Design Document
Safestreets

Luca Massini Daniele Nicolò

Release Date: 9/12/2019

Contents

1 Introduction **2**
1.1 Purpose 2
1.2 Scope 2
1.3 Definitions,Acronyms and Abbreviations 2
1.3.1 Definitions 2
1.3.2 Acronyms 3
1.3.3 Abbreviations 3
1.4 Revision History 3
1.5 Reference Documents 3
1.6 Document Structure 4

2 Architectural Design **5**
2.1 Overview 5
2.2 Component view 7
2.3 Deployment view 12
2.4 Runtime View
14
2.5 Component interfaces 24
2.6 Selected architectural styles and patterns 24
2.7 Other design decisions 25

3 User interface design **28**

4 Requirements Traceability **29**

5 Implementation, Integration and Test Plan **32**
5.1 Overview 32
5.2 Implementation 32
5.3 Unit tests 34
5.4 Integration tests 34
5.5 System testing 36

6 Effort spent **37**

1 Introduction

1.1 Purpose

This document is necessary to describe the architecture of the system from several points of view. This document has the purpose of giving more details (especially to developers) about the S2B in terms of system architecture, in terms of software implementation and in terms of integration and testing. This document follows a top down approach. In fact as the reader will proceed toward the end of the document there will be an increasingly detailed description of all the design aspect concerning the system.

1.2 Scope

Safestreets is a mobile application that allows private users to inform authorities about parking and traffic violations. A user must take a picture of the violation, describe it and insert the place where it occurred. An image recognition algorithm is run in the back end on the sent picture. A user has also the possibility to see the safety of the streets and the parking areas and to check the most reported streets and vehicles.
The system offers also the possibility to receive suggestions in order to improve the streets safety. This feature is available only for municipality accounts. These suggestions are generated by the system using an algorithm that retrieves the information from the users' reports and also from the data about accidents given by the municipality.

1.3 Definitions,Acronyms and Abbreviations

1.3.1 Definitions

- **Violation** = Any kind of infringement of the driving code. Here we distinguish violations concerning the traffic and violation concerning parking.

- **Personal data** = Data belonging to the user, needed in the moment of the signing-up. They are FC, username, password and e-mail.

- **S2B** = System to be developed. It will contain all the services provided by SafeStreets.

- **Report of a violation** = A user can use SafeStreets to report an infringement he/she has seen. In particular he/she can add the type of violation, the date and the hour, the street (that can also be retrieved automatically) and he/she must add a picture of the violation.

- **Municipality request** = The municipality can ask for the same things aforementioned for the user. They will receive more accurate information rather than a private user, including for example data about the violators.

They can also request suggestions (provided by SafeStreets) to improve the quality of their service.

- **Fiscal code** = It's a 16-characters code identifying an Italian citizen.
- **Credentials** = username and password used by a user to log-in to the system.
- **MVC** = It's the Model-View-Controller design pattern. It represents the structure of the software where the Model represents the data and the data structures used to store it, the Controller manages all the logic of the software, sending and receiving data and messages from the View and consequently changing the Model. The View represents the presentation part, that contains the user interface through which the user can receive and send messages.

1.3.2 Acronyms

S2B	System to be
GDPR	General Data Protection Regulation
FC	Fiscal Code
GPS	Global Positioning System
UX	User Experience
API	Application Programming Interface
DMZ	Demilitarized Zone
DBMS	Database Management System
MVC	Model-View-Controller
SQL	Structured Query Language

1.3.3 Abbreviations

- Gn = n-th goal;
- Dn = n-th domain assumption;
- Cn = n-th constraint;
- Rn = n-th requirement;

1.4 Revision History

- Work started on 13/11/19;
- Work finished on 9/12/19;
- Document delivered on 9/12/19.

1.5 Reference Documents

- IEEE standard for Design Document;

1.6 Document Structure

The following sections are structured as below:

- **Architectural Design:** In this section there will be the description of the system from the architectural point of view. This means to show the components and their interactions with each other and to explain the design patterns choices and the architectural styles. The architecture will be shown both from a software point of view and also from a physical point of view.
- **User Interface Design:** In this section there is an explanation,in terms of user experience (UX), of the user interfaces already showed in the RASD mockups.
- **Requirements Traceability:** Here we describe how the requirements already explained in the RASD match with the design choices done in this document.
- **Implementation, Integration and Test plan:** Here there is the description of how we will implement all the components, how we will integrate them together and finally how we will test both the single components and also the integrated system.
- **Effort spent:** Here there is the division of the work hours of each member of the group and the description of the tasks completed and related time spent.

2 Architectural Design

2.1 Overview

In the figure below the architecture of the entire system is shown. This description has to be intended only as an overview of the entire system. Details will be shown later in the document.

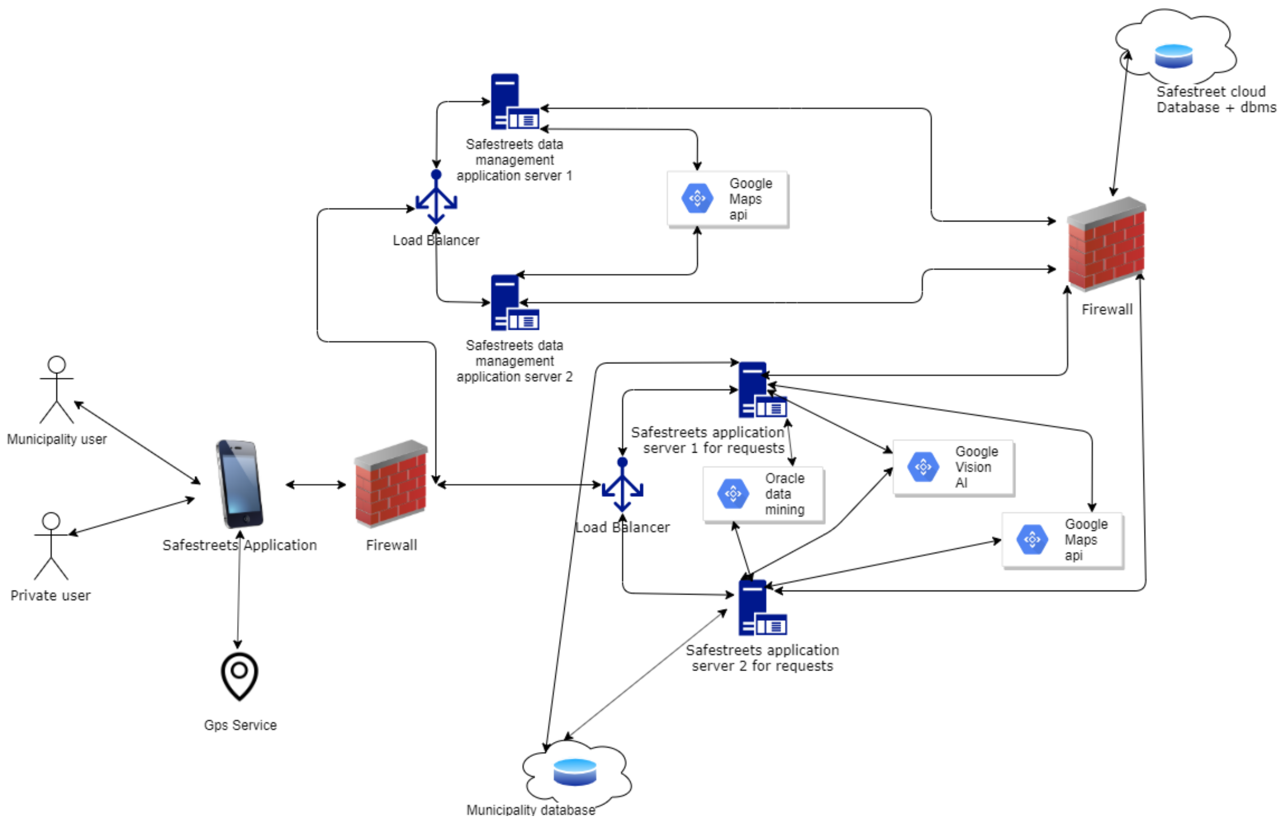

Figure 1: Architectural overview

In the diagram above we can immediately distinguish 3 layers that make up the entire architecture. Servers manage the logic layer whereas databases manage the data layer and finally the client smartphone is where the presentation layer takes place.
The architecture has 2 clusters of application servers and two "types" of servers which deal with different tasks. One type of server handles the request by the user and the other one manages the reports by the private users and the authentication. Every cluster contains 2 servers of the same typology. Every server contained in one cluster access the system database whereas instead only the two servers dedicated to requests access the external one (which is the external database of the municipality). Servers which reside in different clusters don't communicate with each other to guarantee the service. Also servers of the same cluster don't communicate with each other. The workload balancing between servers of the same cluster is managed by to load balancers.
The client, in order to receive the service, will speak directly and only to the firewall placed at the entrance of the system. The first firewall (the one near the SafeStreets application in the diagram) is the element that will route the user messages to the right cluster based on the type of the request. In this case that firewall has not only a security function but it also behaves as a gateway which routes packets. All the clusters are placed inside the DMZ (between two firewalls) for security reasons. They must be accessible from the external network. The second firewall divides clusters from the system database. The database will have to be accessible only from the clusters for security reasons and because of this fact they are outside of the DMZ.

2.2 Component view

Here there is a description of the software components that will be implemented in the specific hardware. The diagram is not horizontal in order to make it more readable.

