POLITECNICO
MILANO 1863

Department of Computer Science and Engineering

Implementation and testing document (ITD)

SafeStreets

- v1.0 -

Authors:	*Student Number:*
Morreale Federico	945258
Maddes Evandro	945642
Innocente Federico	870726

Github repository
Source code repository

January 12th , 2020

Contents

List of Tables

Chapter 1

Introduction

1.1 Purpose

This Document contains a detailed and exhaustive explanation of the implementation of the SafeStreets project. The purpose of this document is to explicitly detail the rationale of all choices made and provide a complete overview of how the project source code is structured and designed. The document also tackles the software testing activities and shows what are the main tests that have been performed on the product.

1.2 Scope

This document is a follow up on the previous two, *Requirement analysis and Specification document* and *Design Document*: it will describe the prototype for SafeStreets system that has been developed in accordance to them. It's important to remember that SafeStreets intends to provide users with the possibility to notify authorities when traffic violations occur, and in particular parking violations. The application allows users to send pictures of violations, including their date, time, and position, to authorities. In more details, SafeStreets stores the information provided by users, completing it with suitable meta-data. In addition, the application allows both end users and authorities to mine the information that has been received. Of course, different levels of visibility could be offered to different roles. If the municipality offers a service that allows users to retrieve the information about the accidents that occur on the territory of the municipality, SafeStreets can cross this information with its own data to identify potentially unsafe areas, and suggest possible interventions. In addition, the municipality could offer a service that takes the information about the violations coming from SafeStreets, and generates traffic tickets from it. This addition information about issued tickets can be used by SafeStreets to build statistics.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **User:** is a general customer that use the application. It is used to refer to both an authority and a citizen.
- **Citizen:** is the basic customer of the application. He can upload violations and query the system to get statistics on a selected area.
- **Authority:** advanced customer of the application, is a registered user that is qualified to make fines and view sensitive information. Must verify himself during the registration.
- **Municipality:** is always intended as the local municipality. Every municipality provides and receives information exactly and only about its own jurisdiction.
- **Violation:** is a general infringement reported by a user.
- **Dossier:** is the instance of a violation on the system, that includes the pictures, position, classification, textual specification and the mark.
- **Authority database**: is the database in which are stored all data about incidents. This data are uploaded by authority while the interface to access the database is offered by the municipality. SafeStreets uses this interface to retrieve data from the database.

1.3.2 Acronyms

- RASD: Requirement Analysis and Specification Document
- LP: License Plate
- GPS: Global Positioning System
- API: Application Programming Interface
- UML: Unified Modeling Language

1.3.3 Abbreviations

- $[Gn]$: n-goal.
- $[Dn]$: n-domain assumption.

- $[Rn]$: n-functional requirement.
- $[UCn]$: n-use case.

1.4 Revision history

- V1.0, January 12th 2020: First release

1.5 Document structure

Chapter 1: Introduction. A general introduction and overview of the *Implementation and testing document*. It aims giving general but exhaustive information about what this document is going to explain.
Chapter 2: Implemented requirements. The requirements/functions that are actually implemented in the software.
Chapter 3: Development frameworks. It includes adopted development frameworks with references to sections in the DD. It presents adopted programming languages and its advantages and disadvantages and middleware used and its advantages and disadvantage.
Chapter 4: Structure of the source code. It explains and motivates how the source code is structured both in the front end and in the back end.
Chapter 5: Testing. It provides the main testing cases applied to the the application.
Chapter 6: Installation instruction. It explains how to install and set up the application.

Chapter 2

Requirements

This section describes which functional requirements were implemented in the presented SafeStreets prototype in reference to the ones already stated in the RASD.

2.1 Implemented requirements

We decided to implement two of the three functionalities that define the SafeStreets project. In addition to the basic functionality where the application stores the information provided by users, completing it with suitable metadata, we developed also the feature that allows the generation of traffic tickets from the municipality and the building of statistics from the application. We also implemented query to the database to retrieve specific reports after having set some parameters.
Below is shown a mapping of requirements and the implemented ones.

