Politecnico di Milano

Software Engineering II project

SafeStreets

Requirements Analysis and Specifications Document

Authors:
Matteo Pacciani
Francesco Piro

Professor:
Matteo Giovanni Rossi

November 10, 2019

version 1.0

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This document completely describe the system in terms of functional and non-functional requirements and is used as a contractual basis between the customer and the developer. The structure follows the template studied at the lessons and is integrated with specific documents relative in particular to: the template used [2], the requirements engineering process [1] and the categorization of the phenomena [4]. Hence this document is the result of the requirements elicitation and the analysis activities paired with a specific description aimed to precise the behavior of the system.

1.2 Context

SafeStreets is a *crowd-sourced* application that intends to provide a way to notify authorities when a traffic violation occurs. Nowadays almost everyone uses a device for most of the activities and this gives the opportunity to achieve products whose knowledge is based on the concept of *outsourcing*. Thanks to the system we are going to describe, in fact, each person will be able to report a violation to the authority in a very easy and fast way. The big amount of data SafeStreets expects to receive allows also to provide additional functionalities to its customers relative to statistics and important considerations about safety. However the main purpose of the application still remains to come up with a simple and effective way to report traffic violations, a problem always more important in particular in big cities where authorities are unable to deal with all the infractions that may occur. Without loss of generality we will consider the system to work only in Italy, this assumption is made in particular to obtain a way to recognize authorities as such using a technology already in use (PEC). The correct recognition of the authorities allows also to manage correctly one of the most difficult problem that SafeStreets has to deal with: the *law*. In this way we accomplish both the problem of the security of the data relative to the violations and the one related to the correctness about the violations reported by the users.

1.3 Scope

The aim of the system is to achieve with outsourcing a simple way to notify traffic violations and use this information to retrieve interesting data for both its customers: users and authorities. Hence we can think as the main objective of SafeStreets the one of the notification and then describe the other functionalities in the basic and advanced services.

The system will describe the notification functionalities in terms of *parking violations* but is thought for further extensions in order to deal with the other types. Users are the clients allowed to report a violation and authorities will be notified whenever a new one will be reported. This process takes place entirely inside the application where both customers, once recognized, will be able to benefit only of the services related to their role. As we have already said SafeStreets is a crowd-sourced application; thanks to this property the big amount of data that is going to be managed will be used to provide additional functionalities based on the processes of mining and crossing this information. We call the *basic functionality* the one related to the data mining used to retrieve statistical data interesting for both the users and authorities. It is important to highlight that this functionality considers the role of each customer in order to provide him a way for retrieving the data with a precise level of visibility. As completely described in the section related to the functionalities provided by the application (2.2) the authorities in fact benefit of an additional service that allows them to retrieve the information about the parking violations in the most detailed way (for example with the entire description of an infraction that contains also the data about who reported it and the plate of the vehicle). The *advanced functionality* instead is related to the concept of **safety** that can be attributed to a street. This functionality is based on the crossing process between the data stored in SafeStreets system related to the violations and the one of the accidents possibly provided by a municipality. The safety of each street our application is able to retrieve is strictly bound with an additional service that wants to find a suggestion for a possible intervention for a street that has been marked as *unsafe*.

Now that we have a first description of what SafeStreets aims to achieve we are able to have a look to the **goals** that have been chosen in order to accomplish these functionalities. Further in the document we will have the precise definition of each goal with the prove of their satisfaction described in terms of requirements and assumptions (section 3.2). The description will be carried on first with the identification of the phenomena (section 2.5) and then with the additional strict relation with the part that describes the requirements used for the goals and the use case diagrams (section 3.3.3 and traceability matrix 3.3.4).

1.3.1 Goals

- **G1** Users should be able to notify authorities when traffic violations occur, in particular parking violations.
- **G2** Users and authorities should be able to mine the information stored by SafeStreets, with different levels of visibility.
  - **G2A** Users and authorities should be able to know where the highest number of violations occur.
  - **G2B** Users and authorities should be able to know what types of vehicle make the most violations.
  - **G2C** Authorities should be able to consult every violation report sent by users.
- **G3** Users should be able to know which streets are safe and which ones are not.
- **G4** Users and authorities should be able to know the possible interventions that could be done in a city.

1.4 Glossary

1.4.1 Definitions

- **Data Mining:** is the activity related to the automatic extraction of the information hidden in a database of big dimensions.
- **Data Mining Engine:** is the system assigned to the processes that extract hidden information in a data mining activity.
- **Datawarehouse:** is an OLAP system able to structure the data in order to obtain particular functionalities like data mining.
- **Posta Elettronica Certificata:** is a technology that allows to send email with a legal approach.
- **Crowdsourcing:** is a sourcing model in which individuals or organizations obtain goods and services from a large, relatively open and often rapidly-evolving group of internet users.

1.4.2 Acronyms

- **PEC:** Posta Elettronica Certificata
- **API:** Application Programming Interface
- **IRI:** Image Recognition Interface
- **MI:** Map Interface
- **GPS:** Global Positioning System
- **OS:** Operating System
- **OLAP:** On-Line Analytical Processing
- **JSON:** JavaScript Object Notation
- **XML:** eXtensible Markup Language
- **RDF:** Resource Description Framework

1.4.3 Abbreviations

- **i.i.f.:** if and only if
- **a.k.a.:** also known as
- **G:** goal
- **DA:** domain assumption
- **R:** requirement
- **US:** user scenario
- **AS:** authority scenario

1.5 Document Structure

The document is structured in a double linked way in order to provide an easier and quicker navigation in particular for the parts where several abbreviations are used and the entire text would not fix in the layout (try with requirements in the traceability matrix 3.3.4). The aim is to give a description that interconnects the most interesting parts of the document also related in the theoretical aspects: **World and Machine**, **Goals and Requirements** and **Use Cases.**

Moreover the document is structured as now briefly described:

1. **Introduction:** gives a first description of the problem and describes the purpose of the system SafeStreets. Goals are also highlighted to enforce the previous shallow description; the section ends with the glossary.

2. **Overall Description:** starts with the product perspective where first is highlighted the system from the outside and then from the inside with a high-level description of its structure. State diagrams are then used to clarify the behavior of the most critical objects identified in the modeling process and then product functions are ready to be precisely described. The section ends with the identification of the important phenomena for the problem now clearly described with the World and Machine paradigm.

3. **Specific Requirements:** in this section requirements are precisely described starting with the ones of the interfaces that SafeStreets uses to provide its services to the external world. The core of the section are the functional requirements where the satisfaction of the goals is proved thanks to the requirements and the domain assumptions previously defined. Use Cases are also important in particular to highlight their strict relation with the requirements also highlighted with the traceability matrix.

4. **Formal Analysis Using Alloy:** includes the model described formally thanks to the alloy language [3]. This section highlights the most critical aspects of the entire problems and proves their satisfaction in specific worlds generated for this purpose.

5. **Effort Spent:** this section have been used to keep track of the hours spent to complete the document. The first table defines the hours spent together while taking the most important decisions, the seconds instead contain the individual hours.

6. **References:** includes all the references used to define the document.

2 Overall Description

2.1 Product Perspective

Thanks to the general introduction and the scope definition from the previous sections, we are now able to look at our system first from the outside and then from the inside. To deal with this description we are going to see the external interfaces the system has to interact with and then the definition of the model in order to have a feasible structure with them; at the end of the section **state diagrams** are used to emphasize the dynamic behavior of the most critical classes identified for the model.

2.1.1 System Interfaces

In this section we are going to deal with the interfaces that SafeStreets needs to *use* in order to provide its functionalities. The definition of SafeStreet's interfaces to the external world, instead, will be described in the further related section (3.1).

To accomplish the **goals** stated in the introduction the application interfaces with three main external systems as reported in the following picture (Figure 1).

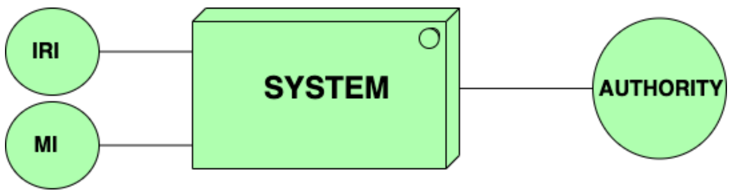


Figure 1: System Interfaces

Two different kinds of interface are distinguished in the picture above:

- **Left hand side:** interfaces that provide functions for the system to perform internal algorithmic operations. In particular:
  - IRI is used to process the pictures received from a violation. Whenever an infraction is reported, the application needs to retrieve the plate's number of the violation in order to be stored and notified to the authority. The image recognition algorithm "has always the last word": it uses the information provided by the user to help the recognition and, in all the cases a plate number will be found, it will be considered correct in the stored information and thus notifiable. Hence, whenever the algorithm does not recognize any plate number the entire notification will be discarded.
  - MI is used to process the map issues. This interface provides two services fundamental for the application: the first is used to retrieve the exact street where a reported violation has occurred (using its GPS position), the second offers the system a way to highlight a map with different colors when it provides to the functions a list of streets

and their related colors. We will describe precisely in further sections how the street retrieval is used in the notification service while the map highlight in the safety one.

- **Right hand side:** interfaces that enable the system to receive from the municipality the information about the accidents in its territory. This interaction is fundamental to obtain the data further crossed with the one of the violations in the system in order to determine the safety of an area together with its suggestions for possible interventions.

2.1.2 Model Structure

The static analysis now continues to define the internal structure of the system, in particular with a high-level class diagram (Figure 2) that considers the most important objects and their relations in order to achieve the functionalities described by the **goals**.

The main objects in the UML class diagram are:

- **Customer:** the system has to track two types of clients. The distinction, in fact, is fundamental to recognize the municipality providing accident's data but also to give a different level of visibility to the information asked by a request. This class keeps track of the information related to the login (username and password).
- **User:** identifies a citizen with all the data he provides in its registration. Users are the clients of the application that report violations and mine the data related both to the frequency of parking violations and the most dangerous vehicles or to the safety of an area and its related suggested intervention.
- **Authority:** identifies both the authority/municipality with all the data related to its recognition. Authorities are the clients of the application that gets notified of the reported violations, but they also: mine for infractions and safety information and provide data about the accidents (crossed by the system).
- **Violation:** represents a general traffic violation. This class is thought to be used in a future extension of the system in case more violations are going to be considered.
- **ParkingViolation:** is the result of a notification provided by a user. In this way the application considers all the possible information that is filled whenever an infraction is reported. As we see in the UML diagram the class contains: multiple images for an improved help to both the image recognition algorithm and the authorities; the position retrieved by the GPS (used to find the name of the street); the type of the vehicle; the plate's number, date and time. The type of the violation, selected by the users will be determined considering one of the possible extending classes: **DoubleParking, DisabledZone, NoParkingZone, BikeLaneParking...** The relation with the safety class helps to highlight how *safety* is determined as it will be precisely described in further sections 2.1.3 and 2.2.3.

- **Accident:** will be used to store the data received by the municipality by characterizing each accident as: **BikeCrash, MotorcycleCrash, CarCrash, ParkingHit...** The relation with the safety class helps to highlight how *safety* is determined as it will be precisely described in further sections 2.1.3 and 2.2.3.
- **Position:** stores the coordinates obtained by the GPS of where the parking violation occurred. The position of each infraction will be used to retrieve the street thanks to the functionalities provided by the MI.
- **Street:** is one of the most important objects to be managed. Each recognized street will be added to the system to achieve its basic and advanced functionalities. As we see is one of the classes with the most relations, in fact: it defines the city and the path, it is queried by the mining and safety requests and has a possible suggestion associated in case it is considered unsafe. The relation with the safety class determines the one of the street.
- **City:** represents the entire area that is governed by a municipality. Also this class is important to be considered as a filter in the functionalities of the system but also for the requests.
- **Path:** represents the last way in which a filter about an area can be performed. This option is mainly thought for the advanced function about retrieving the most unsafe streets in the selected path.
- **Vehicle:** vehicles are considered in order to define the feasible types which have plates and thus that can be reported in a violation. Some examples of classes are: **Truck, Car, Motorcycle...**
- **Request:** is the general class representing the interaction of a user with the system whenever mining, safety or suggestion is asked by the customer. In order to answer with the correct data it will be important to retrieve the user who sends the request and provide it to him with the correct visibility.
- **BasicRequest:** is the request that deals with the basic functionality. It is extended in fact with the three different requests that can be performed by a customer: the two related to the mining service (**MostViolationsRequest** and **DangerousVehiclesRequest**), and the one that provides a different level of visibility to the authority (**FindReportsRequest**).
- **AdvancedRequest:** is the request that deals with the advanced functionality. It is extended in fact by the different requests that can be performed by a customer: **UnsafeStreetsRequest** and **InterventionRequest.**
- **PossibleIntervention:** represents a possible intervention suggested for a precise street. The relation with the safety class helps to highlight how *safety* is determined as it will be precisely described in further sections 2.1.3 and 2.2.3.
- **Safety:** is a support class used to highlight the relations between the objects that determine whether a street is **safe** or **unsafe**.

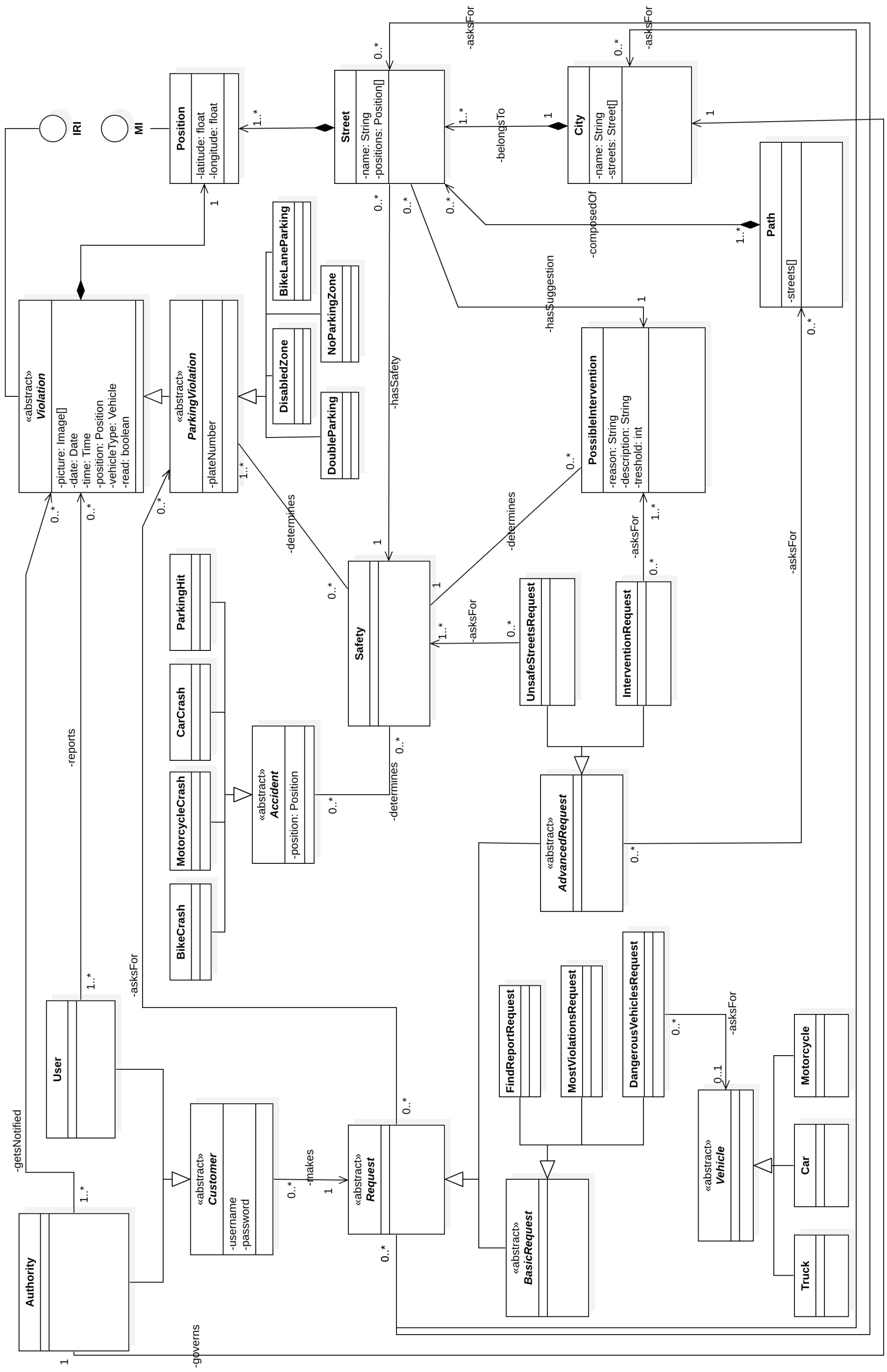


Figure 2: High-level model structure

2.1.3 State Diagrams

Considering now the main functionalities of the system, it is important to highlight the events that make its objects change their state. State diagrams are used to describe the most critical aspects of the objects previously described in the UML class diagram (Figure 2).