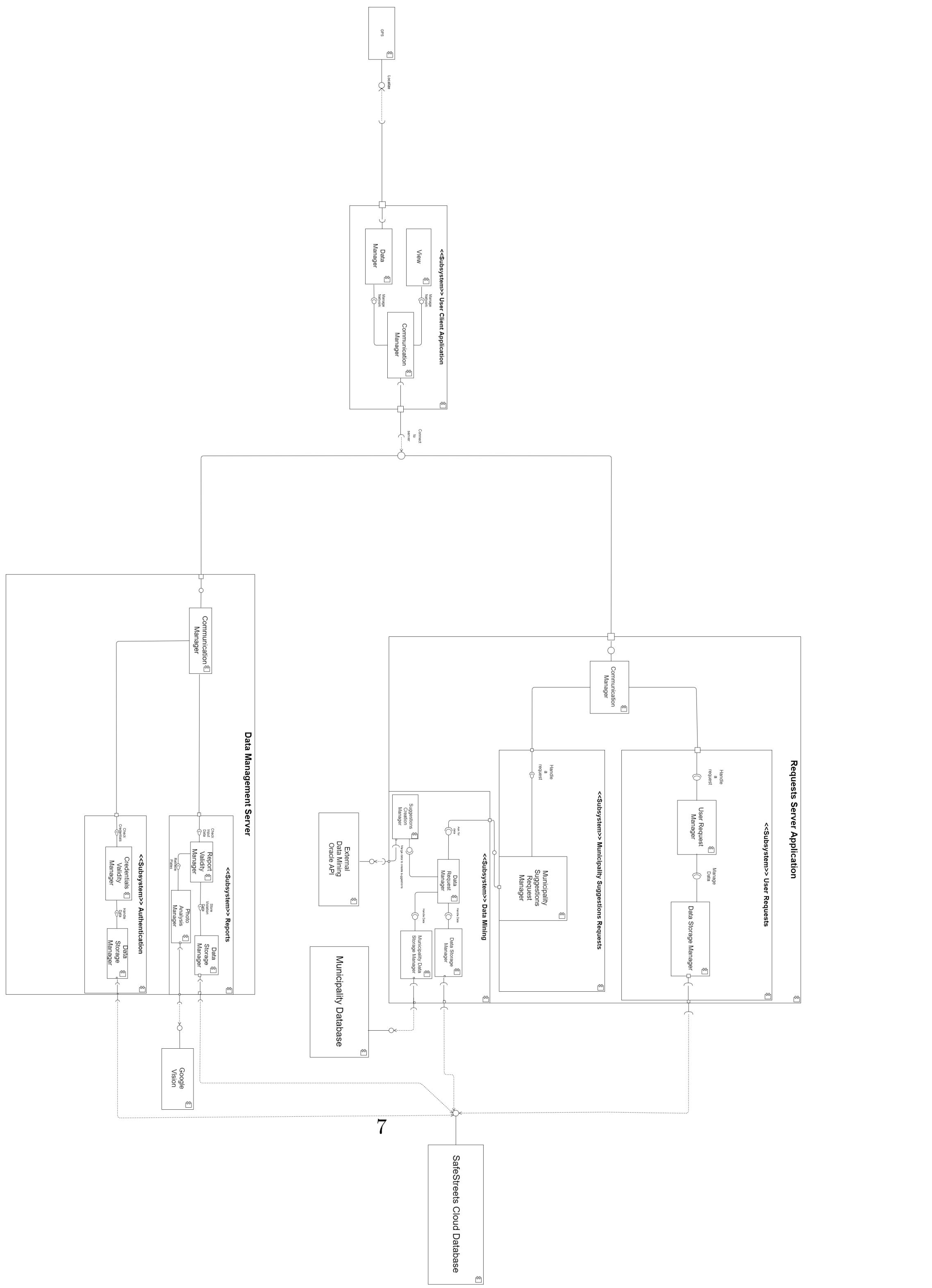

Figure 2: Component diagram

With this diagram our purpose is to show the internal software architecture of SafeStreets.
The main components are the user client application, the requests server application and the data management server application. The external interfaces are SafeStreets cloud database, the municipality database, the GPS interface, Google Maps, Oracle Data Mining API and Google Vision.
These main components contain smaller modular components that communicate with each other and with the external interfaces to complete different functions.
Now we will describe the internal components and the external interfaces in detail.

User Client Application The User Client Application is situated in the user's smartphone. It contains the Data Manager, the View and the Communication Manager:

- **Data Manager**
  This component is responsible of taking the user's position in the world using the GPS service offered by his smartphone. Those data will be sent to the server in case he selects a service that uses the GPS localization.

- **View**
  This component is the view part of the MVC pattern. It takes data provided by the controller situated in the Server and shows them to the user with its interface.

- **Communication Manager**
  The Communication Manager is responsible for the network. It manages all the communication from the client to the server and vice versa.

Requests Server Application This component is situated in the two servers of one of the Requests Server Application cluster and contains a Communication Manager, the User Requests component, the Municipality Suggestions Requests component and the Data Mining one:

- **Communication Manager**
  It has the same function of the one situated on the client. It dispatches and receives message from the client application.

- **User Requests**
  This component manages all the requests coming from the private users and the municipality, except for the suggestions requests. It contains a User Requests Manager, that elaborates the requests and asks for the necessary data to the Data Storage Manager. The User Requests component contains also the Data Storage Manager that retrieves the requested data querying the SafeStreets external database.

- **Municipality Suggestions Requests**
  This component deals with the suggestions requests made by the municipality. It contains the Municipality Suggestions Requests Manager that receives the requests from the Communication Manager and asks for the needed data to the Data Mining component. When the suggestion is ready, the Data Mining component send it back to the Municipality Suggestions Request component, especially to the Municipality Suggestions Requests Manager, that dispatches it to the Communication Manager that sends it to the Client Application.

- **Data Mining**
  This component is responsible for the retrieval of the information needed to provide suggestions to the municipality. It receives from the Municipality Suggestions Requests Component a request for suggestions. It contains the Data Requests Manager that handle the received request and asks for data to other two components, also situated in the Data Mining Component. They are the Data Storage Manager that provides the data about the violation and the streets that SafeStreets has in its cloud database, and the other is the Municipality Data Storage Manager that provides the data, again about the violations and the streets, but now querying the Municipality external database. When all those data arrive to the Data Requests Manager, it dispatches them to the Suggestions Creation Manager, that runs a data mining algorithm provided by an Oracle API in order to merge the received data and retrieve information useful to provide suggestions to the Municipality. Finally the result is sent back to the Data Requests Manager, that sends it to the Municipality Suggestions Requests Component.

Data Management Server Application This component is situated on the two server of the Data Management Server Application cluster. It contains the Communication Manager, Reports Component and the Authentication one:

- **Communication Manager**
  It has the same function than the one situated in the Requests Server Application.

- **Reports**
  This component is responsible for the violation reports. It receives the reports from the Communication Manager and it contains a Report Validity Manager that checks that the violation data inserted by the user are correct. Then it has the Photo Analysis Manager that retrieves the license plate of the violator from the photograph using an algorithm provided by an external API (Google Vision) that scans the photo. Finally there is a Data Storage Manager that takes all the information about the violation (license plate, type of violation, description, position, date and hour) and stores them in the SafeStreets database.

- **Authentication**
  The Authentication component manages all the operation about login, sign up and logout. It contains a Credentials Validity Manager that checks the user's input (both in login and sign up) and checks if he is logged in whenever he wants to logout. Then it contains also a Data Storage Manager that queries the SafeStreets Cloud Database in order to receive the account data that has to be checked by the Credential Validity Manager and stores the user's account data in the database in case of sign up.

External Interfaces Some of the components already described have to communicate with external interfaces in order to guarantee specific functionalities. These interfaces are:

- **SafeStreets cloud database**
  This component is shared with both Servers. In each of the two servers there is a component that is the only one that communicates with this database and it's the Data Storage Manager, that can both query to retrieve data and add new information into the database.

- **Municipality Database**
  This component communicates with the Requests Server Application, in particular with the Municipality Data Storage Manager, that can only query to obtain information in order to create suggestions for the Municipality. This database is in read-only mode.

- **GPS**
  This external component is necessary for the user's automatic localization. It communicates with the User Client Application, in particular with the Data Manager that is the one specialized in retrieving the position information (city and street) from the GPS service offered by the user's smartphone.

- **External Data Mining Oracle API**
  This external component is responsible for the merging and the data mining of the data from which the suggestions for the municipality are created.
  This component communicates with the Suggestions Creation Manager that sends to the external component the data about the violations and the streets condition retrieved both from the SafeStreets Cloud Database and the Municipality Database.
  The result is sent to the Suggestions Creation Manager that will forward it in order to make it available to the Municipality User.

- **Google Vision AI**
  This component is a cloud external API that is responsible for the analysis of the photo sent with the report in order to retrieve the license plate.
  This component communicates with the Photo Analysis Manager that sends the image attached to the report that it has received from the client.

- **Google Maps API**
  This component isn't represented in our component diagram because it communicates with different internal component and we have preferred not to insert it, together with all its links, only in order to keep the entire diagram clearer.
  This external component cares about providing all the needed data concerning the maps, the cities and the streets.
  It communicates with the Municipality Suggestions Requests Manager in order to provide the information about the city that the Municipality user has inserted so that the Municipality Suggestions Requests Manager can check if the inserted city exists and it's the one of competence of the applicant Municipality.
  It also communicates with the Report Validity Manager and the User Requests Manager in order to provide the requested information about the city and the street that the user has specified so that these applicant components can check if the inserted place exists or not.

