Politecnico di Milano

SAFESTREETS

POLITECNICO
MILANO 1863

Samuele Meta - Stiven Metaj

Supervisor: Matteo Rossi

SOTTOTITOLO OLEEEEEEEE

Department of Computer Science and Engineering

November 1, 2019

Contents

1 Introduction

The following Requirement Analysis and Specification Document (RASD) aims at illustrating a complete overview of the project *SafeStreets*, providing a baseline for its planning and development. It guides the reader in understanding the specifics of the application domain and the relative System in terms of functional requirements, non functional requirements and constraints. It details how, according to these, the System interacts with the external world, showing concrete use case scenarios. A more comprehensive description of the most relevant features will be modelled with the use of the Alloy language. This document is addressed to all the stakeholders - such as users, system and requirement analysts, project managers, software developers and testers - who will evaluate the correctness of the assumptions and of the decisions contained in it.

1.1 Purpose

SafeStreets is a new crowd-sourced mobile application that allows Citizens to notify Authorities about traffic violations, with particular emphasis on parking contraventions. In fact, every day is possible to encounter minor traffic infringements that affect the driving experience, whether it's a double parking or an unduly occupied parking lot reserved for disables. With *SafeStreets*, Citizens can actively participate in road monitoring, partially compensating for the impossible ubiquity of patrols.
To do so, *SafeStreets* requires the User to create an account as Citizen - providing personal credentials such as name, surname, fiscal code - and to verify it sending a picture of a valid document. From this moment on, the System will allow the Citizen to send photos of traffic violations, enriched by the relative description, position, date and time. At any time, the application allows the Citizen to visualize the history of the reported violations since it's registered. In case of error or incorrect information, the System allows the Citizen to re-

voke the alert produced.
Futhermore, the System offers the possibility to sign up as an Authority, once filled the registration form and verified the institutional identity through the relative burocracy. *SafeStreets* will allow Authorities to have a complete overview of the incoming signalations and to choose if to accept or reject them. Since the System stores all the violations and their metadata, *SafeStreets* is able to process them in order to extract meaningful insights and statistics. The System allows both Citizens and Authorities to access them, with a different visibility level according to the role.
Moreover, if the municipality offers a service that allows to retrieve the information about the accidents occurred in the covered area, *SafeStreets* will cross the two knowledge bases to identify potentially unsafe areas and suggest possible interventions.
The above description can be summarized as follows, in a list of goals:

- [**G.1**] The System allows the User to register to the application either as *Citizen* or as *Authority*, providing an identification code and a password.
- [**G.2**] The System allows the User to report a traffic violation by sending a photo of it and the relative date and position.
- [**G.3**] The System allows the User to find out the streets with highest frequency of violations.
- [**G.4**] The System allows the Authority to find out vehicles that commit the most violations.
- [**G.4**] The System allows the Authority to cross the information in order to identify potentially unsafe areas.
- [**G.5**] The System suggests the Authority possible solutions to the problem.
- [**G.6**] The System allows the User to see the reported violations.
- [**G.7**] The System allows the User to retire a violation.
- [**G.8**] The System completes the report with metadata
- [**G.9**] The System suggests the Authority possible solutions to the problem.
- [**G.10**] The System analyzes plate, model quality of photo

- [**G.11**] The System checks if photo is good (security).
- [**G.11**] The System allows the Authority to accept or refuse a violation.

1.2 Scope

Following the original definition of the so call *World and Machine phenomena* (proposed by M. Jackson and P. Zave), we will clarify the scope and the application range of the proposed System; in order to do this we have to define the *entities* involved.

In particular the *Machine* is the System to be developed, in this case a software application, while the *World* corresponds to the part of the real world that is altered by the System.

This takes us to different types of events which we must describe; in fact the *World* and the *Machine* are associated with distinct *phenomena* whose descriptions follow.

1.2.1 World Phenomena

World phenomena are events that occur in the real world and don't impact directly the System. We identify the following ones:

- **A traffic violation occurrence**: any infraction like double parking, parking on sidewalks, no parking sign violation.
- **A car accident**: an accident that involves the *User* or that the *User* sees.
- **A lane for people with disabilities or for bike riders obstructed**: a parking violation that, in particular, obstruct a reserved lane.
- **An increase of violation in a specific area**: many traffic violations can occur in the same in a specific area or in a precise district.

1.2.2 Shared Phenomena

Shared phenomena are world phenomena that are, as the name suggests, shared with the Machine.

These are splitted in two categories: phenomena *controlled by the World and observed by the Machine* and phenomena *controlled by the Machine and observed by the World*. We identify as the first kind the following ones:

- **A *User* registers or logs in to the application**: a *User* must register in order to send violation occurrences or can login to the application if already registered.
- **A registered *User* try to verify the profile**: after a *User* registers to the application he must verify the profile to have the possibility to report traffic violations.
- **A *User* sends photos and information to the *System***: a *User* can send pictures and other information about a violation he/she wants to report.
- **Municipality offers traffic violations information**: municipality can let the *System* retrieve data about occurred traffic violations.

Instead, we identify as the second kind the following ones:

- **The *System* shows a confirmation message**: after a *User* send a violation occurrence the system shows a confirmation message.
- **The *System* asks to confirm the retrieved violation's street**: the application retrieve the street information from the GPS signal of the *User* device and ask hr a confirmation in order to be sure to have the right violation's street information (if not the *User* input it).
- **The *System* shows the list of reported violations by a *User***: a *User* can ask the application to have a list of all his/her reported violations.
- **The *System* shows lists of areas based on stored violations**: a *User* can ask the application to show the areas where there is the higher frequency of violations.
- **The *System* shows lists of vehicles that commit the most violations**: the *Authorities* can ask the application to show the vehicles that commit most violations (i.e. in a specific area).