Notification Starting with the report of a violation it is important to remember that each infraction will be refused by the system i.i.f no plate is found by the image recognition algorithm. The diagram (Figure 3) starts when a notification is received in the *unprocessed* state. First the system needs to use the functionalities provided by the IRI in order to find the plate of the vehicle. If no plate number is found, the notification is *rejected* and the diagram ends, otherwise the *accepted* violation needs now to use the MI functionalities to retrieve the street where the infraction occurred. All the information about the violation is now known in the *accepted* state, before storing and notifying it, the application has to look for duplicates checking in its stored data for a violation with the same: plate number, street, date and time, type of vehicle and type of violation. If a *duplicate is found* the notification is still rejected by the system, otherwise it is stored and then notified to the related authority. The notification diagrams ends with the violation marked as **unread** for the authority that will be able, once logged in the system, to retrieve all the violations that occurred in its territory.

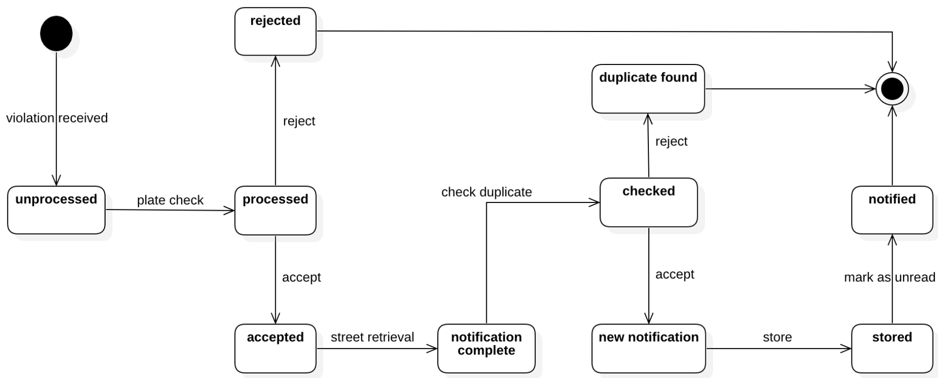


Figure 3: Notification state diagram

Request Information retrieval is another critical functionality that needs a state diagram to define how the information to be mined is provided with different levels of visibility depending on the role of the customer. The diagram (Figure 4) considers in fact the process of how the query obtained by a request will be managed in order to avoid providing secret data to the users. Whenever a request is received the relative query will be retrieved in the *unprocessed* state and then executed in the *processed* one. A check is now needed in order to understand the kind of customer from which the request is coming. If the query comes from a customer who could perform it the data is *accepted* and sent back to the inquirer, otherwise it is *rejected* and the diagram ends.

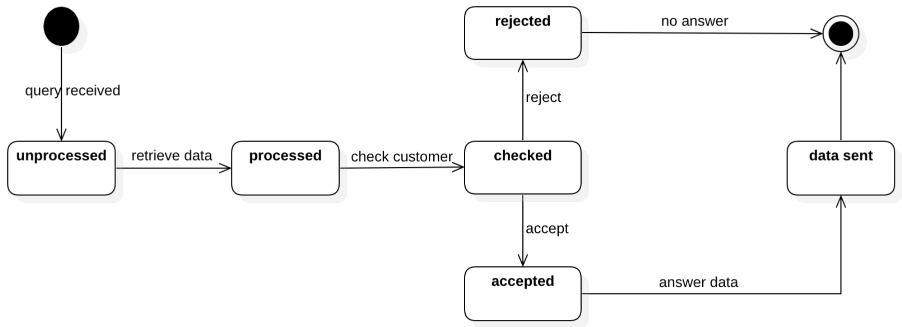


Figure 4: Request for data state diagram

Safety The last state diagram is used to highlight how the system is able to determine the safety of a street and update its state whenever new violations occur or new accidents data is provided by the municipality. The diagram (Figure 5) shows how the safety of a street evolves in the time. The system, each day, does two checks related to all the parking violations and accidents that occurred in a certain street in the previous 30 days. Two tresholds are defined to make the *safe* state trigger: one is related to each **Possible Intervention** the other to the sum of all **Violations** and **Accidents**. It is enough that one treshold for a possible intervention **or** that the sum treshold is exceeded in order to reach the *unsafe[RED]* state; the opposite happens for reaching the *safe[GREEN]* one. In the section 2.2.3 all details are provided to understand how this counting works, for now it is important to understand that thanks to it the safety of each street can be updated daily.

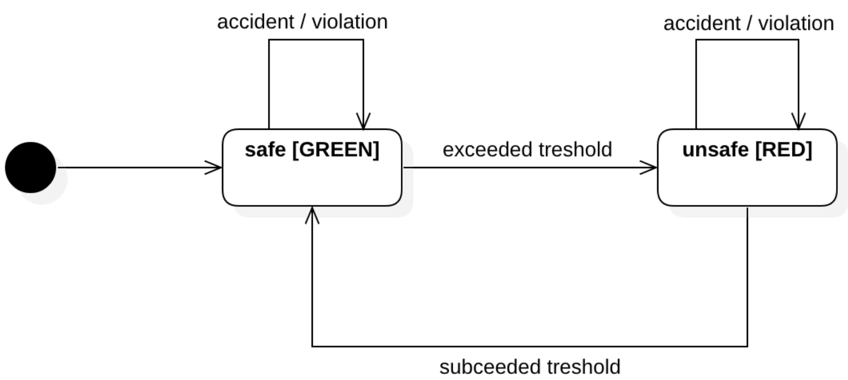


Figure 5: Area safety state diagram

2.2 Product Functions

SafeStreet is a crowd-sourced application that intends to provide users with the possibility to notify authorities when parking violations occur. This has to be considered the main product function of the system because of its importance to obtain **crowdsourcing**. The information provided by users is what SafeStreet focuses on and what makes it able to provide services about the violations in a particular area. Thanks to mining and crossing processes, the system can retrieve filtered data about frequency, safety and suggestions for possible interventions in unsafe streets. As the subsection is structured, the product functions will be precisely described following the same order of how we have just shallowly presented: first the **notification** function then the **mining** one and in the end **safety**.

Before starting with the description it is important to highlight that in order to benefit of SafeStreets functionalities the customer **must** be logged in the system as he can be recognized.

2.2.1 Notification Function

Users are allowed to notify authorities thanks to SafeStreets: the notification process takes place entirely inside the application. It starts when the user reports a violation and ends when the authority has the notification marked as unread in his interface (hence in the system too). Each user can report a violation as soon as he spots it, this means he is not allowed to notify it later; in this way we reduce the number of possible incorrect notifications as we have assumed that the *date* and *time* are automatically added by the application as well as the position, always retrieved because a user **must** enable his GPS before reporting any violation.

The notification functionality is provided to the user by an interface that allows him to insert all the information needed to report a violation. The fields a user fills are reported here with a label that describes if they are mandatory or not.

- **Pictures** `[MANDATORY]` **:** users must provide at least one picture of the violation in order to report it. The pictures are taken **inside** the application where a functionality allows to select one or more of them; in this way we are sure that every image that has been received is authentic. Having always a picture it is possible to run the image recognition algorithm to understand the plate's number, in fact it is important to remark that whenever a number is not retrieved the entire violation will be discarded. Thanks to the option of inserting multiple images we provide a larger number of possibilities to recognize the plate; these pictures can be also very useful for the authorities that may decide in this way on which of the violations to take action first.
- **Position** `[MANDATORY]` **:** this field is marked as mandatory because, as we have previously said, the position will be retrieved by the GPS of the reporting device. The application does not allow a user to report a violation if its GPS service is not enabled, in this way, if a user is trying to report

an infraction without the GPS, he will be forced to switch it on in order to complete the notification.

- **Type of Violation** `[MANDATORY]` **:** users need to give additional information providing also the type of the violation they are reporting. This kind of data is used as a filter in the mining functionalities and it can be a very useful knowledge to both users and authorities.
- **Type of Vehicle** `[MANDATORY]` **:** users must provide also this information in order to complete the notification. It is another filter in the mining functionalities.
- **Plate** `[OPTIONAL]` **:** this field is thought to give additional information to the functionalities provided by the IRI in order to recognize the plate of the reported vehicle. As we have already said it is the image recognition algorithm who has the last word when deciding whether to discard or not a violation: if he recognizes a number, also thanks to this information, the notification will be considered, otherwise it will not be.

Once the notification is received the system first tries to retrieve the plate of the reported vehicle. If the plate's number is found it means that the violation is valid and the check for duplicates can be carried out. If no duplicate is found, the violation is ready to be stored in the system and thus notified to the authority. The **notification** works as follows: each violation that has been accepted is stored as **unread** for the authority that governs the territory that contains its street, in this way, when the authority enters the system it will be able to see all the notifications that have been reported. Once checked by the authorities the notifications will be marked as **read** and removed from the *Check Unread Reports* functionality. Even if the authority of a notification is not registered yet the notification is already considered and stored as unread in the system.

2.2.2 Mining Function

Data mining is used to retrieve the information about the stored data and provide the customers different ways to filter it. Thanks to the details of each infraction, in fact, the system is able to offer this service both to users and authorities.

Two functionalities are offered to customers thanks to the mining process:

- **Highest Number of Violations:** returns an ordered list of the streets where the highest numbers of violations occur.
- **Most Dangerous Vehicles:** returns an ordered list of the types of vehicles that make the highest number of violations.

Both these functionalities but also the one allowed only to the authorities (*Find Reports*) can be filtered with some of the the filters listed below.

- **Area:** there are three levels of filters defined in the area one. The widest is called **everywhere** and returns the information generally all over Italy, the second is called **city** and considers all the streets contained in the specified city, and the most specific one considers just a single **street**.

- **Date:** this filter is allowed only to authorities in the *find reports* functionality, in this way they are able to retrieve the violations in a more detailed search.
- **Time:** customers are able to filter by choosing a time lapse in which they are interested in.
- **Type of vehicles:** the system stores in each notification the type of the vehicle that committed the violation. In this way this data can be filtered choosing one of the types: **trucks, cars, motorcycles...**
- **Type of violation:** thanks to the information about the type of the violation (provided by the user) the application allows also to filter by type. We can think of this mainly filter for the authorities who will look for the highest type of vehicles that commit a violation in a street in order to take action; however this filter is also allowed to users who may be interested in this kind of data too.

Each functionality considers only the filters that are needed to achieve its mining process. As we see in the interfaces (section 3.1.1) and in the use case diagrams (section 3.3.3) just some of the listed filters are allowed to be selected.

It is important to highlight that SafeStreets provides information to customers with different levels of visibility depending on their role. To accomplish this distinction the system provides the authorities an additional functionality that allows them to retrieve the information in the highest level of detail possible. *Highest Number of Violations* and *Most Dangerous Vehicles* are in fact statistical functionalities that are not meant to be provided differently depending on the user that is requiring them. Moreover the **Find Reports** functionality is able to give the authorities a way to retrieve each violation in detail as it is stored in the system. This functionality is the one that contains the highest number of filters as it can provide a specific way to retrieve the violations of interest to the authorities.

2.2.3 Safety Function

SafeStreets is able to process the information provided by the municipality about the accidents that occur in a territory. Thanks to this additional data the system can cross it with its stored one and determine the **safety** of each street as we now precisely describe. Whenever a municipality is not able to provide the information about the accidents to SafeStreets the safety can be still determined thanks to the mathematical model further defined that will just use the information stored about the violations.

The system dynamically changes the safety of a streets thanks to two different kinds of treshold defined for each street:

1. **Intervention Treshold:** each intervention has a "personal" treshold that once exceeded determines a street to be unsafe. Every type of accident or violation is related to one or more intervention so that, once they occur in a street they make increase the counter of each intervention. If one of the counters of the interventions for a street exceeds the treshold it means

that the street has become **unsafe** and that the exceeded intervention should be taken into account as the suggestion to enhance the **safety** of the now dangerous street.

2. **Sum Treshold:** another treshold is needed to consider also the accidents that are not related to any intervention but anyway determine a street to be **unsafe**. This treshold is based on the sum of the accidents and parking violations that occur in the street. If the sum exceeds the treshold it means that the street has become **unsafe** and the related possible intervention is the one whose counter is the nearest to the treshold.

Hence the first of the two types of treshold that is exceeded in a street determines it to be unsafe. The safety of each street and then the related intervention is determined each day by the system with the stored information about the previous month. Thanks to the data retrieved by the *crossing* process, SafeStreets is able to provide the following functionalities:

- **Unsafe Streets:** thanks to the MI SafeStreets can use its services to retrieve the portion of an area and highlight its streets with two different colors to distinguish *safe streets* colored in **GREEN** from *unsafe streets* colored in **RED**. The map interface provides functions that require a set of streets and the colors to be highlighted with and returns the best positioned map with the respective colors.
- **Urgent Interventions:** thanks to the tresholds we always at least one suggestion for a possible intervention whenever a street is considered to be unsafe. This functionality allows the customers to retrieve the possible interventions for the area in which they are interested in.

Both these functionalities in fact can be filtered in order to reduce the size of the interested area, the possible filters used in one or the other are: **city, path, street.**

2.3 User Characteristics

SafeStreets has two different customers that are very important to distinguish in order to provide correctly the functionalities previously described in subsection 2.2. In the entire document a general client is called **customer** and the distinction is made between:

- **USER**
  - Always inserts correct data while registering.
  - Always inserts correct data while reporting violations.
  - Can not register with a username that corresponds to an existing PEC address.
  - Must have a device to take pictures of a parking violation.
  - Must enable the GPS of the device in order to report a violation.
  - Can register to SafeStreets in order to be recognized as such.
  - Must login to SafeStreets to benefit of its services.

  - Can filter the mined data.
  - Can retrieve the safety of a street.
  - Can retrieve the suggestion for a possible intervention of an unsafe street.

- **AUTHORITY/MUNICIPALITY**
  - Are supposed to be in Italy.
  - Always inserts correct data while registering.
  - Knows the PEC address of the territory they manage.
  - Provides correct information about accidents.
  - Can register to SafeStreets in order to be recognized as such with their PEC address.
  - Must login to SafeStreets to benefit of its services.
  - Can search for detailed violations.
  - Can filter the mined data.
  - Can provide information about the accidents that occur in its territory.
  - Can retrieve the safety of a street.
  - Can retrieve the suggestion for a possible intervention of an unsafe street.

