SafeStreets
RASD document

Dario Miceli Pranio
Pierriccardo Olivieri

Academic year: 2019 - 2020

POLITECNICO
MILANO 1863

Contents

1 Introduction

This is the RASD document for *SafeStreets*, that provides a general view about key aspects of the project. The purpose of this document is to formalize a description of the *System's* requirements both functional and non-functional. In the following pages will be covered goals of the application with respect to phenomena. This document is addressed to developers as a guideline to implement the requirements that follows and as an overview for stakeholders.

1.1 Purpose

SafeStreets is a service that aims to provide *Users* with the possibility to notify *Authorities* when traffic violations occur, and in particular parking violations. The application's goal is achieved by allowing *Users* to share photo, position, date, time and type of violation and by enabling *Authorities* to request them.

SafeStreets requires the *Users* to create an account to access its services, the functionalities unlocked after registration depend on the type of account created.
If a *User* creates an account as *Citizen*, he/she must provide name, surname and a fiscal code in order to prove that he/she is a real person. Furthermore, he must provide an email with which he will be uniquely identified and a password. Once the account has been activated, *User* can finally start to report parking violations and can also see statistics of the streets or the areas with the highest frequency of violations.

On the other hand, an officer will create an account as *Authority* and he will need to provide his name, surname, work's Matricola, a password and as for *Citizen*, will be uniquely identified by an email. Once the Matricola has been verified and the account has been activated, the officer can retrieve the potential parking violations sent by *Citizen* that have not been taken into account yet by other officers, analyze them and, if it is the right case, generates traffic tickets. *Authorities*, can see the same statistics of the *Citizen* and can also see statistics about vehicles' license plate that commit the most violations.

From this brief description of the functionalities we may extract the following goals for *SafeStreets*:

- **[G1]**: Allow *Guest* to be registered as a *Citizen* or as *Authority*;
- **[G2]**: Allow *Citizens* to report parking violations;
- **[G3]**: *Citizen* has to be able to input information about the violation that he has reported;
- **[G4]**: Must provide a visualization of the areas with high frequency of violations to *Users*;
- **[G5]**: Must provide a visualization of vehicles that commit the most violations to *Authorities*;

SafeStreets offers also some advanced functions in addition to the basic version.

- **[G6]**: Must ensure the chain of custody of the information sent by *Citizens*;

- **[G7]**: *Authorities* can retrieve traffic violations' in order to generate traffic tickets;
- **[G8]**: *System* must build statistics with the informations about issued tickets;

1.2 Scope

Here we will describe all the relevant phenomena that may occur.

1.2.1 World Phenomena

Those are the events that may occur in the real word and are not affected by the Machine.
We identify:

- *Citizen* sees a parking violation and wants to report it;
- *Users* want to know about some violations that have been occurred;
- A *parking violation* occurs;

1.2.2 Shared Phenomena

Shared phenomena are the events based on the link beetween World Phenomena and Machine Phenomena. We can distinguish them in two types:
Controlled by the world observed by the machine:

- A *Citizen* reports a violation;
- *Users* can enter data for registration/login;
- *Users* can request data;

Controlled by the machine observed by the world:

- Track position of the violation;
- Mark areas with an high rate of violations;
- *System* can fullfill data requests;

1.2.3 Machine Phenomena

The Machine Phenomena are the events that occur inside the machine and are not affected by the real world.
We identify:

- Storing permanently collected data;
- Encryption of sensitive data;
- Retrieving data for a request;

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

- *Users*: can be either *Citizen* or *Authority*
- *traffic violation*: generic violation that can occur in a street
- *parking violation*: a violation caused by a bad parking
- *violation*: general violation, identity both traffic or parking violation
- *unsafe areas*: areas with an high rate of violations

1.3.2 Acronyms

Table with all acronyms used in document.

ACRONYM	COMPLETE NAME
RASD	Requirements Analysis and Specification Document
GPS	Global Positioning Systems
S2B	Software To Be
GDPR	General Data Protection Regulation
FC	Fiscal Code
UC	Use Case

1.3.3 Abbreviations

- **Gn**: n-th Goal
- **Rn**: n-th Requirement
- **Dn**: n-th Domain Assumption
- **Cn**: n-th Constraint
- **UCn**: n-th Use Case

1.4 Revision History

1.5 Reference documents

- ISO/IEC/IEEE 29148: https://www.iso.org/standard/45171.html
- Specification Document: "SafeStreets Mandatory Project Assignement"
- Diagrams: https://www.draw.io/
- Mockups: https://www.figma.com/
- Alloy Official Documentation: http://alloy.lcs.mit.edu/alloy/documentation.html
- Alloy code highlithing for Latex: https://github.com/Angtrim/alloy-latex-highlighting

1.6 Document Structure

- **Chapter 2**: Presents an overall description of the *System* explaining in more datailed way Phenomena described in chapter 1. Provides some diagrams usefull to understand key aspects and general behavior of the *System* and possible type of *Users* with respective functions that they are allowed to do. This chapter is also focused on defining functional requirements such as constraints, domain assumption and dependencies that will be covered later.

- **Chapter 3**: This chapter is intended for developers, dives deeper on the aspects of chapter 2 using use cases and sequence diagrams in order to clarify process and interaction between *Users* and *System*. Describe the interfaces for the application, focusing on *System*'s design constraints and software *System* attributes.

- **Chapter 4**: Uses Alloy to generate a Formal Model for the application.

2 Overall Description

2.1 Product perspective

This section aims to explain in more detail the World, Machine and Shared Phenomena described in the previous Chapter.

2.1.1 World Phenomena

- **Citizen sees a parking violation and wants to report it**: While the *Citizen* is quietly walking, he sees a parking violations like a double parking or a car parked in the middle of bike lane and wants to report it.
- **Users want to know abount some violations that have been occurred**: An *User* has the needs to check some statistics about parking violations on a certain area for some purpose.
- **A parking violation occurs**: Someone in the city decides to not follow parking rules and doesn't park his car in a proper way.

2.1.2 Machine Phenomena

- **Storing permanently collected data**: The *System* needs to store, in a secure way, all the data submitted. In order to achieve this purpose and guarantee the best service the *System* needs to use a DBMS.
- **Encryption of sensitive data**: Personal *User's* data and all the data relative to the violations that can only be seen by *Authorities* need to be encrypted in order to proctect it from non-allowed third parties.
- **Retrieving data for a request**: *System* has to fullfill the data request from the *Users*. Data requests can be of two types, a *Citizen* request to see statistics of a certain city area or data request by *Authorities* who want to receive the violation reports collected by *SafeStreets* or statistics about unsafe city areas and vehicles.

2.1.3 Shared Phenomena

Controlled by the World observed by the Machine

- **A Citizen reports a violation**: Situation in which a *Citizen* spots a generic violation and wants to report it through the application. Using *SafeStreets* he can take the photo of the violation.
- **User can enter data for registration/login**: A *User* decide to use the application and provides his personal data in order to register if it's the first time he use the app, or to identify himself.
- **Users can request data**: In this phenomena we make a distinction between *Citizen* and *Authorities*. A *Citizen* may want to see violation statistics of a certain area, *Authorities* can request violation statistics and most egregious offender's vehicles statistics.

Controlled by the Machine observed by the World

- **Track position of the violation**: The *System* can retrieve the position where the violation occurred by fetching it from GPS service.
- **Mark areas with an high rate of violations**: Once some violations have been occured, the *System* mines the information that it has in order to highlight the areas with the highest frequency of violations.
- **System can fullfill data requests**: After processing a request, the *System* will show to the *User* the result of the DBMS query in a proper way.