1.2.3 Machine Phenomena

Finally, machine phenomena are events that take place inside the System, but there is no way to observe them in the real world. We identify the phenomena that follows:

- **The *System* store the reported violations**: all the reported violations are stored in the database of the *System* (including pictures, date, time, type of violation, position and additional information input by the *User*.
- **The *System* compute a veracity control of reported violations**: the *System* uses a Machine Learning algorithm in order to find if the violation pictures are authentic or not (the information is saved to permit to understand veracity of *Users*).
- **The *System* creates lists of areas and vehicles of interest periodically**: a computation is periodically performed in order to have "ready-to-view" lists of areas with higher frequency of violations and lists of vehicles that commit most of the violations.
- **The *System* retrieve information from Municipality**: the API offered by the Municipality is periodically used to check if new data from *Authorities* can be retrieved in order to cross them with information sent by the *Users*.
- **The *System* compute a list of possible interventions in areas of interest**: a Neural Network is used with the data stored in the database in order to execute a Natural Language Process that permits to create sentences about possible interventions based on the reported violations.

1.3 Definitions, Acronyms, Abbreviations

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

1.3.1 Definitions

- *User*: whoever interacts, in any way, with the application
- *Guest*: an User who has not completed the registration and can only use the application to view the statistics
- *Logged User*: an User who completed the registration and who is logged in the application
- *Citizen*: a person with a strong public spirit who registered and logged in to report traffic violations

- *Authority* : a member of the law enforcement who registered and logged in to monitor the incoming traffic violations
- *Municipality API*: service offered by the Municipality to the System to retrieve data of interest to compute statistics
- *Traffic violation*: occurrence of infringement of the laws that regulate vehicle operation on streets and highways
- *Parking violation*: specific category of traffic violations that involves all the incorrect ways to park a vehicle
- *Report*: document generated on the basis of the data provided by the Citizen signaling a violation. It includes the date, the time, the street name, the plate and the photos useful to testify the contravention
- *Historical Data*: all data provided to the System by the Citizen since his/her registration
- *Fiscal Code*: a 16 characters code used in Italy to uniquely identify a person

1.3.2 Acronyms

API	Application Programming Interface
DBMS	DataBase Management System
GPS	Global Positioning System
UI	User Interface
URL	Uniform Resource Locator

Table 1.1: *Acronyms*

1.3.3 Abbreviations

- [**Gn**]: n-th Goal
- [**Dn**]: n-th Domain Assumption
- [**Rn**]: n-th Requirement

1.4 Revision History

[TBD]

1.5 Reference Documents

- Specifications document: "SafeStreets. Mandatory project assignment"
- IEEE Standard 830-1993: IEEE Guide to Software Requirements Specifications
- IEEE Standard 830-1998: IEEE Recommended Practice for Software Requirements Specifications
- Alloy documentation: `http://alloy.lcs.mit.edu`
- UML documentation: `https://www.uml-diagrams.org`

1.6 Document Structure

The rest of the document is organized as follows:

- **Overall Description**: in this section a more in-depth description of the application's domain will be provided, highlighting the context of the System and detailing the phenomena previously mentioned. The most relevant functions of the System will be pointed out and their interactions with all the actors will be illustrated with the help of Class Diagrams. Finally, this section includes all the constraints, dependencies and assumptions that can be used to entail each Goal.
- **Specific Requirements**: in this section the requirements will be fully explained and organized according to their type, associating to them the relative use cases and Sequence Diagrams to clarify the interactions between the User and the System. The main aim is to provide a useful item for both designers and testers.
- **Formal Analysis**: this section includes the formal model generated according to the Alloy language.

2 Overall Description

2.1 Product Perspective

In this section we discuss the details about all the *shared phenomena* defined in the previous section; furthermore we provide an inspection of the domain model at different levels of specification (adopting the use of class and state diagrams).

A User registers or logs in to the application *(world controlled, machine observed)*
A guest can compile the registration form in order to register to the application (without the registration step no violation reports are allowed). If a *User* is already registered the login form is available.

Two different types of credentials, in the registration phase, are possible: if the *User* is a *Citizen* the form contains the Fiscal Code, the Name, the Surname, the Date of Birth and the Password fields; instead, if the *User* is an *Authority* the form contains the Identification Code and the Password fields.

If the registration is valid a confirmation email il sent.

A registered *User* try to verify the profile *(world controlled, machine observed)*
Once the *User* is registered and logs in the application no violation report is possible until the profile is verified. In order to do that the *User* must first send a photo of an Identity Document (i.e. ID Card, license or passport) and a selfie in a certain position. The *System* must process the data received to confirm the Identity of the *User* and send him a notification of the occurred verification.

A User sends photos and information to the System *(world controlled, machine observed)*
After that the *User* is verified the violation reporting form is available. With

that the *User* can send to the application a report with all the information of a specific violation; here he must add at least one picture of the violation, permits to retrieve the GPS signal position (writing however the right position if the GPS is wrong) and indicates the type of violation.

Furthermore the *System* runs an algorithm in order to retrieve the license plate of the vehicle in the pictures, the *User* can help it writing the license plate (in that case the algorithm will verify it).

Municipality offers traffic violations information *(world controlled, machine observed)*
The application idea is based on the goal of making the traffic situation as safe as possible; to support this concept an help from the Municipality is needed. In fact, thanks to an API offered by the municipality itself, when new data about traffic violations are provided, the *System* retrieve them automatically and perform a cross operation with the own data stored in the database.

With this effort the data can be verified and extended in order to train a model implementing a Natural Language Process that permits to the application to provide not only a list of unsafe areas, but also possible intervention (using the natural language of the country of interest) to improve the safeness of them.

