POLITECNICO
MILANO 1863

Academic year 2019-2020
Software Engineering 2 Project

SafeStreets

DD

Design Document

Version 1.1

Authors:
Marcer Andrea - 941276
Marchisciana Matteo - 945878
Motta Dennis - 940064

Professor:
Rossi Matteo

1. Introduction

1.1. Purpose

The purpose of the Design Document consists in extending what has already been specified in the RASD, by giving more technical details about the SafeStreets system.
This document aims to identify:
- The high level view of the architecture;
- The components in the architecture with their interfaces;
- The deployment of the nodes of the system;
- The runtime execution of the main features;
- The design patterns used;
- The database structure;
- A guidance about the design of user interfaces;
- A mapping between the requirements and the components;
- Implementation, integration and testing plan.

1.2. Scope

The idea behind this product is to give to the citizen the possibility to report traffic violations he sees during the day, to the authorities. Normally, the citizen would stop, call the non emergency police number, give information about current location, type of violation, brand of the car, license plate number, ecc., spending a lot of time on the phone just to report a violation. Our purpose is to make all of this quicker and easier. With our application, all that the user must do is snap a few pictures of the car in violation including the license plate and send the report to our system. In turn, the system will send the report to the municipality that operates in the corresponding city. We believe that reducing the effort will lead to an increase in the number of reports and ticket issued, reducing overall traffic violations.

Furthermore, the product will have an additional function: we want to give the citizens and municipalities the opportunity to mine the information regarding our data and that of the municipalities. Both the citizens and the municipalities will be able to see and filter violations occurred on the map of the city.

Finally the system will be able to give the municipality some suggestions on how to improve the condition of the roads. These suggestions will be based on the number of similar violations reported in the proximity of a specific area. For example, if in a certain area a lot of cars park on the street because there are not enough parking lots the system would suggest to increase their number or redesign them in a more space efficient way.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

- User: a citizen registered to the SafeStreets service.
- Municipality: a city or town that has corporate status and local government.
- Timestamp: a representation of date and time.
- Anonymized data: data that don't give any personal information, such as license plate, pictures and names.
- Valid violation report: a violation report composed by at least one picture, exactly one location, exactly one timestamp, exactly one type of violation and the license plate of the vehicle.
- Approved violation report: a valid violation report that may represent a correct violation report.
- Correct violation report: an approved violation report that the municipality evaluated as a traffic violation.
- Area: a district or a neighborhood of a country or city, especially one characterized by a particular feature or activity.
- GPS fix: a position derived from measuring in relation to GPS satellites.

1.3.2. Acronyms

- API: Application Programming Interface.
- GPS: Global Positioning System.
- UX: User experience.
- OS: Operating System.
- RASD: Requirement Analysis and Specification Document
- HTTPS: Hypertext Transfer Protocol Secure
- HTML: Hypertext Markup Language
- MS: Microservice
- DBMS: Database Management System

1.3.3. Abbreviations

- Gn: nth goal.
- Dn: nth domain assumption.
- Rn: nth requirement.

1.4. Revision history

- Version 1.0: Initial release.

1.5. Reference Documents

- Specification Document: "SafeStreets Mandatory Project Assignment.pdf".

1.6. Document Structure

Chapter 1: Introduction
In this chapter it is described the purpose of the Design Document while also giving useful definitions and explanations about acronyms and abbreviations.
It is also recalled the scope of the SafeStreets system.

Chapter 2: Architectural design
This chapter is the most complex and structured chapter of the document. It explains and motivates the architecture of the system in finer and finer details.

Chapter 3: User interface design
In this chapter it is developed the user interfaces for both users and municipalities.
First it is introduced a high level view were it is explained the flow for the UI pages, then the most important page are viewed in detail thanks to mockups.

Chapter 4: Requirements traceability
Here it is shown how the various components contribute to grant the different requirements.

Chapter 5: Implementation, integration and testing
In this chapter first it is developed a table to associate a difficulty and an importance to each feature. Then this table is also used to create a plan for implementing, integrating and testing the component present in the architecture.

Chapter 6: Effort spent
Here it is shown the effort spent by each team member working on this document.

2. Architectural design

2.1. Overview

The paradigm that has been followed for designing the SafeStreets system is the Serverless paradigm[1].

This modern approach has been chosen since the functionalities offered by SafeStreets are easy enough to be built on a 2-tier system, where the client interacts "directly" with the databases and the services. So the middle tier is removed and its logic is split upon the remaining tiers. The system relies solely on a combination of third-party services, client-side logic and cloud-hosted procedure calls (microservices). This allows it to be extremely scalable and performant with little to no server-management.
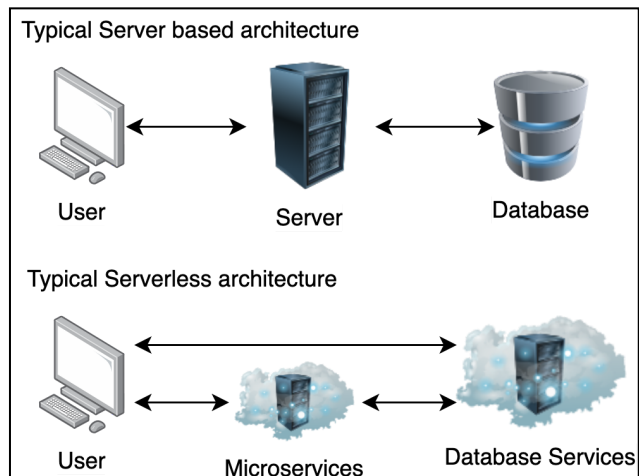

The architecture structure of SafeStreets can be seen from two different points of view: the one of the user and the one of the municipality.

In the next sections these point of views are represented using a high level view of the architecture using informal diagrams. It is important to note, however, that some simplifications have been made so to provide a more clear and understandable architecture:

- Only nodes that belong to one of the three main actors in the system (user, municipality, SafeStreets) are displayed.
- The *Authentication Server* node is not represented since it will make the diagram a lot harder to read. This node provides functionalities for authenticating users and municipalities to the other nodes, so it's heavily used when performing login and sign up of users, login of municipalities and, to make sure that documents in the databases are managed according to the security rules.

2.1.1. User's point of view
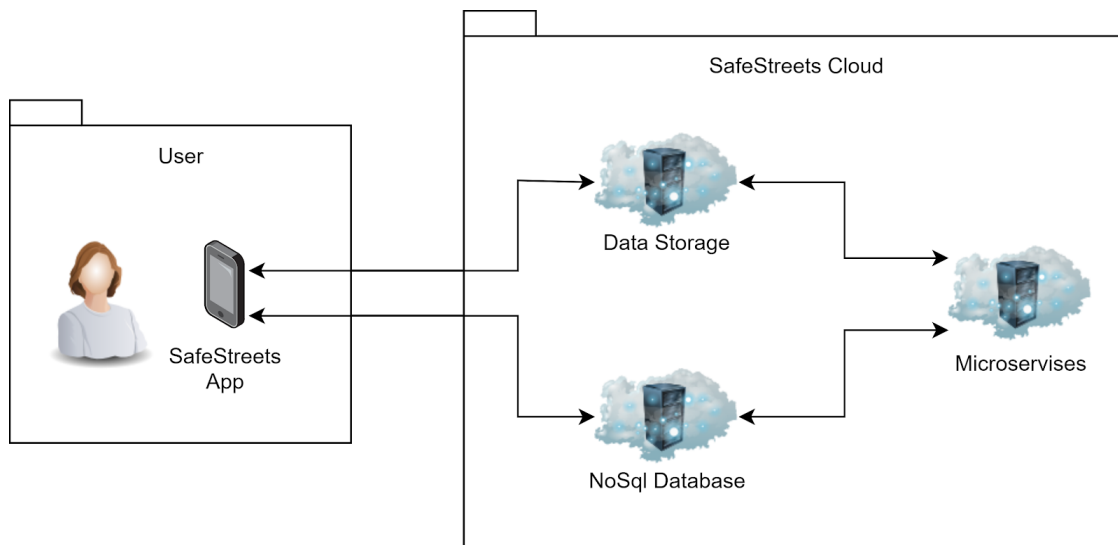
The user mainly interacts with 2 nodes:
- *Data Storage:* using this node the application of the user can easily upload large files, like images, on the cloud.
- *NoSql Database:* by interacting with this node the user's application can read and write data on the database. Note that the access to the database is limited to what the security rules allow; for instance, the user cannot see the photos posted by other users or the licence plates.

2.1.2. Municipality's point of view
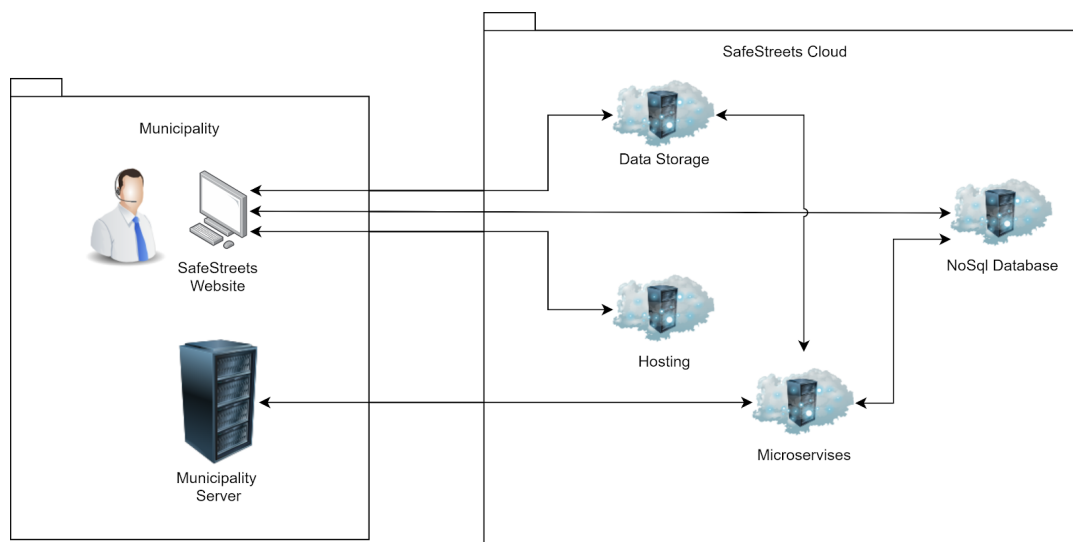
The municipality, similarly to the user, utilizes the *Data Storage* and the *NoSql Database*, but with different access rules.
In order to interact with the cloud the municipality will also uses other two nodes:
- *Hosting Provider:* this node is a simple server that listens for connections on port 443 (HTTPS) and serves static HTML pages to the municipality.

- *Microservices*: this node contains stateless functions that run in their own isolated secure execution context, are scaled automatically and have a lifecycle independent from other functions. They can modify both the database and the data storage. *Microservices* can also connects to the municipality server through its API and pull the data about accidents and issued tickets recorded by the municipality.