2.1.4 Class Diagram

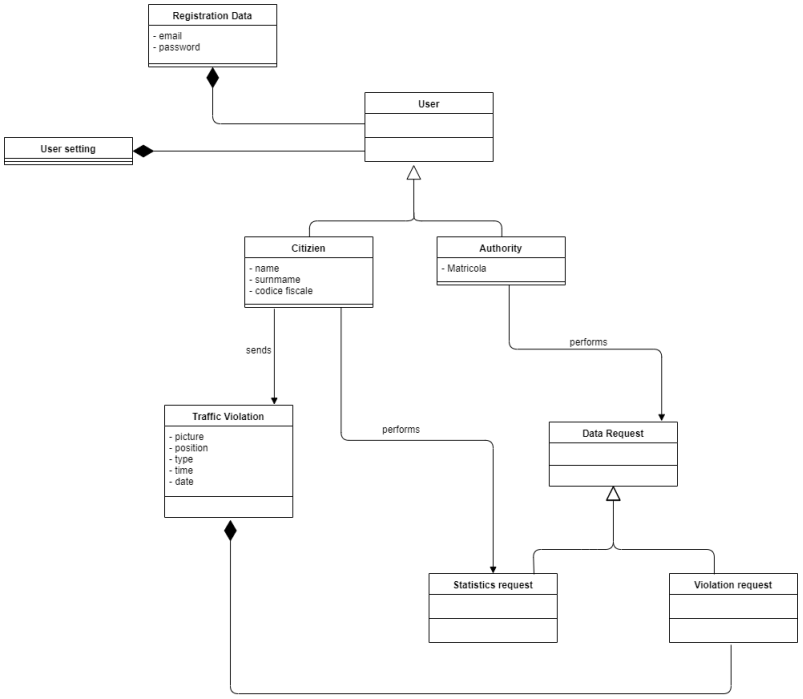


Figure 1: *SafeStreets'* Class diagram

2.1.5 State Charts

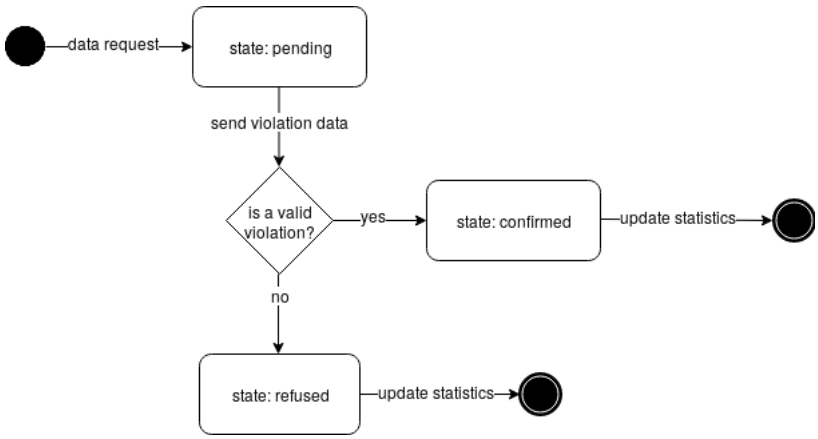


Figure 2: *Authority* requests for violations state chart

Figure 3: *Users* request statistics state chart

2.2 Product functions

In this section are explained the functions associated to *User*.

- **Citizen functions**:

  Report a violation
  When a *Citizen* sees a parking violation, he takes a picture of the vehicle paying attenction to focus on the license plate, inputs the type of the violation and sends it. The *System* will provide to add the position retrieving from GPS, to add the right time and date and to add the license plate obtained through the algorithm and confirmed by the *Citizen*.

  Retrieve statistics about unsafe areas
  Safestreets enables *Citizen* to visualize statistics about *unsafe areas*. *SafeStreets* mines the informations it has and let the *Citizen* retrieves the result through a clear interface containing significant plots, tables and charts.

- **Authority functions**:

  Retrieve statistics about unsafe areas
  Safestreets enables *Authority* to visualize statistics about unsafe areas. *SafeStreets* mines the informations it has and let the *Authority* retrieves the result through a clear interface containing significant plots, tables and charts.

  Retrieve statistics about vehicles

Safestreets enables *Authority* to visualize statistics about vehicles. *SafeStreets* mines the informations it has and let the *Authority* retrieves the result through a clear interface containing significant plots, tables and charts about most egregious offenders.

Request violations data for traffic tickets
SafeStreets enables *Authority* to retrieve all the parking violations sent by *Citizens*. For each parking violation *Authority* can accepts it or declines it. In the first case he can generates traffic ticket, in the second case he discards the informations about the parking violations. In both cases *SafeStreets* records response in order to build statistics.

2.3 User characteristics

Below we describe the convention used to identify the *Users* of the application and the function that those *Users* are allow to perform.

- **Guest**: A *User* that have donwloaded the application but is not registered yet. This type of *User* is not allowed to access the application functionalities.
- **Citizen**: is a generic *User* app not related to *Authorities*, a common *Citizen* that wants to use the application. After the registration process, he can log in the application and use the functionalities such as report a violation or request informations about the statistics of a certain area.
- **Authority**: This *User* is associated to the local municipal police district, any traffic warden, once registered with is matricola number and logged in have full access to statistics, both violations and vehicles, and can request all the violations reported from *Citizens* in order to generate traffic tickets.
- **User**: can be both a *Citizen* or *Authority* type, in this document this name is used when it's not necessary make a distinction between the two.

2.4 Assumption and Dependencies Constraints

2.4.1 Domain Assumption

The following list present all the domain assumption made.

- [**D1**]: *Users* can't make more than one account.
- [**D2**]: The personal informations provided by *User* are valid and belongs to the him.
- [**D3**]: Position data as an accuracy of 10 meters.
- [**D4**]: The *System* can access internet whenewer needs it.
- [**D5**]: Permission to access GPS data is always allowed.
- [**D6**]: Permission to take a photo is always allowed.

2.4.2 Dependencies

This list below represent all the dependencies that S2B need in order to work properly.

- Smartphone needs an internet connection.
- Smartphone needs a Photocamera.
- Smartphone needs a GPS system.
- *SafeStreets* needs a trusted external storage for violations data and personal data.

2.4.3 Constraints

- The S2B must guarantee the European data protection GDPR for *User's* sensitive data.
- The S2B will be used only in Italy due to personal data type like (fiscal code and police matricola).
- The S2B will be developed as a smartphone application.
- The *Citizen* can only take photos from the application.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

Login or register page This is the first page that *Users* see after downloading and installing *SafeStreets* application. Both *Authorities* and *Citizens* can log in from this page without distinction because they have to provide only email & password. If *User* hasn't been registered yet in *SafeStreets* can go to regisiter page by clicking on register button and the *System* will show the default register for *Citizen*.

