POLITECNICO
MILANO 1863

DD

Design Document

Authors

Version: 2.0

Date: 14 December 2019

Professor:

Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide more technical and detailed information about the software discussed in the RASD document. It will represent a strong guide for the programmers that will develop the application considering its different parts: the basic service and the two advanced functions.
In this DD we present hardware and software architecture of the system in terms of components and interactions among those components. Furthermore, this document describes a set of design characteristics required for the implementation by introducing constraints and quality attributes.
It also gives a detailed presentation of the implementation plan, integration plan and the testing plan.
In general, the main different features listed in this document are:

- The high-level architecture of the system
- Main components of the system
- Interfaces provided by the components
- Design patterns adopted

Stakeholders are invited to read this document in order to understand the characteristics of the project being aware of the choices that have been made to offer all the functionalities also satisfying the quality requirements.

1.2 Scope

SafeStreets is a crowd-sourced application that aims to provide users with the possibility to notify authorities when traffic violations occur, and in particular parking violations.
The application can be used both by anonymous users and by municipalities for different purposes and in different ways. On one hand users can create reports attaching pictures of violations and can send them to SafeStreets, on the other hand municipalities can provide information about accidents, send traffic tickets generated thanks to the report received and get important suggestions about the measures that can be adopted to reduce violations.
SafeStreets also provides information when requested and this is done using a different level of details, according to the applicant. Effectively, SafeStreets can be also seen as a broker allowing the communication between the municipalities and the users.
The few paragraphs just read represent an overview of the main functionalities offered by the system: more detailed information can be found on the RASD document.
One important aspect on which we can focus on is the strong impact that such application would have: the deployment of SafeStreets can lead to the increase of wellness in the municipalities that will use it. Citizens will give their contribution and help authorities to enforce the law.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Validation	Municipality sends a validation when confirms that the report contains a punishable violation
Secure	SafeStreets defines secure a report when the photos are clearly not fake and the metadata of the pictures match with the information written on the report
Metadata	Data directly extracted from the picture through an algorithm
Query	Synonym for request
Individual Request	Request referred to a single license plate
General Request	Request referred to a specific type of a violation or vehicle (for example the frequency)
Area Request	Request referred to a specific area of the city (for example the safety)

1.3.2 Acronyms

RASD	Requirements Analysis and Specification Document
DD	Design Document
GPS	Global Positioning System

1.3.3 Abbreviations

Gn	Goal number n
BS	Basic service of SafeStreets
AF1	Advanced function 1 of SafeStreets
AF2	Advanced function 2 of SafeStreets
Rn	Requirement number n

1.4 Revision history

Date	Modifications
9/12	First Version
14/12	Second Version. Remove rest and correction in the document.

1.5 Reference Documents

- Specification Document: “SafeStreets Mandatory Project Assignment.pdf”
- Slides of the lectures

1.6 Document Structure

- **Chapter 1** describes the scope and purpose of the DD, including the structure of the document and the set of definitions, acronyms and abbreviations used.

- **Chapter 2** contains the architectural design choice, it includes all the components, the interfaces, the technologies (both hardware and software) used for the development of the application. It also includes the main functions of the interfaces and the processes in which they are utilised (Runtime view and component interfaces). Finally, there is the explanation of the architectural patterns chosen with the other design decisions.

- **Chapter 3** shows how the user interface should be on the mobile and web application.

- **Chapter 4** describes the connection between the RASD and the DD, showing the matching between the goals and requirements described previously with the elements which compose the architecture of the application.

- **Chapter 5** traces a plan for the development of components to maximize the efficiency of the developer team and the quality controls team. It is divided in two sections: implementation and integration. It also includes the testing strategy.

- **Chapter 6** shows the effort spent for each member of the group.

- **Chapter 7** includes the reference documents.

2. Architectural Design

2.1 Overview

The architecture of the application is structured according to three logic layers:

- **Presentation level (P)** handles the interaction with users. It contains the interfaces able to communicate with them and it is responsible for rendering of the information. Its scope is to make understandable the functions of the application to the customers.

- **Business logic or Application layer (A)** takes care of the functions to be provided for the users. It also coordinates the work of the application, making logical decisions and moving data between the other two layers.

- **Data access layer (D)** cares for the management of the information, with the corresponding access to the databases. It picks up useful information for the users in the database and passes them along the other layers.

The architecture has to be made in client-server style. Client and server are being allocated into different physical machines and their communication takes place via other components and interfaces located in the middle of the structure, composed by hardware and software modules.

The process begins with the invocation of a method to provide any functionality to the client, like sending a report or requiring some information about violation or accidents. Then, the invocation of a specific method is caught by the server and its behaviour depends on the required function.

- **Basic Service**:

    - User wants to send a report: server analyses the information received and, if it results to be a real violation, stores them in the database and re-route the same information toward the nearest authority.
    - A client (user or municipality) wants to require information: the server receives the specific request, takes the proper information from the database and sends data back to the client.

- **Advanced Function One**:

    - User wants to check unsafe areas also knowing about accidents occurred/Municipality wants tips for possible intervention and can send information about accidents: the server receives the specific request, takes the proper information from the database and sends data back to the client

- **Advanced Function Two**:

    - Municipality wants to generate traffic tickets thanks to the information provided by the application: the server performs further checks to guarantee the veracity of the pictures received, then if the violation is real the municipality can generate the traffic ticket, which is sent to the server and stored in the database to build statistics.

The application is a distributed application, therefore the layer described above are being installed in different hardware levels, called tiers. The approach used in that case is described below in the document.

2.1.1 High Level Components

The hardware architecture chosen for the distributed application is **Three-Tier**. The three application layers are subdivided, therefore, among many dedicated machines, i.e.: a tier to interface with client, a middle tier for the application level, and another server for the database management. This approach is beneficial because the middle tier can maintain persistent connection with the DBMS, which is less expensive. Furthermore, having an intermediate machine between client and server can guarantee more security to the access control of the database from the users.

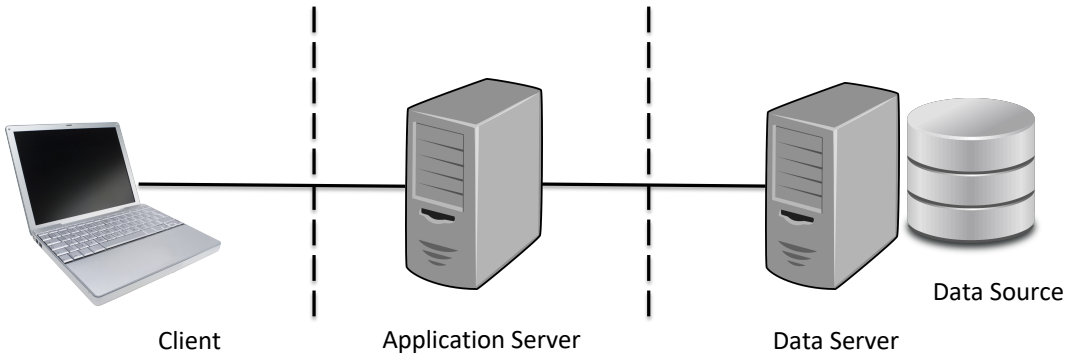


Figure 1 – Three-Tier Architecture

The figure below describes very high-level components and their interaction.

The client's devices can be a computer or a mobile phone for users, and a personal computer for the municipality. Smartphone using the application is connected directly with the application server, while personal computer using the web app are managed by the web server. Also, the two servers which constitute the middle tiers communicate with each other and the application server communicates with the database server (called also DBMS). Finally, the application server is responsible for using the API that allows to use the maps for different functions of the application: this API is provided by GoogleMaps.

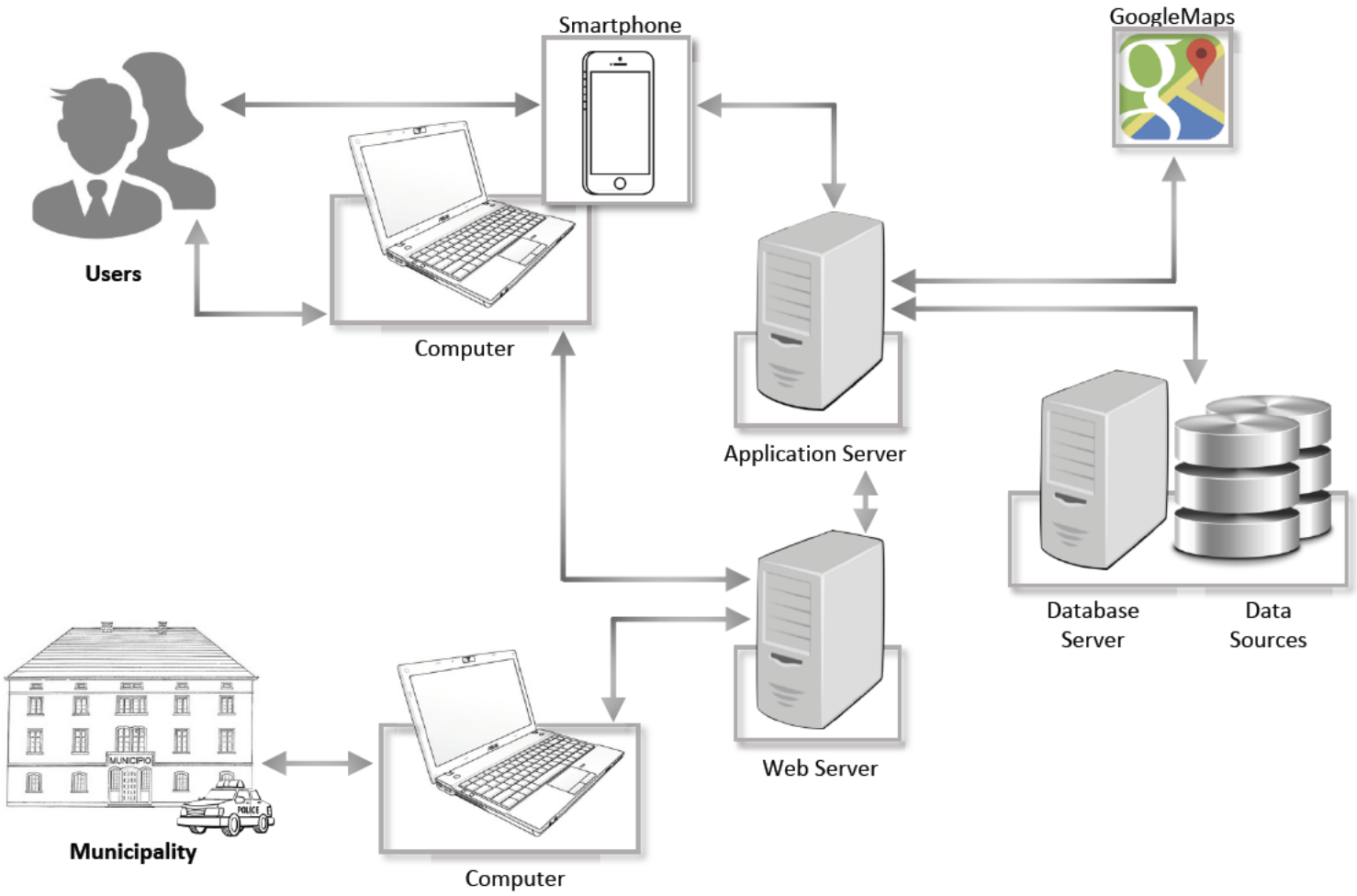


Figure 2 – System Architecture

In the figure above are not included so many details of the architectural choice that have been made for the application. In the next sections, a more accurate description of the additional architectural design aspects that have been chosen to improve the realization is presented.

