Politecnico di Milano

SAFESTREETS - DD

POLITECNICO
MILANO 1863

Samuele Meta - Stiven Metaj

Supervisor: Matteo Rossi

Department of Computer Science and Engineering

December 6, 2019

Contents

1 Introduction

1.1 Purpose

The following Design Document (DD) is aimed to provide an overview of the *SafeStreets* application, explaining how to satisfy the project requirements declared in the RASD and stating the successive refinements made together with the Stakeholders according to their needs. The document is mainly intended to be used by developers teams as a guidance in the development process, by testing teams to write automated testing and to avoid structural degradation of the system in case of maintenance or future extension. Indeed, its purpose is to provide a functional description of the main architectural components, their interfaces and their interactions, along with the design patterns and algorithms to be implemented.

1.2 Scope

As explained in the RASD document, *SafeStreets* is a crowd-sourced mobile application that allows Citizens to notify Authorities about traffic violations, with particular emphasis on parking contraventions. Citizens are able to manage their reports, visualizing the whole history and modifying or removing them. On the other hand, Authorities can access a complete overview of the incoming reports and choose if to accept or reject them.

1.3 Definitions, Acronyms, Abbreviations

In this section follow definitions, acronyms (including their meaning) and abbreviations used in this document.

1.3.1 Definitions

- *Client*: a desktop computer or workstation that is capable of obtaining information and applications from a Server.
- *Server*: a computer or computer program which manages access to a centralized resource or service in a network.
- *Firewall*: a part of a computer system or network which is designed to block unauthorized access while permitting outward communication.
- *Port*: an endpoint of communication in an operating system.
- *Design Pattern*: reusable software solution to a commonly occurring problem within a given context of software design.

1.3.2 Acronyms

DBMS	DataBase Management System
HTTPS	Hyper Text Transfer Protocol over SSL
API	Application Programming Interface
REST	REpresentational State Transfer
MVC	Model View Controller
OS	Operative System
UI	User Interface
UX	User Experience
URL	Uniform Resource Locator
RASD	Requirements Analysis and Specification Document
SPA	Single Page Application
ERD	Entity-Relationship Diagram
SSL	Secure Sockets Layer
DDoS	Distributed Denial of Service

Table 1.1: *Acronyms*

1.3.3 Abbreviations

- [**R.n**]: n-th Requirement in the RASD document

1.4 Revision History

[TBD]

1.5 Reference Documents

- Specifications document "SafeStreets. Mandatory project assignment"
- SafeStreets RASD Document
- IEEE Standard 1016-2009: IEEE Standard on Software Design Descriptions
- UML documentation: `https://www.uml-diagrams.org`

1.6 Document Structure

The rest of the document is organized as follows:

- **Architectural Design**: details the System's architecture by defining the main components and the relationships between them as well as specifying the hardware needed for the System deployment. It will also be focused on design choices and architectural styles, patterns and paradigms.
- **User Interface Design**: provides further details on the UI defined in the RASD document through the use of UX modeling.
- **Requirements Traceability**: shows the relations between the requirements from the RASD and the design choices of the DD and how they are satisfied by the latter.
- **Implementation, Integration and Test Plan**: provides a roadmapping of the implementation and integration process of all components and explains how the integration will be tested.
- **Effort Spent**: describes how the work has been split between the members of the team and how long did the DD take to be completed.

2 Architectural Design

2.1 Overview

2.2 Component View

Web Client, Mobile Client

This component represents the Client machines that access to the API of the Business Logic of the System. They do not have any notable functionality to be outlined, due to the fact that they are implemented as thin clients as explained below. The Web Clients, accessing trough a browser, need the Presentation component in order to display the web pages of the application. This layer provides to the it the structure of the User Interface without accessing data and application logic. On the other hand, the Mobile Client embeds the Network Manager, the Presentation Component and the Data Manager.

User Manager

This Manager includes all the operations that affect the user-related data. It exposes methods to change account credentials and preferencies. Furthermore, it manages the data stored in the DB through the interaction with the interface of the Model Interface.

Authentication Manager

This Manager handles all the methods related to the authentication task and access to the System. Specifically, it deals with both the User registration and login, making use of the Model Interface to interact with the DB. It handles credentials verification and constraints, guaranteeing the creation of consistent accounts. Moreover, it also uses the Mail Service Interface to communicate

with the Mailing Service and send registration emails.

Report Manager

This Manager allows the Citizen to create reports and send them to the platform. It interacts with the Model Interface in order to store them permanently and to retrieve the list of the previous reports. On the other hand, it allows to the Authorities to have a complete overview of the incoming reports and gives the possibility to accept or refuse them.

