SafeStreets project
Manuel Pedrozo, Tomás Perez Molina

POLITECNICO
MILANO 1863

Implementation and Testing Deliverable

Source code: https://github.com/lethanity/PedrozoPerez/tree/master/implementation

Installation: https://github.com/lethanity/PedrozoPerez/tree/master/DeliveryFolder/implementation

Deliverable:	ITD
Title:	Implementation and Testing Deliverable
Authors:	Manuel Pedrozo, Tomás Perez Molina
Version:	1.0
Date:	January 12, 2020
Download page:	https://github.com/lethanity/PedrozoPerez
Copyright:	Copyright © 2019, Manuel Pedrozo, Tomás Perez Molina – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose and scope

This is the Implementation and Test Deliverable document, its purpose is to explain the implementation and testing process for the SafeStreets system based on the RASD and DD documents. In the following section the topics to be covered are:

- An overview of the proposed architecture

- Implemented requirements and their motivation

- Adopted development frameworks, along with their advantages and disadvantages.

- The structure of the source code

- Testing

- Installation instructions

1.2 Definitions, Acronyms, Abbreviations

1.2.1 Definitions

- Traffic violation: An action performed by a driver of a vehicle which is against the local traffic regulations.

- Report: Information submitted by a user to notify the system and, by extension, authorities of a traffic violation.

- Authority: A local agency whose purpose is, as indicated by the current law, to enforce traffic rules. For example: the police.

- Ticketing system: A government database containing information about issued traffic tickets.

- License plate registry: A government database connecting a license plate with the car registered to it and information about it such as the make, model and color.

- Widget: Flutter UI component.

1.2.2 Acronyms

- API: Application Programming Interface.

- HTTP: HyperText Transfer Protocol.

- HTTPS: HyperText Transfer Protocol Secure.

- UI: User Interface.

- RASD: Requirements Analysis and Specification Document.

- DD: Design Document.

1.2.3 Abbreviations

- [FRn]: n-th functional requirement

- [SRn]: n-th security requirement

2 Implemented requirements

In this section, the requirements specified in chapter 3 of the RASD document are categorized according to their implementation state.

2.1 Included requirements

The following requirements were fully implemented:

[FR1] - The user is able to take pictures from the mobile application to add to a report.

[FR2] - The user is able to fill out a form providing information about a traffic violation, consisting of:

- Type of violation
- License plate
- At least one photo of the scene

[FR3] - The application can determine date, time and location based on information provided by the device when the pictures are taken and adds this metadata to the report.

[FR4] - The user can submit a full report to the system

[FR5] - The user is provided a map where the location of reports are indicated by markers.

[FR6] - The user can filter the reports shown on the map by date and type of violation.

[FR9] - A secured API is provided to obtain and filter report information. The following filters are possible:

- Date and time
- Location
- Type of violation

[SR1] - Only photos taken through the mobile application are submitted, the user cannot upload a picture previously saved on their device.

[SR2] - Special API keys are generated and provided to authorities

[FR10] - A secured API accessible only to authorities, provides the following information about reports:

- Photos
- License plate

[FR13] - The API includes the system's degree of confidence in the report to the provided report information after a query.

[FR14] - Report information provided by the API can be filtered by their degree of confidence.

[FR15] - Users can review photos with a license plate recognition lower than 80% to adjust the confidence in the recognition.

[FR18] - The user can register by inputting his full name and email, and choosing a username and password.

[FR19] - The user has access to view his full account information.

[FR20] - The user can edit his email and full name.

[FR21] - An API key is provided to the user via their profile screen in the mobile application

[FR22] - The user can login to the mobile application by providing their email and password.

2.2 Partially included requirements

Location is shown, and searched, by coordinates only. This is because, in order to get an approximate address, it is required to integrate with the Google Places API, which is paid per request.

[FR7] - The user can search for a specific location on the map by inputting coordinates or an address.

[FR8] - The user can select a report on the map and obtain information about it. This information includes:

- Report ID: unique reference to each report
- Type of violation
- Date and time
- Location: gps coordinates, approximate street name and number

Since there was no integration with an external license plate registry, the degree of confidence is based on the license plate recognition only.

[FR12] - A degree of confidence is assigned to each report, where a high confidence report will have:

- License plate recognition of 80%+ confidence.
- Car recognition of 80%+ confidence.
- The license plate and car pair are found in the license plate registry.

If any of these criteria are not met the report is considered low confidence.

2.3 Excluded requirements

As previously mentioned, currently there is no integration with an external license plate registry.