[R1] Unregistered users cannot use SafeStreets.

- Implemented.

[R2] In signing up, users must provide an email and a password.

- Implemented.

[R3] In signing up, an authority must also provide an ID.

- Implemented.

[R4] In signing up, users must accept "terms & conditions".

- Implemented.

[R5] SafeStreets identifies a user by its email.

- Implemented.

[R6] SafeStreets collects user data in its database.

- Implemented.

[R7] SafeStreets stores all reports in its database.

- Implemented.

[R8] SafeStreets retrieves reports from its database.

- Implemented.

[R9] SafeStreets authenticates a user when tries to log in.

- Implemented.

[R10] Users can view the reports in the map.

- Implemented.

[R11] Authorities can see all the sensitive information contained in a report.

- Implemented.

[R12] Citizens cannot see sensitive information in a report.

- Implemented.

[R13] Users can view unsafe areas in the map.

- Not implemented.

[R14] Users can delete reports they submitted within a day.

- Implemented.

[R15] Users can edit reports they submitted within a day.

- Not implemented.

[R16] Users can upload *valid* reports.

- Implemented.

[R17] A report must contain at least one image.

- Implemented.

[R18] A report must indicate the type of violation.

- Implemented.

[R19] A report is *valid* if and only if R17 and R18 are satisfied.

- Implemented.

[R20] Images containing sensitive information (like license plates) must be blurred by the system.

- Implemented (For now, you need a high resolution of the device, still to be fixed).

[R21] The system must notify the user if the report submitted is not valid.

- Implemented.

[R22] For each type of statistics there must be at least one user from which SafeStreets takes data.

- Implemented.

[R23] A user can access to feedback area for each report.

- Implemented.

[R24] A user can select negative feedback for each report.

- Implemented.

[R25] Authorities can see suggestions for possible interventions.

- Not implemented.

[R26] During adding a new report, if there is a report of same type, in the same position and at the same date, SafeStreets shows the report stored to the user if the violation is the same.

- Implemented.

[R27] Authorities can mark a report as "fined".

- Implemented.

[R28] User can query SafeStreets to achieve a specific report.

- Implemented.

[R29] SafeStreets retrieves incidents from municipality database.

- Not implemented.

R	Implemented
1	*Yes*
2	*Yes*
3	*Yes*
4	*Yes*
5	*Yes*
6	*Yes*
7	*Yes*
8	*Yes*
9	*Yes*
10	*Yes*
11	*Yes*
12	*Yes*
13	*No*
14	*Yes*
15	*No*
16	*Yes*
17	*Yes*
18	*Yes*
19	*Yes*
20	*Yes*
21	*Yes*
22	*Yes*
23	*Yes*
24	*Yes*
25	*No*
26	*Yes*
27	*Yes*
28	*Yes*
29	*No*

Table 2.1: *Traceability matrix*

Chapter 3

Development frameworks

In the following chapter is presented a list of the frameworks and technologies used to develop SafeStreets system.

3.1 Adopted programming language

The programming languages that have been used are:

3.1.1 Dart

Dart is an object-oriented, class-based, client-optimized programming language used to build apps on multiple platform. It has been used as language of the whole system, because of its support of Flutter framework.

Advantages

- Flutter support: the main pro in the choice of dart is the support of flutter framework.
- Object-oriented: allow to create a modular program and simplify code reusability.
- Ahead of time: dart is ahead of time compiled to fast and predictable native code, to make it fast and fully customizable.
- Just in time: dart can be just-in-time compiled for an exceptionally fast development cycle.
- Secure: the dart language, compiler, interpreter, and runtime environment were each developed with security as objective.

Disadvantages

- Maturity: not as mature as other programming languages, like java.

3.2 Adopted middle-wares

3.2.1 Flutter

Flutter is an user interface development kit developed by Google used to build application for multiple systems (mobile, web and desktop) from a single codebase. It is based on Dart programming language.

