POLITECNICO
MILANO 1863

*Academic year 2019-2020*
Software Engineering 2 Project

# SafeStreets

# ITD

Implementation and Test Deliverable

Version 1.0

**Authors:**
Marcer Andrea - 941276
Marchigiana Matteo - 945878
Motta Dennis - 940064

**Professor:**
Rossi Matteo

# 1. Introduction

## 1.1. Purpose

This document is the Implementation and Testing Document for the SafeStreet system. Here will be listed the implemented requirements, the used frameworks and the testing done.

## 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. Acronyms

- UI: user interface
- RASD: Requirement Analysis and Specification Document
- DD: Design Document
- HTTPS: Hypertext Transfer Protocol Secure
- HTML: Hypertext Markup Language
- DBMS: Database Management System

## 1.4. Revision history

- Version 1.0: Initial release.

## 1.5. Reference documents

- Specification Document: “SafeStreets Mandatory Project Assignment.pdf”.
- Gradle (build automation system): https://gradle.org/
- Android Spinners: https://developer.android.com/guide/topics/ui/controls/spinner
- Google Maps API: https://developers.google.com/maps/documentation
- Firebase API Reference: https://firebase.google.com/docs/reference
- Npm (JavaScript package manager): https://www.npmjs.com/
- Mocha (JavaScript test framework): https://mochajs.org/
- Chai (JavaScript assertion library): https://www.chaijs.com/
- Sinon (JavaScript mocks & stubs): https://sinonjs.org/
- Nyc (JavaScript test coverage): https://istanbul.js.org/

## 1.6. Document structure

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

**Chapter 2: Implemented requirements**
Here it is shown which of the requirements specified in the RASD have been implemented, which have been partially implemented and which ones have not been implemented. To each requirement it is associated a short description on how the requirement has been achieved or why it has not been implemented.

**Chapter 3: Adopted frameworks**
In this chapter it is specified which frameworks have been used and why they have been adopted.

**Chapter 4: Source code structure**
The structure of the source code is presented and explained.

**Chapter 5: Testing**
Here there are listed the tests done on the different subsystems. Each test has an identifier, an input condition, the expected output, a description on how the test has been performed and the result of the test.

**Chapter 6: Installation instructions**

In this chapter it is shown where to retrieve all the files needed to install and run the system.

**Chapter 7: Effort spent**
Here it is shown the effort spent by each team member working on this document and the time spent during the implementation of the system.

# 2. Implemented requirements

## 2.1. User Signup and Login

### Requirements implemented:

**[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.**

Both requirements are implemented and granted through Firebase libraries. When the user opens the application for the first time they need to sign in to the system. During the registration it is asked to insert an email, that uniquely identifies the user, and a password. After the registration the user is automatically logged in, and can obviously choose to log out.

## 2.2. Violation Reporting

### Requirements implemented:

**[R3] The application must allow users to report a violation.**

Granted simply by allowing only logged in users to access the *Violation Report* interface.

**[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.**

Achieved by checking the status of the GPS before sending the report.

**[R5] The application must allow reporting of violations only from devices equipped with a camera.**

Achieved by checking the status of the camera at runtime.

**[R6] The application must allow reporting of violations only from devices with an active internet connection.**

Achieved by both requiring that the device can connect to the internet and checking the status of the internet connection at runtime.

**[R7] A user has the possibility to specify the type of the reported violation choosing from a list.**

In the UI it is present an Android *Spinner*[1] that allows the user to choose the type of violation.

**[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.**

Granted by creating an immutable object and checking that all the mandatory fields are specified before sending it to the server.

**[R18] All connections used by the system use modern encryption protocols.**
Granted by the standard libraries of Android and by the Firebase library.

### Requirements not implemented:

**[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.**
The developers of Android are refactoring the code concerning the camera. So most of the old method and libraries are deprecated and do not work anymore. Since this change is recent, there is a lack of clear official documentation. So, instead of taking the picture within an Activity, an external camera application is called.
To stop the user from modifying the picture directly from the device storage, the photos taken are saved in the internal memory dedicated to the application.

## 2.3. Database & Storage

### Requirements implemented:

**[R9] The system saves all the information regarding the violations reported by users.**
Granted by the design itself since the application writes directly to the NoSql Database.