2.4 Domain Assumptions

Domain assumptions are used to define clearly the world in which SafeStreets is involved. Thanks to them we are able to add constraints that define the bounds of the environment. It is first important to give the description of one of these assumption to precise why we decided to consider it. The fact that *The system is allowed to work in Italy* is used to deal with the recognition of the authorities. It is very important in fact to be sure of the identity of each customer in order to manage the security of the stored data, problems relative to the law and simplify the description with only the correct level of details. Thanks to this assumption in fact we can consider that every authority knows its PEC address and thus use this existing technology to define the recognition process.

We assume that these assumptions hold true in the domain of the system:

DA1 Customers always insert correct data while registering to SafeStreets.

DA2 Users always insert correct data while reporting violations to SafeStreets.

DA3 The association between cities and PEC is well-known by SafeStreets.

DA4 PEC addresses are unique.

DA5 Special characters are all the characters that are not letters nor numbers.

DA6 Date and time on the devices on which SafeStreets runs are always correct.

DA7 The GPS module of the devices on which SafeStreets runs always works correctly and has an accuracy of 2 meters.

DA8 The camera module of the devices on which SafeStreets runs always works correctly.

DA9 Internet connection works always without errors.

DA10 The system is assumed to work in Italy.

DA11 Maps of Italy are well known, complete and up to date.

DA12 There are no streets without a name.

DA13 No one physically and maliciously replaces license plates.

DA14 There are no cities with the same name in a given region.

DA15 In a city there are not streets with the same name.

DA16 Every street belongs exactly to one city.

DA17 Accidents data provided by municipalities are always correct.

DA18 No multiple violations of the same vehicle occur in the same place at the same time.

2.5 The World and the Machine

Thanks to the constraints on the domain of interest for the system (section 2.4), we are now able to describe SafeStreets with the *World and the Machine* approach that allows to underline the most important phenomena that are present in the problem we are dealing with. The machine represents the system do be developed while the world (a.k.a environment) is the portion of the real world affected by the machine.

Thanks to this distinction we highlight in the next picture (Figure 6) the main phenomena of our problem: the ones in the world affect the world only, the ones in the machine affect the machine only and the ones in between affect both the world and the machine. This figure can be also used to start guessing the requirements (section 3) of our system as they can be defined as: *prescriptive assertions formulated in terms of shared phenomena.*

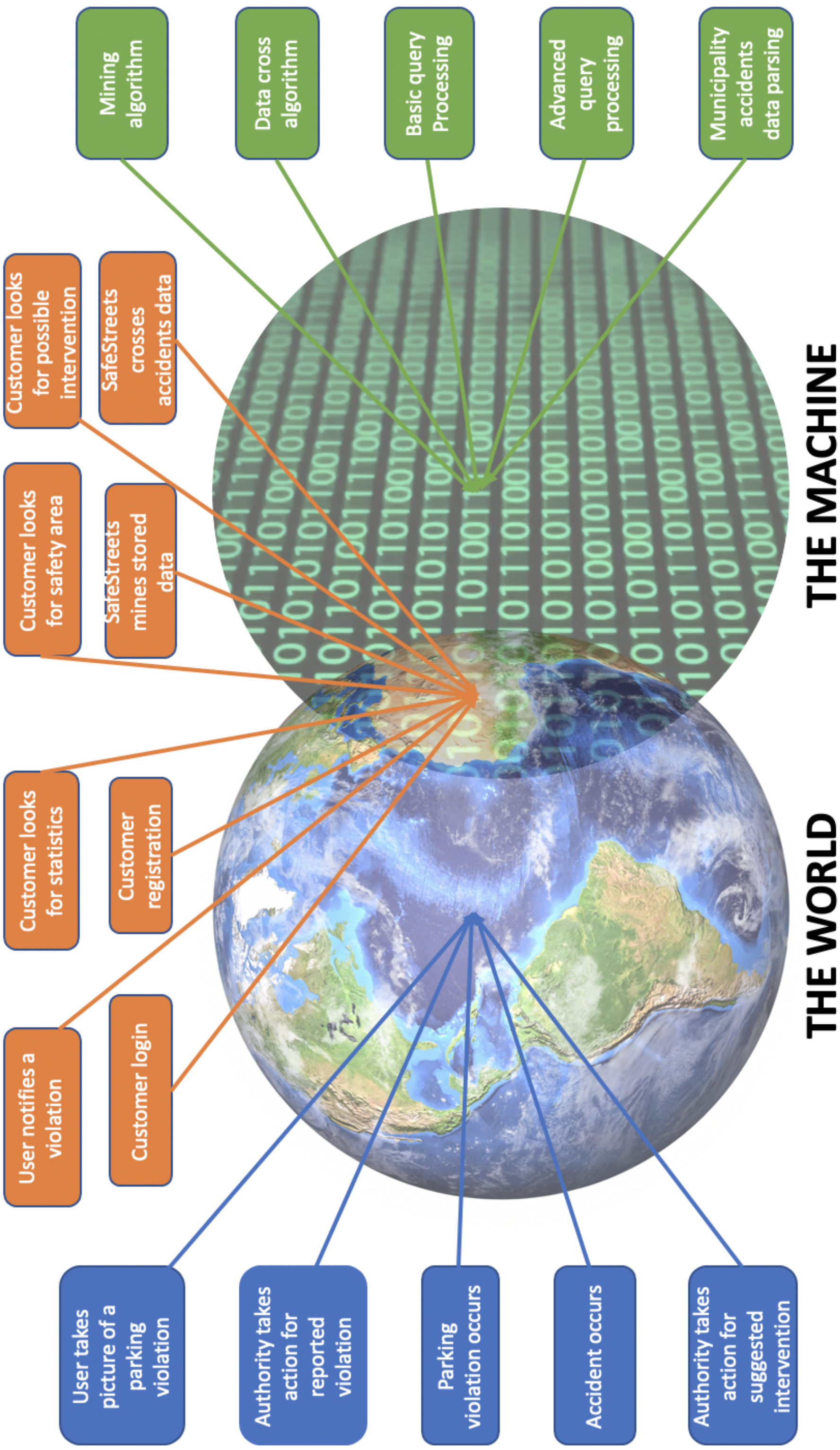


Figure 6: The World and the Machine

3 Specific Requirements

This section is devoted to a specific description of every kind of requirement our system has to deal with in order to achieve all the functionalities described.

3.1 External Interface Requirements

3.1.1 User Interfaces

The following subsection illustrates thanks to some mockups the interfaces SafeStreets supplies to its customers in order to provide the functionalities. Each picture is described only by the caption as complete and precise descriptions are given in the sections related to the definition of the functionalities and how they are managed (2.2, 3.3.3).

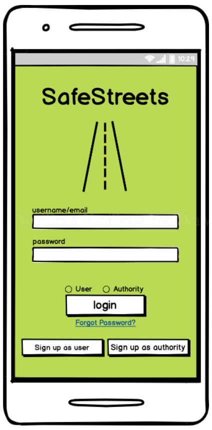


Figure 7: Login Interface

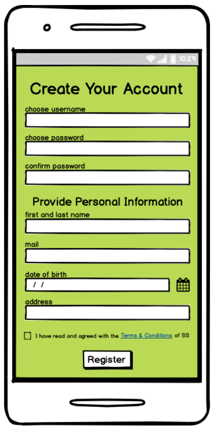


Figure 8: Register User Interface

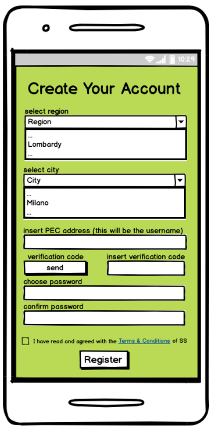


Figure 9: Register Authority Interface

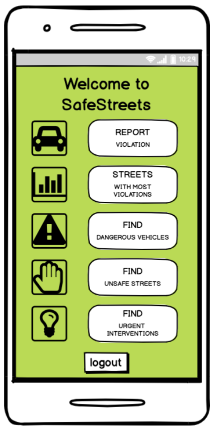


Figure 10: User Home Interface

Figure 11: Authority Home Interface

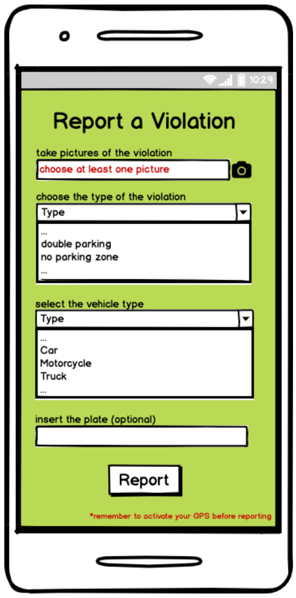


Figure 12: Report Violation Interface

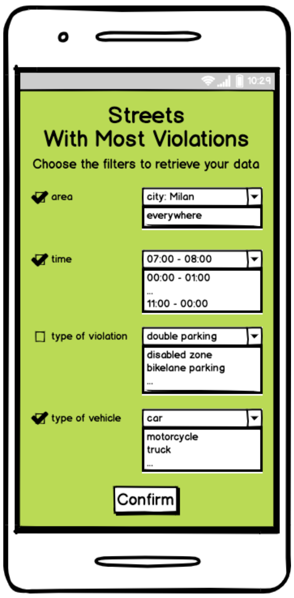


Figure 13: Streets With Most Violations Interface

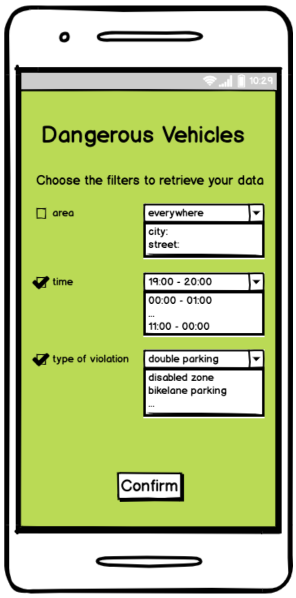


Figure 14: Dangerous Vehicles Interface

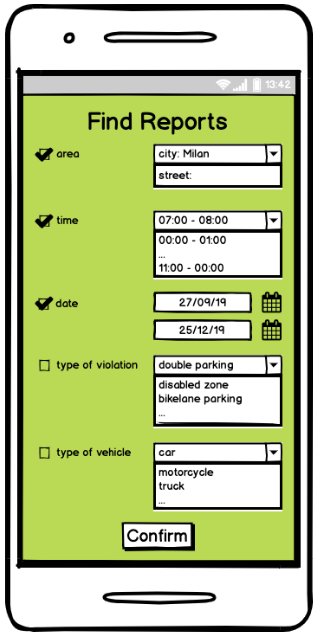


Figure 15: Find Reports Interface

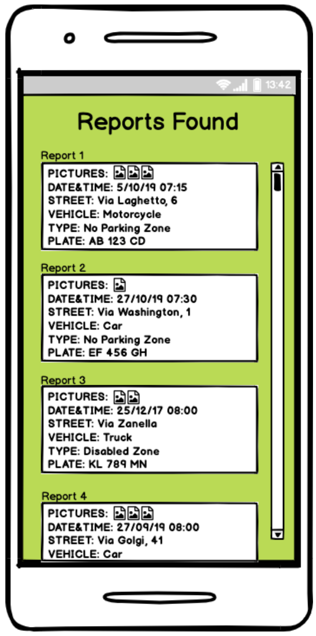


Figure 16: Reports Found Interface

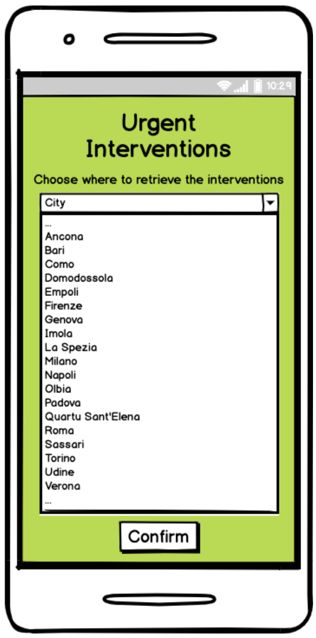


Figure 17: Urgent Interventions Interface

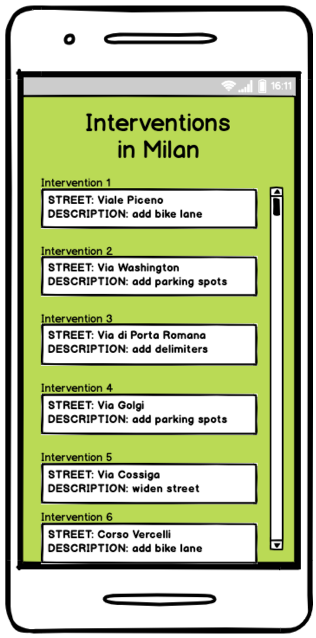


Figure 18: Intervention Found Interface Result

Figure 19: Streets Safety Interface

3.1.2 Hardware Interfaces

Hardware interfaces have to be considered on the clients' software in order to interact with the camera and the GPS modules. As described in the section 2.2.1 in fact, a user must always take at least one picture and enable the GPS in order to report a violation.

3.1.3 Software Interfaces

The interfaces used by SafeStreets to benefit of services provided by external systems are precisely described in the section 2.1.1. Internal interfaces should be also considered in order to interact with a *datawarehouse* as the mining processes can be performed and the data can be structured in the best way according to the model defined (see section 3.5.1). SafeStreets does not provide APIs to external possible clients.

3.1.4 Communication Interfaces

An important communication interface that must be considered while developing the system is the one that makes SafeStreets receive the accidents data provided by the municipality. We are now unable to know how the municipality will provide this information but we can start considering the most popular formats that would be used: JSON, XML, RDF...

3.2 Functional Requirements

This subsection aims to give a *complete* description of the *functional requirements* of our system by defining the ones together with the domain assumptions that satisfy the **goals** identified.

3.2.1 Requirements

R1 The system must allow users to register.

R2 The system must allow authorities to register.

R3 The system must allow the user to log in.

R4 The system must allow authorities to log in.

R5 The system must guarantee that each username is unique.

R6 The system must save the customers registration data.

R7 The system must prevent users to use special characters in their username.

R8 The system must allow users to take pictures of the violation they want to report.

R9 The system must retrieve date and time while the user reports a violation.

R10 The system must retrieve the GPS position while the user reports a violation.

R11 The system must allow users to insert the type of violation they're reporting.

R12 The system must allow users to insert the type of the vehicles they're reporting.

R13 The system must allow users to insert the license plate number of the vehicle they are reporting.

R14 The system must read the license plate from every picture sent by the user.

R15 The system must store the data received along with the name of the street where the violation occurred.

R16 The system must be able to retrieve the name of the street from the GPS coordinates.

R17 The system must be able to show authorities all the violation reports sent by users.

R18 The system must be able to mine the stored violation reports.

R19 The system must allow the customer to filter by city.

R20 The system must allow the customer to filter by street.

R21 The system must allow the customer to filter by path.

R22 The system must allow the customer to filter by type of violation.

R23 The system must allow the customer to filter by time slot.

R24 The system must allow the authority to filter by date intervals.

R25 The system must allow the customer to filter by type of vehicle.

R26 The system must be able to sort streets by number of violations.

R27 The system must be able to sort types of vehicle by number of violations they make.

R28 The system must be able to provide different levels of visibility.

R29 The system must be able to store and manage data about accidents, if provided by the authority.

R30 The system must be able to cross accidents data with violations one.

R31 The system must be able to determine whether a street is safe or not.

R32 The system must be able to access all the available maps.

R33 The system must find a path between two given points in a map.

R34 The system must be able to color the streets in a map according to their safety.

R35 The system must be able to determine the most urgent interventions in a street.

