POLITECNICO
MILANO 1863

AA 2019/2020

Computer Science and Engineering

Software Engineering 2 Project

SafeStreets

RASD

Requirement Analysis and Specification Document

version 1.0 --- 10/11/2019

Authors:
Aida Gasanova
Alexandre Batistella Bellas
Ekaterina Efremova

Professor:
Elisabetta Di Nitto

Table of Contents

1 Introduction

1.1 Purpose

1.1.1 General purpose

This document represents the Requirement Analysis and Specification Document (RASD). The main purpose of this document is to fully describe the software product in order to help developers model it.

This document describes SS application, which can help the prevention of traffic violations, and in particular parking violations, by sending the information obtained by ordinary pedestrians who are users of this application to the authorities. Both sides, users and authorities, can use this data for useful purposes, for example, see areas that have a high frequency of violations, or even the vehicles that commit the most violations. Besides this, the application can cross its own data with external data from municipally (if available) to identify potentially unsafe areas and suggest possible interventions. Lastly, the application can provide information to a municipally system that emits traffic tickets to people that committed violations, since the information came from the application is with guaranteed integrity.

1.1.2 Goals

- G1: Users should be allowed to report traffic violations, and in particular parking violations;
- G2: The picture sent by a user should have credibility automatically calculated;
- G3: Users and authorities should be allowed to mine the information about violations;
- G4: The interventions should be created from the crossing information of accidents and violations;
- G5: Interventions should be allocated to an unsafe area;
- G6: Users and authorities should see truthful information about the registered violations;
- G7: Statistics from the issued tickets should be built.

The goals G4 and G5 can only be considered if the municipality offers a service that allows information retrieval about the accidents that occur on the territory of the municipality.

1.2 Scope

The SS service is a crowd-sourced application offered to common users and authorities that want to follow the violations occurred on the municipality territory.

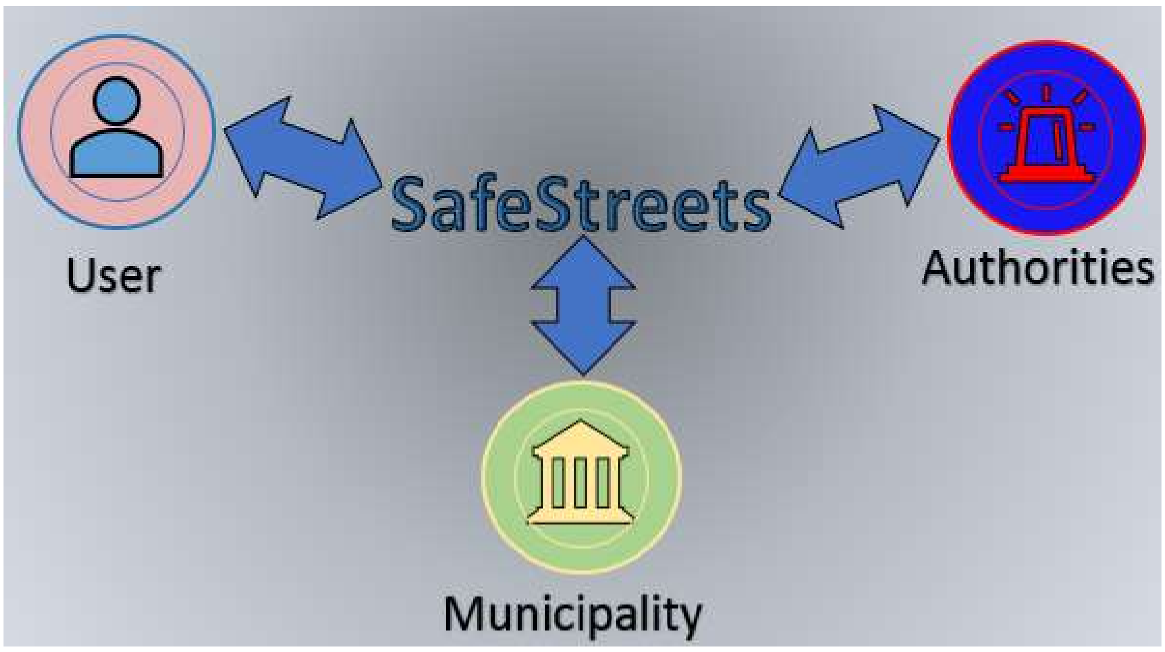


Figure 1. *SS as a sharing information system*

The system has various directions of action. First, the largest mass of users are ordinary people. Everyone can download the application to their phone and use it to the benefit of the welfare of their city. authorities

Every time when the citizens see the violation, they can take a photo and upload it to this application. Based on this information, a register of violations will be compiled for mapping threats on the streets of the city.

