



POLITECNICO MILANO 1863

Academic year 2019-2020
Software Engineering 2 Project

SafeStreets

Design Document

Version 1.0

Authors:

Marcer Andrea - 941276

Marchisciana Matteo - 945878

Motta Dennis - 940064

Professor:

Rossi Matteo

1. Introduction	7
1.1. Purpose	7
1.2. Scope	7
1.3. Definitions, Acronyms, Abbreviations	7
1.3.1. Definitions	7
1.3.2. Acronyms	8
1.3.3. Abbreviations	8
1.4. Revision history	8
1.5. Reference Documents	8
1.6. Document Structure	8
2. Architectural design	9
2.1. Overview	9
2.1.1. User's point of view	10
2.1.2. Municipality's point of view	11
2.1.3. Overall high level view of the architecture	11
2.2. Component view	12
2.2.1. Component Diagram of the Cloud	12
2.2.2. Component Diagram of the Microservices	13
2.2.3. Component Diagram of the Application	14
2.2.4. Component Diagram of the Municipality	15
2.3. Deployment view	16
2.4. Runtime view	17
2.4.1. User violation reporting	17
2.4.2. Municipality violation accepting	18
2.4.3. Retrieving of municipality's data	19
2.4.4. Visualizing SafeStreets Data	20
2.4.5. Suggestions for the municipality	20
2.5. Component interfaces	20
2.6. Selected architectural styles and patterns	20
2.7. Other design decisions	21
2.7.1. Data Structure	21
2.7.2 External APIs	23
3. User interface design	24
3.1. SafeStreets App	24
3.1.1. User authentication - UX diagram	24
3.1.2. App features - UX diagram	25
3.1.3. Mockup of the "Report Violation" page	26
3.1.4. Mockup of the "License plate confirmation" page	26
3.1.5. Mockup of the "License plate insertion" page	27
3.1.6. Mockup of the "SafeStreets data" page	27
3.2. Municipality web interface	28

3.2.1. UX diagram	28
3.2.2. Mockup of the “Login” page	28
3.2.3. Mockup of the “Accept violations” page	29
3.2.4. Mockup of the “Accept single violation” page	29
3.2.5. Mockup of the “SafeStreets data” page	30
4. Requirements traceability	31
3.2.4.1. Satisfying goal 1	34
3.2.4.2. Satisfying goal 2	34
3.2.4.3. Satisfying goal 3	34
3.2.4.4. Satisfying goal 4	34
3.2.4.5. Satisfying goal 5	35
3.2.4.6. Satisfying goal 6	35
5. Implementation, integration and testing	36
5.1 Feature identification	36
5.2 Implementation, integration and testing plan	37
6. Effort spent	39
7. References	42
Notes	42

1. Introduction

1.1. Purpose

TODO

1.2. Scope

As seen in the RASD, 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

- Entity: a “natural person”, or a “juridical person” in the case of the municipality.
- 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.
- UI: User Interface.
- S2B: Software To Be.
- GDPR: General Data Protection Regulation.
- OS: Operating System.
- RASD: Requirement Analysis and Specification Document
- HTTPS: Hypertext Transfer Protocol Secure
- HTML: Hypertext Markup Language

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

TODO

1.6. Document Structure

TODO

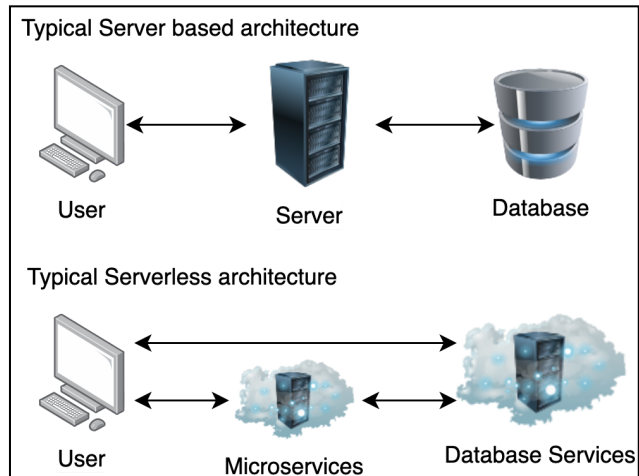
2. Architectural design

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 on 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.

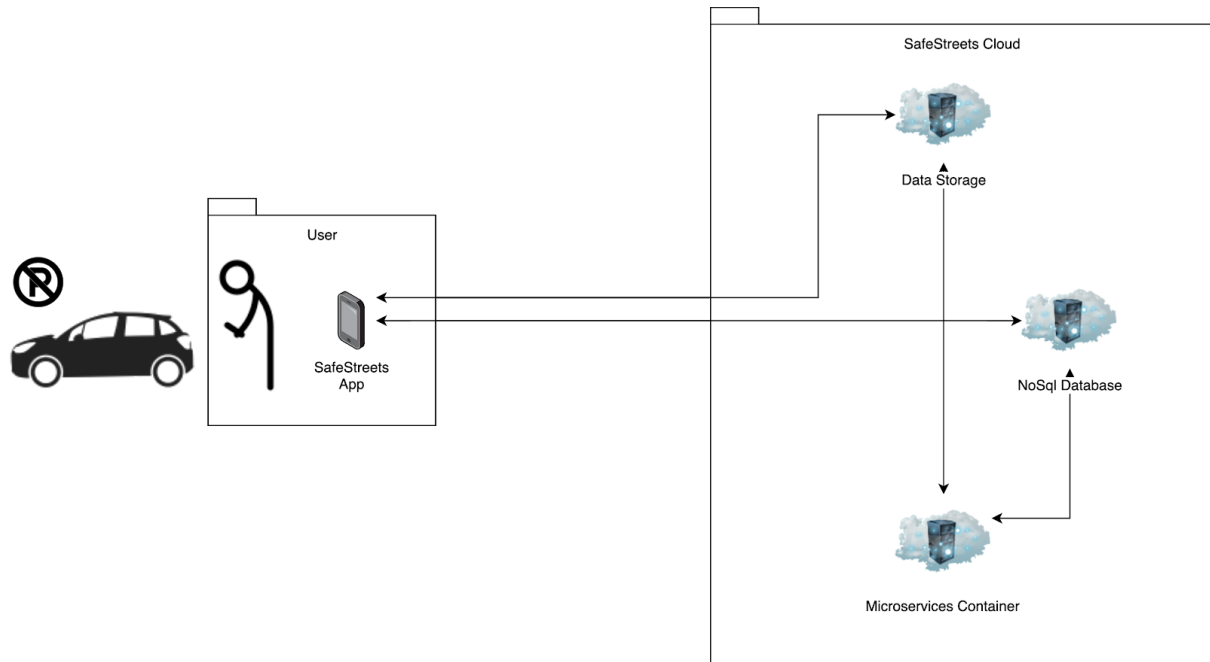


The architecture structure of SafeStreets can be seen from two different points of view: the one of the user and the one of the municipality.

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 for allowing a better understanding of the 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.

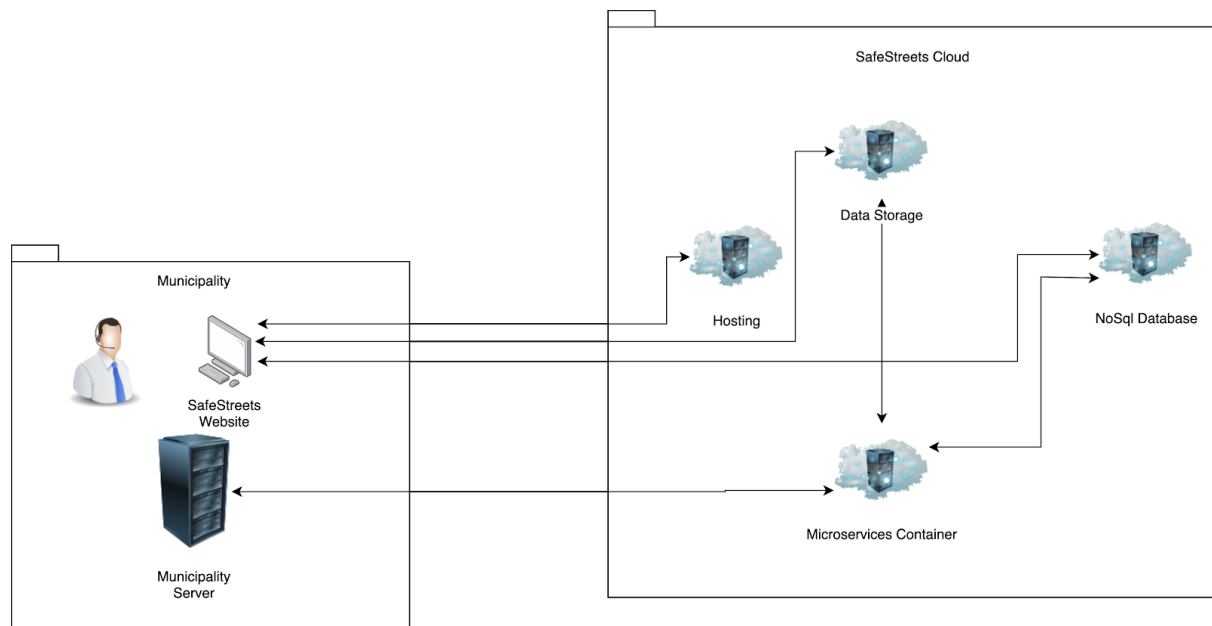
2.1.1. User's point of view



The user mainly interacts with 3 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.
- **Microservices Container:** this node contain stateless functions that are 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 storage while also interacting with the user.

