SafeStreets

Marco Premi, Fabrizio Siciliano, Giuseppe Taddeo

November 3, 2019

Contents

1 Introduction

1.1 Purpose

1.1.1 General Purpose

The purpose of this document is to correctly analyze all requirements, goals and actions needed in order to correctly develop SafeStreets.

SafeStreets is a crowd-sourced application that intends to provide users with the possibility to notify authorities when traffic violations occur (eg. traffic violations) and will help to maintain stability and order within the streets. A reporting system will also be available to police offices and municipality employees in order to allow them to analyze (and take actions accordingly) different areas of the city and assess which areas have the most violations committed in.

The core application will focus on storing useful traffic violation data provided by users, mainly with the help of input forms and hard evidence such as images. At any violation input, SafeStreets will also store useful metadata such as date and time the violation was retrieved, geolocate where it is and update a city wide map highlighting the areas where violations happen.

In addition to that, with the help of third parties (eg. municipality), SafeStreets will be able to retrieve the data given by such party and cross-reference them with its own data retrieved by users. By doing so, it will be possible to identify unsafe areas, assess which kind of problems happen more frequently and suggest possible intervention. It will also be possible for third parties (municipality and police officers) to automatically generate traffic tickets. This will be happening in order to cross-reference all available data in order to build statistics such as the most (or less) egregious offenders or the effectiveness of the SafeStreets initiative

1.1.2 Goals

Follows a list of all goals that will be reached with the SafeStreets initiative.

- ⟨**G0**⟩ The system will allow new customers to automatically sign up and create new accounts with the minimum required ID, email and password (all three will have to be valid);
- ⟨**G1**⟩ The systen will allow customers to sign in, upon check of the validity of email and password the customer inputs;
- ⟨**G2**⟩ The system will allow new traffic violations to be input by users, upon providing a list of required information;
- ⟨**G3**⟩ The system will provide an efficient reporting system in order to highlight different unsafe areas and will allow to filter the shown areas around the municipality;

- ⟨**G4**⟩ The system will allow all customers to modify all account information and to delete the existing account;
- ⟨**G5**⟩ The system will allow third parties to check all violations' details input in the municipality area, connect with an external software for photo forensics purposes and report the validity of the traffic violation in order to emit tickets or discard the violation itself.

1.2 Scope

With SafeStreets users can notify the authorities when traffic violations occur, and in particular parking violations. Both user and authorities must register to the application and agree that SafeStreets stores the information provided, completing it with suitable meta-data. The whole system, because it tracks users information, must respect the standards defined for processing of sensitive information such as GDPR if it is used in Europe. The user sends the type of the violation to the municipality and direct proofs of it (like a photograph). The system runs an algorithm to read the license plate and also asks the user to directly insert the license for a better recognition. Of course other information are required, like the name of the street when the violation has occurred, which can be retrieved from user's direct input or from the geographical position of the violation (using Google Maps API). Both users and authorities can highlight the streets with the highest frequency of violations or the vehicles that commit the most violations. SafeStreets crosses information about the accidents that occur on the territory of the municipality with his own data to identify potentially unsafe areas and suggest possible interventions. Because municipality could generates traffic tickets from the information about violations SafeStreets should guarantee that information is never altered (if a manipulations occurs, the application should discard the information). Such features are made possible trough the use of two mobile applications(one for the citizens and one for the officers on the field). The collected information are sent to a back-end. All the services can also be accessed through a specific web-site.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

User: it is identified as a civilian customer of the product. It will be the main source for the SafeStreets initiative to obtain information about traffic violations and therefore be successful;

Third parties: those kind of organization/company that could provide services useful to SafeStreets and that will be able to retrieve data in order to improve the streets' safety;

Customer: it defines both third party SafeStreets users (police officers or municipality employees) and civilians;

Ghiro: image manipulation detection software, used by third party users in order to detect any image manipulation and assess the veracity of the hard evidence connected to the traffic ticket

1.3.2 Acronym

UI: User Interface

GDPR: General Data Protection Regulation

API: Application Programming Interface

GPS: Global Positioning System

PO: Police Officer

ME: Municipality Employee

1.3.3 Abbreviations

Gn: nth goal;

Dn: nth assumption;

Rn: nth requirement;

ID: identifier (Fiscal Code fomunir Users, a municipality identifier for Third Parties)

1.4 Revision History

1.5 Reference Documents

- Specification document: "Mandatory Project Assignement AY 2019-2010"
- Alloy doc: http://alloy.lcs.mit.edu/alloy/documentation/quickguide/seq.html
- UML diagrams: https://www.uml-diagrams.org/
- Plate Recognizer: https://app.platerecognizer.com
- Ghiro: https://www.getghiro.org/

1.6 Document Structure

Chapter 1 - Introduction Gives an introduction to the problem by describing the purpose of SafeStreets. It also shows the goals and the scope of the application.

Chapter 2 - Overall Description Offers an overall description of the project. It identifies the actors involved in the application and lists all the assumptions in order to identify all the boundaries of the project. The product perspective includes details on the shared phenomena and the domain models. The class diagram describe the domain model used and the state diagrama analyzes:

- The process of collecting violations from users
- The process of sharing informations with the municipality

The majority of functions of the system are more precisely specified by taking in mind the goals of the system.

Chapter 3 - Specific Requirements Contains external interface requirements which are: user interfaces, hardware interfaces, software interfaces and communication interfaces. Few scenarios describing how the system acts in real world are listed here. Furthermore it provides the description of the functional requirements, through the use of use cases and sequence diagrams. The non-functional requirements are defined through performance requirements, design constraints and software system attributes.

Chapter 4 - Formal analysis using Alloy Includes the alloy model of some critical aspects with comments and documentation.

Chapter 5 - Effort Spent Shows the effort spent by each single group member while working on the RASD.

Chapter 6 - References Includes the documents we used as reference.

2 Overall Description

2.1 Product Perspective

afeStreets is designed to be a completely new software applications. It uses some already proven services (like Google Maps and PlateRecognizer APIs) for its critical tasks. The software uses these services in order to double check whether both addresses and license plates are correctly standardized in order to be stored into the violations database.
The system is composed of two different mobile applications: one for the citizens that want to reports violations and one for the officers acting on the field. It also provides a web site for third party users which allows them to assess and analyze potential unsafe areas, thanks also to a powerful reporting system.
Taken into consideration that the municipality could generate traffic tickets from the input violations, the software will be critical when it comes to handling chain of custody. The latter is assured to never be broken by not allowing any kind of customer (user, PO or employee) to modify the reported violation. Supposedly, when some traffic violations might be erroneous or do not have any reason of existence, the inputting user can warn the responsible third party by attaching a warning explaining why it should not be taken into consideration. The systems also ensures the veracity of each violation and the hard evidence attached to it by running a image manipulation detection software (Ghiro, per instance). This process is used by third parties before the emission of each ticket to the corresponding offender.

A high-level class diagram can be found below, which provides a model of the application domain. The most important classes (not all of those which will be implemented once the software will be ready) are shown in order to define how the different components of SafeStreets will be communicating with each other. It is possible to identify two kinds of third party users: police officers and municipality employees. The first ones will be given access to both mobile application and web application; the latter will be provided access just to the above mentioned web application which will help these users assess the veracity of the hard evidence attached to the traffic violations and to further analyze unsafe areas around the municipality.

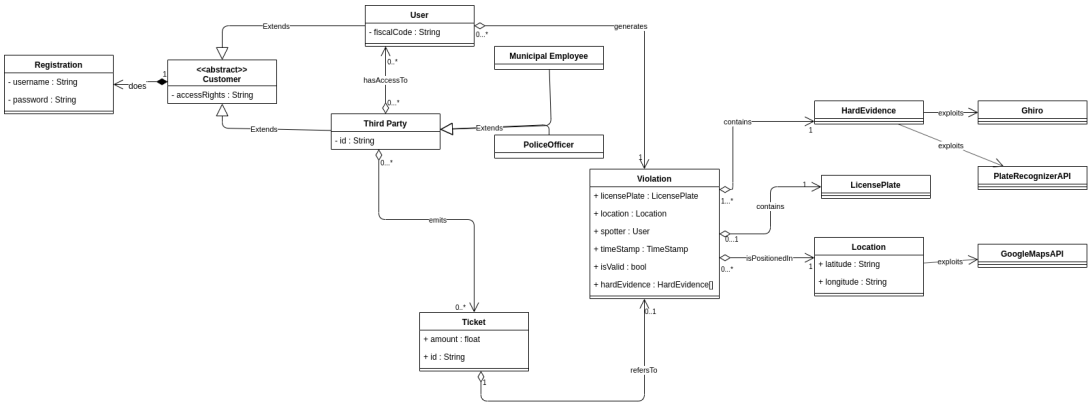

Figure 1: Class diagram

It is possible to notice that there are two actors that model the third party. Police Officers and municipality employees have both access to a web version of the application, since this version provides the tools for checking input violations, emit traffic tickets and assess possible improvements to unsafe areas.

We will now furtherly analize some critical aspects of the system with the use of activity diagrams. Actual system functions will not be presented, just some high level system main processes.

