will be implemented since it is needed by the ApplicationServer to store and perform its computations. Following we complete the first part of the product by implementing the MobileApp. Finally, the WebServer and the WebApp are implemented. Unit testing is performed on each component when it is implemented, and when they are all implemented we merge them performing integration testing.

The ordered list of components to be implemented is then:

1. MobileAppRouter and the components it drives (green components in figure 19)
2. DBManager
3. MobileApp
4. WebAppRouter and the components it drives (pink components in figure 19)
5. WebServer
6. WebApp

The routers are implemented and tested once the components to which they must forward requests are unit tested, just before the integration test.

5.2 ApplicationServer

Since the crucial features of SafeStreets are implemented on the ApplicationServer, we will now examine its implementation, integration and testing in detail.

We identify a set of features the ApplicationServer is responsible for, by providing a table specifying for each of those their importance for the customer and their implementation difficulty.

Feature	Importance for the customer	Difficulty of implementation
SignUp and SignIn	Low	Low
Send new segnalation	High	Medium
Compute statistics	High	High
Visualize maps	Medium	Medium
Send municipality data	Medium	Low
Receive solutions	High	High
Visualize segnalations	High	Low
Mark segnalation as verified	Medium	Low

The implementation of the features “Visualize maps” and “Receive solutions” is not considered to be easy because they both require the developers to exploit some machine learning techniques.

From the Component view of the ApplicationServer we can see that Manager components don’t interact directly with one another, still they exchange data through the DB. For example, the MapsManager does its job by using data provided by the StatisticsManager. For this reason, a precedence relation is identified inside the ApplicationServer.
As mentioned before, the components which offer the interfaces exploited by the MobileAppRouter will be implemented first.
In the following the reader can visualize a graphical overview of the dependencies between system components and as a consequence the order which will be followed for the implementation.