When the app receives a picture, it runs an algorithm to read the license plate, and stores the retrieved information with the violation, including also the type of the violation and the name of the street where the violation occurred. In addition, the application allows both end users and authorities to mine the information that has been received, by highlighting the streets (or the areas) with the highest frequency of violations, and the vehicles that commit the most violations. In this case there are more user's levels lake the municipality and authorities, and different levels of visibility are offered to different roles.

Another, also a very important part of users is the municipality. Its role is divided into two parts. First, the municipality can upload accident information to the application, thereby complementing existing databases. The application mixes information from users and from the municipality. As a result, we get a new map with more relevant information about the city. Secondly, the municipality can receive information about offenses from the application. This application initially checks the accuracy of the information (including date, Photoshop using, etc.). Based on this information, the municipality, in cooperation with the police, may issue fines.

Police could also offer a service that takes the information about the violations coming from SS and generates traffic tickets from it. In this case, mechanisms should be put in place to ensure that the chain of custody of the information coming from the users it never broken, and the information is never altered.

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

- **User:** the "normal" customer of the application that send the information about the violations to authorities or extract the information that have been received (to use it for useful purposes);
- **Authorities:** the customer of the application that receive the information about violations that have been received from "normal" customers;
- **Customer:** general SS customer;
- **Municipality:** a town or district that has local government;
- **Violation:** general traffic violation, and in particular parking violation;

1.3.2 Acronyms

- **API:** Application Programming Interface
- **GPS:** Global Positioning System
- **UI:** User Interface
- **AI:** Artificial Intelligence
- **RASD:** Requirement Analysis and Specification Document
- **SS:** SafeStreets

1.3.3 Abbreviations

- **Gn:** n-th goal
- **Dn:** n-th domain assumption
- **Rn:** n-th functional requirement

1.4 Revision history

- 1.0.0 – Release version

1.5 Reference documents

- Specification document: "Mandatory Project Assignment AY 2018-2019"
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications
- UML diagrams: https://www.uml-diagrams.org/
- Alloy doc: http://alloy.lcs.mit.edu/alloy/documentation/quickguide/seq.html

1.6 Document structure

The RASD document is composed by five chapters, as outlined below.

Chapter 1 is an introduction: it describes the purpose of the system informally and also by making use of the list of goals which the application has to reach. Moreover, it defines the scope, where the aim of the project is defined in greater detail and the application domain and the most important shared phenomena are shown.

Chapter 2 offers an overall description of the project. Here the actors involved in the application's usage lifecycle are identified and the boundaries of the project are defined, listing all the necessary assumptions. Furthermore, a class diagram is provided, aid to better understanding the general structure of the project, with all the related entities. Then some state diagrams are listed to make 10 the evolution of the crucial objects clear. Finally, the functions offered by the system are here more clearly specified, with respect to the previously listed goals.

Chapter 3 represents the body of the document. It contains the interface requirements, which are: user interfaces, hardware interfaces and software interfaces. It then lists some scenarios to show how the system acts in real world situations, followed by the description of the functional requirements, using use cases and sequence diagrams. All the requirements necessary in order to reach the goals are given, linked with the related domain assumptions. Lastly, the non-functional requirements are defined through performance requirements, design constraints and software system attributes.

Chapter 4 contains the Alloy model of some critical aspects with all the related comments and documentation in order to show how the project has been modeled and represented through the language.

Chapter 5 contains the list of the tools used and shows the effort which each member of the group spent working on the project.

2 Overall Description

2.1 Product perspective

The system will be developed from scratch, and it'll make interface with the service provided by the municipality when dealing about the accidents occurred on the territory of the own municipality and when providing the violations to it as well. Besides this, the application will need to use the device's GPS service to register the location where the pictures were taken, and also the API from an online service for showing a map inside the application, specifically for the function of highlighting the streets with higher numbers of violation.