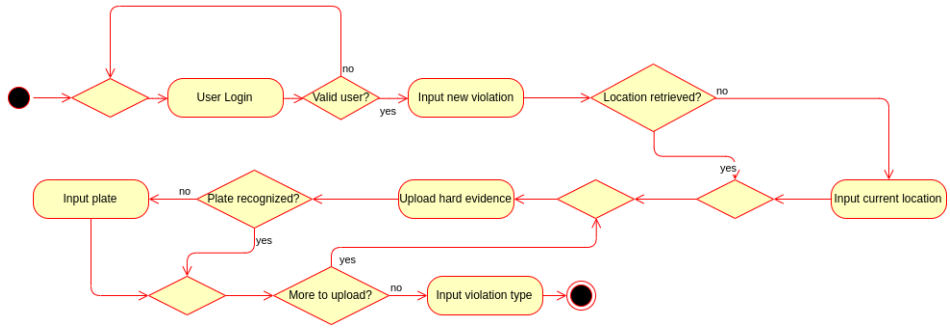

Figure 2: Activity Diagram: Input violation

It is easily understandable that for any unathorized user trying to access the system (those who have not correctly signed up) is not possible to input any new violation. Furthermore, the location retrieving and the plate recognizing processes are a bit tricky. The system tries to execute both of them automatically by calling the respective API from GoogleMaps or PlateRecognizer. If one (or both) processes fail, the system recognizes the problem and asks the user to input the missing piece of information. Once all information is ready,

everything is sent to a shared database and saved for future check by third party employees (PO or ME).

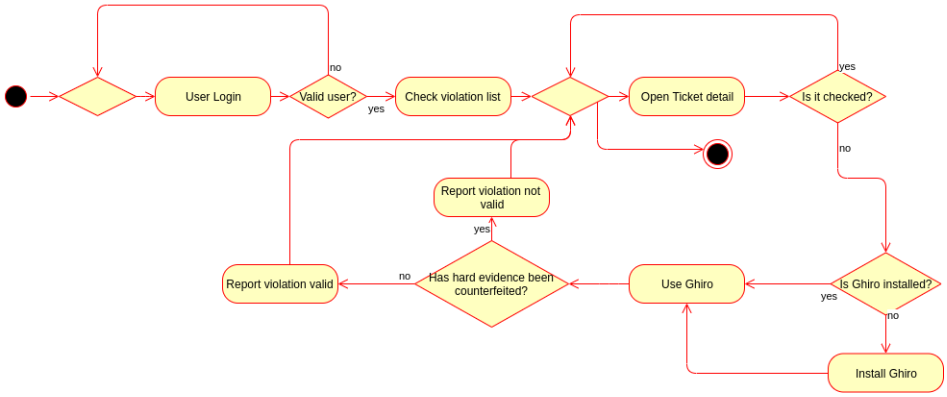

Figure 3: Activity Diagram: Checking violation

As it is mentioned by the description above this one, no user can access the system without a proper username and password. What is interesting in this part of the process, is the possibility to end the whole process only once the traffic violation has been checked, which means that the hard evidence attached to it has to be approved as not being counterfeited or else. A running version of the Ghiro software has to be already installed on the machine which is accessing the system through a web browser; if not, the system itself will automatically redirect to the latest downloadable version of it.

2.2 Product Functions

In the following section the most important product functions of the system are reported.

2.2.1 Reporting traffic violations

This function is the main action that the user can use. After having logged in within its application, pressing on "Report Infraction" the system will ask to insert an image specifying its format and characteristics (clear image, defined context, visible plate). Once the image is loaded, the system will ask him to enter the type of infraction reported, the date and time, as well as the position detected by GPS signal or manually via user input. At this point the user will confirm the data entered and save the report. Moreover he will see in his personal area, under the list of his reports, the new report with all the data he has entered. In addition, the report will be added to the list of reports made in the same municipality.

2.2.2 Police Officer - Reportings Management (APP)

This function allows the police officer to have in real time all the reports made in his municipality of competence. In fact, after logging into his application and providing his identification as a police officer, the application through an analysis system of the latter will identify the municipality of competence of the agent and will show him the list of all the appropriate reports. He can therefore go to the place of the report to resolve the infraction manually by clicking "Resolve Live" and closing the infraction or, in the event that the violation can no longer be assessed in person (i.e. the car has moved) he can leave the report opened to then resolve it from the website.

2.2.3 Police Officer - Reportings Management (WEB)

This function allows the police officer to resolve the reports (i.e. generating a traffic ticket for the owner of the machine) via the website without being physically present on the spot. Once logged in using codes provided by the municipality, the police officer will be able to see all the reports still open (i.e. no one has clicked "Resolve Live") and analyze them. He can take the photo posted by the user and submit it to specialized programs to ensure that it is not tampered with and will therefore be able to generate the ticket and close the report.

2.2.4 Municipal Employee - Interventions Management

This function allows a municipal employee, after logging in through his own identifier, to be able to view all the reports made in the municipality of competence and be able to insert in the application the accident data that occur in the territory. In this way, by pressing the "Start Analysis" button, SafeStreets will identify potentially unsafe areas and show them to the municipal employee and the municipal employee will be able to decide whether to take precautions (i.e. insert barrier between cycle path and road).

2.3 User characteristics

The actors of the application are the following:

- User: a customer of the service who, after having installed the application on his mobile device, having registered through univocal identification data and accepted the processing of his data and the civil and criminal liability of his own reports, can post an infraction report attaching a photo of this (with clear and legible license plate) followed by date, time and position (the latter acquired by GPS detection or manual entry)
- Authority: a customer who uses the system based on its role;
  - Policeman: after installing the application on his mobile device, having registered through univocal data and codes provided by your

municipality, he have the possibility to see the list of reports of infractions made in the Municipality of his competence and also has the ability to access to the system through a website to carry out analysis of the reports (image photoshop) and generate traffic ticket to the accused.

- Municipal Employee: through the website, he has access to the unsafe areas of his own Municipality to decide road interventions to be applied within it
- Municipal Director: through the website, he has access to both reports and traffic tickets both to unsafe areas and to statistical data produced by SafeStreets in order to assess the effectiveness of the application.

2.4 Assumptions, dependencies and constraints

2.4.1 Assumptions

- ⟨**D0**⟩ The device acquires users' location with an error of 5 meters at most.
- ⟨**D1**⟩ Each fiscal code is unique.
- ⟨**D2**⟩ Each authorithies ID is unique.
- ⟨**D5**⟩ The system automatically recognizes the city using the Authorities ID.
- ⟨**D3**⟩ Once a violation has been saved it can't be modified.
- ⟨**D4**⟩ The system is always able to contact to user in order to ask him/her more information.
- ⟨**D5**⟩ If the customer is connected to internet he/she is always able to contact the system.

2.4.2 Dependencies

The system uses external suitable services to make its architecture simpler.

- PlateRecognizer API
- Google Maps API
- Ghiro

2.4.3 Constraints

2.4.3.1 Regulatory policies

SafeStreets uses sensitive data which handling differs from nation to nation. For example, in UE, GDPR provides the guidelines to transmit, store and analyze information.
SafeStreets personalizes the process according to the country in which the services are used.

2.4.3.2 Parallel Operations

The system must provide support for multiple parallel operations from a big number of different users. The user must not wait in order to be able to connect to the service.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

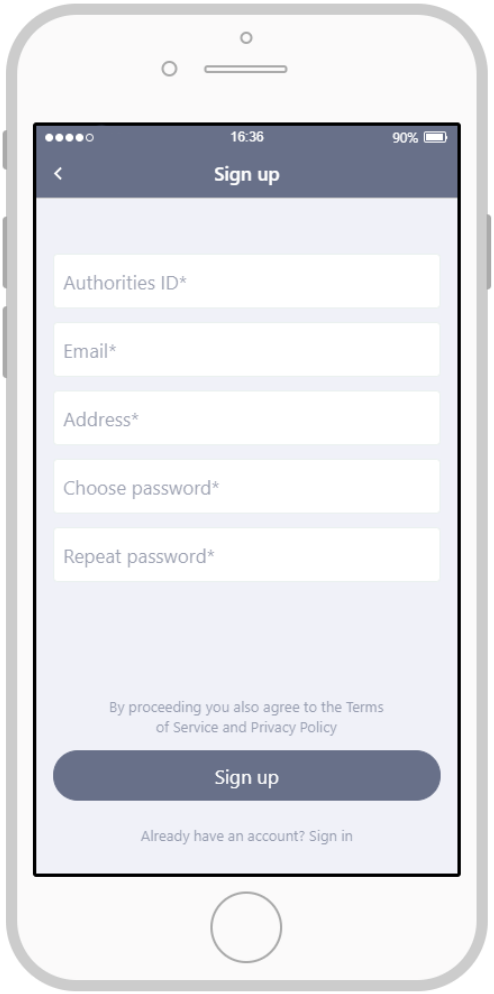

Figure 4: SignUp authorities

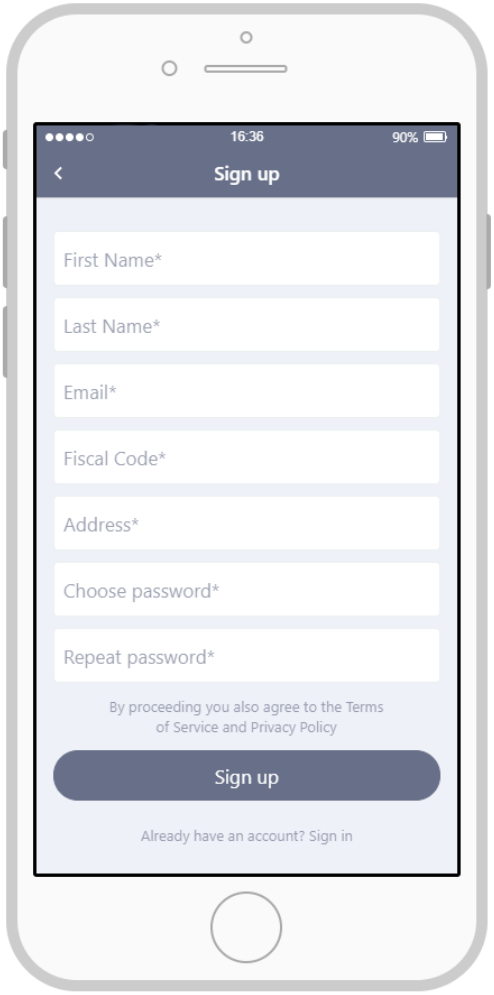

Figure 5: SignUp User

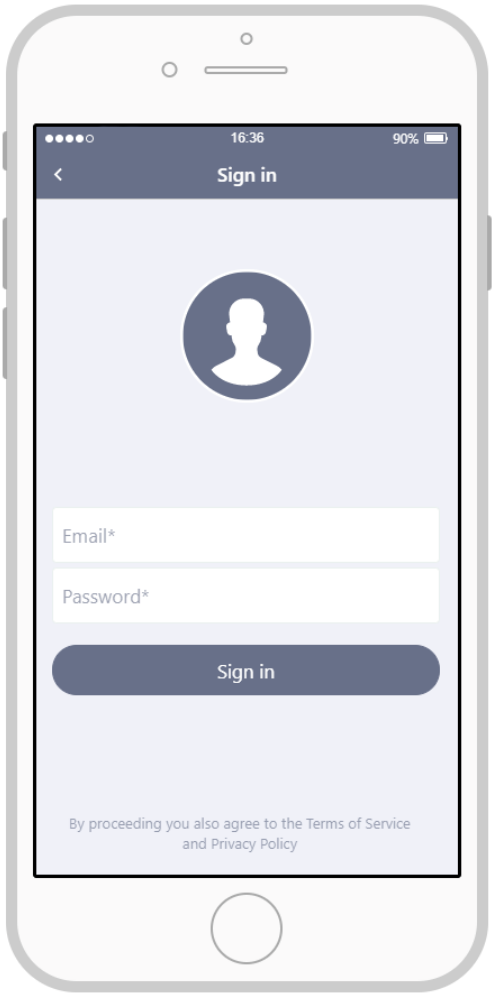

Figure 6: SignIn

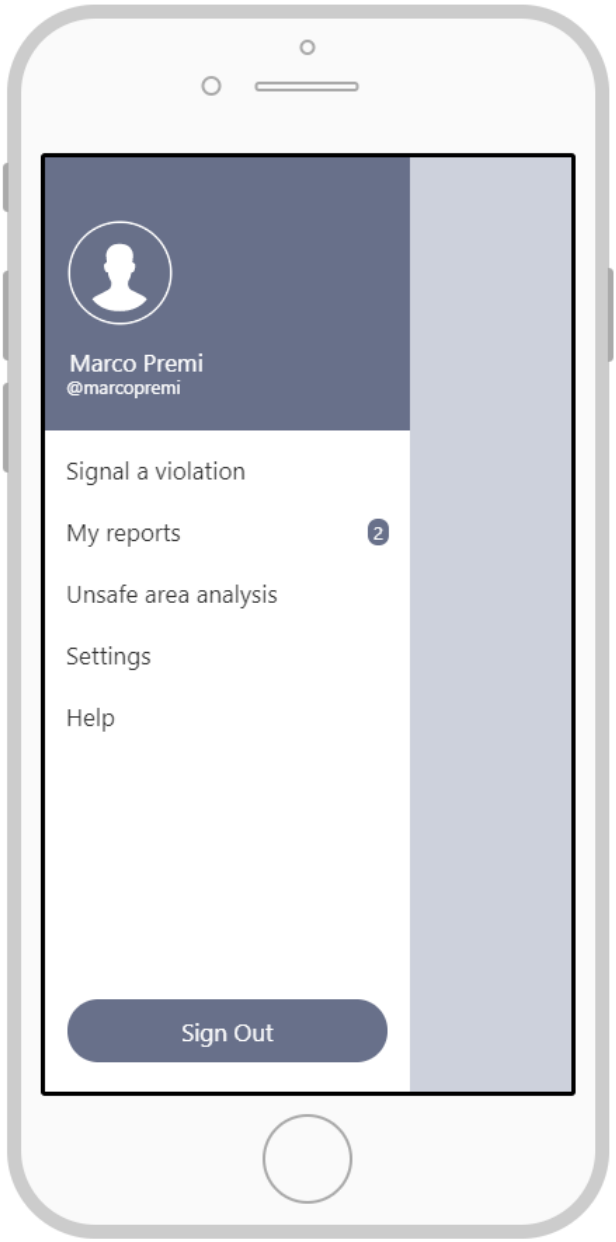

Figure 7: Menu

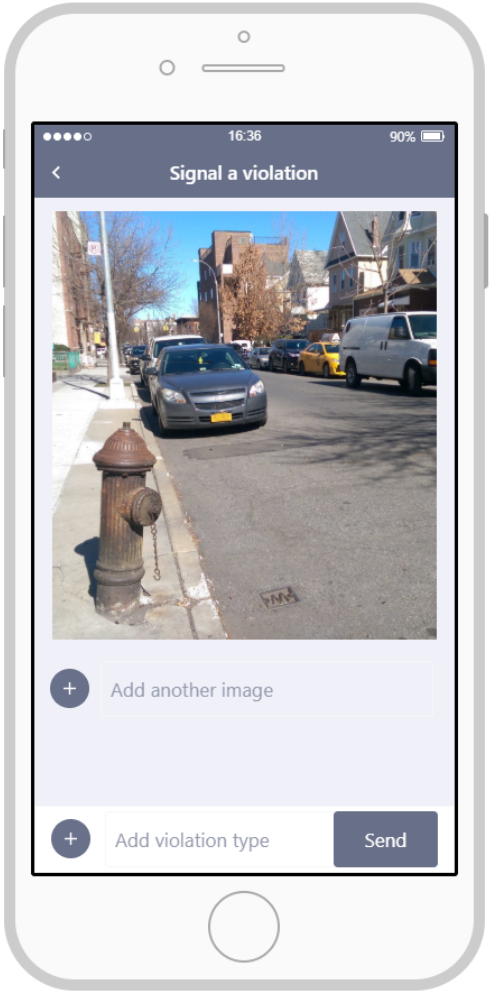

Figure 8: SignIn

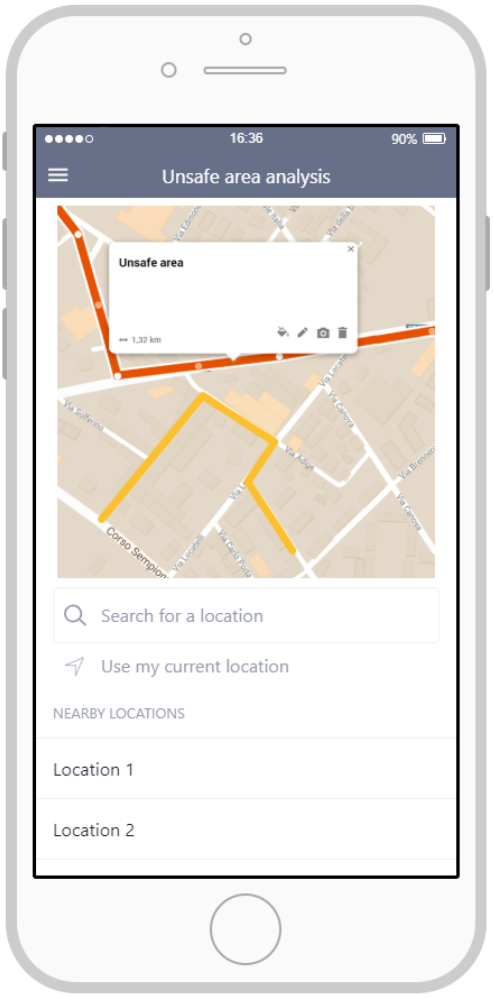

Figure 9: SignIn

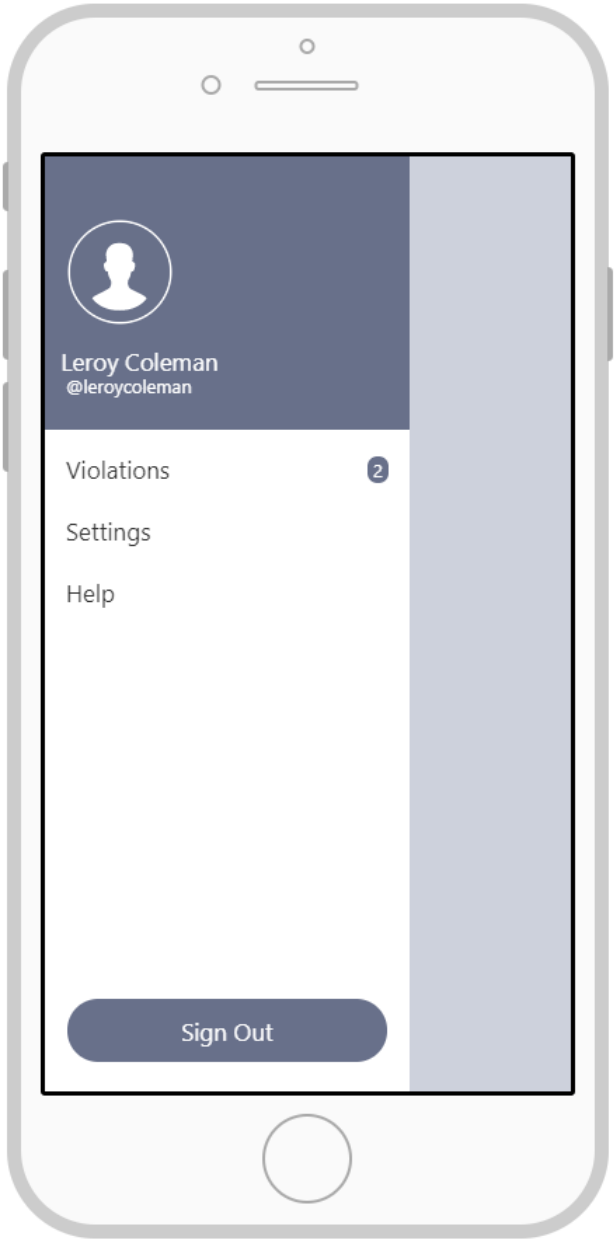

Figure 10: Officer Menu

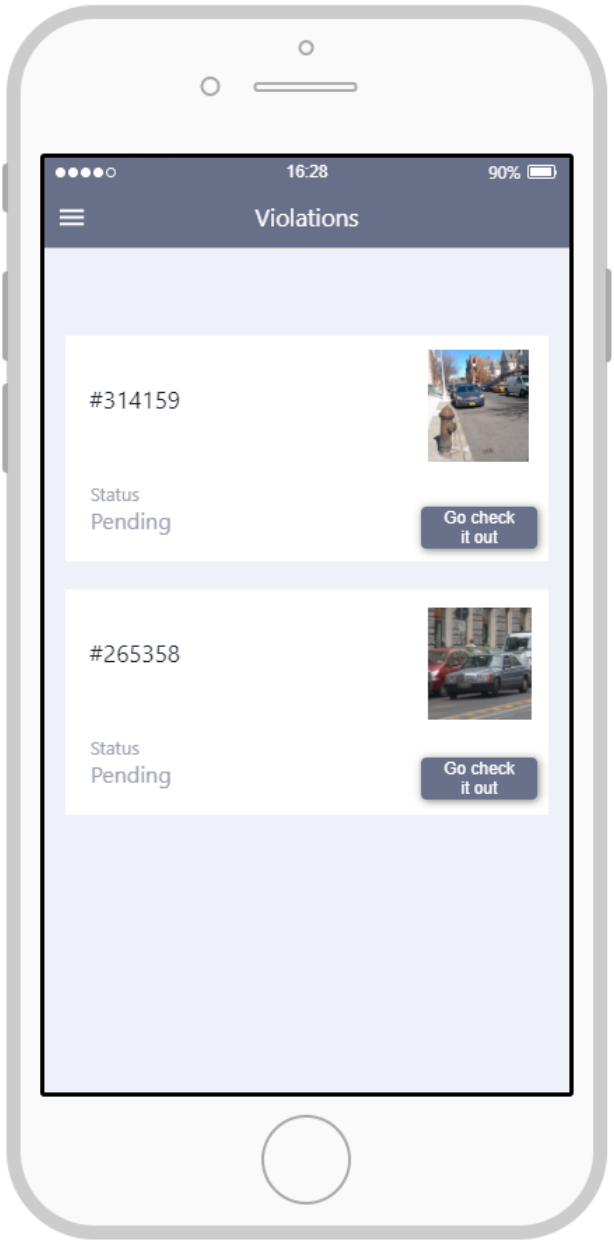

Figure 11: Violations check

Figure 12: SignUp

Figure 13: SignIn

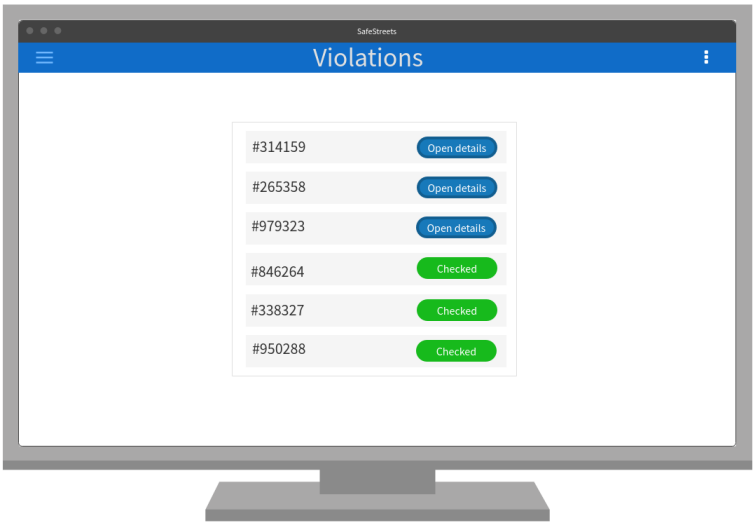

Figure 14: Violations