2.2 Component View

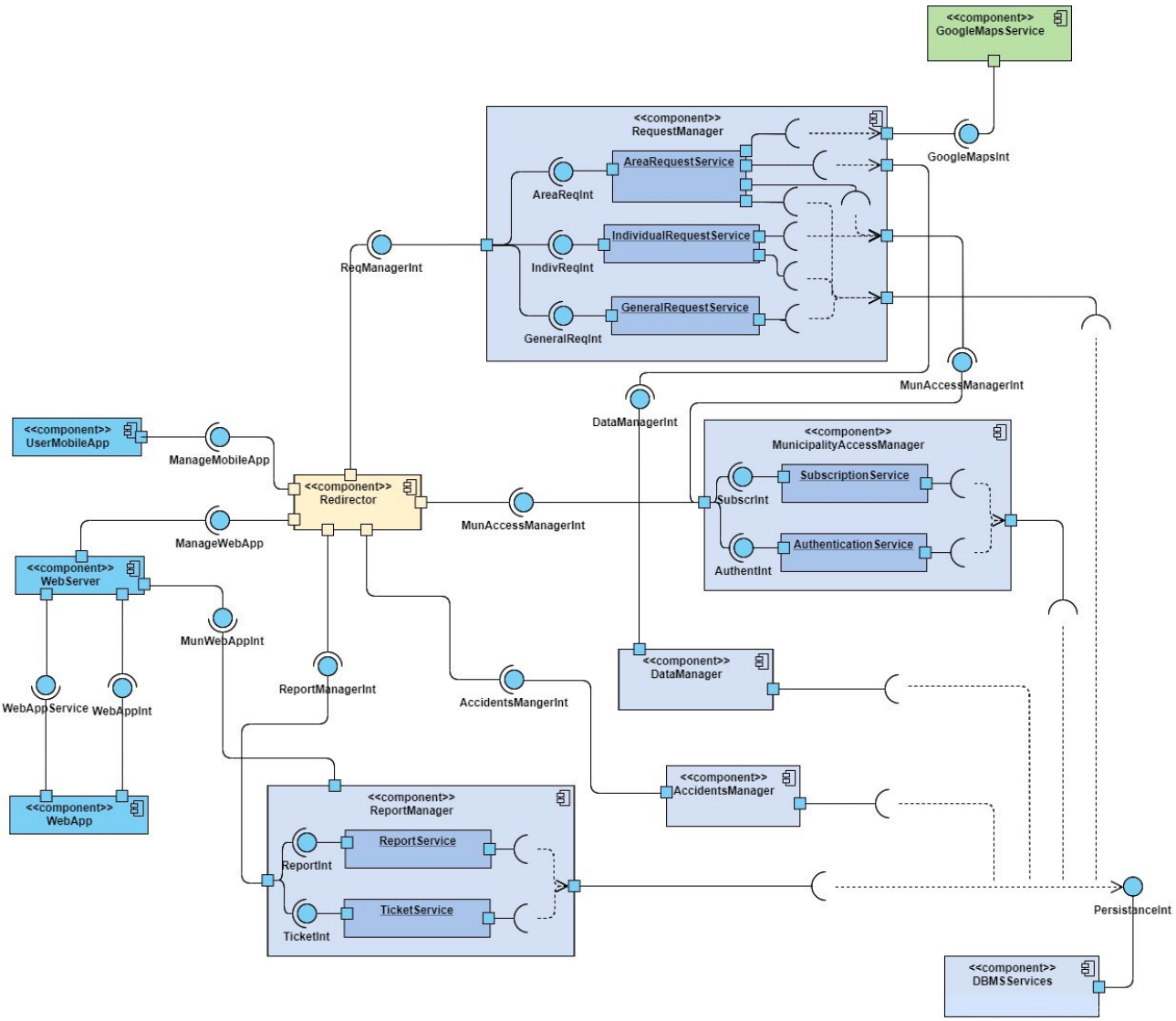


Figure 3 – Component Diagram

The following component diagram gives a specific view of the system focusing on the representation of the internal structure of the application server, showing how its components interact. The application server contains the business logic of our software. Other elements in the diagram, besides the application server, have been depicted in a simpler way just to show how the communication is structured among these components and the application server.

- **MunicipalityAccessManager:** this component comprises two subparts:
    - **SubscriptionService:** it manages the subscription of a municipality to the system. To validate the subscription a form, including some documents needed to certify its identity, has to be forwarded. Furthermore, the willingness to subscribe to the additional functions is also notified to this component that handles the request in a proper way.

    - **AuthenticationService**: it is meant for the municipality which accesses the web app and needs to log in to be able to use all the functionalities to which it is subscribed. A certain level of security is needed to be sure to authenticate the right subject and for this purpose some security mechanisms have been chosen to avoid malicious people to have access to personal information. These mechanisms are not mentioned in this document.

- **RequestManager:** this component manages the Requests done through our application. It handles all the different types of requests (General, Individual and Area) and, using the Authentication Service, performs also an access control based on the requested queries. For example, it prevents the common user to access individual data of an offender making an Individual Request.

- **AccidentsManager**: this component manages the notifications of the municipality that sends information about the accidents occurred in the municipality itself. These data are then collected in the database.

- **DataManager**: this component is meant for different purposes. The main function provided is to update periodically the safety level of different areas also considering the last violations and accidents in that specific location. When possible, the Data Manager also supplies some suggestions to the municipality together with other useful data. In addition, it provides to cross periodically the information in the database and build statistics based on them.

- **ReportManager**: this component handles the receipt of the report from the user and the retransmission to the municipality in which the violation occurred. Furthermore, if the addressed municipality has the advanced function 2 enabled, the report manager (in particular, the **ReportService**) also runs an algorithm to check the veracity of the report so as to provide a more secure service. Every other aspect related to the control of reports is handled in this component: the validation of reports from municipality is notified to the report service which then handles their memorization; the **TicketService**, instead, keeps track of the tickets issued to validated reports.

- **Redirector**: this component simply dispatches the requests and calls to methods from the users and the municipality to the core of the application server. Every method is redirected to the proper component that can handle it. Also, responses and data sent back pass through this component to reach the applicant.

- **DBMSServices**: this is the component that allows every other component in the system to interact with the database. The Interface provided by this component contains all useful methods to store, retrieve, update data into the database from different actors. Every internal component of the application server uses some methods of its interface.

The external component of the system is also mentioned:

- **GoogleMapsService**: this is the component that provides a useful interface used to visualize the map of a specific location requested by the user/authorities. It is used by the component **AreaRequestService**.

It is also important to notice that the **ReportManager** uses an interface provided from the WebApp through the WebServer. It is mainly meant to call methods that allow to send reports to the municipality.

2.2.1 Additional Specifications

The configuration adopted is of type called **Thick client** or Fat client, in which, in addition to the presentation logic, in the client node is allocated also a part of the business logic. This can be an advantage for the mobile application that is able not only to display functions but also to compute functions, for example checking the correct compilation of the fields directly from the client application.

To fasten the communication cache is used on the client side. This introduces a good advantage on performance given to the fact that part of the commun cation over the nte net can be avoided when he cache already contains the requested data: this limits the traffic over the network and decreases the load on the server. Obviously, on the other hand, a mechanism has to be implemented to invalidate data on the cache when they become obsolete.

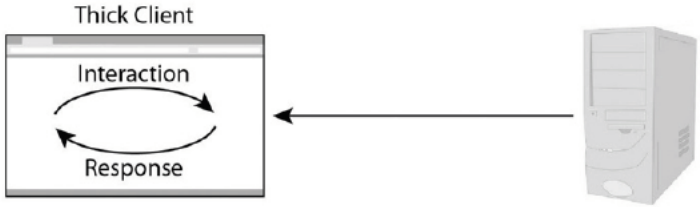


Figure 4 – Thick-Client

The **Application Server** and the **Web server** have been replicated two times to guarantee the reliability of the application if one of them goes down.

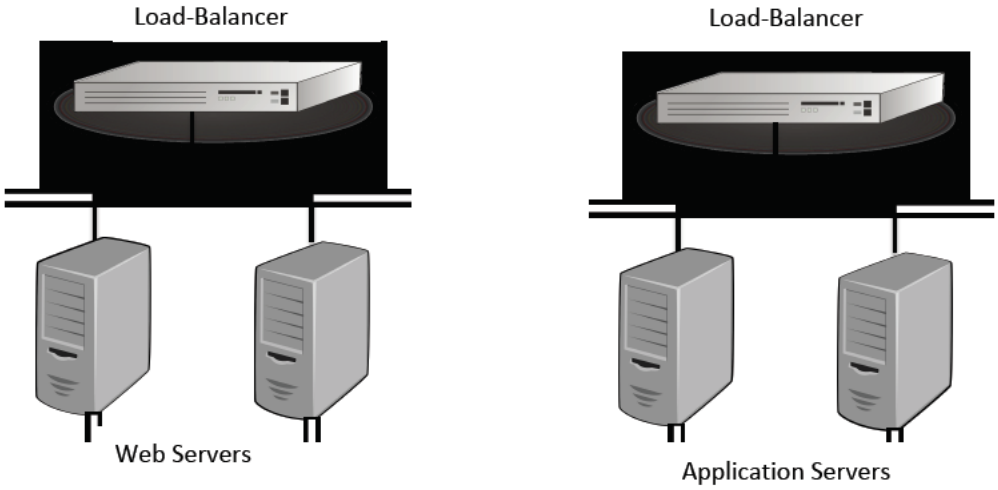


Figure 5 – Replication of Web and Application Servers

In addition, also the **Redirector** component has been replicated to avoid overload of requests at the same time. In the diagrams, nevertheless, it is reported like a unique element for the sake of simplicity.

2.3 Deployment View

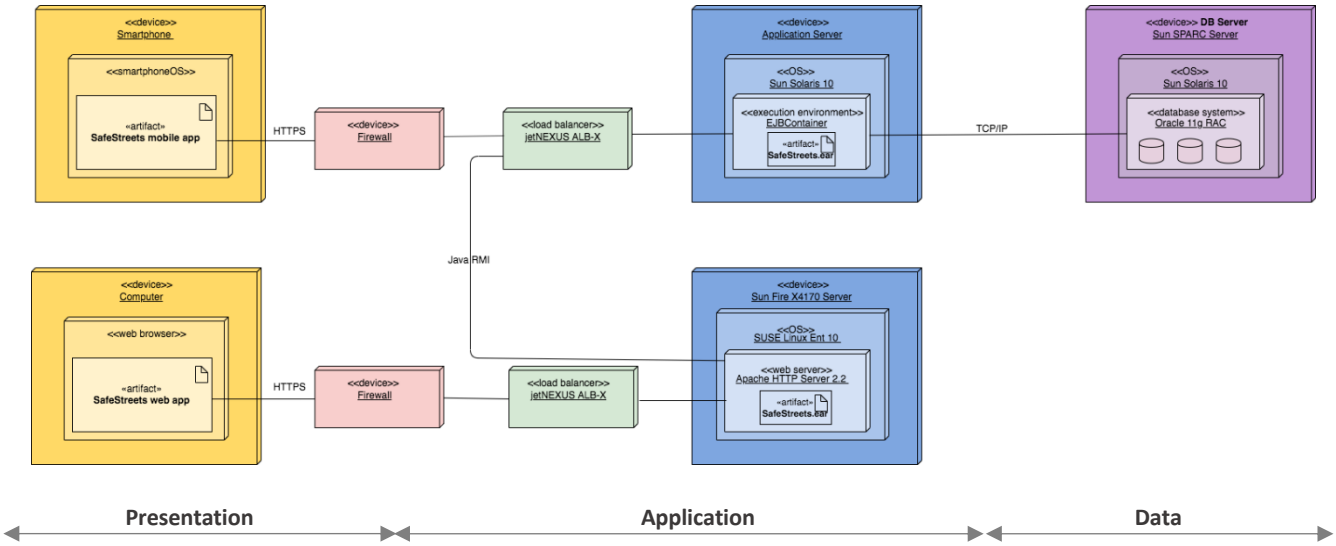


Figure 6 – Deployment Diagram

In the Deployment diagram in the figure are shown the most important components. A better explanation of these ones follows here:

- **Smartphone:** The user can use the smartphone to report a violation or to mine information sending the information directly to the Application Server using HTTPS.

- **Computer:** Can be used both from the user or the municipality. The user can only perform the action of mining information or see the registry of reports done. While the municipality, after having done the login, can access the information of its competence like control each violation or receive violations from SafeStreets and use the information to generate traffic tickets.

- **Firewall:** Provides safety access to the internal network of the system as part of the safety of the system against external attacks.

- **Load balancer:** Distributes the workloads across multiple servers to increase capacity (concurrent users) and reliability of applications trying to avoid the overload of any single server.

- **Web Server:** Receives contents and requests from clients through web app and sends data received, using Java RMI, to the Application Server for processing. Moreover, it is replicated to avoid a single point of failure and to guarantee a better performance.

- **Application Server:** Here we have the application logic, but not all because also the client has a part of this. The Application Server handles all the requests and provides the appropriate answers for all the offered services. It is directly addressed by the mobile application and handles also some requests that are forwarded by the Web Server and sent by the web application. In addition, it is replicated just like the Web Server for the same reasons.

- **Database Management System (DBMS)**: Oracle Real Application Clusters (RAC) has been selected to perform the job of the DBMS. A cluster comprises multiple interconnected computers or servers that appear as if they are one server to the end users and applications. Oracle RAC enables you to cluster Oracle databases and it is a configuration of Relation DBMS (RDBMS).
  There are some nodes that contain the same instance of the database and the instances are connected to each other and share the cache in the Global Cache (GC). Then, there is a pool of replicated databases that contain datafiles. Datafiles are common, shared and they are accessed from all instances in a parallel and synchrony way.
  The choice of using Oracle RAC is due to provide performance, scalability, resilience and high availability of data at instance level. Performance comes from the load balancing because multiple machines are holding Oracle software, so it creates a better performance and the load is shared between all these servers. Scalability is due to the fact that it is possible to add more machines to hold Oracle software thus you are adding more nodes to the existing cluster. Resilience is provided: if one of the machines fails, that one could be taken down without bringing down the whole environment. Then the application will still be running thanks to the communication with other replicas. So, one machine could be brought down and fixed while others are running, hence this also brings to another important point that is the maintenance.
  In addition, to provide a higher availability we combine the RAC architecture with that of Data Guard that offers the functionality of Disaster Recovery. Data Guard has a working instance (called Primary) and one or more sleeping instances (called Standby) that are continuously updated and maintained aligned to the Primary instance in order to take his place in case of fail.

