POLITECNICO MILANO 1863

Academic Year 2019 / 2020

Safe Streets

https://github.com/andreacappelletti97/CappellettiMaglione

https://play.google.com/store/apps/details?id=com.cmprogrammers.safestreets

Cappelletti Andrea, Maglione Sandro

Implementation and Test deliverable (ITD)

Deliverable:	ITD
Title:	Implementation and Test deliverable
Authors:	Cappelletti Andrea, Maglione Sandro
Version:	1.0
Date:	12 January 2020
Download page:	https://github.com/andreacappelletti97/CappellettiMaglione
Copyright:	Copyright © 2020, Cappelletti Andrea, Maglione Sandro – All rights reserved

Contents

Table of Contents . . . 3

1 Introduction . . . **5**
1.1 Purpose . . . 5
1.2 Scope . . . 5
1.3 Definitions, Acronyms, Abbreviations . . . 5
1.4 Revision history . . . 5
1.5 Document Structure . . . 6

2 Implementation . . . **7**
2.1 Functions implemented . . . 7
2.1.1 Login - Sign In . . . 7
2.1.2 Registration - Sign Up . . . 7
2.1.3 Logout . . . 7
2.1.4 Report a violation . . . 7
2.1.5 Report History . . . 7
2.1.6 Area Safety . . . 7
2.1.7 License Plate LookUp . . . 7
2.1.8 Notification System . . . 8
2.1.9 Area Violation Look-Up . . . 8
2.2 Functions not implemented . . . 8
2.2.1 Interventions suggestion . . . 8
2.2.2 Validation of reports . . . 8

3 Framework . . . **9**
3.1 Flutter . . . 9
3.1.1 Choice reasons . . . 9
3.2 Dart . . . 9
3.2.1 Advantages . . . 10
3.2.2 Disadvantages . . . 10
3.3 Firebase . . . 10
3.3.1 Choice reasons . . . 10
3.4 Firebase Functions . . . 10
3.5 Firebase Auth . . . 10
3.6 Cloud Firestore . . . 10
3.7 Firebase Storage . . . 11
3.8 Firebase ML Vision . . . 11
3.9 Firebase Messaging . . . 11
3.10 Third party services and plugins . . . 11
3.10.1 Flutter BLoC . . . 11
3.10.2 Data Connection Checker . . . 11
3.10.3 Image Picker . . . 11
3.10.4 Geolocator . . . 11

4 Source code structure . . . **12**
4.1 Data . . . 12
4.1.1 Data sources . . . 13
4.1.2 Models . . . 13
4.1.3 Repositories implementations . . . 14

4.2 Domain . . . 14
4.2.1 Repositories interfaces . . . 14
4.2.2 Use cases . . . 15
4.2.3 View models . . . 15
4.3 Presentation . . . 15
4.3.1 Bloc . . . 15
4.3.2 Screens . . . 16
4.3.3 Views . . . 16
4.3.4 Widgets . . . 16
4.4 Core . . . 16
4.4.1 Services . . . 17
4.4.2 Enums . . . 17
4.4.3 Utils . . . 17
4.4.4 Network . . . 17
4.5 Test . . . 17

5 Testing . . . 18
5.1 Strategy . . . 18
5.2 Main test cases . . . 18

6 Installation . . . 19
6.1 Install the framework . . . 19
6.2 Install the application . . . 19

7 Appendix . . . 20
7.1 Used tools . . . 20
7.2 Hours of work . . . 20
7.2.1 Cappelletti Andrea . . . 20
7.2.2 Maglione Sandro . . . 21

1 Introduction

1.1 Purpose

The following document represents the Implementation and Test Deliverable (ITD) for the system SafeStreets. The Goals of the Implementation and Test Deliverable Document are:

- Introduce the reader to the technologies used during the development
- Exaplain in detail the Implementation choices adopted
- Underline advantages and disadvantages of the framework
- Define the code structure and explain its peculiarities
- Exaplain the testing strategy and test cases
- Provide to the reader a guide for the installation of the system

This document is intended for design stakeholders such as Developers, Software Enginners, Software Designers and Project Managers.