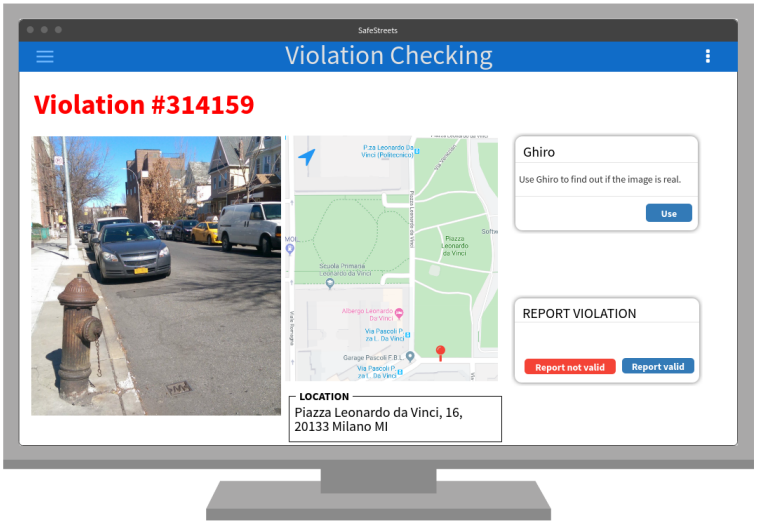

Figure 15: Violations checking

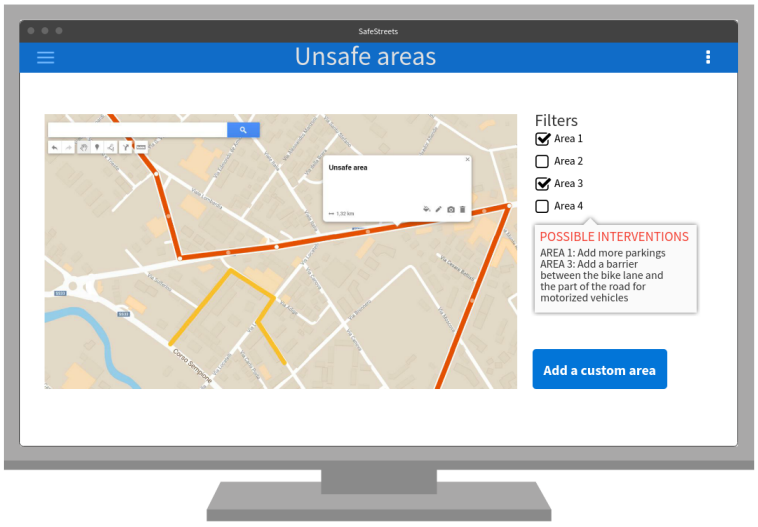

Figure 16: Unsafe Areas

3.1.2 Hardware Interfaces

The system has no hardware interfaces.

3.1.3 Software Interfaces

The system doesn't provide any API to external applications.
However some softwares part of SafeStreets is developed by other companies.

- Google Maps API: for localization and creation of unsafe areas.
- Plate Recognizer API: for License Plate recognition.
- Ghiro: a digital image forensics tool used for find out if the image in the violation report is real.

3.1.4 Communication Interfaces

The system only uses HTTP(more precisely HTTPS) as communication service.
HTTP/HTTPS is used for:

- User and third parties registration.
- Sending violations both to SafeStreets and to authorities.
- Using Google Maps API and Plate Recognizer API

3.2 Scenarios

3.3 Functional requirements

User
⟨**G0**⟩ SignUp to the system

⟨**R1**⟩ the customer must not be already registered in the system

⟨**R2**⟩ the customer must provide a valid ID and email

⟨**R3**⟩ the customer must agree to the Terms of Use

⟨**G1**⟩ SignIn to the system

⟨**R4**⟩ the customer must be already sign up

⟨**R5**⟩ the customer must insert its email and password

⟨**G2**⟩ Signal a violation

⟨**R6**⟩ the user must be able to insert one or more photos of the violation

⟨**R7**⟩ the user must send information about its location

⟨**R8**⟩ the user can add the type of violation that is being reported

⟨**G3**⟩ Show Unsafe Areas

⟨**R9**⟩ the user is shown the unsafe areas around him

⟨**R10**⟩ the customer is allowed to filter the unsafe areas

⟨**R11**⟩ the customer is allowed to search unsafe areas

⟨**G4**⟩ Manage Account

⟨**R12**⟩ the customer must be able to modify its account information

⟨**R13**⟩ the customer must be able to delete his/her account

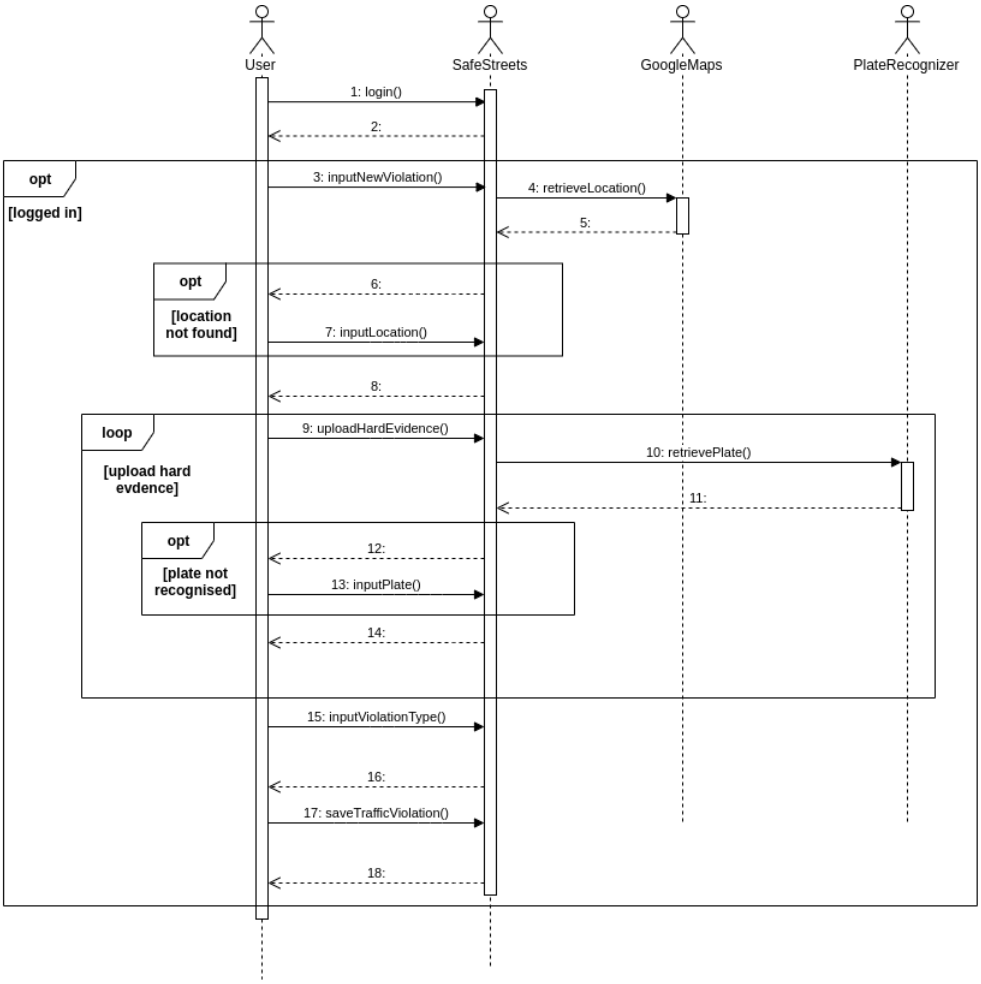

Figure 17: Sequence diagram: input traffic violation

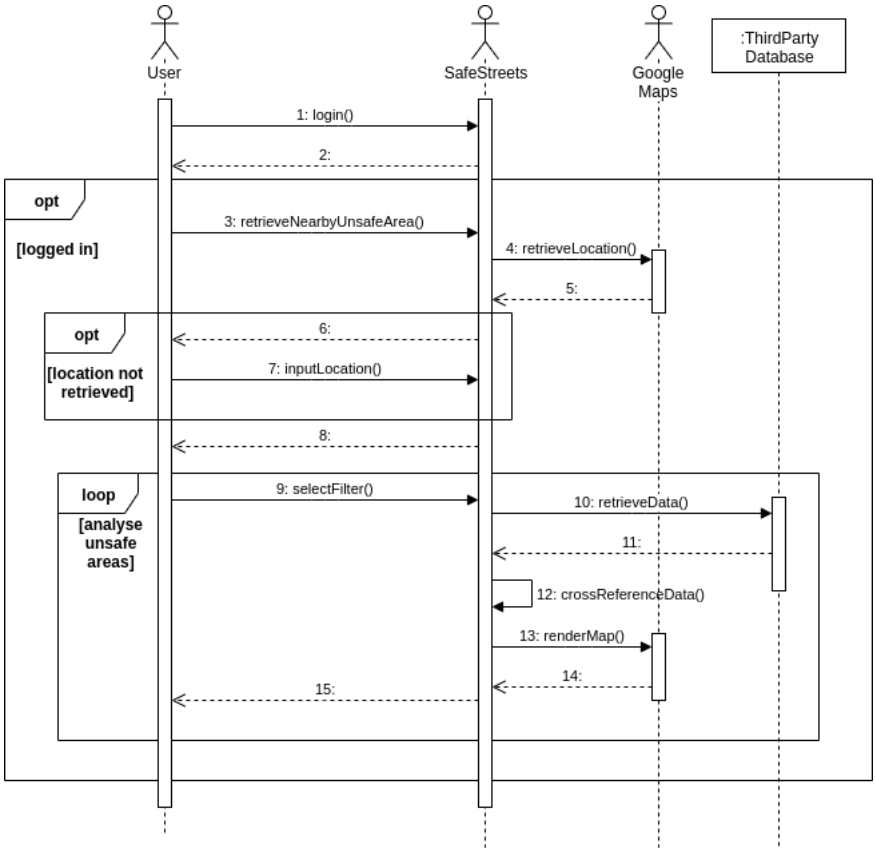

Figure 18: Sequence diagram: analyse unsafe areas

Sequence diagrams

Third parties

⟨**G0**⟩ SignUp to the system

⟨**R1**⟩ the customer must not be already registered in the system

⟨**R2**⟩ the customer must provide a valid ID and email

⟨**R3**⟩ the customer must agree to the Terms of Use

⟨**G1**⟩ SignIn to the system

⟨**R4**⟩ the customer must be already signed up

⟨**R5**⟩ the customer must insert its email and password

⟨**G5**⟩ Check Violations

⟨**R14**⟩ the third party must be able to check all the new and past violations details

⟨**R15**⟩ the third party must be able to use Ghiro

⟨**R16**⟩ the third party must be able to report the validity of the violation

⟨**G3**⟩ Show Unsafe Areas

⟨**R10**⟩ the customer is allowed to filter the unsafe areas

⟨**R11**⟩ the customer is allowed to search unsafe areas

⟨**R17**⟩ the third party is shown a list of possible improvements to be made to the shown areas by the system

⟨**G4**⟩ Manage Account

⟨**R12**⟩ the customer must be able to modify its account information

⟨**R13**⟩ the customer must be able to delete his/her account

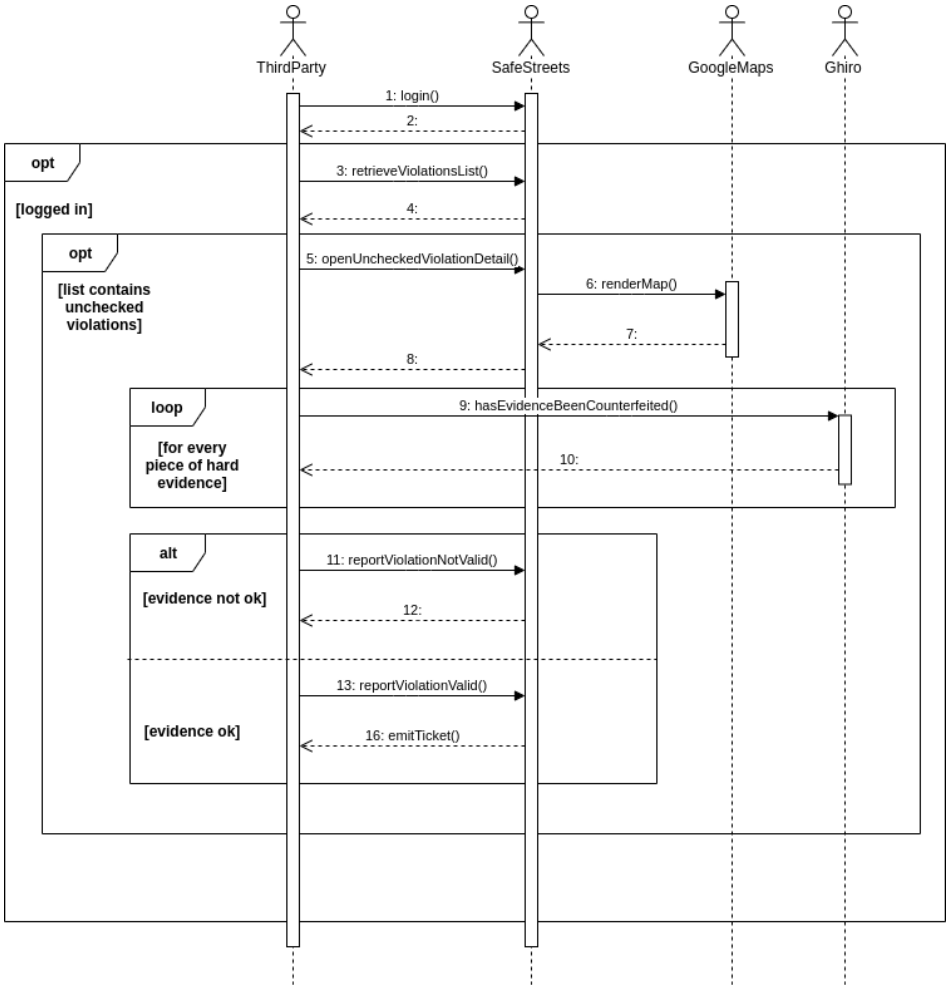

Figure 19: Sequence diagram: check violation and report its validity

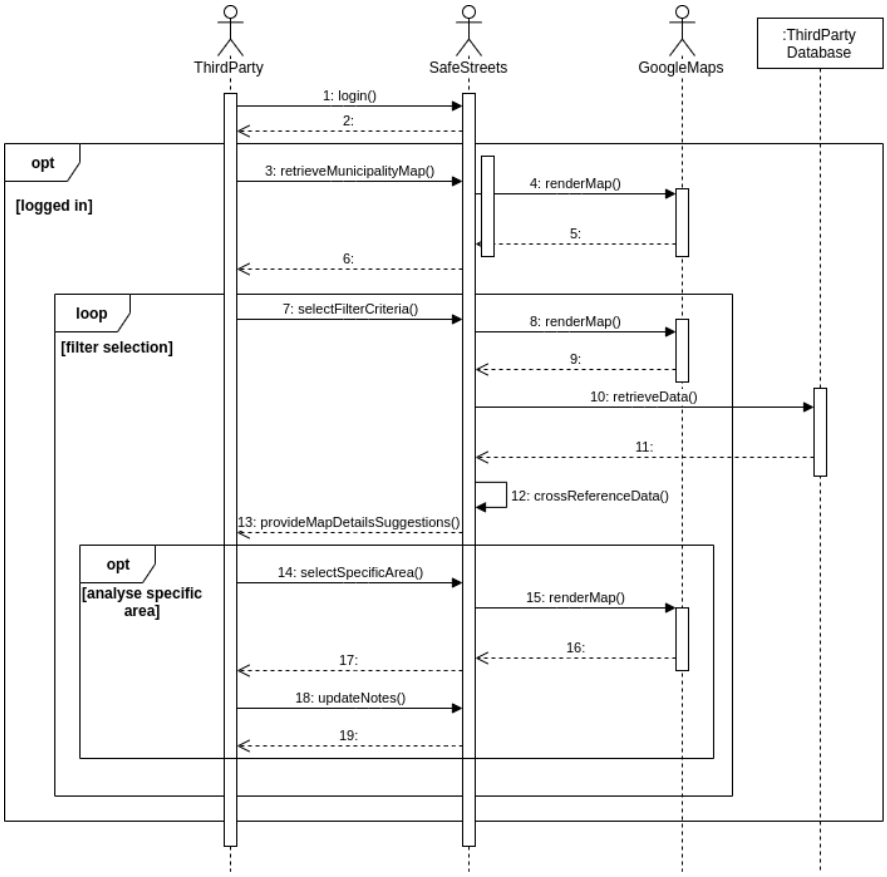

Figure 20: Sequence diagram: analyse unsafe areas

3.4 Use Cases

3.4.1 User use cases

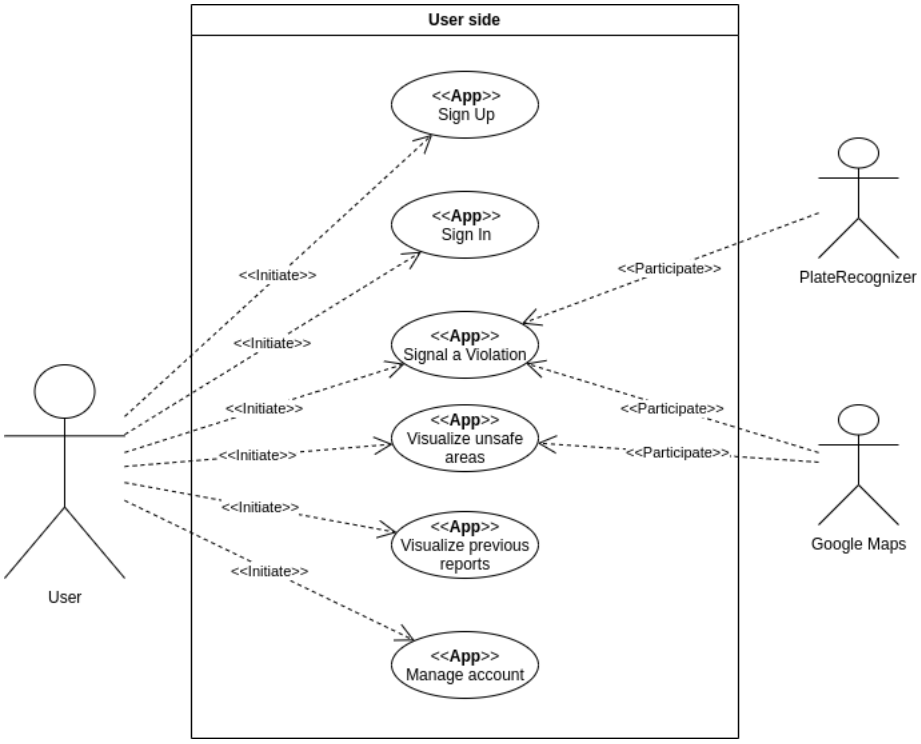

Figure 21: Use Case Diagram: User

Name	**Sign Up**
Actor	User
Entry conditions	The user has downloaded the application on his/her device
Events flow	1. The user opens the application on his/her device 2. The user clicks on "Sign Up" button 3. The user fills the registration form with all the mandatory fields 4. The user clicks the confirmation button 5. The system saves the data
Exit conditions	The system has stored user data, the user is registered and now is able to use te application
Exceptions	1. The user was already signed up 2. The user doesn't fill all the mandatory fields with valid data 3. The email or the fiscal code is already registered 4. The user closes the application before the process has ended

Name	**Sign In**
Actor	User
Entry conditions	1. The user has already downloaded the the application on his/her device 2. The user has already signed up
Events flow	1. The user opens the application on his/her device 2. The user inserts his/her credentials in the "Email" and "Password" fields. 3. The user clicks on the "Sign In" button
Exit conditions	The user is successfully signed in
Exceptions	1. The user inserts invalid Email 2. The user inserts invalid Password 3. The user closes the application before the process has ended

Name	**Signal a violation**
Actor	User
Entry conditions	The user has already logged in
Events flow	1. The user opens the Menu 2. The user clicks on "Signal a violation" button in the Menu 3. The user uploads one or more photos 4. The user adds additional information 5. The user adds one or more violation types 6. The user clicks "Send" 7. SafeStreets receives the violation
Exit conditions	The user has successfully reported a violation
Exceptions	1. The user closes the application before the process has ended 2. The user doesn't have internet connection

Name	**Visualize unsafe areas**
Actor	User
Entry conditions	The user has already logged in
Events flow	1. The user opens the Menu 2. The user clicks on the "Unsafe area analysis" button in the Menu 3. The user is allowed to select different filters and the area he/she wants to see
Exit conditions	The user can see all the unsafe areas proposed by SafeStreets and the third party
Exceptions	The user doesn't have internet connection

Name	**Visualize previous reports**
Actor	User
Entry conditions	The user has already logged in
Events flow	1. The user opens the Menu 2. The user selects the "My reports" button in the Menu 3. The user is allowed to see all the previous reports he/she made
Exit conditions	The user is provided with the requested data
Exceptions	The user doesn't have internet connection

Name	**Manage Account**
Actor	User
Entry conditions	The user has already logged in
Events flow	1. The user opens the Menu 2. The user selects the "Settings" button in the Menu 3. The user is allowed to change his/her adress, the password, the email or to delete the account
Exit conditions	New user settings are saved to his/her account or the account is deleted
Exceptions	The user closes the application before the process has ended

3.4.2 Third party use cases

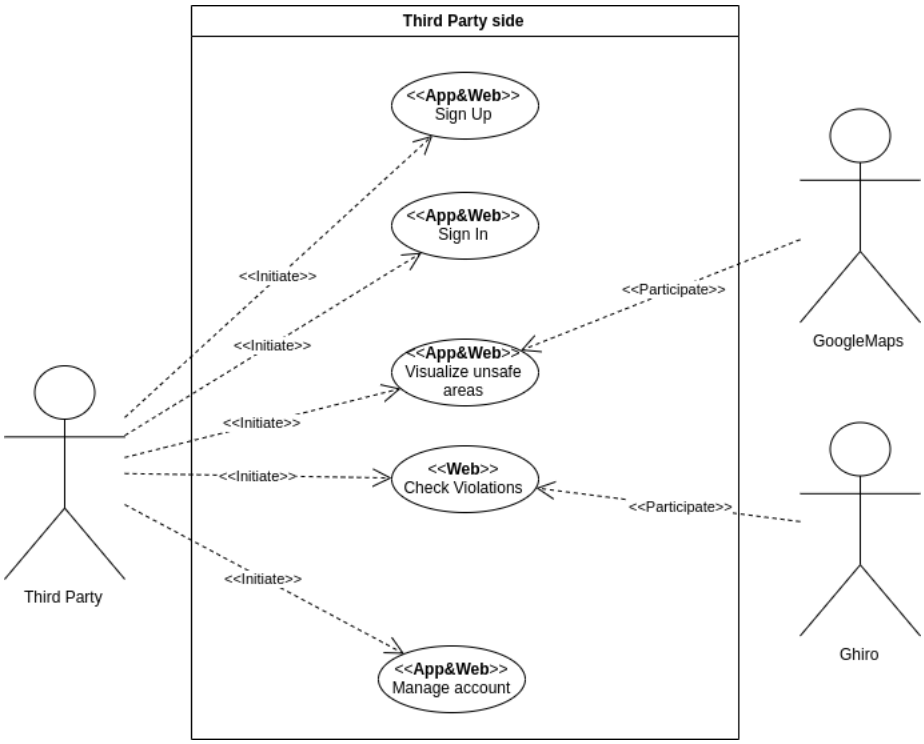

Figure 22: Use Case Diagram: Third Party

Name	**Sign Up**
Actor	Third party
Entry conditions	The third party is not already registered
Events flow	1. The third party accesses SafeStreets website 2. The third party is asked to sign in 3. The third party clicks on "Sign Up" 4. The third party fills the registration form with all the mandatory fields 5. The third party clicks the confirmation button 6. The system saves the data
Exit conditions	The system has stored third party data, the third party is registered and now is able to use the website
Exceptions	1. The third party was already signed up 2. The third party doesn't fill all the mandatory fields with valid data 3. The email or the AuthoritiesID is already registered 4. The third party closed the application beforse the process has ended

Name	**Sign In**
Actor	Third party
Entry conditions	The third party has already signed up
Events flow	1. The third party accesses SafeStreets website 2. The third party is asked to sign in 3. The third party inserts the credentials in the "Email" and "Password" fields. 4. The third party clicks on "Sign In"
Exit conditions	The third party is successfully signed in
Exceptions	1. The third party inserts invalid Email 2. The third party inserts invalid Password 3. The third party closes the website before the process has ended

Name	**Unsafe Areas**
Actor	Third party
Entry conditions	The third party has already logged in
Events flow	1. The third party clicks on the menu icon 2. The third party click on "Unsafe ares" 3. The third party is allowed create and delete unsafe areas, to select different filters and to search different areas
Exit conditions	The unsafe areas editor is presented to the third party
Exceptions	

Name	**Check Violations**
Actor	Third party
Entry conditions	The third party has already logged in
Events flow	1. The third party clicks on the menu icon 2. The third party clicks on "Violations" 3. The third party is allowed to see the list of previous and actual violations (checked or not) 4. The third party click on "Open Details" of one particular not checked violation 5. The third party is allowed to see information about the particular violation 6. The third party can click "Use" on Ghiro card to find out if the image is real 7. The third party can click "Report not valid" if the image is fake or "Report valid" if the image is real
Exit conditions	The third party can manage the violations
Exceptions	The third party closes the website before the process has ended

Name	**Manage Account**
Actor	Third party
Entry conditions	The third party has already logged in
Events flow	1. The third party clicks on the menu icon 2. The third party click on "Manage Account" 3. The third party is allowed to change the email, the password or to delete the account
Exit conditions	New user settings are saved to the third party account or the account is deleted
Exceptions	The third party closes the website before the process has ended

Name	**Check violations on the field**
Actor	Third party(officer)
Entry conditions	The third party has already logged in
Events flow	1. The third party opens the Menu 2. The third party clicks on "Violations" button in the Menu 3. The third party is allowed to see the list of violations 4. The third party clicks on "Go check it out" button 5. SafeStreets opens Google Maps or one other turn-by-turn navigation app to reach the violation location
Exit conditions	The officer can reach the violation location
Exceptions	1. The officer closes the application before the process has ended 2. The officer doesn't have internet connection

3.5 Performance requirements

The system is provided to serve a great number of users and third parties simultaneously. The back-end must be powerful enough to accept thousands of requests at same time during all the day.
The front-end applications (mobile and web) don't have particular performance requirements.

3.6 Design Constraints

3.6.1 Standars compliance

With regard to the privacy, security for the mobile application and the back-end is a big issue, so the whole project is subject to the the GDPR. Furthermore it's a good practice to apply W3C's Standards to ensure intercompatibility.

3.6.2 Hardware Limitations

Even if SafeStreets is a software-based service there are some hardware limitations regarding the smartphones.

- must be able to make HTTPS requests (connection to internet 4G/3G/2G/Wi-Fi)
- must have on board GPS (mobile devices only)

3.6.3 Any Other Constraint

3.7 Software System Attributes

3.7.1 Reliability

The system must be able to run 24/7 without any interruptions. Only small concessions from this requirement might be tolerated. The system must also be fault tolerant. In order to exploit this last point the data saved on the servers should be duplicated.

3.7.2 Availability

Maximum availability is a necessity because users should be able to report violations 24/7. One or two days per year may be used for maintainance, so availability of 99% is expected.

3.7.3 Security

User and third party provide data that contains sensitive information, so the security aspect is of primary relevance.
Communications between user, third party and SafeStreets must be encrypted and the database with all the data must be protected from any possibile internal or external attack.

3.7.4 Mantainability

The system shoule be flexibile and easy to mantain. For this purpose the whole system will be developed using a modular architecture.
Fixing and modifying the system will be easy with the modular architecture.

3.7.5 Portability

SafeStreets in order to be useful must be used by a very large number of people. It is for this reason that third parties' registration website must be available on the majority of browsers (Firefox, Google Chrome, Safari, Opera, Microsoft Edge and Internet Explorer 11).
Naturally the mobile application must be available on both Android and iOs.

4 Formal Analysis using Alloy

```
abstract sig Bool{}

one sig True extends Bool{}

one sig False extends Bool{}

sig FiscalCode{}

sig ThirdPartyId{}

sig Username{}

sig Password{}

sig Registration{
    username: one Username,
    password: one Password
}

abstract sig AccessRights{}

--This right allows to see the list of my reports (User
    ↪ right)
one sig MySignalViolations extends AccessRights{}

--This right allows to see unsafe areas (User and Third
    ↪ Parties right)
one sig UnsafeAreaAnalysis extends AccessRights{}

--This right allows to make a report (User's right)
one sig SignalViolation extends AccessRights{}

--This right allows to see all violations (Police Officer
    ↪  and Municipal Director right)
one sig CheckViolations extends AccessRights{}


--This right allows to validate signalations (Police
    ↪ Officer right)
one sig ValidateSignalViolations extends AccessRights{}

--This right allows to see statistics (Municipal Director
    ↪  right)
```

```
one sig Statistics extends AccessRights{}

abstract sig Customer{
    registration: one Registration,
    accessRights: set AccessRights,
}

sig User extends Customer{
    fiscalCode: one FiscalCode,
    mySignalations: set Signalation
}

--The system automatically assigns the municipality to
    ↪ the third party based on its Id
abstract sig ThirdParty extends Customer{
    id: one ThirdPartyId,
    municipal : one Municipality
}

sig Municipality{
    violations: set Violation
}

sig Signalation{
    violation: one Violation,
    spotter: one User,
}

sig PoliceOfficer extends ThirdParty{
    listSignalations: set Signalation
}

sig MunicipalEmployee extends ThirdParty{
}

sig MunicipalDirector extends ThirdParty{
    listViolations: set Violation
}

sig Location{
    latitude: one Int,
    longitude: one Int
}
--{latitude >= -90 and latitude <= 90 and longitude >=
    ↪ -180 and longitude <= 180}
--NB: scaled valued for simplicity
```

```
{latitude ≥ -3 and latitude ≤ 3 and longitude ≥ -6 and
    ↪ longitude ≤ 6}

sig LicensePlate{}

sig TimeStamp{}

sig Violation{
    id: one Int,
    licensePlate: one LicensePlate,
    location: one Location,
    spotter: one User,
    timestamp: one TimeStamp,
    isValid: one Bool,
    municipality: one Municipality
}
{id ≥ 0}
--'isValid' is put on 'True' after a check of police
    ↪ officer

sig Ticket{
    amount: one Int,
    signalation: one Signalation,
    policeOfficer: one PoliceOfficer
}
{amount ≥0}

--
    ↪ ------------------------------------------------------------------------
    ↪

--All Registrations have to be associated to one Customer
fact RegistrationCustomerConnection{
    all r : Registration | some c : Customer | r in c.
        ↪ registration
}

--The Registration has a unique Customer
fact NoSameRegistration{
    no disj c1,c2 : Customer | c1.registration = c2.
        ↪ registration
}

--All Usernames have to be associated to a Registration
fact UsernameRegistrationConnection{
```

```
    all u : Username | some r : Registration | u in r.
        ↪ username
}

--All Passwords have to be associated to a Registration
fact PasswordRegistrationConnection{
    all p : Password | some r : Registration | p in r.
        ↪ password
}

--Every Customer has a unique username
fact NoSameUsername{
    no disj c1,c2 : Customer | c1.registration.username =
        ↪  c2.registration.username
}

--All FiscalCodes must be associated to a User
fact FiscalCodeUserConnection{
    all f : FiscalCode | some u : User | f in u.
        ↪ fiscalCode
}

--All ThirdPartyId must be associated to a ThirdParty
fact IdThirdPartyConnection{
    all i : ThirdPartyId | some t : ThirdParty | i in t.
        ↪ id
}

--The Fiscal Code can be associated to only one user
fact OneUserFiscalCode{
    no disj u1,u2 : User | u1.fiscalCode = u2.fiscalCode
}

--The id can be associated to only one third party
fact OneThirdPartyUserId{
    no disj t1, t2 : ThirdParty | t1.id = t2.id
}

--All LicensePlates have to be associated to a Violation
fact LicenseViolationConnection{
    all l : LicensePlate | some v : Violation | l in v.
        ↪ licensePlate
}

--All Timestamps have to be associated to a Violation
fact TimestampViolationConnection{
```

```
    all t : TimeStamp | some v : Violation | t in v.
        ↪ timestamp
}

--All Violations have to be associated to a Signalation
fact ViolationSignalationConnection{
    all v : Violation | some s : Signalation | v in s.
        ↪ violation
}

--Only one id per violation, no replicas!
fact OneIDViolation{
    no disj v1,v2 : Violation | v1.id = v2.id
}

--It is not possible to have two different locations with
    ↪ the same plate and timestamp
fact SamePlateLocationAndTimestamp {
    all v1,v2 : Violation |
    v1.location = v2.location implies (v1.licensePlate =
        ↪ v2.licensePlate and v1.timestamp = v2.timestamp
        ↪ )
}


fact PoliceOfficerRights{
    all p : PoliceOfficer | CheckViolations in p.
        ↪ accessRights and ValidateSignalViolations in p.
        ↪ accessRights and UnsafeAreaAnalysis in p.
        ↪ accessRights
}

fact UserRights{
    all u : User | MySignalViolations in u.accessRights
        ↪ and UnsafeAreaAnalysis in u.accessRights and
        ↪ SignalViolation in u.accessRights
}

fact MunicipalEmployeeRights{
    all me : MunicipalEmployee | UnsafeAreaAnalysis in me
        ↪ .accessRights
}

fact MunicipalDirectorRights{
    all md : MunicipalDirector | UnsafeAreaAnalysis in md
        ↪ .accessRights and CheckViolations in md.
```

```
        ↪ accessRights and Statistics in md.accessRights
}


--Police Officer sees all the violations of his
    ↪ Municipality of competence
fact PoliceSeeViolations{
    all p : PoliceOfficer | all s : Signalation | p.
        ↪ municipal = s.violation.municipality implies s
        ↪ in p.listSignalations
}

--Police Officer sees only the violations of his
    ↪ municipality of competence
fact PoliceDontSeeViolations{
    all p: PoliceOfficer | all s: Signalation | p.
        ↪ municipal ≠ s.violation.municipality implies s
        ↪ not in p.listSignalations
}

--Municipal Director sees all the violations of his
    ↪ municipality of competence
fact MunicipalDirectorSeeViolations{
    all md : MunicipalDirector | all v : Violation | md.
        ↪ municipal = v.municipality implies v in md.
        ↪ listViolations
}

--Municipal Director sees only the violations of his
    ↪ municipality of competence
fact MunicipalDirectoreDontSeeViolations{
    all md : MunicipalDirector | all v : Violation | md.
        ↪ municipal ≠ v.municipality implies v not in md.
        ↪ listViolations
}

--All Signalations are referred to one User
fact SignalationCorrespondToOneUser{
    all s : Signalation | all u : User | (s in u.
        ↪ mySignalations implies s.spotter = u) and (s.
        ↪ spotter = u implies s in u.mySignalations)
}

--Each Violation is referred to only one Municipality
fact ViolationOneMunicipality{
```

```
    all v : Violation | all disj m1,m2 : Municipality | v
        ↪ .municipality = m1 implies v not in m2.
        ↪ violations
}

--All Violations are referred to Municipality
fact ViolationsOfMunicipality{
    all v : Violation | all m : Municipality | v.
        ↪ municipality = m implies v in m.violations
}

--All Tickets are referred to a Signalation of the
    ↪ Municipality of the Police Offer who erogates the
    ↪ ticket
fact TicketSameMunicipalityPoliceOfficer{
    all t : Ticket | t.signalation.violation.municipality
        ↪  = t.policeOfficer.municipal
}

--All Tickets are referred to a Signalation of a valid
    ↪ Violation
fact TicketsForValidViolation{
    all t : Ticket | t.signalation.violation.isValid =
        ↪ True
}

--All Municipalities are referred to a Third Party or
    ↪ Violation
fact MunicipalityToThirdPartyOrViolation{
    all m : Municipality | some t : ThirdParty | m in t.
        ↪ municipal
}

--
    ↪ ------------------------------------------------------------------------
    ↪

--Different Police Officer of the same Municipality see
    ↪ the same violations
assert DifferentOfficerSameViolations{
    all disj p1, p2 : PoliceOfficer | all s1 : p1.
        ↪ listSignalations | all s2 : p2.listSignalations
        ↪  | p1.municipal = p2.municipal
    implies (s1 in p2.listSignalations and s2 in p1.
        ↪ listSignalations) else (s1 not in p2.
        ↪ listSignalations and s2 not in p1.
```

```
        ↪ listSignalations)
}

check DifferentOfficerSameViolations for 10

--All the signalations are present in the list of
    ↪ violations of the Police Officer
assert SignalationPresentInTheList{
    all s : Signalation | all p : PoliceOfficer | s.
        ↪ violation.municipality = p.municipal implies s
        ↪ in p.listSignalations
}

check SignalationPresentInTheList for 10

--All Tickets refer to a signalation of the list
    ↪ signalations of the Police Officer
assert TicketFromPoliceOfficer{
    all t : Ticket | all s : Signalation | some p :
        ↪ PoliceOfficer | s in t.signalation implies s in
        ↪  p.listSignalations
}
check TicketFromPoliceOfficer for 10

assert TicketOnlyTrueViolation{
    no t : Ticket | t.signalation.violation.isValid =
        ↪ False
}
check TicketOnlyTrueViolation for 10

pred world1{
    #PoliceOfficer ≥ 3
    #User = 2
    (some disj p1,p2,p3 : PoliceOfficer | p1.municipal =
        ↪ p2.municipal and p1.municipal ≠ p3.municipal)
}
run world1 for 10 but 0 Ticket, 0 Bool, 0 Signalation, 0
    ↪ Violation, 0 AccessRights

/*
From the first world in Figure ? it can be noticed that
    ↪ every Police Officer has a different Id, every User
    ↪  has a different
FiscalCode and every Username is different from another.
    ↪ Both Users and Third Parties can share the password
    ↪  and there can be
```

```
many Police Officer in the same Municipality, same
    ↪ analogy is for MunicipalDirector and
    ↪ MunicipalEmployee
*/

pred world2{
    #PoliceOfficer = 2
    #Signalation = 2
    #Ticket ≥ 1
    one s : Signalation | s.violation.isValid = False
    one s : Signalation | s.violation.isValid = True
}
run world2 for 10 but 0 AccessRights

/*
From the second world in Figure ? it can be noticed that
    ↪ there are two Violations, one it's a real violation
    ↪  that it doesn't tampered
(so it has True as 'isValid') and the other one it's a
    ↪ fake violation that it has tampered (so it has
    ↪ False as 'isValid'). The ticket
genereated by the Police Officer is for only the real
    ↪ violation (omitting all the checks that led to '
    ↪ true') and not for the other one.
Moreover, the ticket is generated only one time (in this
    ↪ case by PoliceOfficer0 and not also by
    ↪ PoliceOfficer1 of the same Municipality)
*/

pred world3{
    #PoliceOfficer = 2
    #User = 2
    #MunicipalDirector = 1
    #MunicipalEmployee = 2
}
run world3 for 10 but 0 Ticket, 0 Bool, 0 Signalation, 0
    ↪ Violation

/*
From the third world in Figure ? it can be noticed that
    ↪ every PoliceOfficer have the same accessRights as
    ↪ well as all the Users have the
same accessRights and so on. Classes of accessRights have
    ↪  been created for a reason of granularity: in this
    ↪ way the accessRight can change
```

```
in the future or can be added and the application keeps
    ↪ track of the privilege of everyone.
*/

pred world4{
    #PoliceOfficer = 2
    #MunicipalDirector = 1
    #User.mySignalations ≥1
    all disj p1,p2 : PoliceOfficer | one d :
        ↪ MunicipalDirector | all s : Signalation | p1.
        ↪ municipal = p2.municipal and p1.municipal = d.
        ↪ municipal
    and (s in p1.listSignalations implies (s in p2.
        ↪ listSignalations))
    and (s in p1.listSignalations implies (s.violation in
        ↪  d.listViolations))
    and (s in p2.listSignalations implies (s in p1.
        ↪ listSignalations))
    and (s.violation in d.listViolations implies (s in p1
        ↪ .listSignalations))
}
run world4 for 10 but 0 Ticket
/*
From the fourth world in Figure ? it can be noticed that
    ↪ every signalation is present on the list of
    ↪ signalation of every PoliceOfficer of
the same Municipality and the MunicipalDirector sees the
    ↪ violations in his list of violations of all the
    ↪ signalations of his Municipality of
competence
*/
```

5 Effort spent

Description of the task	MP	FS	GT
Introduction	2.5	2	0
Overall Description	3	8	2
Specific requirements	0	4	0
Formal analysis using Alloy	0	0	10