2.3 Deployment view

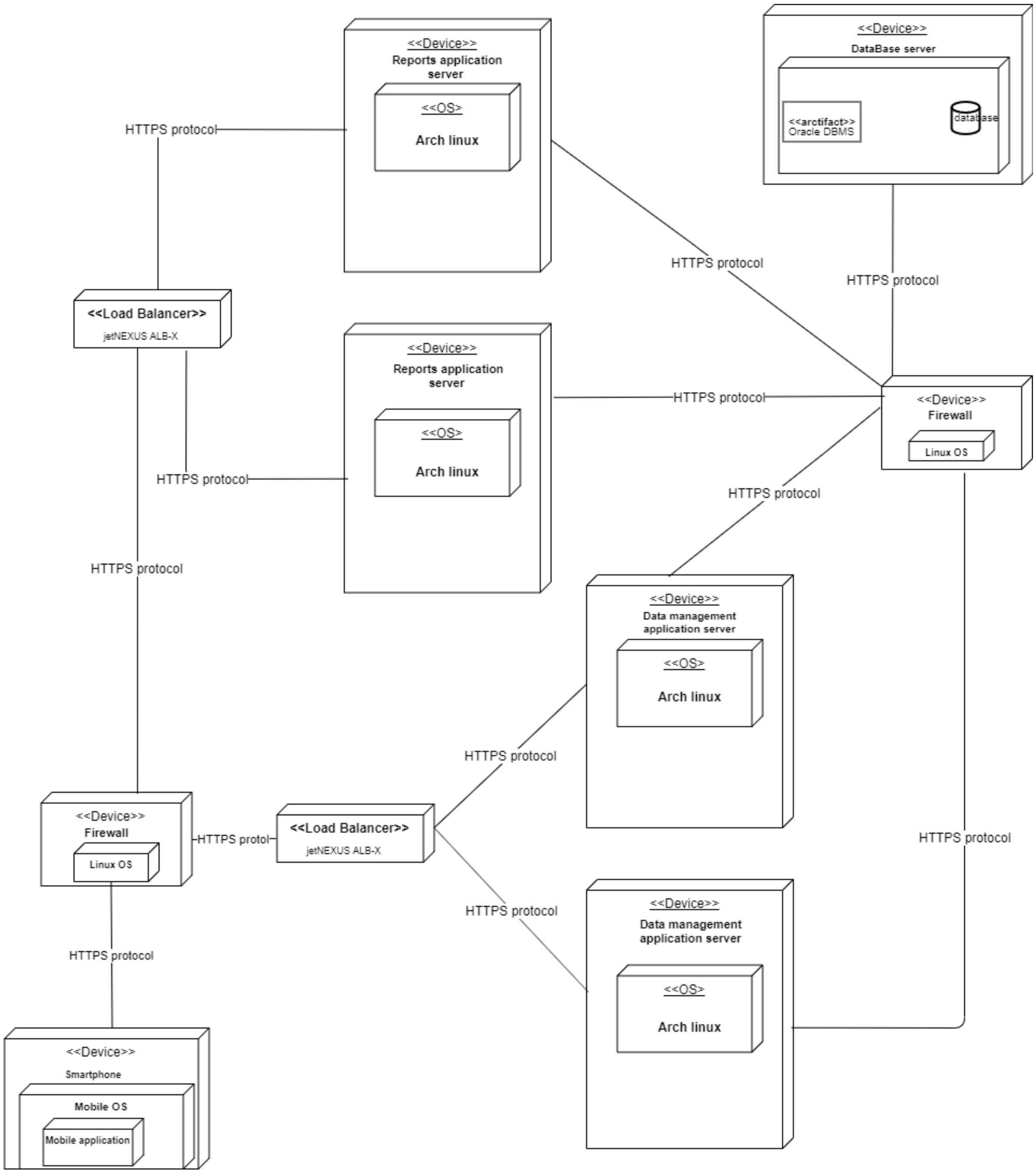

Figure 3: Deployment diagram

This diagram is very similar to the one in the overview. It expresses some aspects which have been already explained and some others explained only here. In fact we can see all the operating systems for each server and also the protocol used in the communication between the devices. Here you can find a brief description of what the single components do:

- **Firewall:** Firewalls are well known for sure by the majority of the people but we think that it is important to describe their operation the same. Firewalls are placed between the internal network and the external ones.They are used to filter the incoming/outgoing packet traffic. Through a set of rules they decide if a packet can pass or be blocked. This can be very useful to guarantee a better system security.

- **Load Balancer:** Load Balancers are used to improve the workloads across multiple computing resources. Thanks to the load balancer the load can be balanced in an optimal way.This can make the system more performing and can also improve the fault tolerance of the system because of redundancy.

- **Servers:** In this architecture we can find two clusters, each one containing 2 application servers. The clusters are specialized with no overlap of functionalities. One is specialized in the managing of the reports by the private users and authentication. It must update the database with the new incoming data only after having done all the checks about the validity and correctness of the report.
  In case of authentication, this cluster is responsible to manage all the requests of login, sign up and logout and it communicates with the system database in order to retrieve credentials and insert new accounts data.
  The other server deals with the management of the requests by the private users or municipalities. It is in charge of checking the correctness and validity of the request. In case of a suggestions request by the municipality, the system merges the data from the Safestreets and the Municipality databases with the help of a Data Mining external API provided by Oracle in order to provide the suggestion to the municipality user.
  For all the other types of requests, the system takes the necessary information from the Safestreets database in order to send them to the user.

- **Databases:** They are cloud databases managed thanks to an API that will be given by the provider of the service. We have only one cloud database used to store all the Safestreets data. Another cloud database is the external one of the municipality. Safestreets has only a special view of the part of the municipality database concerning the accidents. All these cloud databases use SQL.

2.4 Runtime View

Sign Up Runtime View

With this sequence diagram we explain which components come into play and how they interact when a user signs up. This process is the same both for private users and municipality.
When the user pushes the sign-up button, the View sends the sign up request to the Client Communication Manager that provides to dispatch it, together with the user's submitted personal data, to the Server Communication Manager, through the use of the Load Balancer in order to manage the load and assign it to one of our two servers. Then the Server Communication Manager is responsible to send all the data to the Credentials Validity Manager that checks the validity of the inserted data, controlling if the password complies our restrictions. Then it sends to the Data Storage Manager a request to query the SafeStreets Cloud Database to discover if there is already or not an existing account associated with the username inserted by the user. After having received a response from the Database, the Data Storage Manager sends the response to the Credentials Validity Manager that checks it. In case there is already an existing account, the Credentials Validity Manager sends an error message to the Server Communication Manager that propagates it to the Load Balancer and then to the Client Communication Manager informing it about a malformed request or an already-in-use account. The Communication Manager sends the message to the View that shows it to the user.
In case all the verifications are ok, the Credentials Validity Manager sends the account data to the Data Storage Manager, that cares about inserting it in the database. After having received the feedback from the database, the Data Storage Manager sends it to the Credentials Validity Manager that checks it. If the confirmation isn't valid the Credentials Validity Manager propagates a message error that, with the usual path, is received by the user. If the confirmation is ok the Credentials Validity Manager uses the same procedure to send to the user the confirmation message.