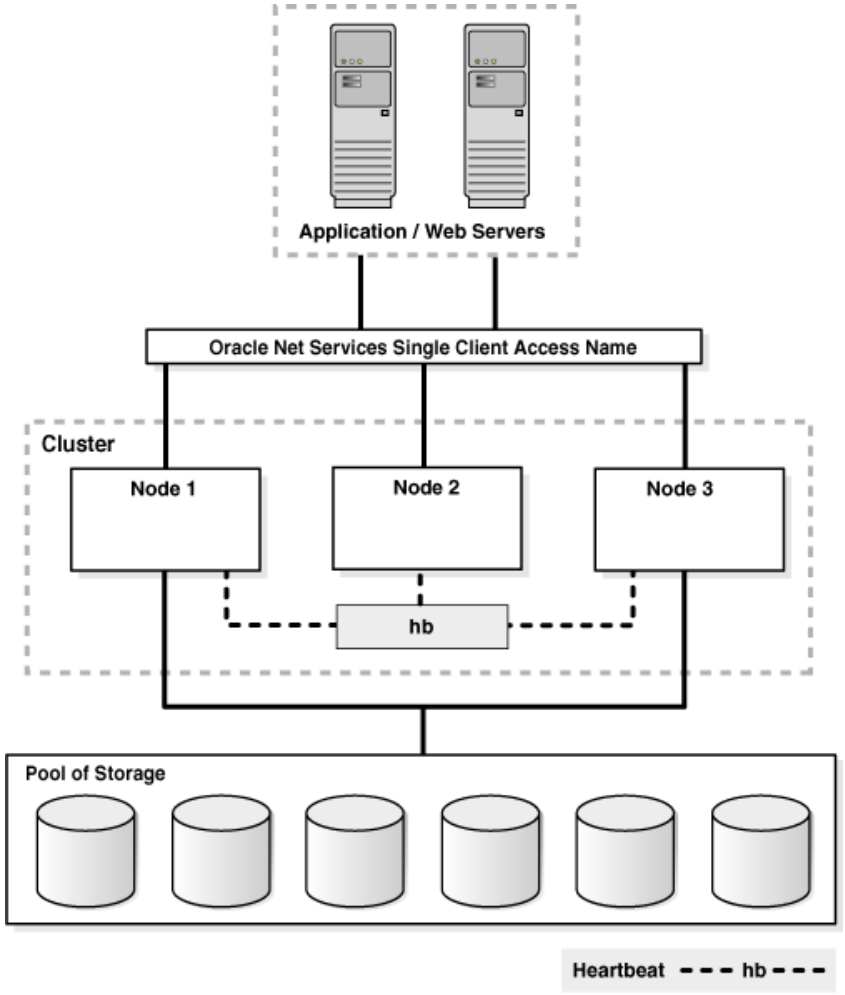
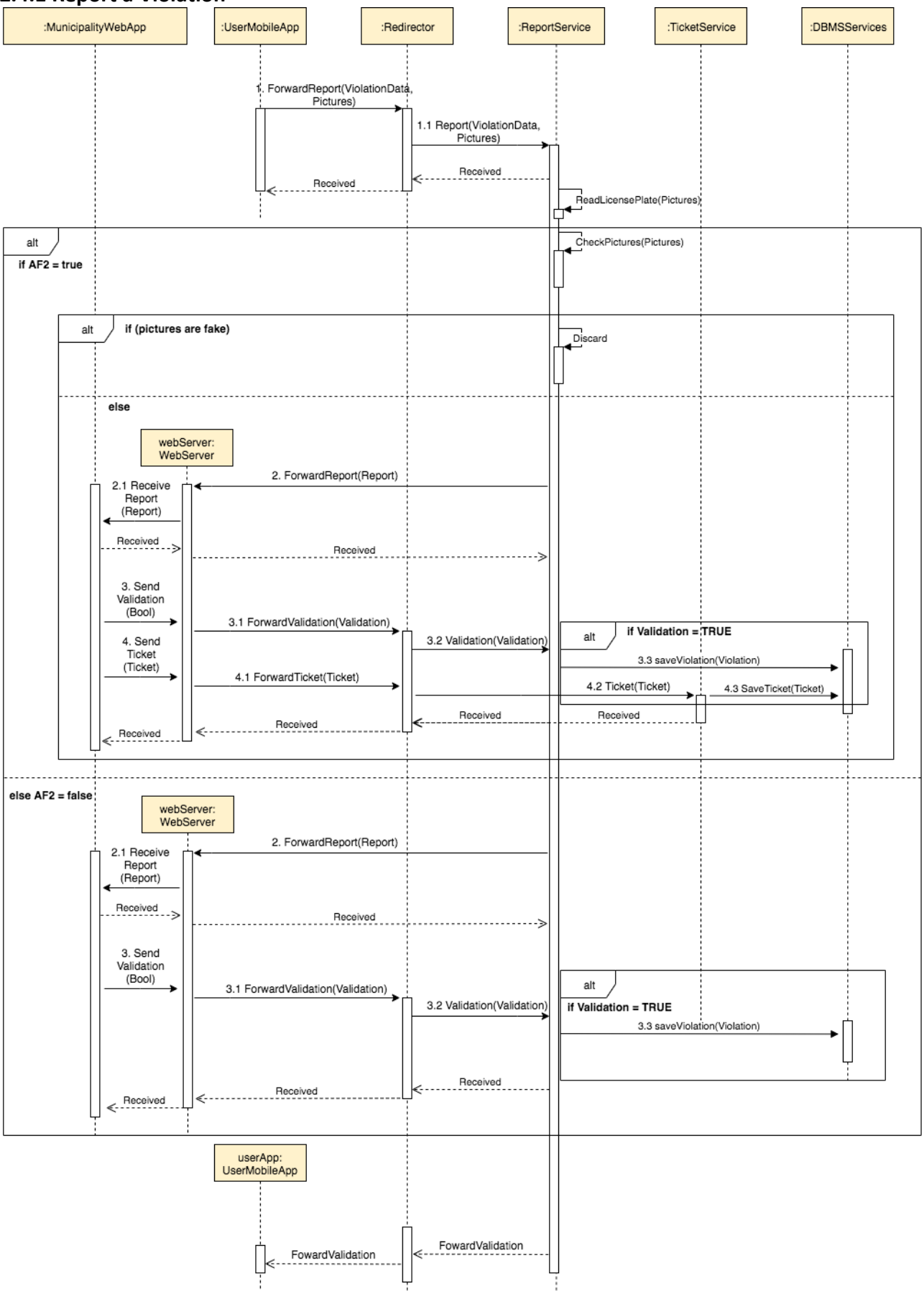


Figure 7 – Oracle Real Application Clusters

2.4 Runtime View

2.4.1 Report a Violation



In this sequence diagram is described the sending process of a report by a user.

For the sake of simplicity, the entity UserMobileApp has been divided, but it is always active, in order to guarantee SafeStreets application to track the user in the session.

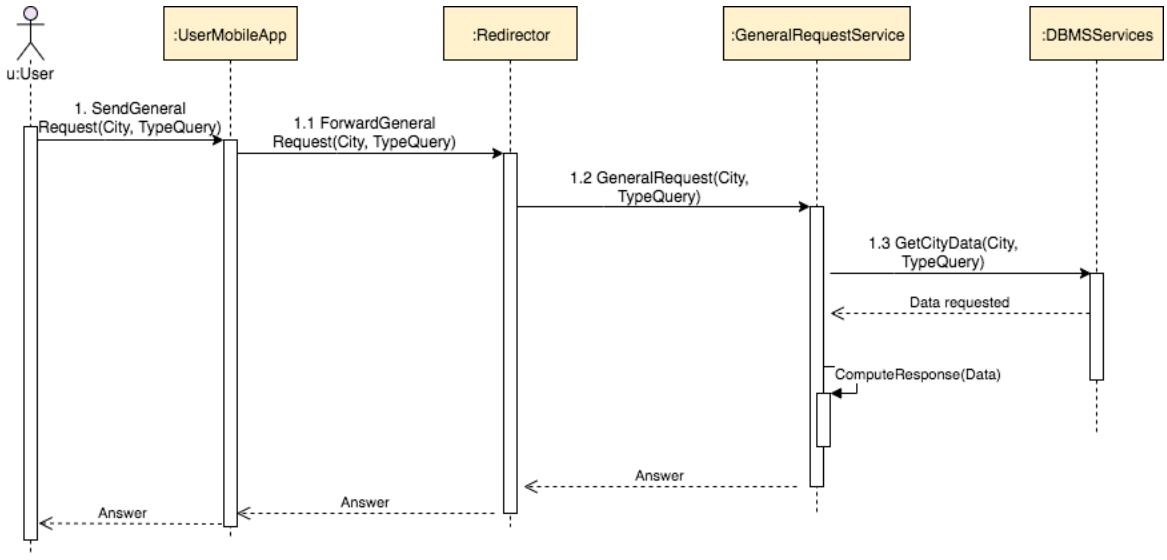
The user app sends the report compiled to the redirector, which forwards it to the correct component, ReportService. If the nearest municipality joins advanced function 2 the report is checked and, if it is not fake, it is sent to the municipality web app, through the corresponding web server. It is sent to the municipality even if is not subscribed to the AF2.

This step is possible thanks to the interface which allows the invocation of methods (ForwardReport) directly from the WebApp of the municipality without passing through the Redirector.

Then, the municipality web app sends to the system the corresponding validation of the report received, this validation is given to the redirector, which forwards it to the ReportService. If the report is valid for the municipality, the corresponding violation is stored in the database using DBMSServices.
If the municipality join AF2, it includes in the messages the ticket generated, which is also stored using the DBMSServices.

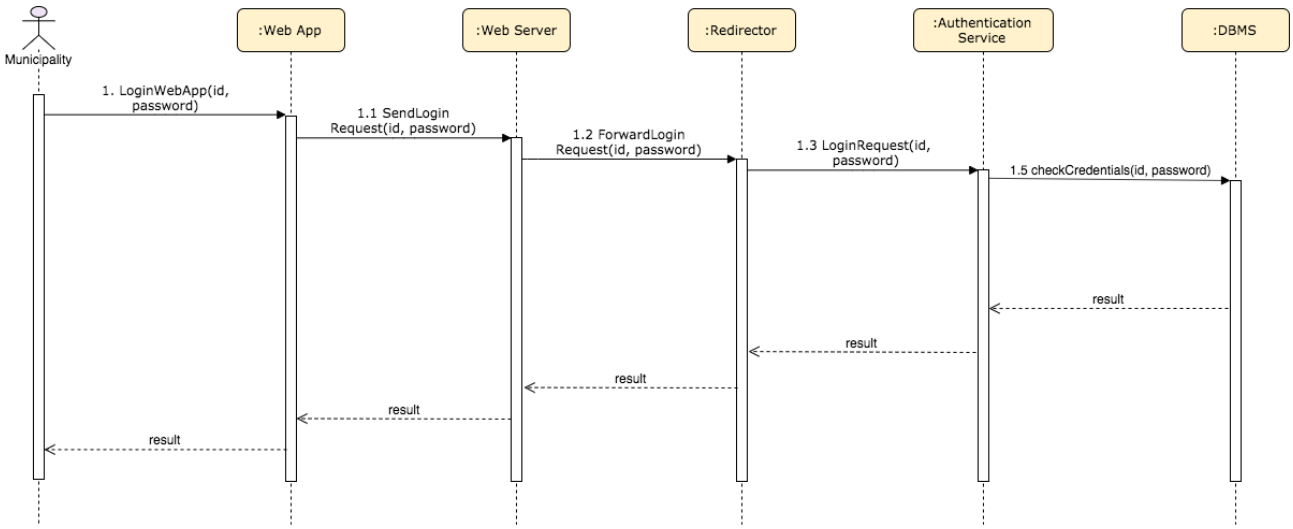
Finally, the user app is notified about the validation of its report.
For the sake of simplicity, actors such as User and Municipality are omitted and the process starts directly from the MobileApp and WebApp components respectively.

2.4.2 Make a General Request



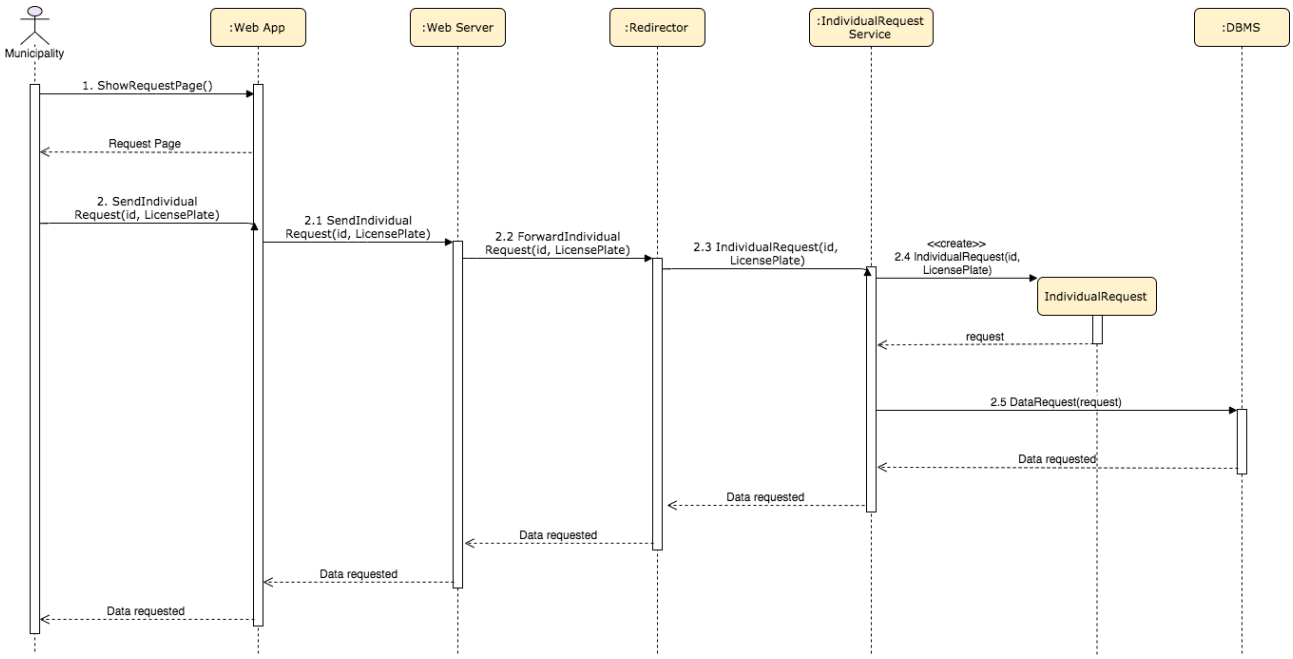
In this sequence diagram, a user sends a general request about a city using the mobile app. The request is given to the redirector which forwards it to the correct component, GeneralRequest. This component communicates with DBMSServices, which provides the corresponding data. These data are computed by the GeneralRequest interface and forwarded to the user app, through the redirector.