Advantages

- Hot reload: it is possible to see the changes made to the code right away in the app.
- Cross platform: flutter can be used to build applications for different platform with just one codebase.
- Widgets: flutter takes everything on a widget approach, modeling and making everything intuitive.
- Google support: since flutter has been developed by Google, it gives an easy and optimized interaction with the other Google products.

Disadvantages

- Maturity: even if the flutter is supported by Google that provides all its feature and many useful libraries, the framework is relatively new, so it may miss something compared with some older one.

3.2.2 Firebase

Firebase is a Google's mobile platform that allows to store and synchronize data from web applications and mobile apps. It is a NoSQL database with great resources, high availability and that can be quickly integrated into other software projects. Firebase stores data as huge JSON file document with dynamic schema. It provides both a data and file database, and since it does not store data in table like the usual relational databases, it is more flexible to store different type of information.

Advantages

- Easy implementation: Firebase is thought to be implemented in mobile and web application, so it perfectly fits in the context of SafeStreets.
- Scalability: Firebase can easily scale in terms of size, virtually to infinity since it is provided by Google.

- Data synchronization: Firebase can instantly update data, making it powerful in the context of the application.
- Secure: data stored in Firebase are periodically backed up and always encrypted when are managed.
- Cost reduction: Firebase allows to bypass the managing of backend databases and corresponding hardware.
- User interface: it is possible to access to data via user interface.

Disadvantages

- Limited querying capabilities: since the database is implemented as a single file, it is not possible to define relation like in a classic relational database, and because of that complex query are more difficult to be managed.

3.2.3 Mockito

Mockito is a mocking framework used for testing. The framework is used to unit test the code with the help of Junit. Moreover, it allows the creation of test double object (mock objects) in automated unit tests for the purpose of test-driven development.

3.3 Additional APIs

3.3.1 Google Maps

Google Maps is Google's web mapping service. It is one of the most accurate mapping service, and it can be easily integrated with Flutter since both of them have been developed by Google.

3.3.2 Plate Recognizer

Plate Recognizer is a web service used to find the plates in a picture. It can spot the position of the plates and read them, and works with the plats of almost every country in the world.

Chapter 4

Structure of the source code

The following section's aim is to explain the structure of the Flutter project managing the SafeStreet operations. This section only explains the content of every file so does not deep into details; for more information, please look at the source code. This section will cover only the content of *lib* folder, as the *android* and *ios* folder contain platform specific files, automatically generated by the Flutter project creator. In this folder there are two files:

- **auth_manager.dart**: this manager is used to redirect the user, to the right page when it opens the application.
- **main.dart**: the main entry point of the app; it loads the first screen shown to the user.

There are also three subfolder which are described in the following sections.

4.1 Model

This is the model component of the MVVM pattern chosen for this project. Model refers either to a domain model, which represents real state content, or to the data access layer, which represents content. In this folder there are three subfolders.

4.1.1 Enum

- **level.dart**: different level of visibility.
- **violation.dart**: different type of the violations, more can be included here.

There is also present another file **location.dart** which manages the position that is used by different classes of SafeStreets.

4.1.2 Report

It contains the different representations of a report and its components.

- **report.dart**:this is the abstract class of a report in SafeStreets.

- **report_to_get.dart**: this class is used to create report objects that are fetched as map from firebase.
- **report_to_send.dart**: this class is used to create report object to send to firebase.
- **violation_image.dart**: this is the image object created by a user when he takes a photo.

4.1.3 User

It contains the different type of users of the app.

- **user.dart**: this is the abstract class of a user in SafeStreets.
- **authority.dart**: this is one type of user.
- **citizen.dart**: this is another type of user.

4.2 View model

The view model is an abstraction of the view exposing public properties and commands. In this folder there are all the managers already presented in the design document (sec 2.2).