Statistics Manager

This Manager handles the requests of Guests, Citizens and Authorities to obtain insights about data through statistics. It is responsible to aggregate the information, eventually interacting with the Municipality Interface, and return it to the User. It also invokes the Authentication Manager to check the role of the User, in order to return only the allowed statistics.

External Services Interfaces

Some of the components in the System are also dedicated to communicating with external services through specific interfaces. These interactions are bilateral and essential to guarantee the application's functionalities. These components are both on the Client side and on the Server side. In particular the interfaces needed by the System are:

- *Mail Service Interface*: is responsible of the interaction with the mail service. It is used to send an email confirmation to the User, at the request of the Authentication Manager, during the registration phase.
- *Map Service Interface*: is responsible of providing, at the request of the Statistics Manager, a visual representation of raw data whenever these have a spatial dimension. It can also be invoked by the Report Manager in the eventuality that the Citizen is not able to remember the name of the street and needs a map as an helper.
- *Push Notification Interface*: interacts with the Push Notification Service and is responsible to notify to the Citizen messages of interest, such as the outcome of the profile verification or the acceptance of a report.

- *Municipality Interface*: is responsible of providing, at the request of the Statistics Manager, additional data in possession of the Municipality, in order to join them with local statistics and provide to the User a complete overview.

Model Interface

The Model Interface includes two sub-components. The Data component provides the set of Classes corresponding to the tables contained in the Database. The Storage Interface provides the methods for querying the Database.

Data Base

This component represents the DBMS, which provides the interfaces to retrieve and store data. In the data base, for each user, credentials and application data are safely and securely stored.

2.3 Deployment View

2.4 Runtime View

2.5 Component Interfaces

2.5.1 REST API

The application will use the REST paradigm (explained better later on). So we focus on the different REST API endpoints that the application uses, describing in details the following attributes for each of the requests:

- The operation name of the request
- The precise endpoint URL
- The method type (GET, POST, PUT or DELETE)
- The URL parameters (in case of GET or DELETE)
- The data parameters (in case of POST or PUT)
- The success and the error responses. Let's specify right now the possible responses in out application APIs:

  - Code: 200 OK
  - Code: 400 BAD REQUEST
  - Code: 401 UNAUTHORIZED
  - Code: 403 FORBIDDEN
  - Code: 404 NOT FOUND
  - Code: 422 UNPROCESSABLE ENTRY

- A briefly more precise description of what the request existence permits (what operation the *User* is allowed to do thanks to this?)

In particular we have different API interfaces for each *Manager* in the application. The managers names and their functionalities follow:

- **Authentication Manager**: The Authentication Manager provides what is needed for the registration and the login to the application; furthermore we want the possibility to verify an account, so we expect the Authentication Manager to manage this.

- **User Manager**: This is the manager related to the possibilities given to the *User* in order to obtain and update information such as preferences, settings or credentials.

- **Violation Manager**: The Violation Manager provides all the necessary operations regarding violations reported by *Citizens*. For example, it has to permit to add or delete (if not already accepted) the reports or to accept them by *Authorities*.

- **Statistics Manager**: This last manager is related to the feature of statistics visualization and process.

For reading purposes for each following page we will have only one endpoint (two at maximum if the sizes of the tables are such that they can stay without problems in only one page).

Authentication Manager - Registration of a *Citizen*

Endpoint	*/auth/register/citizen
Method	POST
URL Params	
Data Params	fiscalCode: [alphanumeric] name: [text] surname: [text] mail: [alphanumeric] username: [alphanumeric] password: [alphanumeric] birthDate: [Date]
Success Response	Code: 200 Content: {message: "Registration successful, check your email to complete the creation of your account"}
Error Response	Code: 403 Content: {error: "Already registered"} Code: 422 Content: {error: "Registration Data not correct"}
Notes	Allows a Client to request the registration of a new *Citizen*

Authentication Manager - Registration of an *Authority*

Endpoint	*/auth/register/authority
Method	POST
URL Params	
Data Params	authorityID: [alphanumeric] name: [text] surname: [text] mail: [alphanumeric] username: [alphanumeric] password: [alphanumeric]
Success Response	Code: 200 Content: {message: "Registration successful, check your email to complete the creation of your account"}
Error Response	Code: 403 Content: {error: "Already registered"} Code: 422 Content: {error: "Registration Data not correct"}
Notes	Allows a Client to request the registration of a new *Authority*

Authentication Manager - Login of a *User*

Endpoint	*/auth/login
Method	POST
URL Params	
Data Params	username: [alphanumeric] password: [alphanumeric]

Success Response	Code: 200
	Content: {
	userType: [text]
	accessToken: [alphanumeric]
	}

Error Response	Code: 401 Content: {error: "Wrong mail or password"} Code: 422 Content: {error: "Login Data not correct"}
Notes	Allows a Client to obtain an authentication token and to login as *Citizen* or as *Authority*