**[R13] The system must analyze valid violations report and approve which of them may represent a correct violation.**
Achieved by using Google Cloud Vision and by analyzing if at least one of the pictures uploaded by the user actually contains a vehicle.

**[R19] Data saved in the server cannot be manipulated.**
By using the Firebase security rules the access to the NoSql database is strictly limited and no users can modify the data in a non recognized way.

### Requirements partially implemented:

**[R14] The system must be able to elaborate data about violations, accidents, issued tickets and generate useful suggestions about possible interventions.**
The elaboration about violations is achieved by analyzing every report when it is received and by putting it in the appropriate group/cluster which are then used to generate suggestions based on the type of violation.

While the elaboration about accidents and issued tickets was not implemented since it would have required to have an actual database for every municipality to retrieve data from.

### Requirements not implemented:

**[R20] The system must be able to elaborate data about violations, accidents, issued tickets and generate useful statistics for each municipality.**

This requirement refers to a functionality of the “Advanced function 2” of the Project Assignment, which we chose not to implement.

## 2.4. Municipality

### Requirements implemented:

**[R15] The system must offer an interface to the municipality for retrieving useful suggestions about possible interventions.**

By hovering over a marker on the map, the system will display info about the marker and, if it is a cluster, the related suggestions.

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

In the website, the page “Accept violations” connects to the database and retrieves the reports.

**[R18] All connections used by the system use modern encryption protocols.**

Granted by the Firebase platform and by the hosting system.

### Requirements not implemented:

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

The elaboration about accidents and issued tickets was not implemented since it would have required to have an actual database for every municipality to retrieve data from.

**[R21] The system must offer an interface to the municipality for retrieving statistics.**

This requirement refers to a functionality of the “Advanced function 2” of the Project Assignment, which we chose not to implement.

# 3. Adopted development frameworks

## 3.1. Web

Google Maps API and Geocoding were used to display the map and data in it.

## 3.2. Application

The most used operating systems for smartphones are Android and IOS. The first one has been chosen because Android applications support Java, a widespread programming language that is well supported by the community. Furthermore the Android documentation is complete and contains useful examples on how to implement in the most efficient and correct way different functionalities.
To help the developing the following libraries have been used:

- play-services:
- Androidx:
- Firebase:
- places:
- butterknife:

## 3.3. Cloud

For the implementation of the cloud structure of SafeStreets we chose to use Firebase for its simplicity and for the amazing advantage that an all-in-one package brings.
So for each one of the 5 subcomponents of the cloud, a Firebase functionality has been associated:

- Authentication: uses “Firebase Authentication”
- Hosting: uses “Firebase Hosting”
- Data Storage: uses “Firebase Storage”
- DBMS: uses “Firebase Cloud Firestore”
- Microservices: uses “Firebase Cloud Functions”

The first 4 components can’t run code, they are limited to using only rules and settings.
While the Microservices component runs stateless functions in their own isolated secure execution context, that are scaled automatically and have a lifecycle independent from other functions. So for this component it is required an analysis on its own.

### 3.3.1. Microservices

The “Firebase Cloud Functions” platform allows to use two different programming languages: JavaScript and TypeScript; We chose to use JavaScript for its simplicity and since its use is common with the Firebase Cloud Functions.
The code for the Microservices is managed by *npm*, a famous package manager for JavaScript, and through it the dependencies of the Microservices are integrated.

The code for microservices on the production server depends on these external packages:

- “Google Cloud Vision”: used to approve reports by recognizing vehicles in the pictures.
- “Firebase Admin”: provides access to the other Firebase components (Authentication, Storage, DBMS).
- “Firebase Functions”: allows the microservices to be triggered by events in the other components or by time.

While for testing the microservices other packages have been used:

- “ESLint”: it’s a famous static code analysis tool for identifying problematic patterns found in JavaScript code.
- “ESLint Promise Plugin”: “Promises” are used to handle asynchronous operations in JavaScript, this plugin allows to check common anti-patterns with their use.
- “Firebase Functions Test”: takes care of the appropriate setup and teardown for the tests, such as setting and unsetting environment variables needed by the “Firebase Function” package.
- “Mocha”: the most famous test framework for JavaScript
- “Chai”: an assertion library, used to test the correctness of the results.
- “Sinon”: the most famous mocking library for JavaScript, used for mocking the Google Cloud Vision API during tests.
- “Nyc”: used to find the coverage of the tests.