2.1.2. Municipality's point of view

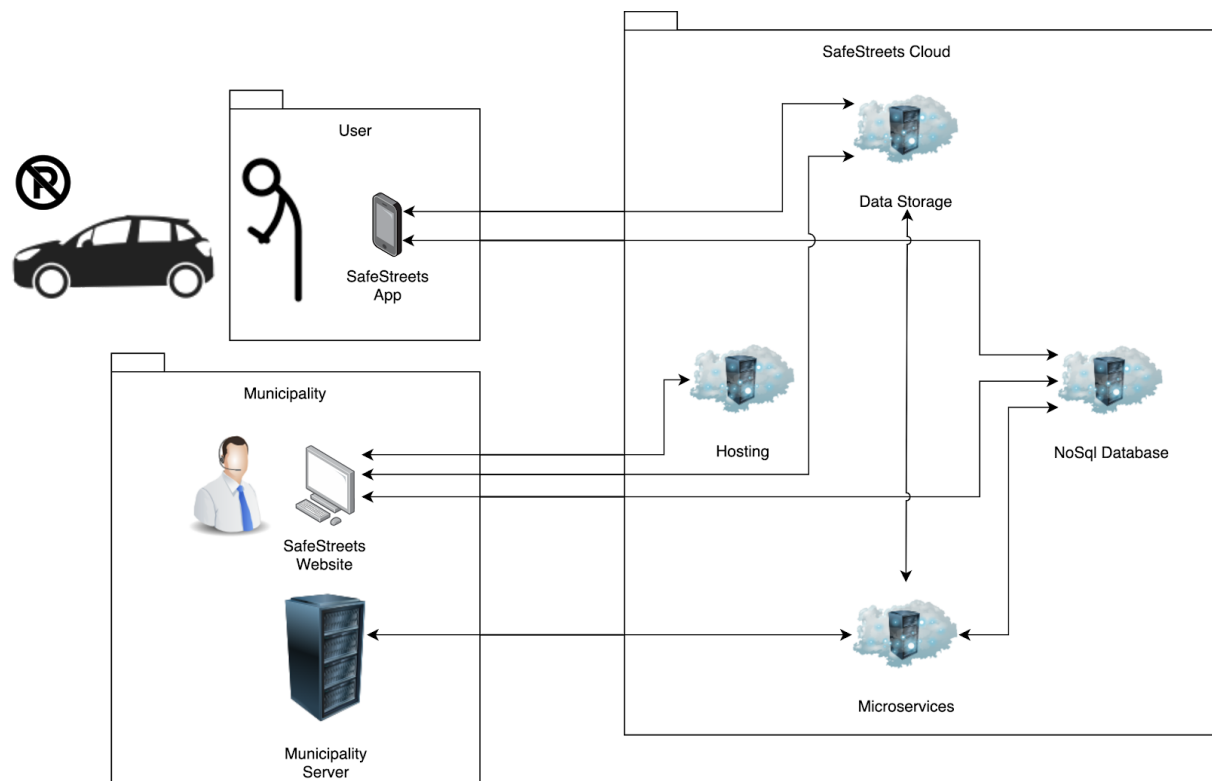


The municipality, similarly to the user, utilizes the *Data Storage*, the *NoSql Database* and the *Microservice Container* nodes, but now their behaviour depends on the actions of the municipality.

In order to interact with the cloud the municipality will also use another node:

- **Hosting Provider:** this node is a simple server that listens for connections on port 443 (HTTPS) and serves static HTML pages to the municipality.

2.1.3. Overall high level view of the architecture

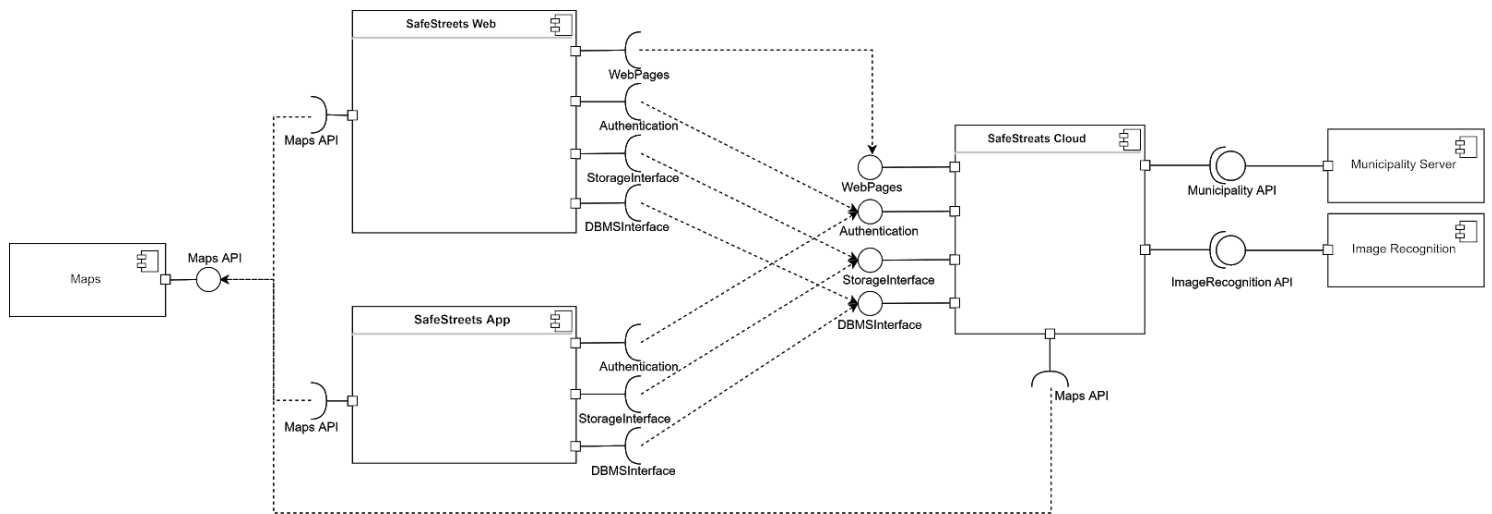


2.2. Component view

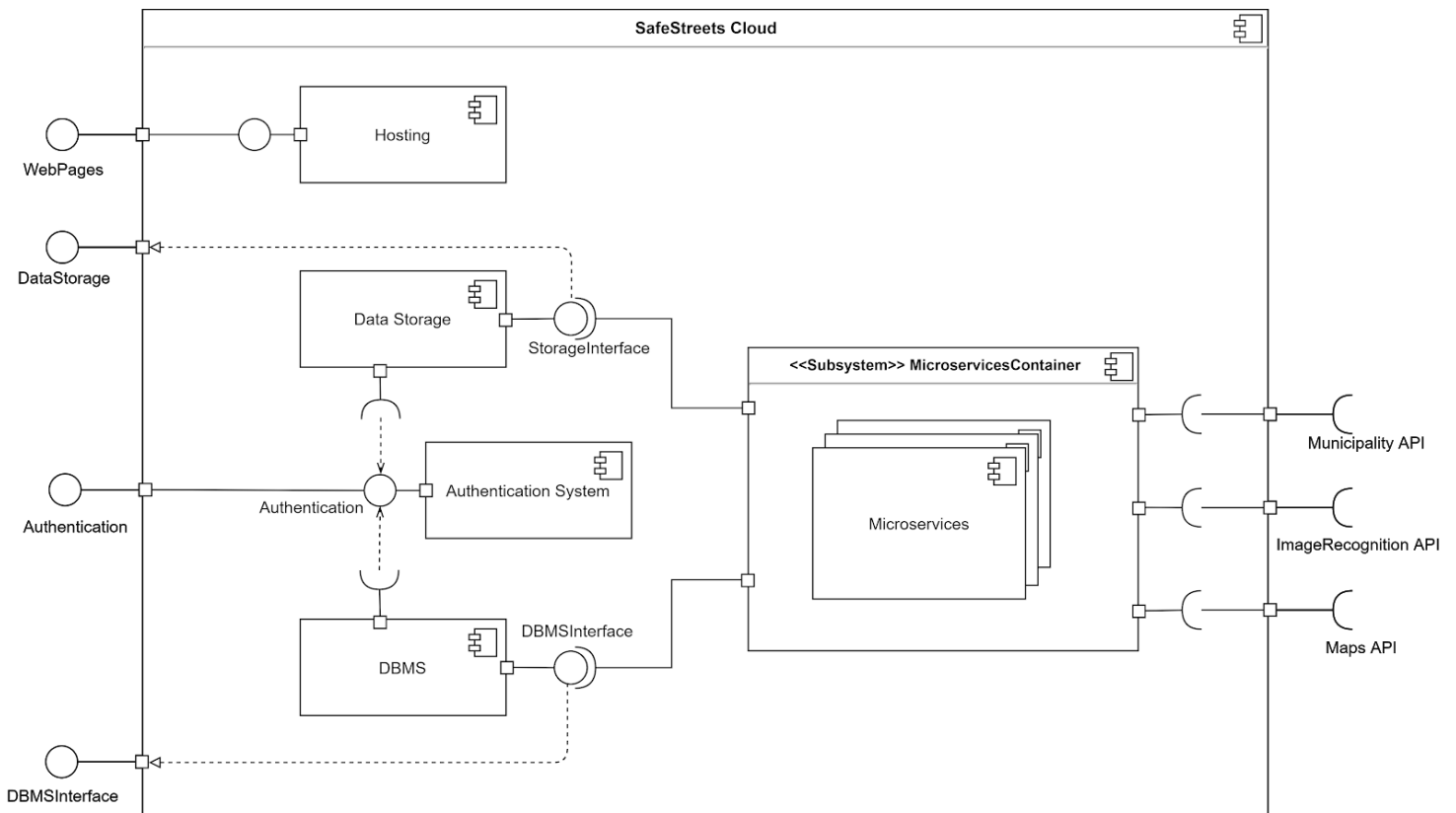
The following diagrams show the main components of the system and the interfaces through which they interact.