1.2 Scope

SafeStreets is a new platform, so it has to be developed from scratch. The prototype of SafeStreets will cover what we consider the basic functionalities of the service. Particularly Normal User would be able to report traffic violation to the Municipality. The report contains a picture of the violation, the current location (retrieved from the device's GPS), the type (input by the Normal User). The Machine Learning Algorithm will recognize from the picture the licese plate code, if the latter is not able to recognize the license plate code the Normal User will be asked to insert it manually. Users and Authorities can consult the Area Safety level: this index is computed based on the report received for that specific Area. Authorities can also search for a license plate and retrieve all the violations report related to that or retrieve all the reports of a specific Area. They can also enable a notification system divided by Area, to get a real-time notification when a violation occurs. Users have also access to the history of their report's uploads.

1.3 Definitions, Acronyms, Abbreviations

- **API:** Application Program Interface;
- **ML:** Machine Learning;
- **UI:** User Interface;
- **GPS:** Global Positioning System.
- **BLoC:** Business Logic Component

1.4 Revision history

ITD v1.0: Document registered and delivered on 12/01/2020.

1.5 Document Structure

This Implementation and Testing Deliverable Document is composed of different sections.

Section 1: in which the purpose of the Implementation and Testing Deliverable Document is explained, followed by the scope of the System, which provides the description of the problem and the application domain, and a reference of Definitions, Acronyms, Abbreviations which are used throughout the document.

Section 2: in which the Implementation choices are presented. We discuss the functions implemented and the functions not implemented and the related motivations behind our choices.

Section 3: in which Flutter, the framework used for the development, is presented and explained. We outline the choice reasons and discuss the pros and cons. Then we go through every platform we have used to Implement the prototype.

Section 4: in which the code structure is detailed explained. We list all the file we have created in order to make the prototype run.

Section 5: in which the testing strategy is exploited. Particularly we show how the application developed has passed the test cases we have created.

Section 6: in which we explain how to install the framework and the application.

Section 7: in which the used tool and the hours of work of each team member are presented. Each working session has been registered; we provide the date, a description of the work, the total amount of time and the start and end time for each session.

2 Implementation

2.1 Functions implemented

In this section we are going to list all the functions implemented. The functions listed below Authority are only accessible by an Authority, while all the others from both Users.

2.1.1 Login - Sign In

The login function is a required function for the application. It allows the system to distinguish between Normal User and Authority, indeed Authority are registered via pec-email and they have different privileges of access granted. In order to login is necessary to provide email and password.

2.1.2 Registration - Sign Up

The registration function allows Users to register to the system. If a User register itself via pec-email it will be saved as Authority. The registration requires name, surname, email, password. A confirmation of the password is not required because we have focused more on other functionalities of the system listed below.

2.1.3 Logout

The logout function allows User to log off from their account, after this action the login page will be displayed. This function complete the login, basically it has been implemented to make the application functional.

2.1.4 Report a violation

Through this functionality Users can upload a violation on the system. The process is pretty simple: the User takes a photo of the violation. The ML Algorithm works and try to retrieve the License Plate code from the Photo. If it fails, the User has to input the License Plate by hand. The system retrieve the location throught the GPS and let the User select the type of the violation, this is relevant to distinguish between different kinds of violations. Once the process is completed, the system uploads the new violation report. This is the main funcion of the application, that reflects is concept.

2.1.5 Report History

Users can access to their report history to see all the violations reported until that moment. The system will display the license plate, the picture taken, the lat and long of the Area and the type of violation. This function is useful to keep track of your report.

2.1.6 Area Safety

This function is useful to give the Users an idea of the Area safety. The indicator is based on the quantity of report received and the type.

Authority

2.1.7 License Plate LookUp

An Authority can search for a specific License Plate Code. The system will return all the report linked to that particular License Plate. In this way authorities can check real-time all the violations of a vehicle.

2.1.8 Notification System

An Authority get a notification every time a violation is uploaded in a specific Area. The Authority can also decide to enable/disable the notifications for a specific Area. In this way authorities can receive real-time notification based on their Areas of interest/work.

2.1.9 Area Violation Look-Up

An Authority can retrieve all the violation related to a specific Area. This function can help Authorities to identify areas with specific violations and apply improvements where possible.