Figure 4: Login or Register page

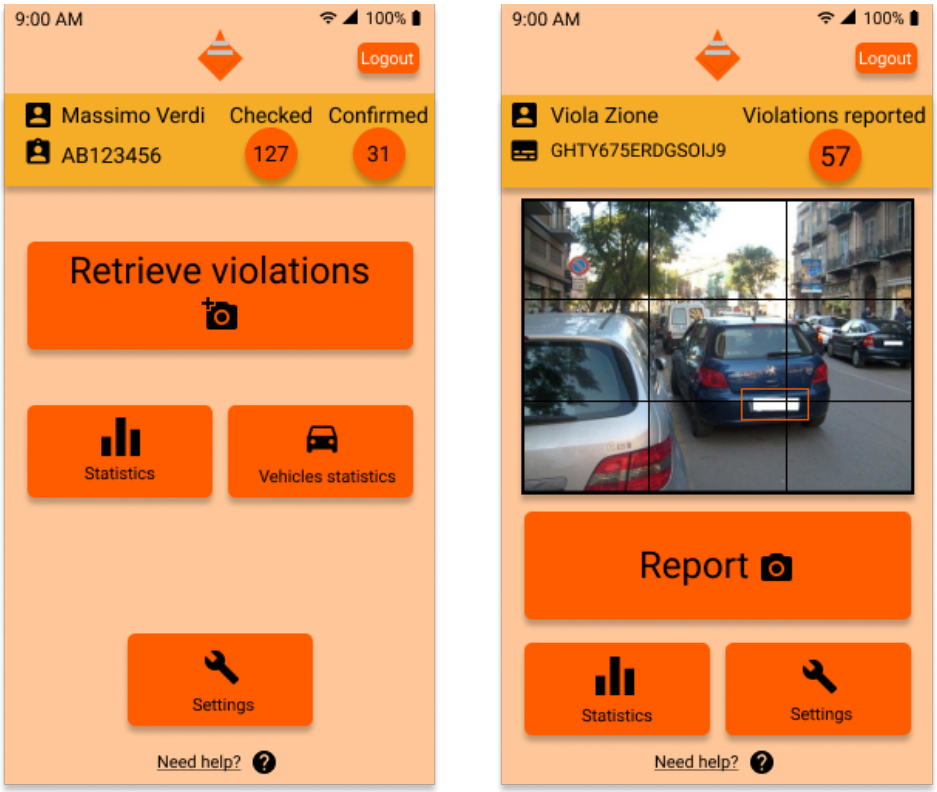
Registration page Registration pages ask *Guests* to input name, surname, email and a password. If the *Guest* is a *Citizen* he must also input his Fiscal Code otherwise if he is an *Authority* he must input his Matricola. The default page that the *System* shows when the register button is clicked is the *Citizen* registration page. If the *Guest* wants to register him as *Authority* he must click the "Register as *Authority*" button. From this page it is possible to return in the *Citizen* registration page by clicking the "Register as *Citizen*" button.

(a) Register for *Authorities* (b) Register for *Citizens*

Figure 5: Registration pages

Home pages

- **Citizen**: Home page shows a bar at the top of screen with some data such as name, surname, FC and the number of violations reported. In the center of screen *System* shows the open photocamera ready to take a picture by clicking the report button. It is possible to take a picture only if a license plate is framed with the photocamera. Once the report button is clicked, a picture is taken and the *Citizen* is redirected to the *Citizen* report info page. The two button at the bottom allow *Citizen* to access statistics and account's settings.
- **Authority**: Home page shows a bar at the top of screen with some data such as name, surname, Matricola, the number of violations checked and the number of violation confirmed. In the center of screen there are 3 buttons: Retrieve Violation, Statistics, Vehicles statistics. The first one allows *Authority* to access *Authority* report info page, the second one allows to access violation statistics and the last one allows to access vehicles statistics. It is also present the settings button to access account's settings.

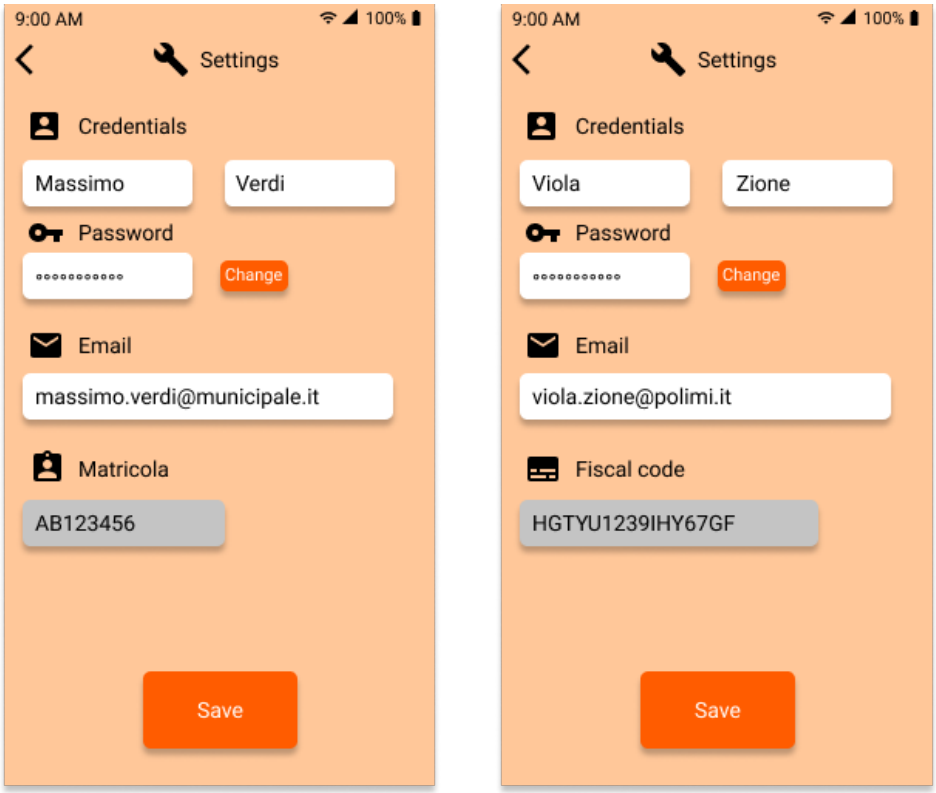


(a) Home page for *Authorities*

(b) Home page for *Citizens*

Figure 6: Home pages

Settings This two pages below represents the settings page in which the *Citizen* and *Authority* can change their personal data or visualize it. Only some informations can be modified, those who can't be modified are showed with grey color.



(a) Settings for *Authorities* (b) Settings for *Citizens*

Figure 7: Settings Pages

Statistics for Citizens In the page below we can see usefull statistics for *Citizen*, there are 3 graphs that summarize all the interesting informations. The first one shows the violations during a specific year, that can be changed in the filters below. The button 'type' can filter the violation's type to show only the type of interest, like 'double-parked' for instance. Those statistics are dislpayed with the violations number per month in the position selected. Below the map displays a more generic view of the violations in a city by highlithing the zone with a color graduation that represent the most dangerous zone. The more darker the more violations occur in that area, by changing the position in the map by clicking in a certain point the graph above will update the statistics for that area. In the last gaph there is a perspective of the violations reported by *Citizen* during a certain year that can be changed.