We divided the architecture into three subsystems:

- SafeStreets Cloud
- SafeStreets App
- SafeStreets Web



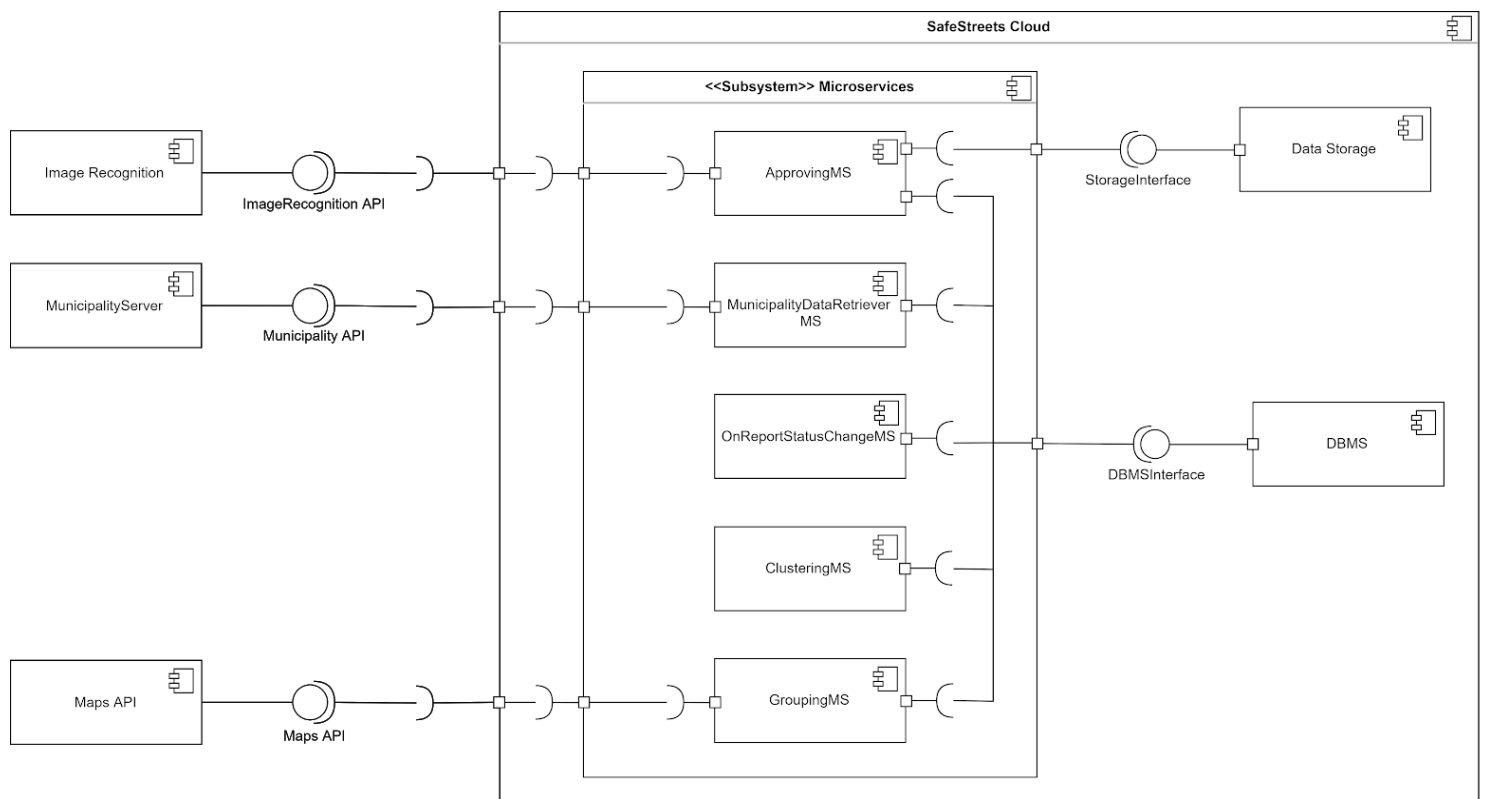
2.2.1. Component Diagram of the Cloud



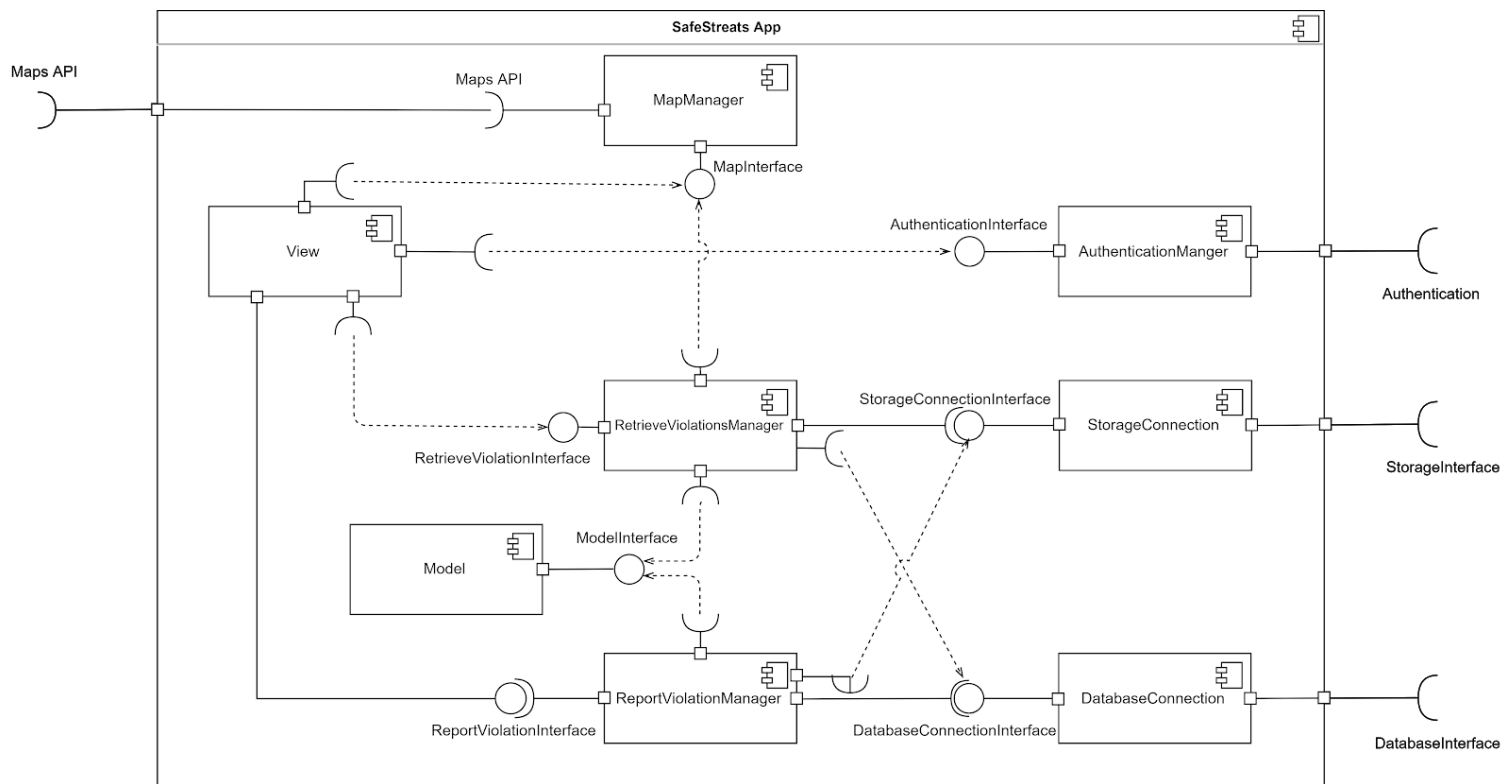
The cloud architecture is simple and has only a few components:

- The *hosting service*, which hosts the web interface.
- The *authentication system*, used both by the SafeStreets App and the web interface.
- The *MicroservicesContainer*. This is a collection of Microservices that are used to manage the subsystem and to provide useful APIs to both users and municipality. It is the most important and complex component of this subsystem and will be analyzed in detail later in the document.
- The *DBMS*, which manages the database.
- The *data storage*, in which we mainly store pictures of violations.