2.1.3. Overall high level view of the architecture
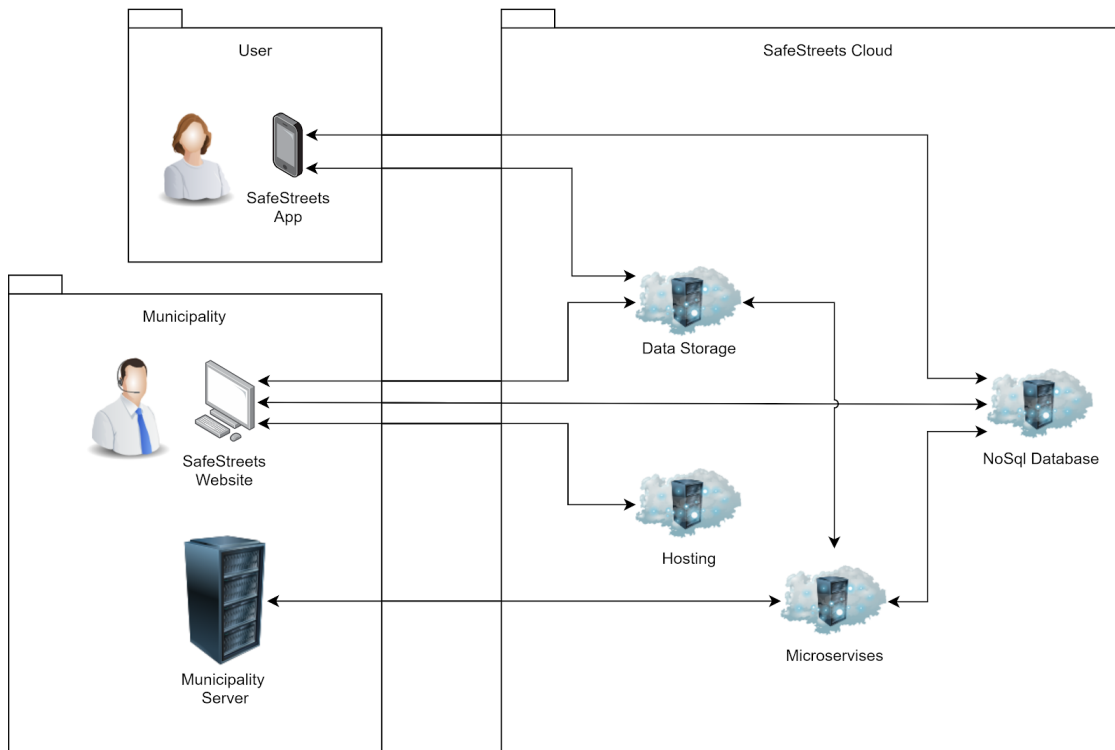
2.2. Component view

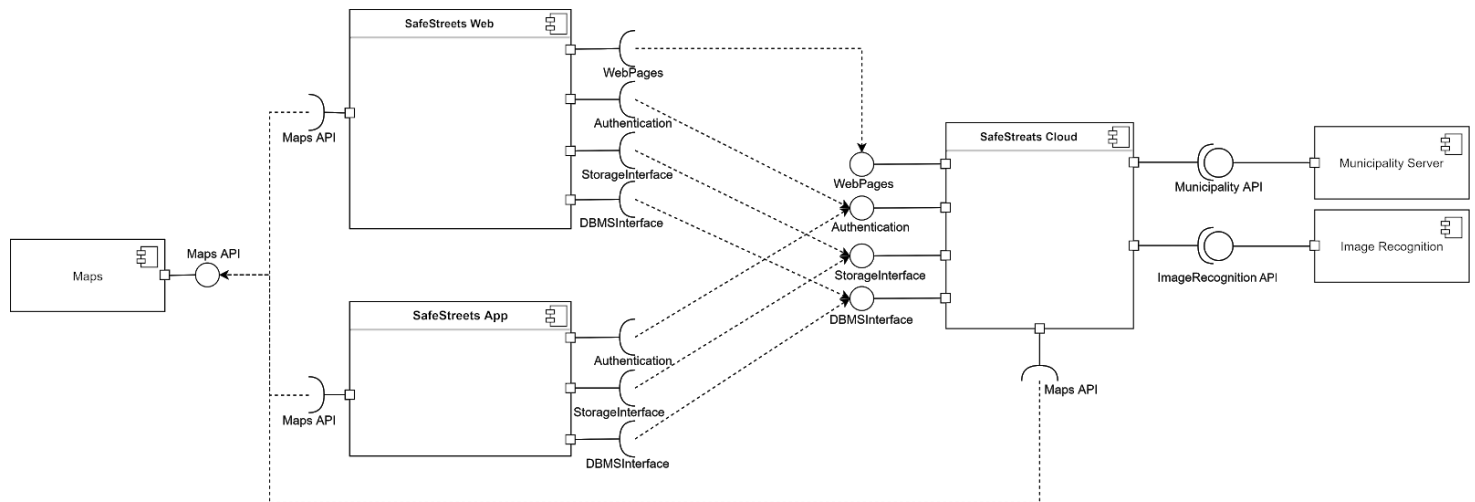The following diagrams show the main components of the system and the interfaces through which they interact.
The architecture is divided into three subsystems:

- SafeStreets Cloud
- SafeStreets App
- SafeStreets Web

Moreover, the system will use some external services to grant and enhance the service provided, that will be better explained later:

- Maps API;
- Municipality API;
- Image recognition API

2.2.1. Component Diagram of the Cloud
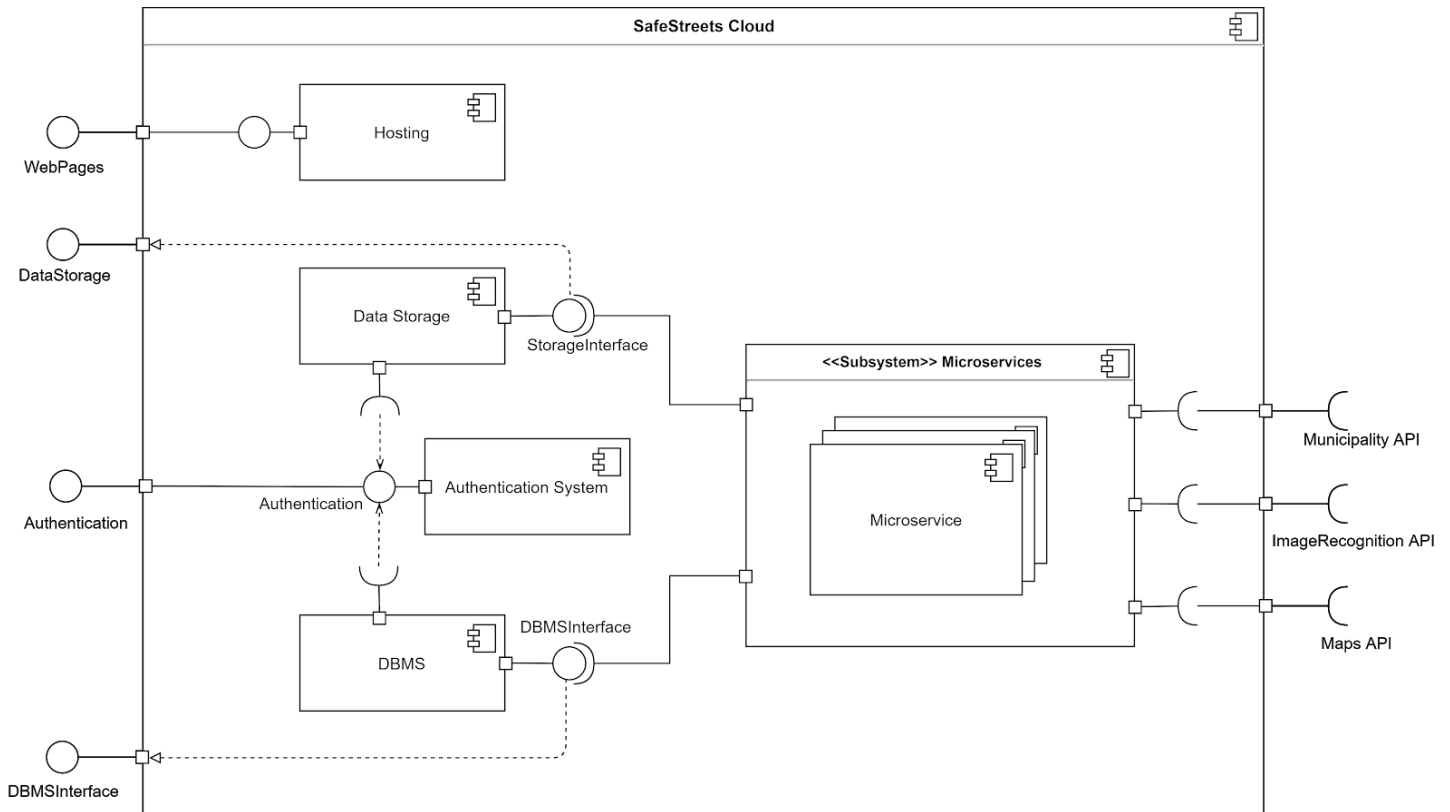
The cloud architecture is simple and has only a few components:
- The *Hosting* service: which hosts the web interface.
- The *Authentication system*: used both by the SafeStreets App and the Web interface to grant authentication.
- The *Microservices*: this is a collection of microservices that are used to manage both the data storage and the database. It is the most important and complex component of the SafeStreets Cloud component so it will be analyzed in detail later in the document.
- The *DBMS*: which manages the database.
- The *Data storage*: in which there are mainly stored pictures of violations.

2.2.2. Component Diagram of the Microservices
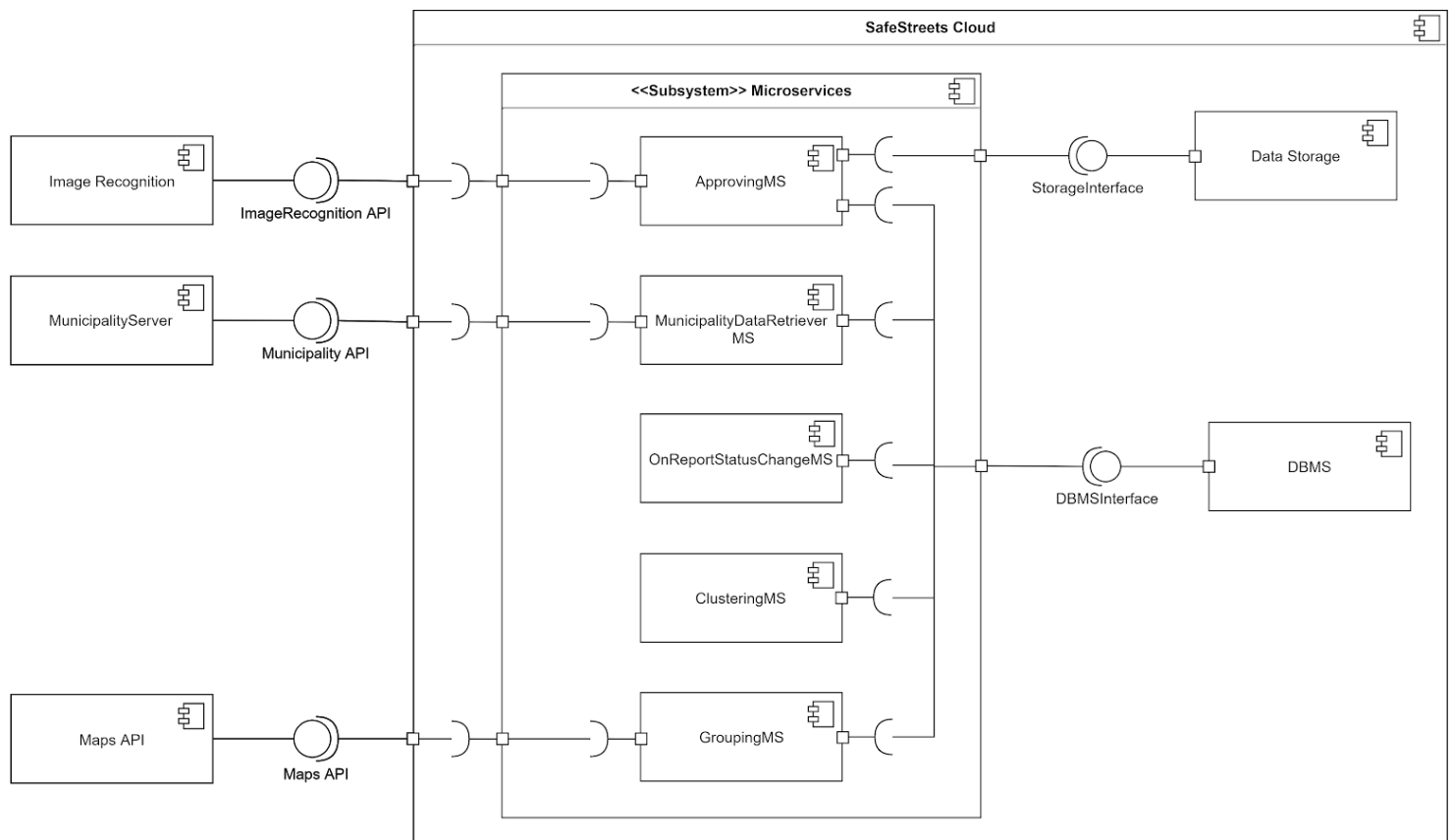
As it can be seen, we devised five Microservices. Here we explain their function:

- *ApprovingMS*: when receiving a report, this microservices checks and approves it. It uses an Image Recognition service to do so;
- *MunicipalityDataRetrieverMS*: this is the microservice that connects to the interface of the municipality to retrieve data about accidents and tickets, and stores it in the database;
- *OnReportStatusChangeMS*: it is triggered when the municipality confirms or rejects a violation group; it updates the status of all the reports contained in the group;
- *ClusteringMS*: this service groups together reports that have the same type of violation and close location creating a *Cluster. Clusters* are a data structure that helps to display violation reports on the map and to find useful suggestion for the municipality. The microservice is triggered each time a report is received.
- *GroupingMS*: this service groups together reports that belong to the same traffic violation. This is done by checking the location, time, type of violation and license plate. If two or more reports that represents the same traffic violation are posted, they become associated with the same *group. Groups* are a data structure that helps to display the reports to the municipality,

ensuring that there is only one group for each traffic violation. The microservice is triggered each time a report is received.