3.2.2 Goals

G1 Users should be able to notify authorities when traffic violations occur, in particular parking violations.

R1 The system must allow users to register.

R2 The system must allow authorities to register.

R3 The system must allow the user to log in.

R4 The system must allow authorities to log in.

R5 The system must guarantee that each username is unique.

R6 The system must save the customers' registration data.

R7 The system must prevent users to use special characters in their username.

R8 The system must allow users to take pictures of the violation they want to report.

R9 The system must retrieve date and time while the user reports a violation.

R10 The system must retrieve the GPS position while the user reports a violation.

R11 The system must allow users to insert the type of violation they're reporting.

R12 The system must allow users to insert the type of the vehicles they're reporting.

R13 The system must allow users to insert the license plate number of the vehicle they are reporting.

R14 The system must read the license plate from every picture sent by the user.

R15 The system must store the data received along with the name of the street where the violation occurred.

R16 The system must be able to retrieve the name of the street from the GPS coordinates.

R17 The system must be able to show authorities all the violation reports sent by users.

DA1 Customers always insert correct data while registering to SafeStreets.

DA2 Users always insert correct data while reporting violations to SafeStreets.

DA3 The association between cities and PEC is well-known by SafeStreets.

DA4 PEC addresses are unique.

DA5 Special characters are all the characters that are not letters nor numbers.

DA6 Date and time on the devices on which SafeStreets runs are always correct.

DA7 The GPS module of the devices on which SafeStreets runs always works correctly and has an accuracy of 2 meters.

DA8 The camera module of the devices on which SafeStreets runs always works correctly.

DA9 Internet connection works always without errors.

DA10 The system is assumed to work in Italy.

DA11 Maps of Italy are well known, complete and up to date.

DA12 There are no streets without a name.

DA13 No one physically and maliciously replaces license plates.

DA15 In a city there are not streets with the same name.

DA16 Every street belongs exactly to one city.

DA18 No multiple violations of the same vehicle occur in the same place at the same time.

G2 Users and authorities should be able to mine the information stored by SafeStreets, with different levels of visibility.

G2A Users and authorities should be able to know where the highest number of violations occur.

R1 The system must allow users to register.

R2 The system must allow authorities to register.

R3 The system must allow the user to log in.

R4 The system must allow authorities to log in.

R6 The system must save the customers registration data.

R15 The system must store the data received along with the name of the street where the violation occurred.

R18 The system must be able to mine the stored violation reports.

R19 The system must allow the customer to filter by city.

R22 The system must allow the customer to filter by type of violation.

R23 The system must allow the customer to filter by time slot.

R25 The system must allow the customer to filter by type of vehicle.

R26 The system must be able to sort streets by number of violations.

DA1 Customers always insert correct data while registering to SafeStreets.

DA2 Users always insert correct data while reporting violations to SafeStreets.

DA3 The association between cities and PEC is well-known by SafeStreets.

DA4 PEC addresses are unique.

DA5 Special characters are all the characters that are not letters nor numbers.

DA9 Internet connection works always without errors.

DA11 Maps of Italy are well known, complete and up to date.

DA12 There are no streets without a name.

DA15 In a city there are not streets with the same name.

DA16 Every street belongs exactly to one city.

G2B Users and authorities should be able to know what types of vehicle make the most violations.

R1 The system must allow users to register.

R2 The system must allow authorities to register.

R3 The system must allow the user to log in.

R4 The system must allow authorities to log in.

R6 The system must save the customers registration data.

R15 The system must store the data received along with the name of the street where the violation occurred.

R18 The system must be able to mine the stored violation reports.

R19 The system must allow the customer to filter by city.

R22 The system must allow the customer to filter by type of violation.

R27 The system must be able to sort types of vehicle by number of violations they make.

DA1 Customers always insert correct data while registering to SafeStreets.

DA2 Users always insert correct data while reporting violations to SafeStreets.

DA3 The association between cities and PEC is well-known by SafeStreets.

DA4 PEC addresses are unique.

DA5 Special characters are all the characters that are not letters nor numbers.

DA9 Internet connection works always without errors.

DA11 Maps of Italy are well known, complete and up to date.

DA12 There are no streets without a name.

DA15 In a city there are not streets with the same name.

DA16 Every street belongs exactly to one city.

G2C Authorities should be able to consult every violation report sent by users.

R2 The system must allow authorities to register.

R4 The system must allow authorities to log in.

R6 The system must save the customers registration data.

R15 The system must store the data received along with the name of the street where the violation occurred.

R18 The system must be able to mine the stored violation reports.

R19 The system must allow the customer to filter by city.

R20 The system must allow the customer to filter by street.

R22 The system must allow the customer to filter by type of violation.

R23 The system must allow the customer to filter by time slot.

R24 The system must allow the authority to filter by date intervals.

R25 The system must allow the customer to filter by type of vehicle.

R28 The system must be able to provide different levels of visibility.

DA1 Customers always insert correct data while registering to SafeStreets.

DA2 Users always insert correct data while reporting violations to SafeStreets.

DA3 The association between cities and PEC is well-known by SafeStreets.

DA4 PEC addresses are unique.

DA5 Special characters are all the characters that are not letters nor numbers.

DA9 Internet connection works always without errors.

DA11 Maps of Italy are well known, complete and up to date.

DA12 There are no streets without a name.

DA15 In a city there are not streets with the same name.

DA16 Every street belongs exactly to one city.

G3 Users should be able to know which streets are safe and which ones are not.

R1 The system must allow users to register.

R2 The system must allow authorities to register.

R3 The system must allow the user to log in.

R4 The system must allow authorities to log in.

R6 The system must save the customers registration data.

R15 The system must store the data received along with the name of the street where the violation occurred.

R30 The system must be able to cross accidents data with violations one.

R31 The system must be able to determine whether a street is safe or not.

R29 The system must be able to store and manage data about accidents, if provided by the authority.

R32 The system must be able to access all the available maps.

R33 The system must find a path between two given points in a map.

R34 The system must be able to color the streets in a map according to their safety.

R19 The system must allow the customer to filter by city.

R20 The system must allow the customer to filter by street.

R21 The system must allow the customer to filter by path.

DA1 Customers always insert correct data while registering to SafeStreets.

- **DA2** Users always insert correct data while reporting violations to SafeStreets.
- **DA3** The association between cities and PEC is well-known by SafeStreets.
- **DA4** PEC addresses are unique.
- **DA5** Special characters are all the characters that are not letters nor numbers.
- **DA9** Internet connection works always without errors.
- **DA11** Maps of Italy are well known, complete and up to date.
- **DA12** There are no streets without a name.
- **DA15** In a city there are not streets with the same name.
- **DA16** Every street belongs exactly to one city.
- **DA17** Accidents data provided by municipalities are always correct.

G4 Users and authorities should be able to know the possible interventions that could be done in a city.

- **R1** The system must allow users to register.
- **R2** The system must allow authorities to register.
- **R3** The system must allow the user to log in.
- **R4** The system must allow authorities to log in.
- **R6** The system must save the customers registration data.
- **R15** The system must store the data received along with the name of the street where the violation occurred.
- **R30** The system must be able to cross accidents data with violations one.
- **R29** The system must be able to store and manage data about accidents, if provided by the authority.
- **R32** The system must be able to access all the available maps.
- **R35** The system must be able to determine the most urgent interventions in a street.
- **R19** The system must allow the customer to filter by city.
- **DA1** Customers always insert correct data while registering to SafeStreets.
- **DA2** Users always insert correct data while reporting violations to SafeStreets.
- **DA3** The association between cities and PEC is well-known by SafeStreets.
- **DA4** PEC addresses are unique.
- **DA5** Special characters are all the characters that are not letters nor numbers.
- **DA9** Internet connection works always without errors.
- **DA11** Maps of Italy are well known, complete and up to date.
- **DA12** There are no streets without a name.
- **DA15** In a city there are not streets with the same name.
- **DA16** Every street belongs exactly to one city.
- **DA17** Accidents data provided by municipalities are always correct.

3.3 Use Cases Identification

In the following subsection the usage of the system is going to be described first with a general description using scenarios and then in a more specific way using use case diagrams. These diagrams are going to be described only once for the functionalities that totally coincide between users and authorities. Hence only the ones where critical behaviors are needed to be considered are duplicated highlighting these parts. The subsection ends with the traceability matrix that allows to define the correspondence between *requirements, use cases and scenarios.*

3.3.1 Scenarios

Consider these scenarios in order to clarify the usage of the system and the further description with use case diagrams.

User

US1 A man living in Milan has a son with disabilities named Gianluca. His son in particular suffers leg paralysis and therefore is confined to a wheelchair. Since he wants his son to live his life at its fullest, he made him take up para table tennis, the variant of table tennis designed for people with disabilities. The problem is, although few places are reserved for people with disabilities, there's no much parking in the area where training sessions are held, and so, unfortunately, people get to park in reserved spots anyway, even if they're not allowed to. Gianluca's father got really annoyed because of this situation, especially because sidewalks are not in good conditions and therefore he can't park too far away. In addition, policemen never show up in the area. He discovers SafeStreets, and starts reporting to the authorities all of those vehicles that have not exposed the pass for people with disabilities, by sending pictures of those vehicles. He makes sure that the license plate is readable and hopes that local police will take action.

US2 A group of guys fond of bicycles and competitions is going to organize a treasure hunt in Citta Studi. In this kind of game the competitors have to seek the treasures and who first finds all of them wins. The organizers must position the treasures in each area where the less number of parking violations occur to avoid bikers getting hurt when reaching them on a very high speed. Typically each organizer has an area in which he has to decide where to put the treasure. Using SafeStreet's basic functionality each of them can select Milan as the filter for the area together with the time lapse in which they expect the bikers to arrive and determine the best street in their area where the less number of violations occur.

US3 A sportive family from Milan decides to go by bike on a weekend trip to the Idroscalo. Knowing that the bicycle lane from Milan to Idroscalo gets interrupted once reached the city of Novegro, they have to decide which street is better to reach Idroscalo safely. Thanks to SafeStreets advanced functionality they can select the path and choose whether to continue taking a short part aside the highway or to get inside Novegro and enter the park from the bottom.

Authority

AS1 Circolo Vela Bellano (CVB) is a sailing club located on the high part of the Como lake. Every year it organizes a competition named *Coppa Bellano* where all members of the club tend to participate. CVB every year tries to ask to the municipality if for the entire week-end when the competition is organized the competitors can be enabled to park also on the spots for the residents (highly more than the free ones definitely not enough for the hundreds of people coming). This year, fortunately, the municipality of Bellano has decided to take action for this request, but first wants to be sure that the problem is really the one reported by the club. To solve this doubt it registers to SafeStreets using the PEC address and thanks to the *Find Reports* functionality it can filter all the violations that occurred the previous year when the competition took place. The high level of details in each violation shows that the plates numbers are all of vehicles owned by people not from Bellano. Hence the municipality decides to accomplish the desire of the club understanding that it is better to encourage the hundreds of people arriving in a small town like Bellano rather than giving them fines!

AS2 Cesate Public Library is the only library in Cesate and is usually open for too little time. The problem is the municipality of Cesate has lately become short on money and therefore can't pay enough people to work in there. Students living in that small town have no other place to study in silence and if they really need it, they can do nothing but go to Arese or Saronno, that have really big libraries but are only reachable after 15 minutes by car. Since Lombardy Region hasn't got any plan to give funds to Cesate, the municipality, that really cares about the future of its young citizens, decides to go his own way and tries to give the most traffic tickets it can. In order to do this without wasting time, the municipality uses SafeStreets to figure out where the highest number of traffic violations occur, and then sends policemen in those streets to give fines.

AS3 It's almost election time in Cesate and the outgoing local administration would like to be elected again for their second mandate. In order to gain the highest number of votes in the upcoming elections, the party decides that finally it's time to make some useful public works. The problem is they really have no clue about what their citizens need most, but fortunately, Ascanio, the City council member to the transports, knows SafeStreets. He is able to get the list of the most urgent works, according to SafeStreets. He in particular finds out that an old street that links the old part of the city to the newest one is the place where a lot of cyclists fall because of potholes. They asphalt that street and are elected again.

3.3.2 Use Case Diagram

The following diagram is a high-level description of the possible interactions of actors with the system and highlights the different use case in which actors are involved.

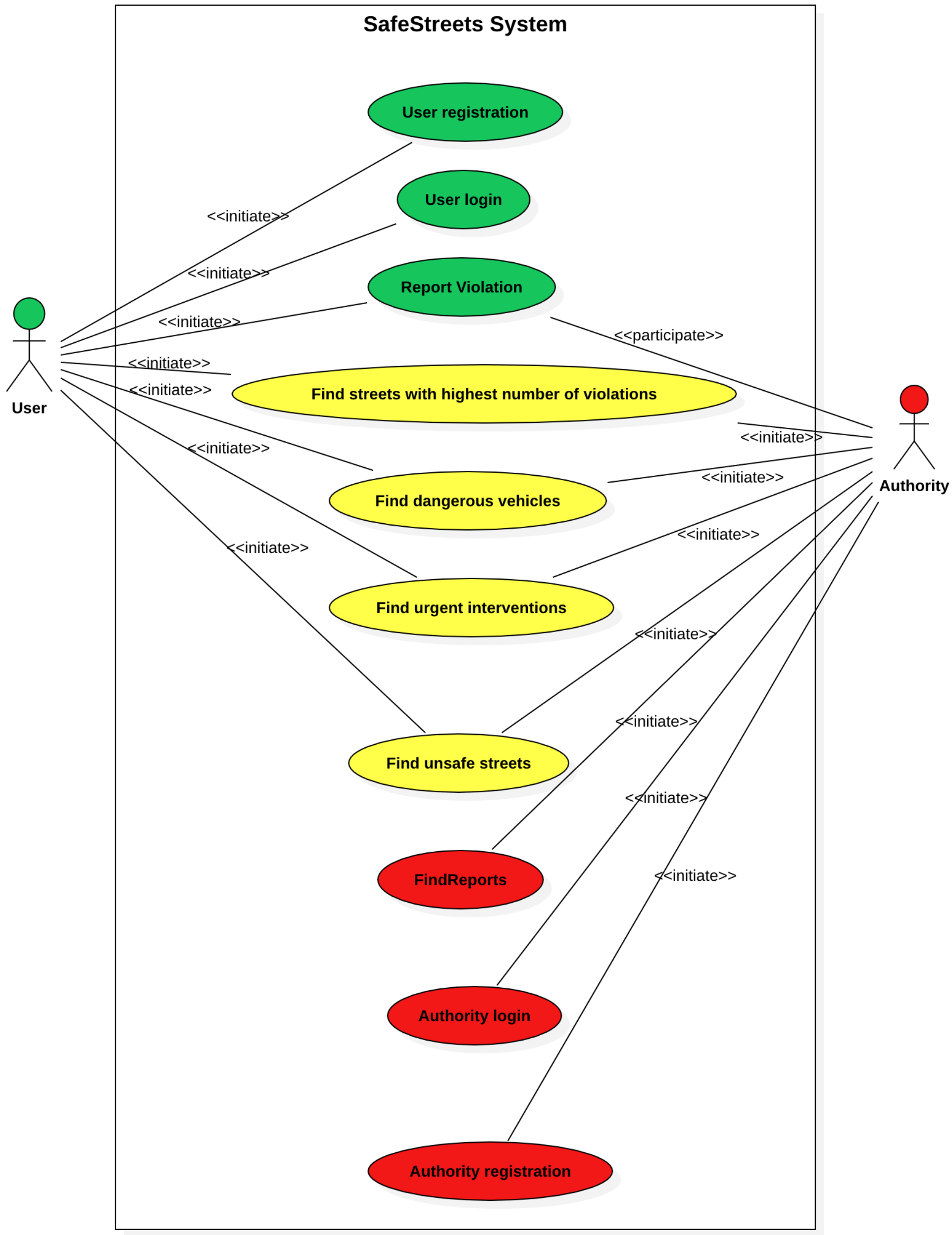


Figure 20: Use Case Diagram

3.3.3 Use Cases Description

User

1. **User Registration**

Name	**User Registration**
Actors	User
Entry conditions	The application has started
Flow of events	(a) The user chooses the sign up option (b) The user selects the 'user' registration type (c) The user chooses a username and a password (d) The user inserts his name, surname and address (e) The user submits the form (f) The system checks the username to be unique (g) The system saves the user data
Exit conditions	The user is registered in the system
Exceptions	• If the username inserted by the user is already used by another user, or if the username contains any special character, the system displays an error message asking the user to insert a different one

