

AA 2019/2020

DD – design document

Version 1.0 – 9-12-2019

Computer Science and Engineering – Software Engineering 2

Amedeo Carrioli – 866256, Andrea Ceruti – 945604

Professor Elisabetta Di Nitto

POLITECNICO DI MILANO

Table Of Contents

[**1.** **Introduction** 2](#_Toc25482130)

[**1.1** **Purpose** 2](#_Toc25482131)

[**1.2** **Scope** 2](#_Toc25482132)

[**1.3** **Definitions, acronyms, Abbreviations** 3](#_Toc25482133)

[**1.4** **Reference documents** 4](#_Toc25482134)

[**1.5** **Document structure** 4](#_Toc25482135)

[**1.6** **Revision History** 4](#_Toc25482136)

[**2.** **Architectural design** 4](#_Toc25482137)

[**2.1** **Overview** 4](#_Toc25482138)

[**2.2** **High level components and their interaction** 4](#_Toc25482139)

[**2.3** **Component view** 4](#_Toc25482140)

[**2.4** **Deployment view** 4](#_Toc25482141)

[**2.5** **Runtime view** 4](#_Toc25482142)

[**2.6** **Component interfaces** 4](#_Toc25482143)

[**2.7** **Selected architectural styles and patterns** 4](#_Toc25482144)

[**2.8** **Other design decisions** 4](#_Toc25482145)

[**3.** **Algorithm design** 4](#_Toc25482146)

[**3.1** **License plate recognizing algorithm** 4](#_Toc25482147)

[**3.2** **Making suggestions algorithm** 4](#_Toc25482148)

[**4.** **User Interface design** 4](#_Toc25482149)

[**4.1** **Mock-ups** 4](#_Toc25482150)

[**4.2** **UX diagrams** 4](#_Toc25482151)

[**4.2.1** **Private mobile app** 4](#_Toc25482152)

[**4.2.2** **Authority mobile app** 4](#_Toc25482153)

[**4.3** **BCE diagrams** 4](#_Toc25482154)

[**4.3.1** **Customer mobile app** 4](#_Toc25482155)

[**5.** **Requirements traceability** 4](#_Toc25482156)

1. **Introduction**
   1. **Purpose**

The DD purpose is to give an overall description of Safestreet. In particular we analyze the architecture, in terms of computational components and interactions among those components.

Unlike the RASD, it will have a more detailed description of the backend of the application, underlining the aspects of the application such as the high-level architecture, the runtime behavior and the algorithm design.

* 1. **Scope**

Here we give an overview of the application based on the descriptions presented on the RASD.

SafeStreets is an application designed to report street violations committed by vehicles on the road.

The users are allowed to make reportings of those violations through the application. In order to do that, a user must be signed in.

Two types of users can access the application and its functionalities: private customers and authorities. The first ones can use the basic functionalities, for example make reportings or interacting with the map on which the reportings made by other users are shown. The second ones, authorities, besides the basic functionalities can also make an information cross with the application, and the application, in specific cases, can send suggestions to authorities, in order to prevent more violations.

The violations reported by the users are stored in a map obtained from GoogleMaps API, which has areas colored with different colors (green, yellow, red) depending on the frequency of the violations in that specific area.

When an area is red for a specific types of violations, for example parkings on the sidewalk, a suggestion is made by the S2B and sent to authorities.

Users, on the map, can filter date and type of the violation (parking, traffic lights, accident and speed violations) and the interval of time they want to see the map (today, last week…).

They can also type on a specific area, the map will zoom in, showing all the reportings in that area, in the position they were uploaded with, then the users can type on a single reporting, and the information about it are shown, for example date, time, position, picture if present.

Authorities, as said earlier, can crosse information with the application, sending accident reportings and receiving from the application parking violation reportings, because are the only reliable repotrings that the application can guarantee.

The main purpose of SafeStreets is to make streets safer by allowing citizens to help each other throughout reportings of streets violations. SafeStreets helps to keep the streets clear also by making suggestions to prevent more violations, these suggestions are sent to authorities, which have more power to take measures on the streets.

* 1. **Definitions, acronyms, Abbreviations**

**Definitions**:

* **Users**: a generic customer of the application.
* **Authority**: specific customers of the application. They are allowed to cross information with the application.
* **Private**: customers of the application that can’t cross information with the application.
* **Reporting**: a signalation on any street infringement made by any user, uploaded on the application.
* **Application analyzing pictures algorithm**: the algorithm through which the license plate is recognized from the picture of the vehicle.