2.2.3. Component Diagram of the Application
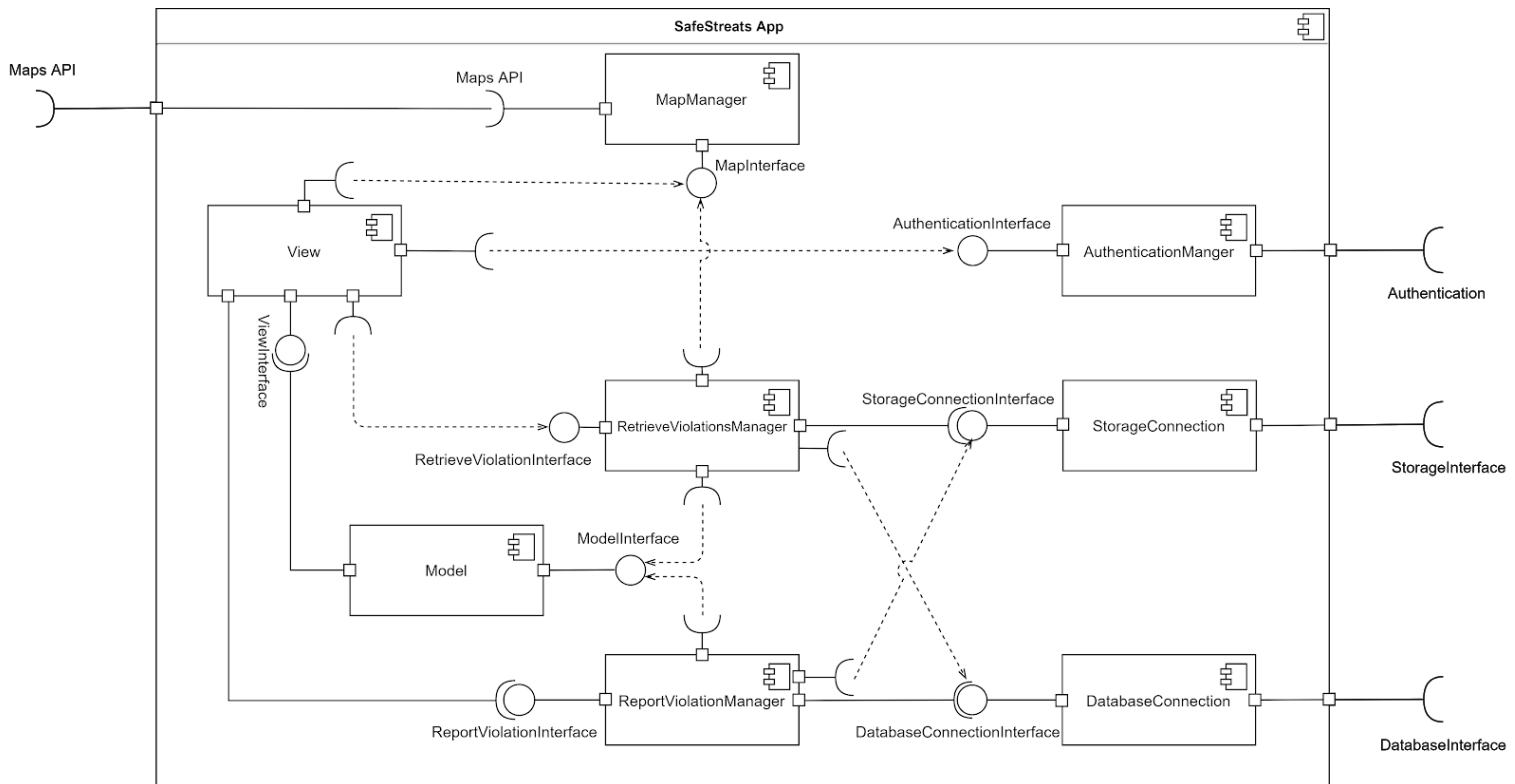
The SafeStreets mobile application is built using the Model - View - Controller[2] design pattern. In this diagram, however, we wanted to highlight the role of the Controller and how it interacts with the various components of the cloud architecture. The Controller has the following components:

- *ReportViolationManager*: the core of the application. It includes the logic that allows the user to submit violation reports.
- *AuthenticationManager*: it interfaces with the Authentication System to provide the authentication service to the user.
- *DatabaseConnection*: allows to connect to the DBMS.
- *StorageConnection*: allows to connect to the data storage.
- *MapManager*: it is used by RetrieveViolationsManager and allows to display violations and other data on the map by interfacing with the external Map API.
- *RetrieveViolationsManager*: manages the queries to the Database for showing SafeStreets data to the user.

2.2.4. Component Diagram of the Web Interface
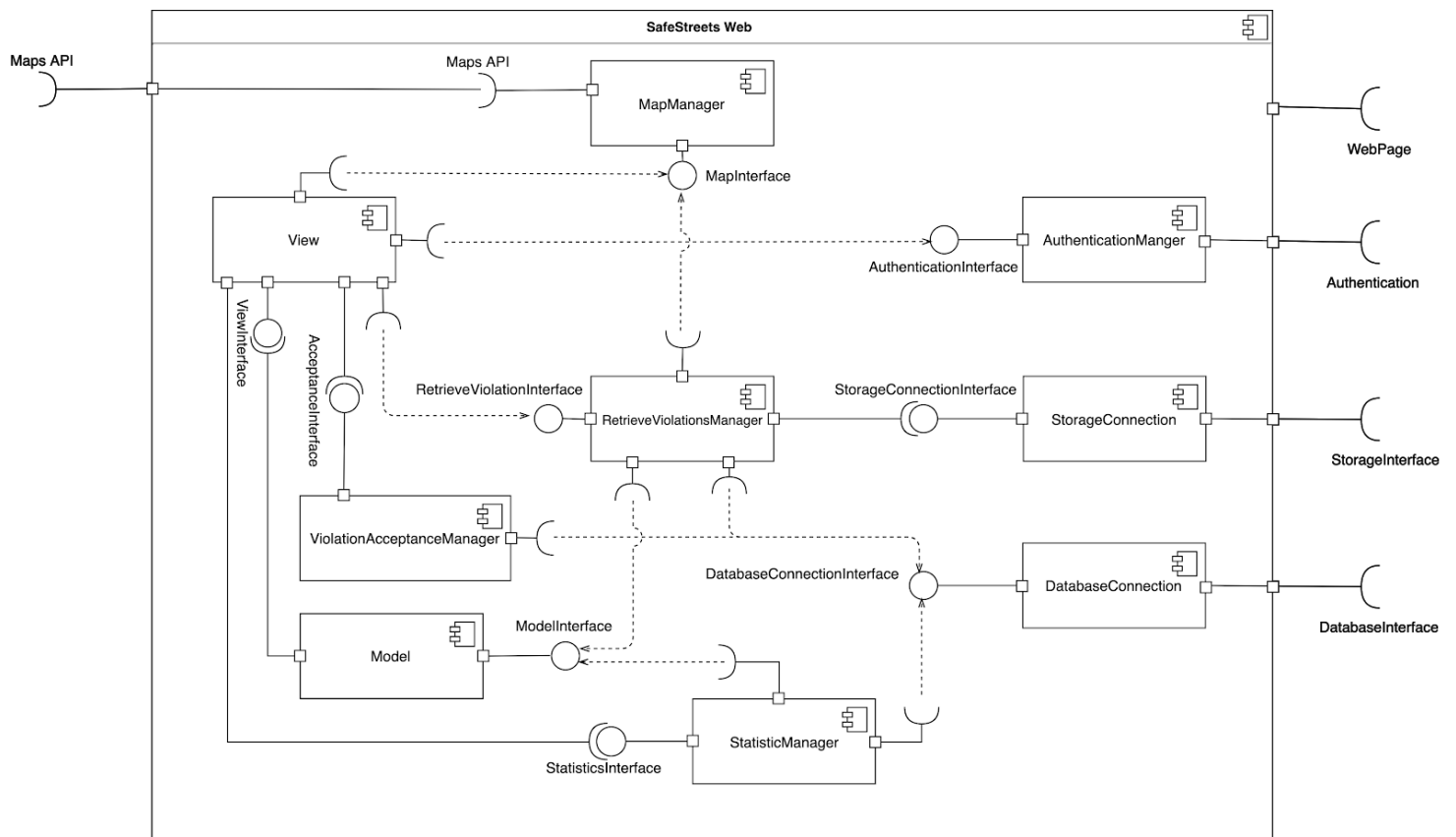
The web interface offered to the municipality has a similar structure to the application with the obvious difference that the view will be generated by an HTML web page. It has, however, some extra components:

- *StatisticsManager:* it retrieves data from the database and generates useful statistics for the municipality, such as the effectiveness of the SafeStreets initiative, the increase in tickets, the eventual decrease of traffic violations, weekly number of reports, ecc;
- *ViolationAcceptanceManager:* this is the component that displays the groups of reports; here, the municipality can see all the pictures of the violation sent by the users, along with all the description of each user, and the license plate. The municipality, after reviewing the report, can update its status by "confirming" or "rejecting" the report.