Below there is the class diagram of the application, with a high-level visualization of the most important classes that will be implemented by the developers. In the diagram is clear the relations between each actor of the system, for example, every user can provide pictures of a seen violation, upload in the system and make it available for the authorities to check, pointing the relation between users and authorities.

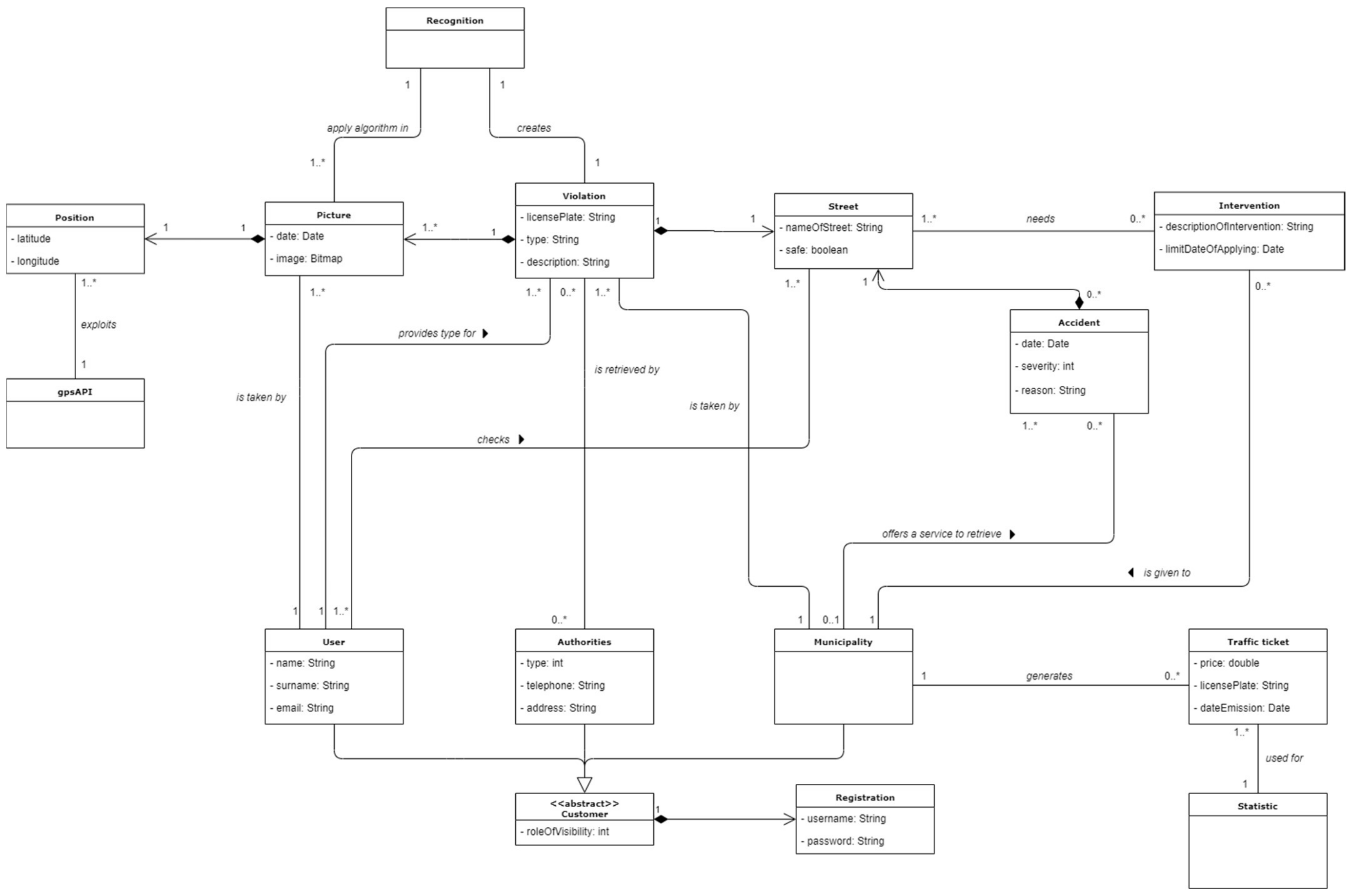


***Figure 2**. Class diagram of the software project*

Now we are going to model the dynamic behavior of the system to represent the condition of the system or part of the system at finite instances of time. To describe the states of different objects in its life cycle we will use statechart diagrams. States can be identified as the condition of objects when a particular event occurs. These diagrams reported below.