- **access_manager.dart**: it manages the sign up and login of the users.
- **firebase_storage_services.dart**: it manages the uploading of pictures on firebase' storage.
- **new_report_manager.dart**: it manages the insertion of a new report.
- **report_map_manager.dart**:it takes care of showing on the map all the reports with the correct color of the marker.
- **stats_manager.dart**: it takes care of building statistics on the app through specific algorithms.
- **user_report_visualization_manager.dart**: it manages the visualization of all reports of one user.
- **utilities.dart**: in this class there are methods used as helpers.
- **violation_query_manager.dart**: it manages all the procedures to make a query on the database to retrieve same specific data.

4.3 View

It displays a representation of the model and receives the user's interaction with the view (clicks, keyboard, gestures, etc.), and it forwards the handling of these to the view model.

- **create_report_page.dart**: this is the creation page of a report.
- **home_page.dart**: this is the home page of the application, from here we can navigate to the other sections of the app, like statistics, query page, my reports and logout.
- **my_reports_page.dart**: here is the list page of all the reports submitted by the logged user.
- **sign_in_page.dart**: the login in page.
- **sign_up_page.dart**: the sign up page.
- **statistics_citizen_page.dart**: this is the statistics page that displays different type of statistics without sensitive information.
- **statistics_authority_page.dart**: this is the statistics page that displays different type of statistics including sensitive information.
- **view_report_authority_page.dart**: this is the authority view of a report.
- **view_report_citizen_page.dart**: this is the citizen view of a report.
- **violation_query_page.dart**: this page displays different filters to set a query.

Chapter 5

Testing

The testing has been done mostly as written in the DD. We performed two different types of tests to validate the behaviour of the complete system: front-end and back-end. In each of the following sections there is a brief description of the group test and a list of test cases.
For more information we suggest to see the *source code of the tests*

5.1 Unit testing

The purpose of a unit test is to validate that each unit of the software performs as designed.
Every test is formed by three section:

- setup: in this section there is the setup and the instantiation of the elements useful during the test.
- run: the core section where functions are called.
- verify: through the command *expect()* there is the comparison between the results and the expected one.

We tested the main features: the building of the statistics, the violation query, the visualization of the reports on the map, the addition of a new report and the user action on a report (send a negative feedback for a picture or delete a picture from report).

5.1.1 Build statistics

We tested all the three types of statistics and for each type we built different test cases to validate different situation:

- effectiveness:
    1. One report uploaded and not fined;
    2. One report uploaded and fined;
    3. Two reports uploaded: one fined and one not;

- daily reports:
  1. Zero violation uploaded today;
  2. One violation uploaded today;
  3. one violation uploaded today and one yesterday;
- most committed violation:
  1. Zero violation uploaded, no most committed violation;
  2. One violation uploaded is the most committed;
  3. Three violations uploaded of the same type are the most committed;
  4. Two violations uploaded of the same type and one of another type, the two ones are the most committed;

  All tests have the expected behaviour, so it performs as designed.

5.1.2 Violation query

To make a violation query, a generic user must set different parameters: city, type of violation and the period in which it wants to retrieve the information (from date, to date). To analyze the behavior of violation query manager, we built a test case for each combination of the previous parameters.

1. One report respects all the bounds of the query;
2. Two report respects the bounds of the query (no bounds on city);
3. Two report respects the bounds of the query (no bounds on violation);
4. Two report respects the bounds of the query (no bounds on timeTo);
5. Two report respects the bounds of the query (no bounds on timeFrom);
6. Two report respects the bounds of the query (no bounds on timeFrom and timeTo);
7. Two report respects the bounds of the query (no bounds on violation and timeTo);
8. Two report respects the bounds of the query (no bounds on violation and timeFrom);
9. Two report respects the bounds of the query (no bounds on violation and timeFrom and timeTo);
10. Two report respects the bounds of the query (no bounds on city and timeTo);

11. Two report respects the bounds of the query (no bounds on city and timeFrom);
12. Two report respects the bounds of the query (no bounds on city and timeFrom and timeTo);
13. Two report respects the bounds of the query (no bounds on city and violation);
14. Two report respects the bounds of the query (no bounds on city and violation and timeTo);
15. Two report respects the bounds of the query (no bounds on city and violation and timeFrom);
16. Two report respects the bounds of the query (no bounds on city and violation and timeFrom and timeTo);

All tests have the expected behaviour, so it performs as designed.