The *System* shows a confirmation message *(machine controlled, world observed)*
When a *User* send a violation to the *System* the application checks that all the mandatory fields are filled and that no information is lost while sending the report (due for example an internet connection problem). If no problems are raised the application must display a confirmation message communicating to the *User* that the violation is correctly stored in the database and it will be processed.

The *System* asks to confirm the retrieved violation's street *(machine controlled, world observed)*
In the violation information the name of the street where the violation occurred is required; during the violation reporting step the application try to retrieve it from the GPS signal of the *User* device (the *User* should already give to the application the right permission in order to access to the GPS data). Definitely the street name must be the right one; for this purpose the application does not perform a "blind" operation but asks the *User* to verify the street name it has retrieved: if it is correct the *User* must only accept it, if not he/she has to provide the right one.

The *System* shows the list of reported violations by a *User* *(machine controlled, world observed)*
Once a *User* is registered to the application and his/her profile is verified there are no limits to the number of violation that can be reported. An evident feature the application must have is that to show the list of violations the *User* has reported. This can allow future implementations of different features like a possible addition of information to reports or like the opportunity to delete or modify old violations.

The *System* shows lists of areas based on stored violations *(machine controlled, world observed)*
The main goal of the application drives this feature: *Users* registered to the application (the verify of the profile is not required in this case) can access to a list of unsafe areas based on their GPS position. This possibility can help Citizens to stay away from those areas or can bring them to explore more often those districts in order to have report more violations (adding more data to the *System* with the purpose of disciplining who commit traffic violations).

The *System* shows lists of vehicles that commit the most violations *(machine controlled, world observed)*
Another goal of the application is to provide more information about violations to the Authorities (directly from the Citizens). Thanks to the data stored by the application the *System* can provide a list of "bad" vehicles. The feature is exclusively enabled only for *Authorities accounts*, and it permits indeed to show a list of vehicles that have committed most of the violations in a specific area or city. This can help to prevent with more efficiency future violations by the drivers of those vehicles.