**Acronyms**:

* **RASD**: Requirement Analysis and Specification Document
* **DD**: Design Document
* **CPU**: Central Processing Unit
* **DB**: Database
* **DBMS**: Database Management System
* **GPS**: Global Position System
* **API**: Application Programming Interface
* **TSL**: Transport Layer Security
* **SQL**: Structured Query Language
* **UI**: User interface
* **HTTP**:  HyperText Transfer Protocol

**Abbreviations**:

* [Gn]: n-th goal
* [Rn]: n-th functional requirement
  1. **Reference documents**
* Specification document: “SafeStreets Mandatory Project Assignment”.
* IEEE Std 830-‐1998 IEEE Recommended Practice for Software Requirements Specifications.
* Examples documents:

· “DD from the car sharing project”.

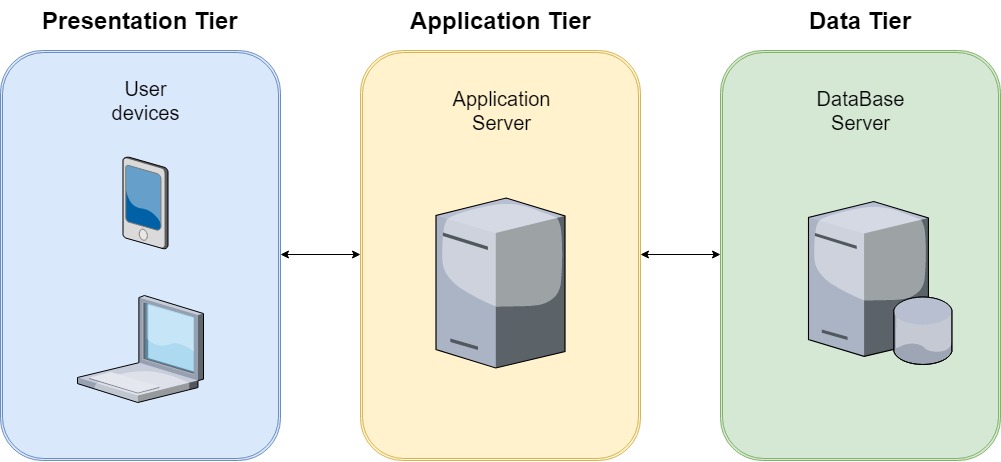
· “DD from Travelendar+”.

· “DD to be analysed AY 2019-20”.

* 1. **Document structure**
* Section 1: An overview of the design document, giving and describing the scope and purpose of it. Definions, acronyms and abbreviation are listed as well.
* Section 2: It gives a description of the architecture of the entire system. This is the core of the document because it contains the most relevant architecture views and decisions:
  + Overview
  + High level components and their interaction
  + Component view
  + Deployment view
  + Runtime view
  + Component interaction
  + Selected architectural style and patterns
  + Other design decision

* Section 3: It presents the two principal algorithms managed by the application. The algorithms are described with the help of a pseudo code in order to let the developers to have the highest degree of freedom.
* Section 4: Here are described in details UI (user interface) already presented in the RASD document.
* Section 5: In this section is present the requirements traceability that links the requirements wrote in the RASD to the design element of the DD.
* Section 6: Description of the implementation, testing and integration of the system.
* Section 6: Shows the effort spent by each member of the group.
  1. **Revision History**
* Version 1.0: First release