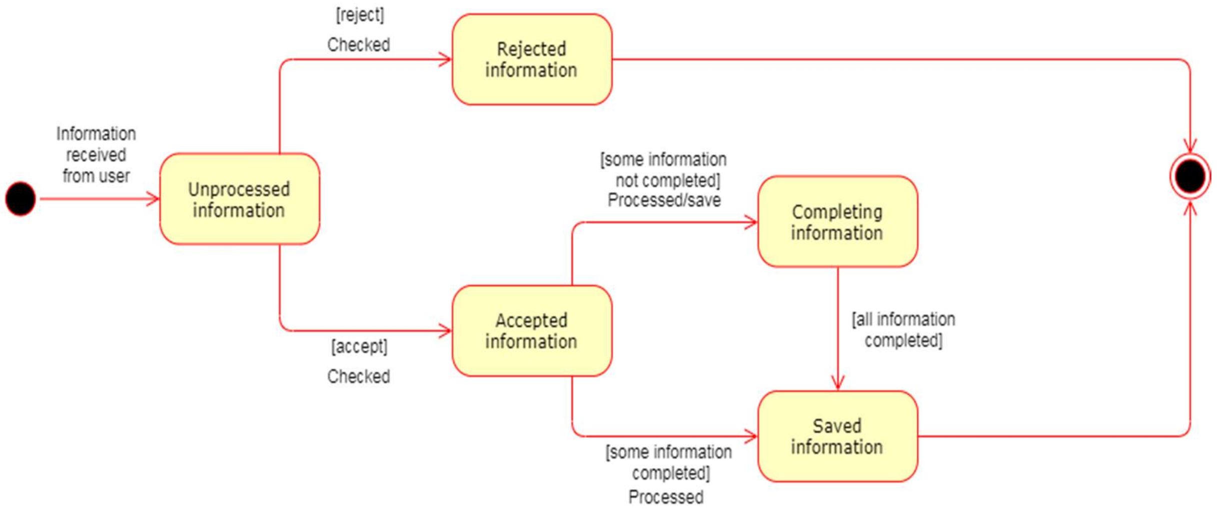

[reject]
Checked
Rejected information
Information received from user
Unprocessed information
[some information not completed] Processed/save
Completing information
Accepted information
[accept]
Checked
[all information completed]
Saved information
[some information completed] Processed

Figure 3. *State diagram 1 – Receiving the information from user.*

The figure 1 represents receiving an information about a violation from user to SS. On the event of an information being received, we transit from our initial state to *Unprocessed information* state. The information can be rejected or accepted depending on ensuring that the chain of custody of the information coming from the users is never broken, and the information is never altered. Also, we need to be sure that the information coming from users is relevant. If the order is rejected, we transit to the *Rejected Information* state. If the information is accepted and it's completed, we transit to the *Saved information* state. If the information is not completed, we need to finalize it with corresponding metadata. For example, when we receive the picture, it launches the license plate reading algorithm. When we have all the information about the violation, including the type of the violation and the name of the street name where the violation occurred, SS saves the data. After the information is completed, we transit to the final state.