2.1.1 Class Diagram

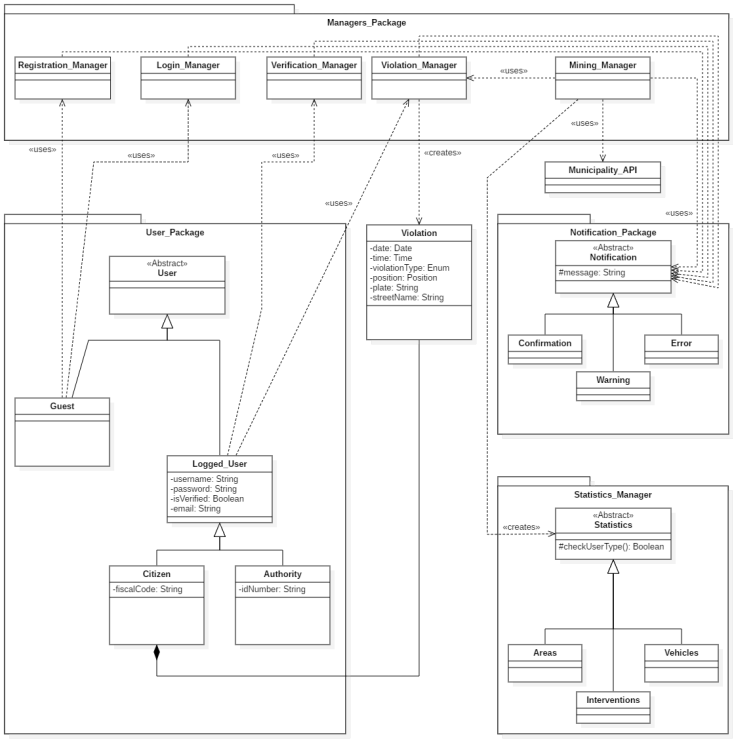

Figure 2.1: *SafeStreets Class Diagram*

2.1.2 State Chart Diagrams

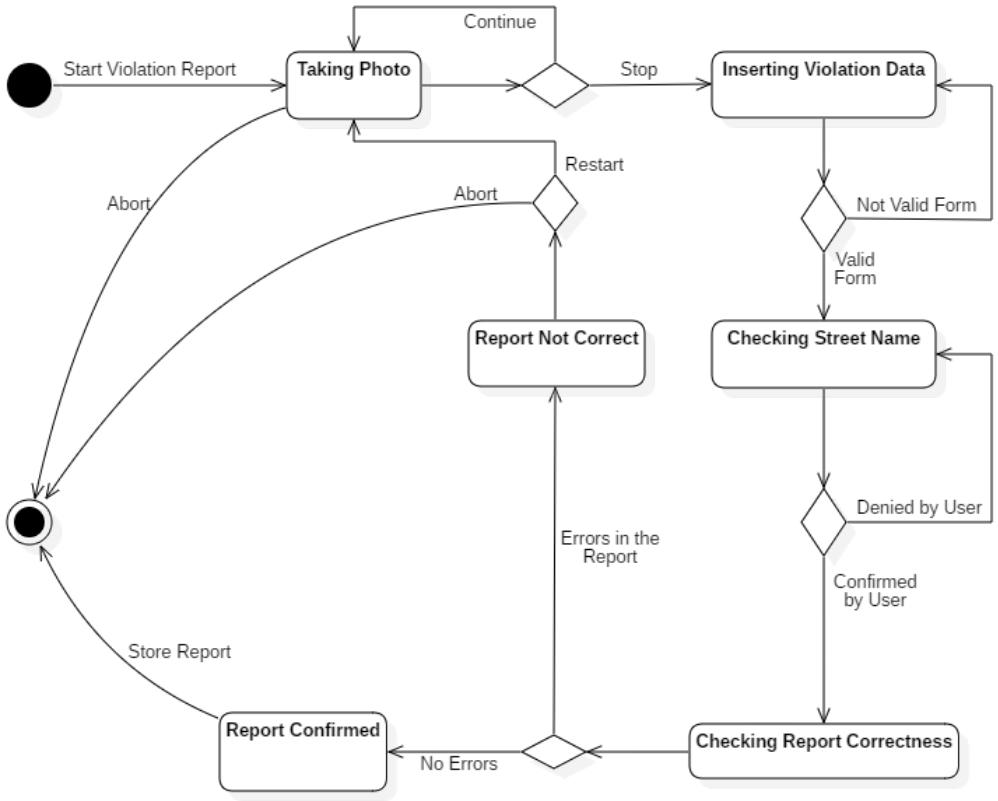

Figure 2.2: *Violation Report's State Chart*

2.2 Product Functions

In this section the main functionalities of *SafeStreets* are presented and grouped according the type of *User* they are addressed:

- **Guest Functions**:
  View Statistics about Unsafe Areas
  The application offers to any *User* (even if not registered or logged in) the possibility to consult the statistics generated by the *System* about unsafe areas close to the GPS signal of the used device (we assume that the device GPS is turned on and the *User* gives permission to the application to use it).

- **Citizen Functions**:
  Report Violations
  The *System* permits to *Citizens* to report street violations (this is actually the main function of *SafeStreets*); the application main GUI contains a primary button that allows the *User* to enter in the Violation Report creation module. The first information the module expect are one or more pictures of the violation, allowing both taking pictures directly from the application or uploading them from the device memory (we assume that the device has a camera and that permissions to use it and to access the memory are given); after the "take pictures step" is finished the application let the *User* enter to the violation's information form: here the information about the report can be input by the *User*. The fields present in the form are the following:

  - *Type of Violation*: the *User* must indicate the type of violation present in the taken pictures. This will help the *System* to check and identify the real type of violation and will also give the possibility to value the report in order to give a positive or negative "score" to the *Citizen User*.
  - *Date and Time*: this field is automatically filled by the application retrieving information about the date and time online (the *System* choose the time when the *User* started to take pictures). The possibility to change this field is, however, given to the *User*, in order to permit to report violations in a second moment (anyhow only date and time that precede the automatic ones are allowed).
  - *License Plate*: this field (not required) can be filled by the *User* in order to give a help to the *System* about the plate of the vehicle committing the violation: the *System* will apply however a computer vision algorithm to retrieve the plate of the vehicle (trying to "read" it from the images and crossing the information about the vehicle model), but if this field is filled (after a check on the data input by the user, for example if a string with a length different from 7 characters is present it will be ignored) the algorithm will use the string to have a better accuracy.
  - *Description*: here the *User* can write a more detailed description of the street violation. This field is not required but permits to add information for helping *Authorities* and the *System* in general (also the description field will be use by the algorithms that run behind the application).

If there is any error in the form the application return a warning and permits the *User* to retry to complete the form.
After completing this form the *User* can proceed to the next step: the street name check; here the application accesses to the device's GPS to retrieve the position (described by latitude and longitude) and try to detect the street name (and approximately the street number, when the street is too long). For a double check of this information (essential to permit the *Authorities* to find as fast as possible the violation location) the app asks the *User* to confirm it or to input the right street name in the case of the retrieved one is wrong

The final step is a correctness check done by the application: if there are errors, like the incapability to retrieve the license plate or an internet connection fail at the ending phase of the report, the application inform the *User* and asks him if he wants to abort the report or if he wants to restart it; if there are no errors the report is stored in the *System* database in order to be retrieved by *Authorities* and to let the *System* itself process it for the statistics it must provide.

View and Delete Old Violations
The *User* is able to access all his/her past violations reports; through a specific tab of the application interface indeed, the *Citizen* can view the list of reports with the possibility to read all the details inputted previously. Furthermore also the possible delete of them is permitted: if a *Citizen* is aware only after some time of a bad old report or notice that a report has some errors not identified before he/she can delete it. This option grants better statistics and, if done in time, to avoid *Authorities* waste of time.

View Specific Statistics
As the *Guest* a *Citizen* is able to view statistics information about the violations stored in the *System*; in particular a *Citizen* has the same ability of highlighting unsafe areas close to him/her in order to avoid them or to keep more attention when he/she is there (both in sense of self safety and in sense of keeping attention to possible violations to report in order to improve the *System* information).

Manage the Application Settings
The application permits the *Citizen* to open the "settings" tab that permits some actions briefly described here:

- Change how and which notifications are displayed.
- Choose if the *System* must also send emails for certain type of notifications.
- Choose the theme of the application (possible Dark Mode or dynamic theme according the time of the day).
- Send a feedback to the developers in order to notify bugs or possible improvements.
- Log out from the application.

.

- **Authority Functions**:
  View and Filter Violations Reports:
  An *Authority* has the possibility (after the verification of the profile) to view all the violations reports stored by the *System* (unlike *Citizens* who can only view their own violations). Furthermore an *Authority* can filter the list he views (i.e. let the application shows only violations occurred in a certain day, or in the last week in a certain area); in this way we try to speed up a possible *Authority* intervention.

  Verify or Deny a Violation Report
  An *Authority* can also, besides the possibility to view and filter violations, verify or deny reports; in this way the *System* can split the violations set in different subsets considering if they are verified, denied or not judged yet (furthermore this feature can help the *System* to assign scores to *Citizens* that have violations verified and to *Authorities* themselves if they work on the "verify/deny" process. To help the *Authority* to handle this the *System* gives a probability of truth at each violation report; thanks to this violations can be viewed in a descendent order with respect to this probability.