[FR11] - Detected cars and license plates are cross referenced with the license plate registry to ensure the license plate belongs to the car.

The following requirements belong to the advanced functionality, which was not required for the implementation:

[SR3] - End-to-end encryption is provided for the submission of reports.

[FR16] - Information obtained from the ticketing system is used to determine if a report contributed to the issuing of a traffic ticket.

[FR17] - Whether the report contributed to a traffic ticket is included in the information provided by the mobile application and the API.

3   Adopted development frameworks

Since the development time for the application is short, for the most part the team opted for known technologies as to reduce the risk of missing the timeline by encountering unexpected obstacles. This also allowed the team to focus on unknown areas, like the integration with external APIs.

3.1   Mobile

As stated in section 5.1.1 of the Design Document, the mobile application is developed in the Flutter framework, which uses the Dart programming language. The advantage of Flutter is that it is a complete toolset, allowing for development without the necessity of third party packages for core functionality, which can bring incompatibilities. In spite of that, some packages more specific to the application were used to speed up development (such as the map, geolocation and camera). Also, being completely written in Dart, a typed language, makes implementation more intuitive and easier to follow compared to web technologies, which juggle HTML, CSS and Javascript at the same time. The main disadvantage of Flutter is, at the same time, the Dart language which is very young (2013) and unknown to most developers currently. However, its syntax is familiar and easy to pickup.

3.2   Back-end server

As stated in section 5.1.2 of the Design Document, the backend is implemented in Kotlin utilizing the Spring boot framework. Kotlin was chosen over Java because its more concise, safer and allows for faster development for those familiar with it. Spring is one of the most popular frameworks for the development of web applications, based on the JVM. It allows for fast development and easy integration with documentation tools that help speed up the work. Because of its maturity and popularity, there is a great amount of documentation on the framework itself. There are a few alternatives, like Play and Grails, which offer similar functionalities, but Spring has been proven to be the most performant of the three.

4 Structure of the source code

4.1 Mobile

The mobile application resides in the /mobile directory

- /src
    - ◇ /android Android application specific files.
    - ◇ /fonts extra fonts added to the application.
    - ◇ /images asset images.
    - ◇ /ios iOS application specific files.
    - ◇ /lib root of the source files.
        - /data data structures used in multiple parts of the app (structures need in only one screen are kept private to that file).
        - /screens widgets for each screen in the app.
        - /services services used by the app, most of them used to talk to the application server.
        - /util utility functions.
        - /widgets other widgets included in the screens.
    - ◇ /mocks mock images used for testing.
    - ◇ /test_driver E2E test files.
- /test contains the source code files for testing.
    - ◇ /kotlin/se2/SafeStreets/back contains unit test classes.
    - ◇ /resources contains files used for testing, like images.

4.2 Back-end server

The source code is organized following the Gradle standard and Spring boot naming conventions. This resulted in deviations from some names specified in the DD. Furthermore, due to the reduced scale of the implementation in comparison to what was previously specified (see Implemented Requirements 2.1), certain components suffered some changes, specifically:

Controllers handle requests from the system users, these are:

- AuthController
- UserController
- ViolationReportController
- ReviewController
- ImageController
- ApiKeyController

Services handle the system logic:

- AuthService: Acts as the previously defined IdentificationManager and acts as the authentication provider of Spring Security.
- UserService: Was merged with UserManager.
- ViolationReportService: Was merged with ViolationReportManager.

- ReviewService: Was merged with ReviewManager.

The AuthorizationGuard component was removed since its responsibilities are now handled by Spring security.

The application resides in the /back directory, where Gradle configuration files can be found.

- /src

  ◇ /main contains the source code files.

  ◇ /java/com/openalpr/jni contains the OpenALPR java files.

  ◇ /kotlin/se2/SafeStreets/back contains the kotlin files, organized according to Spring boot framework.

    - /controller contains Spring controller classes.
    - /model contains application models and DTOs.
    - /repository contains Spring repository classes.
    - /security contains Spring security configuration and filters.
    - /service contains Spring service classes.

  ◇ /resources contains resources used by the application, like Spring properties files.

- /test contains the source code files for testing.

  ◇ /kotlin/se2/SafeStreets/back contains unit test classes.

  ◇ /resources contains files used for testing, like images.

5   Testing

Due to time constraint, the team decided against unit testing every single component of the system. Instead, integration tests at different levels were performed.

5.1   Integration testing

The following tests were performed in the backend server by testing individual controllers, which provide the entry point of the exposed API. The tests were implemented making use of the Spring boot test module. Here we list some of them.