5.1.3 Report visualization on map

We tested the interaction between the reports stored and the reports on the map; in particular we tested the bound on the visualization of the reports of the last 24 hours.

1. No violation uploaded, no violation on the map;
2. Two violations not fined today, two violations on the map;
3. Two reports not fined yesterday, zero violation on the map;
4. Report uploaded yesterday, zero violations on the map;
5. Two reports,only one today, one violation on the map;
6. Two reports on same position, two reports on the map;

All tests set the expected number of reports in the correct position.

5.1.4 Add a new report

In this group of tests we validated the behavior of the app when it receives a new report. In particular the addition of a report already uploaded by someone.

1. Similar report doesn't exist;
2. Report with same location and day;

3. Report with same location and violation;

Test case 1 is the base case so the report is added.
Test cases 2 and 3 rejects the addition of a new report because there is already one report in the database with same location and day/violation.

5.1.5 User feedback and action

We validated all the methods that allow users to give a bad feedback to an image and to delete an image from the report to be sent.

1. Add the user to the list of violation feedback sent;
2. Add a feedback to a picture;
3. Delete one picture from the report to be sent;

All tests have the expected behaviour so performs as designed.

5.2 Widget testing

A widget test validates a single widget. The goal of a widget test is to verify that the widget's UI looks and interacts as expected so they can actually test the UI. Note that the reason why they are only present tests on log in and sign up page is because all the other widgets are tested in the integration testing (next section).

5.2.1 Login

1. Email, password and button are found;
2. It validates empty email and password;
3. It calls sign in method when email and password is entered;

5.2.2 Sign up

1. Email, password, confirm password and button are found;
2. It validates empty fields;
3. It calls sign up method when email and password is entered,

5.3 Integration testing

The integration testing validates the integration and interaction of different features. The goal of the integration testing is to verify that the system works properly and that all the features coexist and communicate correctly. These tests are self driven, and works as a simulation of the application behaviour. The simulation can be run by using the following command in the safe_streets folder:

```
flutter drive --target=test_driver/app.dart
```

If you are using iOS simulator, if the user is already signed you must logout first. To success the last two tests ("feedback a violation for the fist time" and "feedback a violation for the second time"), there must be at least one report located in Milan uploaded on Firebase's database. This was not considered as a limitation, even if it generates a test that may fail, since it is reasonable to think that it will always be present this kind of report because of the provenience of the application, and the fact that a test like that is not thought to be run frequently (since every 5 runs of those tests a report is deleted, see the DD document for more specifications).

5.3.1 Tests

Here a list of all the integration tests is presented

1. The user tries to login with an unregistered email and password.
2. The user tries to register himself without accepting terms and conditions.
3. The user registers himself with a new email and password.
4. The user tries to upload a report without any image.
5. The user checks that there are no reports associated with his account.
6. The user queries all the reports uploaded in Milan.
7. The user gives a feedback to the first report queried.
8. The user tries to give a second feedback to the first report queried.

Chapter 6

Installation Instructions

6.1 Flutter Application Setup

In this section will be presented all the necessary information to build the Flutter project.

6.1.1 Install Flutter Framework

Detailed installation instructions of the framework depending on the operating system (Windows, MacOS, Linux) can be found following *this* link. our team did not test flutter on Windows.