2.4.3 Municipality Login on Web App



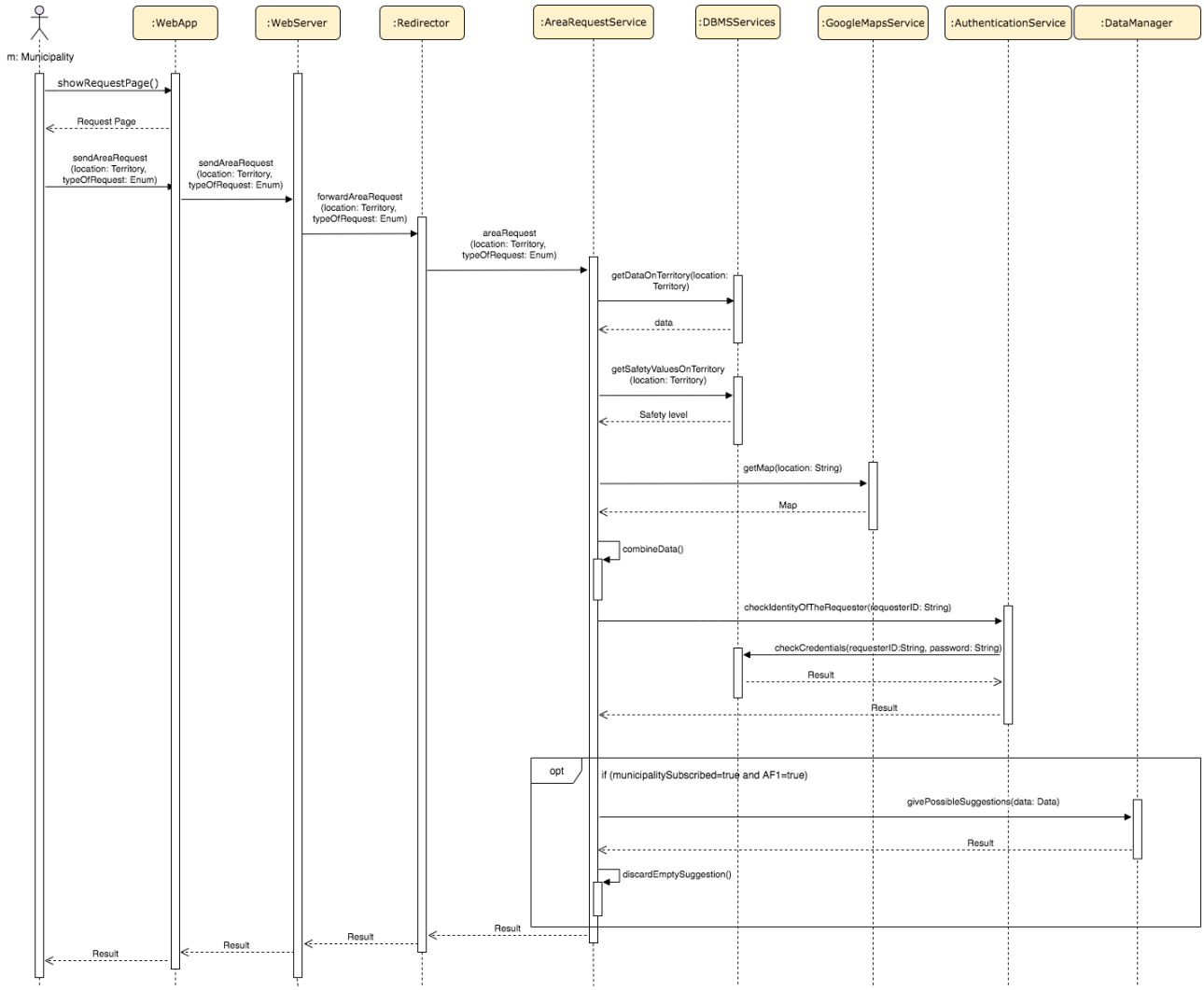
In this sequence diagram it is shown the process of login attended by the Municipality. The Municipality, using his own computer, does the login filling the form with his personal id and password. Then, the login request is sent from the Web App to the Web Server. After, the last one, thanks to the Redirector, will forward the login request to the Authentication Service. Here the Authentication Service controls if the id of the Municipality exists in the database and if the password matches. So, the result of this operation is propagated back to the Municipality, following the same route as before, and if the result is positive the login is successful, otherwise should be repeated.

2.4.4 Make an Individual Request



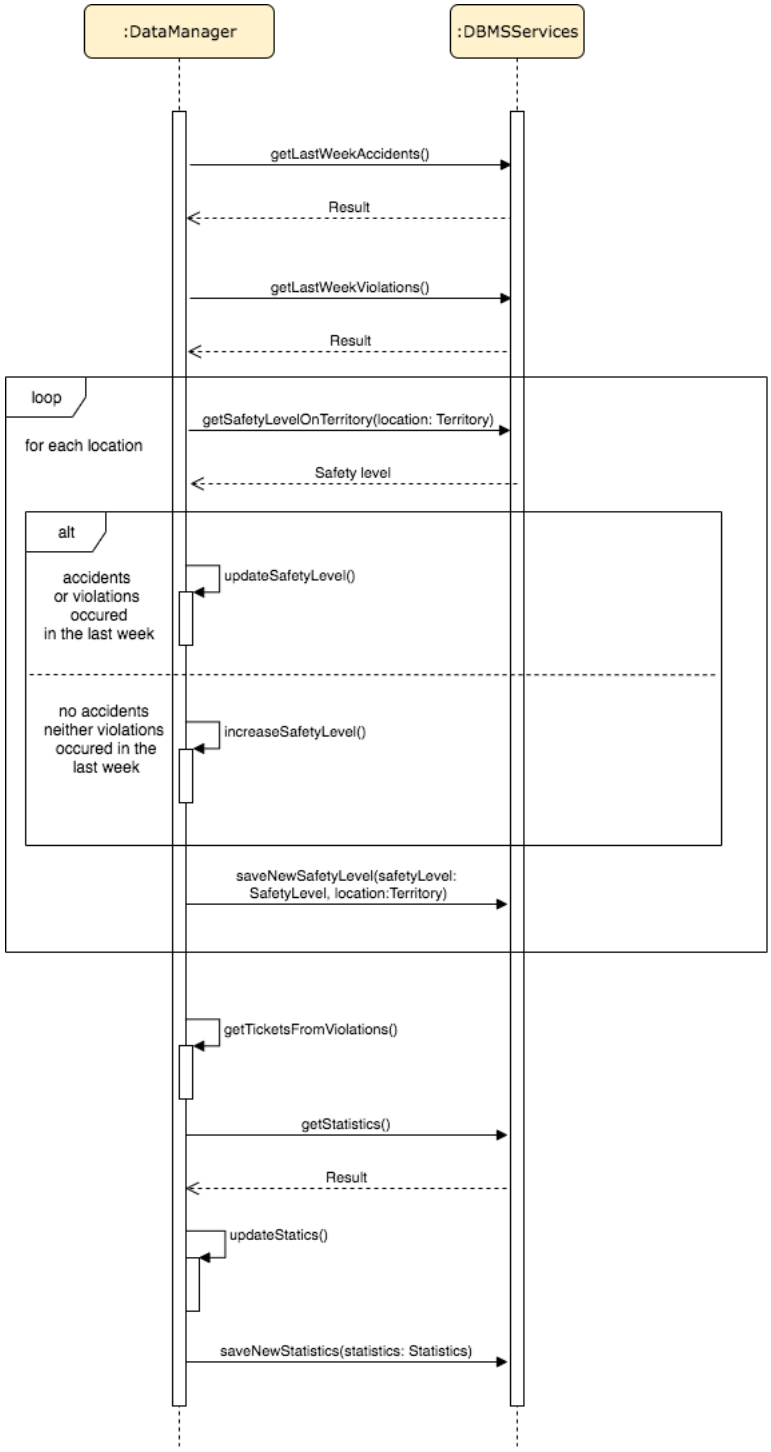
In this sequence diagram it is illustrated the process of making an individual request attended by the Municipality. An assumption is made: Municipality is already logged in. The first action taken from the actor is to ask the request page to the Web App. The last one can reply directly showing the requested page. Then the actor performs the individual request attaching to it the id of the Municipality and the license plate to which they are interested. The Web App asks to the Web Server that in turn sends it to the Redirector. The Redirector forwards the request to the proper component, thus, in this case, it forwards the request to the IndividualRequestService. The latter creates a request based on the given parameters and then, using the DBMS, the query with that specific license plate is performed. The result of this query, that is the violations committed by the offender with that license plate, is sent back to the Municipality that has made the query.

2.4.5 Make an Area Request



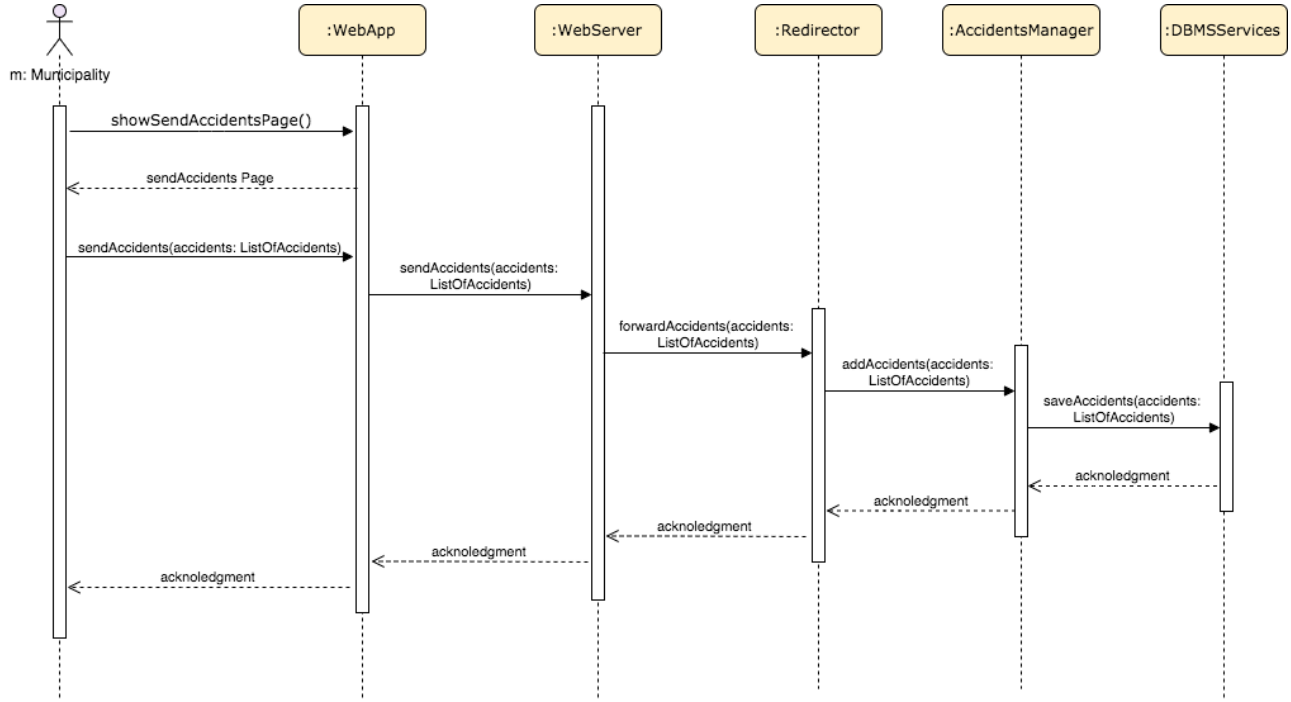
The “Make an Area Request” sequence diagram shows the interactions between components when the municipality asks for information about safety levels in a specific area. This request is forwarded from the redirector to the RequestManager and in particular to the AreaRequestService. Given the area of interest specified as parameter, the AreaRequestService, making a call to the DBMS, gets all the necessary information including the accidents and the violations occurred in that location.
It then gets the representation of the map through the GoogleMapsService and decorates the map painting the different parts according to their safety level and adding dots where accidents occurred: this is all done through the internal method combineData().
After that, AreaRequestService checks the identity of the requestor to see if it is a municipality subscribed to the advanced function 1: in this case a method is called on DataManager to look for a possible suggestion matching the data related to that specific territory. If a feasible intervention is found, it is piggybacked on the created map and everything is sent back to the municipality.

2.4.6 Build Statistics



As already mentioned in the document, every week the application starts the procedure to update the safety level of different areas and the general statistics stored by the system. The two only components interacting are the DataManager and the DBMS. At the beginning the DataManager gets all the useful information about accidents and violations occurred in the last week. In a second stage it loops on the different locations stored in the database updating in a proper way the safety level of each one that presented some infractions or violations and increasing the safety level of the ones that were safe in the previous week: everything is stored again on the database through calls to the DBMS.
Then, after extracting also information about tickets related to the last violations, statistics are modified adding the new values and the updated version is stored in the database.

2.4.7 Send Accidents



The previous sequence diagram is clearly explicative and shows the different steps involved in the process of sending information about new accidents from the municipality to the system. Through the traced path the notification reaches the AccidentsManager that makes a call to the DBMS to store the information in the database.

