POLITECNICO
MILANO 1863

SafeStreets

Marco Premi (941388)
marco.premi@mail.polimi.it

Fabrizio Siciliano (939895)
fabrizio.siciliano@mail.polimi.it

Giuseppe Taddeo (928360)
giuseppe.taddeo@mail.polimi.it

Computer Science and Engineering

2019/2020

Software engineering 2

DD

Design Document

Version 1.0 - [Data da inserire]

Reference professor:

Matteo Giovanni Rossi
matteo.rossi@polimi.it

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to give more technical details than the RASD about SafeStreets system. The RASD presented a more abstract and general view of the system and of the functions is supposed to execute. Indeed, this document presents more details about the design, run-time processes, deployment and algorithm. It also provideds more information about implementation, integration and testing with a testing plan.
In particular, the document presents the following topics:

- Overview of the high-level architecture
- The main components, their interfaces and deployment
- The run-time behavior
- The desing patterns
- Requirements on architecture components
- Implementation plan
- Integration plan
- Testing plan

1.2 Scope

Here it's presented a review of the application scope, made referring to what has been stated in RASD document.
With SafeStreets users can notify the authorities when traffic violations occur, and in particular parking violations. Both user and authorities must register to the application and agree that SafeStreets stores the information provided, completing it with suitable meta-data. The whole system, because it tracks users information, must respect the standards defined for processing of sensitive information such as GDPR if it is used in Europe. The user sends the type

of the violation to the municipality and direct proofs of it (like a photograph). The system runs an algorithm to read the license plate and also asks the user to directly insert the license for a better recognition. Obviously, other information are required, like the name of the street when the violation has occurred, which can be retrieved from user's direct input or from the geographical position of the violation (using Google Maps API). Furthermore, the system, by cross referencing data from third party services, automatically can highlight the streets with the highest frequency of violations or the vehicles that commit the most violations. SafeStreets crosses information about the accidents that occur on the territory of the municipality with his own data to identify potentially unsafe areas and suggest possible interventions. Because municipality could generates traffic tickets from the information about violations SafeStreets should guarantee that information is never altered (if a manipulations occurs, the application should discard the information). Such features are made possible trough the use of one mobile application with two different UIs which are determined by the kind of customer that logs in (user or PO). The collected information are sent to a back-end and they all of those can be accessed by municipality employees in order to execute different actions (emit ticket, analyze unsafe areas, etc...).

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **User:** it is identified as a civilian customer of the product. It will be the main source for the SafeStreets initiative to obtain information about traffic violations and therefore to be successful;
- **Third parties:** those kind of organization/company that could provide services useful to SafeStreets;
- **Customer:** it defines both authorithy users (police officers or municipality employees) and civilians;
- **Authority user:** all of those customers who have a responsibility role in regard of the streets' safety and the SafeStreets initiative. Example of these category are: police officers, municipal employees, director and basically anyone in charge and able to issue fines and deal with road violations;
- **Ghiro:** image manipulation detection software, used by authorithy users in order to detect any image manipulation and assess the veracity of the hard evidence connected to the traffic ticket

1.3.2 Acronyms

- **UI:** User Interface

- **GDPR:** General Data Protection Regulation
- **API:** Application Programming Interface
- **GPS:** Global Positioning System
- **PO:** Police Officer
- **ME:** Municipality Employee

1.3.3 Abbreviations

- **Gn:** nth goal;
- **Dn:** nth assumption;
- **Rn:** nth requirement;
- **ID:** identifier (Fiscal Code for Users, a municipality identifier for Authorithy Users)

1.4 Revision history

1.5 Reference Documents

1.6 Document Structure

Chapter 1 - Introduction

Chapter 1 is an introduction of the design document. It describes the purpose and the scope of the document and it highlights the differences with the RASD. It also shows some abbreviations, definitions and acronyms in order to provide a better understanding of the document to the reader.

Chapter 2 - Architectural design

Chapter 2 deals with the architectural design of the system and it's the core section of the document.
It provides an overview of the architecture and it contains the most relevant architecture views:

- Component view
- Deployment view
- Run-time view

It also shows the interaction of the interfaces and the selected architectural styles and patterns, with an explanations of each one of them.

Chapter 3 - User interface design

Chapter 3 specifies the user interface design and refers to the mock-ups already presented in the RASD. Furthermore it shows some UX diagrams to describe the interaction between the customer and the application.

Chapter 4 - Requirements traceability

Chapter 4 explains how the requirements defined in the RASD map to the design elements defined in this document.

Chapter 5 - Implementation, integration and test plan

Chapter 5 specifies the description and the order of implementation, integration and testing plan of the sub-components of the system.

