Software Engineering 2 - Mandatory Project

Immagine che contiene serviziodatavola

Descrizione generata automaticamenteAY 2019/2020

DD

Version 1.0 – 9/12/19

Maserati Alessandro - 944593



**Table of Contents**

1. **Introduction**
   1. Purpose
   2. Scope
   3. Definitions, acronyms and abbreviations
      1. Definitions
      2. Acronyms
      3. Abbreviations
   4. Revision history
   5. Reference documents
   6. Document structure
2. **Architectural design** 
   1. Overview
   2. Component view
      1. Components functionalities
   3. Deployment view
   4. Runtime view
   5. Component interfaces
   6. Selected architectural styles and patterns
   7. Other design decisions
3. **Requirements traceability**
4. **Implementation, integration and test plan**
5. **Effort spent**

**1. Introduction**

1.1 Purpose

The purpose of this document is to provide details about the design of SafeStreets application. This will be done illustrating the chosen architecture, giving the description of all the components that form the system, representing the related run-time processes and showing which patterns are used to develop the system. Furthermore, the requirements illustrated in the RASD are mapped on the relative architecture’s components. Lastly, the implementation, the integration and the test plan are identified and described in detail.

1.2 Scope

………………

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

* *Authority*: public institution related to street safety (e.g. municipality, local police).
* *End user*: people (unrelated with authorities) using Safe Streets application with the aim of report traffic violations and know the streets where the most violations occur.
* *My Reports*: section visible on End User mobile app in which all the reports made by that end user are shown, underlying the current state of each report (it can be Unchecked, Confirmed or Rejected depending on the decision of the authority).

1.3.2 Acronyms

* RASD: Requirements Analysis and Specification Document
* DD: Design Document
* API: Application Programming Interface
* DB: Data Base

1.3.3 Abbreviations

* [Rn]: n-th requirement

1.4 Revision history

* 09/12/19: Version 1.0
  + First release

1.5 Reference documents

* Specification Document “*SafeStreets Mandatory Project Assignment*”
* 1016.1-1993: IEEE International Standard

1.6 Document structure

This DD is composed by 5 sections:

**Section 1** is an introduction containing the purpose, the scope, the revision history and the structure of the document and terminology conventions.

**Section 2** represents the core of the document. It shows the architectural design with a detailed description of components, relations and interactions between them. This description is made exploiting component, deployment and sequence diagrams. Then, the patterns used in the design are listed.

In **Section 3**, an explanation on how the requirements, identified in the RASD, map on the design elements, illustarted in Section 2, is given.

**Section 4** contains the identification of the order in which the implementation, the integration of subcomponents of the system and the test of them is planned.

In this document, a section that provides an overview on how the user interfaces will look like is not provided because that topic is presented in detail in Section 3.1 of the RASD.

**2. Architectural design**

2.1 Overview

SafeStreets application is thought to be developed on a three-tier architecture. In figure 1, the three logical levels (presentation, application and data) are represented with their dedicated hardware components.

With regards to presentation layer, both end users and authorities exploits the same mobile application, it will be the Router application subcomponent that chooses the right functionalities to offer them. For what concerns the application layer, the application server communicates with the DB in two ways: synchronously when it must retrieve information and asynchronously when it has to store information.

2.2 Component view

Figure 2 figure represents the component diagram of the application. Because of readability reasons, the ports that represent external interfaces are not shown, the application server (the core of the application) is the only subsystem described and the interactions between application’s components and the DB are omitted as they are illustrated in section 2.2.1 Components functionalities.