2.2.2. Component Diagram of the Microservices



2.2.3. Component Diagram of the Application

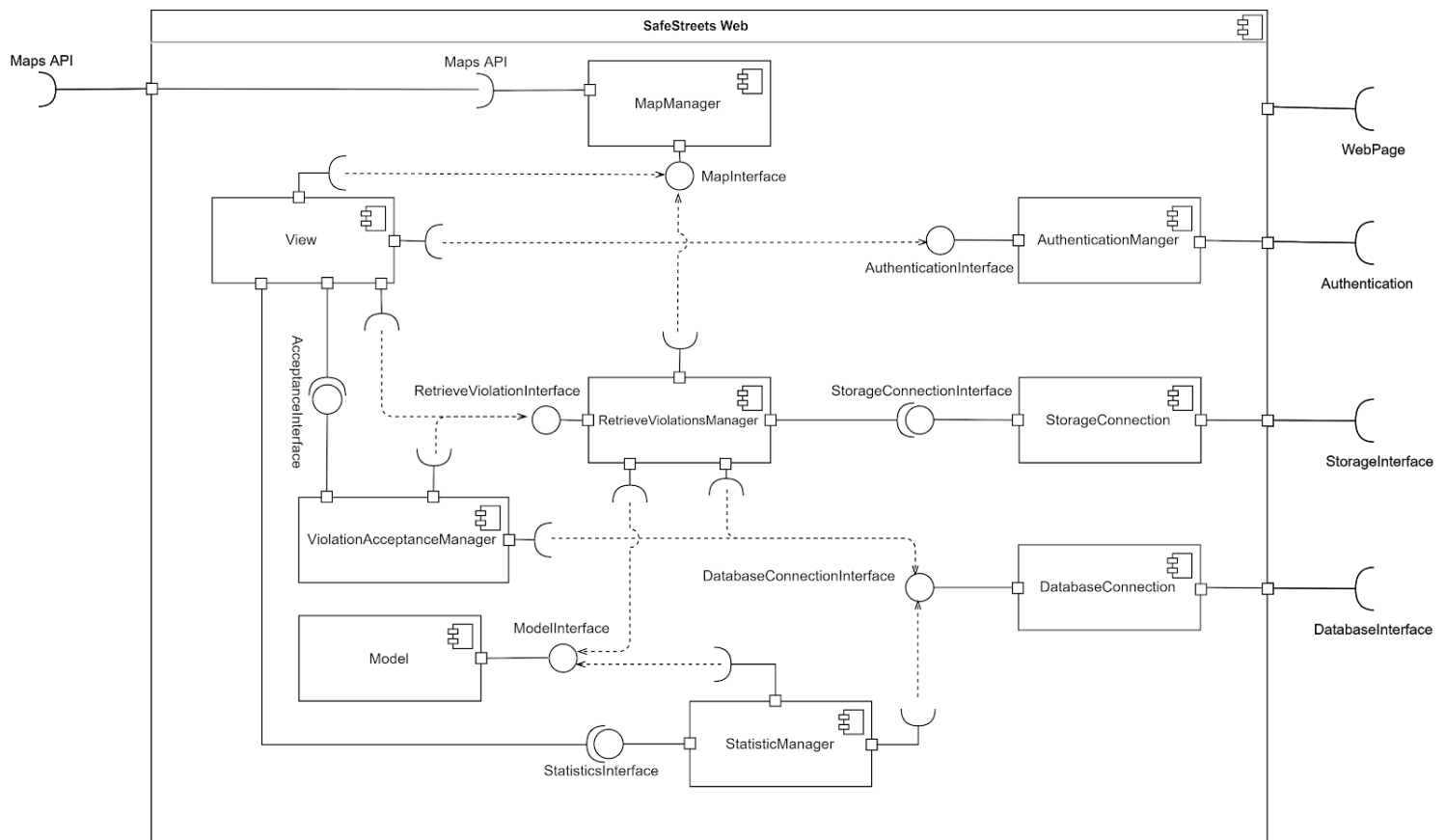


The SafeStreets mobile application is built using the Model - View - Controller design pattern. In this diagram, however, we wanted to highlight the role of the Controller and how it interacts with the various components of the cloud architecture; the Model and the View will be analyzed later.

The Controller has the following components:

- **ReportViolationManager:** the main function 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:** connects to the DBMS.
- **StorageConnection:** connects to the data storage.
- **MapManager:** uses the StatisticsManager and the Maps API to display violations and other data on the map.
- **RetrieveViolationsManager:** *non si è ancora capito sinceramente TODO: Write this component that handles all the queries.*

2.2.4. Component Diagram of the Municipality



The web interface offered to the municipality. This has a similar structure to the application with the obvious difference that the view will be an HTML web page. It has, however, an extra component.

- **StatisticsManager**: it uses the statistics microservice to provide useful statistics to 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

The system will use some external services to grant and enhance the service. They are the following:

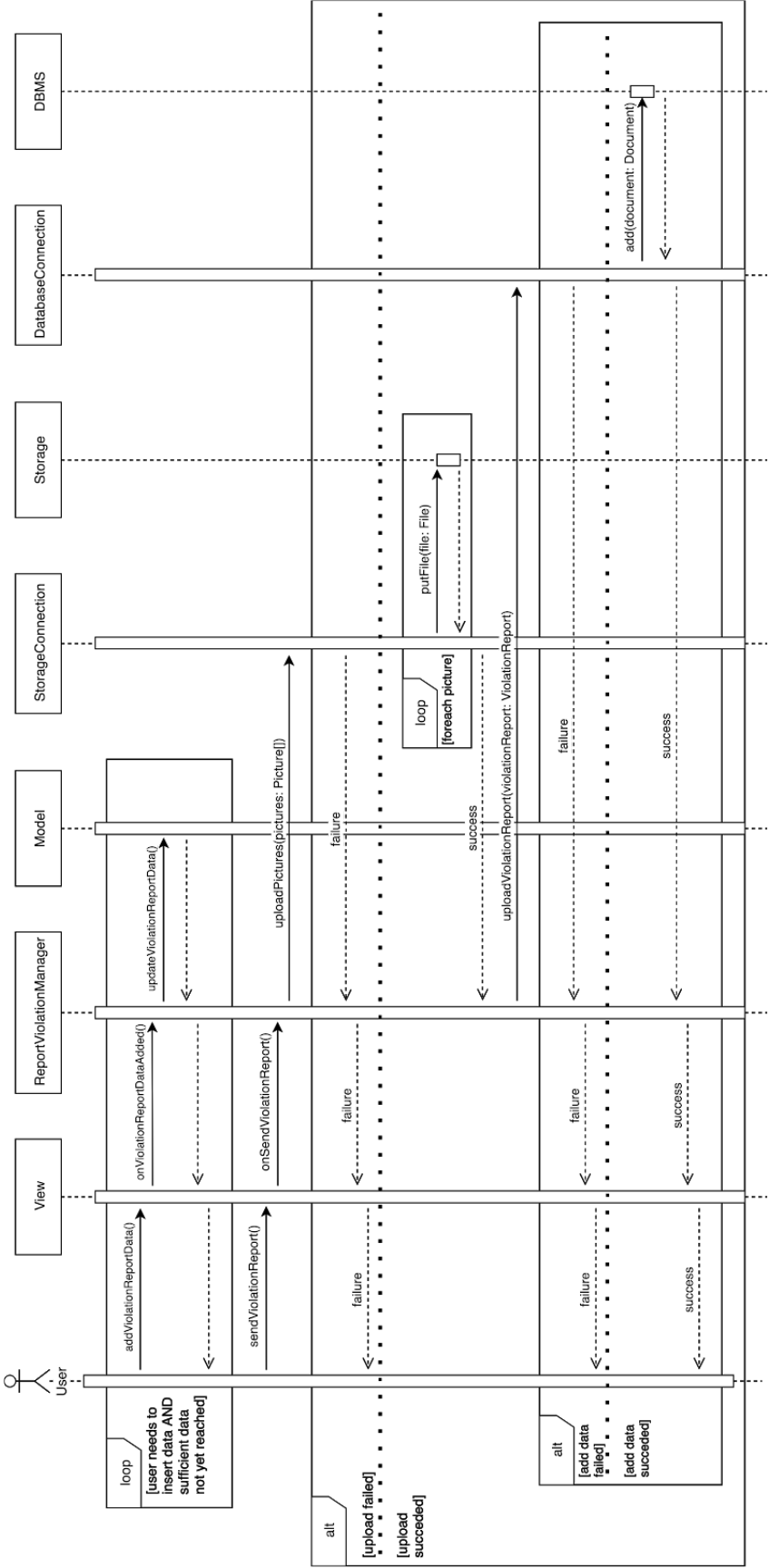
- **Maps API**: we use a geolocation service to display data about violations and accidents in the map.
- **Image Recognition service**: this is a powerful tool that will be used in order to automatically validate the report; in particular, we will automatically discard the reports that don't have at least one vehicle in the photos. This is in order to avoid storing unwanted data, to prevent flooding attacks to the system and to increase efficiency.
- **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, both to the user and to the municipality.