Authentication Manager - Activation of an account

Endpoint	*/auth/activate
Method	GET
URL Params	activationCode: [alphanumeric]
Data Params	
Success Response	Code: 200 Content: {message: "Account activated"}
Error Response	Code: 401 Content: {error: "Invalid token"} Code: 403 Content : {error: "Account already activated"} Code: 404 Content: {error: "Incorrect activation Code"}
Notes	Allows a Client to activate the account

User Manager - Get a *Citizen* account information

Endpoint	*/citizen/{id}
Method	GET
URL Params	accessToken: [alphanumeric]
Data Params	

Success Response	Code: 200
	Content: {
	fiscalCode: [alphanumeric]
	name: [text]
	surname: [text]
	email: [alphanumeric]
	birthDate: [Date]
	username: [alphanumeric]
	password: [alphanumeric]
	applicationSettings: [List <Setting>]
	}

Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 403 Content: {error: "User ID does not match with the provided authentication token"} Code: 404 Content: {error: "User not found"}
Notes	Allows a Client to obtain the information related to the *Citizen* associated to the provided Token

User Manager - Modification of a *Citizen* account information

Endpoint	*/citizen/{id}
Method	PUT
URL Params	
Data Params	accessToken: [alphanumeric] oldPassword (optional): [alphanumeric] newPassword (optional): [alphanumeric] name (optional): [text] surname (optional): [text] birthDate (optional): [Date] email (optional): [alphanumeric] applicationSettings (optional): [List <Setting>]
Success Response	Code: 200 Content: {message: "Settings correctly updated"}
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 403 Content: {error: "User ID does not match with the provided authentication token"} Code: 404 Content: {error: "User not found"} Code: 422 Content: {error: "New settings data not correct"}
Notes	Allows a Client to modify the information or the settings related to the *Citizen* associated to the provided Token

User Manager - Get an *Authority* account information

Endpoint	*/authority/{id}
Method	GET
URL Params	accessToken: [alphanumeric]
Data Params	

Success Response	Code: 200
	Content: {
	authorityID: [alphanumeric]
	name: [text]
	surname: [text]
	email: [alphanumeric]
	username: [alphanumeric]
	password: [alphanumeric]
	applicationSettings: [List <Setting>]
	}

Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 403 Content: {error: "User ID does not match with the provided authentication token"} Code: 404 Content: {error: "User not found"}
Notes	Allows a Client to obtain the information related to the *Authority* associated to the provided Token

User Manager - Modification of an *Authority* account information

Endpoint	*/authority/{id}
Method	PUT
URL Params	
Data Params	accessToken: [alphanumeric] oldPassword (optional): [alphanumeric] newPassword (optional): [alphanumeric] name (optional): [text] surname (optional): [text] email (optional): [alphanumeric] applicationSettings (optional): [List <Setting>]
Success Response	Code: 200 Content: {message: "Settings correctly updated"}
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 403 Content: {error: "User ID does not match with the provided authentication token"} Code: 404 Content: {error: "User not found"} Code: 422 Content: {error: "New settings data not correct"}
Notes	Allows a Client to modify the information or the settings related to the *Authority* associated to the provided Token

Violation Manager - Showing violations reports by a *Citizen*

Endpoint	*/citizen/home
Method	GET
URL Params	accessToken: [alphanumeric]
Data Params	

Success Response	Code: 200
	Content: {
	violations: [
	violationID: [alphanumeric]
	reportDate: [Date]
	reportTime: [Time]
	plate: [text]
	latitude: [float]
	longitute: [float]
	streetName: [text]
	violationType: [int]
	photos: [List <Photo>]
	]
	}

Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"}
Notes	Allows a Client to show violations reports made by the *Citizen* associated to the provided Token (at the start of application in the home page)

Violation Manager - Filtering violations reports by a *Citizen*

Endpoint	*/citizen/home
Method	GET
URL Params	accessToken: [alphanumeric] endDate: [Date] startDate (optional): [Date] distance (optional): [int] violationTypes (optional): [List <int>] plate (optional): [text]
Data Params	

Success Response	Code: 200
	Content: {
	violations: [
	violationID: [alphanumeric]
	reportDate: [Date]
	reportTime: [Time]
	plate: [text]
	latitude: [float]
	longitute: [float]
	streetName: [text]
	violationType: [int]
	photos: [List <Photo>]
	]
	}

Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 404 Content: {error: "Too many filters, no violation reports found"}
Notes	Allows a Client to show violations reports made by the *Citizen* associated to the provided Token (at the start of application in the home page)

Violation Manager - Showing a *Citizen*'s violation report (by the *Citizen*)

Endpoint	*/citizen/report/{id}
Method	GET
URL Params	accessToken: [alphanumeric]
Data Params	