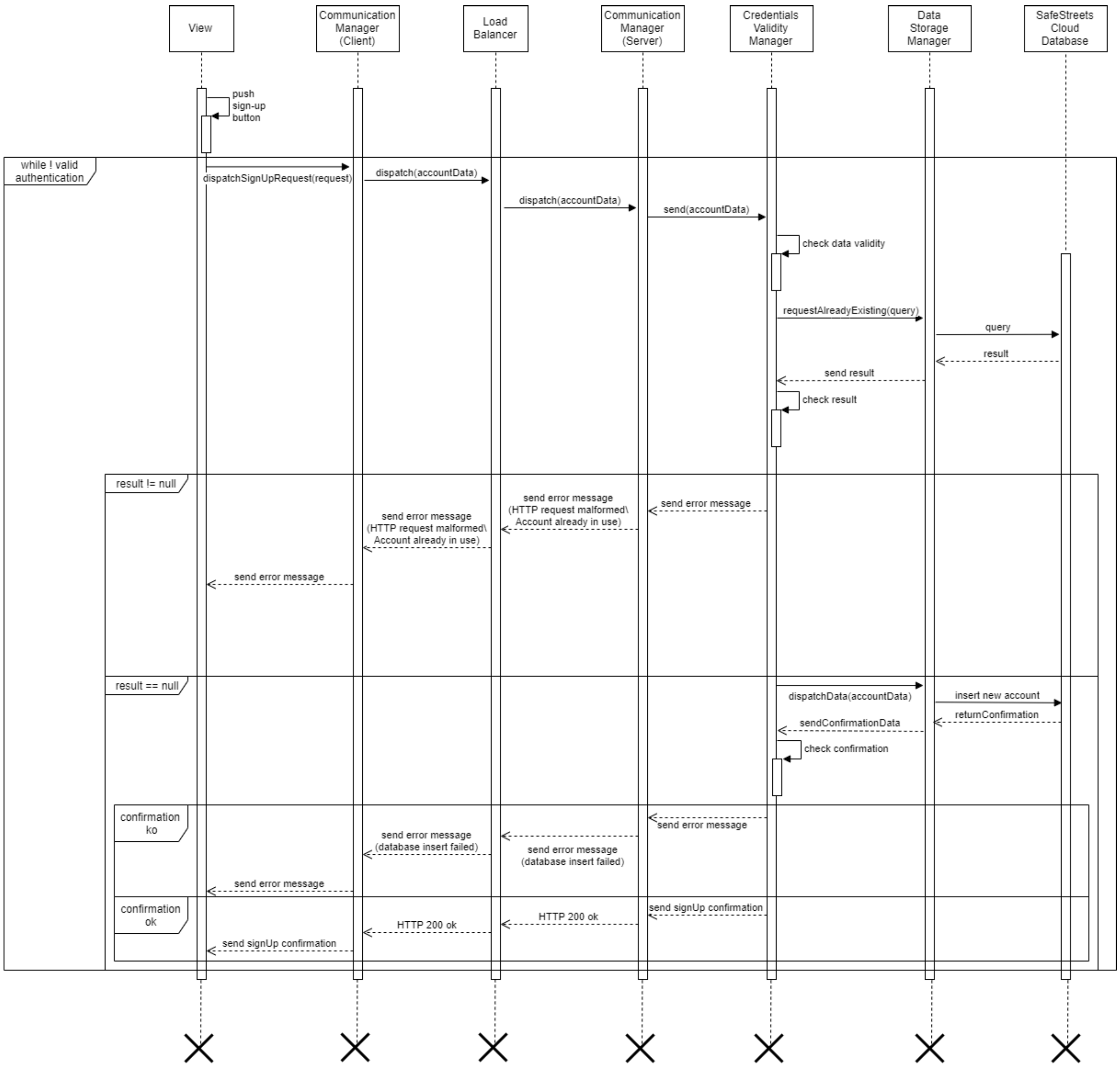

Figure 4: Sign Up Sequence Diagram

Login Runtime View

In this sequence diagram we explain which are the components involved and how they communicate with each other.
When the user pushes the login button, the view propagates the login request to the Client Communication Manager that is responsible to dispatch it to the Server Communication Manager, also thanks to the use of the Load Balancer that has the same function explained for the signing up.
Then the Server Communication Manager sends the request with the credentials to the Credentials Validity Manager, that elaborates a query to check if the credentials submitted by the user match an existing account in the database. The query is sent to the Data Storage Manager that dispatches it to the SafeStreets Cloud Database. When the Data Storage Manager receives an answer from the database, it sends the result to the Credentials Validity Manager that checks if it's valid or null.
If it's null it dispatches an error message to the Server Communication Manager that sends it to the client (through the Load Balancer) as a HTTP malformed request or wrong credentials. Finally, when the Client Communication Manager receives the error message, it sends it to the View that displays it to the user.
If the result is not null, the Credential Validity Manager sends the login confirmation message to the Server Communication Manager, that cares about sending it to the user in the usual way.

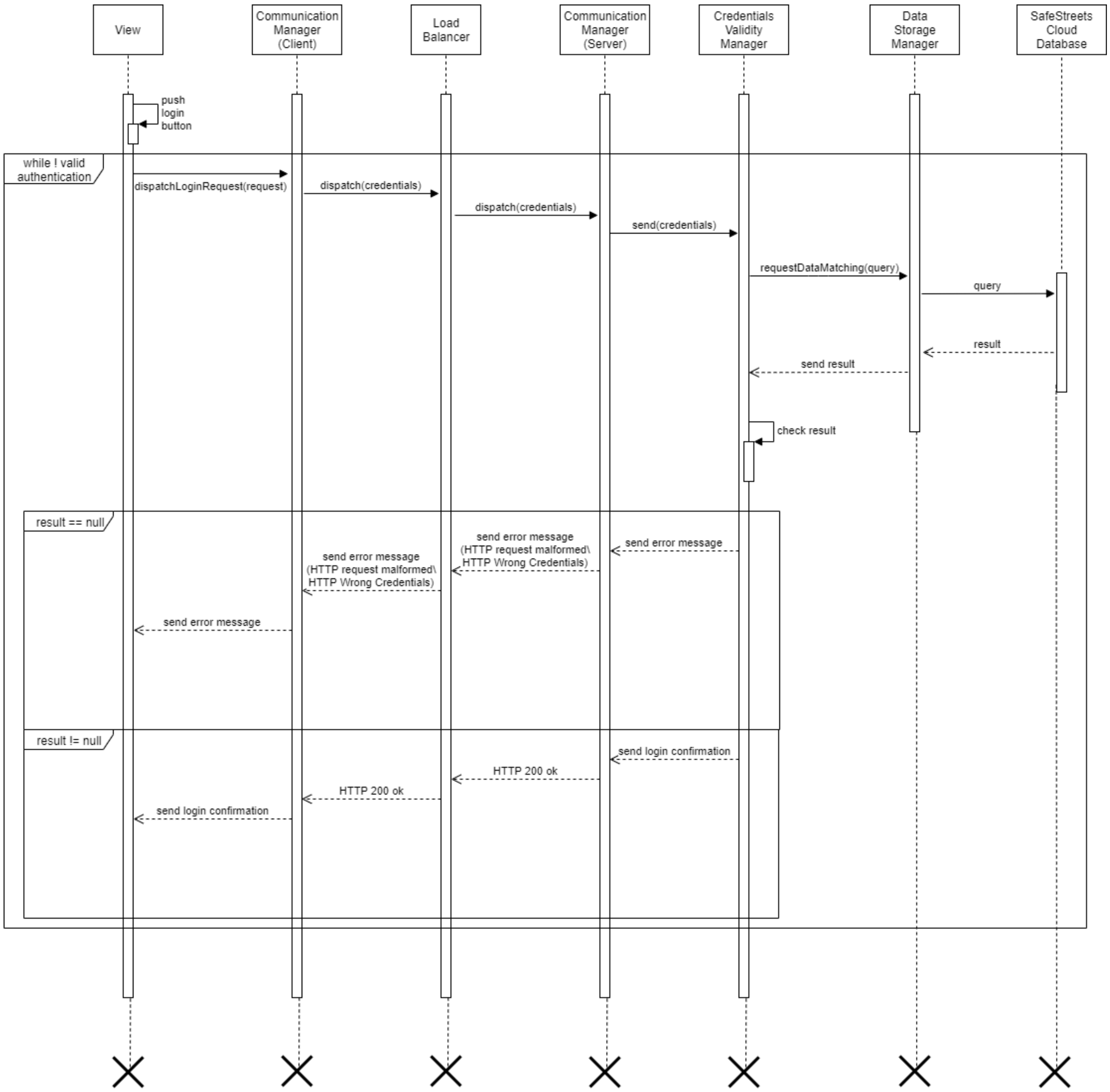

Figure 5: Login Sequence Diagram

Report Runtime View

In this sequence diagram we explain the process of a Violation Report.
When the user submits all the violation information by pushing the report button, the View sends the report to the Client Communication Manager. It dispatches the report to the Load Balancer that sends it to the Server Communication Manager. It sends the report to the Report Validity Manager that sends the city and the streets to Google Maps in order to check if they exist. Then it checks the correctness of the report. To do this it checks if the violation type belongs to one of the categories suggested by the municipality. After that, if the report isn't correct it sends back to the Server Communication Manager an error message. The Server Communication Manager will propagate it to the client (through the Load Balancer) as invalid input. Then, the Client Communication Manager sends the error message to the View, that shows it to the user, that can reformulate the report and redo the same process in order to do a new submission.
If the report is correct, the Report Validity Manager sends the confirmation back to the Server Communication Manager that, with the same process of the error communication, dispatches it to the client as HTTP 200 OK in order to finally make the View display the confirmation to the user.
In the meanwhile the Report Validity Manager sends to the Photo Analysis Manager the photo appended to the report. Then the photo is sent to Google Vision that analyses it in order to retrieve the license plate. Then the Photo Analysis Manager receives the plate found by Google Vision and dispatches it to the Report Validity Manager.
It send all the data about the report to the Data Storage Manager that provides to insert it in the SafeStreets Cloud Database that sends back a confirmation. The Data Storage Manager checks the response. If the data has not been inserted into the database, the Data Storage Manager sends an error message to the Server Communication Manager that propagates it to the user in the usual way. Otherwise, if the data has been correctly inserted into the database, the Data Storage Manager sends the confirmation to the Server Communication Manager that cares about propagating it to the user.