Test ID	INT-001-SIGN_UP
Components	UserController, UserService, UserRepository, Mongodb
Input	Perform a "sign up" request.
Expected output	"No content" response, user saved in the database.
Description	POST request to route "/user/sign-up".
Data used	
	email: test@test.com password: password
Outcome	Success

Test ID	INT-002-GET_CURRENT_USER
Components	UserController, UserService, UserRepository, Mongodb
Input	Perform a get current user request.
Expected output	"Ok" response with user data matching the user performing the request.
Description	GET request to route "/user/me" with a registered user in the security context.
Outcome	Success

Test ID	INT-002-GET_CURRENT_USER
Components	UserController, UserService, UserRepository, Mongodb
Input	Perform a "get current user" request.
Expected output	"Ok" response with user data matching the user performing the request.
Description	GET request to route "/user/me" with a registered user in the security context.
Outcome	Success

Test ID	INT-003-SIGN_IN
Components	AuthController, AuthService, UserRepository, TokenProvider, Mongodb
Input	Perform a "sign in" request.
Expected output	"Ok" response with the generated JSON token.
Description	POST request to route "/auth".
Data used	email: user1@mail.com password: password1
Outcome	Success

Test ID	INT-004-SIGN_IN-WRONG_PASSWORD
Components	AuthController, AuthService, UserRepository, TokenProvider, Mongodb
Input	Perform a "sign in" request with incorrect password.
Expected output	"Unauthorized" response.
Description	POST request to route "/auth".
Data used	email: user1@mail.com password: wrongPassword
Outcome	Success

Test ID	INT-005-SUBMIT_REPORT
Components	ViolationReportController, ViolationReportService, Violation-Repository, Mongodb
Input	Perform a "submit report" request.
Expected output	"Created" response.
Description	POST request to route "/violation".
Data used	license plate: "EX2631" description: bad parking datetime: <current time> violation type: Parking location: [9.225708, 45.479183]
Outcome	Success

Test ID	INT-006-GET_REPORT_IN_BOUNDS
Components	ViolationReportController, ViolationReportService, Violation-Repository, Mongodb
Input	Perform a "get reports in bounds" request.
Expected output	"Ok" response with a list of violation reports matching the provided filters.
Description	POST request to route "/violation/query/bounds".
Data used	south west: [8.0, 43.0] north east: [10.0, 45.5] from: <3 hours ago> to: <current time> violation types: [Parking] violation report status: [low-confidence, high-confidence, review]
Outcome	Success

Test ID	INT-007-GET_API_KEY
Components	ApiKeyController, ApiKeyService, UserService, ApiKeyService, UserRepository, ApiKeyRepository, Mongodb
Input	Perform a "get current user api key" request.
Expected output	"Ok" response with the generated api key.
Description	GET request to route "/api-key/me" with a registered user in the security context.
Outcome	Success

5.2   System testing

Complete system tests were performed in an E2E fashion, utilizing flutter_driver to control the application running in an Android emulator, which connects to the application server. The test cases considered are the use cases defined in the RASD, following the conventional event flow, with no exceptions. This ensures the application functions as expected while maintaining a fast test suite, as E2E tests are expensive and slow compared to unit tests.

Test ID	SYS-001-SIGN_UP
Input	Perform a sign up through the application.
Expected output	Successful sign in with the new user created, redirecting to home screen.
Description	While on the sign in screen, press the "sign up" button which redirects to the sign up screen where the form is filled and submitted. Once redirected to the sign in screen, fill the form with the new users email and password and submit it. Check that the application redirects to the home screen.
Data used	email: email@mail.com password: password name: name surname: surname username: username
Outcome	Success

Test ID	SYS-002-SIGN_IN
Input	Perform a sign in through the application.
Expected output	Successful sign in with an already created user, redirecting to the home screen.
Description	While on the sign in screen, fill the form with email and password of an existing user in the database. Then check that the application redirects to the home screen.
Data used	email: user0@mail.com password: password1
Outcome	Success

Test ID	SYS-003-SHOW_PROFILE
Input	Enter profile screen.
Expected output	Correct data displayed in the profile screen.
Description	While on the home screen, navigate to the profile screen. Check that the information displayed matches the one stored in the database.
Outcome	Success

Test ID	SYS-004-OBTAIN_API_KEY
Input	Tap "get API key".
Expected output	API key shown on screen.
Description	While on the profile screen, tap the "get API key" button. Wait for an API key to be displayed.
Data used	
	aaaa
	aaaa
	aaaa
	aaaa
	aaaa
Outcome	Partial success (see note).
Note	Due to a limitation of the framework, the widget used to show text that is also selectable (so the user can copy the key) cannot be read with the test driver, so the key cannot be compared to make sure its the same as a predefined value. Only the presence of a key is checked.