2.5 Component Interfaces

In the next diagram are described the main methods which can be invoked on the interfaces and their interactions, referring to the most important processes reported in the runtime view section.

One aspect is fundamental to be pointed out: in general, methods written in the Component Interfaces diagram are not to be intended exactly as the methods that the developers will write, but they are a logical representation of what component interfaces have to offer. They will be adapted facing the various aspects that will come out during the implementation of the code.

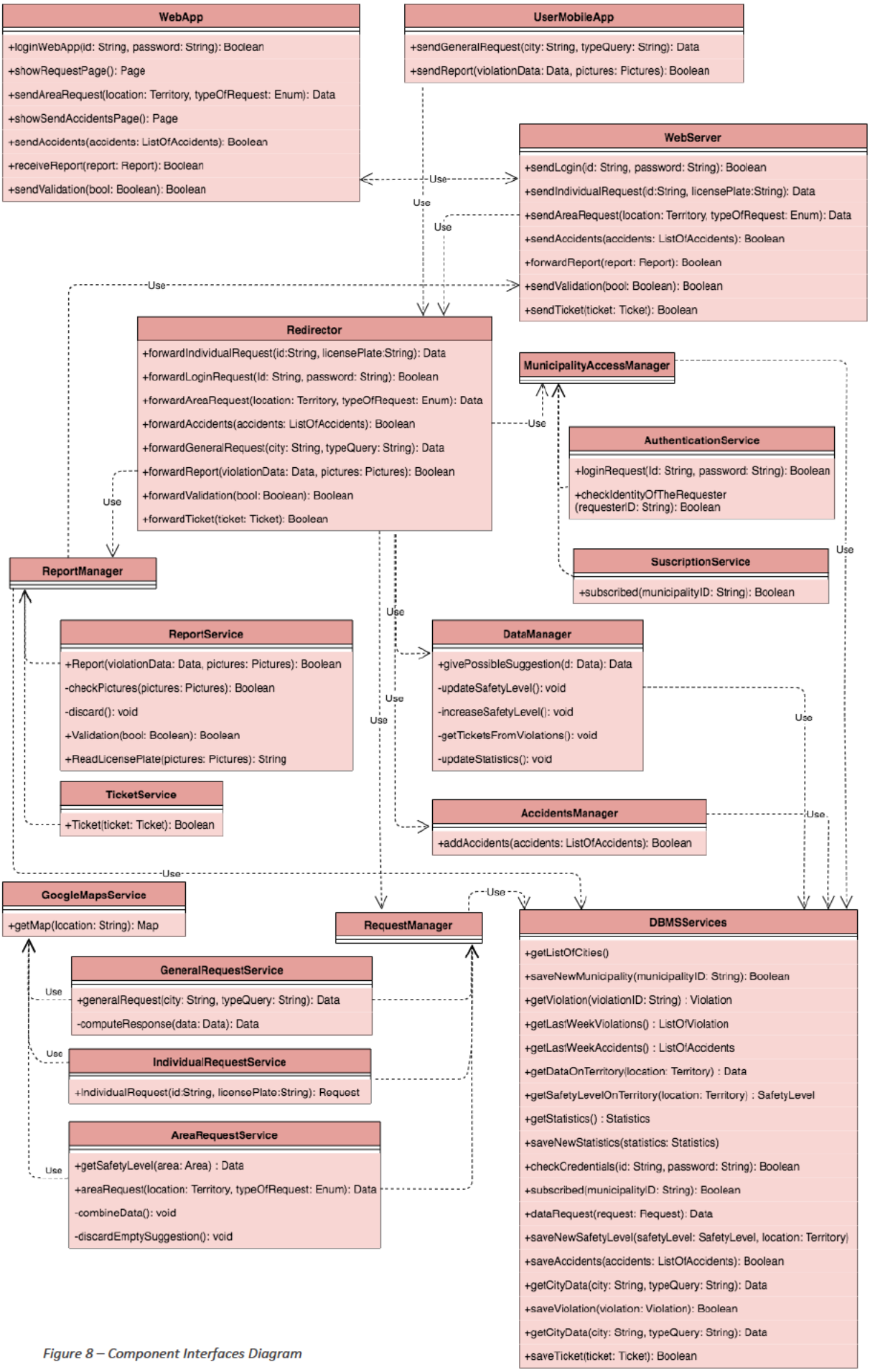


Figure 8 – Component Interfaces Diagram

2.6 Selected Architectural Style and Patterns

As partly described in the overview, a three-tier client-server architecture has been used to develop the system. This choice promotes a major decoupling of the system, increasing the reusability, scalability and flexibility. Furthermore, components in the application server have been thought to be cohesive and with low coupling among modules to make the system more comprehensible and modifiable.
In this particular case however, the system does not embody a pure client-server paradigm. This is clear considering the interaction between the WebServer and the ReportManager: when the application needs to send reports to the municipalities, it performs a call on the WebServer (forwarded to the WebApp) which swaps the role of client and server. This interaction expresses the inclination towards a data-centered model.

The communication protocol used to exchange messages is HTTP. This is also endorsed by the importance that data plays in the system. In this way it is possible to reduce the coupling among client and server components. When dealing with sensitive data, TSL is used to guarantee the security and the reliability of the connection.

Furthermore, as already said the system uses the cache on the client side: this allows to avoid some interactions between the client and the server, speeding up the communication. The use of cache is common in case of fat clients.

Regarding the format in which the data are transmitted, JSON is used of its simplicity. It is less verbose and this makes it more readable and allows a much faster parsing. The JSON file is transmitted over the network via XmlhttpRequest.

When dealing with GoogleMaps, the communication protocol used is REST: GoogleMaps is indeed used as an external service and this protocol clearly suits this situation. Furthermore, the public REST API provided by GoogleMaps is used: it also gives the possibility to customize the requested contents.

With respect to the database server, instead, the application server acts as a client making queries and waiting for response.

To perform the application the following pattern are used:

Model View Controller (MVC)

It has been selected to adopt Model-View-Controller (MVC) in order to guarantee the maintainability and the reusability of the code. This software pattern is particularly adapted for the development of both web and mobile application in an object-oriented style of programming as java.
MVC separates the application into multiple layers of functionalities:

- Model: responsible for managing the data of the application and for receiving user input from the controller
- View: responsible for presents data to the users
- Controller: responsible for connecting the user input and performs interactions on the data model objects that belongs to the model. The controller receives the input, optionally validates it and then passes the input to the model

Thanks to this it is possible to create components independently of each other and simultaneous development is simplified.

Facade pattern

This pattern is used from the redirector to provide an interface to the client using which the client can access the system. A facade is an object that serves as a front-facing interface masking more complex underlying or structural code. In addition, in this pattern the client interacts with a simpler interface instead of an intricate one thanks to the hiding of complexity of the larger system.

2.7 Other Design Decisions

In this last paragraph it is specified further what kind of database has been selected to store all the data of SafeStreets. As mentioned in the 'Deployment View' paragraph, Relational database is chosen to perform this role and in particular the Oracle RAC database.
The choice of a relational database is due to the fact that is possible to easily categorize and store data that can later be queried and filtered to extract specific information for reports. Relational databases are also easy to extend and aren't reliant on physical organization. After the original database creation, a new data category can be added without all existing applications being modified.

In addition, other advantages of this kind of database are:

- Accuracy: Data is stored just once, eliminating data deduplication.
- Flexibility: Complex queries are easy for users to carry out.
- Collaboration: Multiple users can access the same database.
- Trust: Relational database models are mature and well-understood.
- Security: Data in tables within a RDBMS can be limited to allow access by only particular users.

Moreover, it is adopted an Oracle RAC DBMS about which there is a better explanation in the chapter 2.3. The following lines are intended as a short recap of what is RAC and why it has been chosen.
Oracle Real Application Clusters (RAC) allows multiple instances running on different servers to access the same physical database stored on shared storage. The database spans multiple hardware systems and yet appears as a single unified database to the application. This enables the utilization of commodity hardware to reduce total cost of ownership and to provide a scalable computing environment that supports various application workloads. If additional computing capacity is needed, it is possible to add additional nodes instead of replacing their existing servers. Finally, this architecture also provides high availability as RAC instances running on different nodes provides protection from a server failure.

2.8 Algorithms

2.8.1 Algorithm on metadata

```
BEGIN
      Report receivedReport = SafeStreets.getNewReport();
      Picture[] pictures = receivedReport.getPictures();

(1)   Timestamp firstPicture =
      pictures.stream().min(Comparator.comparing(DateAndTime)).get();
      Timestamp lastPicture =
      pictures.stream().max(Comparator.comparing(DateAndTime)).get();

      if (lastPicture.getDelayFrom(firstPicture) > admittedSkewAmongPictures){
            discardReport();
            return False;
      }
(2)   Timestamp reportCreationTime = report.getTimeOfCreation();
       float skew = firstPicture.getDelayFrom(reportCreationTime);
       if (skew > adimittedSkewFromFirstPicture) {
            discardReport();
            return False;
       }

(3)   float distance;
      float maxDistance = 0;
      Location[] locations = pictures.stream().map(p->
      p.getGPSPosition()).collect(Collector.toArray());
      for (Location loc1: locations) {
            for (Location loc2: locations) {
                  distance = loc1.getDistance(loc2);
                        if (distance > maxDistance) {
                              maxDistance = distance;
                        }
            }
      }
      if (maxDistance > maxDistanceAllowed) {
            discardReport();
            return False;
      }

(4)   ExactPosition statedPosition = report.getPlaceOfViolation();
      for (Location loc: locations) {
            if (statedPosition.getDistance(loc) < maxDistanceAdmitted) {
                  sendReportToMunicipality(receivedReport);
                  return True;
            }
      }
      discardReport();
      return False;
END
```

The previous algorithm describes, using a Java-like pseudocode, the set of controls performed to increase the reliability of the received report.

- On the first step a check is made on the time in which the pictures are taken. As already said, when pictures are shot, the time and data are inherently incorporated in the details of the picture: what the **ReportManager** does is to extract these data, find the first and the last pictures according to the timestamps and compare them. If they present a considerable difference in time, which is greater than the admitted one (*admittedSkewAmongPictures*), it means that the report is untrustworthy and is discarded.

- The second step is still focused on time consistency and checks the difference between the timestamp of the first picture shot and the time of creation of the report (*reportCreationTime*) that indicates the moment when the user notices the violation. Again, if the difference exceeds the *adimittedSkewFromFirstPicture*, the report is considered suspicious and is discarded.

- The third step takes into account all the different *GPSPositions* inherently incorporated in the pictures' info. If the *GPSPosition* of at least two pictures exceed the maximum distance allowed between them, the report is discarded.

- The fourth step instead makes a comparison between the *statedPosition*, inserted by the user and all the *GPSPositions* extracted from the pictures. If the distance with at least one *GPSPosition* is lower than the upper bound *maxDistanceAdmitted*, the report is accepted: this proves that the stated position is quite close to the position detected through the GPS. This is done to cope with the inaccurate measures of the GPS.

The algorithm presented in this section has constant upper bound in terms of computation time thanks to the limited number of pictures that can be inserted in a report: in the worst case the reports will have 10 pictures and the third step, which is the heaviest in computational terms, will execute at most 100 times the instructions within the two "for" loops.

2.8.2 Algorithm on license plate

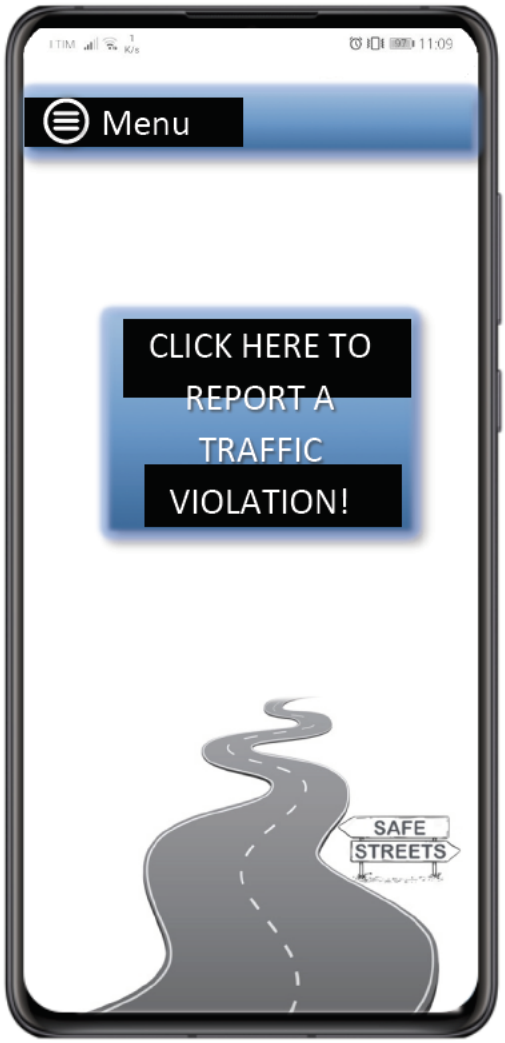
The application takes advantage of an algorithm which manages to identify the alphanumeric code in a picture. It uses artificial intelligence to match the shapes of the characters with the correspondent letters and numbers: this mapping is helped also through the detection of a white background in different tonalities that recalls the background of a typical license plate.

3. User Interface Design

3.1 Mockups

The following mockups show how it should be the look of the mobile application used by a user (3.1.1 to 3.1.8) and the aspect of the web application used by municipality (3.1.9 to 3.1.11).