Table 1: *User Registration* use case description

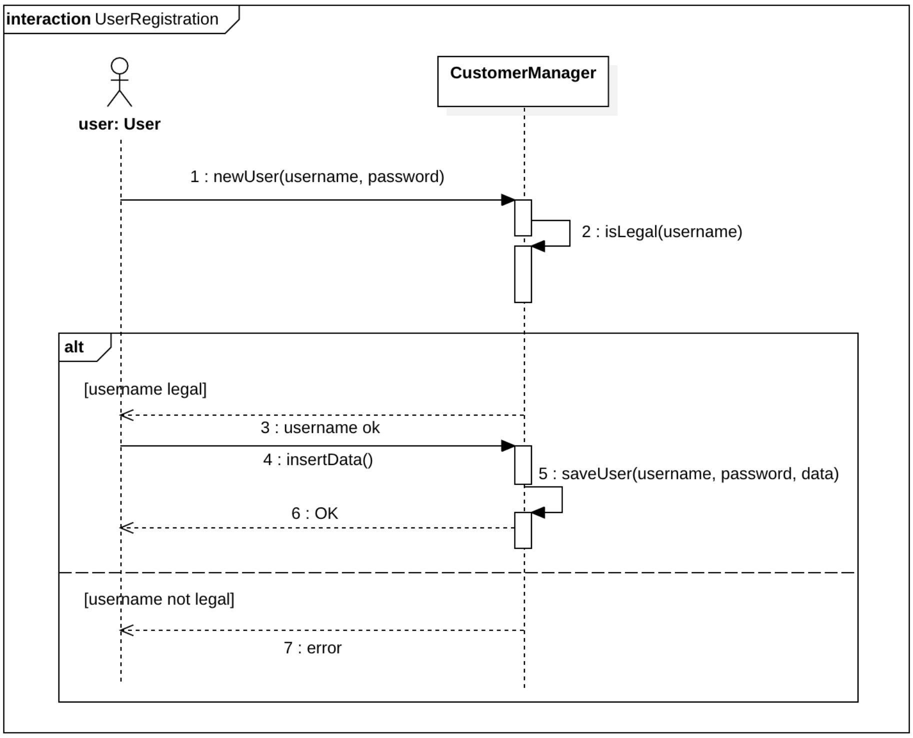


Figure 21: User Registration sequence diagram

2. **User Login**

Name	**User Login**
Actors	User
Entry conditions	The application has started
Flow of events	(a) The user chooses the login option (b) The user chooses the 'user' login type (c) The user inserts his username (d) The user inserts his password (e) The user submits the form (f) The system checks the username to be existing (g) The system checks the password to be right for that username (h) The system notifies the user that login is successful

Exit conditions	The user is logged in
Exceptions	• If the username is not recognized by the system, that means that the user is not registered yet, or the username is incorrect. The system notifies the user and the procedure is aborted • If the inserted password is wrong, the system notifies the user and the procedure is aborted

Table 2: *User Login* use case description

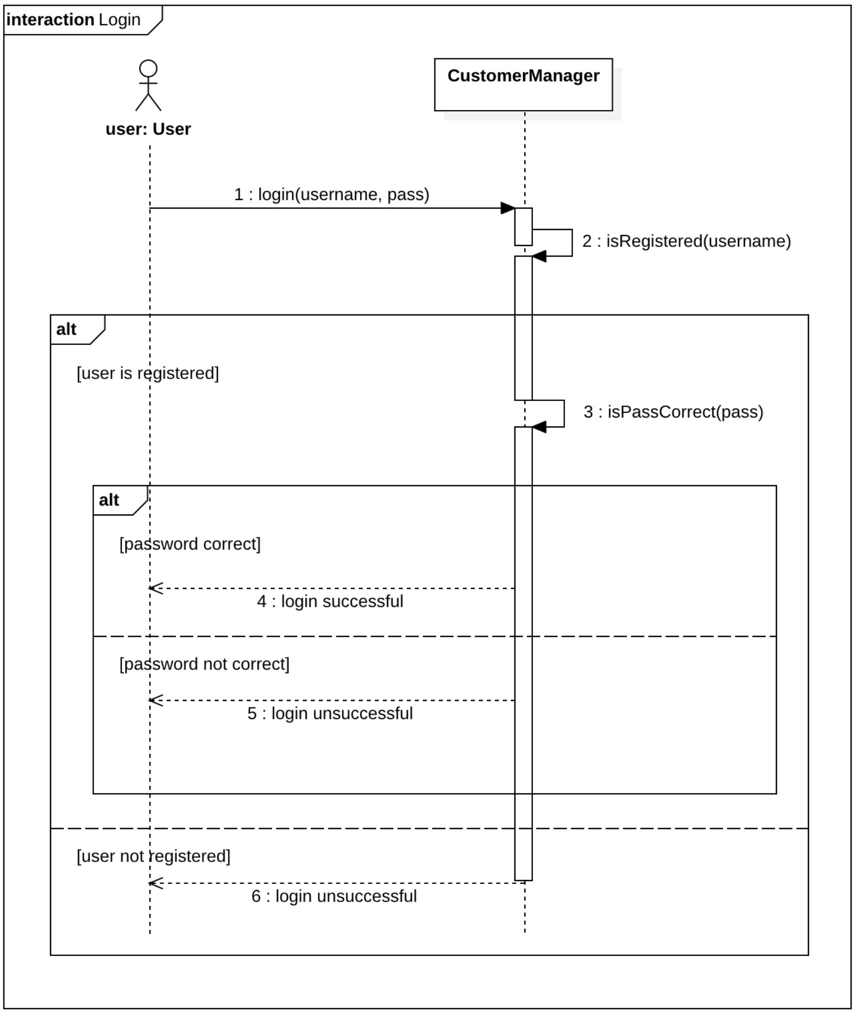


Figure 22: User Login sequence diagram

3. **Report Violation**

Name	**Report violation**
Actors	User
Entry conditions	The user is logged in
Flow of events	(a) The user selects the report violation option (b) The system retrieves the GPS location (c) The user chooses the option to take pictures (d) The user takes some pictures through the application (e) The user selects the pictures he wants to send (f) The user optionally inserts the license plate number (g) The user chooses the type of violation from a list (h) The user chooses the type of vehicle from a list (i) The user chooses the option to confirm (j) The system receives the sent data (k) The system runs an algorithm to read the license plate, with the help of the information provided by the user (l) The system retrieves the name of the street from the GPS location (m) The system stores the violation report
Exit conditions	The information about the violation is stored
Exceptions	• If the system fails to retrieve the GPS location, the user is notified and the application shows the home page • If the system fails to read the license plate, or what it reads does not match the information provided by the user, the field will be left empty

Table 3: *Report violation* use case description

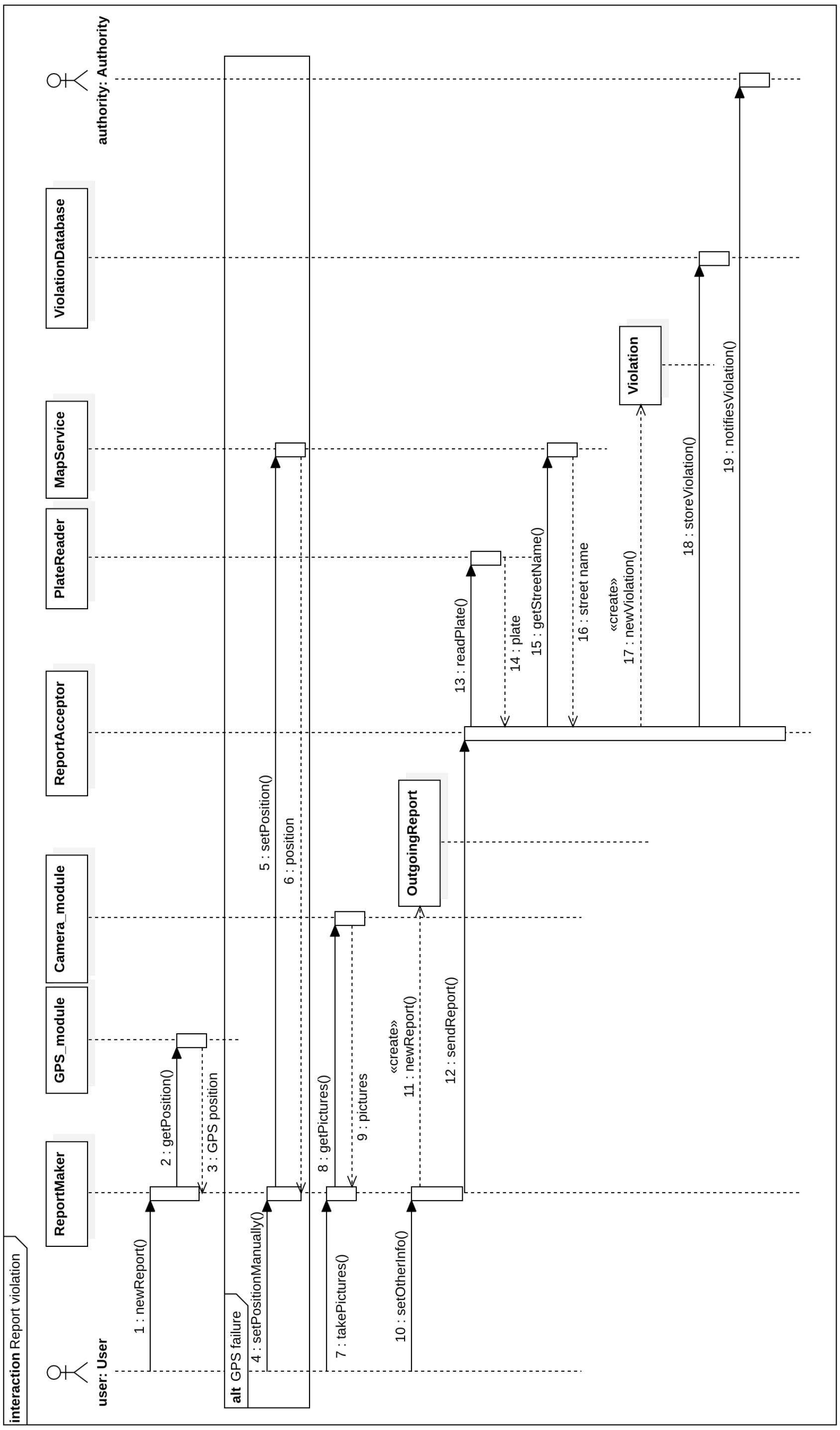


Figure 23: Report Violation sequence diagram

4. **Find streets with the highest number of violations**

Name	**Find streets with the highest number of violations**
Actors	User
Entry conditions	The user is logged in
Flow of events	(a) The user selects the 'streets with highest number of violation' option (b) The user chooses the region he is looking for from a list (c) The user chooses the city from a list (d) The user chooses the types of violation to be included (e) The user chooses the time slot (f) The user chooses the types of vehicle to be included (g) The user confirms the query and sends it (h) The system returns a list of the streets ordered by the highest number of violations, with the actual number next to the name of the street, according to the filters
Exit conditions	The list of the streets is shown to the user
Exceptions	• If no type of violation is selected, the system shows an error message and the procedure is aborted • If no type of vehicle is selected, the system shows an error message and the procedure is aborted

Table 4: *Find streets with the highest number of violations* use case description

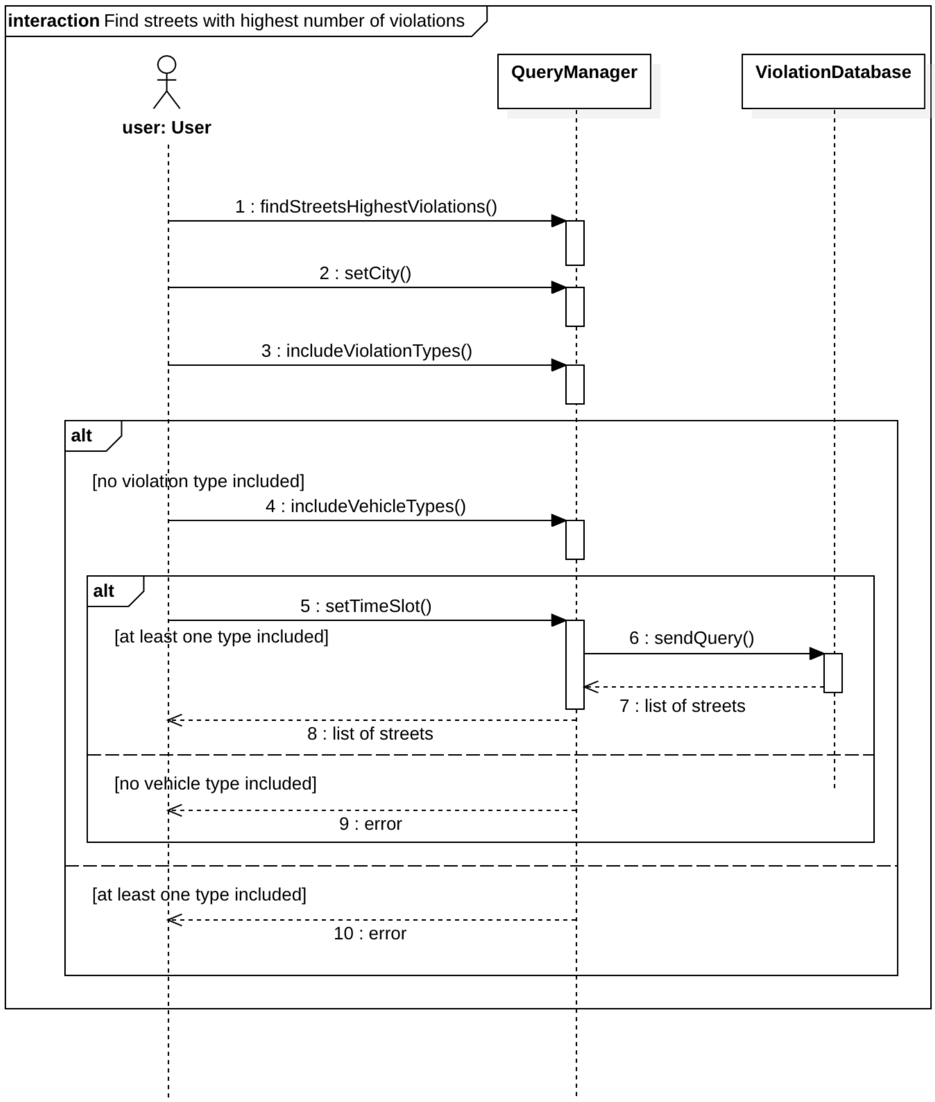


Figure 24: Highest Number of Violations sequence diagram

5. **Find Most Dangerous Vehicles**

Name	**Find dangerous vehicles**
Actors	User
Entry conditions	The user is logged in
Flow of events	(a) The user selects the 'most dangerous vehicles' option (b) The user selects the region and the city from a list, or selects a street, or selects everywhere (c) The user selects the types of violation he wants to include (d) The user confirms the query and sends it (e) The system returns a list of the types of vehicle, ordered by the highest number of violations they committed, according to the filters
Exit conditions	The list is shown to the user
Exceptions	• If no type of violation is selected, the system notifies the user and wait for him to insert at least one