# 4. Source code structure

## 4.1 Website

There are four HTML pages corresponding to the four pages of the website, that are:

- Index.html (main page)
- AcceptViolations.html
- DetailedViolationView.html
- DisplayData.html

Excluding the index, the other three pages contain an “Hamburger menu”, used to navigate between pages. The menu, along with the underlying structure of the page, are styled in an apposite CSS file called “BaseStylesheet.css”. However, the CSS code that styles the element of the pages is embedded in the HTML files.

The Javascript code used is also embedded in the HTML files except for the common features, such as logging out and initializing the platform, that are contained in “api.js” and “logout.js”.

## 4.2 Application

During the developing of the application the MVC pattern has been adopted: allowing to better separate the application logic from the data structures. This can be easily seen from the structure of the source code. The packages are:

- Controller: contains classes that handle the application logic;
- Model: collects the different representations of the documents of the database, i.e. ViolationReports, Clusters and Groups;
- View: Activities and fragments of the UI;
- Interfaces: interfaces used in the application;
- Utils: collection of classes and methods used by different classes.

Moreover, under the *res* folder are collected the set of resources used by the application. Especially in the *layout* folder there are the XML files that are used to define the UI layout.

## 4.3 Cloud

The SafeStreets Cloud, as seen in the DD, can be divided into 5 subcomponents:

- Authentication
- Hosting
- Data Storage
- DBMS
- Microservices

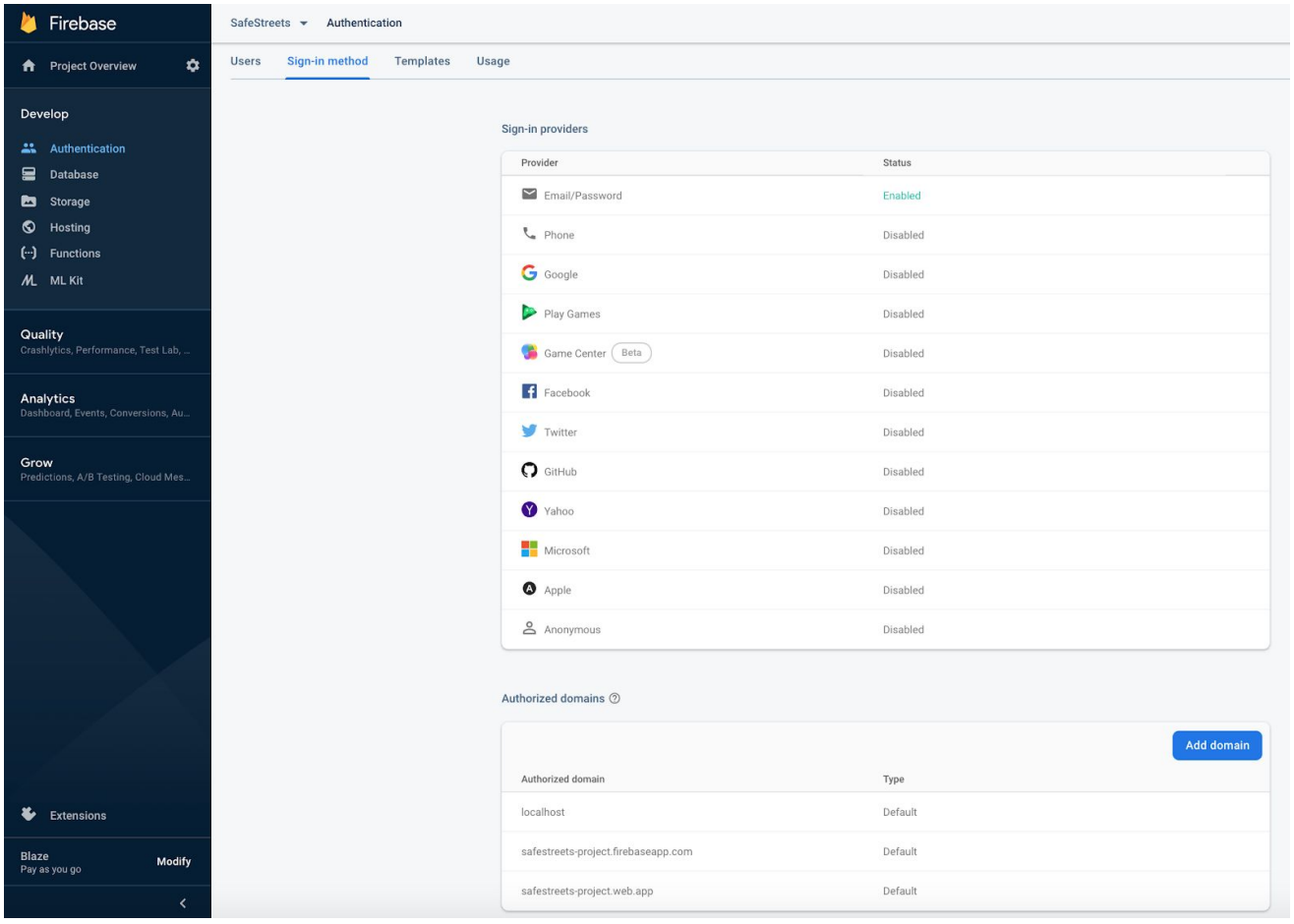
All the files for the SafeStreets Cloud are put in the *SafeStreetsCloud* folder.
Now each subcomponent is going to be analyzed on its own.