2.2.1 Components functionalities

* **Router**: it interacts directly with user’s mobile applications and it manages all the function calls from them. Depending on the result of the signup process, it forces the mobile app to show to the end user the right functionalities. It has also the task of acting as intermediary between the various application subcomponents.
* **SignUpManager**: it contains the procedures that allow a user to register to the system (both end users and authorities). It communicates with the DB to store registrations data and to perform changes on them (changes on credentials).
* **LoginManager**: it contains the procedure that allow users to log into the system. It interacts with the DB in order to check the inserted credentials.
* **ReportManager**: it manages all the processes that involves traffic violation reports. It is divided in two subcomponents: the **ReportNotificationManager**, that controls the function of reporting a new traffic violation report from an end user and the visualization of *MyReports* section, and the **ReportCheckManager**, that queries the DB for unchecked violation reports in order to show them to authorities. It interacts directly with the DB in order to extract the right report instances and with the NotificationManager. Notice that this component, after the submission of the report, runs the algorithm able to retrieve the license plates visible from the picture entered by the end user.
* **DataVisualizationManager**: it contains the applicative logic that handles users’ requests of visualize data mined from the DB. It has two subcomponents: the **ViolationMapManager**, that, interacting directly with the MapService, “builds” the TrafficViolationMap asked by both type of users, and the **VehiclesManager** that queries the DB for the list of most reported vehicles. Obviously, these components manage also the filtered requests from users.
* **NotificationManager**: it contains the applicative logic for pushing notifications to users’ mobile app. It comes into play in two different situations: when an end user reports a new traffic violation report to the system, pushing a notification to all authorities and when an authority checks a traffic violation report, sending the response to the end user that notified it.

It must be noticed that the only components that exploits the MapService API are the ViolationMapManager, that create the map using the information stored in the DB, and the Mobile App, in order to visualize the map in a correct way.

2.3 Deployment view

…

2.4 Runtime view

In this section, some of the most relevant use cases (from Section 3.2 Of the RASD) are described using sequence diagrams, to highlight the interactions between the involved design components and sub-components.

Immagine che contiene screenshot

Descrizione generata automaticamenteRegistration and Login

Data entered by the End User in the registration process are not listed for the sake of simplicity. The messages coming from the End User are to be understood as simple input data (clicking on App buttons). If the information entered contains some mistakes or the credentials are incorrect (after the DB has checked that) an error message is reported to the End User (as described in the relative Use Case in the RASD). Notice that the same process is also valid for authorities, the only difference is that they must provide their ID Number, instead of the Fiscal Code.

Reporting a traffic violation

Also in this Sequence Diagram, the parameters that a Traffic Violation Report requires are not listed, they consists of one or more pictures (taken calling the device’s camera) and a textual description of the violation. It’s important to notice that, after the storage of the data provided by the End User, the ReportNotificationManager “calls” the NotificationManager in order to send a notification to all registered authorities, that can now check the report.

Traffic Violation Checking Process

The Authority Mobile APP retrieves the list of Unchecked reports from the DB, so the Authority can choose which is the report to check. After the storage of the response, that report is no more visible in the list. Furthermore, a notification, containing the response, is sent to the End User who made that report.

Visualizing the Traffic Violation Map and focusing on a specific street

When a user wants to visualize the Traffic Violation Map, the ViolationMapManager retrieves all the necessary information from the DB. Interacting with the MapService and the MobileAPP, ViolationMapManager can show to the user the requested map. When the user enters the name of the street that he want to visualize, the call to ViolationMapManager is not necessary, because the information is already available from the App. If the street’s name doesn’t exist, the MobileAPP notifies the error. Notice that if the user wants to exploit some filters, another call to the ViolationMapManager and to the MapService is necessary, in order to restrict the information used to build the new map.

The “OK” messages are simple acknowledgement sent to the requesting component, in order to detect communication errors, when this detection is necessary.

2.5 Component interfaces

Figure … shows the interfaces offered by the Application Server components.

**ManageSignUp** contains methods exploited by the Router in order to successfully register a new user, the necessary information to identify a generic user is passed as parameter.

**ManageLogin** offers the method that allows a user to log into the system. Also this interface is exploited by the Router, that “forwards” the credentials to the DB, in order to check their correctness. The method returns a Boolean value, depending on that a message of success or failure is shown to the user.

The **ManageReports** interface offers all the methods that handles Traffic Violation Reports. It communicates with the **ManageNotification** interface in order to send notification to the customer MobileAPP when a new unchecked report is stored into the DB or when an authority takes a decision about one of it.

The **ManageVehicles** and **ManageViolationMap** interfaces are those allowing the visualization of information by a user. They offer methods called by the Router, that, after consulting the DB, respectively return the notified vehicles list and the TrafficViolationMap requested by the user.