2.3. Deployment view
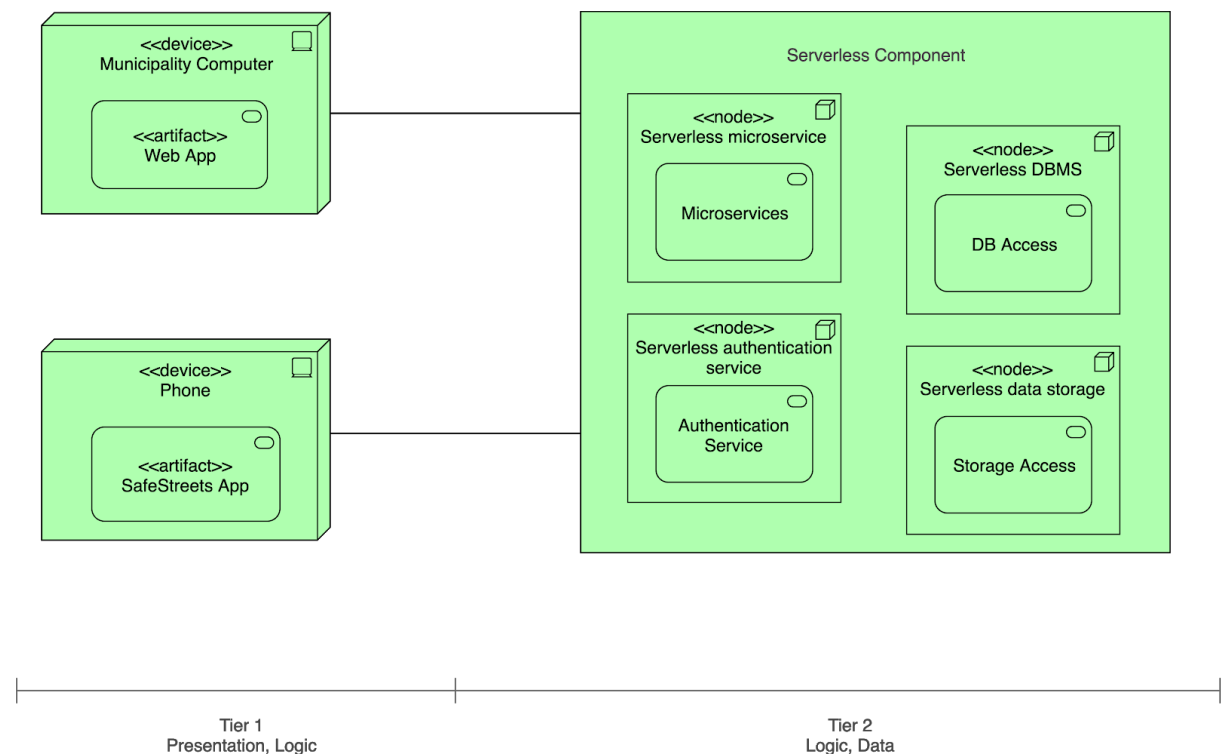
2.4. Runtime view

The following sequence diagrams represent some of the most important aspects of the system:
- User violation reporting,
- Municipality violation acceptance,
- Visualizing SafeStreets data for Users,
- Visualizing SafeStreets data for Municipality,
- Cluster's details (e.g. suggestions) for the municipality.

2.4.1. User violation reporting
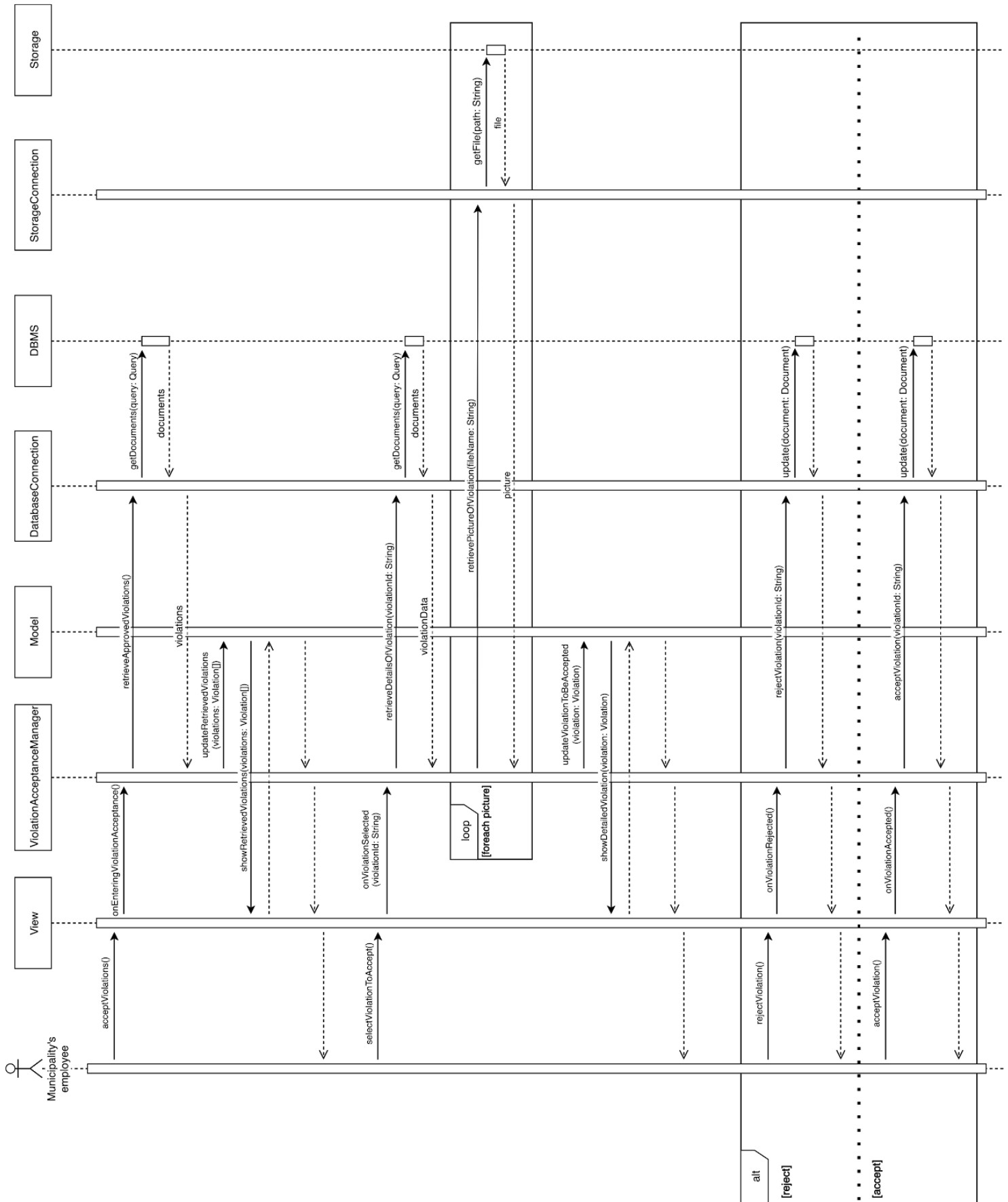
2.4.3. Retrieving of municipality's data
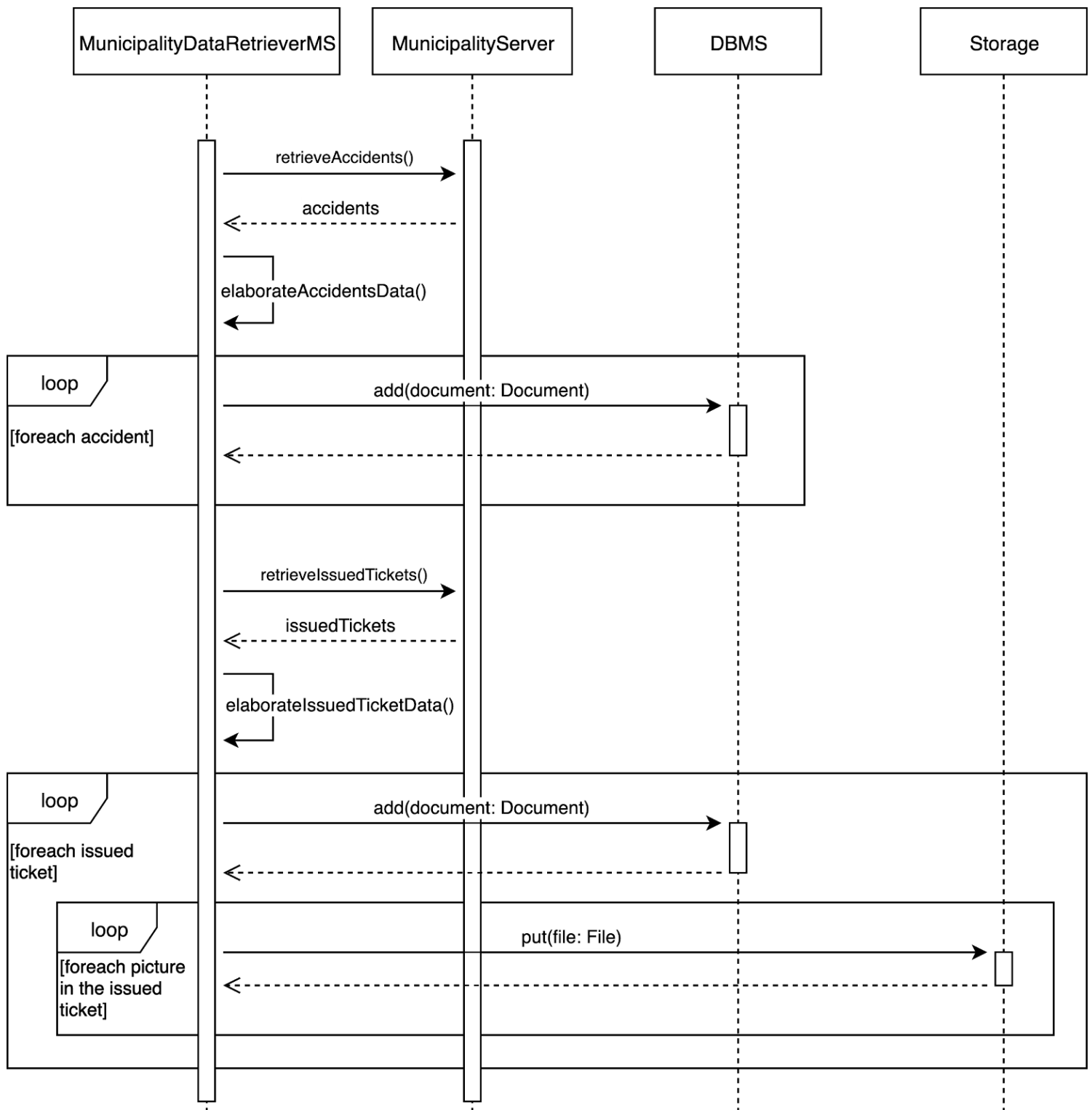
2.4.4. Visualizing SafeStreets Data for Users
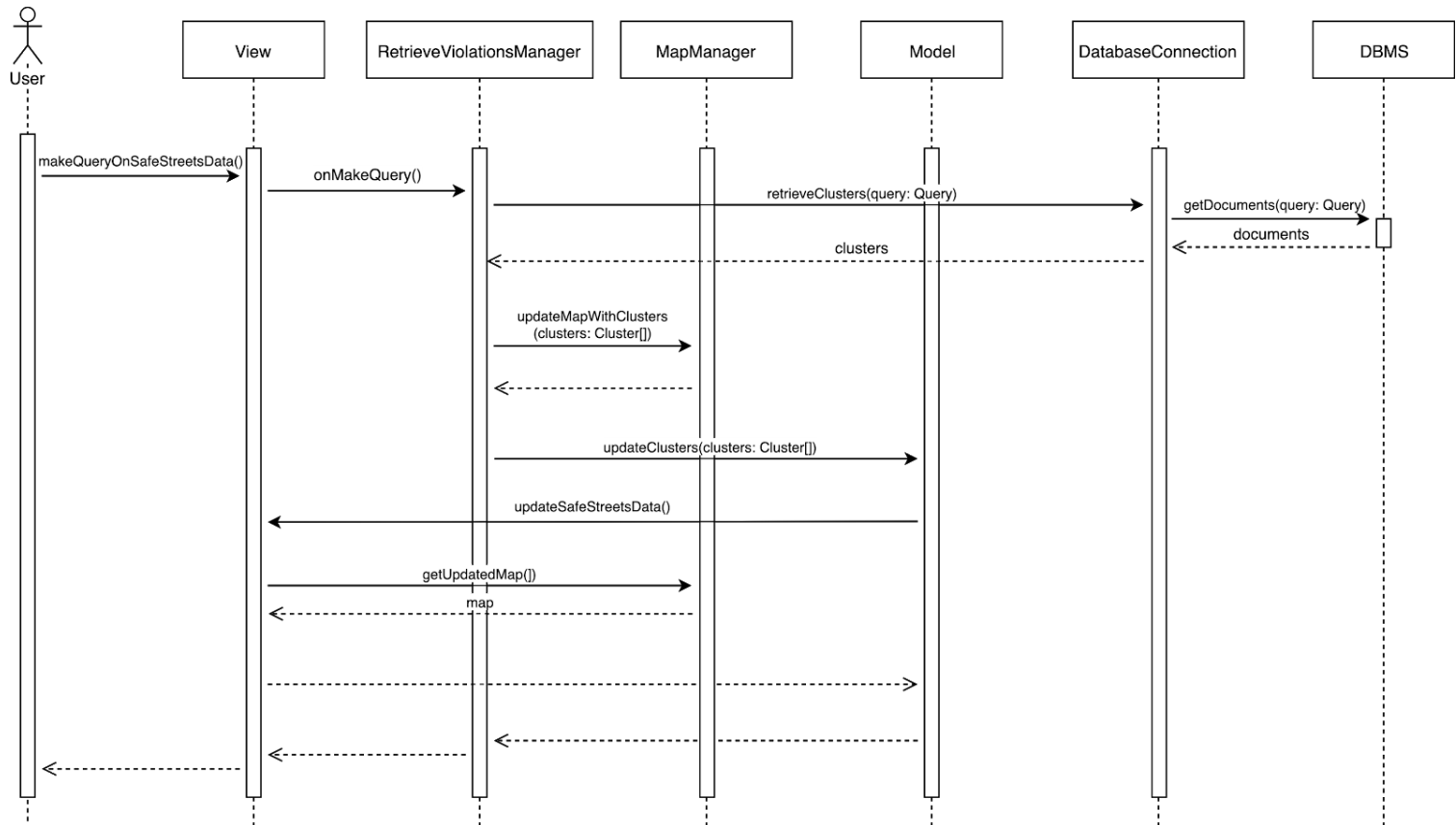
2.4.5. Visualizing SafeStreets Data for Municipalities
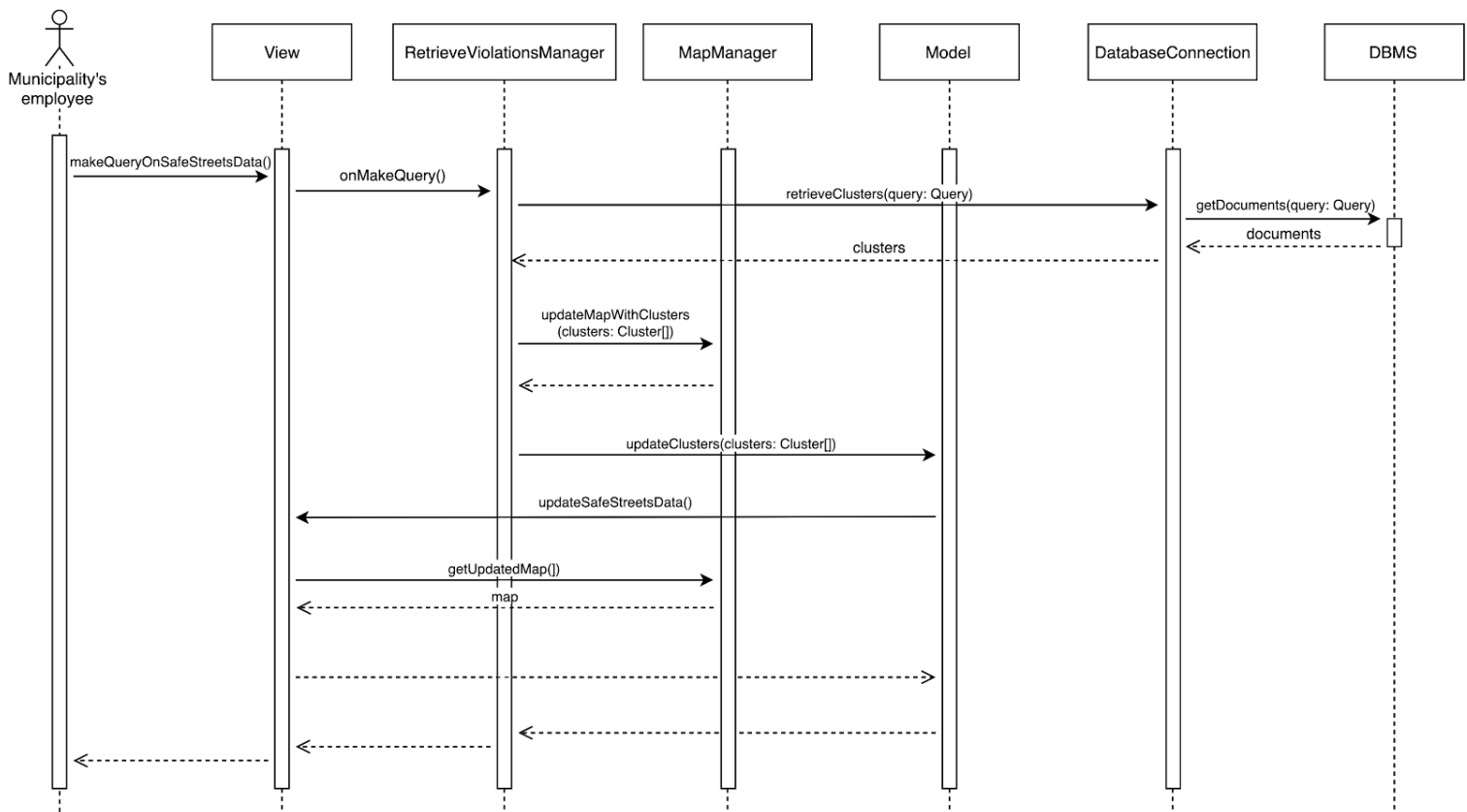
2.4.6. Cluster's details for the municipality
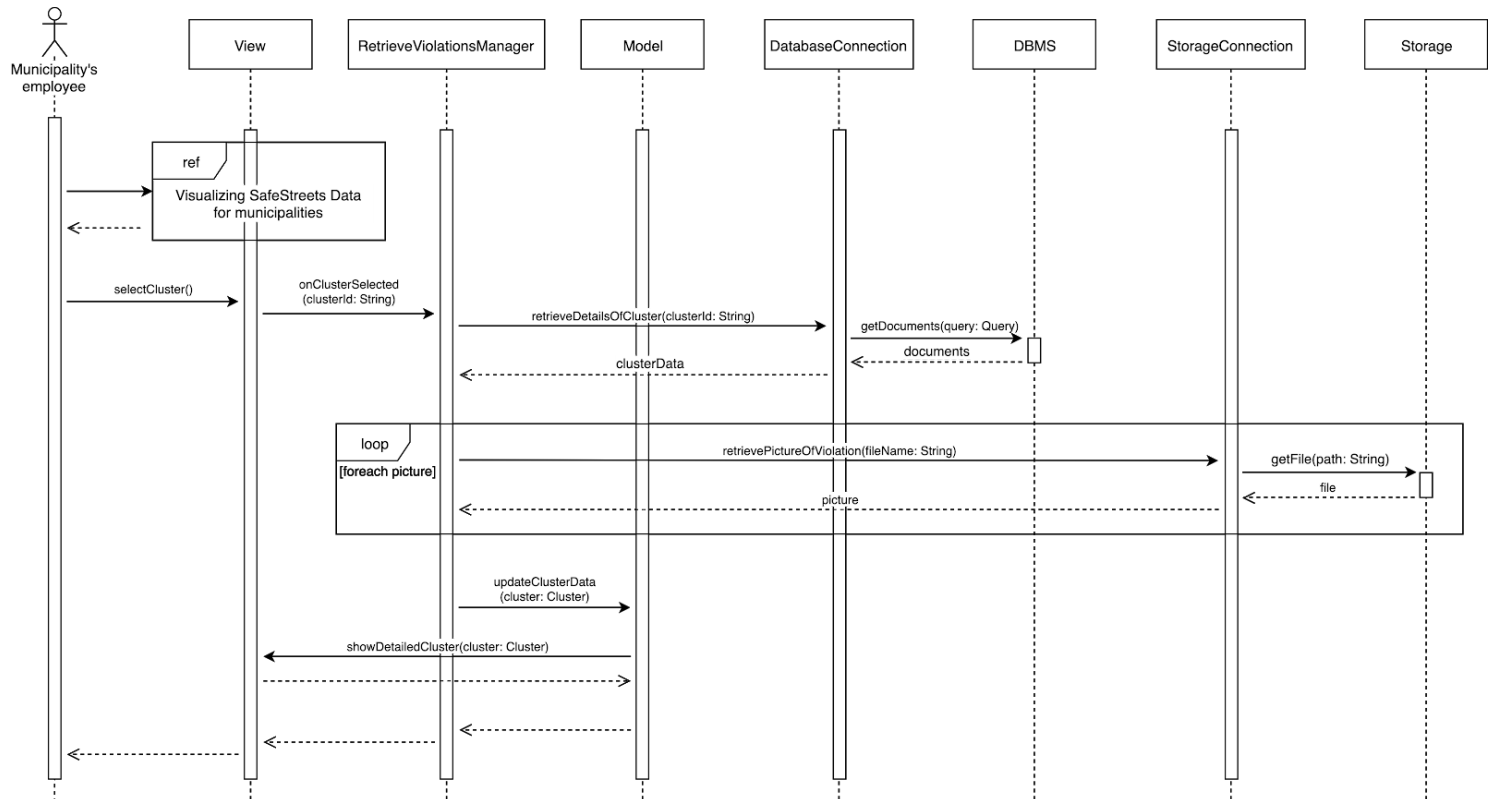
2.5. Component interfaces