2.2 Functions not implemented

In this section we present all the fucntions that have not been implemented. We focused only on the functions listed above to have a base running prototype of the application with a solid code base.

Authority

2.2.1 Interventions suggestion

This function was designed to suggest to Authorities possible interventions in the Area when requested. We did not developed this function because we consider it as a nice-to-have function. It does not add primary functionalities to the system.

2.2.2 Validation of reports

This function was designed to make the Authority validate each report every time a specific license plate reach 10 violations. We did not implemented this function because we do not consider it relevant for our prototype.

3 Framework

3.1 Flutter

The framework we choose to develop the SafeStreets application is called Flutter.
Flutter is a framework, made by Google, which has been released in 2017. It allows the development team to build mobile, web, and desktop applications in one codebase.
Here below the definition of Flutter from the official website (flutter.dev):

> Flutter is Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase

3.1.1 Choice reasons

Since the main interaction between the user and the application is the violation reporting process, which itself requires the user to take a picture of the violation, we thought that choosing a mobile centered application was an obvious consequence of the main goals of the system.

In the mobile development ecosystem, there exist mainly two options:

- Native code: Which requires the development team to be splitted in two to handle both an Android and IOS codebase.
- Cross-platform frameworks: Which allows one codebase to be exported for both Android and IOS.

The second option seems more reasonable and modern in our option.
After some research we discovered Flutter, a modern framework for building cross-platform applications.

Here below a list of reasons of why we choose this particular framework over others:

- Hot-reload: A great feature of Flutter is the possibility to repaint the UI instantly after each change in the code. It is therefore possible to quickly iterate between different options, which makes the development expereince much faster and enjoyable.
- Active community: Being a new and popular framework, a lot of people are currently contribuiting and expanding the potential of the framework.
- New features release: Since we started using Flutter for the project, a lot of new functionalities has been added to the framework. That is because Flutter is currently evolving fast, and the future prospect is really promising.
- Plugins: Flutter makes available a huge collection of open source plugins to handle the most common task for a mobile application. We exploited some of those plugins to develop some extra features that otherwise would have required too much effort and time.
- Firebase integration: Being developed by Google, Flutter integrates seamlessly with Firebase, which is our BaaS provider of choice (more information provided in section 3.3).
- Easy to learn: Flutter has been really easy to learn and master. A lot of resources are available online and the documentation is one of the best we found.

3.2 Dart

Dart is a object-oriented, statically typed programming language made by Google. Flutter applications are completely written in Dart, both on the front and back end side. Here below the definition of Dart from the official website (dart.dev):

> Dart is a client-optimized language for fast apps on any platform

3.2.1 Advantages

The main advantages of Dart are the same that made us choose Flutter as a framework.
Dart is seamlessly coupled with Flutter. Both are made and maintained by Google, which is spending a lot of effort in supporting those two open-source solutions. Dart enables the hot-reload feature, it has all the main features of a modern object-oriented programming language, and it has been develop from scratch to handle mobile application.
Furthermore, Dart has native support for all the main reactive programming patterns, which are crucial nowadays to develop a fast and reliable mobile application.

3.2.2 Disadvantages

The main disadvantage of Dart is that, because it is relatively young as a programming language, it is missing some common features of other more mature languages like Swift, Java, and Kotlin.
Nonetheless, some plugins and third party solutions are currently available to fix most of the most common issue that we found during development.

3.3 Firebase

Firebase is a BaaS platform, made by Google, which provides many services to handle the most common requirements of a mobile application.

3.3.1 Choice reasons

We choose not to develop and maintain our own back end server. We choose Firebase because it already integrates all the service which we needed for the application.
Furthermore, the pricing model of the platform allows a develop to exploit all the feature for free within a scale limit. Specifically, all the accounts are granted full access within certain usage limits, namely limits in the number of database reads, writes, deletes.

Here below we present a list of the main Firebase services we used to develop the application.

3.4 Firebase Functions

Firebase Functions is a back end solution provided by Firebase. It is already configured with ExpressJs. It allows us to write some back end functions in javascript, which will be called by the application. In this way, we can for example handle both the authentication and the user data storage process in the database at the same time from one common endpoint.

