Software engineering project

SafeStreets

ITD : Implementation and Testing Document.

Authors:
Mattia Micheloni,
Francesco Montanaro,
Amedeo Pachera.

Professor:
M.Rossi.

Version 1.0
11/1/20

Contents

Chapter 1

Introduction

1.1 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. The document is structured in the following way

- $Chapter_2$: It presents the goals and requirements implemented in the prototype.
- $Chapter_3$: It presents the frameworks adopted for the development and testing process.
- $Chapter_4$: It presents and explains the structure of the source code.
- $Chapter_5$: It gives the information on the testing process.
- $Chapter_6$: It gives instructions on how to install and run all the application's components.

1.2 Definitions, acronyms, abbreviations

This section gives some acronyms in 1.2.1, definitions in 1.2.2 and abbreviation in 1.2.3 which will be used in the document, in order to explain some concept and help the general understanding.

1.2.1 Acronyms

- API: Application Programming Interface
- GPS: Global Positioning System
- UI: User interface
- RASD: Requirement Analysis and Specification Document.
- DD: Design Document.
- UT: Unit Testing.

1.2.2 Definitions

- User: The customer of the application who can upload reports concerning traffic violations and have access to the system's statistics services.
- Authority: The costumer of the application who can access the information on reports made by the users in order to validate or modify them, and eventually generate tickets. They can also upload on the system information about accidents that occur on the territory and have access to the system's statistics services.
- Traffic Violations: The set of all the infractions that occur on the territory which are recognized by the system.
- Metadata: The information extracted by the system from a report such as license plate, geographical position, data, time and traffic violation type.

1.2.3 Abbreviations

- G_n : nth Goal.
- R_n : nth Requirement.

1.3 Revision history

- Version 1.0 : 11/1/2020

1.4 Reference documents

- RASD-SafeStreets: “Mandatory Project Assignment AY 2019/2020”.
- DD-SafeStreets: “Mandatory Project Assignment AY 2019/2020”.

Chapter 2

Implemented goals and requirements

In this chapter all the requirements and main functions that are implemented in the prototype are presented. They are listed below:

- G_1 : Users can report traffic violations to authorities by sending pictures to the system.
- G_2 : The System has to make easier the process of reporting violations by extracting metadata from the pictures.
- G_3 : Authorities can access sensitive data for specific purposes, such as traffic tickets generation (so their consistency and reliability must be granted).
- G_4 : Both the users and authorities can access to statistics (based on accidents and violations) built by the system.
- G_5 : System can suggest intervention to authorities, based on shared data obtained by crossed information between police and SafeStreets.
- R_1 : The system allows to take pictures only within the software itself.
- R_2 : The report can be made only if the internet connection is available.
- R_3 : The system imposes the user to include a picture of the license plate of the vehicle that committed the violation.
- R_4 : The system gets date and time from its internal clock when the picture is taken.

- R_5 : The system obtains the violation's position from the device's GPS.
- R_6 : The system converts the geographic coordinates obtained in the corresponding address.
- R_7 : The system automatically detects the vehicle's license plate number.
- R_8 : Access to the violation reports is only permitted to authorities.
- R_9 : The system stores violations data.
- R_{10} : Authorities must log in to the system.
- R_{11} : The system checks that a fake GPS service is not active o the user's device.
- R_{12} : Authorities can modify, discard or validate the violation reports present on the system.
- R_{13} : Authorities can insert data concerning accidents that occur on the territory.
- R_{14} : The system builds statistics based on validated data.
- R_{15} : The system produces statistics concerning possible unsafe areas built on the accidents information.
- R_{16} : Authorities and users can set the temporal granularity and the category of the statistics that they want to consult.
- R_{17} : The authorities can consult the interventions suggested by the system.
- R_{18} : The system has an associated suggestion for each type of violation.
- R_{19} : The system allows to filter the suggestions on the map by category.

Chapter 3

Development framework

With regard to the Component View presented in the DD, various frameworks and programming languages have been adopted for development, depending on the component and the platform on which it runs on.

3.1 Frameworks and programming languages

3.1.1 Application server

Node JS has been used for the application server development, which is an environment that helps in the execution of JavaScript code on server-side. It has been chosen because it offers an easy scalability of the system by the integration of additional nodes. Moreover, it contains all the tools to deal with asynchronous programming which has been used in order to manage in a correct way the communication between the various system components and APIs.

3.1.2 Website client

The SafeStreets website client is built with HTML, CSS and JavaScript, using a prebuilt template with Bootstrap and jQuery-min. The choice to use pure JavaScript is due to the confident in using this programming language but with the disadvantages to haven't a development framework which helps to control all the classes and states (such as React.js).

3.1.3 Mobile App client

SafeStreets mobile client was developed for the Android Platform. Android Studio IDE has been used with the Java programming language. It has been chosen because it is the main development environment for Android applications, so it provides all the tools needed to build a complete application such as emulators, debugging tools and Junit for testing.