2.5.1. Component interfaces for the SafeStreets App

App
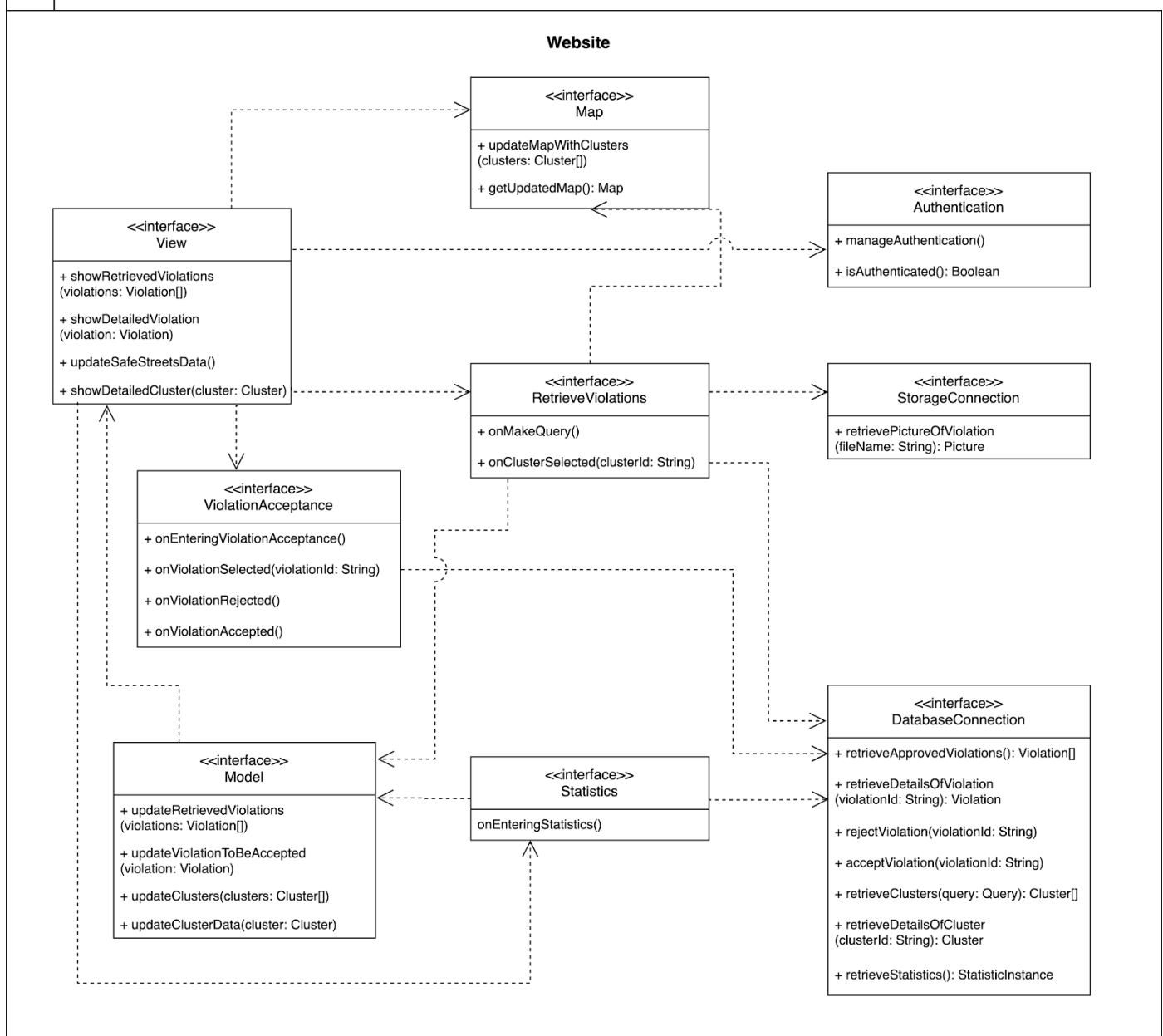
2.6. Selected architectural styles and patterns

Model–View–Controller[2]: it is a software design pattern commonly used for developing user interfaces which divides the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to the user.