2.3. Deployment view

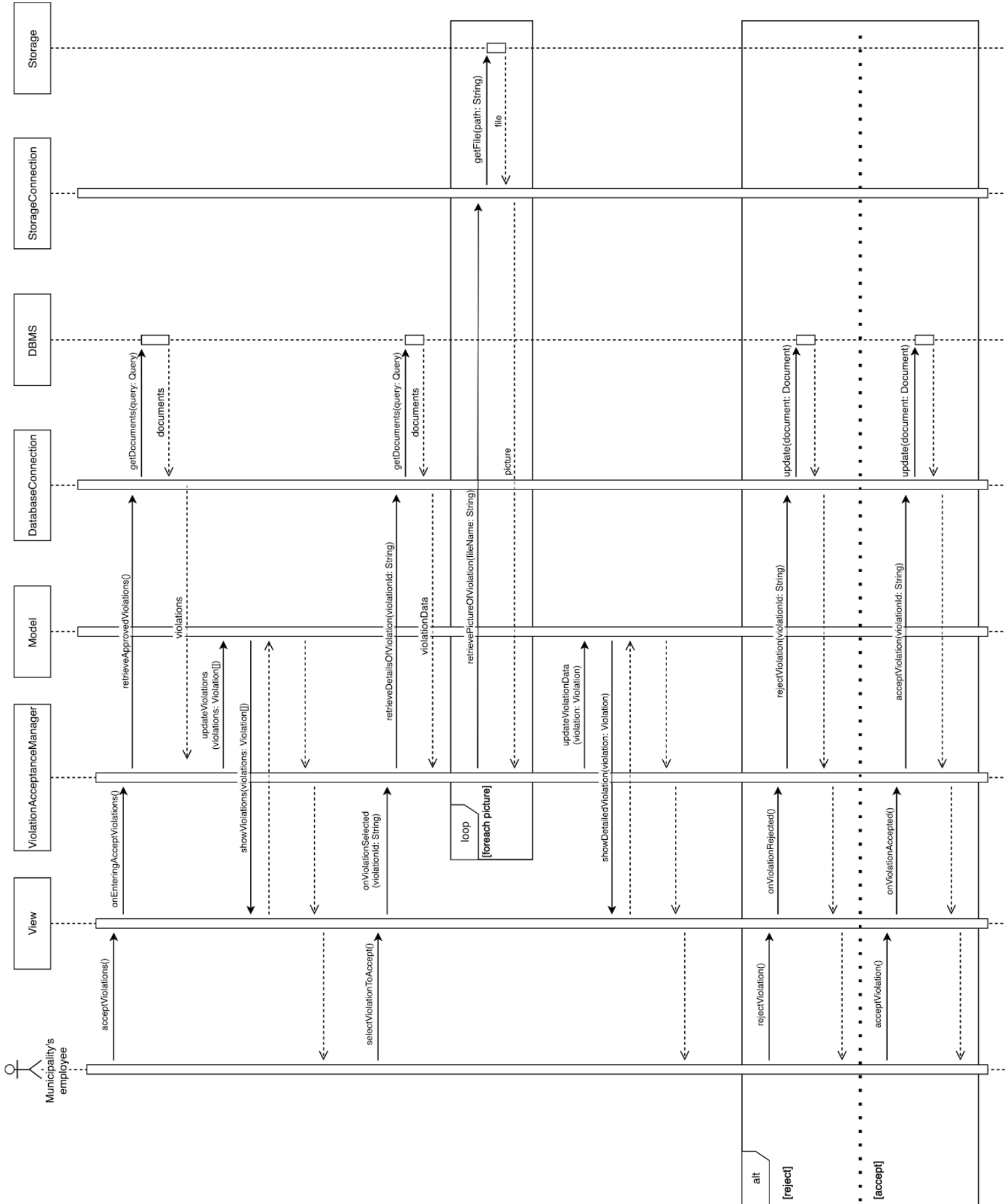
TODO

2.4. Runtime view

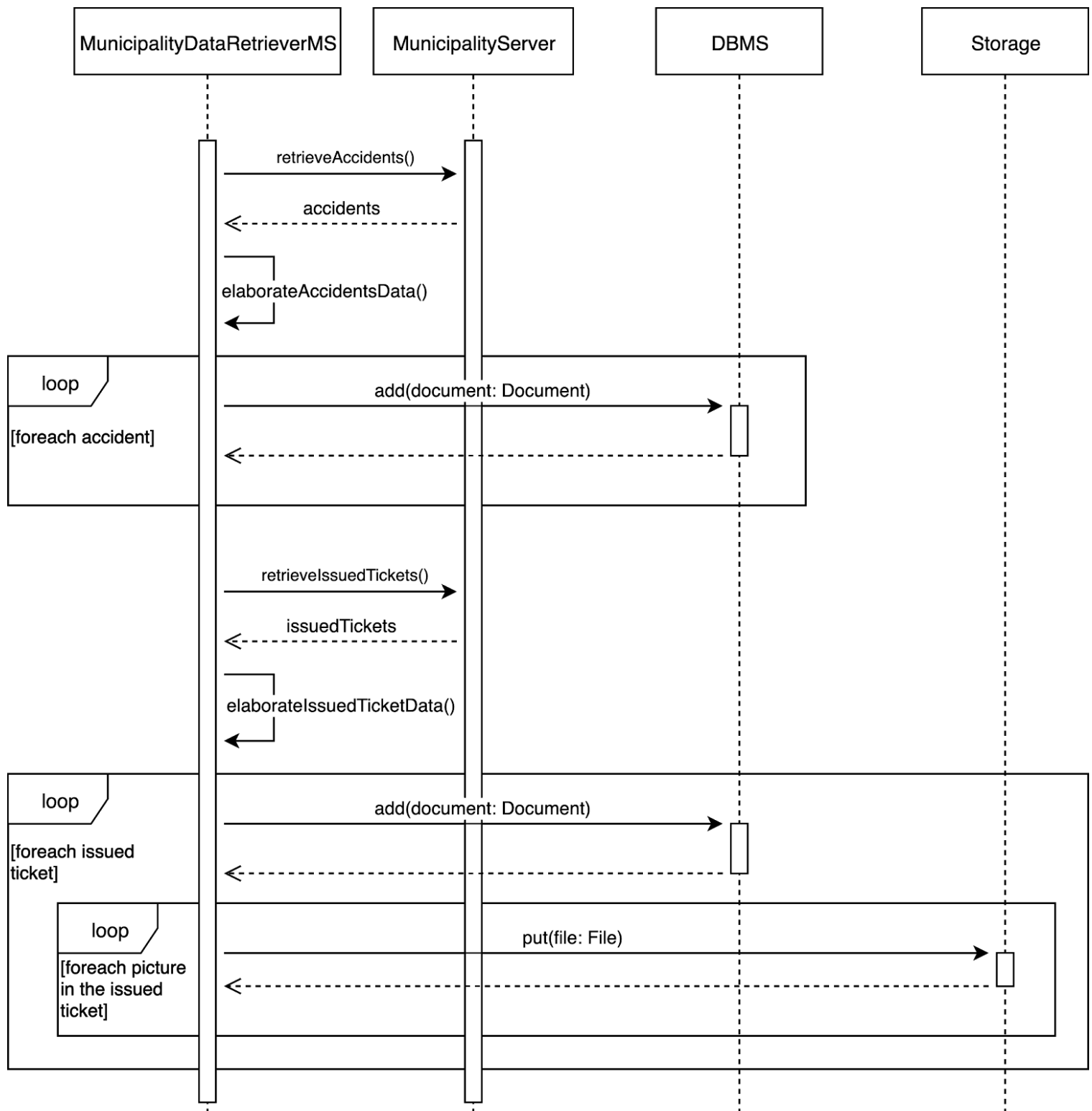
2.4.1. User violation reporting



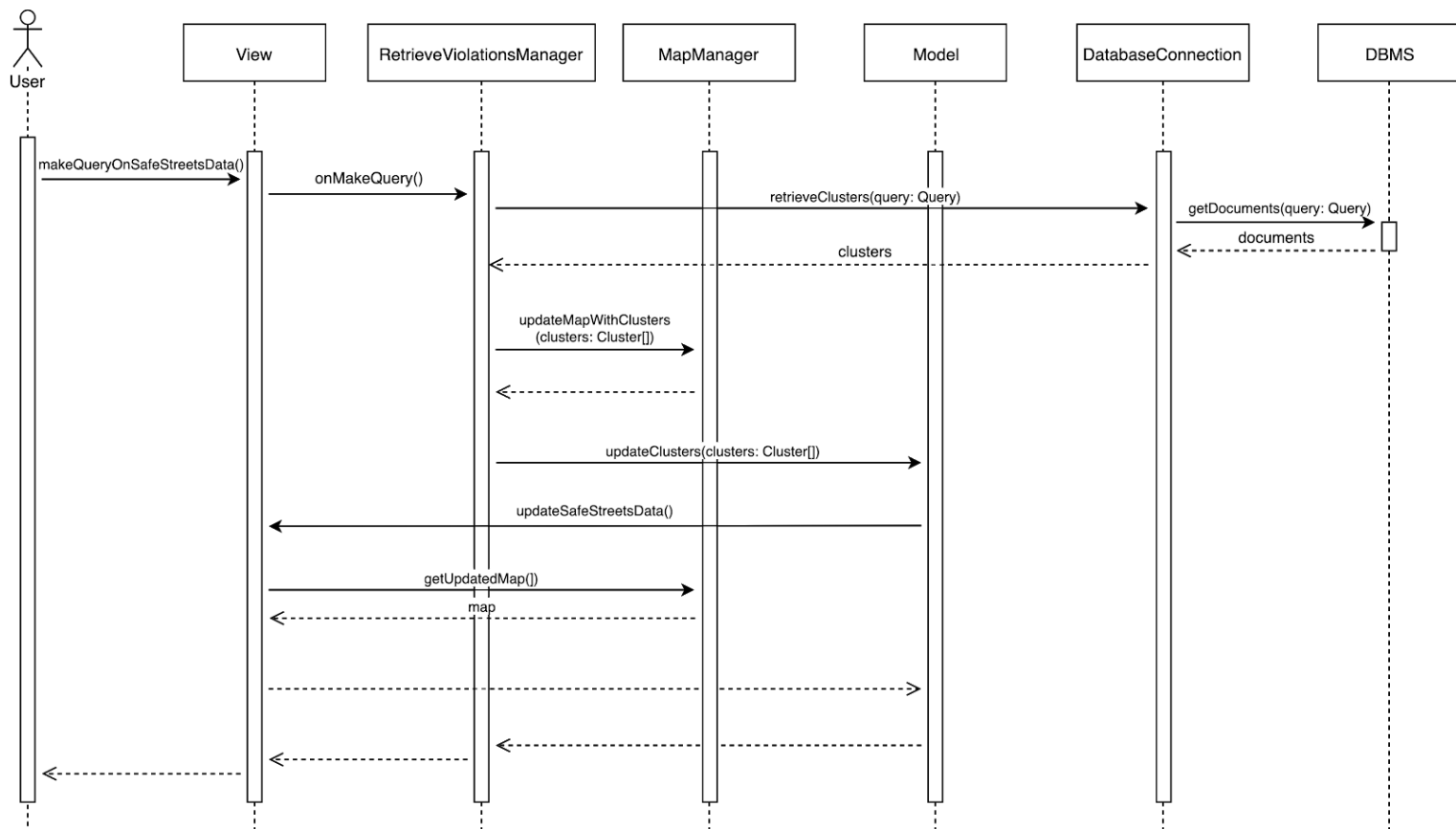
2.4.2. Municipality violation accepting



2.4.3. Retrieving of municipality's data



2.4.4. Visualizing SafeStreets Data



Note: visualizing SafeStreets data for the municipality is the same but they can see private data like pictures, so the municipality also uses the StorageConnection component to retrieve the pictures.

2.4.5. Suggestions for the municipality

TODO

2.5. Component interfaces

TODO

2.6. Selected architectural styles and patterns

Model–View–Controller: 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. Following the MVC architectural pattern decouples these major components, which is one of the most common and effective ways to avoid a dangerous level of coupling between the various parts of the whole system.