Lastly, the **ManageAPP** interface consists of methods called by the User MobileAPP, when a request from a user came.

2.6 Selected architectural styles and patterns

2.7 Other design decisions

**3. Requirements traceability**

In this section, the requirements identified in the RASD are mapped on the components (identified in section 2) that have the task of fulfil them.

* [R1] – End Users must be able to register to the system, providing personal information (and identified by Fiscal Code).
  + **SignUpManager**: it contains the logical procedures necessary to register to the system an end user.
* [R2] – Registered End Users must be able to login, using their credentials.
  + **LoginManager**: it contains the logic function necessary to log an end user into the system, checking the information stored into the DB.
* [R3] – End Users must be able to take pictures of traffic violations, opening their device’s camera from the app.
  + **ReportNotificationManager**: it contains the functionality of opening the device’s camera in order to take a picture of the violation and, after the submission of the report, to store the information into the DB.
* [R4] – End Users must be able to specify a textual description of the traffic violations.
  + **ReportNotificationManager**
* [R5] – The system must offer the possibility of being informed of the “Rules for a well-formed traffic violation report” to the user.
  + **ReportNotificationManager**: if the end user wants to know those kinds of rules, it provides them a textual list of the rules.
* [R6] – The system is able to read the notified vehicles’ license plate from the reported pictures.
  + **ReportNotificationManager**: it runs the algorithm able to read a license plate from the picture taken by end users.
* [R7] – Authorities must be able to register to the system, providing personal information (identified by ID Number).
  + **SignUpManager**: it contains the logical procedures necessary to register to the system an authority.
* [R8] – Registered Authorities must be able to login, using their credentials.
  + **LoginManager**: it contains the logic functions necessary to log an authority into the system, checking the information stored into the DB.
* [R9] – The system must be able to show a map reporting the number of violations that have occurred in every street. This must be made highlighting the streets with different colours.
  + **ViolationMapManager**: exploiting the MapService API, it builds the TrafficViolationMap requested by users, using data stored into the DB.
* [R10] – Users are allowed to filter the data, by date and type of violation, which are used to “build” the map.
  + **ViolationMapManager**: exploiting the MapService API, it builds the TrafficViolationMap requested by users, using data stored into the DB.
* [R11] – Users are allowed to ask for information about a specific street and the system must show the corresponding map.
  + **ViolationMapManager**: exploiting the MapService API, it searches the requested street and shows the map, highlighting that street.
* [R12] – The system must be able to show (only to authorities) the list of vehicles that have been reported one or more time in a traffic violation report.
  + **VehiclesManager**: retrieving information from the DB, it shows the list of reported vehicles to authority.
* [R13] – Authorities are allowed to filter the list of notified vehicles by date, street and type of violation.
  + **VehiclesManager**: retrieving information from the DB, it shows the list of requested reported vehicles to authority.
* [R14] – Authorities must be able to know everything (pictures, license plate, date, street, textual description) about the reports made by end users.
  + **ReportCheckManager**: retrieving information from the DB, it shows to authority all the information about reports.
* [R15] – Authorities are allowed to consult and investigate the list of unchecked traffic violation reports notified by end users.
  + **ReportCheckManager**: retrieving information from the DB, it shows to authority the list of actually Unchecked reports.
* [R16] – Authorities are allowed to take decisions about any traffic violation report, after consulting it.
  + **ReportCheckManager**: it contains the logic to change the status of a report, after the decision of authorities. It works interacting with the NotificationManager.
* [R17] - The system must notify that response to the related user.
  + **NotificationManager**: it handles the notifications to send, after the checking process made by an authority, to the correct end user.
* [R18] – The possibility to visualise all personal previous reports, with the corresponding authority check decision, must be offered to End Users.
  + **ReportNotification**: retrieving information from the DB, it shows all the reports made by a specific end user.

Notice that the Router has never been mentioned not because it’s not involved in fulfil any requirement, but because its role is to route to the right component all the messages coming from the client side and also some messages between some subcomponents.

**4. Implementation, integration and test plan**

**5. Effort spent**