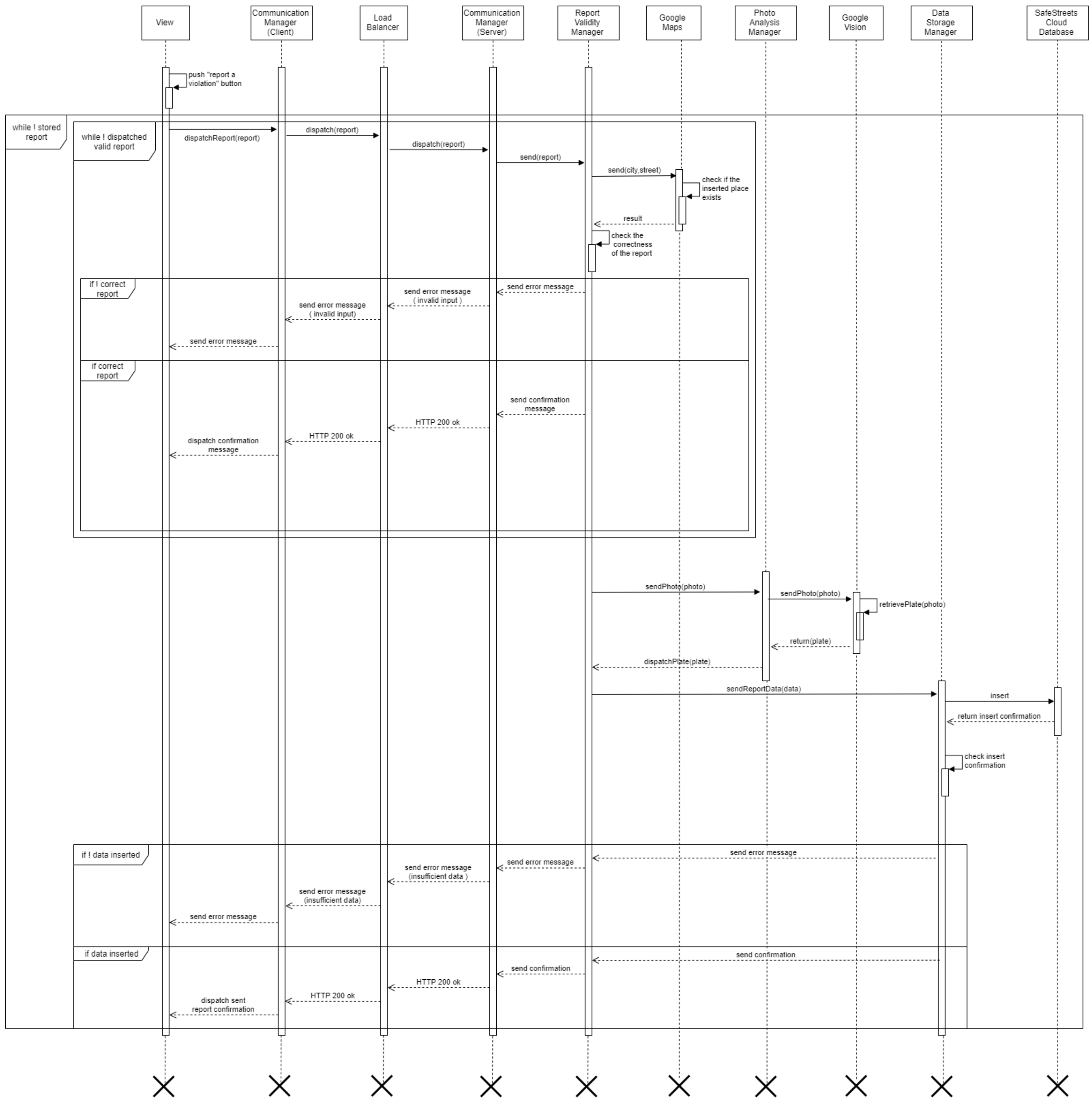

Figure 6: Report Sequence Diagram

User Request Runtime View

With this sequence diagram we show how the process for user requests is structured. This process is the same both for private users and municipality and this process is the same for all the types of requests except the municipality's request for suggestions.
When the user submits a request, the View dispatches it to the Client Communication Manager that cares about sending it to the Server Communication Manager (through the Load Balancer). Then the request is sent to the User Request Manager that sends the city and the street found on the request to ask Google Maps if there is an existing city and street.
After having received a response, the User Request Manager checks it and if the place inserted by the user doesn't exist it provides to send an error message to the Server Communication Manager, that sends it to the Load Balancer and then to the Client Communication Manager as HTTP Malformed Request. Then the Client Communication Manager dispatches it to the View that shows it to the user.
Otherwise, if the request is valid, the User Request Manager sends a data request to the Data Storage Manager, that queries the SafeStreets Cloud Database in order to receive the necessary information. After having received the result, the Data Storage Manager dispatches it to the User Request Manager that checks it.
If the database has sent a null result, the User Request Manager sends an error message to the Server Communication Manager that cares about sending it to the Client Communication Manager (through the Load Balancer). Then the message is sent to the View that displays it to the user. If the database returns a not null result, the User Request Manager sends the requested data to the Server Communication Manager that, in the usual way, dispatches it to the Client Communication Manager, that sends it to the View. Finally the View displays the information to the user.

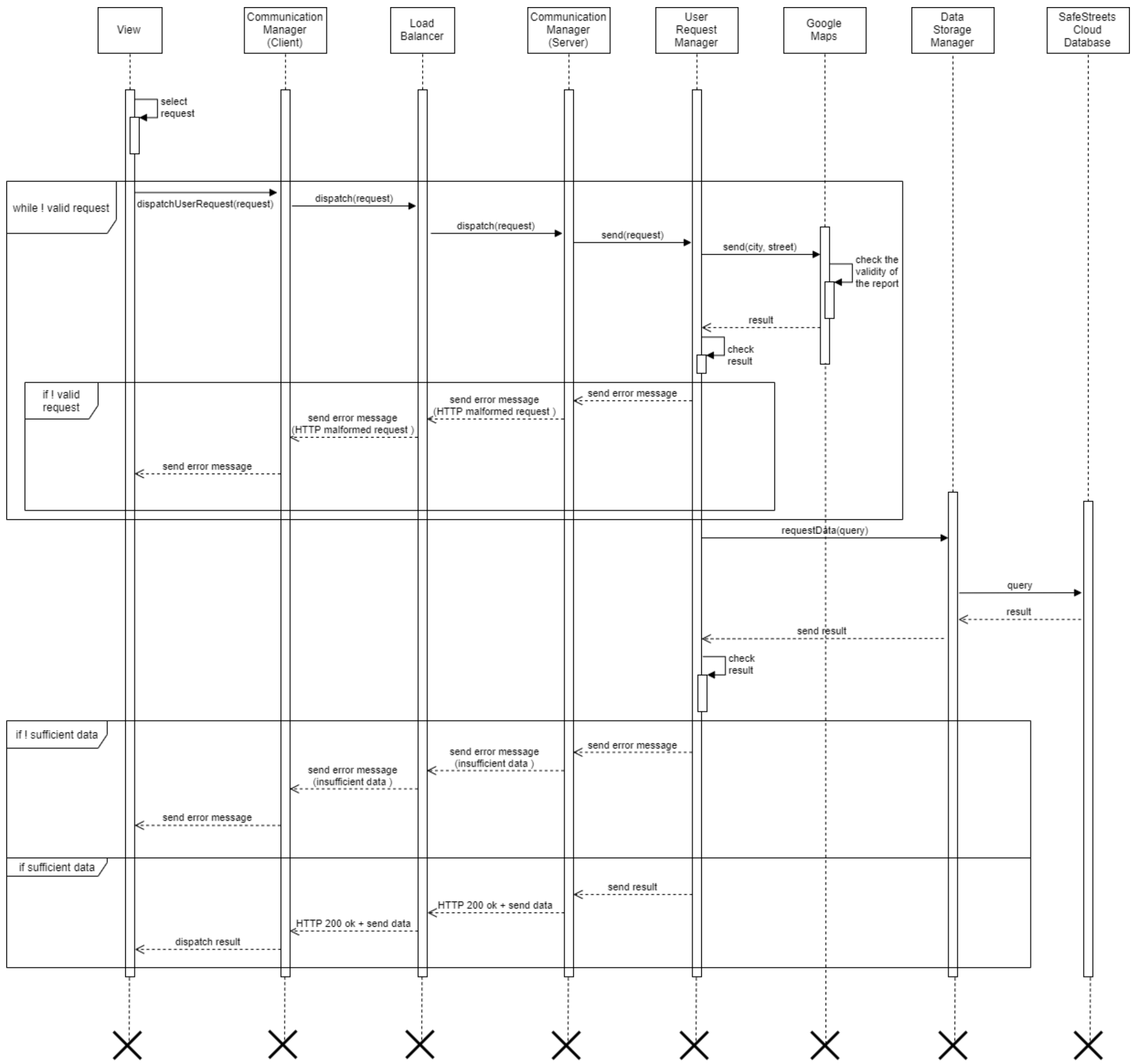

Figure 7: User Request Sequence Diagram

Municipality Suggestions Request Runtime View

This sequence diagram explains the process with which the municipality requests and receives suggestion to make the streets safer.
When the municipality user submits the request for suggestions, the View shall forward it to the Client Communication Manager. It dispatches the request to the Load Balancer that sends it to the Server Communication Manager. It sends the request to the Municipality Suggestions Request Manager, that searches through Google Maps if the user has inserted an existing city. After having received a response, the Municipality Suggestions Request Manager checks it and checks also if the city matches with the Municipality. If the city doesn't exist or isn't correct, The Municipality Suggestions Request Manager sends an error message to the Server Communication Manager, that propagates it to the user in the usual way.
If the city is correct the Municipality Suggestions Request Manager asks the Data Request Manager for the necessary data. it sends a request to the Data Storage Manager, that queries the SafeStreets Cloud Database. After having received a result from the Database, the Data Storage Manager sends it to the Data Request Manager. It does the same procedure for the Municipality data, making a request and sending it to the Municipality Data Storage Manager that queries the Municipality Database.
After having collected all the information, the Data Request Manager sends the data to the Suggestions Creation Manager, that cares about sending it to the External Data Mining API offered by Oracle. It is responsible for merging all the information and creating suggestion from the result, through the use of a data mining algorithm.
After that it returns the suggestions to the Suggestions Creation Manager that dispatches them back to the Data Request Manager. It sends all the new created suggestions to the Municipality Suggestions Request Manager. This component sends the created suggestions to the Server Communication Manager that, in the usual way, cares about sending the data to the View, that will display it to the Municipality User.