### 4.3.1. Authentication

No code has been written for this component since it only needs to be set up through the online Firebase console.
The things needed for the setup were:

- Enabling the “Email/Password” sign-in method;
- Specifying the domains authorized to use the authentication component.

The image below shows the current setup.



### 4.3.2. Hosting

The hosting subcomponent is a passive component. No code has been written for this component since it is just used as a way to transmit the HTML pages of the website through an HTTPS connection.
The website code is deployed on the component using the *firebase* package of *npm*.

### 4.3.3. Data Storage

The Data Storage subcomponent uses security rules to control the access to this subcomponent: only registered users can upload an image, while images can only be downloaded by the users that created them or by a municipality.
The source code for the security rules is written in */storage/storage.rules* while this code is deployed on the component using the *firebase* package of *npm*.

### 4.3.4. DBMS

Also the DBMS subcomponent uses security rules to control the access to it.
The source code for its security rules is written in */firestore/firestore.rules*.
What these security rules enforce is:

- Violation report can only be created by users and with the correct format (all fields must be present: location, timestamp, type of violation, etc...);
- Groups can be marked as “Correct” or “Rejected” only by the municipality that has control over its location.
- No other changes can be made to any other document.

Furthermore for the DBMS there is another file in the *firestore* folder, called *firestore.indexes.json*. This file specifies the indexes that are created in the NoSql Database; 3 indexes are created to access the main documents of the DBMS, an index for clusters, for groups and for violation reports.
Indexes and security rules are deployed on the component using the *firebase* package of *npm*.

### 4.3.5. Microservices

The Microservices subcomponent is the most complex in the SafeStreets Cloud. It contains a total of 6 microservices, for which the source code can be found in the *functions/main* folder:

- *ApprovingMS*: when receiving a report, this microservices checks and approves it. It uses an Image Recognition service to do so.
- *ClusteringMS*: this service groups together reports that have the same type of violation and close location creating a Cluster.
- *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.

- *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.
- *MunicipalityAuthorizerMS*: allows to associate an email to a municipality authorization.
- *MunicipalityDeauthorizerMS*: allows to disassociate an email to a municipality authorization.

The microservices reported here also use other code snippets written in the *model* folder and in the *utils* folder.
Note that the Microservices are managed by *npm* so there are also present files relative to its operation, like *package.json* and *package-lock.json*.
As the other components also the code for the microservices is deployed using the *firebase* package of *npm*.

# 5. Testing

The implementation has followed a bottom-up approach, so the tests will mirror this decision. First of all the components that do not depend on other component will be tested separately. After this first phase integration tests will be performed for each subsystem, i.e. for the microservices, the application and the web interface. At the end a general test will be performed to ensure that all requirements listed in the RASD are satisfied.