3.5 Firebase Auth

One of the main goals of SafeStreets is to allow each user store his/her personal data about reports and violations. In order to handle the authentication of the users, we used the Auth library of Firebase.
The library allows us to send the user's parameters, namely name, surname, email, and password, to the Firebase auth service, which will handle the whole authentication process automatically.

3.6 Cloud Firestore

Cloud Firestore is a NoSQL database solution offered by Firebase. It exposes an easy to use API to handle or the major requestes and use cases of the application. Inside Cloud Firestore we store all the reports, the users' data, and the requests.

3.7 Firebase Storage

Firebase Storage allows us to store on the cloud the pictures of the users' violation reports. Every time a user sends a report to the application, the server stores all the data inside the database, and uploads the violation image inside Firebase Storage.

3.8 Firebase ML Vision

In order to recognize the license plate from the violation's picture, we used Firebase ML Vision. After the user takes the picture, a Firebase Machine Learning model is called to recognize the license plate inside the picture. Since it is possible that the service may fail, the user is required to manually confirm the recognized code.

3.9 Firebase Messaging

One of the requirements of the system is to send notifications to users when certain triggers occur. We use Firebase Messaging to subscribe users to topics and recive notifications when a new report is stored inside the database. Firebase Messaging integrates with Firebase Functions to send the actual notification to the user device.

3.10 Third party services and plugins

Here below a list of the main third party services and libraries used to develop the main functionalities of the app.
Some more libraries not listed here has been also integrated to handle some more common development task and to improve the development efficiency.

3.10.1 Flutter BLoC

Flutter BLoC is a state management library made by Google for Flutter.
It allows us to decouple the front end code of the application from the business logic. Flutter BLoC works with Stream of data: the UI triggers events after each user interactions; those events are listened by a BLoC class, which returns a new application State. The UI is then listening to this state changes and it rebuilds itself accordingly.
All this process is seamlessly handled by the library, which allows us to focus on implementing the core functionalities of the application.

3.10.2 Data Connection Checker

This small library is used to check the internet connection availability of the device. If the internet connection is not available, the system cannot work and connect with the back end server to retrieve the data.

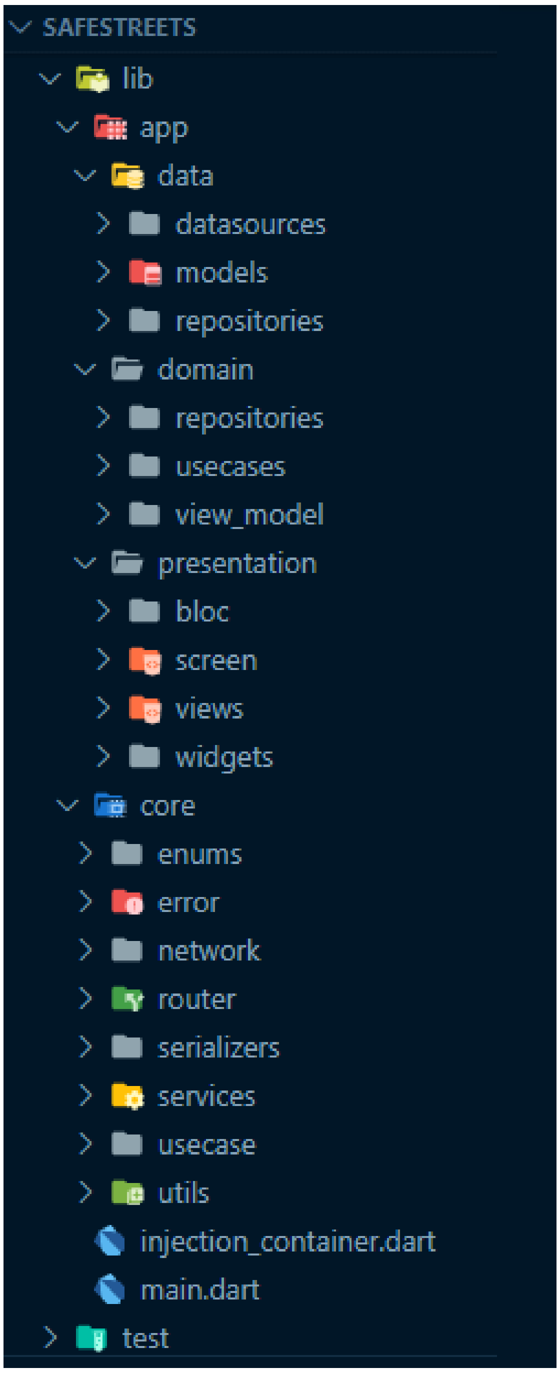
3.10.3 Image Picker

We used this library to connect to the device camera and take the violation picture. The library integrates code for both IOS and Android, and with just a simple function call it return the image taken by the user with the camera.