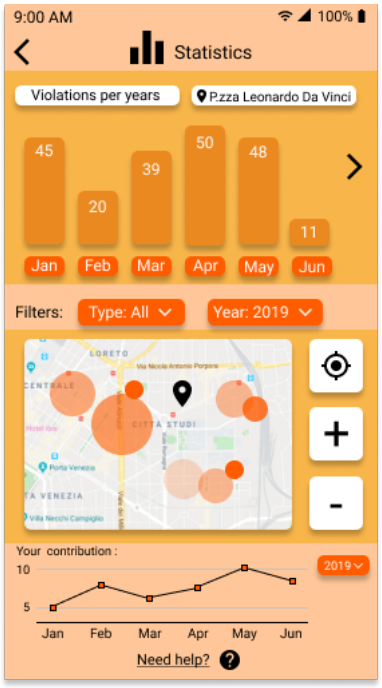


Figure 8: Statistics *Citizen*

Statistics for Authorities

- **Violations statistics**: The first two graph have the same function and are equale as the two for *Citizen*. The violations reported graph isn't present because *Authority* can't report a violation. In the bottom page there is a button for receive violation's statistics via mail.
- **Vehicles statistics**: In that page, that can only be seen by *Authority* are showed the statistics of the most egregious offender's vehicles. In the first part is possible to search for a specific license plate, and if the vehicle related has commited some violations the stats will be displayed below, divided per months and visualized by year, that can be changed. Below are listed the license plate of the most egregious offenders in a certain area selected at the top with the two buttons on the right *Authorities* can receive those data via specified mail. By clicking on a license plate the bottom graph will be updated with the data relative to that, divided per month and visualized by year, that can be changed with the button on the right.

(a) Violations Statistics (b) Vehicle Statistics

Figure 9: Statistics for *Authorities*

Violation's report Citizen Below we can see the page that is displayed when a *Citizen* clicks on the report button taking a photo of a violation. In this page is showed the photo taken and some metadata retrivied automatically by the *System* like date, time, position and the license plate that is read by the algorithm. *Citizen* can change the license plate if the algorithm doesn't read it properly he can also choose the type of violation. The confirm button, if no error occurs, will send the data collected to the *System*.

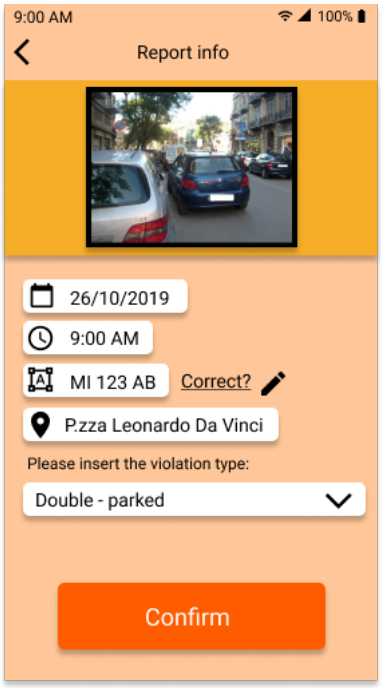


Figure 10: Page *Citizen*

Violation's check page for Authority In this page showed below the *Authority* can confirm the violations reported by *Citizen* and they can generate traffic tickets from the data. In the page are showed the photo and the data necessary for the traffic ticket. The *Authority* can also receive this data by clicking on the button in the bottom.

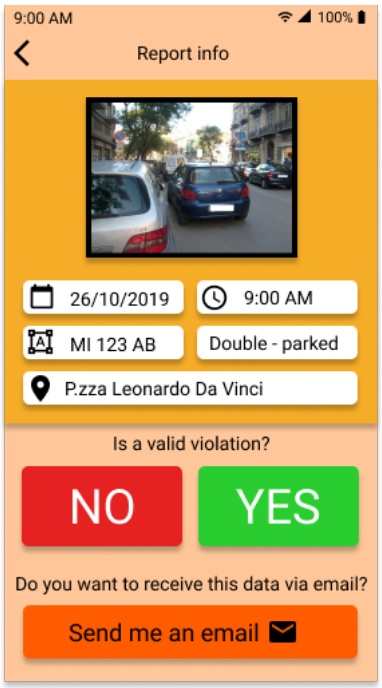


Figure 11: Page *Authority*

3.1.2 Hardware Interfaces

The *System* does not offer any Hardware Interfaces

3.1.3 Software Interfaces

As mobile applications, the main software interfaces are:

- iOs
- Android

3.1.4 Communication Interfaces

HTTPS protocol: to safely communicate through the internet

3.2 Functional Requirements

3.2.1 Use Case Diagrams

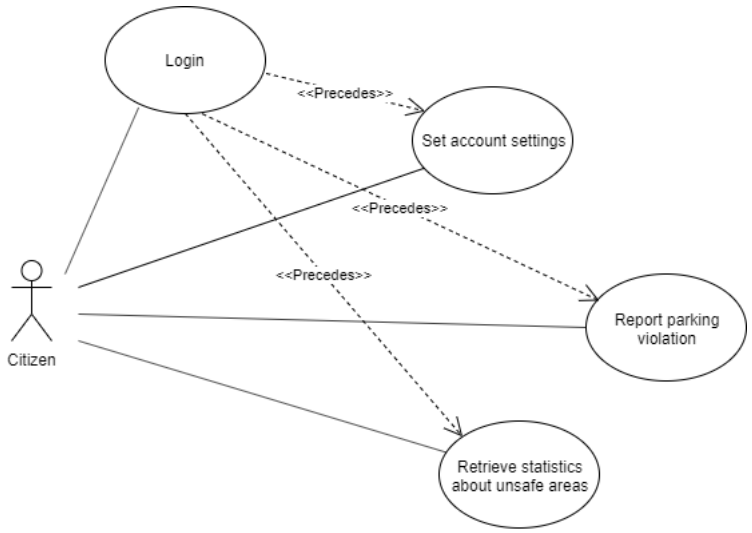


Figure 12: *Citizen* Use Case Diagram

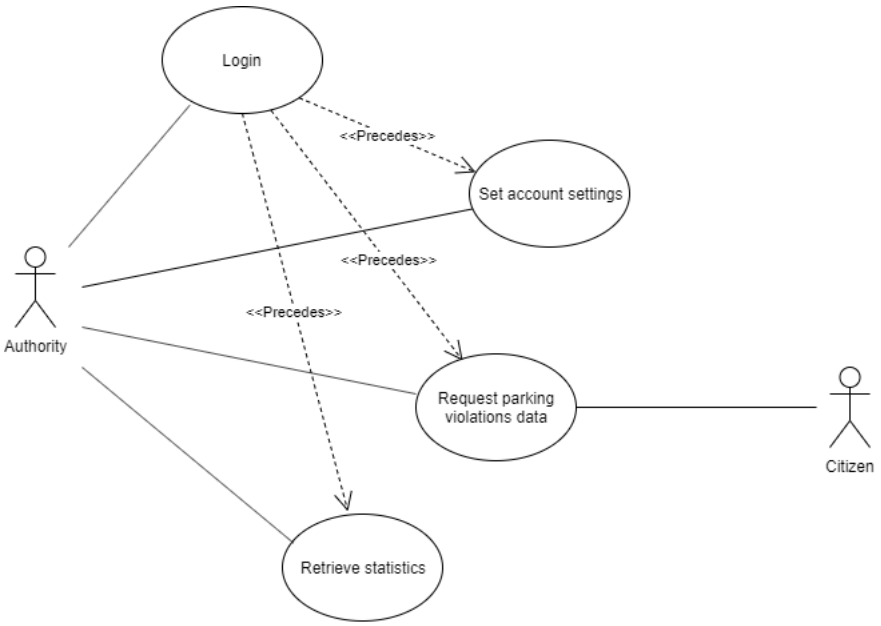


Figure 13: *Authority* Use Case Diagram

3.2.2 Scenarios

- **Scenario 1**: Luca is walking towards work when outside, in the street of his house, he sees that many cars are parked badly. Some of these are parked near the strips, obstructing the view of the pedestrian who must cross the strips. Luca tired of the situation, that endangers him and many other *Citizens*, decides to report the incident. With his smartphone he opens *SafeStreets* application and after logging in, clicks on the report button to report the fact. He takes the photo by clicking on the report button in the application's home, then he receives the data retrieved, the plate is correctly recognized and, after inserted the type of violation, Luca clicks on confirm button.