1. **Architectural design**
   1. **Overview**



* 1. **High level components and their interaction**



* 1. **Component view**

**Immagine che contiene screenshot

Descrizione generata automaticamente**

**User mobile services projection**

**Immagine che contiene mappa

Descrizione generata automaticamente**

**User web services projection**

**Immagine che contiene mappa, testo

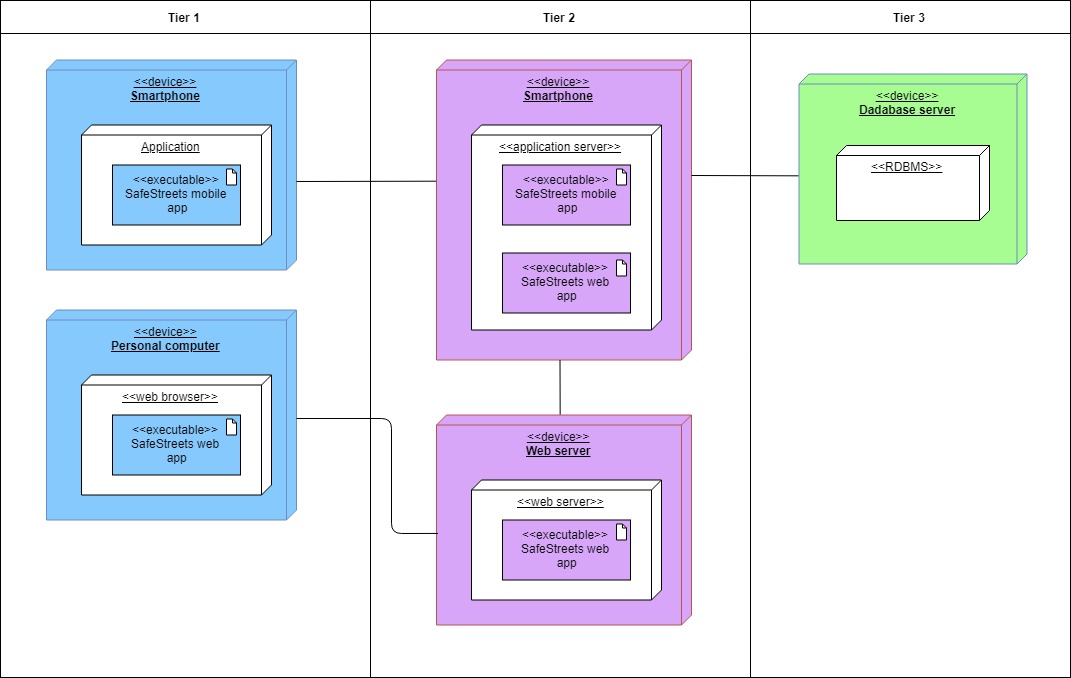
Descrizione generata automaticamente**

**Entity-relationship diagram**

**Immagine che contiene testo

Descrizione generata automaticamente**

* 1. **Deployment view**



* 1. **Runtime view**
* Login

Immagine che contiene screenshot

Descrizione generata automaticamente

* Make a reporting

Immagine che contiene screenshot

Descrizione generata automaticamente

* Cross data (from web page or from mobile app?)

Immagine che contiene screenshot

Descrizione generata automaticamente

* Show particular things on map (specific time and reportings type or specific reporting)?

Immagine che contiene screenshot

Descrizione generata automaticamente

* Show personal info or past reportings

Immagine che contiene screenshot

Descrizione generata automaticamente

* Make suggestion and send it to authorities? (algorithm?)
  1. **Component interfaces**

**Immagine che contiene screenshot

Descrizione generata automaticamente**

**Immagine che contiene screenshot

Descrizione generata automaticamente**

* 1. **Selected architectural styles and patterns**
  2. **Other design decisions**

1. **Algorithm design**
   1. **License plate recognizing algorithm**

This algorithm is activated when the system receives a parking reporting with a photo attacked. The algorithm takes care of reading the license plate of the car in the photo. If it coincides with the license plate manually written by the user, the reporting is sent to the authorities. Otherwise, either if the user only types the license plate without sending a picture of it or the application analyzing pictures algorithm can’t recognize the plate, it is considered not reliable, so the reporting won’t be shared with authorities, but It will only be added on the map. Here follows a more technical explanation about the algorithm flow-chart, without constraining it to particular platforms or programming languages.

The algorithm is divided in these macrophases: firstly, it converts true color image into binary image. Secondly, by using filtering it converts the binary image into binary filtered image to remove the unnecessary information and then by using image processing toolbox, it figures it out the connected components in the binary filtered image. Lastly, using the correlation coefficient, the function identifies each letter from the license plate image. Let’s take a closer look at each stage.

1. Load image: in the first stage, the user loads the image of the car with visible license plate in the GUI; it goes to the second stage.
2. Filtering: This stage converts the true color image into a binary one and it processes and filters it out unnecessary information and noises. Once we have a filtered binary image, it moves to a third stage.
3. Identifying zone of interest: This stage uses an image processing Toolbox to figure it out the connected components in the binary filtered image. The alignment, height, position of this components are taken into considerations and then the program figures it out the set of components which actually holds the required information and select it.
4. Recognizing letter: on the fourth stage the program sends the snaps of these selected components to our read letter function. This function matches the snaps against our pre-recorded database which is analogous to the alphabet in our memory. Using the correlation coefficient, the function identifies each letter and returns it. Thus the program saves the letter in the string. To identify the space, the program simply judges the distance between the significantly enlarge than the average; it inserts a white space into the string.