3.10.4 Geolocator

Geolocator is a library which allows us to retrieve the current user location. It handles all the permissions and it return the longitude and latitude of the user device.

4 Source code structure



```
SAFESTREETS
  lib
    app
      data
        datasources
        models
        repositories
      domain
        repositories
        usecases
        view_model
      presentation
        bloc
        screen
        views
        widgets
    core
      enums
      error
      network
      router
      serializers
      services
      usecase
      utils
    injection_container.dart
    main.dart
  test
```

Figure 1: List of all the folder contained inside the application project.

4.1 Data

The Data folder represents the outermost layer of the application architecture. Its goal is to interact with the external data sources and handle the application model classes. The data from this layer will be then converted and utilized by the other layers of the application.

4.1.1 Data sources

This folder contains the classes responsible for the concrete interaction with the external sources. Each of those classes define an abstract interface which exposes the methods that will be called from the repository classes. Each file then contains a concrete implementation of the interface.
Most classes have several dependencies from third-party libraries and plugins. Those dependencies are injected inside each class from the *injection_container.dart* file, and they are initialized when requested. The methods inside the classes are responsible collecting the requested data asynchronously, convert it and return it for the repository classes.

Following the list of all the file contained in the folder:

- camera_data_source.dart
- location_data_source.dart
- notification_local_data_source.dart
- notification_remote_data_source.dart
- profile_local_data_source.dart
- profile_remote_data_source.dart
- registration_remote_data_source.dart
- report_remote_data_source.dart
- request_remote_data_source.dart

4.1.2 Models

This folder contains all the model classes utilized inside the application. Those classes define some model object which contains the data required by the application to function.
All the class have a serialization and deserialization method defined in order to be converted easily in different format throughout the application.

Following the list of all the file contained in the folder:

- area.dart
- image.dart
- license_plate.dart
- location.dart
- registration.dart
- report.dart
- violation.dart

4.1.3 Repositories implementations

This folder contains the concrete implementation of the interfaces defined in the repository inside the Domain folder.
The methods inside those classes are utilized by the usecases classes to provide the functionalities of the application to the client.
Those classes have a dependency on the data sources classes in order to retrieve the required data for the application functions. Those classes are also created from the *injection_container.dart* file.

Following the list of all the file contained in the folder:

- location_repository_impl.dart
- notification_repository_impl.dart
- profile_repository_impl.dart
- recognizer_repository_impl.dart
- registration_repository_impl.dart
- report_repository_impl.dart
- request_repository_impl.dart

4.2 Domain

The domain layer contains the interfaces and the usecases of the concrete functionalities available in the application. Those classes define the methods which will provide the implementation of the features of the system.

4.2.1 Repositories interfaces

This folder contains the interfaces signature of the usecases exposed by the application to the user. Those methods are then concretely implemented in the repository inside the data folder.
Each once of those methods is mapped to a specific usecase class in the usecase folder.

Following the list of all the file contained in the folder:

- location_repository.dart
- notification_repository.dart
- profile_repository.dart
- recognizer_repository.dart
- registration_repository.dart
- report_repository.dart
- request_repository.dart

4.2.2 Use cases

This folder contains the concrete usecases which are exposes to the client.
Each one of those classes maps to a method inside the repository folder. By using those classes we aim to separate the actual functionality implementation from the client. The front end of the application just needs to call those methods and collect the data required, without knowing were the data comes from or how it is computed.