- **Scenario 2**: Andrea is a disabled boy, he is perfectly able to drive the car but it is difficult for him to walk for long distances. The area in which he works is very busy and it is very rare to find parking nearby, fortunately there are parking spaces reserved for disabled people near the entrance of the building. One morning he finds the place occupied and looking better the machine parked he notice that lack of the certificate necessary for parking in the places reserved for the disabled. Thanks to *SafeStreets* after logging in as a *Citizen*, Andrea can report this violation directly to the *Authorities*. Andrea can now take a photo directly from the application's home by clicking on report button. *SafeStreets* retrieves information such as location and a timestamp with date and time. The application tries to recognize the license plate from the photos and shows the results to Andrea, who after confirming the correctness can click on confirm button and officially send the violation.

- **Scenario 3**: The command of the municipal of the municipality of Milan wants to optimize his patrols, aiming at the most problematic areas of the city. This targeted surveillance is essential and would bring significant benefits including:

a potential reduction of violations in these areas and reduction of unnecessary patrols in areas with fewer violations. Fortunately, having joined the *SafeStreets* initiative, thanks to the contribution of *Citizens*, they can use the application to receive these statistics directly from smartphones. After having registered as an *Authority* and logged in, they can access the violations statistics. From this page they can see not only a map with a general perspective of the areas but also check for a specific location by moving the pointer on the map.

- **Scenario 4**: Maurizio is a young policeman from the city of Milan. He loves putting a lot of passion into his work and to do so he often learns about new technologies. After downloading *SafeStreets* and registering as an *Authority*, he immediately takes an interest in the function to generate traffic tickets thanks to the reports made by *Users*. From the home of *SafeStreets* Maurizio clicks on retrieve violations button, then the application starts showing to him some violations, once a time with a photo and the related data. Maurizio then needs only to analyze the photo and check if it's a valid violations or not. Once he decided he can generate a ticket for that violation and then confirming by clicking on yes/no button. Every answer provided allow *SafeStreets* to update statistics and give more precise information to *Users*.

3.2.3 Use Cases

ID	UC1
Description	A *Guest* creates a *Citizen* account
Actors	*Guest*
Precondition	*Guest*'s smartphone satisfies hardware limitations *Guest* has downloaded the app from the store *Guest* has not an account
Flow of events	1. *Guest* opens the app 2. *Guest* clicks the registration button 3. *System* shows the *Citizen* registration form 4. *Guest* fills the form with his personal data plus mail and password 5. *System* checks the validity of the data inserted 6. *System* sends confirmation email 7. *Guest* receives the email and clicks the URL to complete the registration
Postconditions	*System* has stored a new *Citizen* account *Guest* can login as *Citizen*
Exceptions	*Guest* inserts an email that has been used by another account *Guest* inserts a FC that has been inserted by another account *Guest* inserts an invalid FC In these case *System* shows *User* an error message and the flow of events restart from point 3

Table 1: *Guest* creates a *Citizen* account

ID	UC2
Description	A *Guest* creates an *Authority* account
Actors	*Guest*
Precondition	*Authority*'s smartphone satisfies hardware limitations *Guest* has downloaded the app from the store *Guest* has not an account
Flow of events	1. *Guest* opens the app 2. *Guest* clicks the registration button 3. *System* shows the *Citizen* registration form 4. *Guest* clicks on register for *Authority* button 5. *System* shows the *Authority* registration form 6. *Guest* fills the form with his personal data plus Matricola, mail and password 7. *System* checks the validity of the data inserted 8. *System* sends confirmation email 9. *Guest* receives the email and clicks the URL to complete the registration
Postconditions	*System* has stored a new *Authority* account *Guest* can login as *Authority*
Exceptions	*Guest* inserts an email that has been used by another account *Guest* inserts a Matricola that has been inserted by another account *Guest* inserts an invalid Matricola In these case *System* shows *User* an error message and the flow of events restart from point 5

Table 2: *Guest* creates an *Authority* account

ID	UC3
Description	A *User* logs in
Actors	*Citizen*, *Authority*
Precondition	*User* has already created the account
Flow of events	1. *User* opens the app 2. *System* shows login/register interface 3. *User* inputs his credentials 4. *User* clicks login button 5. *System* checks the validity of the data inserted
Postconditions	*User* can use properly the app
Exceptions	*User* inserts wrong credentials In this case *System* shows *User* an error message and the flow of events restart from point 2

Table 3: *User* login

ID	UC4
Description	A *Citizen* reports a parking violation
Actors	*Citizen*
Precondition	*Citizen* has already logged in
Flow of events	1. *System* opens the photocamera 2. *Citizen* clicks the report button 3. *System* shows the report info page for *Citizens* 4. *Citizen* inputs the type of violation 5. *Citizen* checks the correctness of the license plate 6. *Citizen* clicks the send button
Postconditions	*System*'s DB stores the violation
Exceptions	*Citizen* takes a bad picture In this case *System* discards the picture and the flow of events restart from point 1

Table 4: *Citizen* reports a parking violation

ID	UC5
Description	An *Authority* retrieves a legitimate parking violation
Actors	*Authority*
Precondition	*Authority* has already logged in
Flow of events	1. *Authority* clicks the retrieve button 2. *System* shows the report info page for *Authorities* 3. *Authority* checks that it is a real parking violations 4. *Authority* clicks the YES button
Postconditions	*Authority* generates a traffic ticket and *System* uploads statistics
Exceptions	

Table 5: Legitimate parking violation retrieved by *Authority*

ID	UC6
Description	A *Authority* retrieves a wrong parking violation
Actors	*Authority*
Precondition	*Authority* has already logged in
Flow of events	1. *Authority* clicks the retrieve button 2. *System* shows the report info page for *Authorities* 3. *Authority* checks that it is not a real parking violations 4. *Authority* clicks the NO button
Postconditions	*Authority* discards the picture and *System* uploads statistics
Exceptions	

Table 6: Wrong parking violation retrieved by *Authority*

ID	UC7
Description	A *User* retrieves statistics
Actors	*Authority*, *Citizen*
Precondition	*User* has already logged in
Flow of events	1. *User* clicks the retrieve statistics button 2. *System* shows summary of statistics
Postconditions	*User* increases his knowledge about parking violations of his city
Exceptions	

Table 7: statistics retrieved by *User*

3.2.4 Sequence Diagrams

Login The following diagram shows how a generic *User* can login into the application. The actors involved are both *Citizen* and *Authority*.