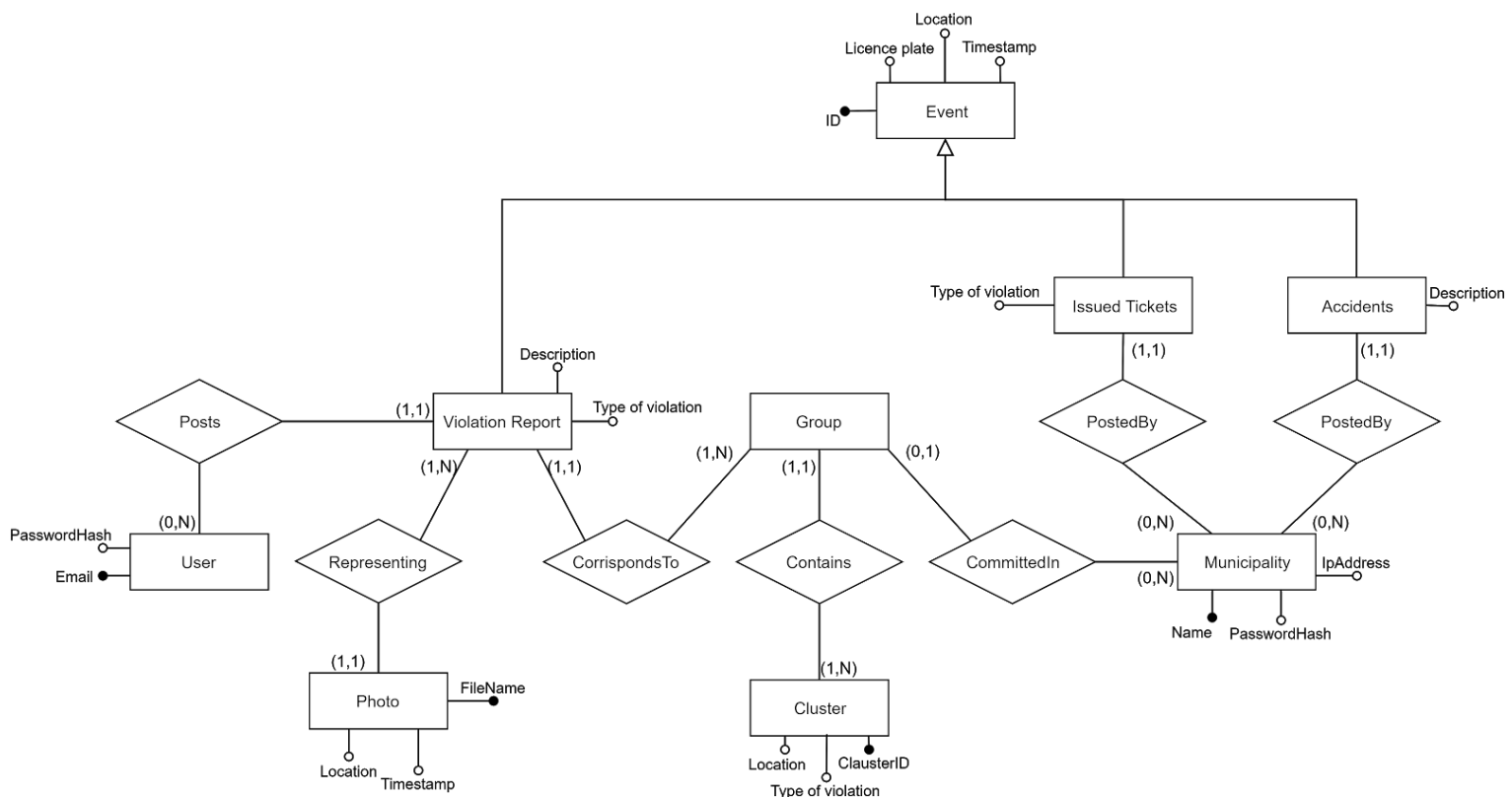
Serverless computing: it is a cloud-computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Serverless computing can simplify the process of deploying code into production. Scaling, capacity planning and maintenance operations may be hidden from the developer or operator. Serverless code can be used in conjunction with code deployed in traditional styles, such as microservices.

2.7. Other design decisions

2.7.1. Data Structure

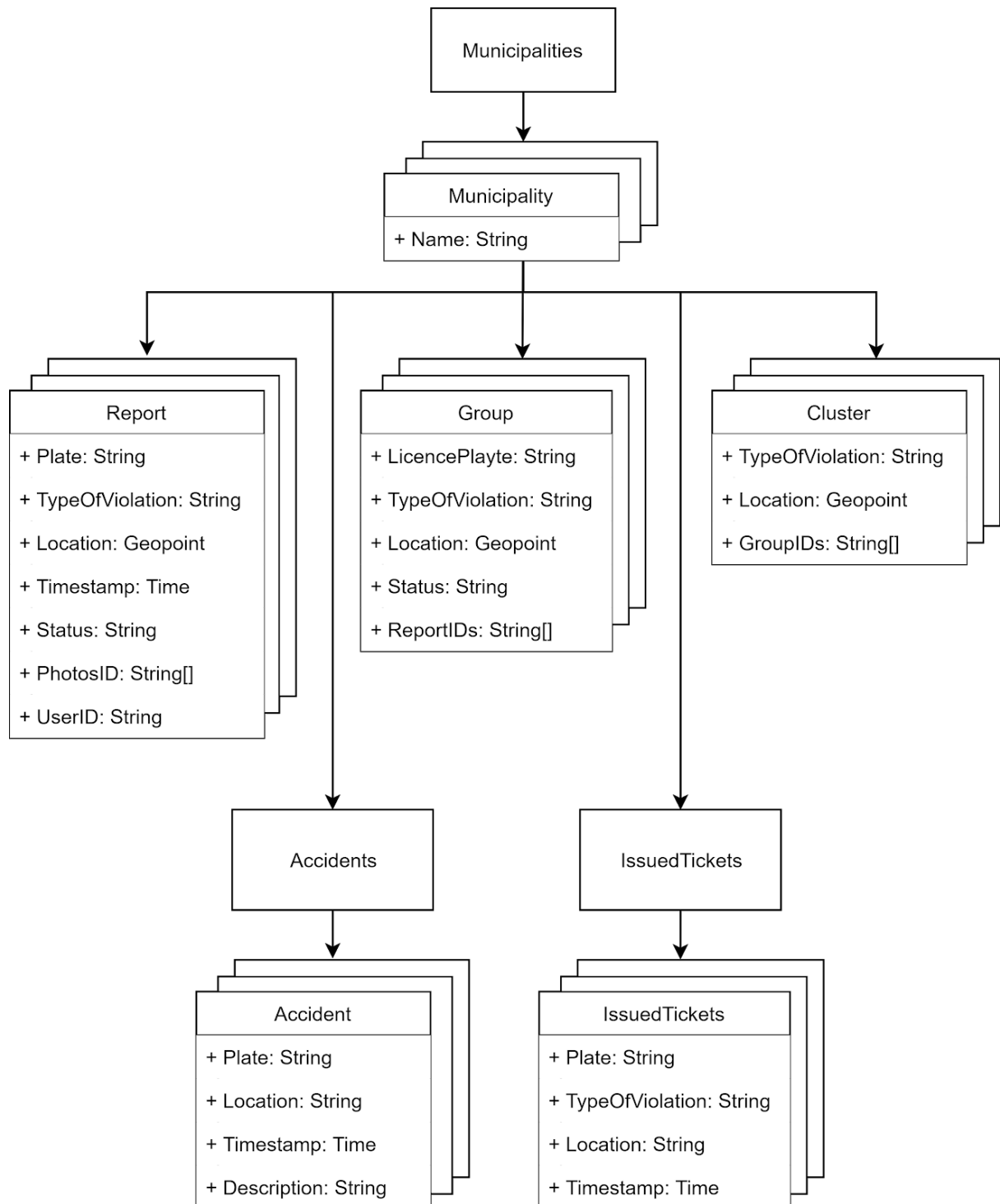
Data will be stored in a NoSql Database, so its structure is crucial in order to have fast and easy-to-write queries. This can be achieved by duplicating information among the files of the DB.

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 on the map of the marks, 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.



2.7.2 External APIs

Maps API:

Municipality API:

Image recognition API: by google?