Test ID	SYS-005-EDIT_PROFILE
Input	Edit the user profile.
Expected output	New profile information shown on profile page.
Description	While on the profile screen, tap the "edit" button which redirects to the edit profile screen. There, fill the form with new information and submit it. This redirects to the profile screen, where the profile information is checked.
Data used	
	name: newName
	surname: newSurname
	username: newUsername
	email: newMail@mail.com
Outcome	Success

Test ID	SYS-006-SIGN_OUT
Input	Perform a sign out.
Expected output	Application redirects to the login screen.
Description	While on the profile screen, press the "sign out" button, then check that the application redirects to the sign out screen.
Outcome	Success

Test ID	SYS-007-SUBMIT_REVIEW
Input	Perform a report review.
Expected output	Review submitted successfully.
Description	While on the home screen with a user with a review request, tap the review notification button. An alert is shown with a license plate field where a license plate is inputted and submitted. Once the alert closes, check that a message is displayed indicating a successful review.
Data used	Logged in as: email: user0@mail.com password: password1 license plate: EX215GC
Outcome	Success

Test ID	SYS-008-SUBMIT_REPORT
Input	Submit a violation report.
Expected output	Report submitted successfully.
Description	While on the home screen press the "Submit a report" card, which redirects to the violation report screen. Once there, fill the form, press the take a photo button, which automatically loads a mocked car picture and press confirm. On the alert to confirm the license plate photo press ok. Check that a message is displayed indicating a successful submission and the application redirects to the home screen.
Data used	violation type: Parking license plate: DX034PS description: testing photo:
Outcome	Success

Test ID	SYS-009-SHOW_REPORTS_MAP
Input	Go to map screen.
Expected output	Report marker shown on the map.
Description	Repeat the procedure for Test SYS-008-SUBMIT_REPORT in order to have a report on the current location. On the home screen, tap the "Reports map" card. Once redirected to the map screen, check that there is a report marker visible on the map.
Outcome	Partial success (see note)
Note	Due to limitations of the framework, the filter functionality could not be tested.

17

6 Installation

6.1 Back-end server and database

The recommended way of installing the application is through Docker and the provided docker-compose file. This will set up both the backend server and the mongodb database.

Pre-requisites:
- Have Docker installed in your computer, can be downloaded from: https://www.docker.com/
- Have an internet connection for installation.

Installation steps:
- Make sure Docker is running.
- Open the terminal/command line at the directory containing the provided docker-compose.yml file.
- Type and run the following command: docker-compose up.
- Wait for Docker to finish the setup.
- To check if everything is ok, open a browser and navigate to http://localhost:8080/auth/ping. The text "pong" should be displayed on your screen.

6.2 Mobile

To install the mobile application:
Pre-requisites:
- Android phone.
- Application server running.

Installation steps:
- Transfer the provided APK file to the phone.
- On the phone, navigate to the directory containing the APK and tap on the file.
- Some warnings may pop up, as the application is not registered, ignore them and continue with the installation.
- The app is installed.
- Open the app and do a tap and hold on the grey "SafeStreets" title.
- An alert will pop up to enter the URL to connect to the application server.
    - Note: instead of localhost, the URL will have the ip of the machine running the server, for example http://192.168.99.100:8080.
- Press connect.
- A tick mark should appear next to the URL field, indicating a good connection, and the "SafeStreets" title will change to blue.
- You can either sign up as a new user or use a preexisting user, for example:
    - email: ewan@mail.com
    - password: password

7 Effort Spent

7.1 Manuel Pedrozo

Task	Hours
Introduction	0.5
Implemented requirements	1.5
Adopted development frameworks	1
Structure of the source code	1.5
Testing	1.5
Installation	0.5

Table 1: Effort spent by Manuel Pedrozo

7.2 Tomás Perez Molina

Task	Hours
Introduction	0.5
Implemented requirements	0.5
Adopted development frameworks	1
Structure of the source code	0.5
Testing	1
Installation	0.5

Table 2: Effort spent by Tomás Perez Molina

8   References

- "SafeStreets Mandatory Project Assignment"

- "Implementation Assignment"

- Requirement Analysis and Specification Document V1.2 (RASD3.pdf)

- Design Document V1.1 (DD2.pdf)

20