  View Specific Statistics
  Like a *Citizen*, an *Authority* can view statistics generated from the *System* using violations information stored (more the app is used, more accurate the statistics are). Unlike *Citizens* in this case *Authorities* can not only view unsafe areas (specifying now around what position search for them), but there are a couple of other possible statistics:

  - *Vehicles that commit most violations*: the *System* processes the stored violations in order to have a list of vehicles that commit

most violations (considering if the violations are verified or not and the probability of truth of them is assigned a "score" also to the vehicles, creating in this way the possibility to show them in a descendent order). Thanks to this feature *Authorities* have one more weapon to discover in a easier way who commit violations or to make their own statistics (crossing for example the result of the application query with private data they have). This feature is not given for *Citizen* for privacy issues.

- *Possible Interventions*: the *System* makes process in another way with the data at his disposal: it tries to create possible suggestions for interventions in discovered unsafe areas. To do that 2 different algorithm are necessary: a Machine Learning one that extract information from data, giving in output possible suggestions (with a relative score that indicates how much is the probability that they are real good interventions) in a mathematical way and a Natural Language Processing one that transform the output of the previous algorithm to sentences easily understandable by human *Users*. Possible interventions suggested are the following:
  - "Add a barrier between the bike lane and the part of the road for motorized vehicles to prevent unsafe parking"
  - "Raise the sidewalk at the right side of the street (looking to north) to prevent vehicles parking on it"

2.3 User Characteristics

SafeStreets is a mobile application where there are two main possible types of *Users*: *Citizens* and *Authorities*. The first type corresponds to individuals that actively take part in the street environment and want to report violations through the application. The second type corresponds to authorities that are able to verify their identity and want to view violations' statistics, view the violations stored in the *System* database (with the possibility to filter them) or want to verify or deny violations (in order to improve the *System* performances).

We give for granted that *Users* have the application installed on their device and can access to Internet while using it. We will now briefly precise which types of users the *System* encounters at different stages:

- **User**: whoever has installed the application and uses it. We identify as *User* who can perform operations where no privileges distinctions are

present.

- **Guest**: a *User* who has downloaded the application, but has not created an account yet. *Guests* can access to the login or registration form and have the possibility to view statistics about unsafe areas.
- **Logged User**: a *User* who is registered with no errors and is logged in the application with his credentials. With *Logged User* we identify *Users* who are logged in as *Citizens* or as *Authorities*.
- **Citizen**: a *Logged User* who, after the verification of the specific profile, can report street violations, view statistics and view the history of his reports (also they have the possibility to delete an old report).
- **Authority**: a *Logged User* who, after the verification of the specific profile, can view statistics (with more options than *Citizens*), the list of violations and can verify or deny violations reports (also for assigning more or less value to the profiles of reported violations).

2.4 Assumptions, Dependencies and Constraints

2.4.1 Constraints

- Users are located in Italy
- Users must have access to an Internet connection
- The System must require Users' permission to acquire, store and process personal data. Therefore, the System must offer the possibility to the Users to delete their personal account and the associated data at any time.
- A violation report contains, at maximum, 4 pictures.

2.4.2 Dependencies

- The System will rely on an external API provided by the municipality to retrieve the information about the accidents that occur on the territory
- The System needs a DBMS in order to store and retrieve Users' data
- The System will make use of a map visualization service

- The System will make use of the GPS services offered by Users' smartphones

2.4.3 Assumptions

- [**D.1**] *Citizens* are uniquely identified, at the registration phase, by their fiscal code, once their identity has been confirmed (??? PERCHè, PRIMA DI ESSERE VERIFICATI POSSONO AVERE UN USERNAME UGUALE A QUELLO DI QUALCUN ALTRO?)
- [**D.2**] *Authorities* are uniquely identified, at the registration phase, by their authority ID, once their identity has been confirmed
- [**D.3**] *Citizens* and *Authorities* are both uniquely identified, at the login phase, by their username!!!!
- [**D.4**] The given email in the registration phase is assumed correct.
- [**D.5**] Position data has an accuracy of 10 meters around the actual position.
- [**D.6**] Permission to access the device memory, the GPS and the camera is always granted to *SafeStreets*.
- [**D.7**] Data is stored on persistent memory.
- [**D.8**] The System storage is replicated and fault-tolerant, so that a copy of saved data is always available.
- [**D.9**] The *User* device operative system supports the application installation and all its features.
- [**D.10**] *Citizens* input correct data in the violation report form.
- [**D.11**] The application can correctly and automatically retrieve the current date and time.
- [**D.12**]

3 Specific Requirements

In this part of the document we discuss in a more detailed way contents presented in the Overall Description. We will describe particularlly what can help and be useful for the development team.

3.1 External Interface Requirements

OLE

3.1.1 User Interfaces

Here we describe and show the different interfaces that the application offer.

- **Logo**
  robaroba
- **Icon**
  ifwfis
- **Sign In/Sign out**
  wefnsnfs
- **Home Page - Citizen**
  isuhself
- **Home Page - Authority**
  isuhself
- **Profile Verification**
  fweinsflk
- **Violation Report**
  fsnflksnf

- **Left Sidebar**
  uwsbflsdf
- **Statistics**
  dijadi
- **Settings**
  idada

3.1.2 Hardware Interfaces

SafeStreets is available for mobile devices that guarantee Internet access. Even if not strictly necessary, a better user experience would be provided by devices that also allow direct access to GPS.

3.1.3 Software Interfaces

- **Operating System**: The application runs both on iOS and Android
- **Municipality System**: The application will communicate with the APIs provided by the municipality to retrieve statistics of interest
- **DBMS**: The application needs to store and retrieve data to perform its main activities

3.1.4 Communication Interfaces

- **HTTPS**: The application will use this protocol to safely communicate over the Internet and with the DBMS.