Table 5: *Find most dangerous vehicles* use case description

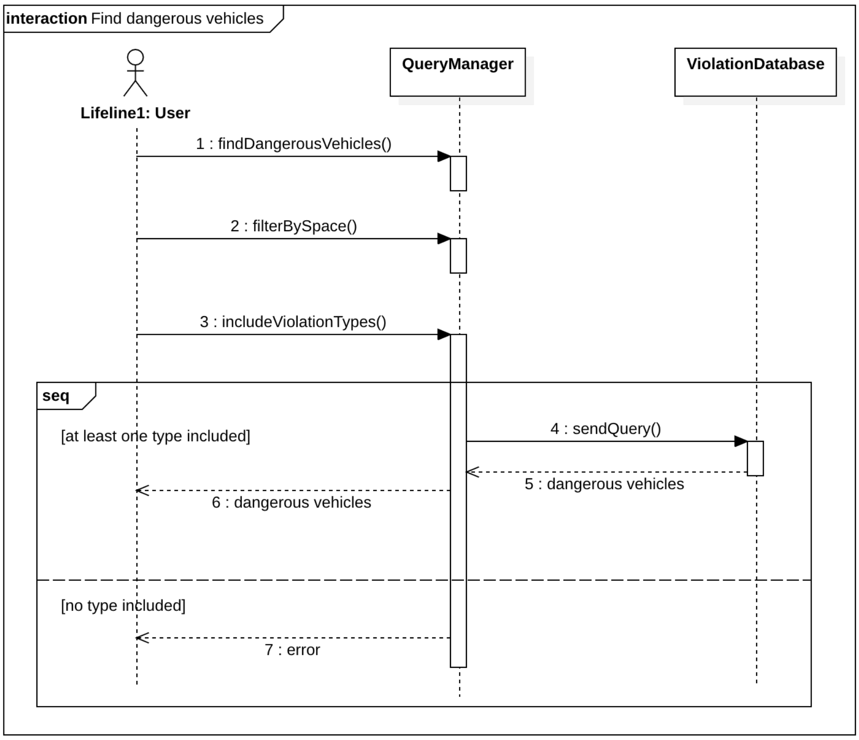


Figure 25: Dangerous Vehicles sequence diagram

6. **Find Urgent Interventions**

Name	**Find urgent interventions**
Actors	User
Entry conditions	The user is logged in
Flow of events	(a) The user selects the 'urgent interventions' option (b) The user selects the region and the city from a list (c) The user confirms the query and sends it (d) The system returns a list of the most urgent interventions in the selected city, each with their respective street
Exit conditions	The list is shown to the user
Exceptions	

Table 6: *Find urgent interventions* use case description

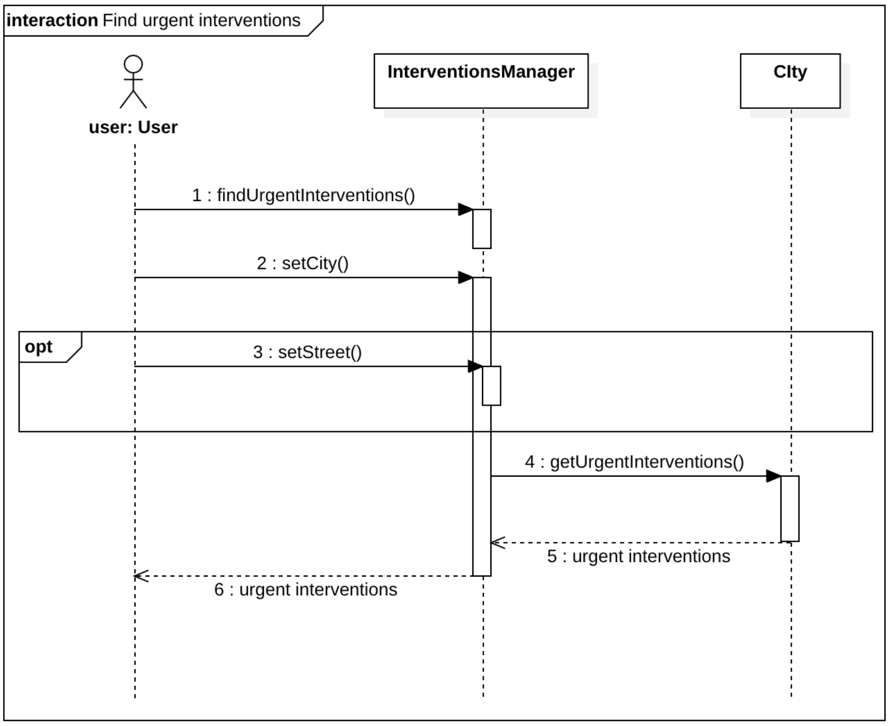


Figure 26: Urgent Interventions sequence diagram

7. **Find Unsafe Streets**

Name	**Find unsafe streets**
Actors	User
Entry conditions	The user is logged in
Flow of events	(a) The user selects the 'unsafe streets' option (b) The user selects the either the 'city' option, the 'route' option or the single street option (c) The user chooses the region and the city from a list if 'city' option is chosen, if 'route' option is chosen he enters start and end points of the route, otherwise he inserts the name of the street (d) The user confirms the query and sends it (e) The system returns a map where the streets selected with the filter are colored according to their safety: green if they're 'safe', red if they're 'unsafe'
Exit conditions	The list is shown to the user
Exceptions	• If no city is selected when 'city' option is chosen, the system notifies the user and waits for him to insert it • If no start or end points are chosen when 'route' option is selected, the system notifies the user and waits for him to insert them

Table 7: *Find unsafe streets* use case description

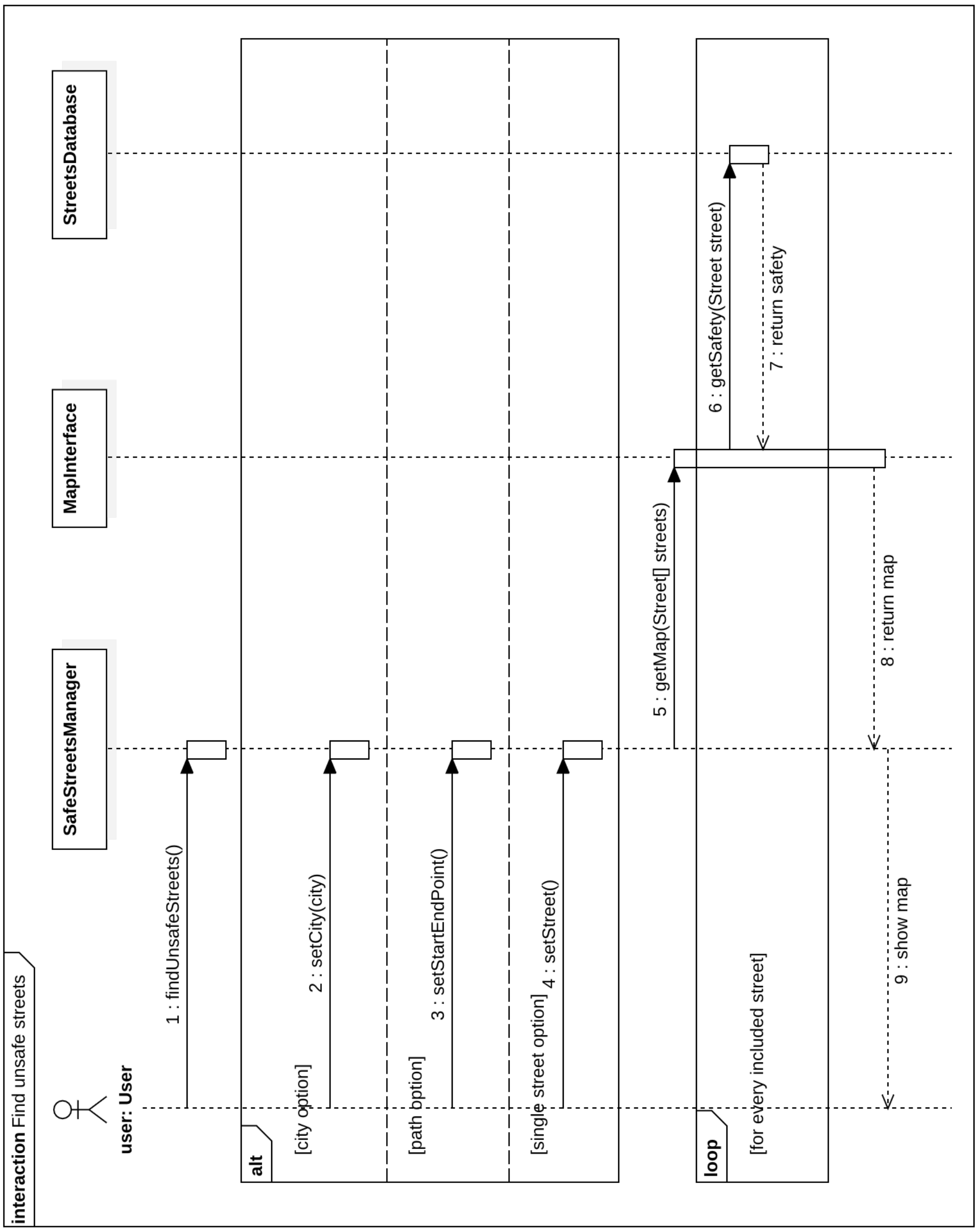


Figure 27: Unsafe Streets sequence diagram

Authority

1. **Authority Registration**

Name	**Registration**
Actors	Authority
Entry conditions	The application has started
Flow of events	(a) The authority chooses the sign up option (b) The authority selects the 'authority' registration type (c) The authority chooses the region and the city of competence (d) The authority inserts its PEC address (e) The system checks that PEC address matches the chosen city (f) The system sends a confirmation code to the PEC address (g) The authority enters the confirmation code in a text box (h) The authority submits the form (i) The system checks the entered code to match the sent one (j) The system saves the authority's data
Exit conditions	The authority is registered in the system
Exceptions	• If the selected city already has an authority registered, an error message is shown and the procedure is aborted • If the PEC address doesn't match the chosen city, an error message is shown and the procedure is aborted • If the entered code doesn't match the sent one, an error message is shown and the procedure is aborted

Table 8: *Authority Registration* use case description

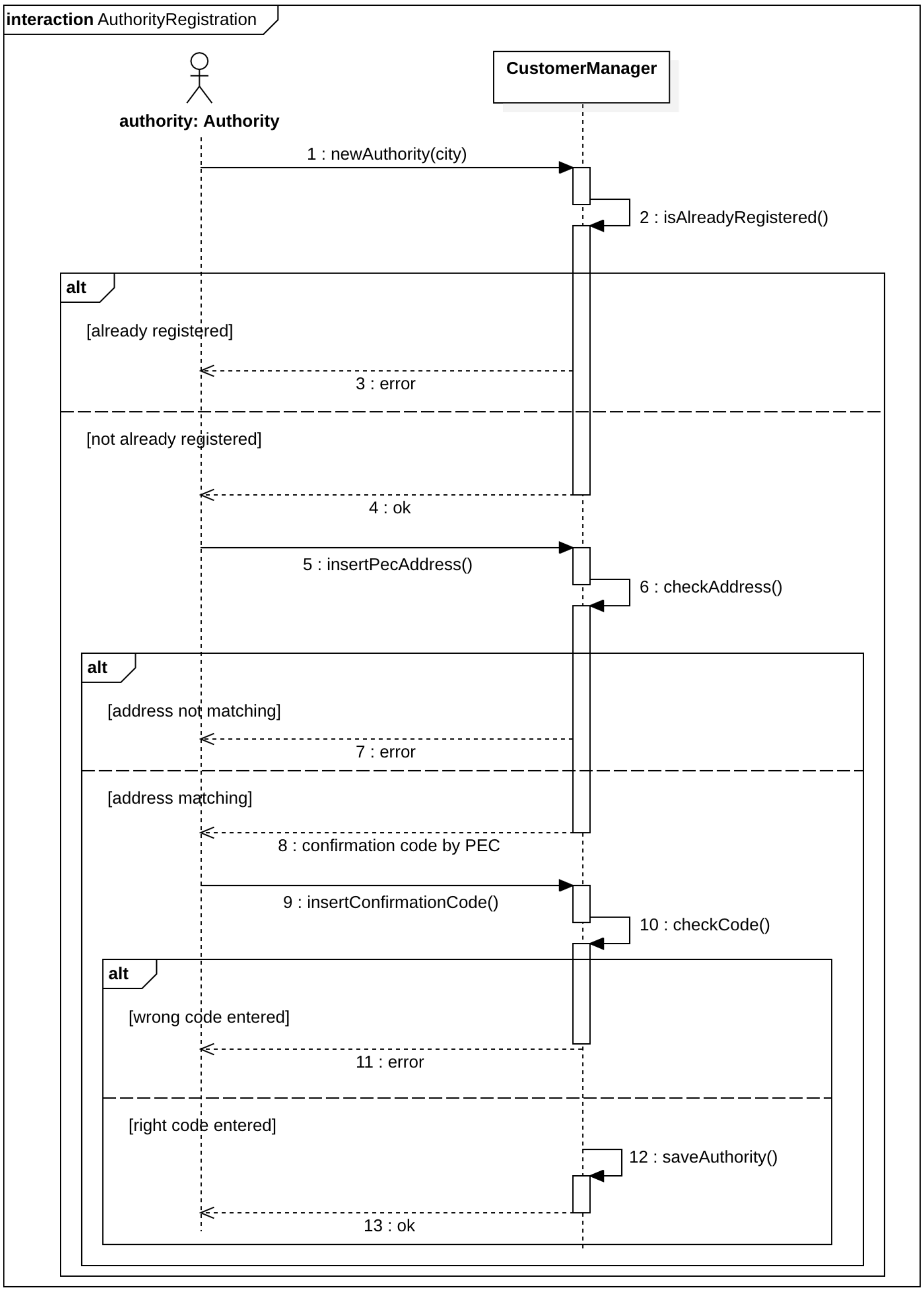


Figure 28: Authority Registration sequence diagram

2. **Authority Login**

Name	**Authority Login**
Actors	Authority
Entry conditions	The application has started
Flow of events	(a) The authority chooses the login option (b) The authority chooses the 'authority' login type (c) The authority inserts its PEC address as username (d) The authority inserts its password (e) The authority submits the form (f) The system checks the username to be registered (g) The system checks the password to be right for that username (h) The system notifies the authority that login is successful
Exit conditions	The authority is logged in
Exceptions	• If the username is not recognized by the system, that means that the authority is not registered yet, or the username is incorrect. The system notifies the authority and the procedure is aborted • If the inserted password is wrong, the system notifies the authority and the procedure is aborted

Table 9: *Authority Login* use case description

3. **Check Unread Reports**

Name	**Check unread reports**
Actors	Authority
Entry conditions	The authority is logged in
Flow of events	(a) The authority chooses the 'unread reports' option (b) The system provides a list with all the unread violation reports that match the city of competence of the authority