Serverless computing: it is a cloud-computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Serverless computing can simplify the process of deploying code into production, allowing us to concentrate on more important matters.

2.7. Other design decisions

2.7.1. Data Structure

Data will be stored in a NoSql Database, so its structure is crucial in order to have fast and easy-to-write queries. This can be achieved, for example, by duplicating information among the files of the DB.

To understand the overall structure is useful to introduce the entity-relationship diagram that will help identify the entities and their relationships and attributes even though it will not represent the actual DB structure.
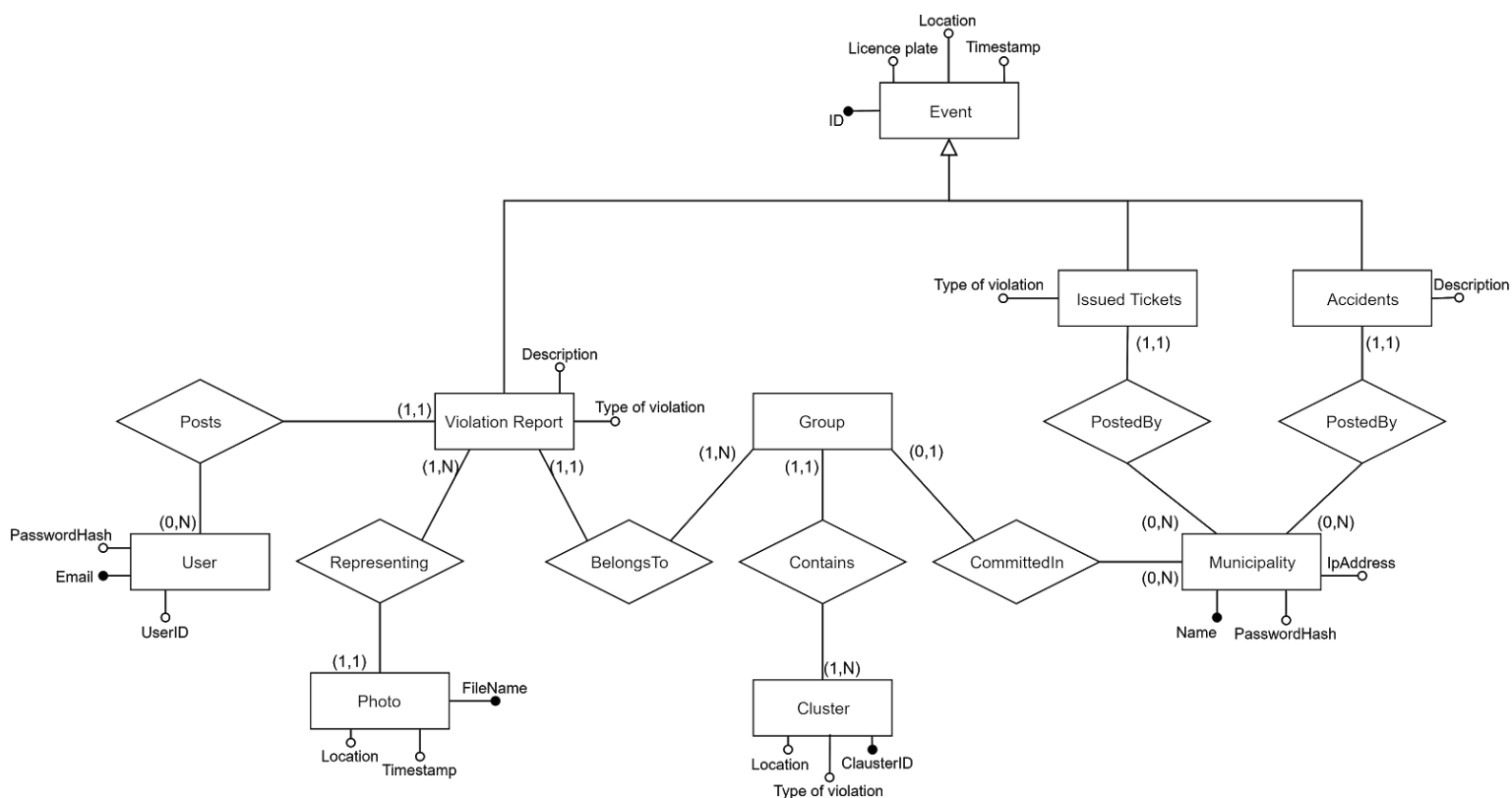
Every *Violation Report* is associated with a *User*, at least one *Photo* and a *Group*. A *Group* is the set of all the *Violation Reports* that represent the same real violation, i.e. the set of violations with the same licence plate, the same type of violation, relatively close locations and timestamps. In order to propose suggestions and optimize the visualization of the marks on the map, close *Groups* that have the same type of violation are grouped together creating a *Cluster*.

Furthermore, each of these entities can be associated with a unique *Municipality* that, in addition, can post to the server its *Issued tickets* and its *Accidents*.

The following tree structure that will be implemented in the database captures all these aspects.

```
┌─────────────────────┐
│   Municipalities    │
└──────────┬──────────┘
           │
           ▼
      ┌─────────────────────┐
      │    Municipality     │
      ├─────────────────────┤
      │  + Name: String     │
      └─────────────────────┘
```

Municipalities → **Municipality**
+ Name: String

Report
+ Plate: String
+ TypeOfViolation: String
+ Location: Geopoint
+ Timestamp: Time
+ Status: String
+ PhotosID: String[]
+ UserID: String

Group
+ LicencePlayte: String
+ TypeOfViolation: String
+ Location: Geopoint
+ Status: String
+ ReportIDs: String[]

Cluster
+ TypeOfViolation: String
+ Location: Geopoint
+ GroupIDs: String[]
+ TypeOfViolation: String

Accidents

IssuedTickets

Accident
+ Plate: String
+ Location: String
+ Timestamp: Time
+ Description: String

IssuedTickets
+ Plate: String
+ TypeOfViolation: String
+ Location: String
+ Timestamp: Time

2.7.2 External APIs

Maps API*:* we use a geolocation service to display data about violations and accidents in the map.

Image Recognition service*:* this is a powerful tool that will be used in order to automatically approve reports; in particular, we will automatically discard the reports that don't have at least one vehicle in the photos. This is in order to avoid sending to the municipality unwanted reports.

Municipality API: if the municipality offers a service where we can retrieve data about violations, ticket issued and accidents on its area of operation, we are able to cross this data with ours to provide more accurate statistics and suggestions.