Finally, the string obtained is compared to the license plate manually written by the user. If the two license plates are the same, the reporting is saved in a staging area and once a day a notification is sent to the authorities, who access the web app and download all reporting in the staging area with a .sfst format (also their reportings to SafeStreets are sent with this format, this is the company official format, which is used to compress reportings). Otherwise if license plates are not equal or license plate is unreadable, the reporting remains saved only in SafeStreets database.

* 1. **Map update**

To allow the use of this algorithm, which as the name shows is responsible for updating the map with the reports received by users, SafeStreets deals with the problem of associating with each received report an area of interest, which can be observed later. The following describes a possible solution.

Once you receive the city map from GoogleMaps API, the software divides the entire area where the application operates in “neighborhoods”. This subdivision is an implementation issue and will be disregarded later in the algorithm description, assuming that designers have already passed this part and provided us with a class of objects called “Neighborhoods” to work with inside the application model.

Each time a user submits a report, the algorithm takes care of saving it in the neighborhood object and consequently on the Map object, which consists of multiple neighborhoods (organized, for example, in a oriented graph). Neighborhood objects will then have saved lists of reportings within them, with their attributes, which is important to mention two of them: the date and the violation type.

Another useful feature of the algorithm is to allow the user to interactively search the map, through another query application that parameterizes user requests, it inserts parameters such as time interval and type of Violation. The algorithm thus returns for each neighborhood the list of reportings that meet the parameters entered by the user, which will then see the results thanks to a method in the client side view (Map Color).

The implementation of this algorithm depends on the platforms on which it will be operated, so we simply described the principle behind it, describing the design part at a high level.

* 1. **Make suggestion algorithm**

This algorithm takes advantage of a Map Update feature, in fact it creates once a day, a reduced Map object with these parameters: Type of violation = parkingViolation, interval of time = last week, thus obtaining from Map Update the map of violations concerning parking of last week. Subsequently for each Neighborhood object in the city performs these operations:

* Separate the reporting list into different lists for each subtype (sidewalk, bus line, ...).
* Count the length of each subtype list.
* If the list number exceeds a certain threshold (we can say a number but we prefer to leave it to the design part of the software) a specific suggestion is generated for that neighborhood that is saved in a list of suggestions.
* Create a new data structure where we find the association neighborhood-list of suggestions

This algorithm then writes the neighborhood/suggestion association to a document in word format, which is loaded together with the list of all the reportings (.sfst format) in the staging area where the authorities interface. It is good to mention that once the word file is created and uploaded, the authorities will get a notification on their web app to remind them to download the staging area.

* 1. **Map color**

Map color is the application that interfaces with the user's map. Map color sees all neighborhood objects saved by the Map Update software. For each neighborhood represented on the map, Map color by default looks at the entire list of reportings present and calculates a value k = Total number of reports on total reportings types, so we get the average of reportings per type.

Next, based on some tables that measure the degree of danger of the zone (for example, for a k value between 0 and 10 means that the area is green, and so on) it colors the neighborhood on the given map so that the user can have a visual feedback.

When a user wants to see the map, it is displayed as the default with all reports of any type in the last month. Once the user wants to filter it, the user must enter the time interval and type of the violation to see the custom map. These parameters are sent to the Map Update application that lightens the model by returning to Map Color the Neighborhood objects with The requested parameters.

The algorithm then reloads the map from the GoogleMaps API and colors the neighborhoods according to the reporting lists received. If neighborhood don't have reportings will be the default color, otherwise the danger scale is green, yellow, red in ascending order. Finally it allows the user to see the coloring from its app on the phone.

Once again we limited ourselves to describing the functionality of the algorithm by exulting from implementation to leave complete freedom to subsequent software developers to choose language and platform.

1. **User Interface design**

The application UI mockups were presented in the RASD document, section 3.1.1 User Interfaces.

Below are presented two diagrams related to both privates and authorities front end application, in particular they expose the flow of the main menu that the user can navigate from their devices. The application windows are represented as colored rectangles half blue and half white, while the available actions in the windows and the credential recovery as a simple rectangles. The action to return to the previous menu is omitted for clarity.

Finally, the small green rectangle (success) represents the access to the application servers and the recognition of the user's activity ,the red one (logout) represents the logout from the application.

Immagine che contiene mappa

Descrizione generata automaticamente

Immagine che contiene screenshot

Descrizione generata automaticamente

* 1. **Mock-ups**
  2. **UX diagrams**
     1. **Private mobile app**
     2. **Authority mobile app**
  3. **BCE diagrams**
     1. **Customer mobile app**

1. **Requirements traceability**