Exit conditions	The list is shown to the authority
Exceptions	

Table 10: *Check unread reports* use case description

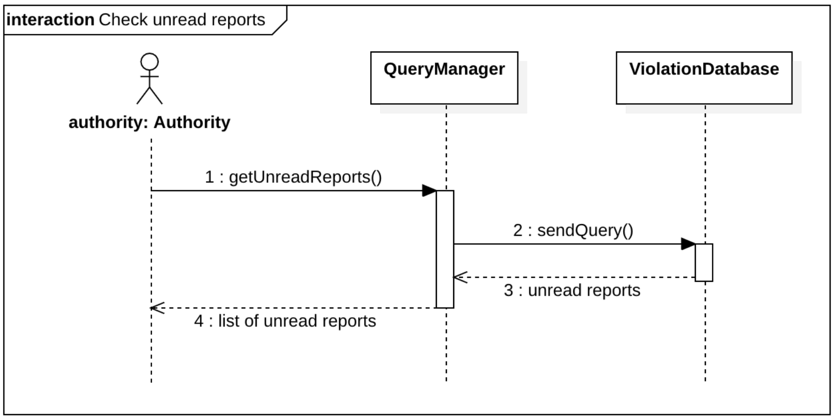


Figure 29: Unread Reports sequence diagram

4. **Find Reports**

Name	**Find reports**
Actors	Authority
Entry conditions	The authority is logged in
Flow of events	(a) The authority chooses the 'find reports' option (b) The authority selects the types of violation to be included (c) The authority selects date interval and time slot (d) The authority selects the types of vehicle to be included (e) The authority optionally selects the street to include, otherwise 'city' will be selected (f) The authority confirms and sends the query (g) The system provides a list with all the violation reports that match the filter, sorted by chronological order from the most recent one. In case 'city' option is selected, the considered city is that of competence of the authority

Exit conditions	The list is shown to the authority
Exceptions	(a) If the date interval is not valid, the system notifies the authority and the procedure is aborted

Table 11: *Find reports* use case description

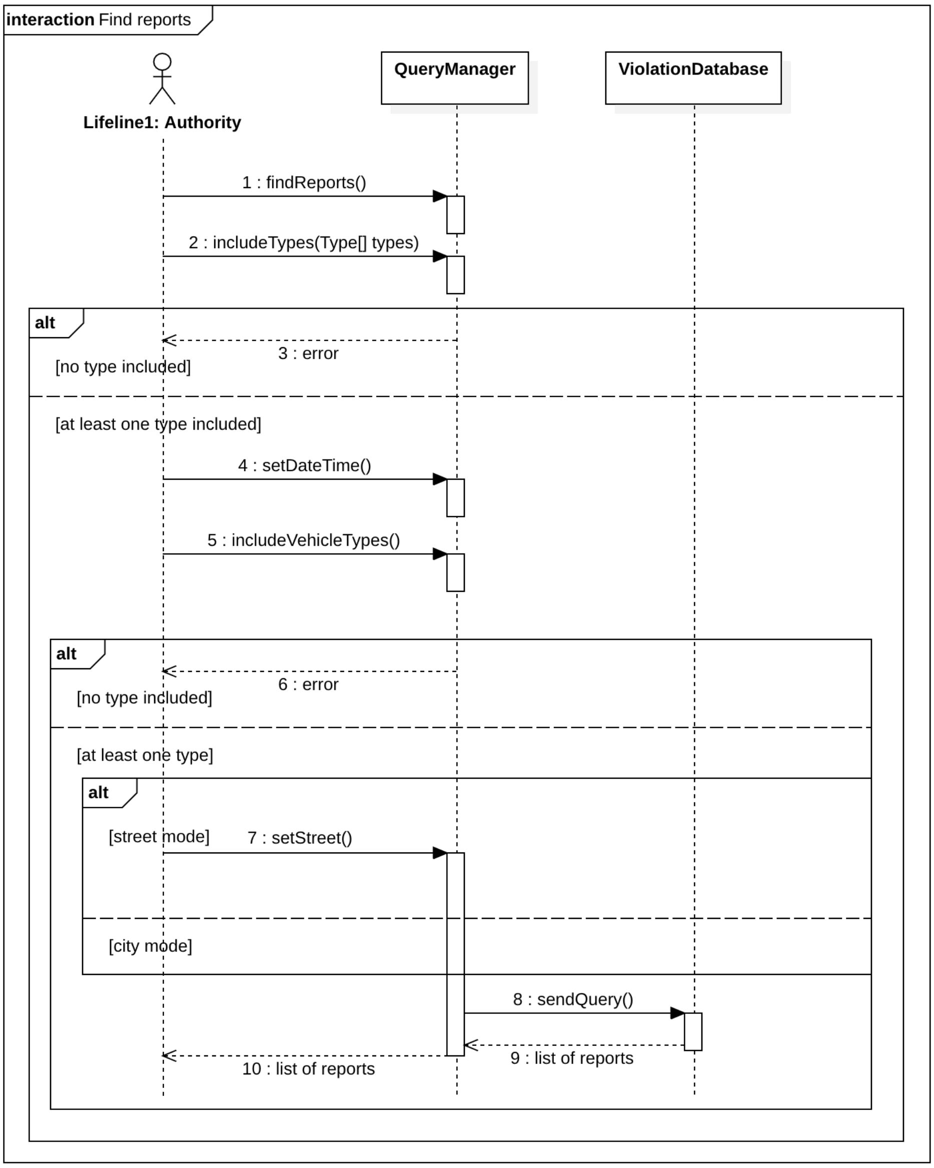


Figure 30: Find Reports sequence diagram

3.3.4 Traceability Matrix

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**
Registration	✘	✘			✘	✘	✘			
Login			✘	✘	✘		✘			
Report Violation	✘		✘				✘	✘	✘	✘
Highest Violations	✘	✘	✘	✘						
Dangerous Vehicles	✘	✘	✘	✘						
Urgent Interventions	✘	✘	✘	✘						
Unsafe Streets	✘	✘	✘	✘						
Unread Reports		✘		✘						
Find Reports		✘		✘						
US1	✘		✘				✘	✘	✘	✘
US2	✘		✘							
US3	✘		✘							
AS1		✘		✘						
AS2		✘		✘						
AS3		✘		✘						

Table 12: Requirements from R1 to R10

	R11	**R12**	**R13**	**R14**	**R15**	**R16**	**R17**	**R18**	**R19**
Registration									
Login									
Report Violation	✘	✘	✘	✘	✘	✘	✘	✘	
Highest Violations								✘	✘
Dangerous Vehicles								✘	✘
Urgent Interventions					✘				✘
Unsafe Streets					✘				✘
Unread Reports					✘				
Find Reports					✘				✘
US1	✘	✘	✘	✘	✘	✘	✘	✘	
US2								✘	✘
US3					✘				✘
AS1					✘				✘
AS2								✘	✘
AS3					✘				✘

Table 13: Requirements from R11 to R19

	R20	**R21**	**R22**	**R23**	**R24**	**R25**	**R26**	**R27**	**R28**
Registration									
Login									
Report Violation									
Highest Violations			✘	✘		✘	✘	✘	
Dangerous Vehicles			✘	✘			✘		
Urgent Interventions	✘								
Unsafe Streets	✘	✘							
Unread Reports									
Find Reports	✘		✘	✘	✘	✘			✘
US1									
US2			✘	✘		✘	✘		
US3	✘	✘							
AS1	✘		✘	✘	✘	✘			✘
AS2			✘	✘		✘	✘		
AS3	✘								

Table 14: Requirements from R20 to R28

	R29	**R30**	**R31**	**R32**	**R33**	**R34**	**R35**
Registration							
Login							
Report Violation							
Highest Violations							
Dangerous Vehicles							
Urgent Interventions	✘	✘		✘			✘
Unsafe Streets	✘	✘	✘	✘	✘	✘	
Unread Reports							
Find Reports							
US1							
US2							
US3	✘	✘	✘	✘	✘	✘	
AS1							
AS2							
AS3	✘	✘		✘			✘

Table 15: Requirements from R29 to R35

3.4 Performance Requirements

SafeStreets is a system that provides to serve all mobile devices that have the possibility to take pictures and to retrieve the position thanks to the GPS. All the computation will take place on the servers of the system thus we expect a light weight application that can be accessed by both the users and authorities. Interaction with modules of the device have to be considered, in particular with the camera and the GPS in order to provide all mandatory data while reporting a violation. The notification functionality requires a quick response to each violation in order to store it in the system for the authorities as soon as possible. Further considerations about responsiveness have to be considered for the basic and advanced functionalities that require mining and crossing processes in order to obtain a result. To accomplish this problem we will give a possible solution in the further section of how to organize the design in order to run algorithms that allow to mine and cross the data. Technical support for installation is not required as the result is a simple device application each customer can download from the store; moreover it is important to highlight that the registration process for the authority needs to provide a verification code and thus a quick interaction needs to be considered once again.

3.5 Design Constraints

3.5.1 Standards Compliance

The most important standard that needs to be considered in a crowd-sourced system and in particular for the functionalities that SafeStreets intends to provide is the *data mining* process. As we are going to expect to receive big amounts of data the approach to consider a *datawarehouse* would be the correct solution paired with a data mining engine working with the data already stored. Considering the system to follow the standard process of data mining we can give a further detail based on how the definition of the model has been carried out: the classification technique seems in fact the better one in order to respect the classification described for the violations (largest instances we expect to manage).

3.5.2 Hardware Limitations

Hardware limitations would be critical in devices that do not provide a camera or a GPS module. As described in the section 2.2.1 in fact a user must provide at least one picture and enable his GPS in order to report a violation. No further considerations have to be considered as we have also specified that the image recognition algorithms will be helped by the information provided by the users; this can make us not to require high definition pictures of the infractions reported and thus nor the newest cameras for the devices.

3.5.3 Any other Constraint

Regulatory policies have to be considered for the interaction between SafeStreets and both users and authorities. The application in fact will ask the position of each user while reporting a violation but will also provide its name to authorities whenever they receive a notification or use the *find reports* functionality.

Authorities instead need to accept the treatment that will be carried out by the crossing process over the accidents data they provide to SafeStreets.

3.6 Software System Attributes

3.6.1 Reliability

As the system aims to be a crowd-sourced application it needs to care of the data he stores, in particular the one used for the mining and crossing processes. Design considerations are needed in order to face any failure, if a decision needs to be made to choose which data is the most important to protect, the one relative to the violations is going to be preferred rather than the one of the customers. The loss of the information of a client in fact is definitely not a problem as he would create a new account without any limitation.

3.6.2 Availability

As nowadays systems tend to be always more available we need to accomplish this feature but with some considerations related to the fact that we expect to receive the highest affluence in the day rather than in the night. Hence an availability of 99% is needed for the notification functionality, for the basic and advanced functionalities instead 96% should be enough to cover the desires of both customers.

3.6.3 Security

SafeStreets needs to deal with security to store the information of a user while registering but also to ensure the correct registration of an authority. The recognition of a customer is the most important feature to consider in order to ensure providing the correct level visibility of the data they query. The recognition process considers to send a verification code to the PEC address of the registering authority, this deals with all security problems avoiding more complex technologies that each authority would not posses.

3.6.4 Maintainability

The software needs to take in account the possibility to encounter bugs even if a high test coverage should be considered in the developing process in particular in the query definition that may lead to critical aspects related to the management of the datawarehouse together with the data mining engine. Talking about the capability to meet new requirements the system is thought to be extensible in particular for the additional types of violations that may be considered.

3.6.5 Portability

The client side of the system would be developed in a native way in order to accomplish a better interaction with the system for both the most used device OSs. The server side instead would consider the possibility to use different mining engines rather than how to store the data in the datawarehouse where a proprietary software would be possibly considered.

4 Formal Analysis Using Alloy

The following section considers the essential properties and constraints identified for the specification of the problem and provides a formal model in which it is shown how they will be satisfied. The alloy modeling language is used to model the problem, and some possible worlds are also provided in order to clarify the most critical aspects. A few things have been simplified, for example the way in which the system computes if a street is safe, or what interventions to suggest in a given street. Only two types of request have been considered and the reason is they're pretty much the same: indeed they are all based on filters that work in the same way. The next section is composed of the description of the model in alloy followed by three generated example worlds.

4.1 Alloy Model

```
sig Username{}

sig Password{}

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

abstract sig Customer{
  registration: one Registration
}

sig User extends Customer{}

sig Authority extends Customer{
  city: one City,
  unreadReports : set StoredReport
}

fact UnreadReportsRule{
  all sr:StoredReport, a:Authority | sr in a.unreadReports
    iff
  (
    sr.status = Unread and sr.report.position.street.city in
     a.city
  )
}

sig City{
  streets: some Street,
  authority: lone Authority
}

sig Street{
  city : one City,
  streetName: one StreetName,
```

```
  positions: some Position,
  reports: set Report,
  accidents: set Accident,
  suggestedInterventions: set Intervention,
  status : one StreetStatus
}

abstract sig StreetStatus{}

one sig Safe extends StreetStatus{}

one sig Unsafe extends StreetStatus{}

sig StreetName{
}

sig Position{
  street : one Street
}

sig Date{}

sig Time{}

sig ViolationType{
  interventions: set Intervention
}

sig VehicleType{}

sig Intervention{}

sig Plate{}

sig Report{
  user: one User,
  position: one Position,
  violationType: one ViolationType,
  vehicleType: one VehicleType,
  date : one Date,
  time: one Time,
  plate : lone Plate,
}

sig StoredReport{
  report : one Report,
  plate : one Plate,
  streetName : one StreetName,
  status : one ReportStatus
}

abstract sig ReportStatus{}

one sig Read extends ReportStatus{}
```

```
one sig Unread extends ReportStatus{}

sig AccidentType{
  interventions: set Intervention
}

sig Accident{
  type : one AccidentType,
  street: one Street
}

////////////////////////////////////////////////////
//STREETS
////////////////////////////////////////////////////

fact NoStreetWithoutCity{
  all s: Street | one c:City | s in c.streets
}

fact StreetCityLinked{
  all s:Street, c:City | s in c.streets iff c in s.city
}

fact NoStreetsWithSameNameInCity{
  all s1, s2:Street | ((some c:City| s1 in c.streets and s2
    in c.streets) and
     not(s1=s2)) implies not(s1.streetName = s2.streetName)
}

fact NoStreetNameWithoutStreet{
  all sn: StreetName | some s:Street | sn in s.streetName
}

fact PositionOnlyInOneStreet{
  no disj s1, s2: Street | s1.positions & s2.positions not =
     none
}

fact NoPositionWithoutStreet{
  all p:Position | some s:Street | p in s.positions
}

fact PositionStreetLinked{
  all s:Street, p:Position | p in s.positions implies s in p
    .street
}

fact ReportStreetLinked{
  all s:Street, r:Report | r in s.reports iff s in r.
    position.street
}

fact AccidentStreetLinked{
```

```
  all s:Street, a: Accident | a in s.accidents iff s in a.
    street
}

///////////////////////////////////////////////////////
//INTERVENTIONS
///////////////////////////////////////////////////////

//an intervention is suggested for a street iff the number
    of reports in
//that street that contain violation types linked to that
    intervention
//is greater than 0 OR the number of accidents in that
    street linked to
//that intervention is greater than 0
fact InterventionSuggestedThreshold{
  all s: Street, i:Intervention| (i in s.
    suggestedInterventions iff
    (#getReportsFromInterventionAndStreet[i, s] > 0 or
    #getAccidentsFromInterventionAndStreet[i, s] > 0))
}

//gets all the reports for that street, having as violation
    type one linked to the
//given intervention
fun getReportsFromInterventionAndStreet[i : Intervention, s
    : Street] : set Report{
  {
    r : Report | i in r.violationType.interventions and r in
     s.reports
  }
}

//same thing as above but for accidents
fun getAccidentsFromInterventionAndStreet[i: Intervention, s
     : Street] : set Accident{
  {
    a : Accident | i in a.type.interventions and a in s.
    accidents
  }
}


///////////////////////////////////////////////////////
//UNSAFE STREETS
///////////////////////////////////////////////////////

--a street is considered unsafe iff
--the number of total accidents in that street is greater
    than 1 OR
--if the number of total violations in that street is
    greater than 1

fact UnsafeStreetThreshold{
```

```
  all s: Street | s.status = Unsafe iff (
    #s.reports > 1 or
    #s.accidents > 1
  )
}

////////////////////////////////////////////////////
//REGISTRATION
////////////////////////////////////////////////////

fact NoRegistrationWithoutCustomer{
  all r: Registration | one c:Customer | r in c.registration
}

fact NoUsernameWithoutRegistration{
  all u:Username | some r:Registration | u in r.username
}

fact NoPasswordWithoutRegistration{
  all p: Password | some r: Registration | p in r.password
}

fact UniqueUsernames{
  no disj r1,r2: Registration| r1.username = r2.username
}

fact CityAuthorityLink{
  all c:City, a:Authority | a in c.authority iff c in a.city
}

////////////////////////////////////////////////////
//REQUESTS AND RESULT
////////////////////////////////////////////////////
//requests

sig ReportsRequest {
  authority : one Authority,
  violationTypes : some ViolationType,
  dates : some Date,
  time : some Time,
  vehicleTypes: some VehicleType,
  street : lone Street

}

sig InterventionsRequest{
  city : one City
}

////////////////////////////////////////////////////
//results

sig ReportsResult{
  request : one ReportsRequest,
```

```
  storedReports : set StoredReport
}

//this is the rule to apply the filter that comes with the ReportsRequest
//only those reports that satisfy these conditions will be included in the result
fact ReportsResultRule{
  all sr:StoredReport, result:ReportsResult | sr in result.storedReports iff
  (
    sr.report.violationType in result.request.violationTypes and
    sr.report.date in result.request.dates and
    sr.report.time in result.request.time and
    sr.report.vehicleType in result.request.vehicleTypes and
    (result.request.street = none implies sr.report.position.street.city in
      result.request.authority.city else sr.report.position.street in
        result.request.street
    )
  )
}

sig InterventionsResult{
  request : one InterventionsRequest,
  interventions : set Intervention
}


//this is the rule to apply the filter that comes with the InterventionsRequest
//only those interventions that satisfy the following conditions will be included
//in the result
fact InterventionsResultRule{
  all i:Intervention, result:InterventionsResult | i in result.interventions iff
  (
    some s:Street | s in result.request.city.streets and
      i in s.suggestedInterventions
  )
}

//every request has exactly one result
fact OneResultForOneRequestReports{
  all req : ReportsRequest | one res :ReportsResult | req in res.request
}

fact OneResultForOneRequestIntervention{
  all req : InterventionsRequest | one res : InterventionsResult | req in res.request
```

```
}

////////////////////////////////////////////////////
//REPORTS
////////////////////////////////////////////////////

//the system must not store reports that have the exact same
    fields
//nor reports that are considered to be duplicate
fact NoDuplicateStoredReports{
  no disj r1,r2 : StoredReport |
    r1.report = r2.report or
    (
      r1.report.position = r2.report.position and
      r1.report.violationType = r2.report.violationType and
      r1.plate = r2.plate and
      r1.report.date = r2.report.date
    )
}

fact CorrectStreetNameInReport{
  all sr: StoredReport | sr.streetName in
    sr.report.position.street.streetName
}

//the system correctly receives and store reports sent by
    users
//even if there's no authority registered for the city to
    which the
//position of the report belongs
fact UnreadReportIfNoAuthority{
  all sr: StoredReport | sr.report.position.street.city.
    authority = none implies
    sr.status = Unread
}

////////////////////////////////////////////////////
//PREDICATES AND COMMANDS
////////////////////////////////////////////////////

pred world1{
  #Report = 0 and
  #ReportsRequest = 0 and
  #InterventionsRequest = 0 and
  #ViolationType = 0 and
  #AccidentType = 0 and

  #City = 3
  #Authority = 2
  #User = 2
}

run world1 for 4