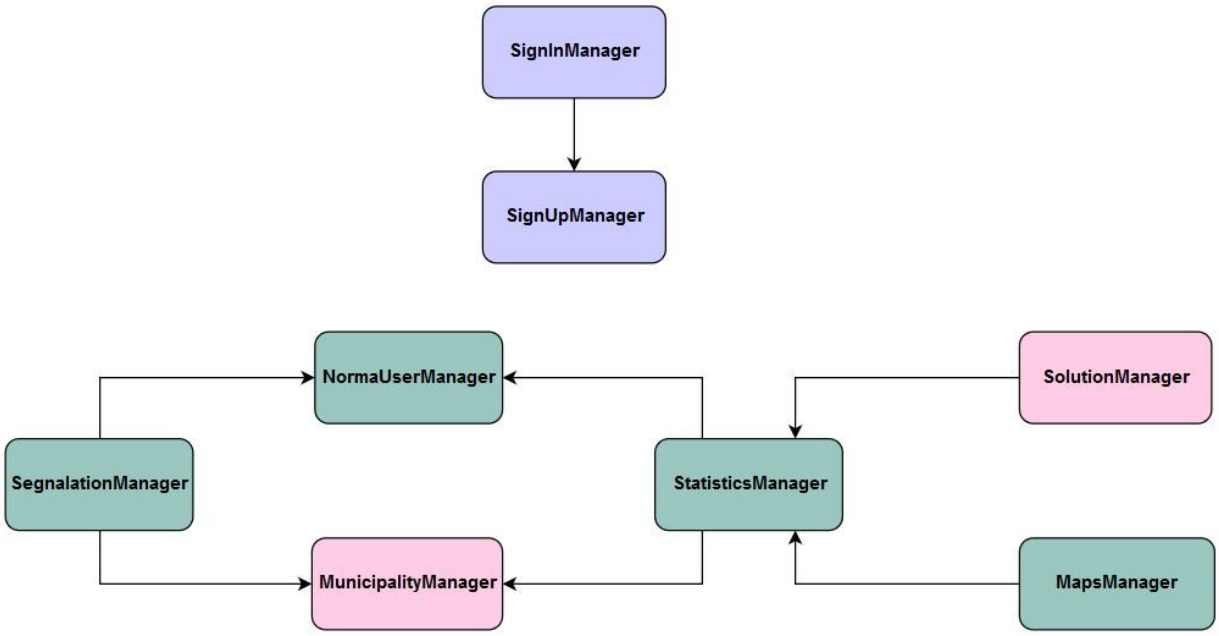

Figure 19 - System's components

1. The **SignInManager** grants access by finding a match email/password in the DB which is stored by the **SignUpManager**, so this latter component must be implemented first. While this pair of components (blue components in figure 19) comes logically before all the others, they considered to be of low difficulty of implementation, so they can actually be implemented in parallel to the other managers. They contribute to the feature "Sign up and sign in"
2. The **NormalUserManager** provides the data on which the system works, so it must be implemented before the other managers which use it. This component contribute to provide the feature "Send new segnalation".
3. Once the system has received data, the **SegnalationManager** and the **StatisticsManager** should be implemented in order to provide the features "Compute statistics", "Visualize segnalations" and "Mark segnalation as verified". They are independent from one another so they can be built in parallel.
4. On top of the StatisticsManager the component **MapsManager** can be built. It exploits data computed by the StatisticsManager in order to provide its services, namely "Visualize maps".
5. Once the system is fully integrated and tested from the MobileApp side, the Web counterpart can be built. In doing so, the **MunicipalityManager** component that provides the feature "Send municipality data" and the **SolutionManager** component which makes possible to receive solutions are developed. These two can also be implemented in parallel.

5.3 Integration

Integration testing is done on the following sets of components in different steps here described:

- MobileAppRouter and the components it drives.
  At the first stage, when all the components which offer the interfaces exploited by the MobileAppRouter are built, we can merge them together with the router itself. As a result, a first version of the ApplicationServer is obtained.
- MobileApp, ApplicationServer and the DBManager.
  This integration makes the system operative on the mobile side.
- WebAppRouter and the components it drives. Then it is integrated with the ApplicationServer.
- All the components mentioned before are integrated together and this completes the full system.

It must be remembered that the OCR library is embedded into the NormalUserManager inside the ApplicationServer. For this reason its integration and integration test phase is included in the NormalUserManager unit test phase.

5.4 System testing

This phase is placed after the integration testing and its aim is to verify that the system actually complies with the declared functional and non-functional requirements. The types of system testing performed are:

- Stress testing: used to check the stability of the system, determine if there are modes of failure and see how the system recovers after a failure.
- Load testing: it is made in conditions of extreme load and it is useful because through it many bugs related to memory can be discovered and corrected.
- Scalability testing: it is used to identify the user limit for web and mobile application and if it is able to offer a good experience to the end users.
- Performance testing: it gives information about how the system behaves in terms of responsiveness and stability under a particular workload. It can also discover bottlenecks the weakness of the system.

When this phase is concluded, the acceptance testing will start but it will not be described below since it goes beyond the scope of this document.

6 EFFORT SPENT

De Vita Simone

13/11/2019: 4h
16/11/2019: 4h
18/11/2019: 2h
20/11/2019: 1h
21/11/2019: 1h
24/11/2019: 1h 30 min
25/11/2019: 2h 30 min
26/11/2019: 1h 30 min
30/11/2019: 1h
01/12/2019: 1h
02/12/2019: 2h
08/12/2019: 30 min

Fabio Fontana

13/11/2019: 4h
15/11/2019: 2h 30 min
18/11/2019: 1h 30 min
19/11/2019: 1h
22/11/2019: 3h
23/11/2019: 4h
25/11/2019: 2h 30min
26/11/2019: 1h
28/11/2019: 3h
29/11/2019: 1h
02/12/2019: 2h
08/12/2019: 2h