3.2 Functional Requirements

The functional requirements of *SafeStreets* are the following:

- [**R.1**] A *Guest* must be able to register. During the registration the *System* will ask to provide credentials (Different distinguish between *Citizen* and *Authority*.
- [**R.2**] The *System* must check if the credentials are valid:
  - The chosen username must be unique (invalid if another *User* has already the same one).

  - The password must be longer then 8 characters and contain at least 1 letter and 1 number.
  - The provided email must be in the correct form.
  - The fiscal code (when the *User* is a *Citizen*) and the authority ID (when the *User* is a *Authority*) must be in the correct form.

- [**R.3**] The *System* must store all *User* personal and application-based data such as registration information, credentials, settings, reported violations and viewed statistics.

- [**R.4**] The System must allow *Citizens* and *Authorities* to log in using their correct personal credentials (username and password).

- [**R.5**] The *System* must allow the *User* to change username, only if the new username is not already in use by another *User*, email, only if the new email is in a correct format (and if verification link is opened) and password, only if the new password is different from the precedent and respects the previous constraints.

- [**R.6**] The *System* must send a confirmation email if username, email or password is changed. The *System* must replace the old credentials with the new one and permit the login with them.

- [**R.7**] A *Citizen* must be allowed to report a violation, specifying the following information:
  - The type of violation
  - The date and time of the violations.
  - The license plate of the vehicle that committed the violation.
  - A description of the violation, adding more details.

- [**R.8**] A *Citizen* must be able to view the history of his/her reports and must be allowed (with some limitations) to delete them.

- [**R.9**]

- [**R.10**]

- [**R.11**]

- [**R.12**]

- [**R.13**]
- [**R.14**]

3.2.1 Use Cases

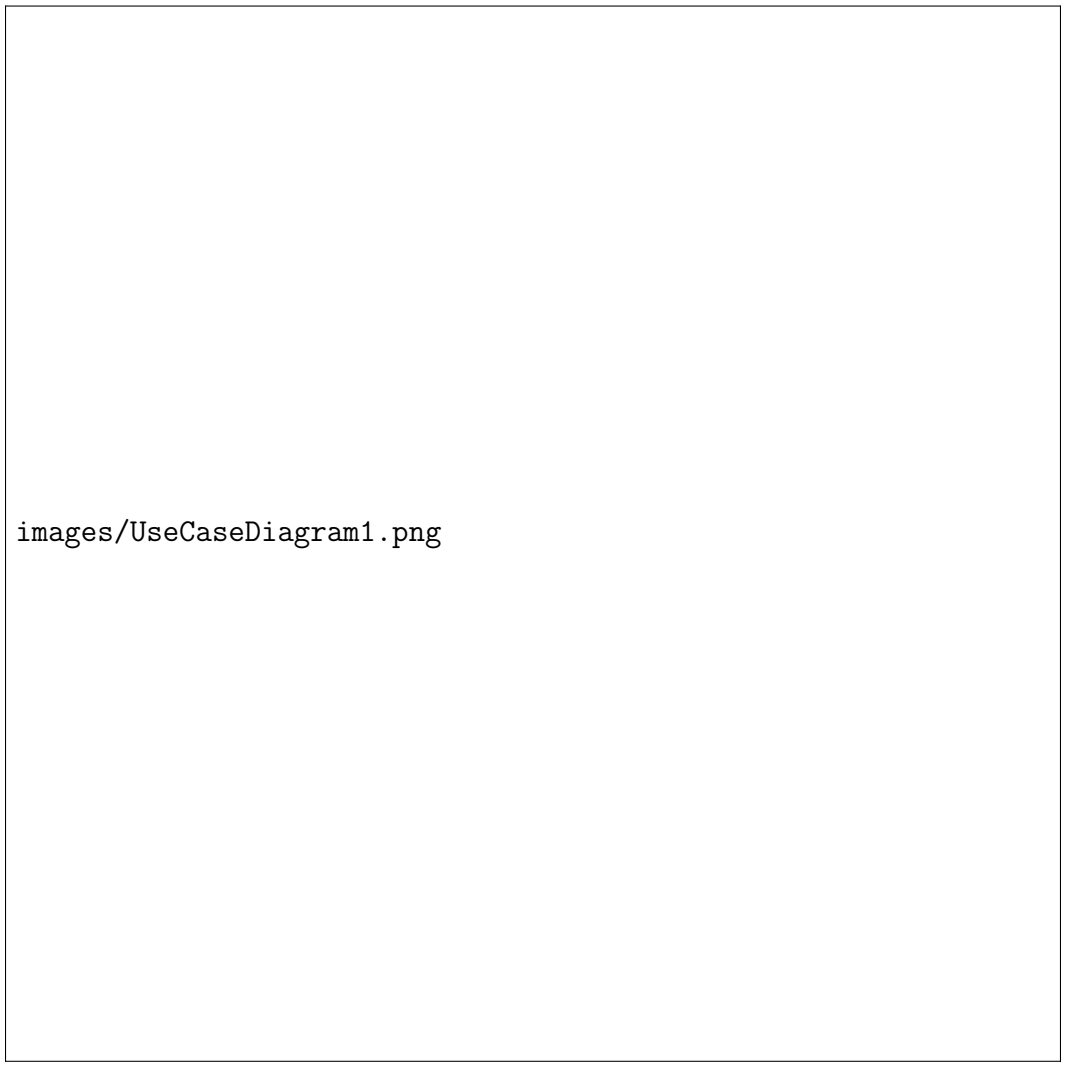

Figure 3.1: *Uses Cases*

3.2.2 Use Case Templates

ID	UC1
Description	A ***Guest*** wants to create a ***Citizen*** account for the application
Actors	***Guest***
Preconditions	• The ***Guest*** has installed and launched the application • The ***Guest*** doesn't already have an account registered
Flow of Events	1. The ***Guest*** selects the *Sign Up* function 2. The ***System*** asks the ***Guest*** to choose a profile between ***Citizen*** and ***Authority*** 3. The ***Guest*** selects the ***Citizen*** profile 4. The ***System*** shows the registration form to enter the required credentials: name, surname, NIN, email address and password. 5. The ***Guest*** fills the form compiling all the fields 6. The ***Registration Manager*** checks the correctness of the information, generates an activation URL and forwards it to the ***Guest***. 7. The ***Guest*** receives the URL and clicks on it to activate its account. The ***Registration Manager*** stores the data provided by the ***Guest***.
Post conditions	The ***Guest*** is able to sign in and, from now on, will be able to login as a ***Citizen***
Exceptions	• The ***Guest*** is already registered to the ***System***. This exception is handled informing the ***Guest*** and taking him back to the homepage. • The ***Guest*** doesn't complete all the mandatory fields or inserts invalid data. This exception is handled showing an alert message to the ***Guest*** and disabling the submit button. The flow of events restarts from point 4.

ID	UC2
Description	A ***Guest*** wants to create an Authority account for the application
Actors	***Guest***
Preconditions	• The ***Guest*** has installed and launched the application • The ***Guest*** doesn't already have an account registered
Flow of Events	1. The ***Guest*** selects the *Sign Up* function 2. The ***System*** asks the ***Guest*** to choose a profile between ***Citizen*** or ***Authority*** 3. The ***Guest*** selects the ***Authority*** profile 4. The ***System*** shows the registration form to enter the required credentials: ?????? 5. The ***Guest*** fills the form compiling all the fields 6. The ***Registration Manager*** checks the correctness of the information, generates an activation URL and forwards it to the ***Guest***. 7. The ***Guest*** receives the URL and clicks on it to activate its account. The ***Registration Manager*** stores the data provided by the ***Guest***.
Post conditions	The ***Guest*** is able to sign in and, from now on, will be able to login as an ***Authority***
Exceptions	• The ***Guest*** is already registered to the ***System***. This exception is handled informing the ***Guest*** and taking him back to the homepage. • The ***Guest*** doesn't complete all the mandatory fields or inserts invalid data. This exception is handled showing an alert message to the ***Guest*** and disabling the submit button. The flow of events restarts from point 4.

ID	UC3
Description	A ***Guest*** wants to login to the application
Actors	***Guest***
Preconditions	• The ***Guest*** already has a valid account • The ***Guest*** opened the application
Flow of Events	1. The ***Guest*** selects the *Login* function 2. The ***System*** asks the ***Guest*** to input his/her credentials 3. The ***Guest*** inserts his/her credentials 4. The ***Guest*** click the *Login* button 5. The ***Login Manager*** checks the correctness of the provided credentials
Post conditions	The ***Guest*** is logged to the application, according to his/her credentials, as a ***Citizen*** or an ***Authority***
Exceptions	• The ***Login Manager*** recognizes invalid credentials then shows an error message. The flow restarts from point 2

ID	UC4
Description	An ***User*** wants to view the statistics regarding the unsafe areas
Actors	***User***
Preconditions	• The ***User*** opened the application
Flow of Events	1. The ***User*** selects the *View Statistics* tab 2. The ***System*** shows to the ***User***, according to his/her role, the available statistics 3. The ***User*** selects *Highlight Unsafe Areas* 4. The ***System*** displays the heatmap of the areas with higher rate of violations
Post conditions	The ***User*** is able to view and analyse the heatmap displayed
Exceptions	• The ***System*** has not sufficient data to extract relevant statistics and shows an error message

ID	UC5
Description	A ***Logged User*** wants to verify his/her account
Actors	***Logged User***
Preconditions	• The ***Logged User*** has not yet verified the account
Flow of Events	1. The ***Logged User*** selects the *My Profile* tab 2. The ***Logged User*** clicks on *Verify My Identity* 3. The ***System*** asks the ***Logged User*** to upload the photo or PDF of a currently valid document that proves univocally his/her identity 4. The ***Logged User*** uploads the required documentation 5. The ***Logged User*** clicks on *Submit* 6. The ***System*** checks the correctness of the documentation received
Post conditions	The ***Logged User*** is verified and, from now on, is able to exploit all the functionalities of the application
Exceptions	• The ***Logged User*** uploads an unsupported file format. The ***System*** notifies the error and the flow restarts from point 3 • The ***System*** does not accept the documentation uploaded by the ***Logged User***. The ***System*** notifies the ***Logged User*** with an error message and the flow restarts from point 1

ID	UC6
Description	A ***Citizen*** wants to report a violation
Actors	***Citizen***
Preconditions	• The ***Citizen*** successfully logged in • The ***Citizen*** has a verified account
Flow of Events	1. The ***Citizen*** selects the *Report Violation* tab 2. The ***System*** shows to the ***Citizen*** the reporting form 3. The ***Citizen*** selects the photos of the violation 4. The ***System*** pre-compiles the *Date*, *Time* and *Position* fields 5. The ***Citizen*** checks the correctness of the *Date*, *Time* and *Position* fields 6. The ***Citizen*** compiles the *Plate* and *Violation Type* fields 7. The ***Citizen*** clicks on *Submit*
Post conditions	The ***Citizen*** is able to submit the violation and this is stored by the ***System***
Exceptions	• The ***System*** pre-compiles the *Date*, *Time* and *Position* fields with wrong data. The ***Citizen*** manually corrects them before point 6 • The ***Citizen*** wants to add or remove an attached photo. The flow restarts from point 3, caching the other compiled fields

ID	UC7
Description	A ***Citizen*** wants to withdraw a violation
Actors	***Citizen***
Preconditions	• The ***Citizen*** successfully logged in • The ***Citizen*** has reported at least one violation
Flow of Events	1. The ***Citizen*** selects the *My Profile* tab 2. The ***Citizen*** clicks on *View Violation History* 3. The ***System*** shows the history of all the violations reported by the ***Citizen*** 4. The ***Citizen*** selects the violation which wants to withdraw 5. The ***System*** asks the ***Citizen*** if he/she wants to procede in withdrawing the selected violation 6. The ***Citizen*** clicks on *Withdraw Violation*
Post conditions	The violation is no longer present in the history of the ***Citizen*** and is deleted by the ***System***
Exceptions	• The ***Citizen*** accidentally selects the wrong violation. When the ***System*** asks him/her for confirmation, the ***Citizen*** cancels the operation. The flow restarts from point 3

3.2.3 Scenarios

- **Scenario**
  Paolo Rossi is an entrepreneur from Florence who just moved to Milan for business reasons. He has found the perfect accomodation for his family in Via Enrico Caruso, which is close both to his workplace and his son's primary school. Unfortunately, the apartment does not include a private parking space, so he would be obliged to park in the streets nearby. Since Paolo doesn't know very well the city, he's scared that finding a good parking can turn into a difficult situation when he is back from work. Then he remembers about *SafeStreets*, the application that appeared in his Instagram feed. Downloading it and without even registering, he is immediately able to highlight the most unsafe areas in Milan, especially looking in proximity of Via Enrico Caruso. Noticing that double parkings are very rare and that generally traffic laws are respect, Paolo, without further hesitation, decides to formalize the acquisition of the house.

- **Scenario**
  Ilaria Bianchi is a nurse who enjoys reaching the hospital by bike in the morning and to keep fit. The hospital, to promote such a virtuous behaviour, has recently converted some car parking slots, close to the main entrance, to bike racks. Nonetheless, who used to park there is very stubborn to continue doing it, despite the prohibition. Since Ilaria is unable to benefit from the racks, many times has tried to report the violation to the hospital offices, but without success. One day, listening to one of her favourite podcasts, she discovers about *SafeStreet*, a new application that allows the users to notify traffic violations directly to the authorities. Immediately she decides to download it and register, verifying her identity. When, once more, she incurs in the same problem, she decides to compile a report providing all the requested data. The next day, since her signalation was accepted, she is very happy to find the free space in front of the racks and to park there her bike.

3.2.4 Sequence and Activity Diagrams

oleole

3.2.5 Mapping on Requirements

oleole

3.3 Performance Requirements

OLE

3.4 Design Constraints

OLE

3.4.1 Standards Compliance

The data format used for the information of the *System* is the following:

- *Meters* are used as unit for distances (like the SI measurement unit for length indicates).
- Standard longitude and latitude measures are used for the position of the violation.

3.4.2 Hardware Limitations

To be able to perform as intended, the application should run in a hardware environment that guarantees, at least, the following specifications:

- 3G connection at 2Mb/s
- 50 MB of mass memory
- 2 GB of RAM

3.5 Software System Attributes

3.5.1 Reliability

The System should guarantees a 99% of reliability: its components must have a failure rate that guarantees this goal.

3.5.2 Availability

The System guarantees an high availability, due to the expected high reliability, in order to offer an ideal 24/7 service.

3.5.3 Security

Since *SafeStreets* involves the storage and usage of many different kinds of private and sensitive data, Security is a key feauture of the System. Thus, the System must:

1. As previously stated, make use of HTTPS to guarantee safe communications, in terms of privacy and integrity, towards the Internet and the DBMS
2. Hash and salt the passwords, avoiding to save them in clear in the DB
3. Encrypt sensitive data before storing them to reduce the effects of data leaks
4. Guarantee strict access control policies, with particular emphasis on avoiding Guest and Citizens to access functionalities meant for the Authorities

3.5.4 Maintainability

In order to speed up the maintenance in case of failure and to facilitate further refinements, the System will have a modular architecture. To achieve this results, good software engineering practices must be followed in order to reduce coupling, avoid code duplication and ensure that modules are self-sustainable.

3.5.5 Portability

Since *SafeStreets* is implemented as a mobile application, portability with regards to the user will be quite natural, as it involves the porting from Android to iOS or *vice versa*. For what concerns the back-end part, it should be OS independent and easily reused with other hosts.

4 Formal Analysis using Alloy

4.1 CODICE SCHIFOSO

OLE

4.2 SCREEN DEL CODICE SCHIFOSO CHE RUNNO E FUNZIONA TUTTO STRABENE

OLE

5 Effort Spent

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

Samuele Meta

Date	Task	Hours
-	First Meeting	-
-	Git and Overleaf Setup	-
-	Assignment Analysis and Workflow Decision	-
-	Purpose and Scope	-
-	Introduction Other Sections	-
-	Product Perspective	-
-	Overall Description Other Sections	-
-	Interfaces Description and Mockups	-
-	Mid-phase Meeting	-
-	Requirements and Constraints	-
-	Alloy	-
-	Effort Tracking	-
-	Global Review	-

Stiven Metaj

Date	Task	Hours
-	First Meeting	-
-	Git and Overleaf Setup	-
-	Assignment Analysis and Workflow Decision	-
-	Purpose and Scope	-
-	Introduction Other Sections	-
-	Product Perspective	-
-	Overall Description Other Sections	-
-	Interfaces Description and Mockups	-
-	Mid-phase Meeting	-
-	Requirements and Constraints	-
-	Alloy	-
-	Effort Tracking	-
-	Global Review	-