6.1.2 Install Plugins on Android Studio

Flutter runs on both Intellij and Android Studio IDE, so you can choose which one to use. A flutter plugin for Xcode is not available, so the application for an IOS device must be run using one of the following tools. Android studio can be found *here*. Once you have installed it, you have to download the plugins for Flutter and Dart (that is the programming language on which flutter is built). In order to do that:

1. Start Android Studio.
2. Open plugin preferences (Preferences > Plugins on macOS, File > Settings > Plugins on Windows and Linux).
3. Select Marketplace, select the Flutter plugin and click Install.
4. Click Yes when prompted to install the Dart plugin.
5. Click Restart when prompted.

6.1.3 Install Plugins on IntelliJ

IntelliJ can be found *here*. you need version 2017.1 or later. Once you have installed it, you have to download the plugins for Flutter and Dart (that is the programming language on which flutter is built). In order to do that:

1. Start IntelliJ.

2. Open plugins tab (File > Settings > Plugins).
3. Select Marketplace, search for the Flutter plugin and click Install.
4. Click Yes when prompted to install the Dart plugin. If it does not ask it, install it manually.
5. Click Restart when prompted.

6.1.4 Get Project Dependencies

Once you have done everything above, you can now navigate to 'MorrealeMaddesInnocente /safe_streets/pubspec.yaml' and click on packages get on the top right (if you are using Intellij), or simply open the terminal and run 'flutter packages get'. If something goes wrong, run 'flutter doctor' in the terminal.

6.1.5 Install Android Emulator on Android Studio

Our team used IntelliJ, so we leave the installation instructions *here*. a FULL-HD screen resolution device is recommended(1080x1920).

6.1.6 Install Android Emulator on IntelliJ

On IntelliJ:

1. Go to Tools > Android > AVD Manager
2. Click on Create Virtual Device
3. Select a Device (a FULL-HD screen resolution device is recommended(1080x1920))
4. Click Next
5. Download Pie or Q from the list, then click Next
6. Click Finish

You have now installed android emulator.

6.1.7 Install IOS Simulator

To install IOS simulator it is necessary to download Xcode, which can be found *here*. Xcode can be downloaded only on a MacOS device, so it is not possible to simulate an iphone on a Windows or Linux system. On Xcode:

1. Open Preferences
2. Go to Location panel
3. Select the most recent version in the Command Line Tool dropdown

Although the iOS simulator is great for rapid development, it does come with a few limitations. Some hardware is unavailable in the Simulator, like the audio input and the motion support. In particular, the camera is unavailable, and this is important in the context of the application because that limitation does not allow the user to upload a new report. To supply this lack, the user must use a mobile device to run the application, or use android emulator.

6.1.8 Run the IOS Simulator

After the Xcode configuration, the iOS Simulator will be available in the dropdown window for both IntelliJ and Android simulation. In case it is not available, it is possible to open the simulator by running this command on Terminal:

```
open -a Simulator
```

6.1.9 Use your own Android phone

Alternatively, you can use your own smartphone to test the app, you have to activate debug usb from the developer options of your device and have adb services installed on your PC.

6.1.10 Use your own iOS phone

To supply the lack of the Simulator's hardware, it is possible to run the application on a iOS device. To do that, you must be log-in in Xcode with your appleID and registered as developer (do it on Xcode > Preferences > Account). Than you must register your device:

1. Open the safe_streets/ios folder with Xcode
2. Select the Runner project in the left navigator
3. Ensure that the device is connected to Xcode
4. Ensure that the selected device on the top left corner of Xcode window is your mobile device

5. Select your development team under General > Signing > Team

Now it is possible to run the application on the iOS device by selecting it on the dropdown window inside the IDE. To run the application, the iOS device must have trusted the computer.

6.1.11 Run the Application

Open the emulator (or connect your device) and then go to 'MorrealeMaddesInnocente/safe_streets/lib/main.dart' and click on the green arrow. If you do not find the arrow, go to Edit Configurations and click on the '+' top left, then click Flutter and in the 'Dart Entrypoint' you select the file main.dart. Now you can run the App, the first run will take longer than usual, don't worry.