Success Response	Code: 200
	Content: {
	violationID: [alphanumeric]
	reportDate: [Date]
	reportTime: [Time]
	plate: [text]
	latitude: [float]
	longitute: [float]
	streetName: [text]
	violationType: [int]
	photos: [List <Photo>]
	}

Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 403 Content {error: "Report ID does not match with the provided authentication token"} Code: 404 Content: {error: "Wrong request, report not found"}
Notes	Allows a Client to show to the *Citizen* associated to the provided Token the violation report with id={id}

Violation Manager - Adding a violation report by a *Citizen*

Endpoint	*/citizen/report
Method	POST
URL Params	
Data Params	accessToken: [alphanumeric] reportDate: [Date] reportTime: [Time] plate: [text] latitude: [float] longitute: [float] streetName: [text] violationType: [int] photos: [List <Photo>]
Success Response	Code: 200 Content: {message: "Violation report correctly created"}
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 422 Content: {error: "Violation report data not correct"}
Notes	Allows a Client to add a violation report related to the *Citizen* associated to the provided Token

Violation Manager - Deleting a violation report

Endpoint	*/citizen/report/{id}
Method	DELETE
URL Params	accessToken: [alphanumeric]
Data Params	
Success Response	Code: 200 Content: {message: "Violation report correctly deleted"}
Error Response	Code: 401 Content: {error: "Login not correct"} Code: 403 Content {error: "Report already accepted"} Code: 403 Content {error: "Report ID does not match with the provided authentication token"} Code: 404 Content: {error: "Wrong request, report not found"}
Notes	Allows a Client to delete a violation reported by the *Citizen* associated to the provided Token

Violation Manager - Showing to an *Authority* a *Citizen*'s report

Endpoint	*/authority/report/{id}
Method	GET
URL Params	accessToken: [alphanumeric]
Data Params	

Success Response	Code: 200
	Content: {
	violationID: [alphanumeric]
	reportDate: [Date]
	reportTime: [Time]
	plate: [text]
	latitude: [float]
	longitute: [float]
	streetName: [text]
	violationType: [int]
	photos: [List <Photo>]
	citizenFiscalCode: [alphanumeric]
	citizenName: [text]
	citizenSurname: [text]
	citizenDateOfBirth: [Date]
	}

Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 403 Content {error: "Report already accepted"} Code: 404 Content: {error: "Wrong request, report not found"}
Notes	Allows a Client to show a violation report made by a *Citizen* to an *Authority* associated to the provided Token

Violation Manager - Accepting a violation report (by an *Authority*)

Endpoint	*/authority/report/{id}
Method	POST
URL Params	
Data Params	accessToken: [alphanumeric]
Success Response	Code: 200 Content: {message: "Violation report correctly accepted"}
Error Response	Code: 400 Content: {error: "Wrong request"} Code: 401 Content: {error: "Login not correct"} Code: 403 Content {error: "Report already accepted"} Code: 404 Content: {error: "Wrong request, report not found"}
Notes	Allows a Client to let the *Authority* associated to the provided Token to accept a violation report

Statistics Manager - Showing area safeness statistics

Endpoint	*/stats/area
Method	GET
URL Params	latitude: [float] latitude: [float] radius (optional): [int]
Data Params	

Success Response	Code: 200
	Content: {
	areas: [
	latitude: [float]
	longitude: [float]
	radius: [int]
	safetyLevel: [int]
	]
	}

Error Response	Code: 400 Content: {error: "Wrong request"} Code: 404 Content: {error: "Wrong request, statistics not found"}
Notes	Allows a Client to showing area statistics related to latitude and longitude parameters

Statistics Manager - Showing intervention suggestions

Endpoint	*/stats/interventions
Method	GET
URL Params	latitude: [float] latitude: [float] radius (optional): [int]
Data Params	

Success Response	Code: 200
	Content: {
	interventions: [
	latitude: [float]
	longitude: [float]
	intervention: [text]
	]
	}

Error Response	Code: 400 Content: {error: "Wrong request"} Code: 404 Content: {error: "Wrong request, statistics not found"}
Notes	Allows a Client to showing suggestions of interventions inside the area related to latitude and longitude parameters

2.6 Selected Architectural Styles and Patterns

2.6.1 Multi-tier Architecture

2.6.2 Use of RESTful guidelines

2.7 Other Design Decisions

2.7.1 Thin Client

2.7.2 MVC

2.7.3 Firewalls

3 User Interface Design

3.1 User eXperience Diagrams

4 Requirements Traceability

5 Implementation, Integration and Test Plan

5.1 Overview

5.2 Component Integration

5.3 Something on Testing?

6 Effort Spent

The effort spent from each member of the team to build the DD can be summarized with the following tables: