Politecnico di Milano

SAFESTREETS

POLITECNICO
MILANO 1863

Samuele Meta - Stiven Metaj

Supervisor: Matteo Rossi

Department of Computer Science and Engineering

November 10, 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, plate, date and time. At any time, the application allows the Citizen to visualize the history of the reported violations since it's regis-

tered. In case of error or incorrect information, the System allows the Citizen to revoke the alert produced. Furthermore, the System offers the possibility to sign up as an Authority, once filled the registration form and verified the institutional identity through the relative bureaucracy. *SafeStreets* will allow Authorities to have a complete overview of the incoming reports 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 username and a password.
- [**G.2**] The System allows the User to report a traffic violation by sending photos and relative information.
- [**G.3**] The System checks that the photos provided by the User have not been compromised.
- [**G.4**] The System allows the Authority to accept or refuse a violation.
- [**G.5**] The System allows the User to view his/her own history of reported violations.
- [**G.6**] The System allows the User to delete a reported violation.
- [**G.7**] The System allows the User to consult the created statistics according to his/her role.
- [**G.8**] The System allows the Authority to elaborate statistics with the help of the Municipality API.
- [**G.9**] The System allows the Authority to consult a list of suggested interventions.

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 split 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* tries 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 or an error message**: after a *Citizen* sends a violation occurrence the *System* shows a confirmation message if the photos updated by the *User* are not compromised; if they are compromised the *System* must show an error message.
- **The *System* asks to confirm the retrieved violation's street**: the application retrieves the street information from the GPS signal of the *User*'s device and asks him/her a confirmation in order to be sure to have the right violation's street information (if the *User* doesn't input it).
- **The *System* shows the list of reported violations by an *User***: an *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**: an *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 follow:

- **The *System* stores 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* computes 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* retrieves 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* computes a list of possible interventions in areas of interest**: a Neural Network is used on 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
GUI	Graphical User Interface
URL	Uniform Resource Locator

Table 1.1: *Acronyms*

1.3.3 Abbreviations

- [**G.n**]: n-th Goal
- [**D.n**]: n-th Domain Assumption
- [**R.n**]: 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 is sent.

A registered *User* tries 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, permit to retrieve the GPS signal position (writing however the right position if the GPS is wrong) and indicate the type of violation.
Furthermore the *System* runs an algorithm in order to retrieve the license plate of the vehicle in the pictures, so the *User* can help it writing the license plate (in that case the algorithm will verify it matching information retrieved in the pictures) or can leave the plate field blank (in that case the *System* will have only the pictures to retrieve the plate).

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 or an error message *(machine controlled, world observed)*
When an *User* sends 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, to 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 tries to retrieve it from the GPS signal of the *User*'s device (the *User* should have already given 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 an *User* is registered to the application and his/her profile is verified, there are no limits to the number of violations that can be reported. An evident feature that the application must have is 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 report more violations (adding more data to the *System* with the purpose of disciplining who commited 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

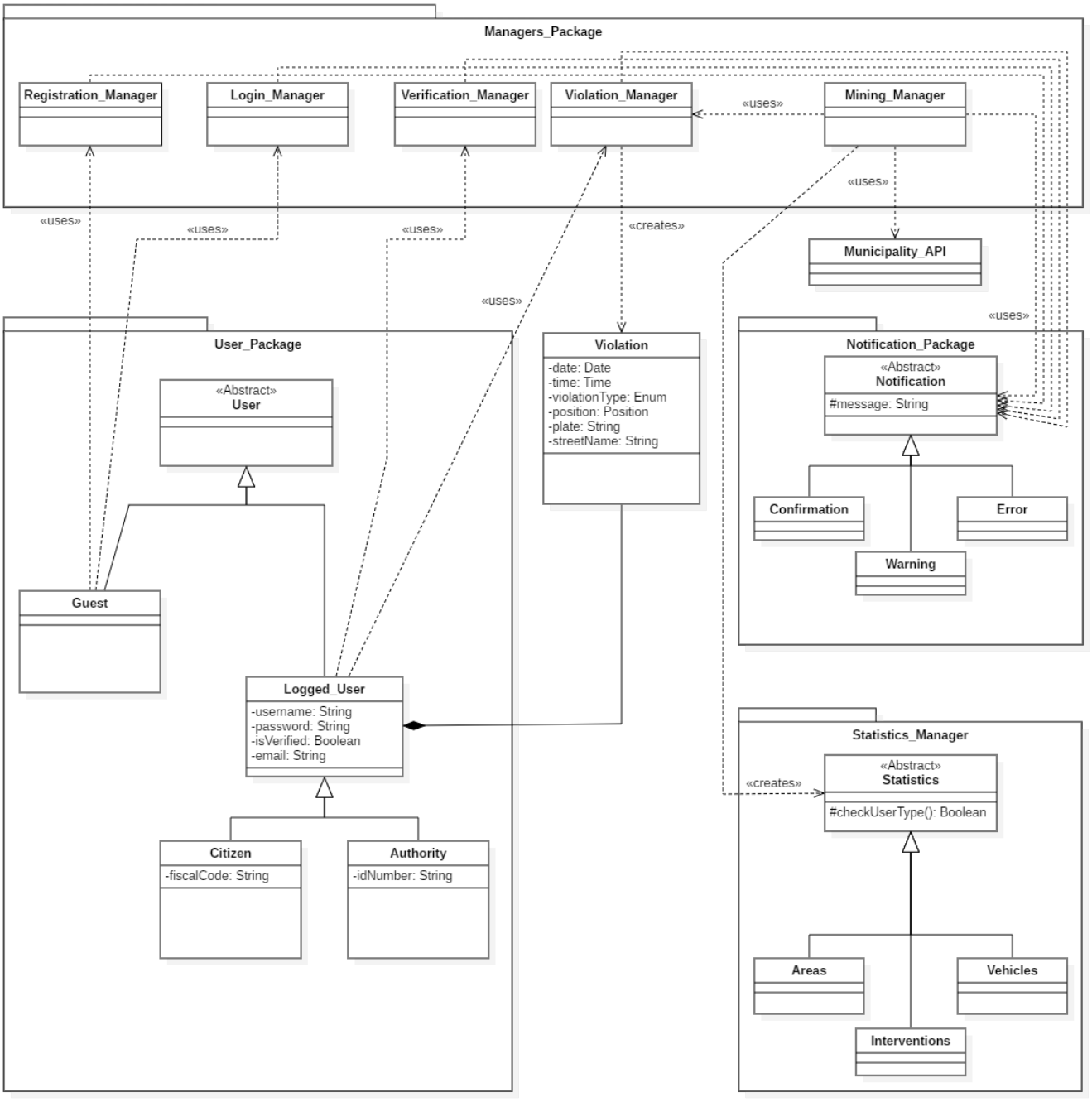


Figure 2.1: *SafeStreets Class Diagram*

2.1.2 State Chart Diagrams

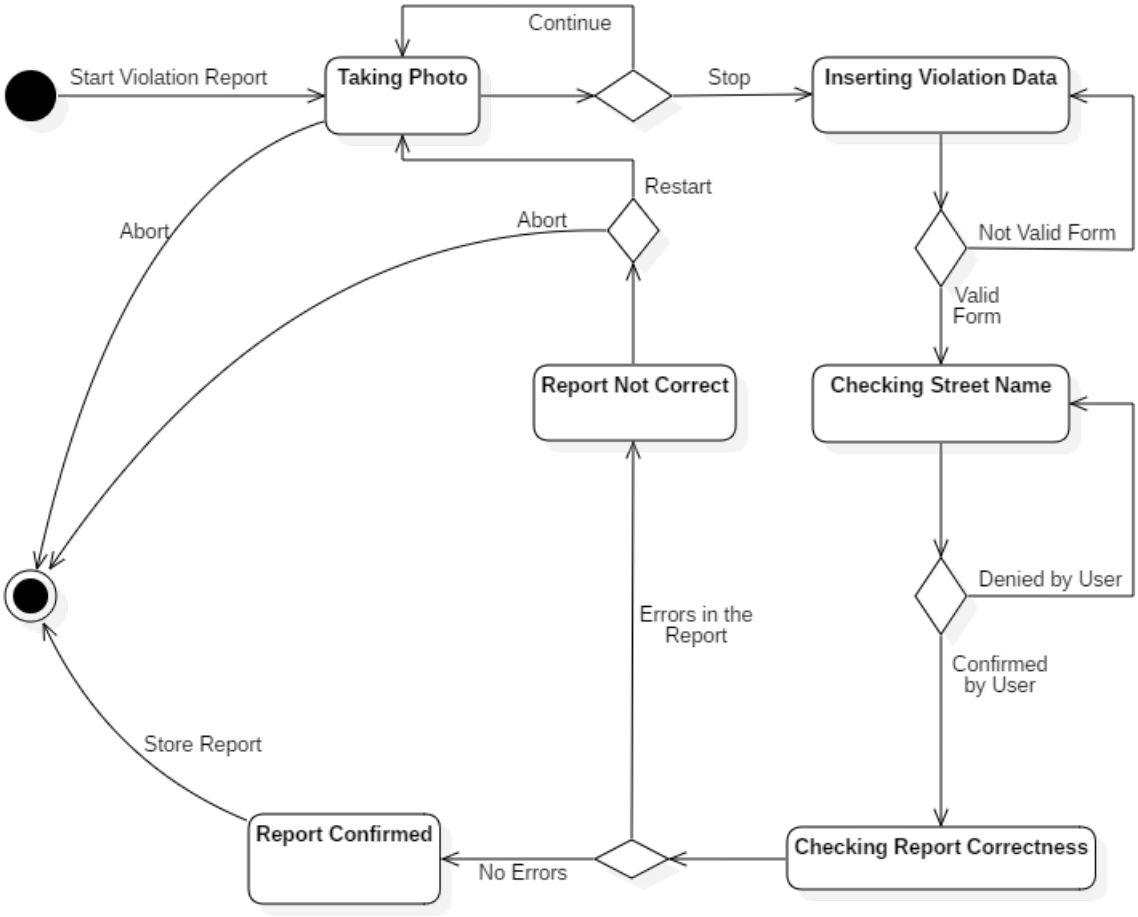


Figure 2.2: *Violation Report's State Chart*

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 expects are one or more pictures of the violation, allowing both taking pictures directly from the application or uploading them from the device's 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 lets 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* chose 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 double check the plate inforamtion.
  - *Description*: here the *User* can write a more detailed description of the street violation. This field is not mandatory, 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 returns 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 tries 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 fails at the ending phase of the report, the application informs 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*'s 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 notices 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*, the *Citizen* is able to view statistics information about the violations stored in the *System*; in particular a *Citizen* has the 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/she 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.

  Accept a Violation Report
  An *Authority* can also, besides the possibility to view and filter violations, accept a violation report; in this way the *Authority* is telling to the *System* that the violation will be processes by him/her so no other *Authorities* will manage that violation. After this choice the *Authority* can decide if it is better to go as soon as possible to the violation position to verify the violation or to immediately make a traffic ticket given the information provided by the report.

  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 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 committed 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 to *Citizens* for privacy issues.
- *Possible Interventions*: the *System* processes in another way the data at his disposal: it tries to create possible suggestions for interventions in discovered unsafe areas. To do that, two different algorithms are necessary: a Machine Learning one that extracts 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 transforms 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 accept violations reports.

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.
- 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
- The System will make use of the Users' smartphone's camera
- The System will make use of the Users' smartphone's clock

2.4.3 Assumptions

- [**D.1**] The *User*'s device operative system supports the application installation and all its features.
- [**D.2**] *Users* are uniquely identified, at the registration phase, by their username.
- [**D.3**] The given email in the registration phase is assumed correct.
- [**D.4**] The sent email is assumed to be correctly received.
- [**D.5**] Position data has an accuracy of 10 meters around the actual position.
- [**D.6**] Permission to access the device memory is always granted to *SafeStreets*.
- [**D.7**] Permission to access the GPS is always granted to *SafeStreets*.
- [**D.8**] Permission to access the camera is always granted to *SafeStreets*.
- [**D.9**] Data is stored on persistent memory.
- [**D.10**] The System storage is replicated and fault-tolerant, so that a copy of saved data is always available.
- [**D.11**] The application can correctly and automatically retrieve the current date and time.

3 Specific Requirements

3.1 External Interface Requirements

The *SafeStreet* application is a mobile based application. In the following section, a more detailed description of the application is given, in terms of hardware, software and communication interfaces. Are also given some basic prototypes of the GUI through mockups.

3.1.1 User Interfaces

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

- **Icon and Logo**
  The icon of *SafeStreets*, which will be displayed in the smartphones' homes, is quite minimal but, at the same time, embraces all the main characteristics of the application. The S stands both for *safe* and *street*, with this last one which is recreated thanks to its dividing line.

Figure 3.1: *Icon*

SafeStreets

Figure 3.2: *Logo*

- **Sign In/Sign Up**
  This form is the first thing the User sees after opening the app. It allows the User to log in, sign up or recover the password if lost. An intermediate tutorial page will be shown before signing up, in order to formally distinguish between the two allowed roles.

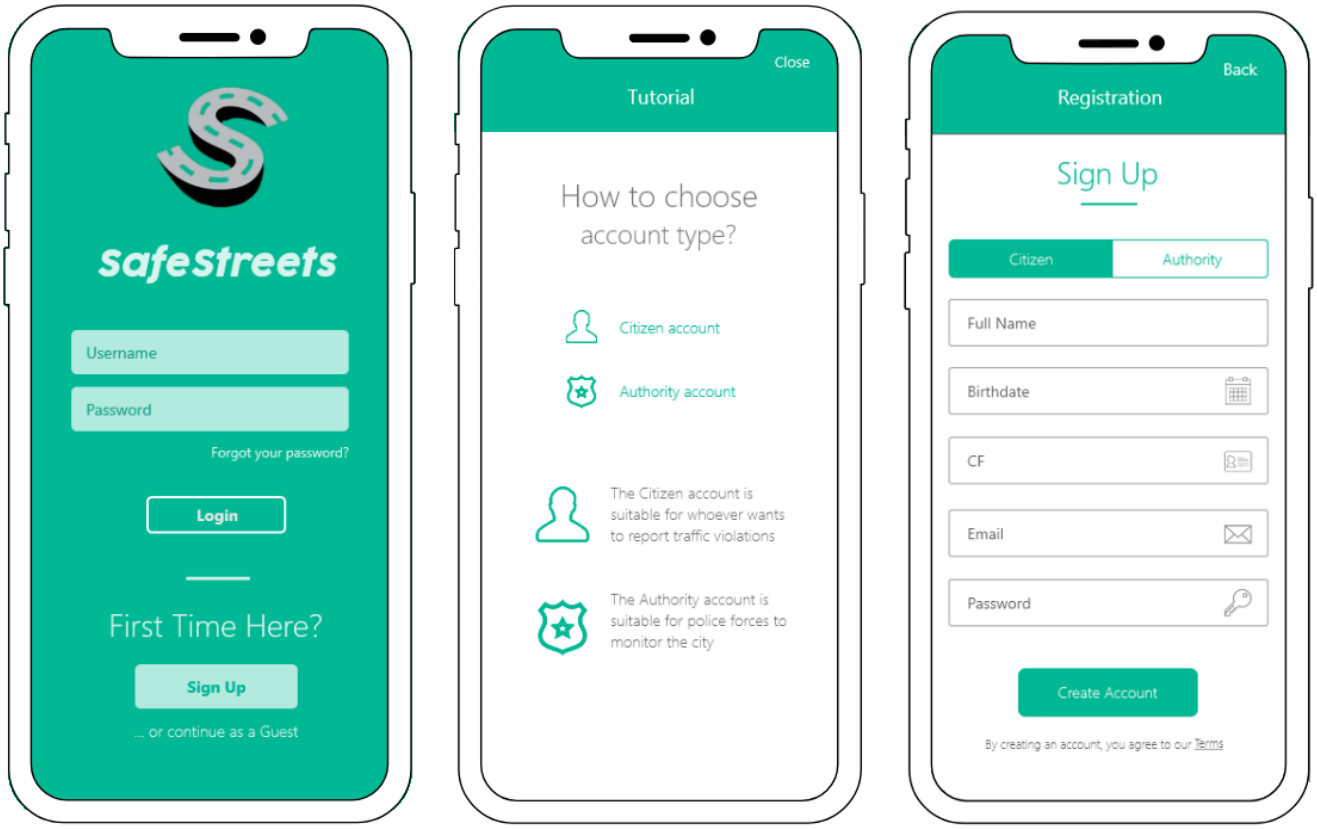


Figure 3.3: *Sign in and sign up procedures*

- **View Statistics**
  In this page Users can highlight some pubblically available statistics, e.g. the most unsafe areas in the city.

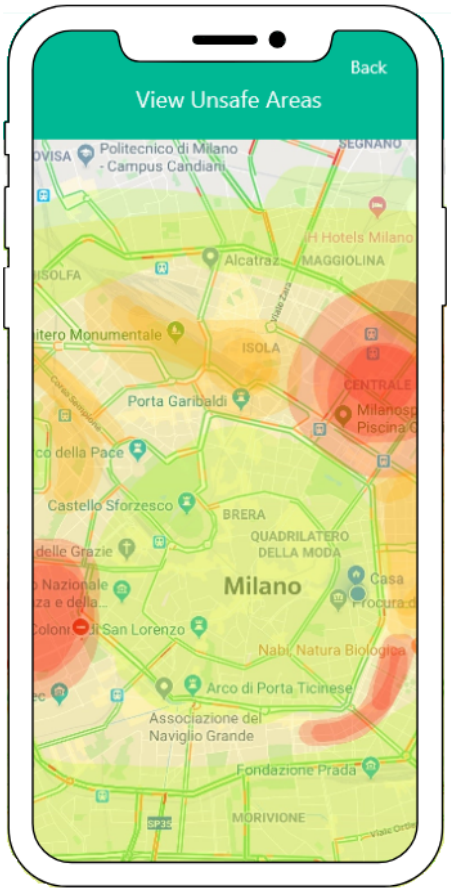


Figure 3.4: *Statistics about unsafe areas*

- **Report Violation**
  In this page Citizens can report a violation they incurred in by compiling the following form.

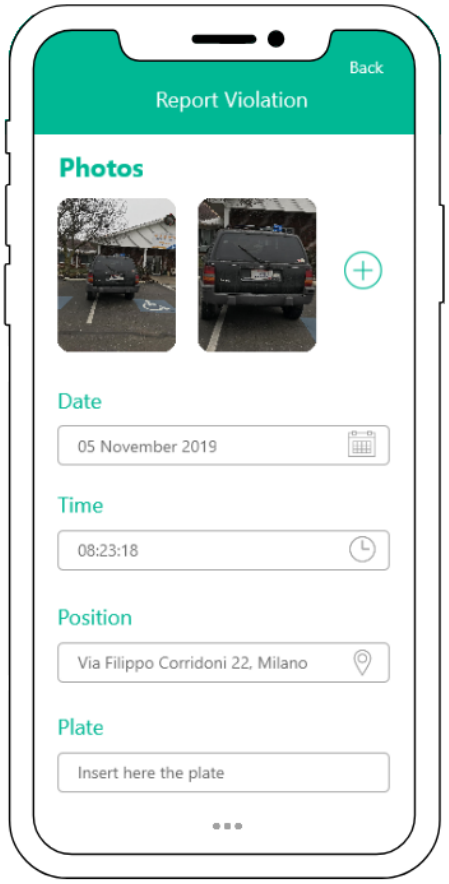


Figure 3.5: *Report violation procedure*

- **Report History**
  In this page Citizens can see all the reports they have generated since the creation of their account. They can also menage reports checking their status and modifying/deleting them if the request is still pending.

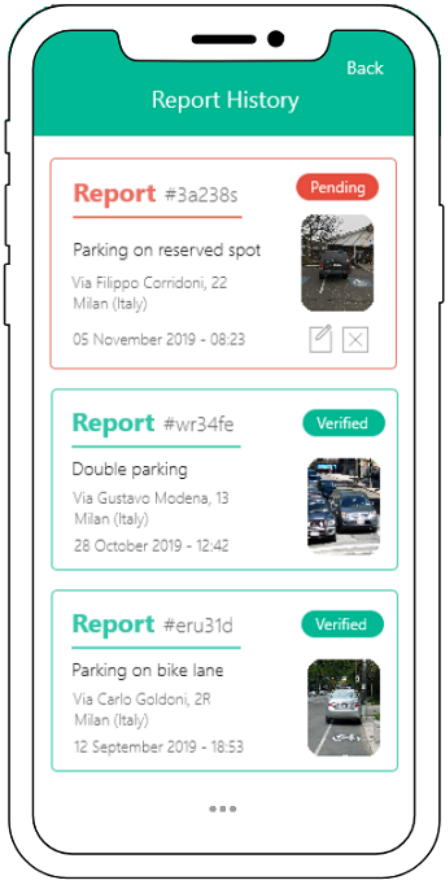


Figure 3.6: *Report history as seen by the Citizen*

- **View Violation**
  In this page Authorities can see in detail the report produced by a Citizen. It shows all the relevant details of the violation and of the associated Citizen. It allows the Authority to accept or refuse the report.

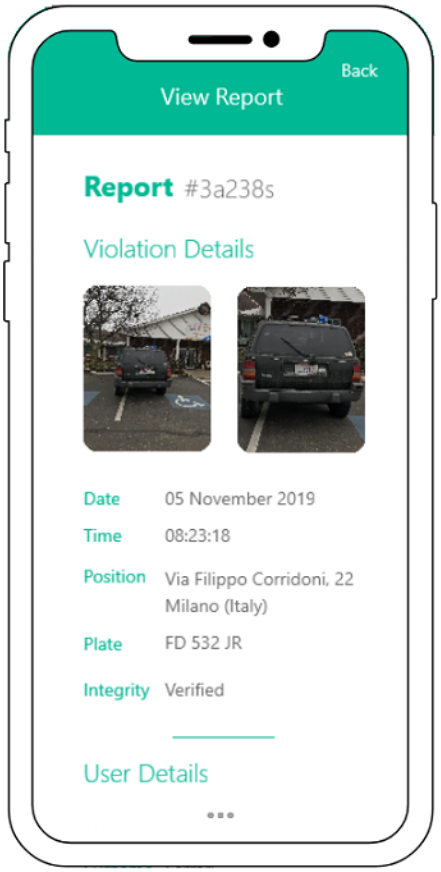


Figure 3.7: *Report of a violation as seen by an Authority*

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 and to the clock.

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 *Citizen* account can be created if and only if the User provides his/her fiscal code.
- [**R.2**] A *Citizen* account can be created if and only if the person is of legal age.
- [**R.3**] An *Authority* account can be created if and only if the User provides his/her badge number.
- [**R.4**] System shall check if the User's credentials are valid
- [**R.5**] System shall allow the User, under certain conditions, to change his email and password.

- [**R.6**] To access the services, *Citizens* and *Authorities* need to login with their credentials.
- [**R.7**] The System shall ask the User to agree on the policy about data acquisition.
- [**R.8**] The System shall allow only verified *Citizens* to report a violation.
- [**R.9**] The System shall provide an interface to report a violation.
- [**R.10**] The System shall allow the *Citizen* to upload up to 4 photos in the report.
- [**R.11**] The System shall precompile the *Street Name* field using the smartphone's GPS.
- [**R.12**] The System shall allow the *Citizen* to manually insert the *Street Name* field.
- [**R.13**] The System shall precompile the *Date* and *Time* using the smartphone's clock.
- [**R.14**] The System shall allow the *Citizen* to manually insert the *Date* and *Time* fields.
- [**R.15**] The System shall prevent the sending of reports with missing or incorrect fields.
- [**R.16**] The System shall store the violations of every single *Citizen*.
- [**R.17**] The System shall check the correctness of the license plate through a Machine Learning algorithm.
- [**R.18**] The System shall check the latitude and longitude metadata attached to the photo and cross them with the *Street Name* provided by the *Citizen*.
- [**R.19**] The System shall check the *Time* field, if manually input by the *Citizen*, is not subsequent to the precompiled one.
- [**R.20**] The System shall perform signal processing in order to spot possible counterfeit images.
- [**R.21**] The System shall allow the *Authority* to see the list of all the reported violations.

- [**R.22**] The System shall allow the *Authority* to filter the results according to the distance.
- [**R.23**] The System shall allow the *Authority* to filter the results according to the time.
- [**R.24**] The System shall allow the *Authority* to view the details of a report.
- [**R.25**] The System shall allow the *Authority* to view the *Citizens* profile.
- [**R.26**] The System shall allow *Citizens* to view the list of all the reported violations since they registered.
- [**R.27**] The System shall allow the *Citizen* to filter his own reported violations.
- [**R.28**] The System shall allow the *Citizen* to modify a pending report.
- [**R.29**] The System shall provide the most recent statistics.
- [**R.30**] The System shall allow the User to filter the results.
- [**R.31**] The System shall allow the User to search for a specific street in the map.
- [**R.32**] The System shall check the role of the User before providing the requested statistics.
- [**R.33**] The System shall allow *Guests* to navigate publicly available statistics.
- [**R.34**] The System shall be able to request Municipality's statistics, if those are provided.
- [**R.35**] The System shall be able to join its own knowledge base with the results returned by the Municipality API.
- [**R.36**] The System shall process the data in order to generate meaningful insights.
- [**R.37**] The System shall be able to display the results in a graphical format.
- [**R.38**] The System shall apply machine learning algorithms on statistical data.
- [**R.39**] The System shall apply natural language processing on the results in order to express them in natural language.

3.2.1 Use Cases

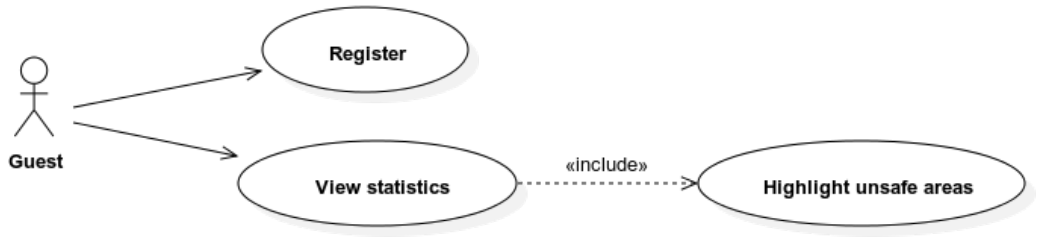


Figure 3.8: *Guest Uses Cases*

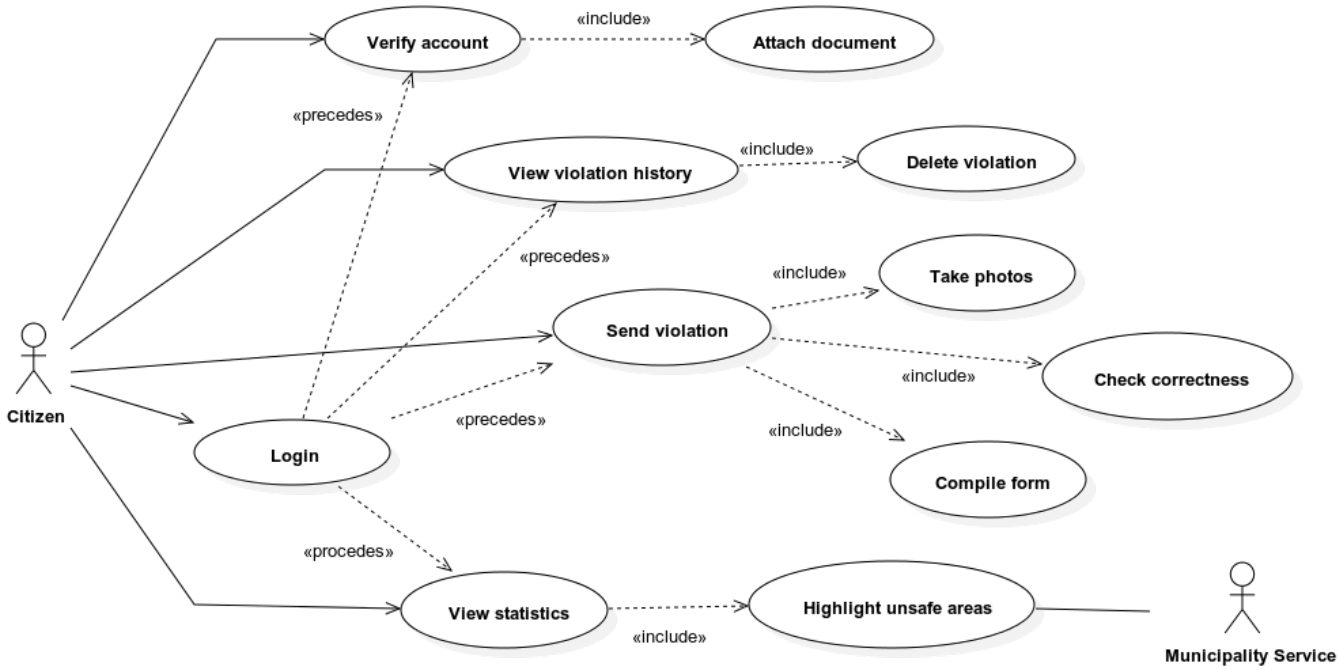


Figure 3.9: *Citizen Uses Cases*

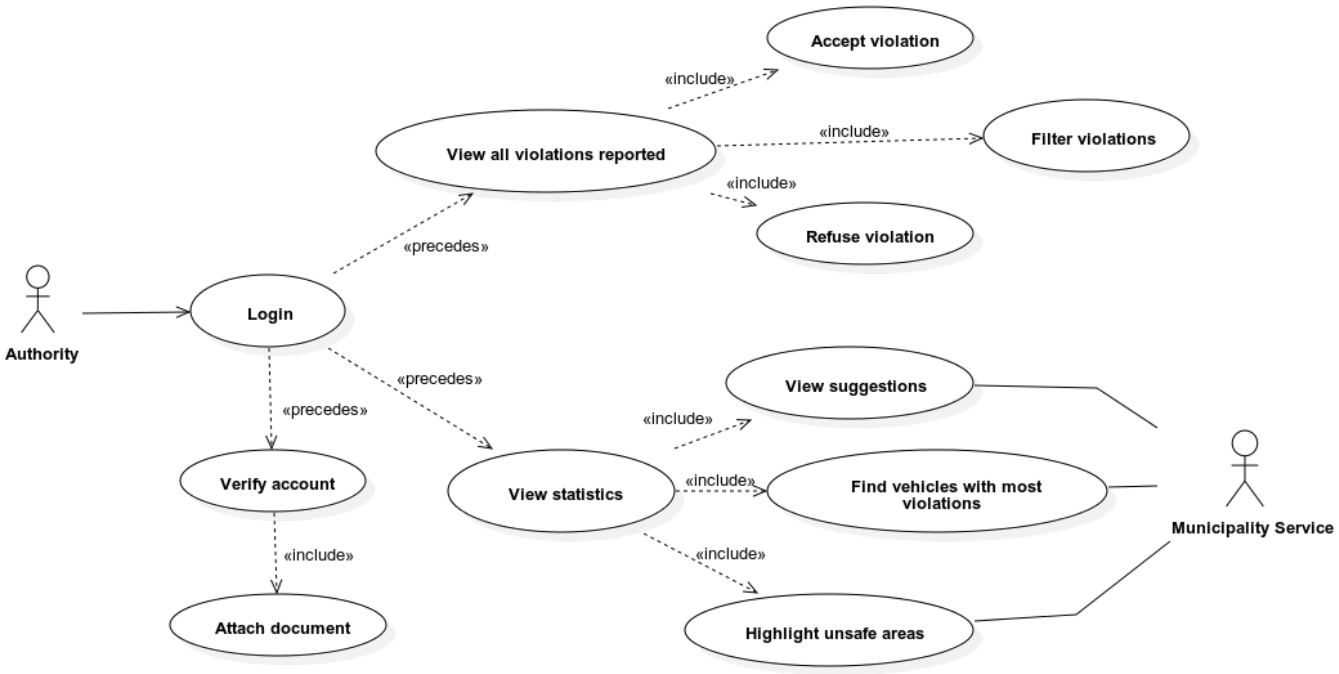


Figure 3.10: *Authority Uses Cases*

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, fiscal code, birthday, 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*** 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: same, surname, badge number, 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 an ***Authority***
Exceptions	• 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 proceed 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

ID	UC8
Description	An ***Authority*** wants to accept the closest reported violation
Actors	***Authority***
Preconditions	• The ***Authority*** successfully logged in • The ***Authority*** has verified the account
Flow of Events	1. The ***Authority*** selects the *View All Violations* tab 2. The ***System*** shows to the ***Authority*** the list of all the violations reported 3. The ***Authority*** clicks on *Filters* 4. The ***System*** shows all the available filters to the ***Authority*** 5. The ***Authority*** clicks on *Distance* 6. The ***System*** shows all the reported violations ordered from the closest to the furthest 7. The ***Authority*** selects the first one 8. The ***Authority*** checks the correctness of the violation 9. The ***Authority*** clicks on *Accept Violation*
Post conditions	The violation is archived as *accepted* and is no longer present in the list of all violations. The Authority, if it's possible, checks by person the veracity of the report with an inspection of a patrol.
Exceptions	• The Authority is unable to personally verify the violation and the provided photos are not sufficient to make a final judgment. The report is signed as *not verified* and discarded.

3.2.3 Scenarios

- **Scenario 1**
  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 2**
  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 report was accepted, she is very happy to find the free space in front of the racks and to park there her bike.

- **Scenario 3**
  Andrea Verdi is a model citizen who is always active in trying to improve the standard of living of the community. In particular, since he started to use *SafeStreets*, the traffic violation rate in his neighbourhood has dramatically dropped in the recent few weeks. Coming back from work as usual, Andrea notices that a car without handicap placard has occupied the reserved parking lot for disables, so he decides to report it thanks to *SafeStreets*. When he arrives at the front door of his house, he meets

Maria, an elderly lady who loves taking long walks in the neighborhood, and tells her about the recent inconvenience. Surprisingly, she replies that the reserved parking lot has been moved to the parallel street and that the signalling was wrong. Andrea, then, decides to immediately access the app and to find the incorrect report in his history, deleting it.

- **Scenario 4**
  Elena Ferrari is a young police officer who has recently been promoted to road patrolling. Since she is the new kid, she has the additional task of monitoring *SafeStreets*, a brand new application that her division has decided to test for two weeks. In order to be able to use all its functionalities, she registered an account providing the department number and her credentials, subsequently verified through the scan of a document. While she is at the traffic light, she decides to check the latest violations reported, filtering them according the distance from her position. Noticing that not far away from them there is a double parking that is obstructing the traffic, she informs her driving partner about the street name and they head for it in order to solve the problem.

3.2.4 Sequence and Activity Diagrams

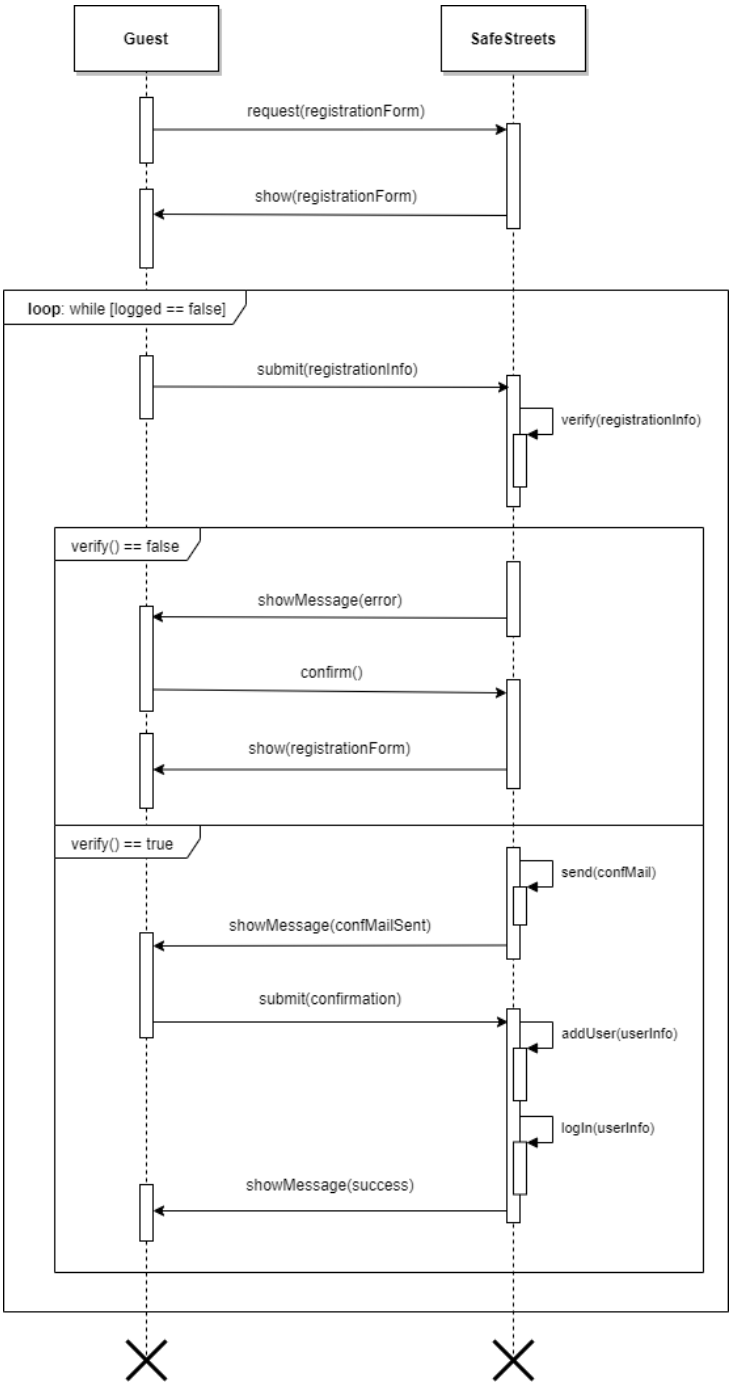


Figure 3.11: *Registration Sequence Diagram*

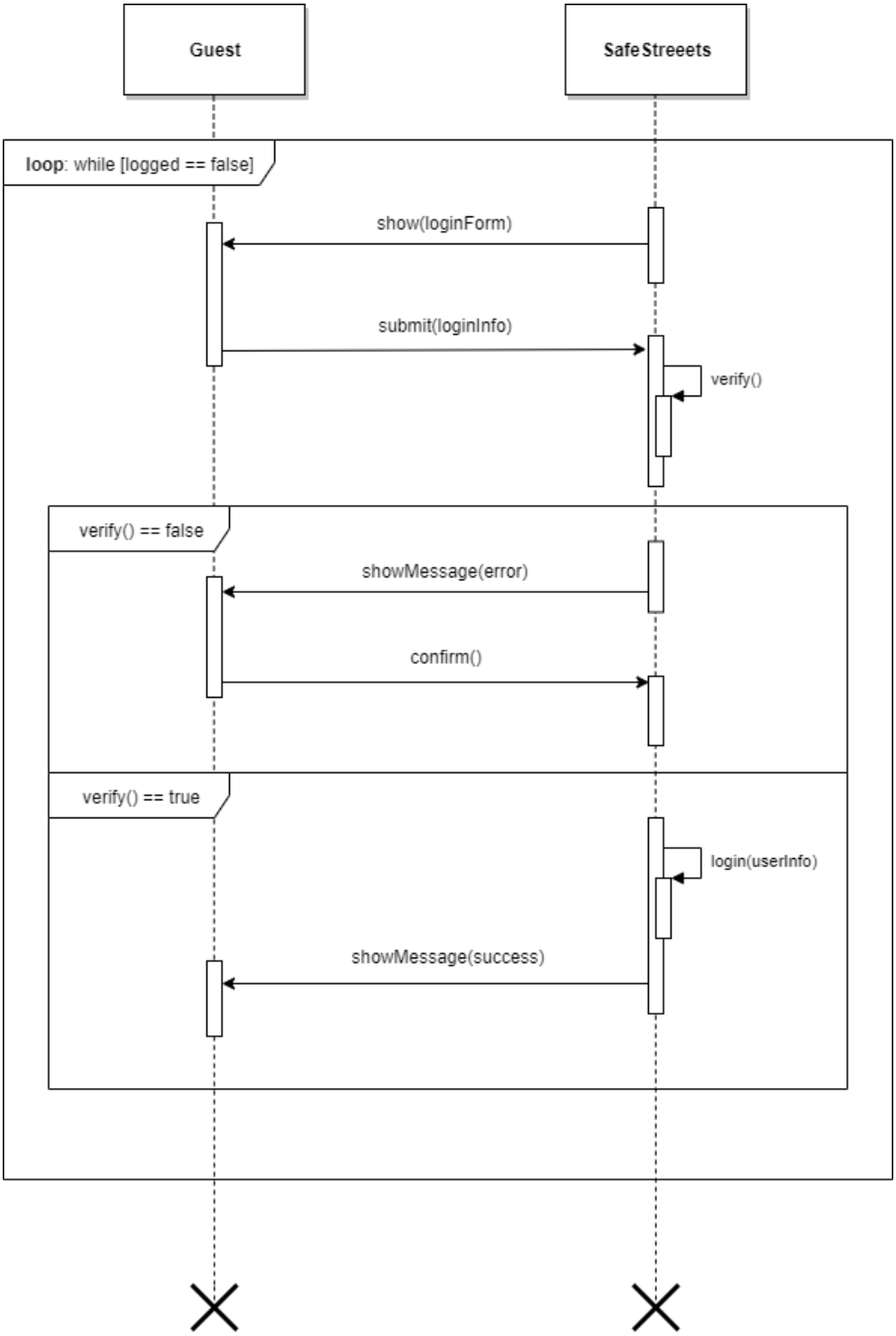


Figure 3.12: *Login Sequence Diagram*

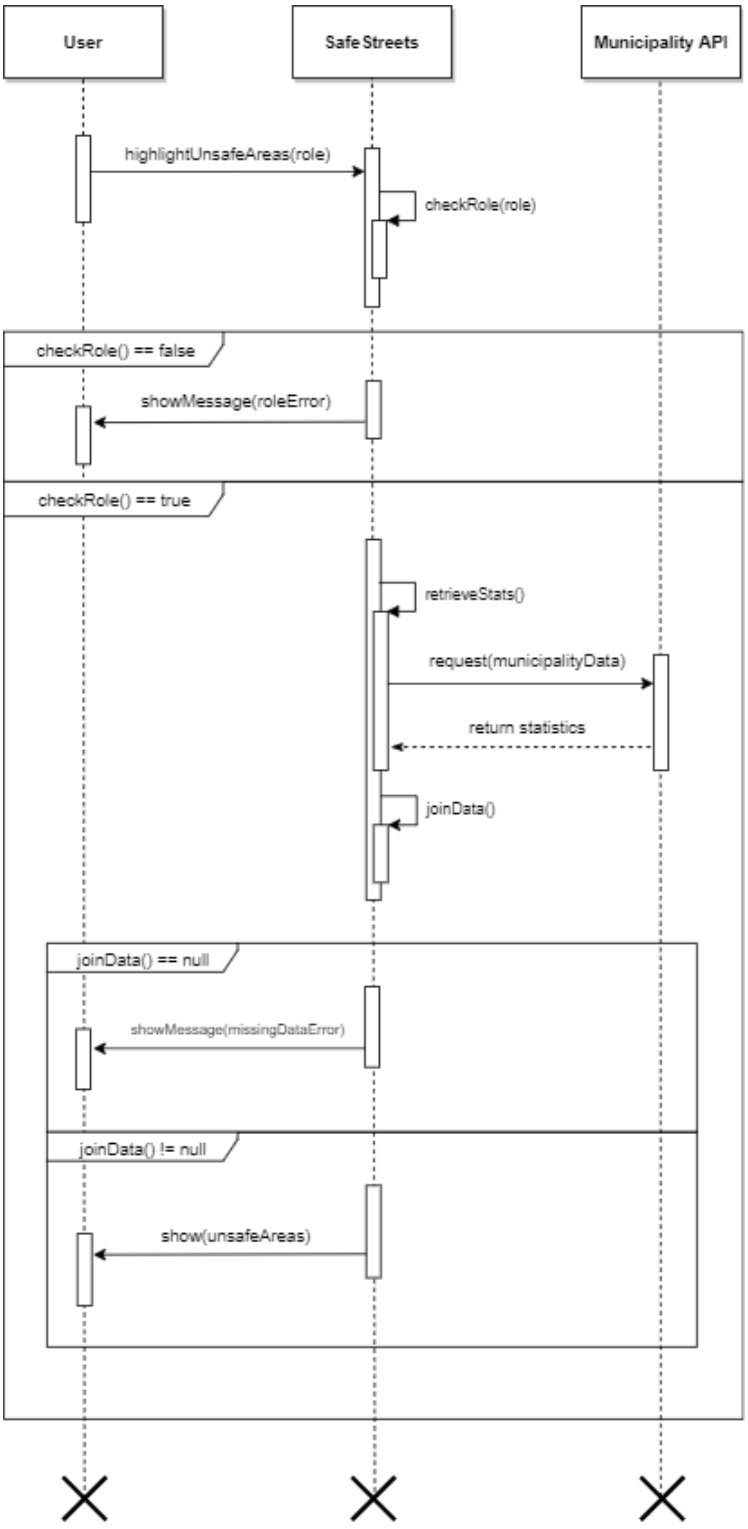


Figure 3.14: *View Statistics Sequence Diagram*

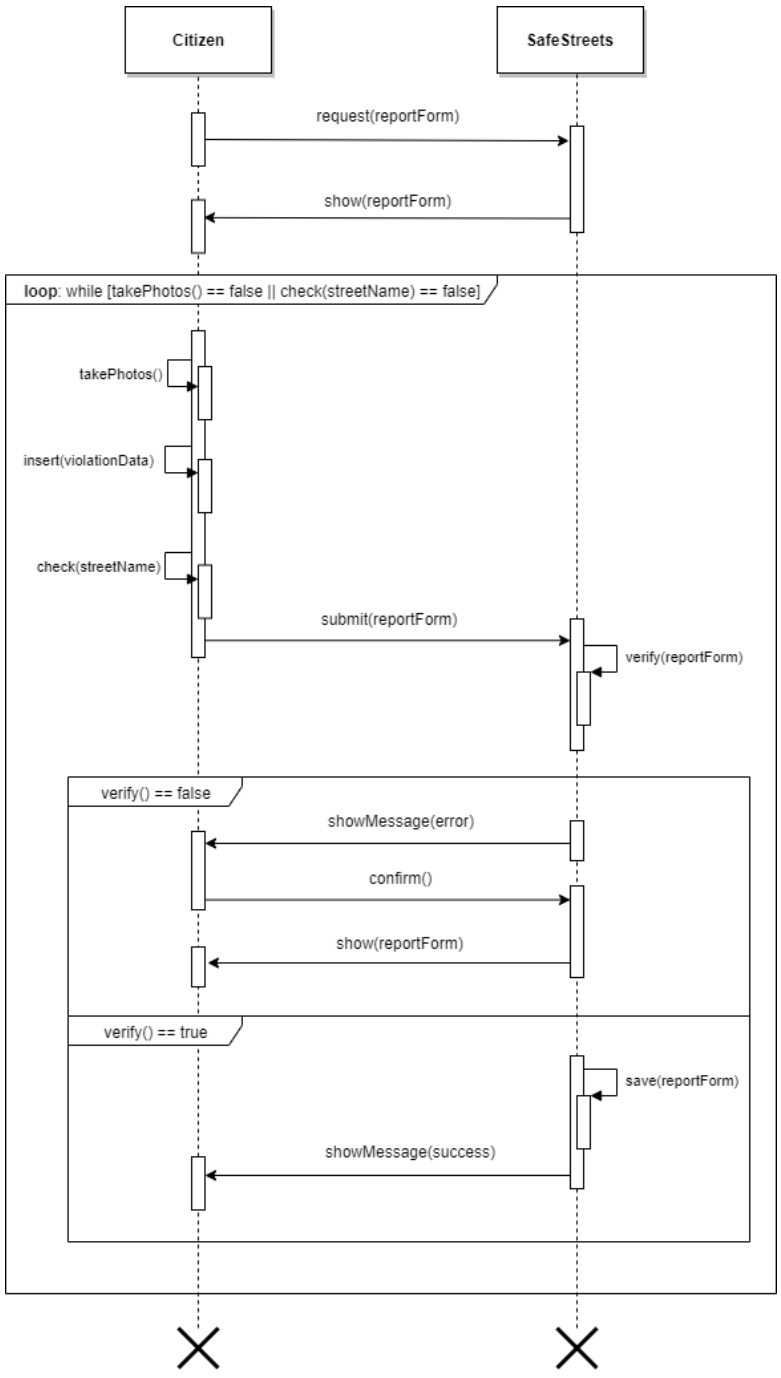


Figure 3.13: *Report Violation Sequence Diagram*

3.2.5 Mapping on Requirements

- [**G.1**] The System allows the User to register to the application either as *Citizen* or as *Authority*, providing an username and a password.
  - [**R.1**] A *Citizen* account can be created if and only if the User provides his/her fiscal code.
  - [**R.2**] A *Citizen* account can be created if and only if the person is of legal age.
  - [**R.3**] An *Authority* account can be created if and only if the User provides his/her badge number.
  - [**R.5**] System shall check if the User's credentials are valid
  - [**R.6**] System shall allow the User, under certain conditions, to change his email and password.
  - [**R.7**] To access the services, *Citizens* and *Authorities* need to login with their credentials.
  - [**R.8**] The System shall ask the User to agree on the policy about data acquisition.
  - [**D.1**] The *User*'s device operative system supports the application installation and all its features.
  - [**D.2**] Users are uniquely identified, at the registration phase, by their username.
  - [**D.3**] The given email in the registration phase is assumed correct.
  - [**D.4**] The sent email is assumed to be correctly received.
- [**G.2**] The System allows the User to report a traffic violation by sending photos and relative information.
  - [**R.8**] The System shall allow only verified *Citizens* to report a violation.
  - [**R.9**] The System shall provide an interface to report a violation.
  - [**R.10**] The System shall allow the *Citizen* to upload up to 4 photos in the report.
  - [**R.11**] The System shall precompile the *Street Name* field using the smartphone's GPS.
  - [**R.12**] The System shall allow the *Citizen* to manually insert the *Street Name* field.

  - [**R.13**] The System shall precompile the *Date* and *Time* using the smartphone's clock.
  - [**R.14**] The System shall allow the *Citizen* to manually insert the *Date* and *Time* fields.
  - [**R.15**] The System shall prevent the sending of reports with missing or incorrect fields.
  - [**R.16**] The System shall store the violations of every single *Citizen*.
  - [**D.5**] Position data has an accuracy of 10 meters around the actual position.
  - [**D.6**] Permission to access the device memory is always granted to *SafeStreets*.
  - [**D.7**] Permission to access the GPS is always granted to *SafeStreets*.
  - [**D.8**] Permission to access the camera is always granted to *SafeStreets*.
  - [**D.9**] Data is stored on persistent memory.
  - [**D.10**] The System storage is replicated and fault-tolerant, so that a copy of saved data is always available.
  - [**D.11**] The application can correctly and automatically retrieve the current date and time.

- [**G.3**] The System checks that the photos provided by the User have not been compromised.
  - [**R.17**] The System shall check the correctness of the license plate through a Machine Learning algorithm.
  - [**R.18**] The System shall check the latitude and longitude meta-data attached to the photo and cross them with the *Street Name* provided by the *Citizen*.
  - [**R.19**] The System shall check the *Time* field, if manually input by the *Citizen*, is not subsequent to the precompiled one.
  - [**R.20**] The System shall perform signal processing in order to spot possible counterfeit images.
  - [**D.5**] Position data has an accuracy of 10 meters around the actual position.
  - [**D.11**] The application can correctly and automatically retrieve the current date and time.

- [**G.4**] The System allows the Authority to accept or refuse a violation.
  - [**R.21**] The System shall allow the *Authority* to see the list of all the reported violations.
  - [**R.22**] The System shall allow the *Authority* to filter the results according to the distance.
  - [**R.23**] The System shall allow the *Authority* to filter the results according to the time.
  - [**R.24**] The System shall allow the *Authority* to view the details of a report.
  - [**R.25**] The System shall allow the *Authority* to view the *Citizens* profile.
  - [**D.5**] Position data has an accuracy of 10 meters around the actual position.
  - [**D.10**] The System storage is replicated and fault-tolerant, so that a copy of saved data is always available.
- [**G.5**] The System allows the User to view his/her own history of reported violations.
  - [**R.26**] The System shall allow *Citizens* to view the list of all the reported violations since they registered.
  - [**R.27**] The System shall allow the *Citizen* to filter his own reported violations.
  - [**R.28**] The System shall allow the *Citizen* to modify a pending report.
  - [**D.10**] The System storage is replicated and fault-tolerant, so that a copy of saved data is always available.
- [**G.6**] The System allows the User to delete a reported violation.
  - [**R.26**] The System shall allow *Citizens* to view the list of all the reported violations since they registered.
- [**G.7**] The System allows the User to consult the created statistics according to his/her role.
  - [**R.29**] The System shall provide the most recent statistics.
  - [**R.30**] The System shall allow the User to filter the results.

  - [**R.31**] The System shall allow the User to search for a specific street in the map.
  - [**R.32**] The System shall check the role of the User before providing the requested statistics.
  - [**R.33**] The System shall allow *Guests* to navigate publicly available statistics.
  - [**D.5**] Position data has an accuracy of 10 meters around the actual position.
  - [**D.10**] The System storage is replicated and fault-tolerant, so that a copy of saved data is always available.

- [**G.8**] The System allows the Authority to elaborate statistics with the help of the Municipality API.
  - [**R.29**] The System shall provide the most recent statistics.
  - [**R.34**] The System shall be able to request Municipality's statistics, if those are provided.
  - [**R.35**] The System shall be able to join its own knowledge base with the results returned by the Municipality API.
  - [**D.10**] The System storage is replicated and fault-tolerant, so that a copy of saved data is always available.

- [**G.9**] The System allows the Authority to consult a list of suggested interventions.
  - [**R.29**] The System shall provide the most recent statistics.
  - [**R.32**] The System shall check the role of the User before providing the requested statistics.
  - [**R.34**] The System shall be able to request Municipality's statistics, if those are provided.
  - [**R.35**] The System shall be able to join its own knowledge base with the results returned by the Municipality API.
  - [**R.38**] The System shall apply machine learning algorithms on statistical data.
  - [**R.39**] The System shall apply natural language processing on the results in order to express them in natural language.
  - [**D.10**] The System storage is replicated and fault-tolerant, so that a copy of saved data is always available. of saved data is always available.

3.3 Performance Requirements

SafeStreets is structured in a three-tier architecture. All the application logic is built in the application servers: the backend performances should guarantee the proper operation in case of traffic peaks and manage multiple Users.

3.4 Design Constraints

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

In this section we provide a model of the *System* using Alloy language and Alloy Analyzer tool. Thanks to this declarative language we can define our domain and check possible *System* views in different moment of its life-cycle.

4.1 Description

We can briefly describe our model before showing the code and the generated world.

In the model we find some main entities and some secondary signatures that give the idea of information the *System* have. The first category is composed by:

- Citizen (a Citizen *sends* ViolationReports)
- Authority (an Authority *accepts* ViolationReports)
- ViolationReport
- Statistic (we can have AreasStatistics, VehiclesStatistics and InterventionsStatistics)

In the second case we have:

- Username (to identify an unique generic *User*)
- FiscalCode and AuthorityID (to identify *Citizens* and *Authorities*)
- Position and StreetName (to give importance to the position features that *SafeStreets* offers)
- ViolationManager and MiningManager (to retrieve information from ViolationReports creating statistics to offer to *Users*)

These entities exchange information with each other in the way we defined in the model (showed next).

4.2 Code and Alloy Analyzer Output

```
-- We import util/boolean to have a boolean variable that indicates if a User
    ↪ is verified or not
open util/boolean

--------------------SIGNATURES----------------------
---------- Users
abstract sig User{}
sig Username, AuthorityID, FiscalCode, Email, Plate{}

-- A Logged User represents both Citizens and Authorities common features:
-- each of them has an unique username and an unique email
abstract sig LoggedUser extends User{
        username: one Username,
        email: one Email,
        verified: one Bool
}

-- A Citizen indicates an unique FiscalCode to register, the date
-- of birth in order to let the application making constraints by age.
-- Further we have the set of ViolationRepors sent to the app and
-- a set of AreasStatistics showed
sig Citizen extends LoggedUser{
        fiscalcode: one FiscalCode,
        dateOfBirth: one Date,
        violations: set ViolationReport,
        areasStat: set AreasStatistic
}{dateOfBirth.year < 4} // We simulate a constraint on the age of the Citizen

sig Date{
        year: one Int // For semplicity we use alloy's Int data type
}{year > 0}

-- An Authority instead uses an AuthorityID to register. Here we have
-- also a set of accepted ViolationReports and all the sets of consulted
    ↪ Statistics
-- (an Authority can view statistics about Areas, Vehicles or Interventions)
sig Authority extends LoggedUser{
        authorityID : one AuthorityID,
        violations: set ViolationReport,
        areasStat: set AreasStatistic,
        vehiclesStat: set VehiclesStatistic,
        interventionsStat: set InterventionsStatistic
}



---------- Violations
sig Picture, StreetName{}

-- A ViolationReport has an unique id, a set of Pictures taken by the Citizen
    ↪ (saved
-- as "user"), the plate of the vehicles (in order to send it to the
    ↪ ViolationManager)
-- and the position of the violation
sig ViolationReport{
        id: one Int,
        pictures: set Picture,
```

```
        user: one Citizen,
        position: one Position,
        plate: one Plate
}{#pictures > 0 and #pictures ≤ 4 and id > 0} // We want a positive id and we
    ↪ set constraints on the number of photos









-- A Position is made of longitude and latitude (Int for semplicity).
-- Furthermore we add the StreetName retrived by the system
sig Position{
        longitude: one Int,
        latitude: one Int,
        streetName: one StreetName
}

-- The real ViolationManager has lots of functionalities; here we focus on
    ↪ the fact
-- that we need a manager that retrieves information from the violations and
    ↪ sends
-- them to a MiningManager to permit statistics creation
one sig ViolationManager{
        violations: set ViolationReport,
        miningManager: one MiningManager // The MiningManager is unique in
            ↪ the model
}



---------- Statistics
one sig MiningManager{
        areasStat: set AreasStatistic,
        vehiclesStat: set VehiclesStatistic,
        interventionsStat: set InterventionsStatistic
}

-- Each Statistic has an id in order to be uniquely identified
abstract sig Statistic{
        id: one Int
}{id > 0}

-- Follow different types of Statistics
sig AreasStatistic extends Statistic{}
```

```
sig VehiclesStatistic extends Statistic{}

sig InterventionsStatistic extends Statistic{}



--------------------FACTS--------------------
-- I must ensure that some elements don't exist without the existance of
    ↪ other ones
fact existentiality{
        -- There is not a username not linked to a LoggedUser
        no u: Username | no l: LoggedUser | l.username = u
        -- Same thing for email, date
        no e: Email | no l: LoggedUser | l.email = e
        no d: Date | no c:Citizen | c.dateOfBirth= d
        -- There is not a FiscalCode or an AuthorityID without the
            ↪ respectively Users
        no f: FiscalCode | no c:Citizen | c.fiscalcode= f
        no au: AuthorityID | no a:Authority | a.authorityID = au
        -- There is not any ViolationReport not linked to a Citizen who made
            ↪ it
        no v: ViolationReport | no c: Citizen | v in c.violations
        -- There is not Positions and Pictures without a linked
            ↪ ViolationReport
        no p:Position | no v:ViolationReport | v.position = p
        no p: Picture | no v: ViolationReport | p in v.pictures
        -- There is not any StreetName without a Position
        no s:StreetName | no p:Position | p.streetName = s

        -- There is not any type of Statistic if not present in the Mining
            ↪ Manager
        no a: AreasStatistic | no m:MiningManager | a in m.areasStat
        no v: VehiclesStatistic | no m:MiningManager | v in m.vehiclesStat
        no i: InterventionsStatistic | no m:MiningManager | i in m.
            ↪ interventionsStat
}

-- I must ensure that some of the attributes of some of the Signatures are
    ↪ unique
fact unicity{
        -- There are not 2 different Citizens with the same FiscalCode
        no disj c1, c2: Citizen | c1.fiscalcode = c2.fiscalcode
        -- There are not 2 different ViolationReports with the same id
        no disj v1, v2: ViolationReport | v1.id = v2.id
        -- There are not 2 different LoggedUsers with the same Username
        no disj l1, l2: LoggedUser | l1.username = l2.username
        -- There are not 2 different LoggedUser with the same Email
        no disj l1, l2: LoggedUser | l1.email = l2.email
        -- There are not 2 different ViolationReports that contain the same
            ↪ photo
        no disj v1,v2: ViolationReport | v1.pictures & v2.pictures ≠ none
        -- There are not 2 different Citizens that have the same
            ↪ ViolationReport in their history
        no disj c1,c2: Citizen | c1.violations & c2.violations ≠ none
        -- There are not 2 different Authorities that have the same accepted
            ↪ ViolationReport
        -- (a Violation Report is accepted by only an Authority)
        no disj a1,a2: Authority | a1.violations & a2.violations ≠ none
        -- There are not 2 different Statistics (whatever the type of
            ↪ statistic is)
```

```
        -- that are identified by the same id
        all a: AreasStatistic, v: VehiclesStatistic, i:
            ↪ InterventionsStatistic | a.id ≠ v.id and a.id ≠ i.id and v.id ≠
            ↪  i.id
}

-- I must ensure that all the LoggedUser are verified or not
fact verifiedOrNot{
        all l:LoggedUser | l.verified = False or l.verified = True
}

-- I must ensure that if the Citizen is not verified no violations are in the
    ↪  set,
-- in the same way also if an Authority is not verified no violations can be
    ↪ accepted
fact name{
        all a: Authority | a.verified = False iff #(a.violations) = 0
        all c: Citizen | c.verified = False iff #(c.violations) = 0
}

-- I must ensure that the "user" field of a ViolationReport is the same
    ↪ unique Citizen who
-- have that in the set of violations
fact uniqueCitizenLink{
        all v:ViolationReport, c:Citizen | v.user = c iff v in c.violations
}

-- I must ensure that all the AreasStatics consulted by some Authorities or
    ↪ Citizens are the ones the MiningManager has created;
-- instead for VehiclesStatistics and InterventionsStatistics we ensure the
    ↪ same thing only for Authorities
fact StatisticsConstraints{
        all m: MiningManager, ar: AreasStatistic | some a: Authority | ar in
            ↪ a.areasStat implies ar in m.areasStat
        all m: MiningManager, ar: AreasStatistic | some c: Citizen | ar in c.
            ↪ areasStat implies ar in m.areasStat
        all m: MiningManager, ve: VehiclesStatistic | some a: Authority | ve
            ↪ in a.vehiclesStat iff ve in m.vehiclesStat
        all m: MiningManager, i: InterventionsStatistic | some a: Authority |
            ↪  i in a.interventionsStat iff i in m.interventionsStat
}

-- I must ensure that the ViolationManager contains all the ViolationReports
fact allReportsInViolationManager{
        all v: ViolationReport, vm: ViolationManager | v in vm.violations
}



--------------------ASSERTIONS AND PREDICATES--------------------
-- We assert that any ViolationReport is sent by an unique Citizen
assert noReportsWithoutCitizen{
        all v:ViolationReport | one c:Citizen | v in c.violations
}

pred show{
# Statistic > 2
}

pred addViolationReport[c: Citizen, v: ViolationReport]{
```

```
        c.violations = c.violations + v
        v.user = c
}

pred showAreasStatistic[c: Citizen, ar: AreasStatistic]{
        c.areasStat = c.areasStat + ar
}

pred acceptViolationReport[a: Authority, v: ViolationReport]{
        a.violations = a.violations + v
}


--------------------CHECKS AND RUNS--------------------
check noReportsWithoutCitizen

run show for 7 but exactly 1 Citizen, 2 Authority, 3 Picture, 2
    ↪ ViolationReport

run show for 7 but exactly 3 Citizen, 1 Authority, 5 ViolationReport

run addViolationReport for 7 but exactly 1 Citizen

run showAreasStatistic for 7 but exactly 2 Authority

run acceptViolationReport for 7 but exactly 3 ViolationReport
```

4.3 Model Checks

Executing "Check noReportsWithoutCitizen"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
6307 vars. 541 primary vars. 13315 clauses. 24ms.
No counterexample found. Assertion may be valid. 3ms.

Executing "Run show for 7 but exactly 1 Citizen, 2 Authority, 3 Picture, 2 ViolationReport"
Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
12060 vars. 966 primary vars. 24021 clauses. 46ms.
Instance found. Predicate is consistent. 40ms.

Executing "Run show for 7 but exactly 3 Citizen, 1 Authority, 5 ViolationReport"
Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
16520 vars. 1249 primary vars. 33135 clauses. 66ms.
Instance found. Predicate is consistent. 52ms.

Executing "Run addViolationReport for 7 but exactly 1 Citizen"
Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
20294 vars. 1576 primary vars. 39629 clauses. 71ms.
Instance found. Predicate is consistent. 70ms.

Executing "Run showAreasStatistic for 7 but exactly 2 Authority"
Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
22631 vars. 1635 primary vars. 45029 clauses. 69ms.
Instance found. Predicate is consistent. 93ms.

Executing "Run acceptViolationReport for 7 but exactly 3 ViolationReport"
Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
21636 vars. 1637 primary vars. 43284 clauses. 83ms.
Instance found. Predicate is consistent. 48ms.

6 commands were executed. The results are:
#1: No counterexample found. noReportsWithoutCitizen may be valid.
#2: **Instance found.** show is consistent.
#3: **Instance found.** show is consistent.
#4: **Instance found.** addViolationReport is consistent.
#5: **Instance found.** showAreasStatistic is consistent.
#6: **Instance found.** acceptViolationReport is consistent.

Figure 4.1: *All the checks done by the tool at the "Execute All" command.*

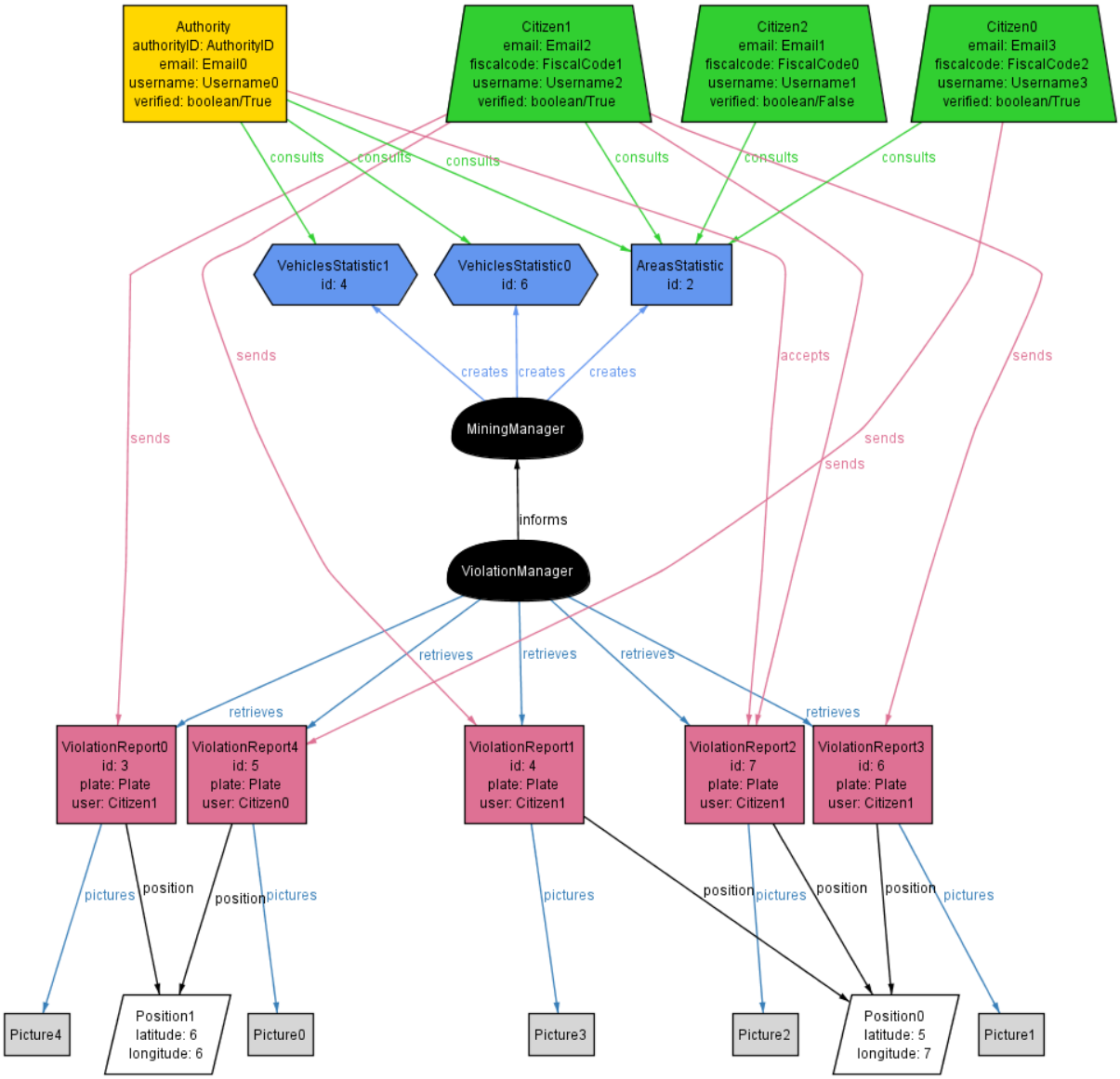
4.4 Generated Worlds

```
pred show{
# Statistic > 2
}

run show for 7 but exactly 1 Citizen, 2 Authority, 3 Picture, 2
    ↪ ViolationReport
```

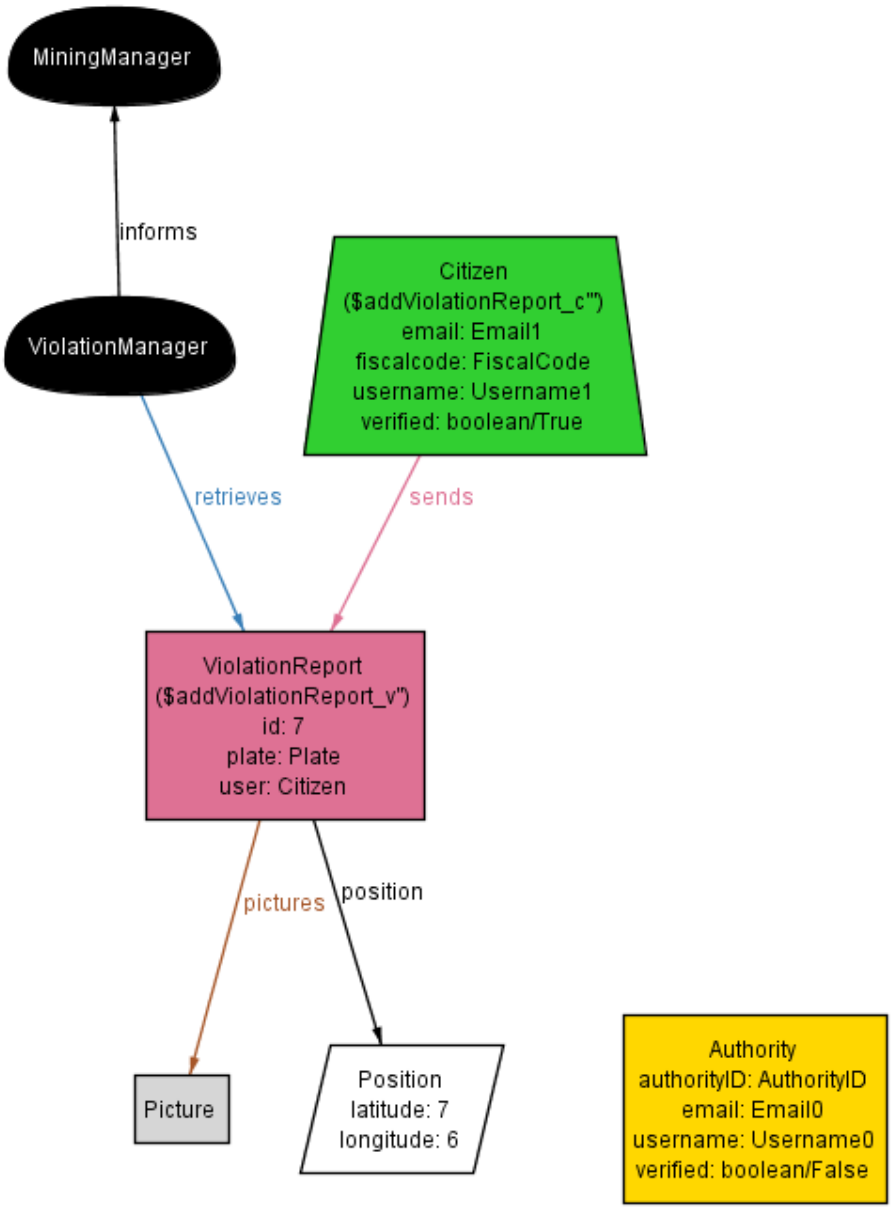
```
pred show{
# Statistic > 2
}

run show for 7 but exactly 3 Citizen, 1 Authority, 5 ViolationReport
```



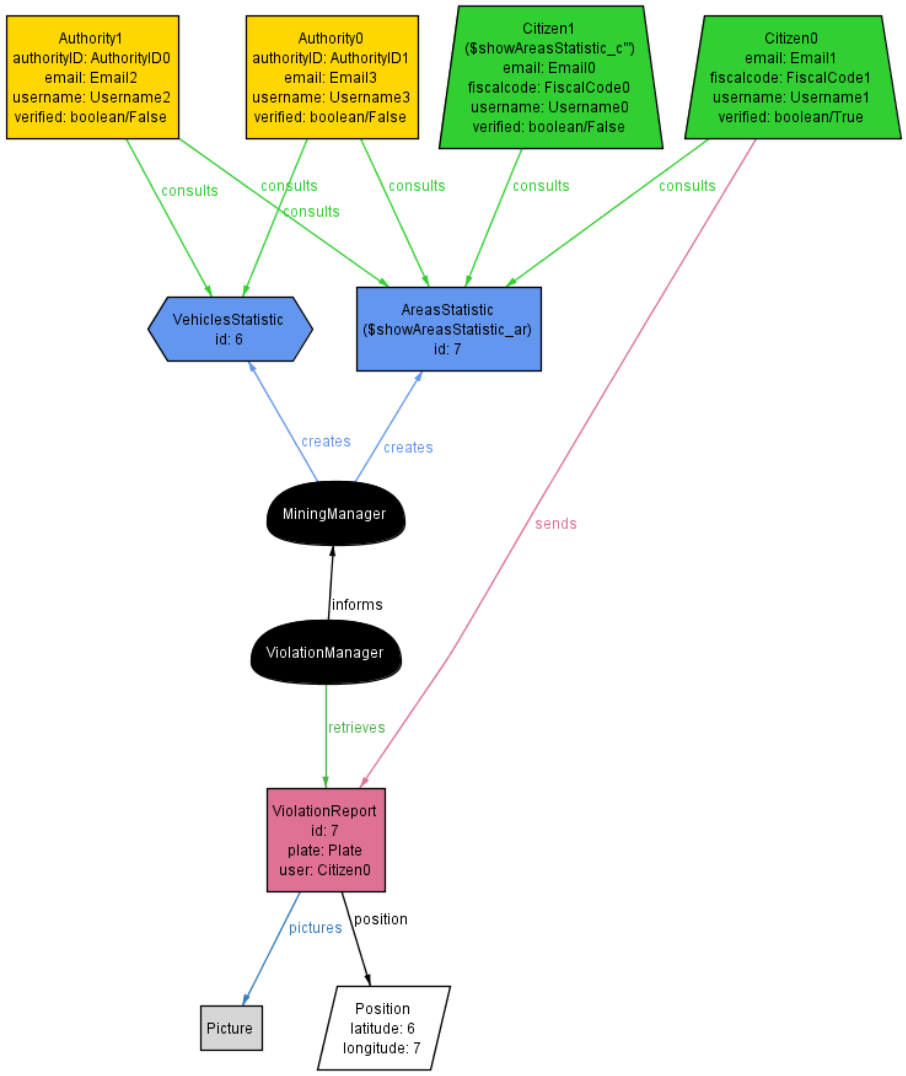
```
pred addViolationReport[c: Citizen, v: ViolationReport]{
        c.violations = c.violations + v
        v.user = c
}

run addViolationReport for 7 but exactly 1 Citizen
```



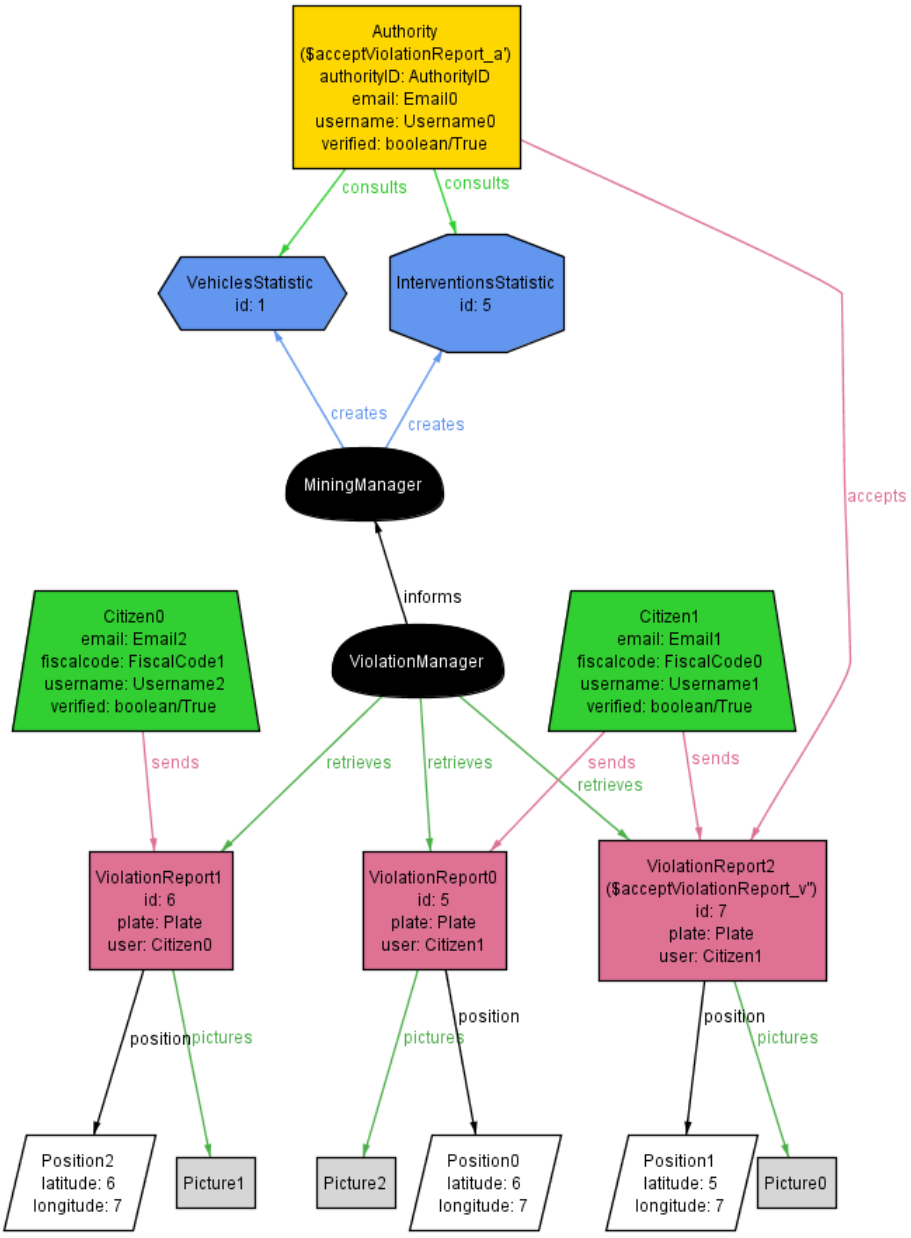
```
pred showAreasStatistic[c: Citizen, ar: AreasStatistic]{
        c.areasStat = c.areasStat + ar
}

run showAreasStatistic for 7 but exactly 2 Authority
```



```
pred acceptViolationReport[a: Authority, v: ViolationReport]{
        a.violations = a.violations + v
}

run acceptViolationReport for 7 but exactly 3 ViolationReport
```



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

Task	Hours
First Meeting	4
Git and Overleaf Setup	2
Assignment Analysis and Workflow Decision	8
Purpose and Scope	3
Introduction Other Sections	2
Product Perspective	1
Overall Description Other Sections	1
Interfaces Description	2
Mockups Creation	9
Mid-phase Meeting	4
Use Cases Creation	7
Requirements and Constraints	6
Mapping of Goals and Requirements	2
Alloy	8
Effort Tracking	2
Global Review	8
Total Hours	69

Stiven Metaj

Task	Hours
First Meeting	4
Git and Overleaf Setup	3
Assignment Analysis and Workflow Decision	8
Purpose and Scope	3
Introduction Other Sections	3
Product Perspective	4
Overall Description Other Sections	3
Interfaces Description	3
Mockups Creation	5
Mid-phase Meeting	4
Use Cases Creation	7
Requirements and Constraints	5
Mapping of Goals and Requirements	1
Alloy	10
Effort Tracking	2
Global Review	4
Total Hours	69