3.1.1 Home

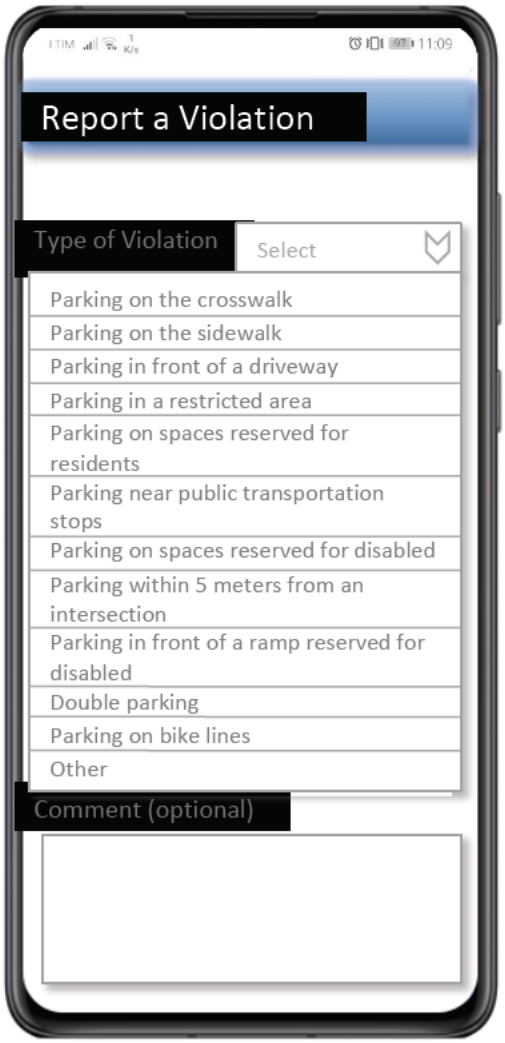


3.1.2 Main Menu

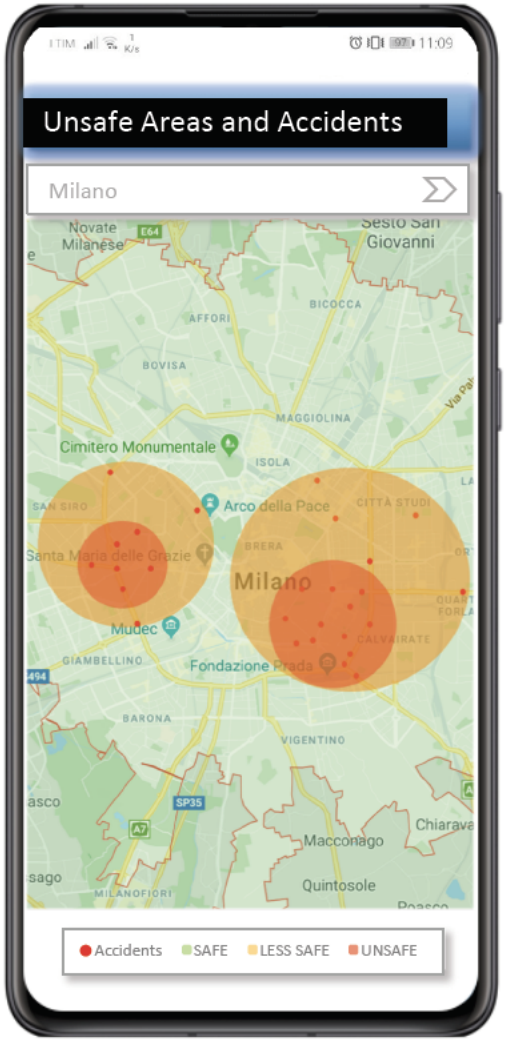
3.1.3 Second Menu (Extract Information)

3.1.4 Report a Violation

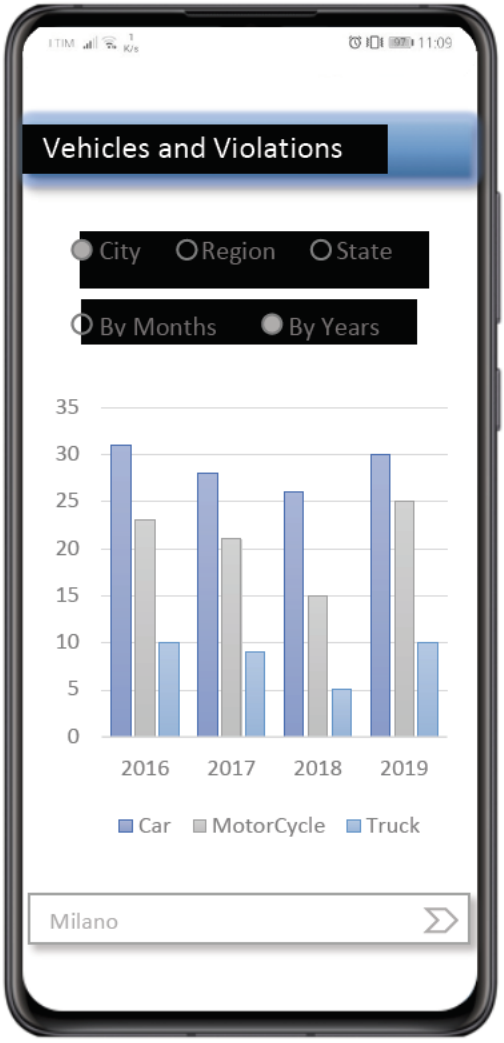
3.1.5 Report of Violation (Violation Type List)



3.1.6 Map of Unsafe Areas and Accidents

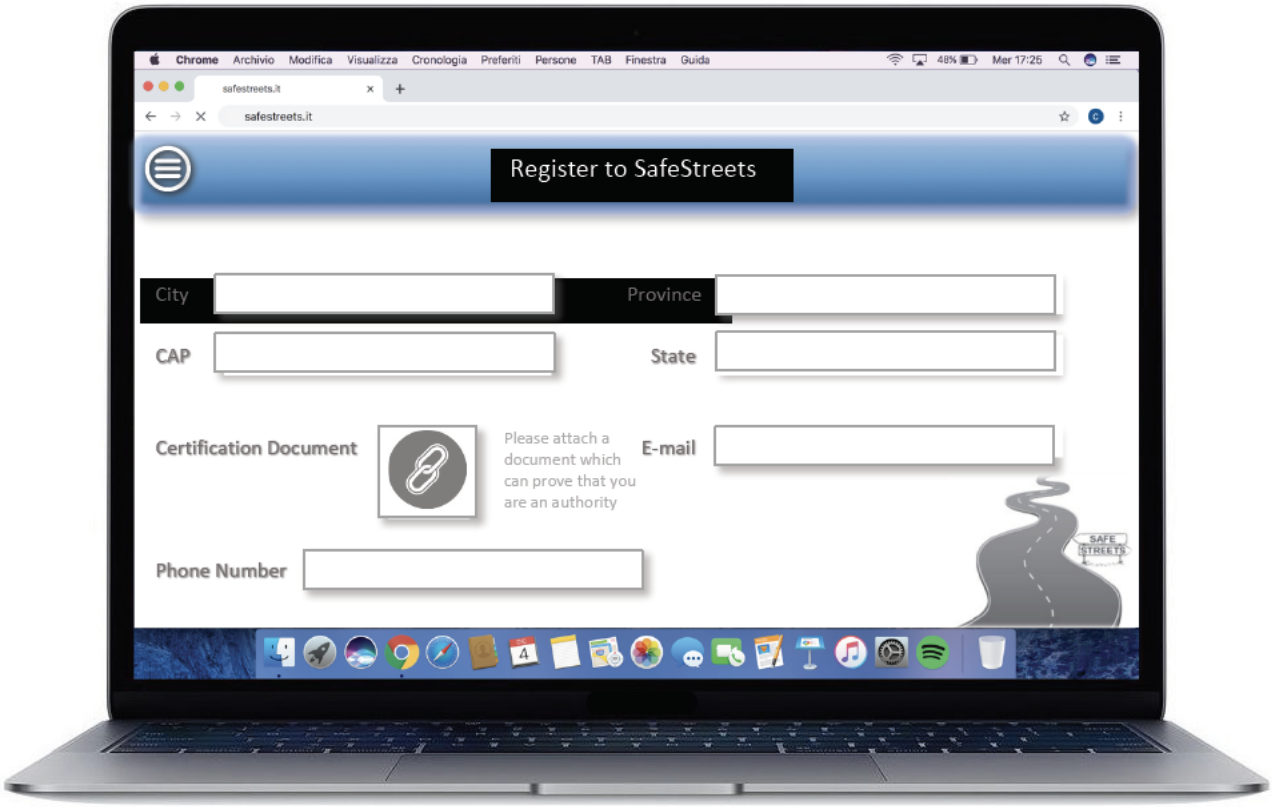


3.1.7 Graphic of Vehicles and Violations

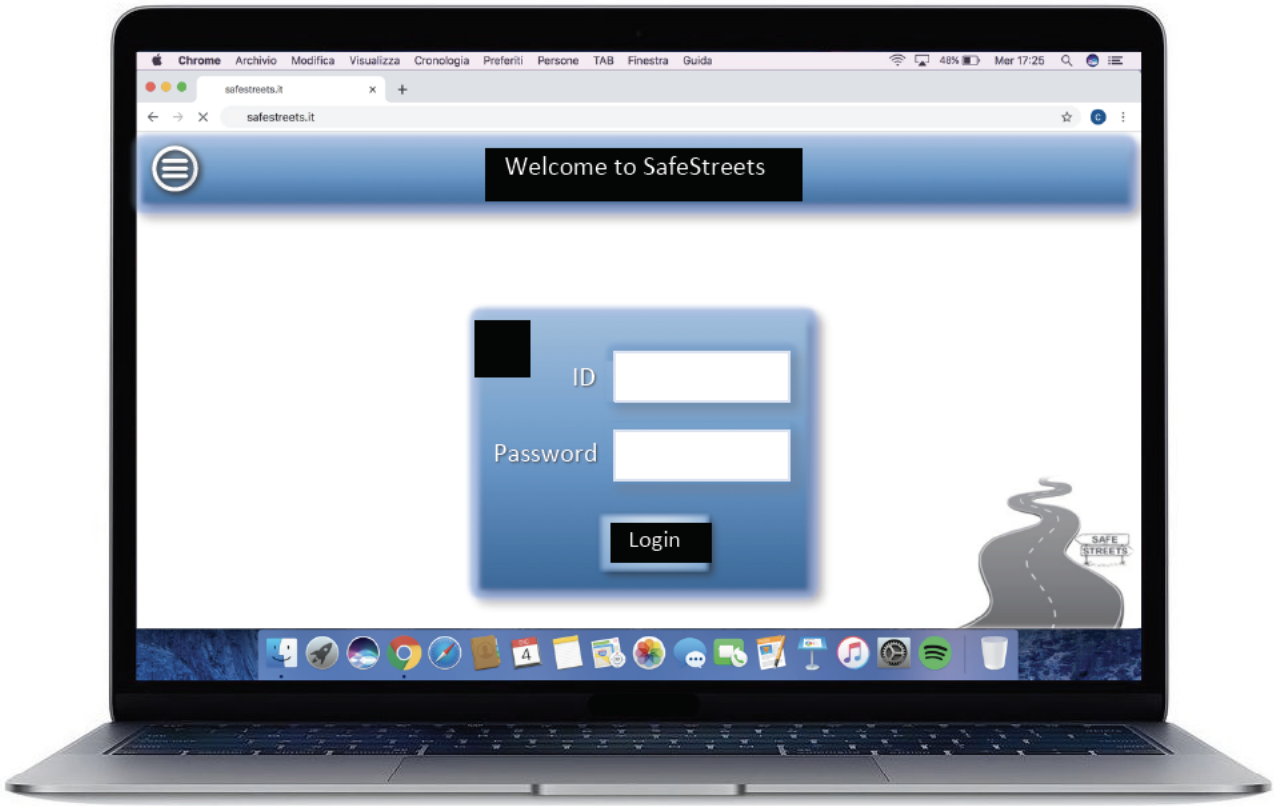


3.1.8 Graphic of Trend of Accidents and Violations

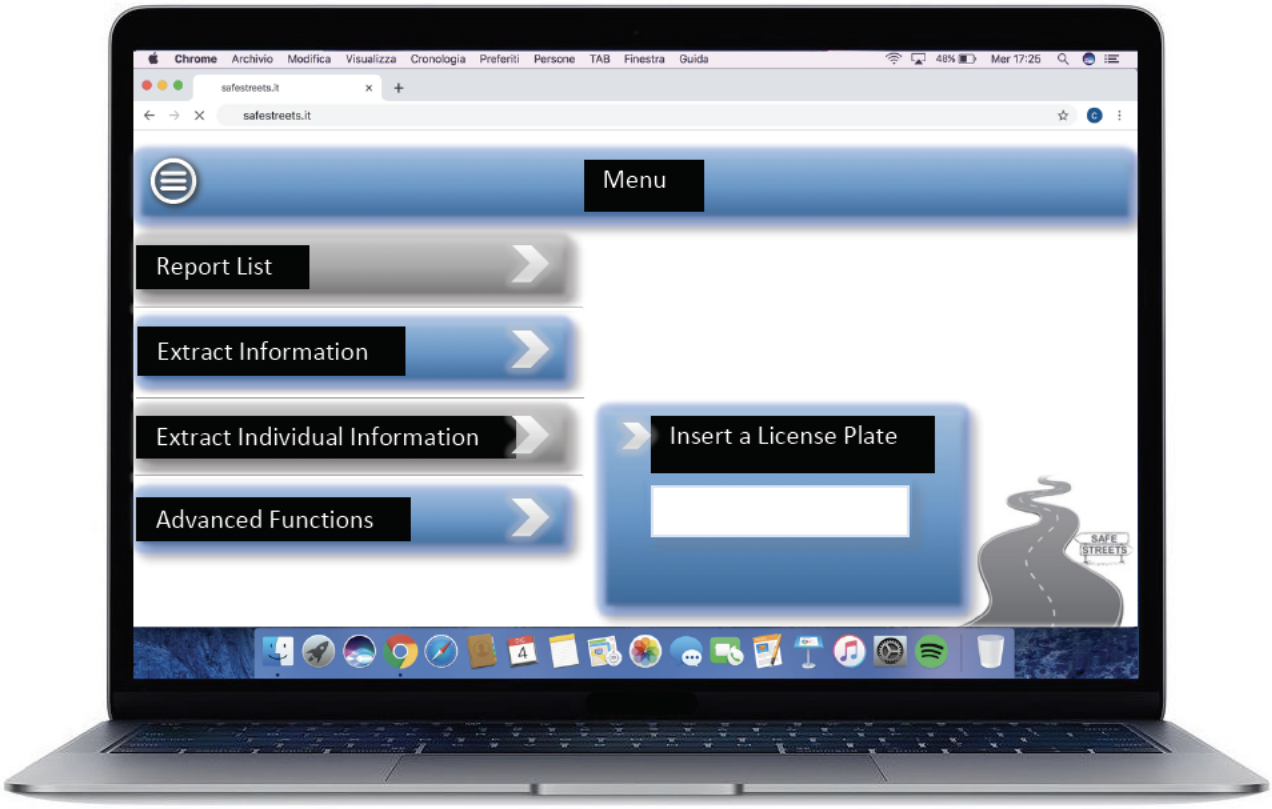
3.1.9 Municipality Registration Form



3.1.10 Municipality Login



3.1.11 Municipality Menu with Extract Individual Information Extension



4. Requirements Traceability

This section binds the goals specified in RASD with design components.