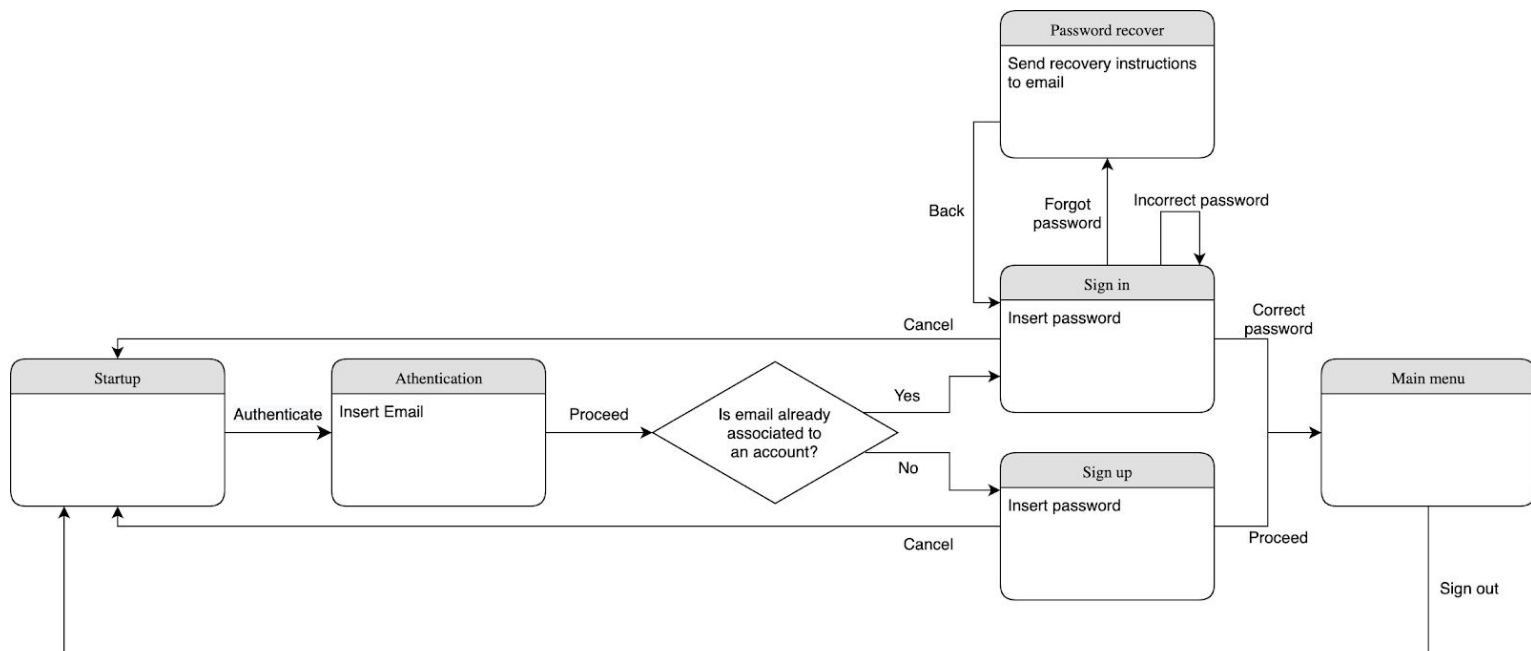
3. User interface design

3.1. SafeStreets App

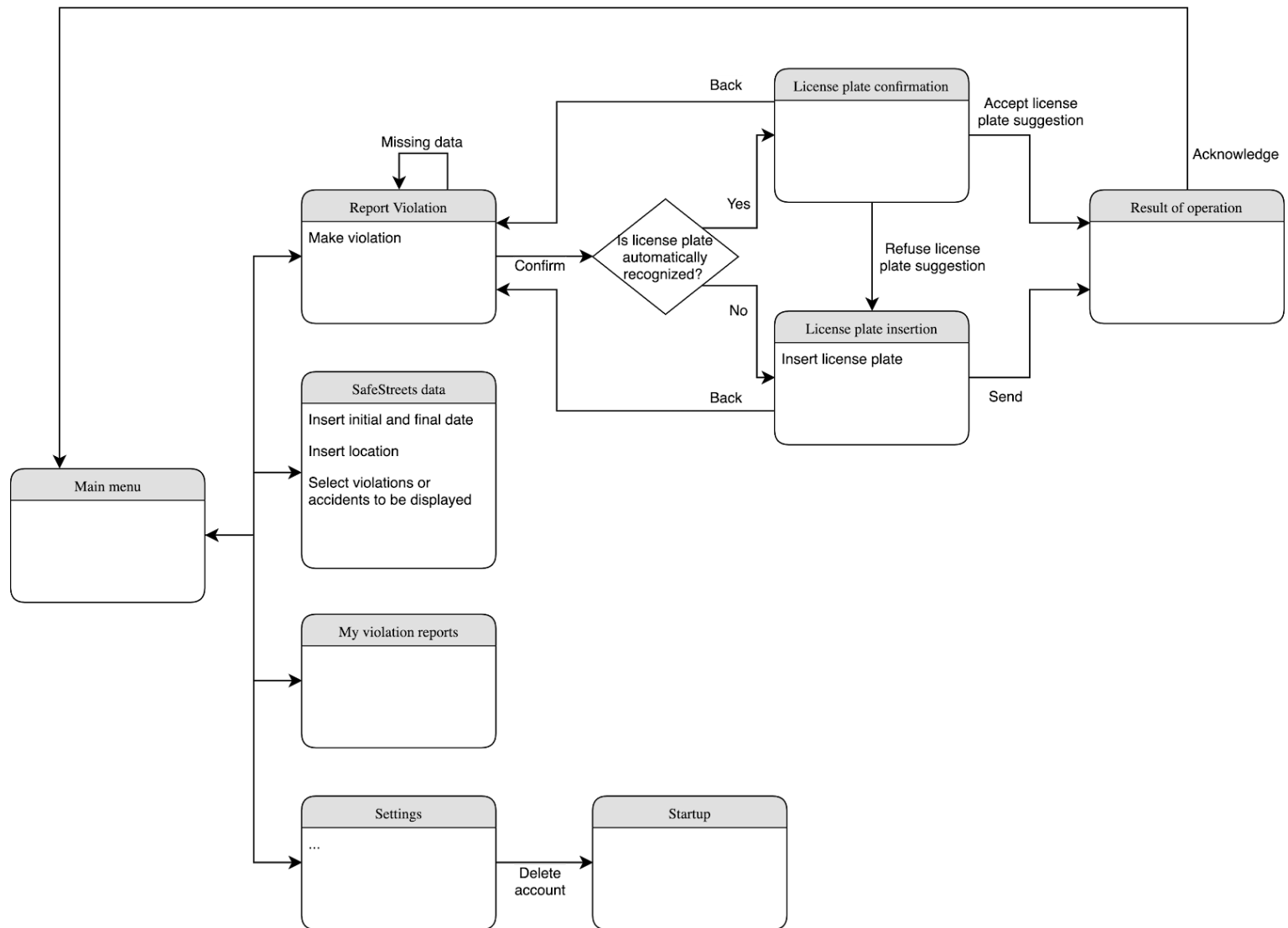
In this section it is shown the user interface design applied to the SafeStreets application.

First are presented the UX diagrams where it is shown the flow followed by the users to utilize the app. 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 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.

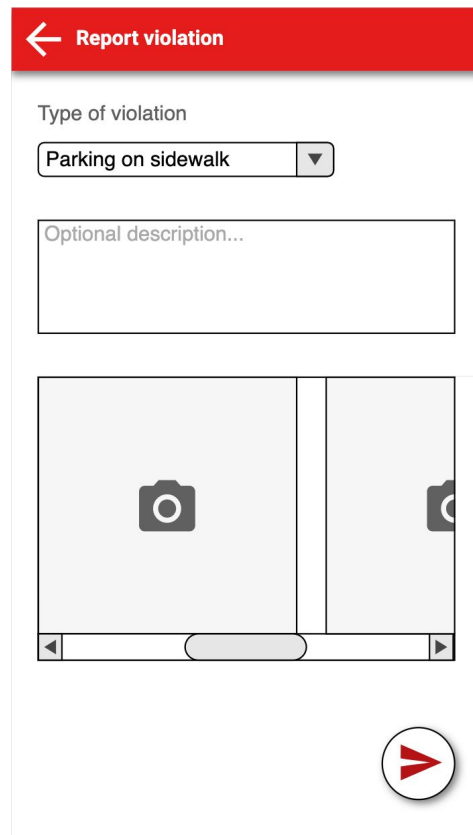
3.1.1. User authentication - UX diagram



3.1.2. App features - UX diagram



3.1.3. Mockup of the “Report Violation” page



← Report violation

Type of violation

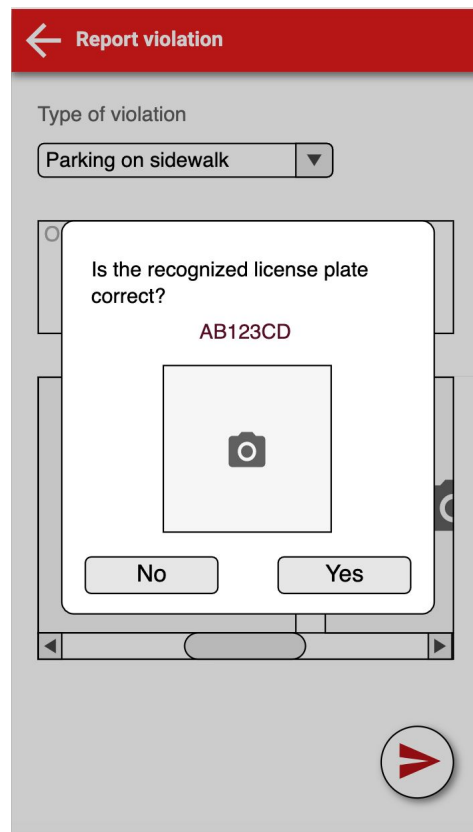
Parking on sidewalk ▼

Optional description...

Image gallery with two slots, each containing a camera icon. A horizontal scrollbar is visible below the gallery.

➤

3.1.4. Mockup of the “License plate confirmation” page



← Report violation

Type of violation

Parking on sidewalk ▼

Is the recognized license plate correct?

AB123CD

Image gallery with two slots, each containing a camera icon. A horizontal scrollbar is visible below the gallery.

No Yes

➤

3.1.5. Mockup of the “License plate insertion” page

The mockup shows a mobile application interface for reporting a violation. At the top is a red header bar with a white back arrow and the text "Report violation". Below the header, there is a section titled "Type of violation" with a dropdown menu currently showing "Parking on sidewalk". Underneath is a text area labeled "Optional description...". A white modal dialog box is centered on the screen, containing the text "Please insert the license plate of the vehicle." and a text input field with the placeholder "License plate...". A red arrow button is located at the bottom right of the dialog. At the bottom of the screen, there is another red arrow button.

3.1.6. Mockup of the “SafeStreets data” page

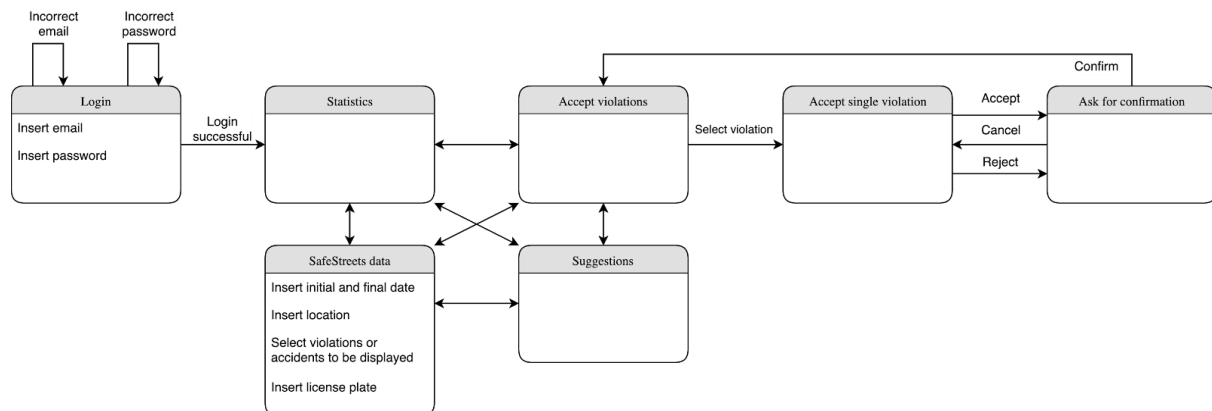
The mockup displays a mobile application interface for viewing violations on a map. The top features a red header bar with a white back arrow and the text "Violations on the territory". Below the header is a search bar with a magnifying glass icon and the placeholder "Search location...". There are two date selection fields: "Start date" with the value "23/10/2019" and "End date" with the value "10/11/2019", each accompanied by a calendar icon. Below these is a "Display..." section with two checked checkboxes: "Violations" and "Accidents". A "Select types" button is positioned to the right of the "Violations" checkbox. At the bottom of the screen is a map showing a street network with several red circular markers indicating violation locations. A grey location pin icon is also visible on the map.