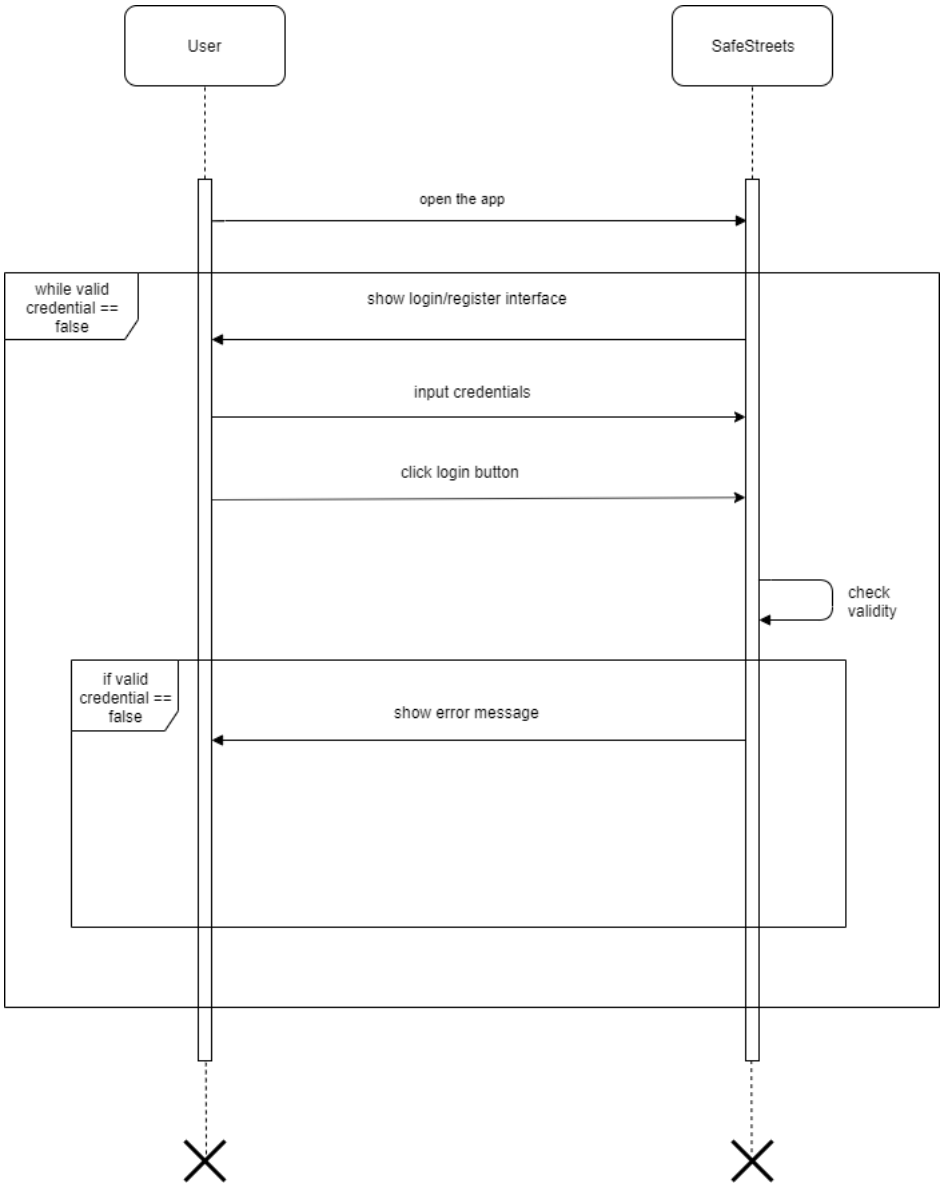


Figure 14: Login

Register Citizen This sequence diagram shows how registration process occur in *Citizen* case, the first steps are mandatory to reach the register page, then until the *User* inserts valid credentials the *System* don't allow him to go next.

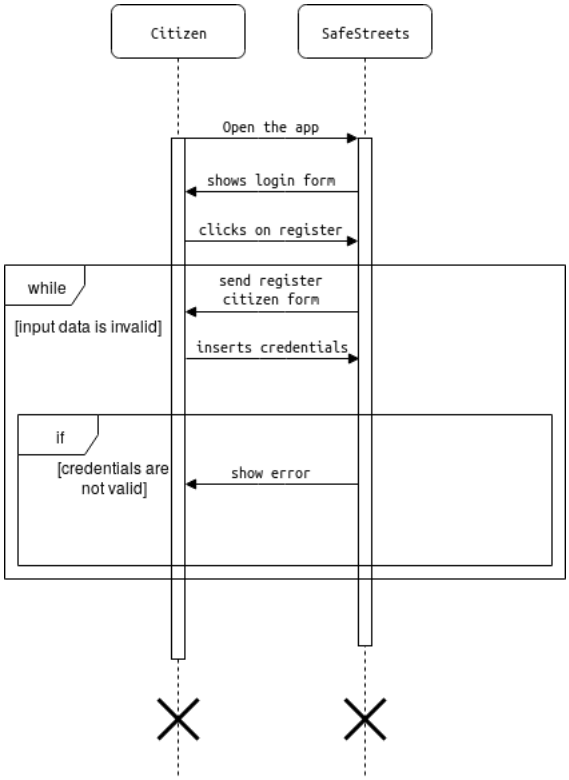


Figure 15: Register for *Citizen*

Register Authority Thi sequence diagram represents the same case as above but referred to *Authorities*, so is necessary another step in order to reach the regster form. Then like in the previous case it's possible to complete correctly the registration only by inserting valid credentials.

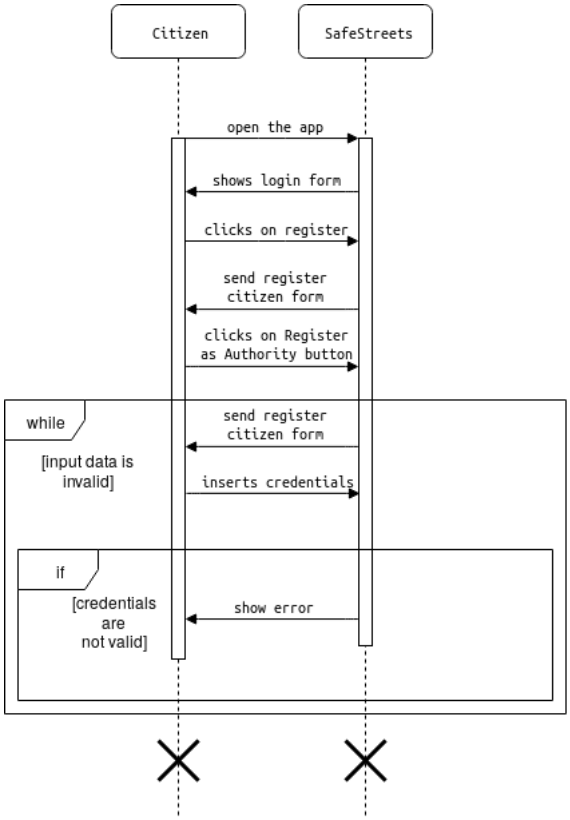


Figure 16: Register for *Authority*

Retrieve Violation The following diagram shows how an *Authority* can retrieve violations. Two cases are considered: in the first case the violation is legitimate, he accepts it and generates traffic ticket. In the second case the violation is wrong so he discards it.