## 5.1. Website

The website functionalities require a human to manually perform tests on them.

| **Test ID** | **W_1** |
|---|---|
| Input | Retrieve violations |
| Output | Groups correctly retrieved |
| Description | Upon opening the page “AcceptViolations.html”, the website should connect to the database and retrieve eligible groups. |
| Result | Groups with status “APPROVED” of the municipality authorized for the account are retrieved. |

| **Test ID** | **W_2** |
|---|---|
| Input | Clicking on a group |
| Output | The page changes and the data is retrieved correctly |
| Description | Upon clicking on a group in the page “AcceptViolations.html”, the website should change the page to “DetailedViolationReport.html” and correctly retrieve information and pictures about the report. |
| Result | The page changes correctly and the data is retrieved. |

| **Test ID** | **W_3** |
|---|---|
| Input | Filtering data on the map |
| Output | Data correctly filtered |
| Description | In the “DisplayData.html” page, the user can filter the data retrieved by setting an upper and lower bound on the date. |
| Result | Data is correctly filtered. |

| **Test ID** | **W_4** |
|---|---|
| Input | Accepting or rejecting a report |
| Output | Group status changes |
| Description | When visualizing a report, the user can click either on Accept or Reject button. |
| Result | When Accept is pressed, the groupStatus changes to “CORRECT”. When Reject is pressed, the groupStatus changes to “REJECTED”. |

## 5.2. Application

Due to the difficulty of directly testing some of the methods of the application, some tests will need a human to verify the correctness of the output.

| **Test ID** | **A_1** |
|---|---|
| Components | MapManager |
| Input | Retrieve Location |
| Output | Correct Location |
| Description | A picture is taken in the violation report interface, the location (latitude and longitude) can be read in the debug log and then compared with the actual location. |
| Result | The location corresponds with the current location. |

| **Test ID** | **A_2** |
|---|---|
| Components | MapManager |
| Input | Retrieve Address |
| Output | Correct Address |
| Description | A picture is taken in the violation report interface, the address can be read directly in the view and then compared with the actual address. |
| Result | The address corresponds with the current address. |

| **Test ID** | **A_3** |
|---|---|
| Components | Report Violation |
| Input | Sending violation report with no licence plate |
| Output | Unable to send the violation report |
| Description | The violation reporting interface is opened, a picture which does not have a licence plate and the “Send” button is pressed. |
| Result | A popup appears notifying the user that an error occurred and the report is not sent. |

| **Test ID** | **A_4** |
|---|---|
| Components | Report Violation |
| Input | Sending violation report with no picture and location |
| Output | Unable to send the violation report |
| Description | The violation reporting interface is opened, the licence plate is specified and the “Send” button is pressed. |
| Result | A popup appears notifying the user that an error occurred and the report is not sent. |

| **Test ID** | **A_5** |
|---|---|
| Components | Report Violation |
| Input | Invalid licence plate format |
| Output | Unable to send the violation |
| Description | The violation reporting interface is opened, an invalid licence plate, i.e. does not correspond to the italian licence plate format [A-Z][A-Z][0-9][0-9][0-9][A-Z][A-Z], is inserted and the “Send” button is pressed. |
| Result | A popup appears notifying the user that the licence plate is invalid and the report is not sent. |

| **Test ID** | **A_6** |
|---|---|
| Components | Report Violation |
| Input | Send violation without internet connection |
| Output | Unable to send the violation |
| Description | The violation reporting interface is opened, all mandatory fields are specified, the internet connection is disabled and the “Send” button is pressed. |
| Result | A popup appears notifying the user that an error occurred and the report is not sent. |

| **Test ID** | **A_7** |
|---|---|
| Components | Report Violation |
| Input | Send violation without an active GPS |
| Output | Unable to send the violation |
| Description | The violation reporting interface is opened, all mandatory fields are specified, the GPS is disabled and the “Send” button is pressed. |
| Result | A popup appears notifying the user that an error occurred and the report is not sent. |

| **Test ID** | **A_8** |
|---|---|
| Components | Report Violation |
| Input | Send violation without internet connection and an active GPS |
| Output | Unable to send the violation |
| Description | The violation reporting interface is opened, all mandatory fields are specified, the internet connection and the GPS are disabled and the “Send” button is pressed. |
| Result | A popup appears notifying the user that an error occurred and the report is not sent. |

| **Test ID** | **A_9** |
|---|---|
| Components | Report Violation |
| Input | Taking a picture without internet connection and an active GPS |
| Output | The location is not retrieved |
| Description | The violation reporting interface is opened, the GPS is disabled and a picture is taken. |
| Result | The location is not retrieved. |