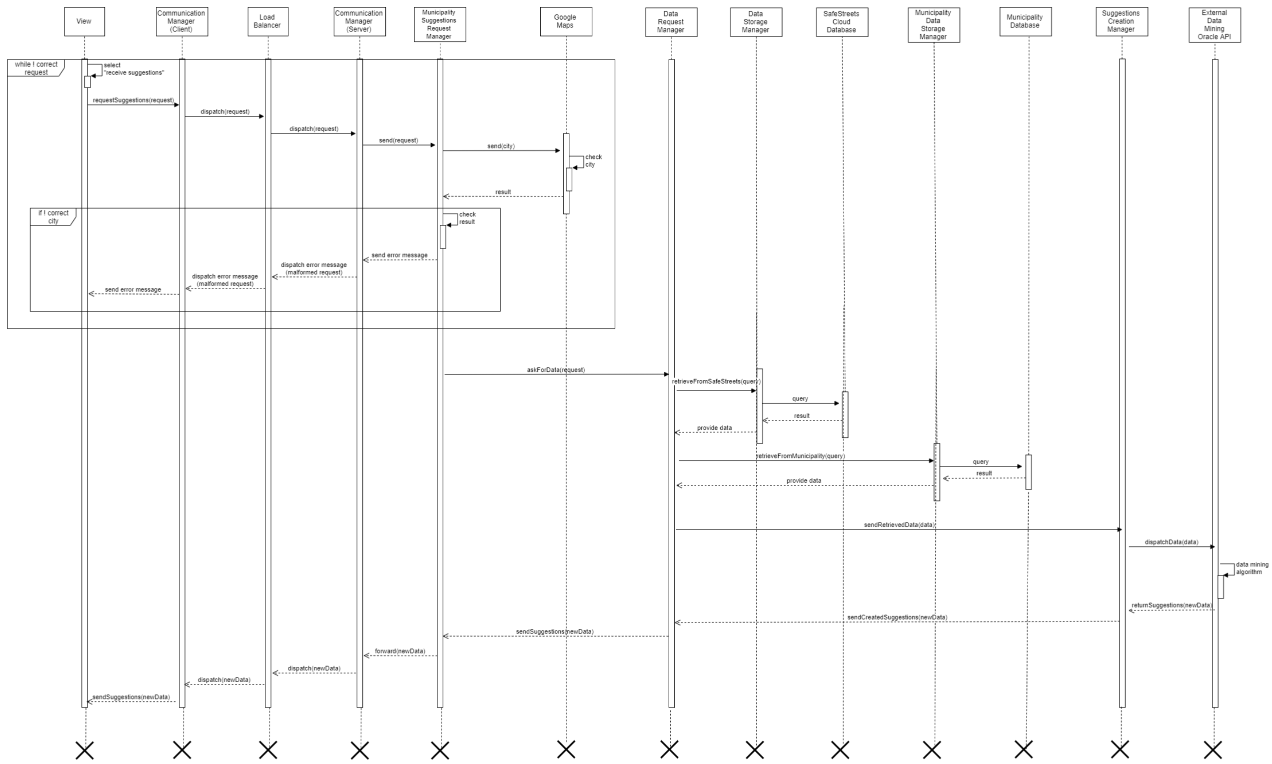

Figure 8: Municipality Suggestions Request Sequence Diagram

2.5 Component interfaces

This diagram is useful to understand which are the methods that the components use and how the components interact with each other. The diagram is made up in order to be consistent with the component diagram, that must have the same structure, and the sequence diagrams in the runtime view, that must have the same methods.

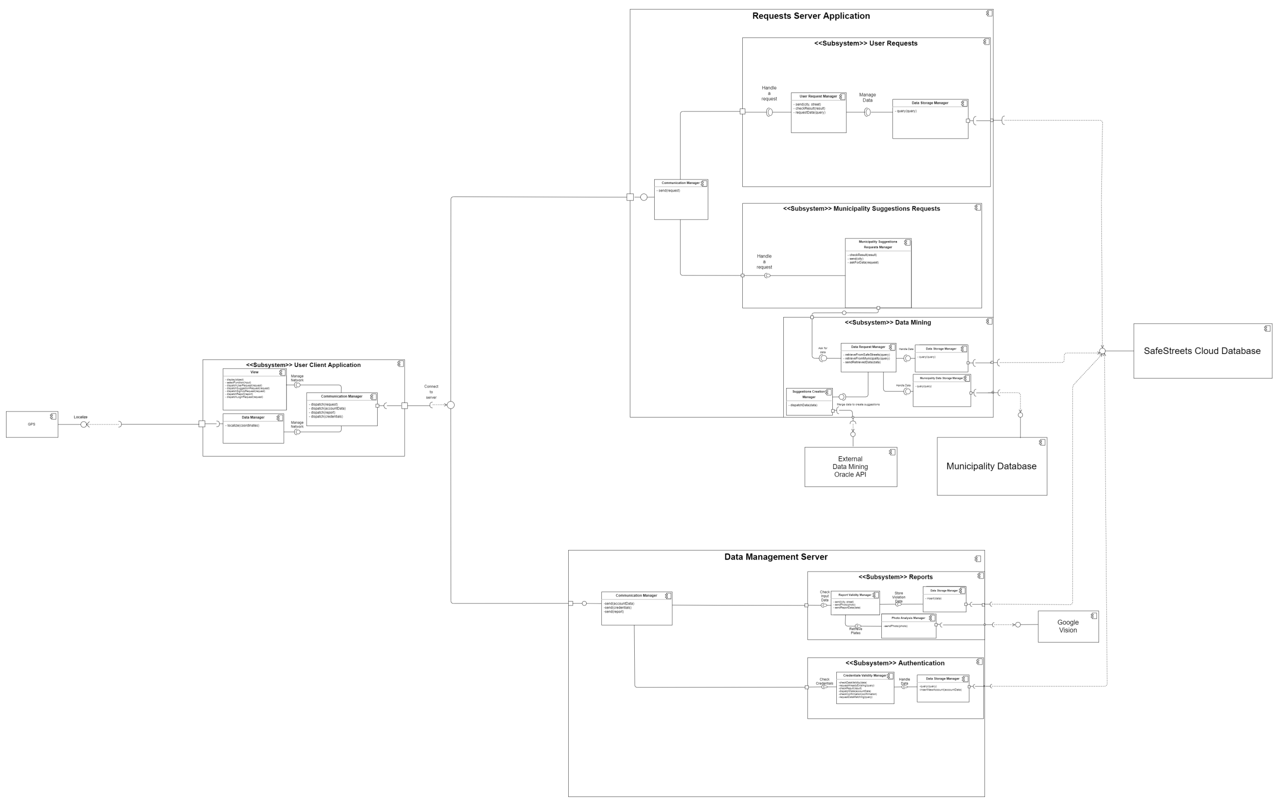

Figure 9: Component Interface Diagram

2.6 Selected architectural styles and patterns

We have selected an 4 tiers architecture. The entire system is a client-server application. The smartphone of the user implements only the presentation layer. This decision is due to the fact that we want the mobile application to be lightweight. We want the app to perform even with non-modern hardware phones. We want an application that limits as little as possible the hardware in a phone in terms of performance. The system data is stored in a database

whereas the system uses also another database which is the one of the municipality. The application logic is divided in two specialized clusters of application servers as already explained. We have chosen this to make the two interactions with the user (a user request or a user report) independent as much as possible. The data layer is implemented in the database of the system.
The decision of having different layers is thought to make the system as maintainable as possible. Every layer will offer only a simple interface to receive data and another one to send data to another layer and nothing more. In this way the implementation details are hidden. This means that when the system is modified it cannot happen that the modification of a layer causes a chain of successive modifications in other layers.

The system can be seen also as an example of MVC pattern:

- **Model:** The model is the component which stores all the application data and checks their integrity and so eventually modifies the data structures to keep it. It is implemented in the databases (both our database and the external municipality one);
- **View:** The view is the software component that deals with the presentation layer. It only shows data and messages to the user. This component is implemented in the user device (as already explained before);
- **Controller:** The controller is the software component that contains and manages the application logic. In our system this logic layer must be implemented in the application servers. In our case, in particular, it is divided in two subcomponents implemented in different devices. One part concerns the management of the requests and the other one concerns the reports and the authentication of the user. The first is implemented in the servers for the requests and the latter in the servers for the reports and the authentication.

2.7 Other design decisions

Here it is possible to find other design decisions that doesn't concern patterns and design styles.
They are:

- **Use of cloud databases:** We have chosen cloud databases to store the system data because in this way it is possible to decrease or increase the resources according to the load of requests or reports. In this way it is possible to balance the storage space according to the load automatically. A solution based on proprietary physical database would have been probably more efficient in terms of performances but surely more critical in terms of allocation or deallocation of resources. Furthermore, another important aspect of this choice is the fact that the physical maintenance of cloud database is entrusted to third parties (the owners of the physical devices which are the providers of the cloud service).

- **Use of clusters:** We have decided to use clusters instead of a single server in order to make the system more performing and more fault tolerant. If the single server is down, then, the entire service will surely be inaccessible for the user.
  The use of clusters makes the system more scalable. In fact, it will be easier to eventually add additional servers in the future.

- **Use of load balancers:** The use of load balancers has been decided to make the system more performing. In fact the throughput can be maximized and the response time minimized considerably. This is due to the fact that with the load balancers it's possible to perform parallel execution of different tasks and it is more likely to avoid the overload of the servers.

- **Use of relational databases:** Our system uses a relational cloud database. The interaction with the municipality database is possible with a SQL interface provided by the municipality. This interface translates the SQL language in order to be adaptable with the municipality database, even if it hasn't a SQL DBMS.
  One of the reasons about the decision to use relational databases is the fact that the data that the system manages and stores is well defined and structured. In fact all the data that needs to be stored can be structured in tables. The photos, that could be the only data treated by SafeStreets that can be more oriented to a non relational database, have only to be analysed to retrieve the license plate and when the analysis is done they don't need to be stored in the database.
  We know that operations like the "join" between different tables can slow down the system. However, we think that this aspect doesn't cause relevant problems in our system as the performance constraints aren't so restricted to need to avoid this kind of timing. Furthermore, relational databases have been used for several years and they are used also nowadays for a lot of applications. So they have a stronger theoretical background with respect to the non relational ones.