Following the list of all the file contained in the folder:

- get_area_request.dart
- get_area_violation.dart
- get_camera_image.dart
- get_current_location.dart
- get_license_plate.dart
- get_my_report.dart
- get_subscription_area.dart
- get_user_id.dart
- login.dart
- logout.dart
- post_registration.dart
- recognize_license_plate.dart
- save_user_notification.dart
- subscribe_area_notification.dart
- unsubscribe_area_notification.dart

4.2.3 View models

4.3 Presentation

The presentation folder contains the front end side of the application. Specifically, it is responsible for the implementation of the actual UI to present to the client, and it also manages the state and business logic of the application.

4.3.1 Bloc

This folder contains all the bloc classes of the application. Those classes are used to separate the business logic of the application from the UI.
Each bloc class is defined by three files:

- file_bloc.dart
- file_state.dart
- file_event.dart

The event class defines the signature for every possible user interaction and the data that is required to perform that specific action.
The state class defines all the possible states in which the application can be.
The bloc class binds together the event and state classes. It listens to all the event and defines the action to perform for each of those. Its main purpose is to compute the data requested from the usecases classes from which it depend, and yield a series of state to display to the user.

Following the list of all the file contained in the folder:

- notification bloc
- report bloc
- request bloc
- user bloc

4.3.2 Screens

This classes contains the actual UI which will be displayed to the user. The end goal is to completely decouple those classes from any business logic concern. They request the concrete data by calling a series of events on the bloc, and they rebuild themselves based on the current state of the application and the data that each state contains.

Following the list of all the file contained in the folder:

- home_screen.dart
- loading_screen.dart
- login_screen.dart
- report_screen.dart
- settings_screen.dart

4.3.3 Views

4.3.4 Widgets

The widget folder contains all the UI components which are utilized in different screens throughout the application. The purpose is to define common UI pieces all in one place and to utilize them in multiple sections of the system. In this way, the application UI will always have a consistent look and it will become easy to update and expand starting from each individual components.

Following the list of all the file contained in the folder:

- loading_indicator.dart
- text_input_field.dart

4.4 Core

The core folder contains all the helper classes and methods utilized in different sections of the application. Those methods are common and consistent inside the whole system and generally they do not depend on any external data source.
In the next section we discuss some of the main functions inside the core folder.

4.4.1 Services

This folder contains the implementation of all the third-party packages and libraries which are required inside the application. Furthermore, it also contains the validation classes, which are used to verify the correctness of the input data from the user.

Following the list of all the file contained in the folder:

- recognizer_service.dart
- registration_service.dart
- report_validation_service.dart
- storage_service.dart
- validator.dart

4.4.2 Enums

Enums are special classes that defines human-readable labels for common values utilize inside the application. Those labels are used to simplify the code readability, in order to make clear what state each attribute could be in.

Following the list of all the file contained in the folder:

- image_type.dart
- license_plate_source.dart
- safety.dart
- violation_type.dart

4.4.3 Utils

Those classes defines some general helper methods which are recurrent throughout the application.

Following the list of all the file contained in the folder:

- validator.dart

4.4.4 Network

This folder contains the class responsible for checking the availability of the internet connection. It only contains the *network_info.dart* file.