3. User interface design

3.1. SafeStreets App

In this section it is shown the user interface design applied to the SafeStreets application.
First UX diagrams are presented where the flow followed by the users to utilize the app is shown. There are two UX diagrams to better represent different parts of the application: one for the authentication process of the user and the other for the main features of the app that are available only after the authentication process.
Then the mockups are presented. The mockups presented here were already developed for the RASD document, but are copied for readability purposes and to associate them with their corresponding page in the UX diagram.
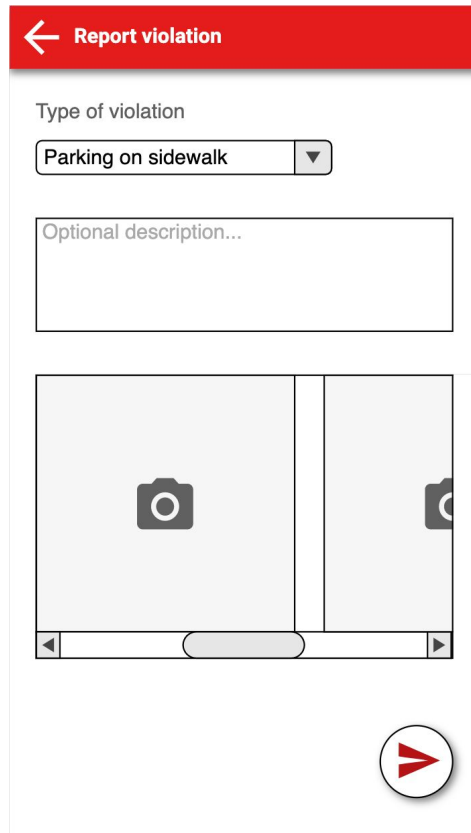
3.1.1. User authentication - UX diagram
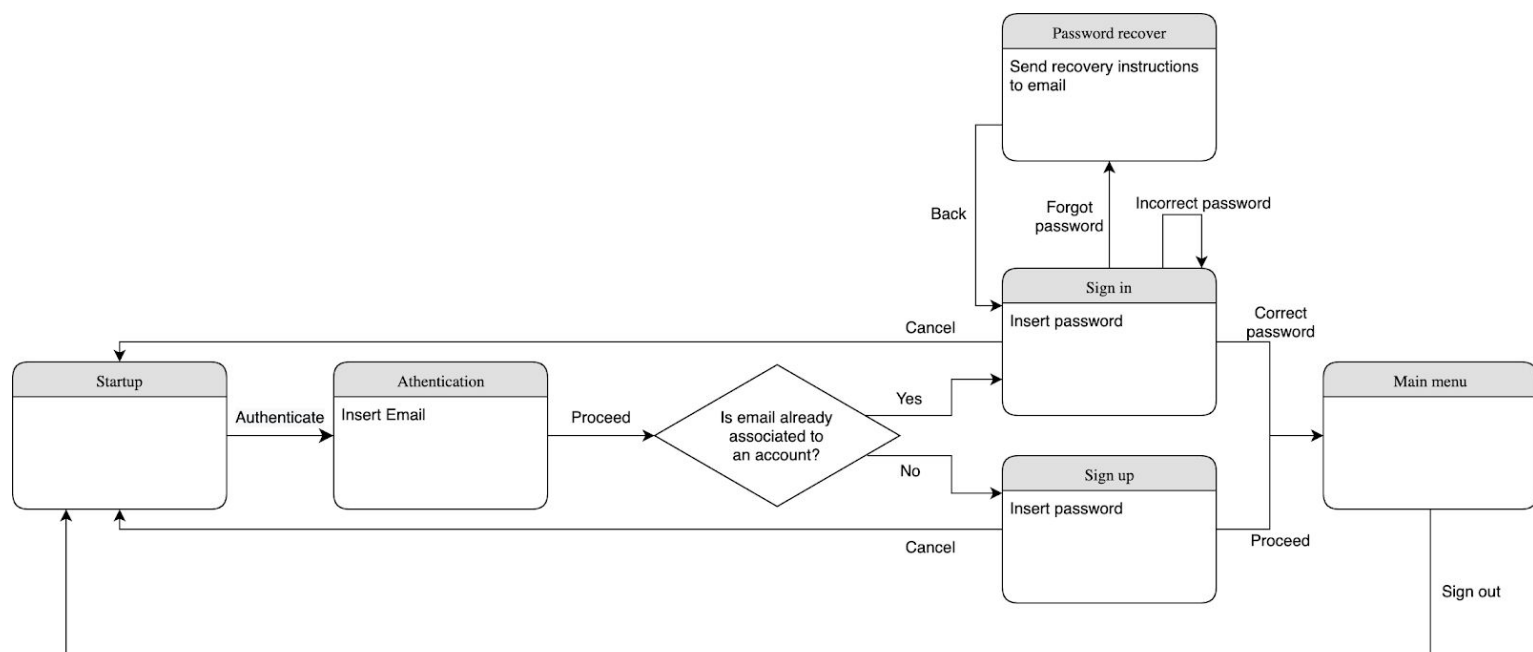
3.1.2. App features - UX diagram
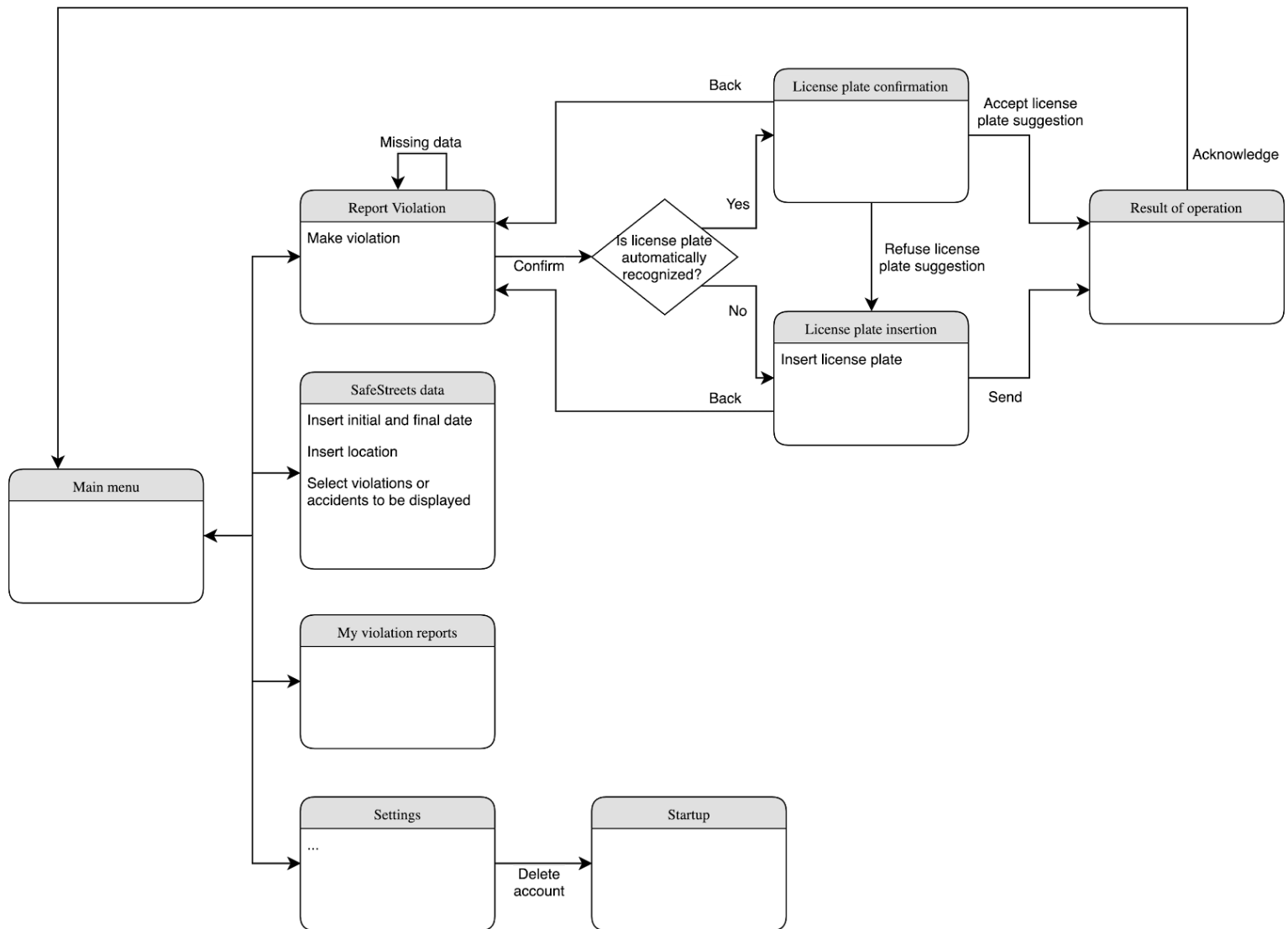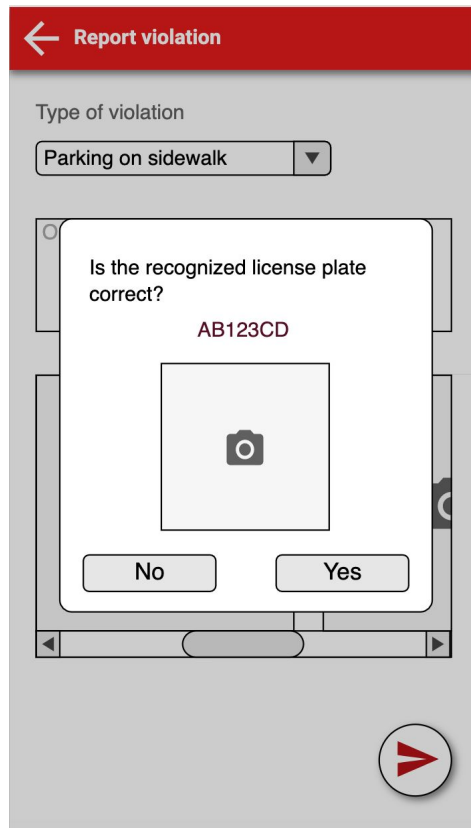
3.1.3. Mockup of the "Report Violation" page

3.1.4. Mockup of the "License plate confirmation" page

3.1.5. Mockup of the "License plate insertion" page

3.1.6. Mockup of the "SafeStreets data" page

3.2. Municipality web interface

In this section it is shown the user interface design applied to the SafeStreets web interface for municipalities.
First it is presented the UX diagram were it is shown the flow followed by the employees of the municipalities, then the mockups are presented.

3.2.1. UX diagram
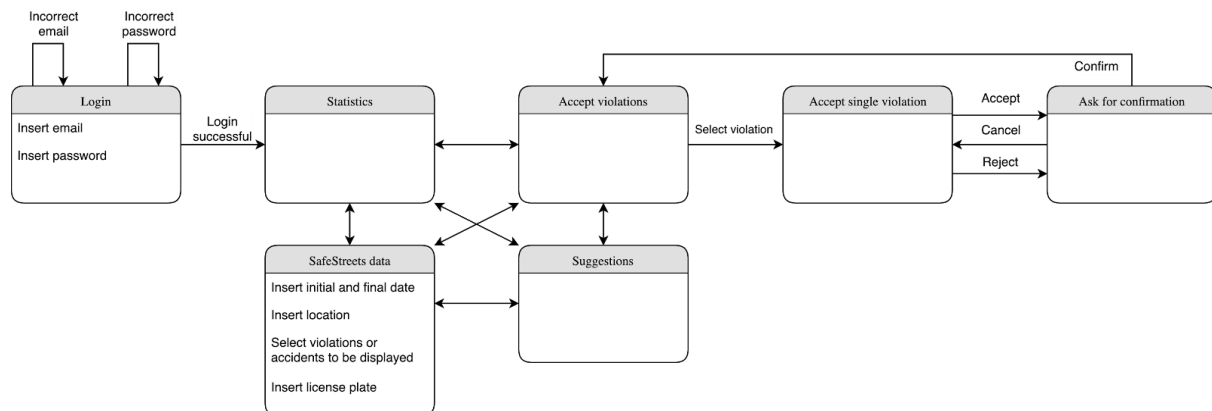
In this diagram the logout feature is not represented since it is possible to execute a logout in any page, and by doing so the web interface exits the page in which the user is in and goes to the "Login" page.

3.2.2. Mockup of the "Login" page

3.2.3. Mockup of the "Accept violations" page

```
Accept violations
← → ↻   https://safestreets-polimi.web.app

SafeStreets        Order by  [ Number of reports  ▼ ]

Accept violations                Violations
Display data
Show statistics    License plate: DC321BA                          ▲
                   Location: Via Respiratoria
                   Violation type: "Parking on crosswalk"
                   Number of reports: 32

                   License plate: AB123CD
                   Location: Via Francesco Petrarca
                   Violation type: "Parking on handicapped spot without permit"
                   Number of reports: 20

                   License plate: DA111NT
                   Location: Via Dante Alighieri
                   Violation type: "Parking in a no-parking zone"
                   Number of reports: 15

                   License plate: RU888UN
                   Location: Corso Fuori
                   Violation type: "Parking on sidewalk"
     Logout         Number of reports: 11                           ▼
```

3.2.4. Mockup of the "Accept single violation" page

```
Accept violation
← → ↻   https://safestreets-polimi.web.app

SafeStreets        Group id: 553432

Accept reports     Number of reports: 32
Display data       Location: Via Respiratoria (44.6588696, 8.3219011)
Show statistics    License plate: DC321BA

                   Violation type: "Parking on crosswalk"

                   Time: from 8:30 to 17:30

                      Descriptions:           Pictures:

                   ┌──────────────┐▲  ┌──────────────────┐▲
                   │ The driver   │   │ [img]    [img]   │
                   │ parked       │   │                  │
                   │ on the       │   │                  │
                   │ crosswalk.   │   │                  │
                   │ Crossing the │   │                  │
                   │ street was   │   │                  │
                   │ very         │   │ [img]    [img]   │
                   │ dangerous.   │   │                  │
                   │              │   │                  │
                   │ Please make a│   │                  │
                   │ ticket to    │   │                  │
                   │ this         │   │ [img]    [img]   │
                   │ imbecile.    │   │                  │
                   │              │   │                  │
                   │ Ticket!!     │▼  └──────────────────┘▼

                      [ Reject ]           [ Accept ]
     Logout
```

3.2.5. Mockup of the "SafeStreets data" page

SafeStreets data

← → ↻ https://safestreets-polimi.web.app

SafeStreets

Accept reports

Display data

Show statistics

Search location...   Search license plate...

Start date   End date

27/11/2019 📅 ▼   09/12/2019 📅 ▼

Display...