Here it is possible to find the relational schema of the system database, shown with an entity-relationship diagram:

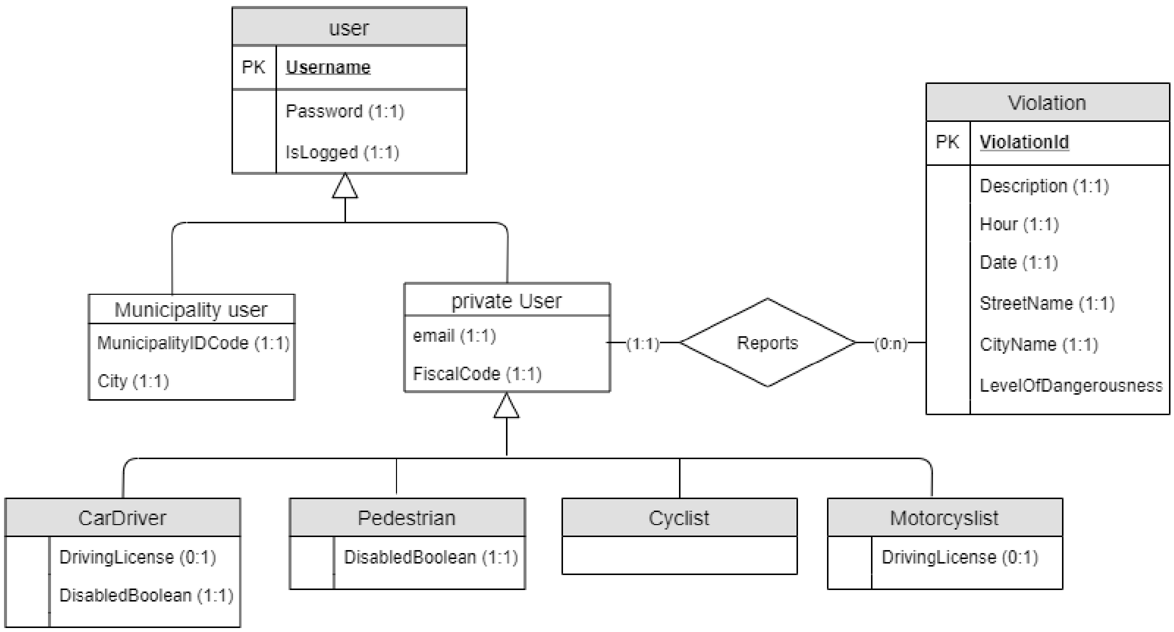

Figure 10: Entity-Relationship Diagram

3 User interface design

Here there will be shown the user interfaces with the help of UX diagrams. In particular the flow that the user will follow to navigate inside the application is shown. Some aspects concerning the user interfaces design have been already treated in the "Requirement Analysis and Specification Document". If the reader wants to understand this interfaces design in a better and more detailed way, he/she is strongly suggested to refer also to the RASD. In this document only the flow that the user must follow to use the application is described.

UX diagram:

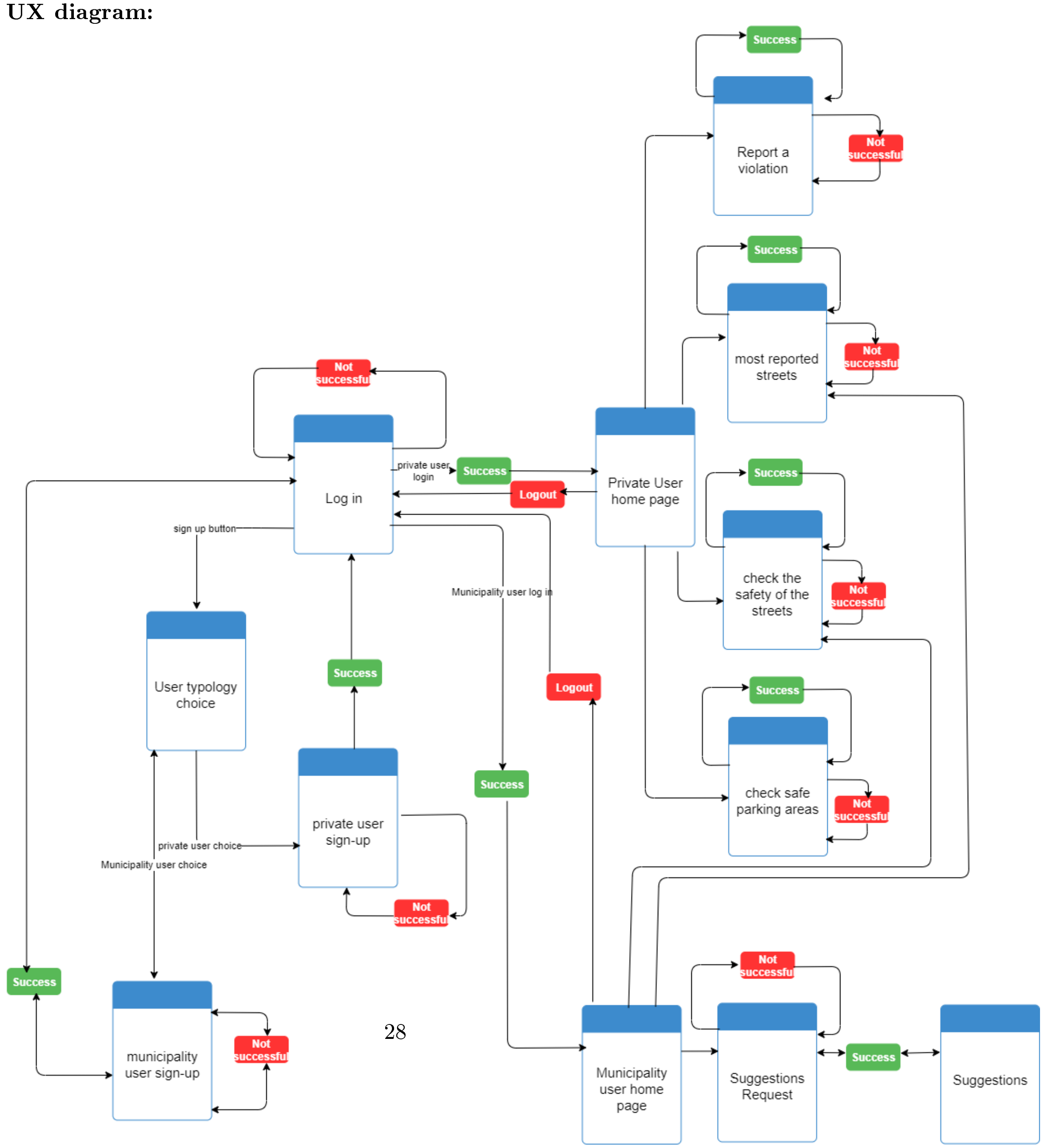

Figure 11: UX diagram

4 Requirements Traceability

Here it is possible to find an explanation about how the requirements in the RASD match with the design elements that can be found in this document.

Component	Requirements (RASD)
View	• [$R1$] :The system must allow the user to take a picture. • [$R2$] :The system must allow the user to indicate the street where the violation occurred. • [$R3$] :The system must allow the user to indicate the hour and the date when the violation occurred. • [$R4$] :The system must allow the user to insert a brief description of the violation. • [$R6$] :The system must allow the user to finally submit the report. • [$R9$] : The pictures must be taken only through the app camera and not imported from the device gallery in order to take photos in real time. • [$R12$] : The system must allow the user to insert the street around which he/she wants to check the safety. • [$R14$] :The system must allow the user to insert the city to which the inserted street belongs. • [$R15$] :The system must allow the user to insert a number representing the radius within which the system must provide the condition of the streets. • [$R16$] :The system must return the safest streets in a map form. These streets must be highlighted on the map. • [$R23$] : The system must allow the user to choose between municipality account or private user one. • [$R27$] :The system must allow the user to recover his/her password if he/she has forgotten it. • [$R28$] : The system must allow the municipality users and the private ones to access the corresponding Home Page after the log-in. • [$R29$] :The system must allow the user to log-out from the platform in every moment by the exit button placed in the Home Page.

User requests manager	• [R16] :The system must return the safest streets in a map form. These streets must be highlighted on the map. • [R18] : The system must retrieve the information about the streets safety and the violation from its cloud database.
Municipality Suggestions Requests manager	• [R22]**:** The system must provide the suggestions created and dispatched by the Data Mining API to the Municipality.
Suggestion Creation Manager	• [R17] : The system must run a data mining algorithm to merge the information about the violations and the streets safety(retrieved from the Safestreets database and the Municipality one). • [R21] : The Data Mining API must create suggestions from the data sent by the system and analysed. • [R22]**:** The system must provide the suggestions created and dispatched by the Data Mining API to the Municipality.
Data Request manager	• [R17] : The system must run a data mining algorithm to merge the information about the violations and the streets safety(retrieved from the Safestreets database and the Municipality one). • [R18]**:** The system must retrieve the information about the streets safety and the violation from its cloud database. • [R19] : The system must retrieve the information about violations from the Municipality external database. • [R20] : The system must use both data received by the users and by the municipality. • [R22]**:** The system must provide the suggestions created and dispatched by the Data Mining API to the Municipality.
Report Validity manager	• [R11] :The system must check if the reported street exists in reality.