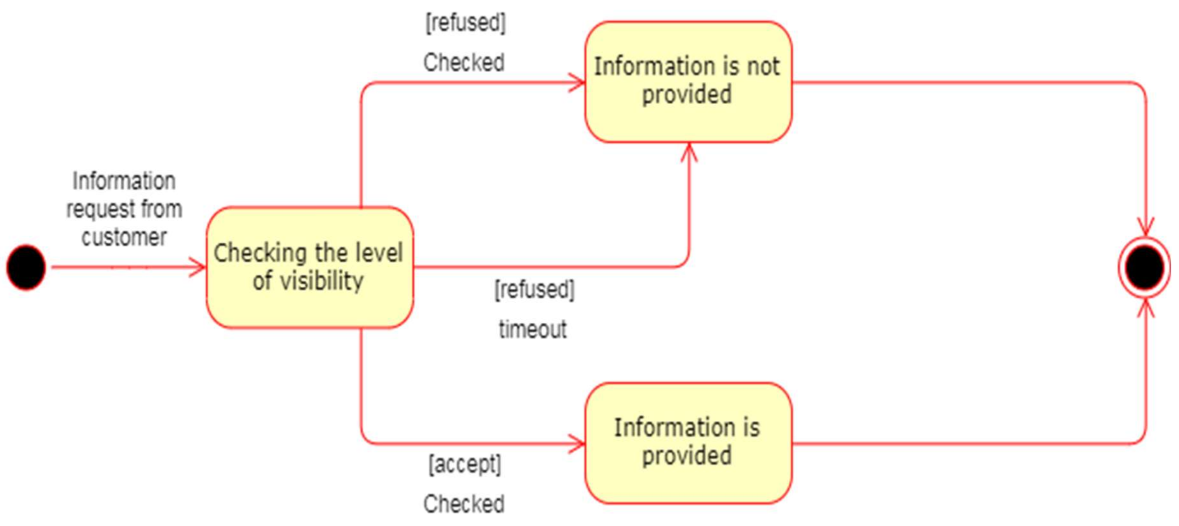

[refused]
Checked
Information is not provided
Information request from customer
Checking the level of visibility
[refused]
timeout
Information is provided
[accept]
Checked

Figure 4. *State diagram 2 – Mining the information from SS.*

In this state diagram (figure 4) before providing the information to customers, first we need to check the level of visibility. For example, both and authority and users can see highlighting the streets with the highest frequency of violations, but only authority can see vehicles that commit the most violations.

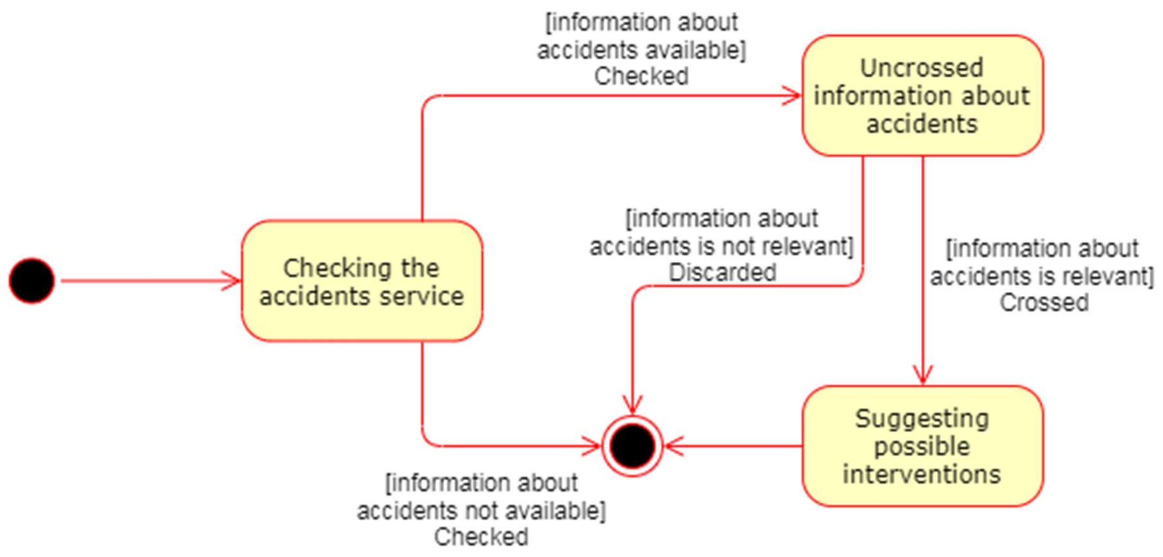

[information about accidents available]
Checked
Uncrossed information about accidents
Checking the accidents service
[information about accidents is not relevant]
Discarded
[information about accidents is relevant]
Crossed
Suggesting possible interventions
[information about accidents not available]
Checked

Figure 5. *State diagram 3 – Using an information from accidents service in SS.*

The figure 5 shows that SS can cross the information about accidents provided by service of municipality with its own data (if the information is relevant) and suggest possible interventions (e.g., add a barrier between the bike lane and the part of the road for motorized vehicles to prevent unsafe parking).