| **Test ID** | **A_10** |
|---|---|
| Components | Retrieve Violation |
| Input | Own reports |
| Output | Correct list of reports |
| Description | Open “My reports” and compare the displayed list of reports with the reports saved on the server. |
| Result | The list coincide. |

| **Test ID** | **A_11** |
|---|---|
| Components | Map Manager and Retrieve reports |
| Input | Retrieve all clusters in a municipality |
| Output | Correct list of reports |
| Description | Open “SafeStreets data”, and set a location. |
| Result | All and only the clusters associated with approved groups are correctly loaded and displayed. |

| **Test ID** | **A_12** |
|---|---|
| Components | Map Manager and Retrieve reports |
| Input | Filter clusters on date |
| Output | Correct list of reports |
| Description | Open “SafeStreets data” and set the date interval so that only one cluster is displayed. |
| Result | The clusters are filtered correctly. |

| **Test ID** | **A_13** |
|---|---|
| Components | Map Manager and Retrieve reports |
| Input | Filter clusters on violation type |
| Output | Correct list of reports |
| Description | Open “SafeStreets data” and set a violation type. |
| Result | The clusters are filtered correctly. |

| **Test ID** | **A_14** |
|---|---|
| Components | Map Manager and Retrieve reports |
| Input | Filter clusters on violation type and date |
| Output | Correct list of reports |
| Description | Open “SafeStreets data”, set a date interval and a violation type. |
| Result | The clusters are filtered correctly. |

## 5.3. Cloud

The microservices have been tested using unit tests, these unit tests have been developed for the 4 main services: ApprovingMS, ClusteringMS, GroupingMS and OnReportStatusChangeMS.
The unit tests interacts with a different but real NoSql Database, identical to the NoSql Database of production. While for tests the interaction with the Image Recognition API have been stubbed since each call of the API costs ≈0.001€.

### 5.3.1. Unit tests for ApprovingMS

Lines coverage reached for the microservice: 100% (of a total of 115 raw lines).

| **Test ID** | **M_Appr_1** |
|---|---|
| Functionality tested | Approval of the report. |
| Input | One valid violation report with the image recognition methods stubbed to return an affirmative answer to the presence of a vehicle. |
| Assertions | Violation report approved as expected. |

| **Test ID** | **M_Appr_2** |
|---|---|
| Functionality tested | Rejection of the report. |
| Input | One valid violation report with the image recognition methods stubbed to return a negative answer to the presence of a vehicle. |
| Assertions | Violation report rejected as expected. |

### 5.3.2. Unit tests for ClusteringMS

Lines coverage reached for the microservice: 97% (of a total of 140 raw lines).

| **Test ID** | **M_Clust_1** |
|---|---|
| Functionality tested | Creation of a new cluster. |
| Input | One new group. |
| Assertions | Cluster created with the information of the group and with reference to the group. |

| **Test ID** | **M_Clust_2** |
|---|---|
| Functionality tested | Incremental change of a cluster by adding a group to it. |
| Input | One new group followed immediately by a group with a near location. |
| Assertions | The cluster is correctly created with the information of the first group, then the second group is added as a reference to it. |

| **Test ID** | **M_Clust_3** |
|---|---|
| Functionality tested | No clusterization if location is distant. |
| Input | One new group followed immediately by a group with a distant location. |
| Assertions | Two clusters are correctly created. |

### 5.3.3. Unit tests for GroupingMS

Lines coverage reached for the microservice: 100% (of a total of 171 raw lines).

| **Test ID** | **M_Group_1** |
|---|---|
| Functionality tested | Creation of a new group. |
| Input | One approved violation report. |
| Assertions | Group created with the information of the violation report and with reference to that report. |

| **Test ID** | **M_Group_2** |
|---|---|
| Functionality tested | Correctness of the trigger. |
| Input | One violation report that is not approved. |
| Assertions | No groups created as expected. |

| **Test ID** | **M_Group_3** |
|---|---|
| Functionality tested | Incremental change of a group by adding a report to it. |
| Input | One new approved report followed immediately by an approved report with a near location and similar timestamp. |
| Assertions | The group is correctly created with the information of the first report, then the second report is added as a reference to it. |

| **Test ID** | **M_Group_4** |
|---|---|
| Functionality tested | No grouping if location is distant. |
| Input | One new approved report followed immediately by a report with a distant location. |
| Assertions | Two groups are correctly created. |

| **Test ID** | **M_Group_5** |
|---|---|
| Functionality tested | No grouping if timestamp is not similar. |
| Input | One new approved report followed immediately by a report with a distant time. |
| Assertions | Two groups are correctly created. |

| **Test ID** | **M_Group_6** |
|---|---|
| Functionality tested | Incremental change of a group by adding a report to it for edge cases with similar but precedent timestamp. |
| Input | One new approved report followed immediately by an approved report with a near location and similar but precedent timestamp. |
| Assertions | The group is correctly created with the information of the first report, then the second report is added as a reference to it. |