G1	Allow users to notify authorities about traffic violation
	Requirements: **R1, R2, R3, R4, R5, R6, R6.1, R7, R9, R10, R12, R17, R19, R20, R22**
	Components: • UserMobileApp • ReportManager [ReportService] • DBMSServices
G2	Allow users to send pictures
	Requirements: **R8, R25, R26**
	Components: • UserMobileApp • ReportManager [ReportService] • DBMSServices
G3	Allow users to insert the type of violation
	Requirements: **R18**
	Components: • UserMobileApp • ReportManager [ReportService] • DBMSServices
G4	Allow users to insert geographical position in two different way
	Requirements: **R8**
	Components: • UserMobileApp • ReportManager [ReportService] • DBMSServices

G5	Allow users to write additional information about violation
	Requirements: **R20**
	Components: • UserMobileApp • ReportManager [ReportService] • DBMSServices
G6	Users and authorities can mine information of violations in an overview
	Requirements: **R10, R13, R15, R24**
	Components: • UserMobileApp • WebApp • WebServer • MunicipalityAccessManager • RequestManager [GeneralRequestService] • DataManager • DBMSServices
G7	Authorities can mine information about individual violations
	Requirements: **R2, R3, R10, R14, R15**
	Components: • WebApp • WebServer • MunicipalityAccessManager • RequestManager [IndividualRequestService] • DBMSServices

G8	Identify potentially unsafe areas
	Requirements: **R6, R11, R15, R24, R29**
	Components: • UserMobileApp • WebApp • WebServer • MunicipalityAccessManager • DataManager • AccidentsManager • RequestManager [AreaRequestService] • GoogleMapsService • DBMSServices
G9	Suggest municipality for possible interventions to secure an area
	Requirements: **R2, R3, R6, R27**
	Components: • WebApp • WebServer • MunicipalityAccessManager • DataManager • AccidentsManager • GoogleMapsService • DBMSServices
G10	Send key information to the municipality to generates traffic tickets automatically
	Requirements: **R2, R3, R4, R5, R6.1, R7, R10, R11, R16, R17, R28, R29**
	Components: • WebApp • WebServer • MunicipalityAccessManager • ReportManager [ReportService] • ReportManager [TicketService] • DBMSServices
G11	Information collected in the database can be used to build statistics
	Requirements: **R11, R29, R30**
	Components: • DataManager • DBMSServices

G12	**Allow users to see the reports sent or incomplete**
	Requirements: R21, R22, R23
	Components: • **UserMobileApp** • **DBMSServices**

R1	Users are anonymous
R2	Authorities are certified with an authentication
R3	Authorities should register to the application filling the form with mandatory fields
R4	Only authorities can receive report about violations from the system
R5	Each report will be forwarded to the related municipality only if it is subscribed, otherwise it will be sent to the closest municipality enrolled to the service
R6	Authorities can choose to adhere or not to the advanced function 1
R6.1	Authorities can choose to adhere or not to the advanced function 2
R7	After the authority subscription, reports of interest of its related territory are forwarded to it
R8	Users shall authorise the system to access the local camera and the local position
R9	Municipality is requested to send to the system validations of received reports
R10	Municipality that uses the system needs to be logged
R11	Municipality allows the system to acquire its data about accidents and issued tickets
R12	After a report is validated positively from municipality its data are stored in the database system
R13	Users cannot access personal data of offenders through any queries
R14	Authorities have no restrictions to access personal data
R15	Authorities and users have to choose from a list of predetermined requests to extract information by queries
R16	The system provides secure reports to municipalities that join AF2
R17	The system sends to the authorities reports as soon as they arrive
R18	Users can choose the type of violation from a predefined checklist
R19	Users should compile a report in all its mandatory fields
R20	Users can add an optional comment to the report
R21	Users can access a queue that contains incomplete reports and the ones completed in absence of an internet connection
R22	Completed reports in the queue are sent to the system as soon as the connection is available
R23	Users can view in a registry the reports already sent and the result of the validation from the municipality
R24	Users can select from a list of city maps to view the level of safety of specific areas characterized with different colours
R25	The system allows users to take pictures only directly from the application
R26	Pictures that are taken collect automatically the date, the time and the GPS position
R27	The system suggests possible solutions to municipalities that join the AF1 to make their territory safer
R28	The system ensures that the pictures sent to the municipality joining the AF2 are not modified
R29	Data of municipalities enrolled in AFs are collected by the system
R30	Statistics are built considering data stored in the database

5. Implementation, Integration and Test Plan

5.1 Overview

This last part is very important because it is almost impossible to develop error free software, so the phase of verification and validation takes an important role. Program testing can be used to show the presence of bugs, but never to show their absence. Thus, for this reason, the aim is to find as many bugs as possible until the release date of the application.

5.2 Implementation Plan

The entire system, with its relative sub-systems, has to be implemented, but also tested and integrated exploiting a **bottom-up** approach. This approach is chosen both for the server side and the client side, that will be implemented and tested in parallel. Using bottom-up, the system could be done in an incremental way so that also the testing can proceed in parallel with the implementation. This is fundamental given the fact that an incremental integration facilitates bug tracking and it turns out to be simpler to build a limited number of components and to integrate them incrementally. It is important to underline that the bottom-up approach will be used as well as for the implementation and testing of the single services that constitute the various components.

The implementation has to be done from the lower components up to the top because in this approach the implementation is gradual. There are some components that rely on some others so a priority among the components is present. For example, every component and sub-system described in the Component diagram, in the section 2.2, interact with the DBMS (someone directly, someone else not).
The first step can be implementing the **DBMSServices**, which is the component implementing all methods that allow to access to the Database and perform queries and updates on it. It has a very important role thanks to the extraction of information from the Database and the possibility to write it on the Database, making it persistent.

After that we can proceed to the implementation of the other components and subsystems. Clearly the two main components are the **ReportManager** and the **RequestManager**: the first manages all that concerns the reports (so both sends reports and traffic tickets), the second provides the information requested by both the user and the Municipality (can be an Area request, an Individual request or a General request). But the latter depends on the implementation of other components so, if we want to keep using the bottom-up approach while testing, it is not possible to do it right away and so we have to implement first the **MunicipalityAccessManager** and the **DataManager** that are the ones on which it relies.

Then, for these reasons it is possible to implement one of the three components: **AccidentsManager**, MunicipalityAccessManager, DataManager. Among these, the implementation of the AccidentsManager can be done quite independently of the others even though it could be better to implement it before so as it could be possible to implement and test in a second moment the functionalities of the DataManager that extracts all useful data already inserted (also by the AccidentsManager) into the database.

So, at the same level, both the AccidentsManager and the MunicipalityAccessManager could be implemented. The MunicipalityAccessManager offers the possibility for a municipality to sign up to the basic service or to the advanced functions and to be authenticated in the login phase. It must include security measures (i.e. validation of the inputs). Speaking about the DataManager, instead, it performs the function of build statistics and the possibility to provide possible suggestions to the municipality, if requested.

After done these components, it should be possible to implement the RequestManager which provides the functionalities of performing an Area request, an Individual request or a General request. This component is also very important because exploits the algorithm that allows both the User and the Municipality to retrieve information thanks to the data collected in the Database.
The order in which we will implement our components in the application server is then:

1. DBMSServices
2. MunicipalityAccessManager
3. ReportManager
4. AccidentsManager
5. DataManager
6. RequestManager

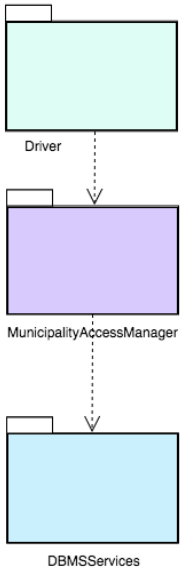
The "**Redirector**" is implemented at the end: its role is to dispatch messages coming from the client to the different parts of the system and plays a fundamental role to assure a correct behaviour of SafeStreets. The implementation of the **UserMobileApp** and of the **WebApp**, as just said, could be done in parallel with that of the components described before.

There is no mention about the implementation of **GoogleMapService** because it is an external service provided by a trusted company: thus, it will be tested only for integration test and not for the unity test. It is important that the verification and validation phases start as soon as the development of the system begins in order to find errors as quickly as possible. As mentioned, the program testing to find bugs has to proceed in parallel with the implementation: unit testing has to be performed on the individual components and, as soon as the first, even partial, versions of two components that have to be integrated are implemented, the integration is performed and tested.

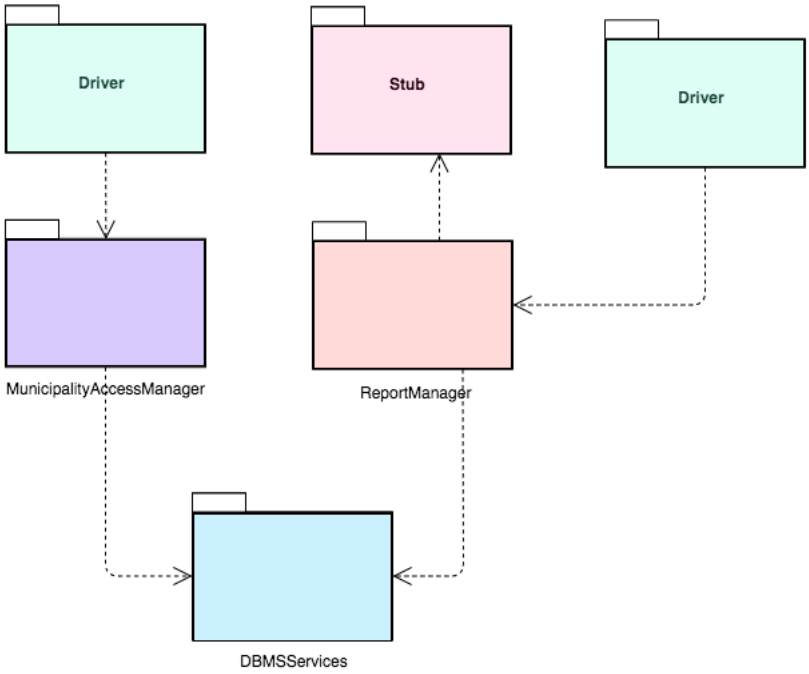
5.3 Integration Strategy

To implement and test the different functionalities of the system a **bottom-up approach** has been used. The following diagrams describe how the process of implementation and integration testing takes place, according to a bottom-up approach.