4.5 Test

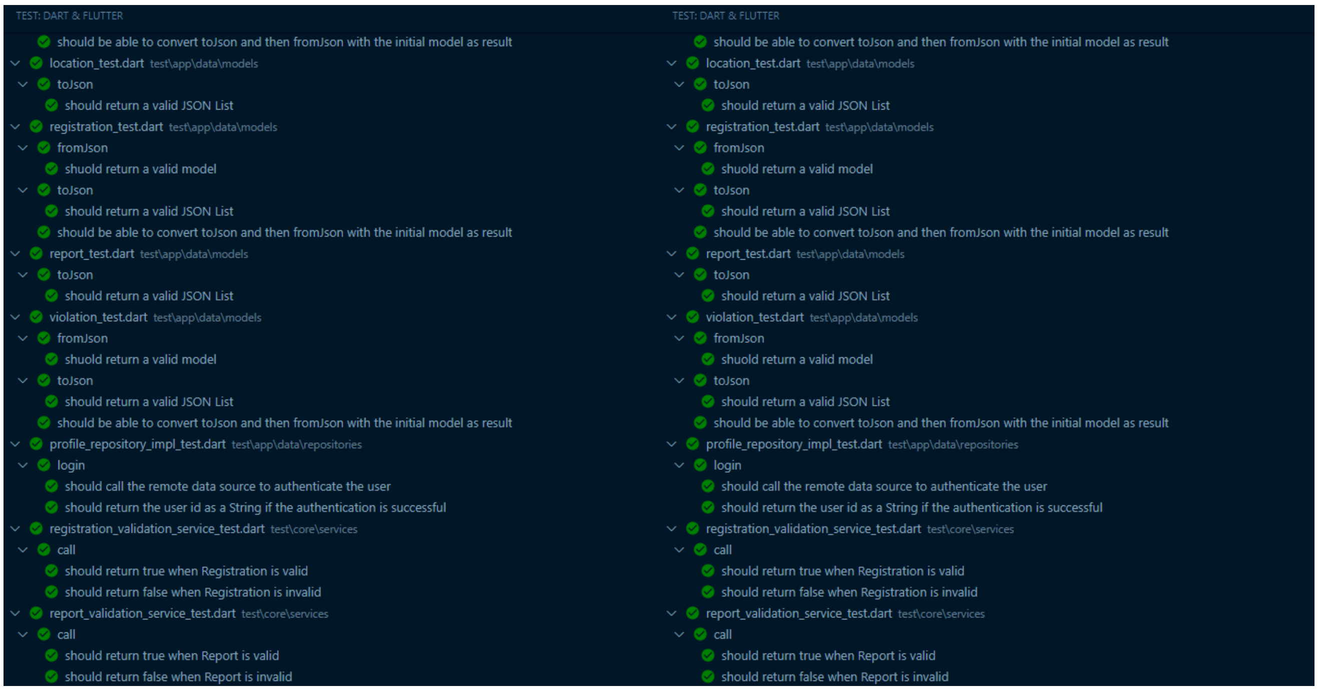
The test folder contains all the test classes for the application.

5 Testing

5.1 Strategy

5.2 Main test cases

A series of unit test has been created. Those tests make sure that the main methods inside the application work as expected.
Here below we report a picture of the main test cases implemented.



```
TEST: DART & FLUTTER
    should be able to convert toJson and then fromJson with the initial model as result
  location_test.dart test\app\data\models
    toJson
      should return a valid JSON List
  registration_test.dart test\app\data\models
    fromJson
      shuold return a valid model
    toJson
      should return a valid JSON List
    should be able to convert toJson and then fromJson with the initial model as result
  report_test.dart test\app\data\models
    toJson
      should return a valid JSON List
  violation_test.dart test\app\data\models
    fromJson
      shuold return a valid model
    toJson
      should return a valid JSON List
    should be able to convert toJson and then fromJson with the initial model as result
  profile_repository_impl_test.dart test\app\data\repositories
    login
      should call the remote data source to authenticate the user
      should return the user id as a String if the authentication is successful
  registration_validation_service_test.dart test\core\services
    call
      should return true when Registration is valid
      should return false when Registration is invalid
  report_validation_service_test.dart test\core\services
    call
      should return true when Report is valid
      should return false when Report is invalid
```

```
TEST: DART & FLUTTER
    should be able to convert toJson and then fromJson with the initial model as result
  location_test.dart test\app\data\models
    toJson
      should return a valid JSON List
  registration_test.dart test\app\data\models
    fromJson
      shuold return a valid model
    toJson
      should return a valid JSON List
    should be able to convert toJson and then fromJson with the initial model as result
  report_test.dart test\app\data\models
    toJson
      should return a valid JSON List
  violation_test.dart test\app\data\models
    fromJson
      shuold return a valid model
    toJson
      should return a valid JSON List
    should be able to convert toJson and then fromJson with the initial model as result
  profile_repository_impl_test.dart test\app\data\repositories
    login
      should call the remote data source to authenticate the user
      should return the user id as a String if the authentication is successful
  registration_validation_service_test.dart test\core\services
    call
      should return true when Registration is valid
      should return false when Registration is invalid
  report_validation_service_test.dart test\core\services
    call
      should return true when Report is valid
      should return false when Report is invalid
```

Figure 2: List of all the test cases implemented and passed.