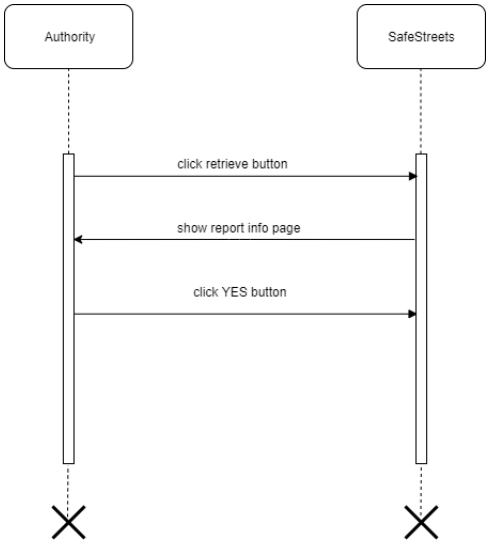


Figure 17: Accept retrieve violation

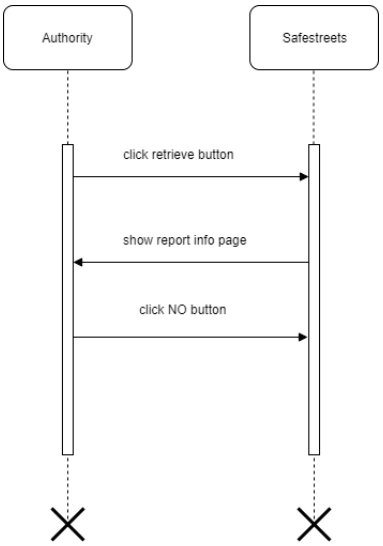


Figure 18: Discard retrieve violation

Report violation The following diagram shows how a *Citizen* can report a violation. After clicking on report the *System* receives data, checks the correctness and tries to recognize the license plate. After *Citizen* confirmation the report is officially sended.

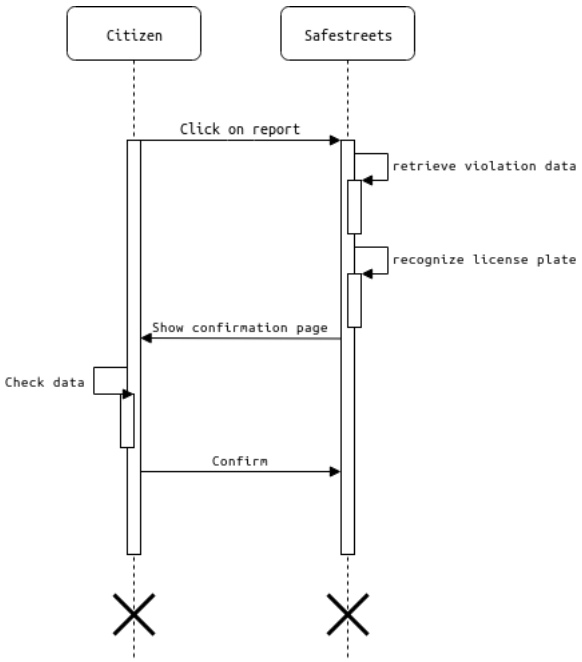


Figure 19: Report violation

Retrieve statistics The following diagram shows how a generic *User* can retrieve statistics. The actors involved are both *Citizen* and *Authority*.

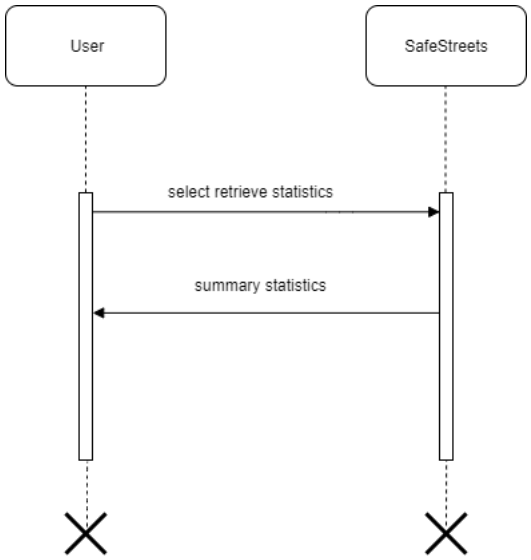


Figure 20: Retrieve statistics

3.2.5 Goal Mapping on Requirements

For each goal now will be described below the requirements:

- [**G1**]: Allow *Guests* to be registered as a *Citizen* or as *Authority*;
  - [**R1**]: Account can be created if and only if *User* provides unique email and password.
  - [**R2**]: The *System* allow *Guest* to create *Citizen* or *Authority* account.
  - [**D1**]: *Users* can't make more than one account.
  - [**D2**]: The personal informations provided by *User* are valid and belongs to him.
- [**G2**]: Allow *Citizens* to report parking violations;
  - [**R3**]: The *Citizen* has to take the violation's photo with the application.
  - [**R4**]: The *System* allows *Citizen* to input some violation's data.
  - [**R5**]: The photo taken must be recognizable by the *System*.
  - [**R6**]: The *Citizen* has to be able to discard the photo taken.
  - [**R7**]: The *System* has to be able to attach the correct date, time and position to the report.
  - [**R8**]: The *Citizen* can't change date, time and position in the report.
  - [**D3**]: Position data has an accuracy of 10 meters.
  - [**D4**]: The *System* can access internet whenewer needs it.
  - [**D5**]: Permission to access GPS data is always allowed.
  - [**D6**]: Permission to take a photo is always allowed.

- **[G3]**: *Citizen* has to be able to input information about the violation that he has reported;
  - **[R9]**: *Citizen* can change the license plate if it isn't recognised properly.
  - **[R10]**: *Citizen* has to be able to choose the correct type of violation.
- **[G4]**: Must provide a visualization of the areas with high frequency of violations to *Users*;
  - **[R11]**: *Users* can change the area of visualization.
  - **[R12]**: *Users* can change the date of visualization.
  - **[D3]**: Position data as an accuracy of 10 meters.
  - **[D4]**: The *System* can access internet whenewer needs it.
  - **[D5]**: Permission to access GPS data is always allowed.
- **[G5]**: Must provide a visualization of vehicles that commit the most violations to *Authorities*;
  - **[R13]**: *Authority* can search for a specific license plate.
  - **[R14]**: *Users* can change the date of visualization.
  - **[D4]**: The *System* can access internet whenewer needs it.
  - **[D5]**: Permission to access GPS data is always allowed.
- **[G6]**: Must ensure the chain of custody of the information sent by *Citizens*;
  - **[R15]**: Violations sent must be digitally signed and hashed.
  - **[R16]**: The *System* must use HTTPS to safely communicate.
  - **[D4]**: The *System* can access internet whenewer needs it.
- **[G7]**: *Authorities* can retrieve traffic violations in order to generate traffic tickets;
  - **[R17]**: The violation retrieved can only be seen by the *Authority* that retrieves it.
  - **[D4]**: The *System* can access internet whenewer needs it.
- **[G8]**: *System* must build statistics with the informations about issued tickets;
  - **[R18]**: The *System* must update the statistics with the most recent data.

3.3 Performance Requirements

In this section we discuss requirements for what regards performance. The *System* must be able to support up to 5 million of registered *Users*. This limitation is not posed by the front-end of the *System*, but rather by the back-end part, specifically the DB. For the same reasons it must be able to handle up to 10 million of parking violations sent by the *Citizen*. In order to avoid any kind of saturation, every parking violation that has not been taken into account by any *Authority* for 30 days, must be automatically discarded. This operation does not update the information about statistics.
Requests about statistics shall be processed in less than 5 seconds. Requests about parking violations, instead, shall be processed in less than 1 second.

3.4 Design Constraints

3.4.1 Standards compliance

The S2B will use certain measures as:

- Standard longitude and latitude measures for the position.

For what concerns the privacy, the S2B is subject to GDPR, a regulation in EU law on data protection and privacy for all individual *Citizens* of UE.