☑ Violations   Select types

☑ Accidents

Show as list

Logout

4. Requirements traceability

- [R1] A citizen not yet registered must be able to sign up and become a user.
  - AuthenticationManager (App)
  - Authentication System

- [R2] The application must allow users to authenticate through log in.
  - AuthenticationManager (App)
  - Authentication System

- [R3] The application must allow users to report a violation.
  - ReportViolationManager (App)
  - AuthenticationManager (App)
  - Authentication System

- [R4] The application must allow reporting of violations only from devices equipped with a GPS receiver which are in the conditions to obtain a GPS fix.
  - ReportViolationManager (App)

- [R5] The application must allow reporting of violations only from devices equipped with a camera.
  - ReportViolationManager (App)

- [R6] The application must allow reporting of violations only from devices with an active internet connection.
  - ReportViolationManager (App)

- [R7] A user has the possibility to specify the type of the reported violation choosing from a list.
  - ReportViolationManager (App)

- [R8] The application creates a violation report with at least one picture, exactly one timestamp, exactly one location, exactly one type of violation and the license plate of the vehicle.
  - ReportViolationManager (App)

- [R9] The system saves all the information regarding the violations reported by users.
  - DBMS
  - Data Storage
  - StorageConnetion (App)
  - DatabaseConnection (App)

- ○ ReportViolationManager (App)

- [R10] The system must be able to retrieve data regarding issued tickets and accidents from the municipality.
  - ○ DBMS
  - ○ MunicipalityDataRetrieverMS

- [R11] The system must offer an interface to retrieve information about violations, accidents and issued tickets.
  - ○ Hosting
  - ○ RetrieveViolationsManager (Web)
  - ○ RetrieveViolationsManager (App)
  - ○ DBMS
  - ○ StorageConnetion (Web)
  - ○ DatabaseConnection (Web)
  - ○ StorageConnetion (App)
  - ○ DatabaseConnection (App)

- [R12] The system must show to users only anonymized data.
  - ○ DBMS
  - ○ RetrieveViolationsManager (App)
  - ○ DatabaseConnection (App)

- [R13] The system must analyze valid violations report and approve which of them may represent a correct violation.
  - ○ ApprovingMS

- [R14] The system must be able to elaborate data about violations, accidents, issued tickets and generate useful suggestions about possible interventions.
  - ○ ClusteringMS
  - ○ GroupingMS
  - ○ DBMS

- [R15] The system must offer an interface to the municipality for retrieving useful suggestions about possible interventions.
  - ○ Hosting
  - ○ MapManager (Web)
  - ○ RetrieveViolationsManager (Web)

- [R16] The system must offer a service to the municipality for retrieving approved violations report.
  - ○ DBMS
  - ○ Data Storage
  - ○ RetrieveViolationsManager (Web)

- ○ StorageConnetion (Web)
- ○ DatabaseConnection (Web)

- ● [R17] The application will allow using pictures in a violation report only if the picture was taken by the application itself, preventing it to be manipulated on the device.
  - ○ ReportViolationManager (App)

- ● [R18] All connections used by the system use modern encryption protocols.
  - ○ StorageConnetion (Web)
  - ○ DatabaseConnection (Web)
  - ○ StorageConnetion (App)
  - ○ DatabaseConnection (App)

- ● [R19] Data saved in the server cannot be manipulated.
  - ○ DBMS

- ● [R20] The system must be able to elaborate data about violations, accidents, issued tickets and generate useful statistics for each municipality.
  - ○ StatisticsManager (Web)

- ● [R21] The system must offer an interface to the municipality for retrieving statistics.
  - ○ DBMS
  - ○ DatabaseConnection (Web)
  - ○ StatisticsManager (Web)

5. Implementation, integration and testing

5.1 Feature identification

The following table specifies the main features of our system, complete with their importance and their difficulty of implementation in order to better plan the effort spent testing the most critical ones.

	Feature	Relevance	Difficulty of implementation
1	(User) Report violations	High	Medium
2	Visualize data	High	High
3	(User) See own violation	Medium	Low
4	(Municipality) Retrieve and review violations	High	High
5	(Municipality) Visualize statistics	Medium	Medium

Feature 1: Report violations
This feature is indeed the foundation on which the app is built. Only when a large pool of data is collected, the users will be able to use the application to retrieve and visualize data.
The difficulty of implementing this is Medium because, as already said, the task of filtering unwanted reports is done by the microservices on the cloud. The difficulty comes from developing security measures that can avoid manipulating pictures on the device, and ensuring the validity of the report also through image recognition.

Feature 2: Visualize data
This feature brings to the user and the municipality the possibility to visualize violations on the map (and possibly data from the municipality).
The system will interface with a mapping API, necessary to visualize the map. Furthermore, the system will be entrusted with drawing inferences from the big amount of data collected, both from the users and the municipality.
For the case of the municipality also suggestions are visualized together with the clusters of the violations.

Feature 3: See own violation
This function will give the user the possibility to see their own violation and their status.

Feature 4: Retrieve and review violations

This is another key point of the system. Using a web interface, the municipality can connect to the SafeStreets Cloud and, after authenticating, it will be able to review violation reports submitted by the users. The system will then update the status of the groups based on the decision of the municipality.

Embedded with the reports, the municipality can also read the most suitable suggestion for improving the safety of the street.

The difficulty of building this feature is "high" because it requires a whole subsystem to be built.

Feature 5: Visualize statistics

Using this function, the municipality will be able to visualize useful statistics such as the effectiveness of the SafeStreets initiative, the increase in tickets, the eventual decrease of traffic violations, weekly number of reports, ecc.

It's important to notice that Feature 1 and 3 are exclusive to the SafeStreets mobile app, whether Feature 4 and 5 are exclusive to the SafeStreets web interface. Feature 2 is common to the two subsystems.

5.2 Implementation, integration and testing plan

The implementation plan will follow a **bottom-up** approach. A bottom-up approach is the piecing together of systems to give rise to more complex systems.

The plan is to implement the reporting of violation first, along with Feature 3 since it is very easy. Then, the web interface for the municipality will be built so we can implement Feature 4. After that, Feature 2 will be implemented both for the app and the web interface. The statistics feature (Feature 5) can also be built in parallel with Feature 2.

This is the implementation, integration and testing plan of the features in order of priority.

Feature 1: Report violations

As said above, this feature is the core of the system and without this, nothing can function, because Feature 2, 3 and 4 rely on reports to work. So it is only natural that it will be the first feature to be implemented and tested.

For this function to work, the core features of the mobile application should be built. As a base, the "Model" and the "View" have to be implemented and tested, as well as the "AuthenticationManager", the "DatabaseConnection" and the "StorageConnection".

Then, the "ReportViolationManager" will be implemented, integrated with the previously stated components, and then tested.

On the Cloud side, the microservices "ApprovingMS", "GroupingMS" and "ClusteringMS" have to be implemented and tested.
In particular, the "ApprovingMS" has to be integrated with the Image Recognition API.

Feature 3: See own violation
For this simple feature, it is necessary to implement the "RetrieveViolationManager" component in the SafeStreetsApp, and integrate it with "StorageConnection" and "DatabaseConnection".

Feature 4: Retrieve and review violations
Before implementing this feature, the web app must be set up. The "Model" and the "View" have to be implemented and tested, along with the "AuthenticationManager", the "DatabaseConnection" and the "StorageConnection", similar to the application.
Then the "RetrieveViolationsManager" and "ViolationAcceptanceManager" have to be implemented, unit-tested and integrated with the aforementioned components.
On the Cloud side, the microservice "OnReportStatusChangeMS" has to be implemented in order to receive updates from municipality. The "MunicipalityDataRetrieverMS" can also be built at this point.

Feature 2: Visualize data
This feature is common to the two subsystem, so the approach to build it is similar.
The function is provided by the "MapManager" component, that will interface with the "RetrieveViolationsManager" and the "Maps API" to retrieve, organize, and display data on the map; a correct integration with these components is indeed crucial.

Feature 5: Visualize statistics
For this feature, it is necessary to implement the "StatisticsManager" module, unit-test it, and integrate it with the "DatabaseConnection" in order to retrieve the data on which to build statistics.

The following is a graph that highlights the order and dependencies between features.
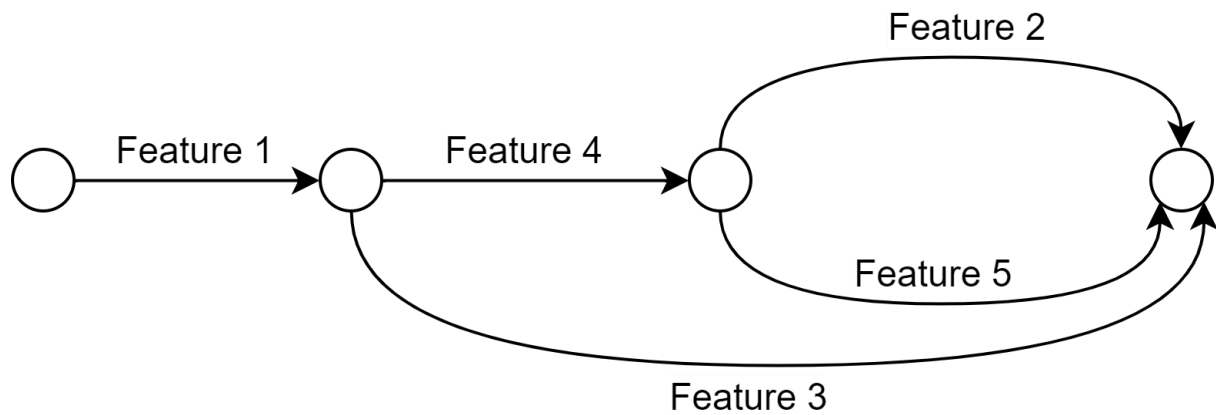
It is important to notice that the "View" component, both for the web interface and the mobile application, will interface with the "Manager" modules in order to let the user and the municipality interact with the system.
All the "Manager" components will in turn interface with the "Model" of the two subsystems.

6. Effort spent

Andrea Marcer		
Date	Task(s)	Time
22/11	Discussion	3 hr 0 min
24/11	Architecture diagram	1 hr 15 min
25/11	Component view	1 hr 0 min
28/11	Component view	1 hr 0 min
30/11	Relationships	2 hr
30/11	General overview	30 min
2/12	Component diagram	4h 30 min
2/12	Entity-Relationship diagram	1hr 30 min
3/12	Component diagram	1hr 45 min
3/12	Component diagram, Entity-Relationship diagram	1hr 45 min
3/12	Entity-Relationship diagram description	1hr
4/12	Entity-Relationship diagram description	30 min
4/12	General overview	3hr
5/12	General overview	1hr
6/12	General overview	2hr
7/12	General overview	1hr
8/12	Requirement traceability	1hr
9/12	General overview	30 min
Total:		28 hr 15 min

Matteo Marchisciana

Date	Task(s)	Time
22/11	Discussion	3 hr 0 min
24/11	Architecture diagram	1 hr 15 min
25/11	Component view	1 hr 0 min
28/11	Component view	1 hr 0 min
29/11	Component view	1 hr 30 min
30/11	Component view	4 hr 0 min
02/12	Implementation, integration and testing	2 hr 0 min
03/12	Implementation, integration and testing	4 hr 0 min
04/12	Implementation, integration and testing, architectural styles, design decisions, deployment view	5 hr 15 min
05/12	Deployment view	1 hr 45 min
06/12	Implementation, integration and testing, component view	1 hr 0 min
08/12	Review	0 hr 30 min
09/12	Review	1 hr 30 min
Total:		27 hr 45 min

Dennis Motta		
Date	Task(s)	Time
22/11	Discussion	3 hr 0 min
23/11	User interface design	2 hr 0 min
24/11	Architecture diagram	1 hr 15 min
24/11	User interface design	1 hr 0 min
25/11	Component view	1 hr 0 min
26/11	Architectural design overview	2 hr 0 min

27/11	Fixes after DD review; Architectural design overview	1 hr 45 min
28/11	Component view	1 hr 0 min
30/11	User interface design	2 hr 15 min
1/12	User interface design	1 hr 0 min
2/12	Component view; sequence diagram	2 hr 0 min
3/12	Sequence diagrams	3 hr 30 min
4/12	Sequence diagrams	1 hr 30 min
4/12	User interface design	0 hr 30 min
5/12	Sequence diagrams	1 hr 15 min
6/12	Component interfaces	2 hr 45 min
8/12	General overview	1 hr 0 min
9/12	General overview	1 hr 0 min
Total:		29 hr 45 min

7. References

1. "Serverless paradigm". https://en.wikipedia.org/wiki/Serverless_computing
2. "Model-View-Controller" https://en.wikipedia.org/wiki/Model-view-controller