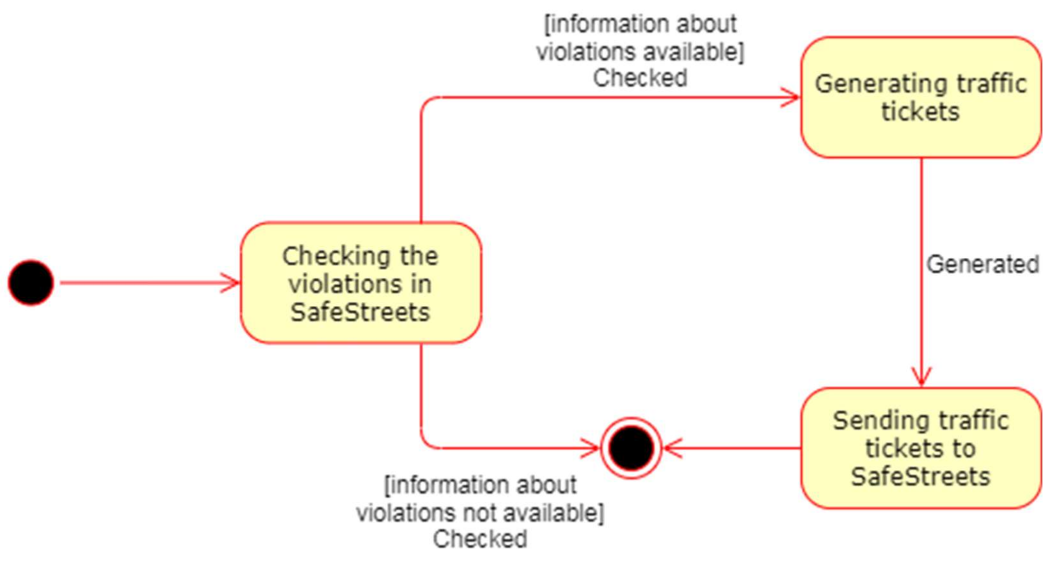

[information about violations available] Checked
Generating traffic tickets
Checking the violations in SafeStreets
Generated
Sending traffic tickets to SafeStreets
[information about violations not available] Checked

Figure 6. *State diagram 4 – Using an information from SafeStreets in traffic tickets service.*

On the state diagram 4 we can see that another service, provided by municipality that can use the information about violations from SS to generate traffic tickets and then provide the tickets to SS, so that SS can build some statistics.

2.2 Product functions

For this section, it's defined all the most important product functions for the development of the project. Since that the application depends on external services, as the one provided by the municipality (if provided), it's needed to define different functions for each service provided by each user of the system.

2.2.1 Violations management

As the main function of the system (basic service), all the users of the application will be allowed to do a registration putting some of their own personal data and choosing an available role (citizen or authority), all of this information being mandatory to insert. In case of being authority, a more complex verification will be needed, as a photo of the work license and a manual validation of this photo. Otherwise, only a photo of the personal document will be needed for verification. For municipality, a professional contact will be needed for providing the offered services from the application via own API, and no registration inside the app is need.

After registration, the users with the role of citizens will be able to upload to the application pictures taken from their mobile devices of violations seen in the street. These pictures, translated as instances of violations inside the app through the recognition algorithm, can be retrieved as a map highlighting the streets with the highest frequency of violations (for both citizen and authority) and a ranking of vehicles with most violations (only for authorities). Besides this, all the violations can be retrieved only by the municipality for applying traffic tickets on the violators. Lastly, the information provided with the pictures, which was uploaded by the user in the system, can be edited by the author, as well that it's possible to see (only for the author) the history of the last done uploads.

2.2.2 Interventions management

This function will only be performed by the system if there is a service from the municipality offering information about the accidents in the territory of itself. In case of this statement is accomplished, all the information about the accidents is retrieved by SS via municipality's API, and with this, the data stored in the system (about violations) will be crossed with the new data (about accidents). If the cross can be performed, an analysis is applied and a possible intervention is created for the cross, being sent to the municipality as a suggestion.