Chapter 6 - Effort spent

Chapter 6 shows the effort spent by each member of the group working on this project.

Chapter 7 - References

Chapter 7 includes the reference documcuments.

2 | Architectural design

2.1 *Overview*: High-level components and their interaction

The whole software that will become the main core of the SafeStreets initiative will be developed as a distributed application, which means that the software will be executed (or run) on multiple devices within a network. It will have a three-layers logic and be divided as following:

- **P**: The ***presentation*** layer will handle all *incoming* (*and outcoming*) relations with the customers
- **A**: The ***application*** layer will work as a "man in the middle" between the **P**resentation layer and the **D**atabase layer and will hold all the needed logic for the software to correctly work;
- **D**: The ***database*** layer will be needed in order to store and manage all needed (and requested) information of the initiative;

Each and every one of the layers the architecture will be composed by a (group of) machines. By doing this, it is meant to provide, to each layer, its own dedicated hardware, for either scalability, failure handling and flexibility reasons. The following image shows the high-level architecture of the system without providing any detail of the components which will form the structure of the software itself, which will be tackled later in this document.

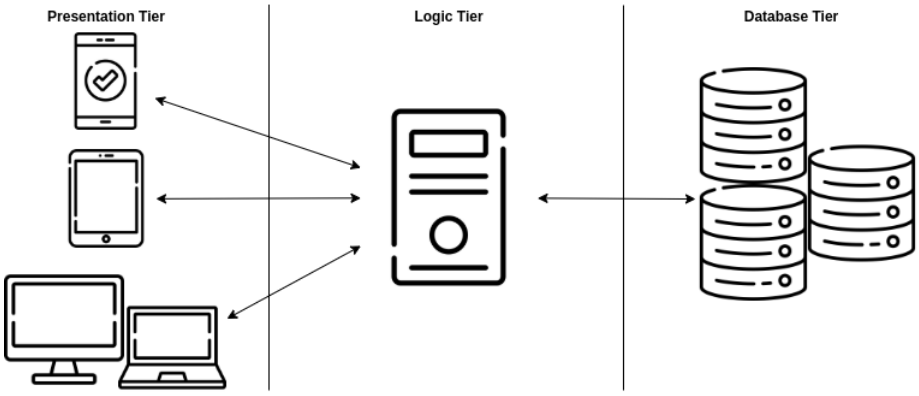

Figure 2.1: High-level architecture

2.2 Component view

- **Router**:it manages messages and function calls coming from other subsystems in order to pass the data to the correct element of the system. It eventually calls the correspondent method/function on it. Furthermore, the router is partitioned according to the type of the interacting components because of the different functionalities.
- **SignUpManager**: this component provides all the procedures to allow customers to register to SafeStreets. Obviously, this components has also to interact with the DBMS to store the registration data and to run a check about the chosen email, password and AuthoritiesID or fiscal code.
- **SignInManager**: it contains all the logic devoted to the authentication of the customers. It check the authentication parameters using the data stored on the DBMS.
- **SignalViolationManager**: it deals with the signalations of violations made by the users. This component has to verify that all the inserted data are correct and if not has to immediately ask the user to provide more information.
- **UnsafeAreasManager**: this component provides all the users the possibility to see the unsafe areas. It receives all the information from the DBMS.
- **ViolationsListManager**: with this component the user can see all the violations he/she has reported. The authority user can see all the violations reported in his/her area.
- **ViolationsCheckingManager**: this component provides the authority user

- **StatisticsManager**: this component provides the authority user the possibility to see all the statistic generated crossing the data on violations. It receives all the information from the DBMS.
- **CrossDataManager**: it crosses data from SafeStreets database and from the municipality database to generate the statistics and the unsafe areas.

2.3 Deployment view

The following image is a deployment diagram which represents the architecture of the system as distribution(deployment) of software artifacts to deployment targets(node). Artifacts represents elements obtained with a development process. Nodes can represent either hardware or software environments.