```

```
pred world2{
  #City = 1 and
  #ReportsRequest = 0 and
  #InterventionsRequest = 0 and
  #StoredReport = 2 and
  #Accident = 2 and
  #AccidentType = 1 and
  #Registration = 2 and
  #Intervention = 2
}

run world2 for 2

pred world3{
  #City = 1 and
  #InterventionsRequest = 1 and
  #ReportsRequest = 0 and
  #Report = 1 and
  #AccidentType = 0 and
  #Accident = 0
  #Intervention = 1
}

run world3 for 2
```

4.2 First World

In this first world, the focus is on the registration and on the map structure. A street is considered as a set of position, and one position belongs to exactly one street. Inside a city, all the streets must have different names, but this constraint is not valid anymore if we consider two different cities. Usernames are unique, but there's no constraint on customer's passwords. An authority is always registered with a city, but a city could have no authority registered.

4.3 Second World

This second world captures a few aspects of the model: they are essentially reports, accidents and interventions. The threshold to activate the suggestion of an intervention has been lowered to 1 here. That means that if there's a report for a street, and that report has a violation type that is linked to an intervention, that intervention will be suggested by the system for that street. Alternatively, the same applies for accidents that are registered for a particular street. The two conditions for activating a suggested intervention are put in 'or'. The street name retrieved by the system while storing the report is of course the name of the street that contains the position that comes with that report. A stored report could be read by the authority. In case it's not, it will be part of the unread reports list for that authority.

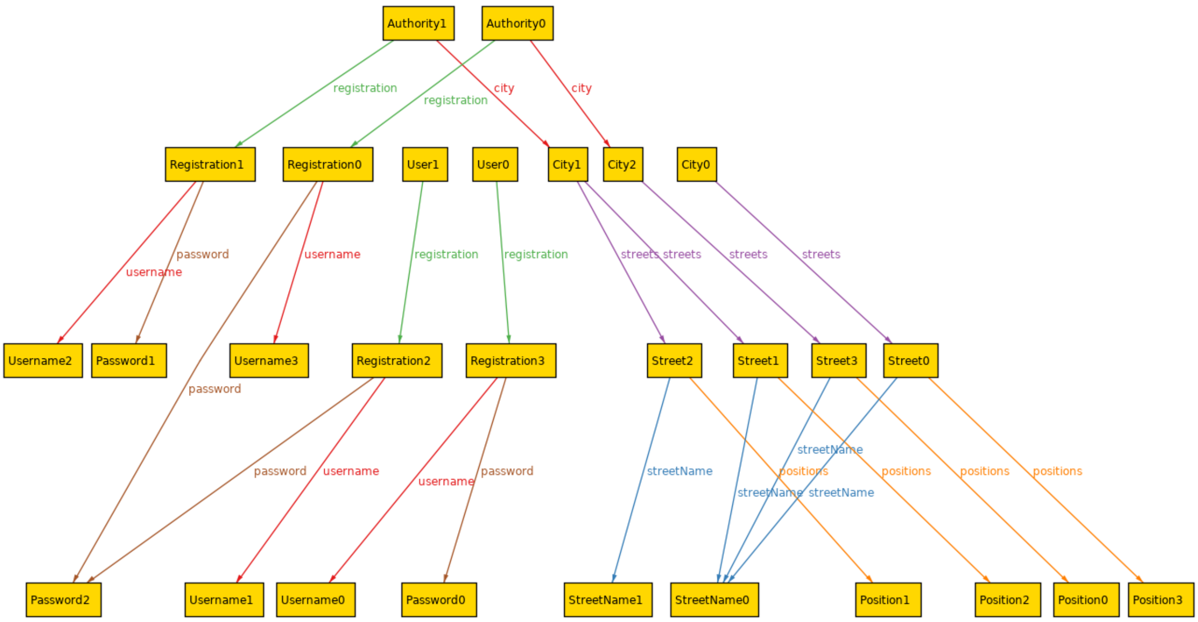


Figure 31: First World generated

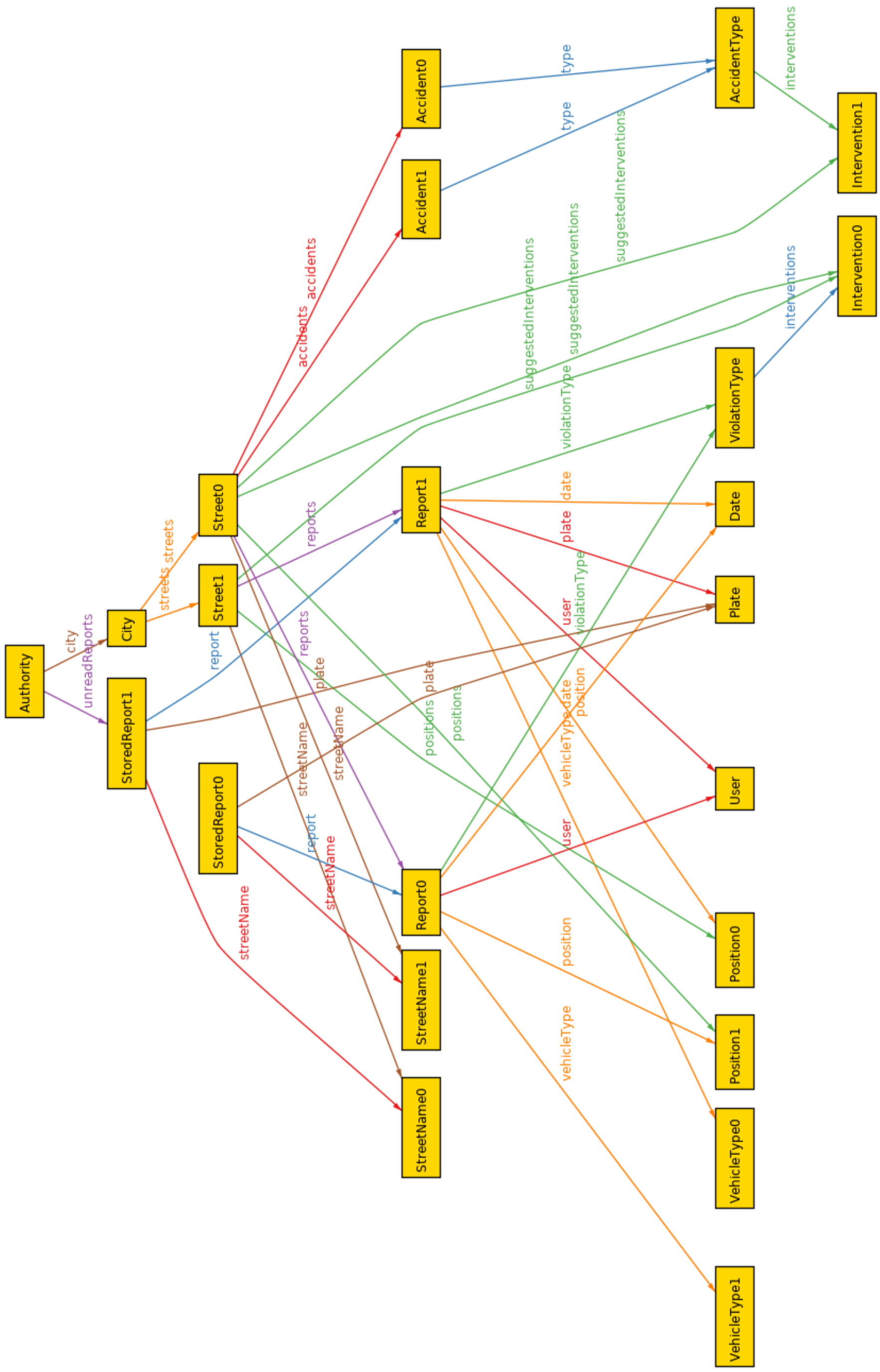
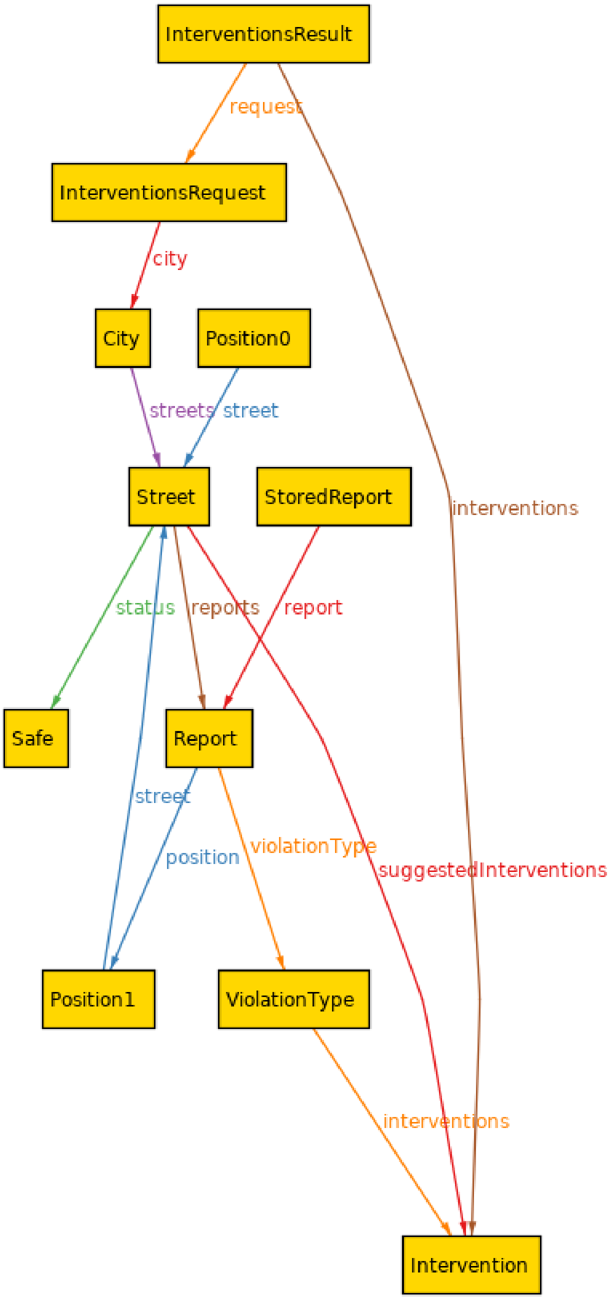


Figure 32: Second World Generated

4.4 Third World

This third and last example world focuses on the request mechanism. The use case is the one where a customer asks the system for possible interventions in a particular city. The result of that request will contain all the interventions suggested by the system, considering all the streets of the specified city.



5 Effort Spent

5.1 Teamwork

Task	Teamwork's Hours
Introduction	0.5
Product Perspective	2
Product Functions	3
Domain Assumptions	1
External Interface Requirements	0.5
Functional Requirements	6
Non-functional Requirements	1
Formal Analysis Using Alloy	5

Table 16: Teamwork effort

5.2 Individual Work

Task	Matteo Pacciani's Hours
Introduction	0.5
Product Perspective	2
Product Functions	3
Domain Assumptions	3
External Interface Requirements	1.5
Functional Requirements	10
Non-functional Requirements	1
Formal Analysis Using Alloy	15

Table 17: Matteo's effort

Task	Francesco Piro's Hours
Introduction	2
Product Perspective	7
Product Functions	11
Domain Assumptions	1
External Interface Requirements	4
Functional Requirements	6
Non-functional Requirements	2
Formal Analysis Using Alloy	3

Table 18: Francesco's effort

6 References

[1] ISO/IEC/IEEE 29148. *Systems and software engineering — Life cycle processes — Requirements engineering*. Technical report, 2011.

[2] IEEE Std 830-1998. *IEEE Recommended Practice for Software Requirements Specifications*. Technical report, 1998. URL: http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf.

[3] alloytool.org. *Alloy Documentation*. Software Design Group. URL: http://alloytools.org/documentation.html.

[4] Michael Jackson. The world and the machine, 1995. URL: http://mcs.open.ac.uk/mj665/icse17kn.pdf.