3.2 Middleware software

Firebase: It is a middleware that offers all the tools to build complete systems. In particular, it manages all the application's infrastructures (clients, server and database), and some of the main back-end functionalities such as database, authentication features, crash reporting and automatic scalability of the system.
Mocha: Testing framework well integrated with Node.js.
Google Cloud Platform: it is a suite of cloud computing services which allows to create, implement and scale applications, Websites and services on the same infrastructure used by Google. Its main feature is providing a place to build and run softwares, allowing to manage all the resources and external APIs.

3.3 External API

The external APIs used are:

- Google Maps API: Library from google to use maps and plot unsafe areas and markers. Link to reference: https://developers.google.com/maps/documentation.

- Google Geocoding API: Library from google to convert GPS location to address and viceversa. Link to reference: https://developers.google.com/maps/documentation/geocoding

- BarChart API: Library to draw charts on Android. Link to reference : com.github.PhilJay:MPAndroidChart:v2.2.4.

- Google Vision API: Library to use OCR in Android. SafeStreets doesn't use Tesseract OCR (as mentioned in the DD) directly, but uses google-vision API that includes the same algorithm of Tesseract optimized for Android development.

Chapter 4

Structure of Source Code

The code of SafeStreets is downloadable at this link:
https://github.com/pake97/MicheloniMontanaroPachera.
It is structured as follows:

- SafeStreets_Server/functions/
  - Index.js: this file contains the source code of all the server's functions implemented.
  - package.json: this files handles all the project's dependencies.
- SafeStreets_website: list of code implemented (the rest is from the template)
  - index.html
  - violation.html
  - register.html
  - mapStats.html
  - accidents.html
  - suggestion.html
  - js/chart.js
  - js/config.js (FIREBASE API SETTINGS)
  - js/suggestion_map.js
  - js/firebase.js
  - js/firebase2.js

  - js/login.js
  - js/uploadAccident.js
  - js/violation.js
  - js/violationVrapper.js
  - js/upload.js
  - js/SuggestionWrapper.js
  - js/control_map.js
  - css/main.css (only some modifications)
  - css/style.css

- SafeStreets_app/SafeStreets/app/src/main
  - AndroidManifest.xml
  - java/com/ap/android/SafreStreets/AccidentWrapper.java
  - java/com/ap/android/SafreStreets/MainActivity.java
  - java/com/ap/android/SafreStreets/MapActivity.java
  - java/com/ap/android/SafreStreets/LicensePlateActivity.java
  - java/com/ap/android/SafreStreets/LicensePlateRecognizerActivity.java
  - java/com/ap/android/SafreStreets/StatisticManager.java
  - java/com/ap/android/SafreStreets/StatisticManagerActivity.java
  - java/com/ap/android/SafreStreets/StartReportActivity.java
  - java/com/ap/android/SafreStreets/ViolationWrapper.java
  - java/com/ap/android/SafreStreets/Violation.java
  - java/com/ap/android/SafreStreets/ReportViolationManagerActivity.java
  - res/ (layout design elements)

Chapter 5

Testing

5.1 UT application

AndroidStudio IDE has Junit included and it's possible to test the application inside the framework.
The testing strategy was to first text each activity verifying if all the UI elements were loaded; then the testing proceeded as the pipeline of the activities, checking if all the data forwarded from a class to another were correct. In addition a control on the computing function of the StatisticManager was made, in order to avoid values that cause errors (such as division by 0).

5.2 UT server

To test Server function the strategy follows the Firebase documentation:
https://firebase.google.com/docs/functions/unit-testing
The testing uses an external framework called Mocha;
https://mochajs.org
All the functions are run with fake requests and responses and every exceptions are caught and log.

5.3 UT website

In the website , Mocha framework is needed to test all the javascript files and functions. The focus of the tests was on computing functions instead of layout functions(functions that manage html elements), creating a new html file with "div" element linked to mocha in order to insert and get the results of the tests. Finally each tests were run with node.js calling mocha functions.

Chapter 6

Installation instructions

6.1 Website

To run and test the website no installation is needed, just launch the website pages on localhost. To login in the website the credentials are :
email: milano@gmail.com,
password: milano12345.

6.2 Android App

To run and test the Android app AndroidStudo IDE is needed, the app is built using an emulator of a Nexus 5X API 28 with android 9.0 Oreo, the minimum SDK version is 26.
To use google services enable the google-play-services SDK on the menu as follows: $Tools \rightarrow SDKManager \rightarrow AndroidSDK \rightarrow SDKtools$ and enable the checkbox named "Google repository". To run the application with your own SDK just clean and rebuild the project in the menu "Build".

6.3 Server

To develop Server functions install firebase-tools with npm as written in https://firebase.google.com/docs/functions/get-started.
To run each functions just copy the https request links on firebase, run it on browser and check the result on the firebase console.