3.4.2 Hardware limitations

In order to work properly the application must rely on hardwares that have certain requirements such as:

- GPS
- internet connection (4G/3G/2G)
- Photocamera with a minimum precision of 5Mp

3.5 Software System Attributes

3.5.1 Reliability

The *System* must be able to run continuously without any interruptions. In order to do that, it must be ensured that the *System* is fault tolerant. To prevent downtime, one of the main goals of architecture design must be ensuring graceful degradation of the *System*.

3.5.2 Availability

SafeStreets does not present any critical functions so 99% availability with 3.65 days/year as downtime should be good.

3.5.3 Security

Security is a key aspect of *SafeStreets* because it is very important that the informations are never altered. The S2B must:

1. use HTTPS to safely communicate with the Server and DBMS.
2. Hash the passwords so that they are not stored in clear in the DB.
3. Encrypt sensitive data before storing it.
4. digital sign the parking violations sent by *Citizens* and then hash them.

3.5.4 Maintainability

In order to achieve maintainability some good practices must be followed to reduce coupling and avoid code duplication. Modularity is also necessary in order to make the code more robust and to make easier adding new functionalities.

3.5.5 Portability

S2B, as it stated previously, will work both in Andorid and iOS and this ensures itself portability. For the back-end part, it should be OS independent.

4 Formal Analysis with Alloy

4.1 A description of The Alloy Model

In this section we will focus on the relevant part of the application described using alloy model language. We decided to focus the model on the interaction between three main objects of *SafeStreets* that are: *Citizen*, *Authority* and Reports. Those interactions are foundamental in order to reach the application's goals. The model will presents an analysis of the main functions such as:

- A *Citizen* sends a report.
- An *Authority* confirms a report.
- An *Authority* discards a report.

Those functions are expressed by using different predicates. In the description of the model we made some semplification in order to make the model cleaner and more readable. First we have reduced the range of the position's attributes (i.e. latitude and longitude) by scaling them and some non-relevant attributes of objects are omited.

4.2 Alloy Model

```
sig FiscalCode {}
sig Matricola {}
sig Email {}
sig Password {}

sig Registration {
    email: one Email,
    password: one Password
}

abstract sig User {
    registration: one Registration
}

sig Citizen extends User {
    fiscalCode: one FiscalCode,
    reportsSended: set Report
}

sig Authority extends User {
    matricola: one Matricola,
    reportsChecked: set Report
}

sig Location {
    latitude: one Int,
    longitude: one Int
} {latitude ≥ -3 and latitude ≤ 3 and longitude ≥ -6 and longitude ≤ 6 }

abstract sig Status {}
one sig Pending extends Status {} --if no Authority checks this report
one sig Yes extends Status {} --if it's evaluated as an effective violation
one sig No extends Status {} --if it isn't evaluated as an effective violation

sig Date {}
sig Time {}
sig License {}
sig Type {}

abstract sig Report {
    location: one Location,
    date: one Date,
    time: one Time,
    license: one License,
    type: one Type,
    status: one Status,
    sender: one Citizen,
    checker: one Authority
}

-------------------- FACTS --------------------

--A report send from a citizen must be in is report sended set
fact EqualityCitizen {
    all r: Report | some c: Citizen| r.sender = c iff r in c.reportsSended
}

--A report checked by an authority must be in is report checked set
fact EqualityAuthority {
    all r: Report | some a: Authority | r.checker = a iff r in a.reportsChecked
}

--No reports with state pending in some authority's checked set
fact NoAuthorityWithPendingReportsInChecked {
    no r: Report | some a: Authority |
        r in a.reportsChecked and r.status = Pending and r.checker = a
}

--If a report is checked must be in one authority checked set
fact ReportInCheckedSet {
    all r: Report | some a: Authority |
        r.status ≠ Pending implies (r in a.reportsChecked and r.checker = a)
}
```

```
--If a report exists must be sended by a citizen
fact ReportInSendedSet {
    all r: Report | some c: Citizen | r.sender = c and r in c.reportsSended
}

--All fiscal code have to be associated to Citizen
fact FiscalCodeCitizen {
    all fc: FiscalCode | some c: Citizen | fc in c.fiscalCode
}

--All matricola have to be associated to Authority
fact MatricolaAuthority {
    all m: Matricola | some a: Authority | m in a.matricola
}

--All registration have to be associated to User
fact RegistrationUser {
  all r: Registration | some u: User | r in u.registration
}

--All email have to be associated to a User Registration
fact EmailRegistration {
    all e : Email | some u: User | e in u.registration.email
}

--All password have to be associated to a User Registration
fact PassRegistration {
    all p : Password | some u: User| p in u.registration.password
}

--Every User has different email
fact NoSameEmail {
    no disj u1, u2 :User | u1.registration.email = u2.registration.email
}

--Every Citizen has different FiscalCode
fact NoSameFiscalCode {
    no disj c1, c2 : Citizen | c1.fiscalCode = c2.fiscalCode
}

--Every Authority has different Matricola
fact NoSameAuthority {
    no disj a1, a2 : Authority | a1.matricola = a2.matricola
}

--If a report is evaluated than the status isn't pending
fact ReportInAuthoritySetsMustBeChecked {
    some r: Report | one a: Authority | r in a.reportsChecked
                                and ((r.status = Yes)
                                        or
                                    (r.status = No))
}

--All the citizens can't have some equal report in their sets
fact OnlyDisjointedReportsSet {
    no r: Report | all c1,c2: Citizen |
    (r in c1.reportsSended) and (r in c2.reportsSended)
}

-------------------- ASSERTS --------------------

--If a report state is pending then it can't be in some authorities checked set
assert ReportCanOnlyBeEvaluatedByAuthority {
    some r: Report | one a: Authority | (r.status = Pending)
                                        implies
                                    (r not in a.reportsChecked)
}

-------------------- CHECKS --------------------
check ReportCanOnlyBeEvaluatedByAuthority

-------------------- PREDICATES --------------------

pred sendReport [c, c1: Citizen, r: Report] {
    c.registration = c1.registration
    c.fiscalCode = c1.fiscalCode
```

```
    r.status = Pending
    r.sender = c
    c1.reportsSended = c.reportsSended + r
}

pred confirmReport [r, r1: Report, a, a1: Authority] {
    a.matricola = a1.matricola
    r.status = Pending
    r1.sender = r.sender
    r1.status = Yes
    a1.reportsChecked = a.reportsChecked + r1
}

pred discardReport [r, r1: Report, a, a1: Authority] {
    a.matricola = a1.matricola
    r.status = Pending
    r1.sender = r.sender
    r1.status = No
    a1.reportsChecked = a.reportsChecked + r1
}

pred world1 {
    #Citizen > 1
    #Authority > 1
    #Report > 2
}

pred world2 {
    #Citizen = 3
    #Authority = 3
    #Report = 3
}

-------------------- RUNS --------------------

run sendReport for 4
run confirmReport for 5
run discardReport for 5
run world1 for 5
run world2 for 7
```

4.2.1 Model images

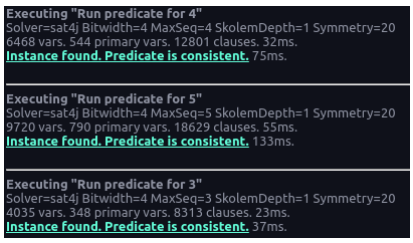


Figure 21: Report violation

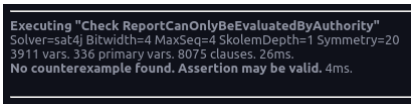


Figure 22: Report violation

5 Efforts