6 Installation

6.1 Install the framework

All the instruction for installing the framework are deatiled on the offical documentation, for Windows, MacOS, and Linux, at this link https://flutter.dev/docs/get-started/install.
Basically, all you need is to clone or download the SDK of Flutter and all the Android or IOS dependencies required to run a mobile application.
We recommend using Visual Studio Code IDE, https://code.visualstudio.com/, and installing the Flutter and Dart plugins, https://dartcode.org/ and https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter.

6.2 Install the application

The Android application is available on the Google Play Store at this link:
https://play.google.com/store/apps/details?id=com.cmprogrammers.safestreets

No installation is therefore required, all you need to do is download the application on your device.

7 Appendix

7.1 Used tools

- Visual Studio Code: Editing LaTeX files, pdf generation, and Github repository management - https://code.visualstudio.com/

- LaTeX: For the production of technical and scientific documentation - https://www.latex-project.org/

- Android Studio - Android Emulator - https://developer.android.com/studio

7.2 Hours of work

7.2.1 Cappelletti Andrea

Date	Task	Hours	Start-End time
08/12/19	Folders and project setup	2h 0min	15:00 - 17:00
17/12/19	Setting Up iOS project	3h 0min	15:00 - 18:00
21/12/19	Firebase and Pods	3h 0min	12:00 - 15:00
30/12/2019	Request Repository Implementation	2h 0min	12:00 - 14:00
31/12/2019	Bloc setUp	3h 30min	12:30 - 16:00
02/12/2020	Request Bloc Implementation	2h 0min	9:00 - 11:00
08/12/2020	Request Complete Implementation, UI	7h 0min	10:00 - 18:00
09/12/2020	UI style and functionalities, application icon	4h 0min	10:00 - 14:00
09/12/2020	ITD document, scope, purpose, functionalities	3h 0min	14:00 - 17:00
Total		**30h 30min**	09/12/2019 to 12/01/2020

7.2.2 Maglione Sandro

Date	**Task**	**Hours**	**Start-End time**
08/12/19	Folders and project setup	2h 0min	15:00 - 17:00
11/12/19	Setting up firebase registration	1h 30min	17:45 - 19:15
12/12/19	Implemented postReport and getCameraImage	1h 30min	18:00 - 19:30
13/12/19	Implemented Location, Request, Notification repository	1h 45min	15:00 - 16:45
14/12/19	Implemented Profile and Recognizer repository, added Validation services	1h 15min	12:00 - 13:15
15/12/19	Full Android app configuration, keystore, sha-1, firebase project created	2h 15min	17:30 - 19:45
16/12/19	Added build value generation	2h 30min	15:00 - 17:30
17/12/19	Impl and working SignIn, SignUp, and SignOut	1h 30min	4:15 - 5:45
18/12/19	Impl MLKit, taking Picture from the Camera, and getting GPS Location	1h 30min	17:45 - 19:15
19/12/19	Impl Firebase Storage image saving and Firestore report saving	1h 30min	18:00 - 19:30
20/12/19	Created Firebase Cloud Firestore project	1h 00min	09:00 - 10:00
21/12/19	Notification, Settings screen and NotificationBloc	3h 00min	14:15 - 17:15
22/12/19	Completed notification service, code clean up and commenting	1h 15min	18:15 - 19:30
27/12/19	Impl validation function for Report and Registration objects	1h 00min	18:15 - 19:15
30/12/19	Setup ITD file structure	30min	12:45 - 13:15
30/12/19	Completed framework section of the ITD document	1h 15min	18:30 - 19:45
01/01/20	Completed source code structure section of the ITD document	1h 15min	18:15 - 19:30
03/01/20	Completed installation section of ITD document. Impl tests services, models, and datasources	1h 45min	15:00 - 16:45
04/01/20	Impl tests location service and profile repository	1h 00min	14:30 - 15:30
08/01/20	Impl authority verification and report license plate and type insertion	1h 30min	14:00 - 15:30
09/01/20	Completed all request and exported app	3h 00min	08:00 - 11:00
10/01/20	Bug fixing, new icon, and publish new release v1.0.1+2	1h 15min	14:15 - 15:30
12/01/20	Minor bug fixing, release v1.0.2+3	1h 15min	11:15 - 12:30
Total		**36h 15min**	09/12/2019 to 12/01/2020