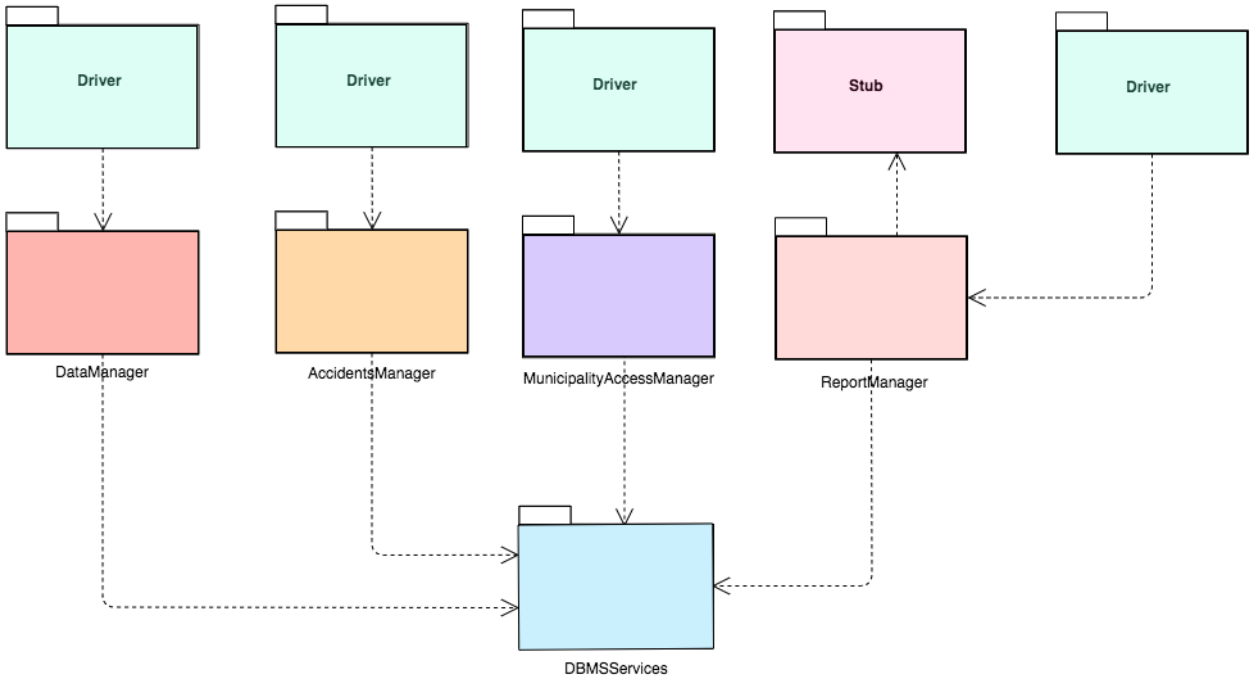
(1) At first the MunicipalityAccessManager is implemented and unit tested using a driver for the components that are still under implementation. It is not a complex component, but other components rely on it to provide some services: that's why it is implemented and tested before the others.



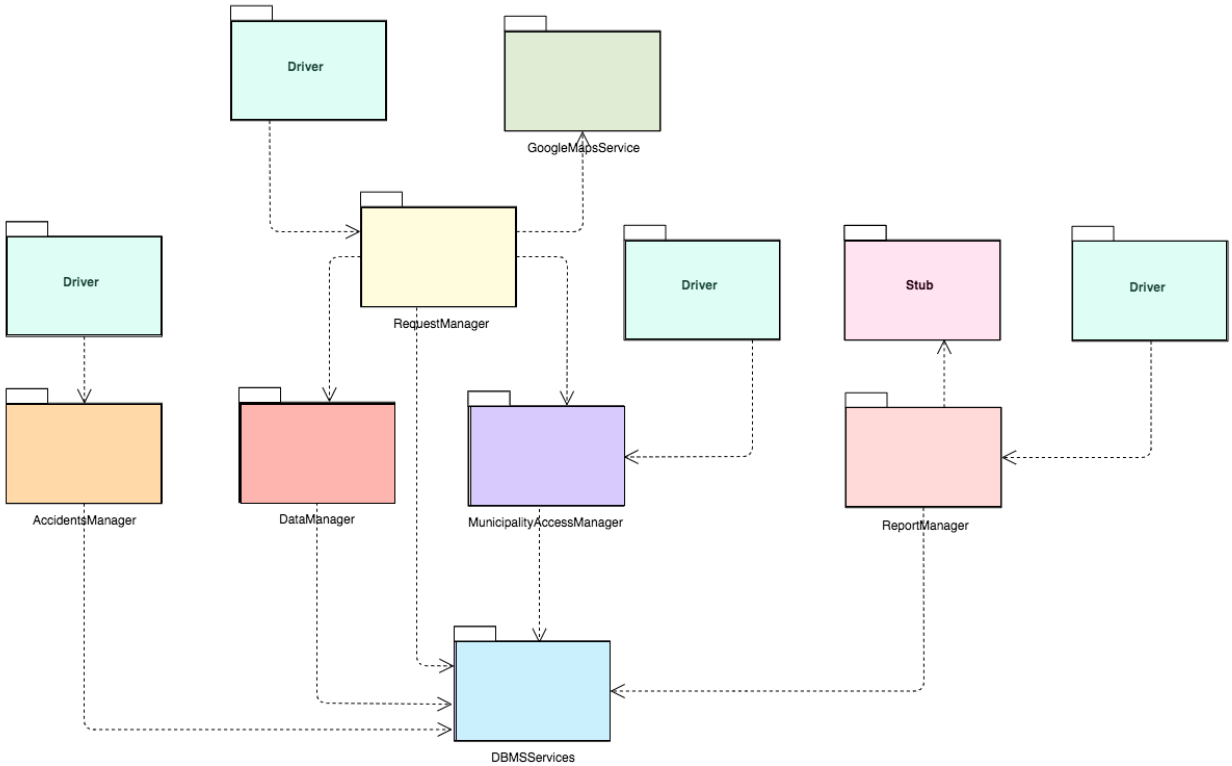
(2) Then, to implement the functionality of notifying a violation from the user and receiving the violation from the WebApp, the ReportManager is needed. However, besides the normal driver, also a stub is needed to cope with the lack of communication with the WebServer that has still to be integrated with the rest. The addition of this stub goes in contradiction with the bottom-up approach but is needed in order to proceed with the integration in the application server. The report manager is unit-tested and integrated into the system.



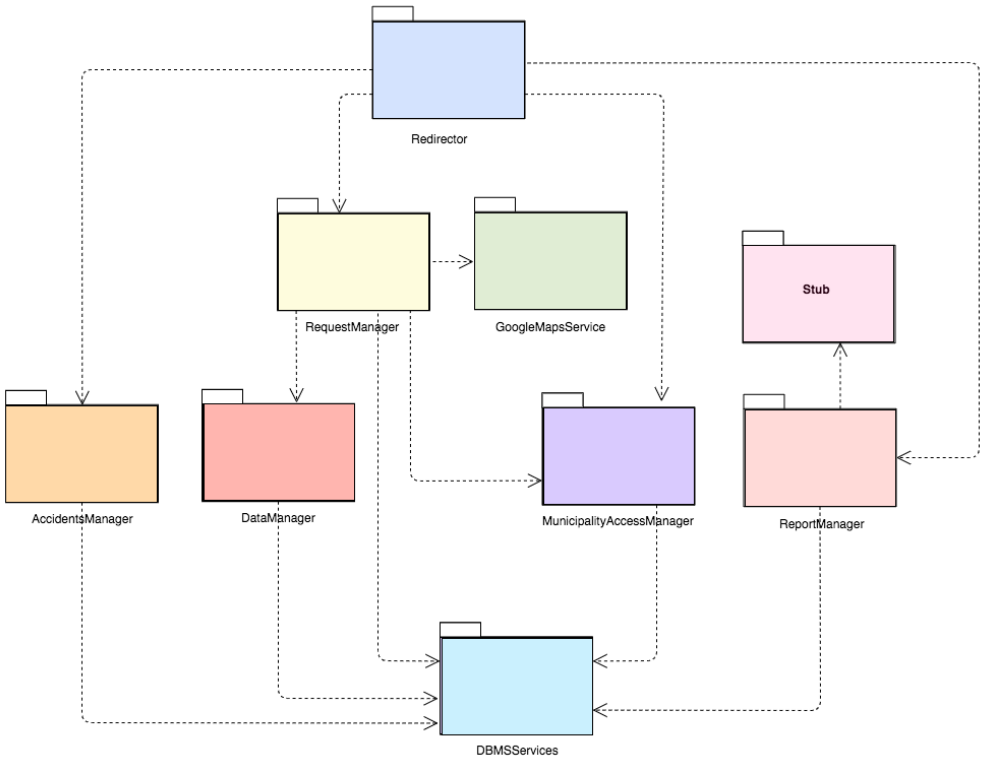
3) Then the implementation and integration of the AccidentsManager and DataManager can take place. These two components can be tested in parallel because there are no direct dependencies between them. Though, it is important that DataManager is integrated into the system before the RequestManager is completed and tested because of the usage-relation between these two components.



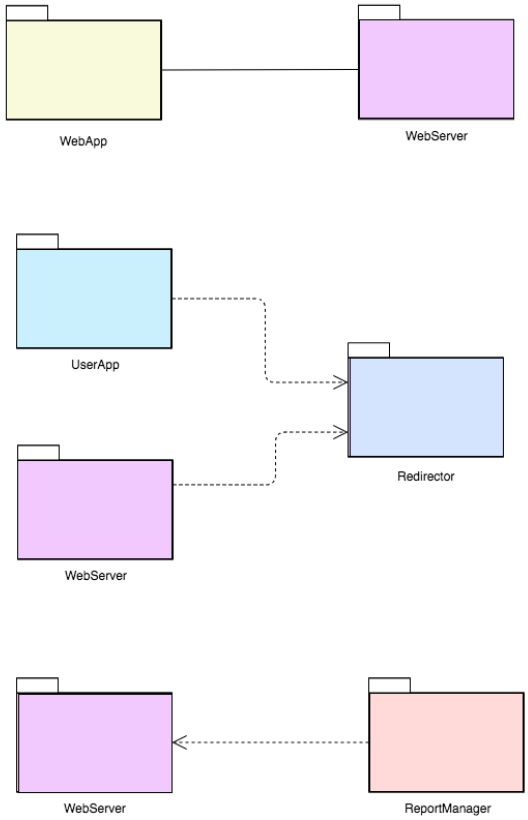
4) In a successive step the RequestManager is implemented, unit-tested and then integrated into the system. It is important to notice that GoogleMapsService will not be unit tested because is offered as a service from a trusted provider: it only needs to be integrated into the system through the RequestManager.



5) Then the different drivers in the previous diagram are substituted by the Redirector which has to be implemented, unit-tested and integrated with the other components of the application server. As previously said, its role is to allow the flow of called methods from the client to the server.



6) Finally, the remaining parts characterizing the client side must be implemented, unit-tested and integrated into the system. This process happens only when components of both sides have been implemented and tested.
Once every component has been tested and put into the entire application, system testing can be performed.



5.4 System Testing

Once the System is completely integrated, it must be tested as a whole to verify that functional and non-functional requirements are satisfied. The system testing is the means by which it is possible to do so. Moreover, the testing environment should be as close as possible to the production environment.

The system testing can be divided in more types:

- **Functional testing**: verifies if the application satisfies the functional requirements described in the RASD.

- **Performance testing**: identifies bottlenecks affecting response time, utilization, throughput and establishes a performance baseline and possibly compares it with different versions of the same product or a different competitive product (benchmarking). Thanks to this it is possible to identify the presence of inefficient algorithms, query optimization possibilities or hardware/network issues.

- **Load testing**: exposes bugs such as memory leaks, mismanagement of memory, buffer overflows and identifies upper limits of components.

- **Stress testing**: makes sure that the system recovers gracefully after failure.

5.5 Additional Specification on Testing

In this final sub-chapter are explained further information regarding the testing.
In addition, with the unit-testing and the integration testing should be done also the analysis activities that require a fully inspector activity at all stages of the development process together with an automated static analysis.

In particular, the inspection starts before the writing of the code and concerns requirements and design specified by RASD and DD (this document). This is particularly important because building the right product implies creating the correct Requirements Specifications that contains the needs and the goals of the software product. If such document is wrong or incomplete the developers are not able to write what the stakeholders want. For this reason, is fundamental that a quality team start with an analysis of RASD and DD documents.

The analysis activity, furthermore, can be divided in walkthrough and inspection. The first should be organized by the developer team with appropriate people in order to present the product step-by-step to the members of the meeting that comment on the correctness of the product (it is more informal).

While, the second should be done during the entire development activity and in this approach the quality team must conduct inspector activity by reading the code. Periodically, quality team is subjected to have “inspection meeting”. During those meetings a member of developer team (called “author” in this case) read the code, part by part and inspector point out what they have noticed as problems in the code. Quality team is not in charge to fix the code that is something that must be accomplished by the authors.

Changes made by the authors are checked to make sure that are correct and that problems have been fixed. The role of the inspection is really important because it is proved that it is more effective than testing and walkthroughs in finding errors.

Another important thing to do is to comment the code. Indeed, the developers are strongly invited to write well commented code. Semi-formal notations are also required to be written in order to assure that the activity of quality team is well supported by a fully description of the code.

Finally, the developers must produce functional test cases together with the code. It is expected that at least 85% of the code belonging to the model (referring to MVC pattern) will be covered with test cases specially the most critical parts of it. A fully coverage (or at least 85%) of unit testing conduct by developers itself helps to find problems early.

In specific test case the activity should consist in systematically testing activities concerning the characteristics and the structure of the code (white-box testing). Developers must choose inputs for the units and determine if the output is appropriate for every module.

6. Effort Spent

Topic	Hours
Introduction	2h
Discussion on second part	3h
Runtime view	5h
Component diagram	4h
Description of component diagram	2h
Component interface	1h
Revision of part 2	2h
Algorithm	2h
Architectural styles and patterns	1h
Implementation, integration and testing	7h
Documents Revision	3h

Topic	Hours
Discussion on second part	3h
Deployment diagram	8h
Description of deployment diagram	2h
Runtime view	5h
Revision runtime view + Component interface	5h
Architectural styles and patterns	2h
Other design decisions	1h
Implementation, integration and testing	10h
Documents Revision	3h

Topic	Hours
Discussion on second part	3h
Overview & high-level architecture	6h
Runtime view	6h
Component Interface	5h
Mockups	12h
Requirements Traceability	3h
Document Composition	4h
Documents Revision	3h

7. References

- The diagrams have been made with *https://www.draw.io*
- Component diagram have been made with *https://online.visual-paradigm.com/*
- Wikipedia *https://www.wikipedia.org/*
- Oracle Documentation *https://www.oracle.com/*