### 5.3.4. Unit tests for OnReportStatusChangeMS

Lines coverage reached for the microservice: 100% (of a total of 61 raw lines).

| **Test ID** | **M_ORSC_1** |
|---|---|
| Functionality tested | Status change of a group of reports set as correct. |
| Input | One group set as correct. |
| Assertions | Status of reports updated to match status of the group. |

| **Test ID** | **M_ORSC_2** |
|---|---|
| Functionality tested | Status change of a group of reports set as rejected. |
| Input | One group set as rejected |
| Assertions | Status of reports updated to match status of the group. |

## 5.4. Integration Test

In order to test the interaction between the different subsystems a series of violation reports have been created specifically to test grouping, clustering, approval and the correct visualization on the map. They are reported here:

Account Test1@mail.com:

1.1: Type: PARKING_IN_A_PROHIBITED_SPACE
Plate: AB001AA

1.2: Type: PARKING_ON_A_SIDEWALK
Plate: AB000AA

1.3: Type: PARKING_TOO_CLOSE_AN_INTERSECTION
Plate: AB002AA

1.4: Type: DOUBLE_PARKING
Plate: AB002AA

1.5: Type: PARKING_IN_A_HANDICAPPED_ZONE
Plate: AB002AA

1.6: Type: PARKING_AT_A_PARKING_METER_WITHOUT_PAYING
Plate: AB000AA

1.7: Type: PARKING_IN_A_ZTL
Plate: AB001AA

1.8: Type: PARKING_FOR_LONGER_THAN_THE_MAXIMUM_TIME
Plate: AB000AA
1.9: Type: PARKING_OUTSIDE_MARKED_SQUARES
Plate: AB000AA
1.10: Type: OTHER
Plate: AB000AA
1.11: Type: PARKING_ON_A_SIDEWALK
Plate: AB005AA

Account Test2@mail.com:
2.1: Type: DOUBLE_PARKING
Plate: AB002AA
2.2: Type: PARKING_AT_A_PARKING_METER_WITHOUT_PAYING
Plate: AB000AA
2.3: Type: DOUBLE_PARKING
Plate: AB003AA
2.4: Type: DOUBLE_PARKING
Plate: AB004AA
2.5: Type: PARKING_OUTSIDE_MARKED_SQUARES
Plate: AB000AA

All of these reports are located in *Piazza A. Diaz, Milano* and use a picture of the same vehicle since this is not the focus of this test. The system should create the following groups with more than one violation report:

Group 1: 1.4, 2.1
Group 2: 1.6, 2.2
Group 3: 1.9, 2.5

Also we expect the system to create the following clusters with more than two groups:

Cluster 1: 1.2, 1.11
Cluster 2: [1.4, 2.1], 2.6, 2.4

After all of these reports have been made, a check on the database, on the map of the application and on the map of the web has been done.
The system has performed as expected, creating the correct groups and clusters, allowing the correct visualization on both the map of the application and the web map.

The second phase consisted in approving and rejecting the different groups from the municipality web interface in order to verify the correct behaviour of the system.
The system correctly updated the data structures.