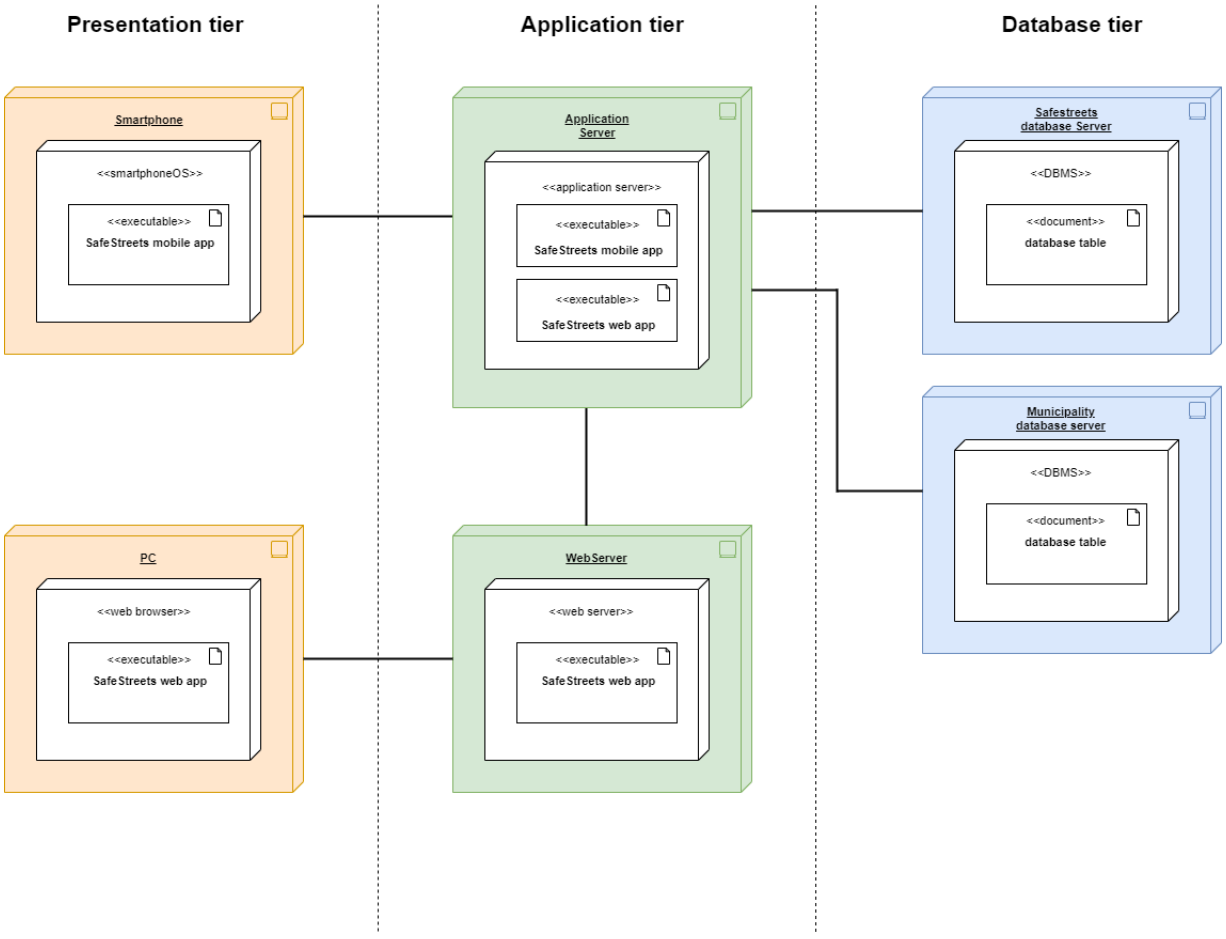

Figure 2.2: Deployment view

The three tiers contain:

- **Presentation tier**: in this tier the presentation logic is deployed. Users are provided with a mobile application on their mobile devices and authority users are provide with both a mobile application and a web application accessible from a common browser. The mobile application must be developed for most of the devices (both iOS and Android version). Both user and authority user ask to communicate to the application server in order to retrieve data, signal a violation, check violations or unsafe areas.
- **Application tier**: in this thier the application logic is deployed. The application server allows the mobile application and the web application to access data stored into the SafeStreets database. The application server also implements the business logic and handles the requests. The mobile application directly addresses the application server. The web server allow authority users to use SafeStreets services. If it can't provide some information it forwards the requests to the application server.
- **Database tier**: in this tier data access must be deployed. The application has to handle data both on SafeStreets database and on the municipality database for cross references.

2.4 Run-time view

2.4.1 Synchronization

2.4.2 Request data regarding a group of people

2.4.3 Request data regarding a particular user by providing his/her UUID

2.5 Component interface

2.6 Selected architectural styles and patterns

The architecture of SafeStreets is multilayered, composed of three tiers:

- **Presentation layer**: is used to present the data in a way that the user can understand. It enables the usage of the services offered to the user. The presentation layer of the the users is the smartphone and for authority users are both the smartphone and the browser.
- **Application layer**: is used to coordinate the application. It receives and computes the requests send by the presentation layer and it also interacts with the database.
- **Database layer**: stores the information provided by the application layer. The information is also passed back to the application tier for processing.