Credentials validity manager Manager	• $[R8]$: The chosen username must identify the private user uniquely. • $[R24]$:The system must allow the municipality to sign-up if and only if they insert a valid ID code. • $[R25]$: The system must allow the user (both private user and the municipality) to sign-up if and only if he/she inserts a username that is not already used. • $[R26]$**:** The system must allow the user (both private user and the municipality) to sign-up if and only if he/she inserts a password that complies the security policy. • $[R27]$**:**The system must allow the user to recover his/her password if he/she has forgotten it. • $[R29]$:The system must allow the user to log-out from the platform in every moment by the exit button placed in the Home Page. • $[R30]$:The system must check that the chosen username (chosen during the sign-up phase) must be unique. • $[R31]$:The system must allow the user to login if and only if he inserts credentials that are associated to an existing account. • $[R32]$: The system must store the data associated to a new account in case the sign up is successful.
Data manager	• $[R5]$:The system can localize his/her position automatically as alternative to the manual insertion. • $[R13]$:The system must be able to retrieve users' position data from the GPS service of their smartphones.
Photo Analysis Manager	• $[R7]$: The system must run an algorithm to retrieve the license plate from the violation picture.

Data Storage manager	• [$R10$] : The system must store the report if it is correct. • [$R18$]**:** The system must retrieve the information about the streets safety and the violations from its cloud database. • [$R32$] : The system must store the data associated to a new account in case the sign up is successful.
Municipality Data Storage manager	• [$R19$] : The system must retrieve the information about violations from the Municipality external database.

5 Implementation, Integration and Test Plan

5.1 Overview

So far, the design of the system has been described with a top-down approach. In fact the reader goes into details as he/she proceeds through the sections and subsections. From here to the end the approach for the integration and testing will be incremental and top-down. We can logically divide our system in 3 subsystems which are the following:

- The client device which contains only the presentation layer as explained before.
- The application servers which contain the application logic and also the interactions with the databases (both external or internal)
- The external systems which, in our case, are:
  - the external data mining api.
  - The cloud api for image analysis.
  - The google maps service for retrieval of the maps, the cities and the streets.

5.2 Implementation

We want to plan the implementation phase in such a way to have the maximum parallelism among groups of developers. The order in which we want the system to be implemented is the following:

1. The databases will be the first to be implemented. In particular the relational schema of the system database and also its constraints will be implemented first.

2. The software which will reside in each one of the application servers will have be the second to be implemented. In particular we have the software related to the management of reports and authentication and also the one related to the management of the user requests. Each one of these 2 software modules will be implemented in parallel since they are independent from each other.
3. The client application will be the last part to be implemented.

Now there is the detailed description of the order in which the developers will have to develop the modules.

On the data management application server the following modules will be implemented in this specific order:

1. Communication manager.
2. Credentials validity manager.
3. Data storage manager.
4. Report validity manager.
5. Photo analysis manager.

On the requests application server the following modules will be implemented in this specific order:

1. User request manager.
2. Municipality suggestions request manager.
3. Data request manager
4. Municipality Data storage manager.
5. Suggestion creation manager.

On the client application the following modules will be implemented in this specific order:

1. Communication Manager
2. View
3. Data manager

5.3 Unit tests

In this subsection it will be explained how to do the unit tests and when.
The choice of starting the testing phase during the implementation phase is due to the fact that we want that problems and bugs will be found as soon as possible. The unit tests will be added every time a module becomes fully implemented. The aspect of finding bugs as soon as possible can make the developing process faster and more efficient with better final results. Obviously, since almost all the modules use at least another one to work, it will be necessary to simulate the modules they depend on. This will have to be done using stubs and drives. When a software component will be tested, the modules it depends on will be simulated thanks to stubs and the modules that run functions present in the module to test will be simulated thanks to drivers.
The unit testing will be supported by an automatic tool to run the tests. This makes the work of running the tests easier for developers. In addition, in this way we are surer that tests present in a previous versions of the software will be present also in the new ones. This is very important because every modification can inadvertently modify some functionalities that maybe worked perfectly in the previous versions. Because of this aspect it is strictly necessary that the tests used in the previous versions of the software will be used and run also in the new ones. The aspect mentioned above could seem very trivial for some readers, but we want to emphasize it the same because it is very important and fundamental for the success of the project. Every software module must have a code coverage of at least 75%.

5.4 Integration tests

It consists in incrementally integrating the software modules and test their behaviour until the entire system is completely formed. For the integration testing we have chosen a top-down approach. With this approach, testing takes place from top to down following the control flow of the software system. This approach has the advantages of testing critical modules first and make the fault detection easier,but it has the disadvantage of requiring a lot of stubs.

Top down integration testing starts from the modules which are at the top level of the hierarchical structure (they use other modules to work but they are not used by any module) to the bottom. The stubs are replaced by the real modules as the testing proceed until the entire system is formed.

Here there are three diagrams which represent the "use" hierarchy of the modules of the 3 subsystems (the 2 servers and the client). The modules pointed by the arrow are used by the modules from which the arrow starts. The integration and testing will have to be done from left to right following the arrows in order to execute it with the right approach (explained before).

Hierarchy of the modules of the Requests application server:

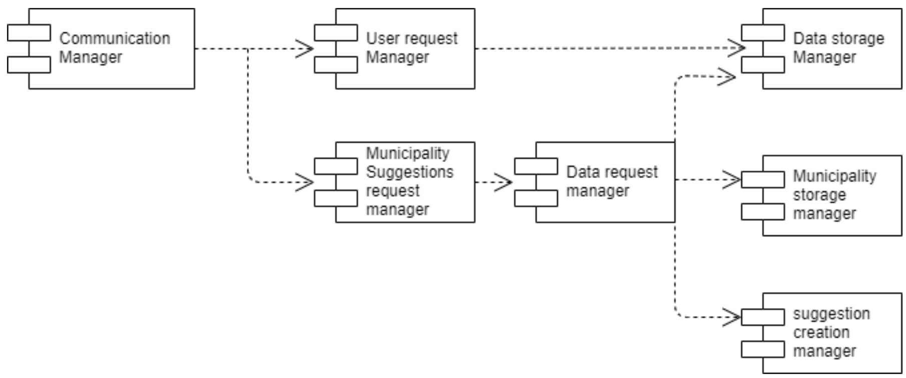

Figure 12: Requests and Authentication application server

Hierarchy of the modules of the reports application server

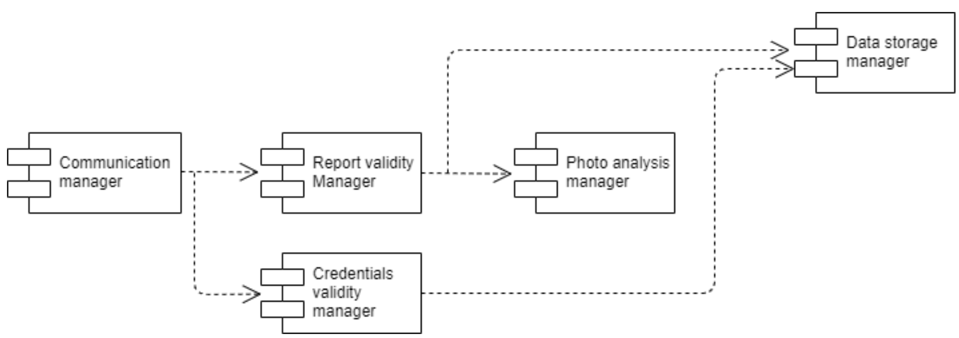

Figure 13: Reports application server

Hierarchy of the modules of the client smartphone application

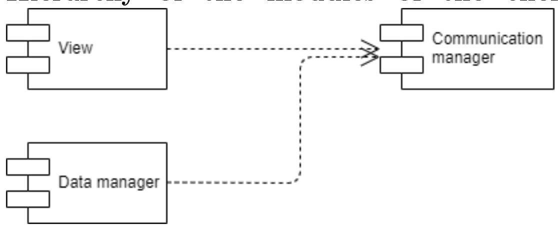

Figure 14: Client

All the subsystems use the databases and so the databases (the system one and also the external one) will be the last components to be integrated with the subsystems. After the databases are integrated, the subsystems will be tested again.

Up to now we have integrated all the components of the subsystems (The requests application server, the data management application server and the client) and so, in order to compose the entire system, we have to integrate and test all the subsystems together. To do that we have chosen a top-down approach as before. The first subsystem to be integrated will be the client, the second will be the servers (both the servers) and the last will be the databases. After every integration the result will be tested. After that all the subsystems will be integrated and, consequently, the entire system will be formed. At this step the system testing will start.

5.5 System testing

The integrated system will have to be tested with black-box tests. This means that the code will be tested only from a behavioural point of view and not a structural one. Functional and non functional requirements will be tested. The functional ones will be tested in order to test that the system meets the functional requirements. Instead, the non functional requirements concern the performances, the load and the stress. All these aspects will have to be tested. To see what are the requirements (both functional or non functional) you have to refer to the RASD.

6 Effort spent

Task	**Hours**
Group work on chapter 1	4
section 2.1, 2.3, 2.6, 2.7	12
Chapter 3	6
Chapter 4 (done in group)	12
Chapter 5	5
Group review	7
	Total
	46.0

Table 2: Luca Massini's effort

Task	**Hours**
Group work on chapter 1	4
Sections 1.3	1
Section 1.4, 1.5	0.5
Section 2.2	6
section 2.4	15
section 2.5	3
Chapter 4 (done in group)	12
Group review	7
	Total
	48.5

Table 3: Daniele Nicolò's effort