# 6. Installation instruction

## 6.1. Violation reporting

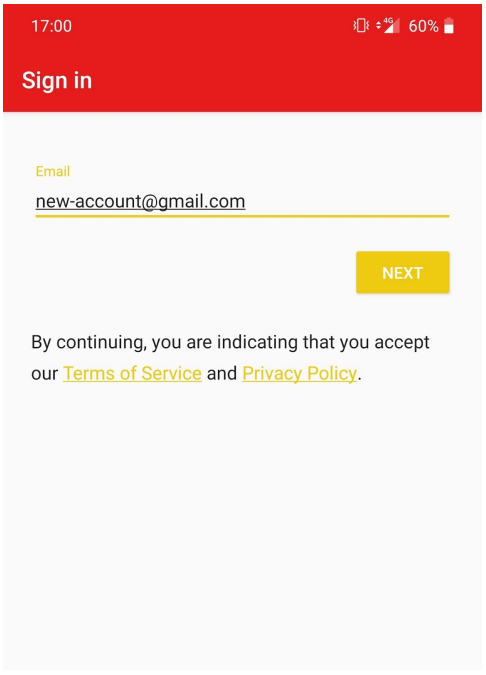
To start making reports it is needed to download and install the SafeStreets Android application.

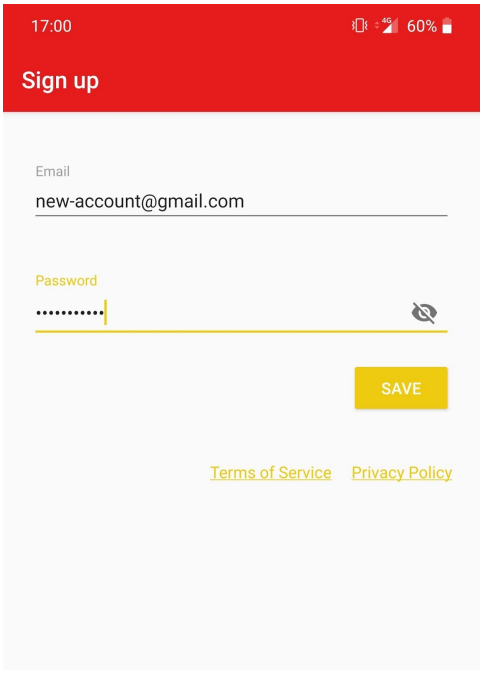
### 6.1.1. Download and install the SafeStreets Android Application

The *.apk* of the application can be downloaded from here: https://github.com/[...]/SafeStreets_1.0.1.apk
Then the *.apk* can be installed on any Android device (for more information on how to install an *.apk* a guide is available here) or on an Android Emulator (a guide to run apps on an Android Emulator is available here).

### 6.1.2. Run the application and create a new SafeStreets account

For the first installation it is needed to create a new account.
To create a new account it is just needed to insert an email, not associated with any account, in the “sign-in” screen of the application and follow the procedure.



## 6.2. Municipality access

The SafeStreets municipality interface is available at this address:
https://safestreets-project.web.app/
Note that only an authorized account (registered as a municipality) can sign-in in this interface.

### 6.2.1. Municipality account authorization

Municipality accounts are created on request by the platform administrators. It only requires an email to be associated to the account.
For testing purposes we created this account that already has access to the municipality of “Milano”:

Email: polizia@milano.it
Password: safestreets

For requesting the creation of other municipality accounts contact us at safestreetspolimi@gmail.com.
Note: one email can have access to maximum one municipality.

# 7. Effort spent

| Andrea Marcer | | |
|---|---|---|
| Date | Task(s) | Time |
| 12/12-12/01 | Implementation | ≈54 |
| 6/01 | Document structure | 10 min |
| 6/01 | Implemented Requirement | 1 hr 50 min |
| 8/01 | Tests | 2 hr |
| 11/01 | Tests and general overview | 1 hr |
| 12/01 | Application test and general overview | 3 hr |
| **Total:** | | 62 hr |

| Matteo Marchisana | | |
|---|---|---|
| Date | Task(s) | Time |
| 12/12-12/01 | Implementation | ≈52 hr |
| 7/01 | ITD | 1 hr 30 min |
| 8/01 | Tests | 1 hr |
| 10/01 | Code structure | 1 hr |
| 12/01 | Review | 1 hr 30 min |
| **Total:** | | 57 hr |

| Dennis Motta | | |
|---|---|---|
| Date | Task(s) | Time |
| 09/12-08/01 | Implementation | ≈51 hr |
| 08/01 | Cloud source code structure | 45 min |
| 09/01 | Cloud source code structure; Cloud testing | 1 hr 45 min |
| 10/01 | Implemented requirements of Database & Storage | 1 hr 15 min |
| 12/01 | Installation instruction; Adopted Framework of Cloud; Review | 3 hr 30 min |
| **Total:** | | 58 hr 15 min |