3.2. Municipality web interface

In this section it is shown the user interface design applied to the SafeStreets web interface for municipalities.

First is presented the UX diagram where 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

The mockup shows a web browser window with the URL <https://safestreets-polimi.web.app>. The page title is "Login". The main content area displays the "SafeStreets" logo and the text "municipality login". Below this, there are two input fields: "Email Address" and "Password". A "Login" button is positioned to the right of the "Password" field.

3.2.3. Mockup of the “Accept violations” page

Accept violations

← → ↻

https://safestreets-polimi.web.app

SafeStreets

Accept violations

Display data

Show statistics

Suggestions

Logout

Order by

Number of reports ▾

Violations

License plate: DC321BA
Location: Via Respiratoria
Violation type: "Parking on crosswalk"
Number of reports: 32

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

License plate: EC111II
Location: Viale Mani Dal Naso
Violation type: "Parking in a no-parking zone"
Number of reports: 15

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

3.2.4. Mockup of the “Accept single violation” page

Accept violation

← → ↻

https://safestreets-polimi.web.app

SafeStreets

Accept reports

Display data

Show statistics

Suggestions

Logout

Group id: 553432
Number of reports: 32
Location: Via Respiratoria (44.6588696, 8.3219011)
License plate: DC321BA
Violation type: "Parking on crosswalk"
Time: from 8:30 to 17:30



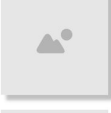
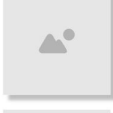


Descriptions:

The driver parked on the crosswalk. Crossing the street was very dangerous.

Please make a ticket to this imbecile.

Ticket!!

Pictures:



Reject

Accept

3.2.5. Mockup of the “SafeStreets data” page

SafeStreets data

← → ↻

https://safestreets-polimi.web.app

SafeStreets

Accept reports

Display data

Show statistics

Suggestions

Search location...

Search license plate...

Start date

27/11/2019

▼

End date

09/12/2019

▼

Display...

☒ Violations

☒ Accidents

Select types

Map

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)
- [R16] The system must offer a service to the municipality for retrieving correct 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
 - ApprovingMS
 - MunicipalityDataRetrieverMS
 - OnReportStatusChangeMS
 - ClusteringMS
 - GroupingMS
 - DBMS
 - Hosting
 - Authentication System
 - Data Storage
 - MapManager (Web)
 - AuthenticationManager (Web)
 - StorageConnetion (Web)
 - DatabaseConnection (Web)
 - StatisticsManager (Web)
 - ViolationAcceptanceManager (Web)
 - RetrieveViolationsManager (Web)
 - MapManager (Web)
 - RetrieveViolationsManager (App)
 - MapManager (App)
 - ReportViolationManager (App)
 - AuthenticationManager (App)
 - StorageConnetion (App)
 - DatabaseConnection (App)

3.2.4.1. Satisfying goal 1

[G1] A citizen can report a violation.

- [R1] A citizen not yet registered must be able to sign up and become a user.
- [R2] The application must allow users to authenticate through log in.
- [R3] The application must allow users to report a violation.

3.2.4.2. Satisfying goal 2

[G2] A violation report received by the system must have enough information to be valid, i.e. has at least one picture of the violation, exactly one GPS position, exactly one timestamp, exactly one type of violation and the license plate of the vehicle.

- [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.
- [R5] The application must allow reporting of violations only from devices equipped with a camera.
- [R6] The application must allow reporting of violations only from devices with an active internet connection.
- [R7] A user has the possibility to specify the type of the reported violation choosing from a list.
- [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.

3.2.4.3. Satisfying goal 3

[G3] Users and municipality can retrieve information about violations, accidents and issued tickets in a certain area, with different levels of visibility.

- [R9] The system saves all the information regarding the violations reported by users.
- [R10] The system must be able to retrieve data regarding issued tickets and accidents from the municipality.
- [R11] The system must offer an interface to retrieve information about violations, accidents and issued tickets.
- [R12] The system must show to users only anonymized data.
- [R13] The system must analyze valid violations report and approve which of them may represent a correct violation.

3.2.4.4. Satisfying goal 4

[G4] Municipality will be able to retrieve suggestions for possible interventions in order to increase safety.

- [R14] The system must be able to elaborate data about violations, accidents, issued tickets and generate useful suggestions about possible interventions.
- [R15] The system must offer an interface to the municipality for retrieving useful suggestions about possible interventions.

3.2.4.5. Satisfying goal 5

[G5] Municipality receives enough information about the violation in order to issue a ticket.

- [R16] The system must offer a service to the municipality for retrieving correct violations report.

3.2.4.6. Satisfying goal 6

[G6] The integrity of the violation report is guaranteed.

- [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.
- [R18] All connections used by the system use modern encryption protocols.
- [R19] Data saved in the server can not be manipulated.

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. (why is it used)

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 writing efficient algorithms to maximize the probability of rejecting an unwanted report and, at the same time, minimizing the probability of generating too many false rejections.* (come si può scrivere meglio?)

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.

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 SafeStreet 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 reports based on the decision of the municipality.

The difficulty of building this feature is “high” because it will require 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 three 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. (breve descrizione, vantaggi, le feature sono milestones, piccole parti / sottosistemi di un sistema ← spiegarlo)

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. Finally, the statistics feature (Feature 5) will be built.

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.

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
Total:		

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 20 min
30/11	Component view	4 hr 0 min
Total:		

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

Total:	
--------	--

7. References

1. "Serverless paradigm". https://en.wikipedia.org/wiki/Serverless_computing

Notes

<http://www.cloudcomputingpatterns.org/>

Display and clustering

When a new violation is added it checks whether the same violation already exists, by checking on the same license plate, location, time. If it already exists it is added on the same report.

When the map is displayed, reports of the same violation type are clustered together.

Suggestions

Suggestions interact with clustering, suggestion are hardcoded.

Location

Each picture is linked with a location. The report however has only one location. The location in the report can be the average between the locations of the pictures.

Database

Each user has the list of each report sent.

Folders

Violation Reports

Clusters

Users

Municipalities

Groups

Accidents

Issued tickets

Groups

Unifies the reports that represents the same violation. It is a list of indexes of the reports that represents the same violation. The tuple also has the state of the reports (since it is the same for all reports that represents the same violation)

Groups merged with municipalities

Municipalities

Structure

Municipalities

- Municipality 1 [attribute = <ip api issued tickets & accidents>]
 - Approved
 - Group 1

- Report1 ID
 - Report2 ID
 - Group 2
- Rejected
 - Group 4
 - Report1 ID
 - Report2 ID
 - Group 73
- Confirmed
 -
- Issued Ticket
 - ticket1
 - plate
 - location
 - type
 - timestamp
 - (done by)
 - ticket2
- Accidents
 - accident1
 - description
 - plate
 - location
 - timestamp
 - accident2
- Municip 2
- Municip 3

Clusters

It is divided by violation type. Each violation type has the list of all clusters

Clusters

- Typeofviolation
 - Cluster1
 - Location
 - UserSuggestions (if implemented)
 - Reports
 - Report1
 - Report2
 - Cluster2

Microservices (triggers)

- GroupingMS
 - Same violation
 - Verify if the violation has already been reported (location, time, licence plate)
 - Triggered by DB insert
 - Uses DB and Maps API
- ClusteringMS
 - Same location/type.
 - Triggered by DB insert
 - Uses DB
- ApprovingMS
 - Approves the report also using cloud vision.
 - Triggered by DB insert
 - Uses DB and Vision API
- MunicipalityDataRetrieverMS.
 - Accidents retriever
 - Tickets retriever
 - Triggered by a time event (like every day updates)
 - Uses DB and Storage
- OnReportConfirmedMS / OnReportRejectedMS.
 - Update the state of the reports from the municipality.
 - This is a trigger that when the state of a group of reports is updated, then we update all the corresponding reports in the DB.
 - Triggered by municipality
 - Uses DB
- Statistics ms
- ????

Reports

Using Google vision the pictures will be checked if they have at least one vehicle in one of their photos. The report has the state, because a user must be able to retrieve his reports and the state, without accessing "Clusters" which contains sensitive information.

Interface

My reports

Settings

Web interface

The interface takes each group that is approved and not yet reviewed.
When the employee clicks on a report, it is opened and the photos are downloaded.

Statistics

List of statistics:

- number of issued tickets filtered by:
 - type of violation
 - interval of time
 - plate
-

- 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.
 - The *MicroservicesContainer*. This is a collection of Microservices that are used to manage the subsystem and to provide useful APIs to both users and municipality. It is the most important and complex component of this subsystem and will be analyzed in detail later in the document.
 - The *DBMS*, which manages the database.
 - The *data storage*, in which we mainly store pictures of violations.
- The SafeStreets mobile application is built using the Model - View - Controller 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 Model and the View will be analyzed later.
 The Controller has the following components:
 - *ReportViolationManager*: the main function 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*: connects to the DBMS.
 - *StorageConnection*: connects to the data storage.
 - *MapManager*: uses the StatisticsManager and the Maps API to display violations and other data on the map.
 - *RetrieveViolationsManager*: *non si è ancora capito sinceramente TODO: Write this component that handles all the queries.*
- The web interface offered to the municipality. This has the same structure of the application with the obvious difference that the view will be an HTML web page. It has, however, an extra component.
 - *StatisticsManager*: it uses the statistics microservice to provide useful statistics to 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

Furthermore, the system will use some external services to grant and enhance the service. They are the following:

- *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 validate the report; 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 storing unwanted data, to prevent flooding attacks to the system and to increase efficiency.

- *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, both to the user and to the municipality.