Three tier architecture is very useful because it allows to change or upgrade one of the three tiers without any problem and so it makes the system more flexible and reusable. Furthermore it makes the system safer because it separates the access to data from the other layers.

2.6.1 Design Patterns

Model View Controller (MVC)

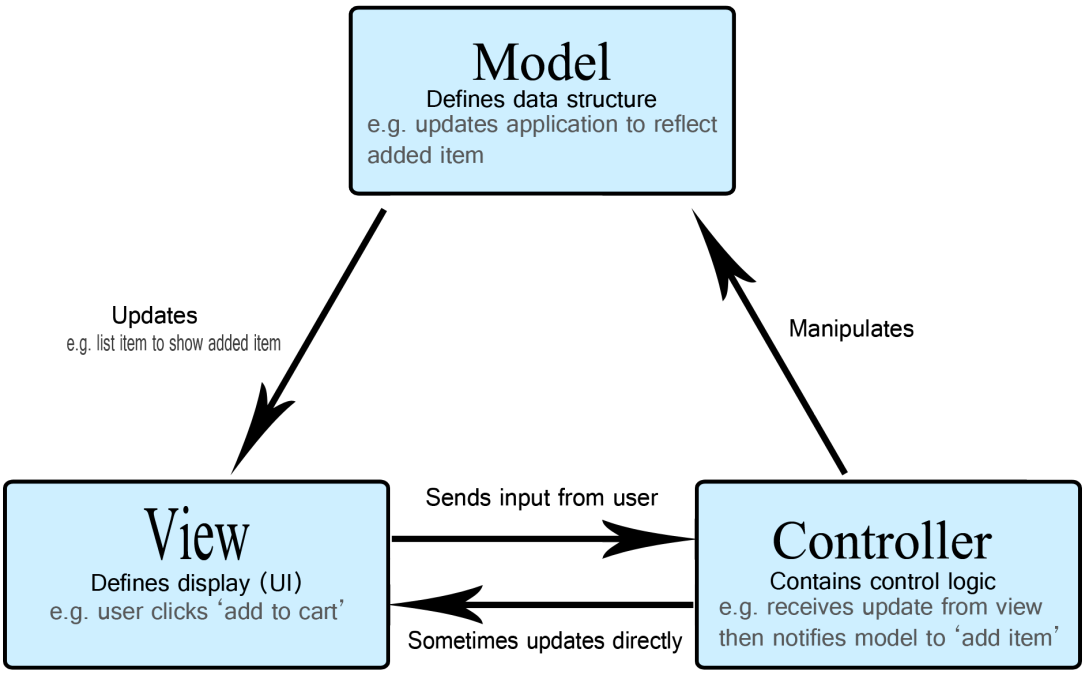

Figure 2.3: Model view controller

Model view controller is a very useful and quoted design pattern. MVC is used to separate the fundamental parts of the application and in particular it emphasizes a separation between the software's business logic and display. The three parts of Model View Controller design pattern are:

- **Model**: manages all the data and the business logic

- **View**: handles the GUI used by all the users
- **Controller**: acts on both model and view, it routes commands to the view and model elements

MVC is also very useful because the decoupling of these three components allows parallel development and code reuse.

2.7 Other design decisions

3 User interface design

3.1 Interface mockups

3.2 UX Diagrams

3.2.1 Mobile application

3.2.2 Web application

4 | Requirements traceability

5 Implementation, integration and test plan

5.1 Implementation plan

5.2 Integration and testing

5.2.1 Entry criteria

5.2.2 Elements to be integrated

5.2.3 Integration testing strategy

5.2.4 Sequence of component/function integration

6 | Effort spent

Marco Premi

Chapter	Effort (hours)
Chapter 1 - Introduction	0
Chapter 2 - Architectural design	0
Chapter 3 - User interface design	0
Chapter 4 - Requirements traceability	0
Chapter 5 - Implementation, integration and test plan	0
Total (hours)	0

Fabrizio Siciliano

Chapter	Effort (hours)
Chapter 1 - Introduction	0
Chapter 2 - Architectural design	0
Chapter 3 - User interface design	0
Chapter 4 - Requirements traceability	0
Chapter 5 - Implementation, integration and test plan	0
Total (hours)	0

Giuseppe Taddeo

Chapter	Effort (hours)
Chapter 1 - Introduction	0
Chapter 2 - Architectural design	0
Chapter 3 - User interface design	0
Chapter 4 - Requirements traceability	0
Chapter 5 - Implementation, integration and test plan	0
Total (hours)	0