2.2.3 Traffic tickets management

Lastly, the municipality can have the benefit from the system service about the violations occurred in the territory of itself, retrieving it by an API offered by the system. Thus, it can generate traffic tickets for the offenders reported by the users through the application. From the violation's management, the information provided by the users will be always verified and processed to be legit for the municipality apply all the punishments. After issuing the tickets, this information will be used by SS to build statistics about the effectiveness of the SS initiative.

2.3 User characteristics

2.3.1 Actors

- **User:** a person who uses the application not only for viewing but also can control the situation in more detail. He gives photos of violations and upload them to applications for further verification.

- **Authorities:** this organization can track road information as a regular user. It also has access to view user information.

- **Municipality:** this is an organization that has the right to view relevant information. It can also download accident information for further processing by the application. Police could also offer a service that takes the information about the violations coming from SS and generates traffic tickets from it.

2.4 Assumptions, dependencies and constraints

- D1: Different levels of visibility are offered to different roles.
- D2: The municipality (and the local police) offers a service that takes the information about the violations coming from SS and generates traffic tickets from it.

- D3: The municipality uploads accident information.
- D4: Users upload pictures of violations, including their date, time, and position, to the application.
- D5: SS control the accuracy of the information.
- D6: The application sorts sections of the city by rating of violations and creates a highlight map.
- D7: The devices on which the services are exploited can provide real time information.
- D8: The internet connection works properly without failure.
- D9: The devices' GPS can get a location with a precision up to 5 meters.
- D10: The local police belong to the municipality actor.

Any device that can connect to internet through a browser (for example a PC, a mobile phone, a tablet, etc.) is enough for both the user and authorities to do those activities (ex: signing up, logging in, upload pictures, see the map) that don't require data acquisition through devices. But, to exploit these various functionalities needing an internet connection, devices with a 2G/3G/4G and, possibly, Wi-Fi connection, must be present on the users' device.

However, some services offered by the system require some other specific hardware: in order to locate the user, it's necessary for him to have a GPS device and to correctly determine its location on the streets. For what concern the exploited software interfaces, the system uses an external suitable service so that its architecture is simpler.

Finally, the SS must be compliant with the GDPR normative for the privacy.

3 Specific Requirements

3.1 External interface requirements

3.1.1 User interfaces

The following mockups show how the SS application approximately should look like. There are some of the most important screenshots of the interaction between SS and users and authority.

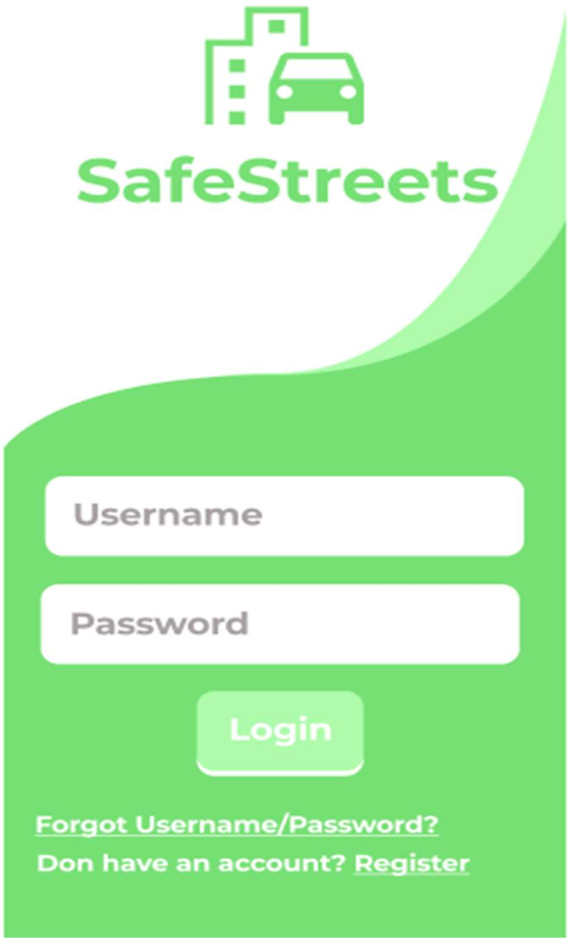


Figure 7. *Mockup - Login page.*

The figure 7 show the login page of SS. The login page is the same for all customers, but different functions available for different roles. The system defines the role after login.

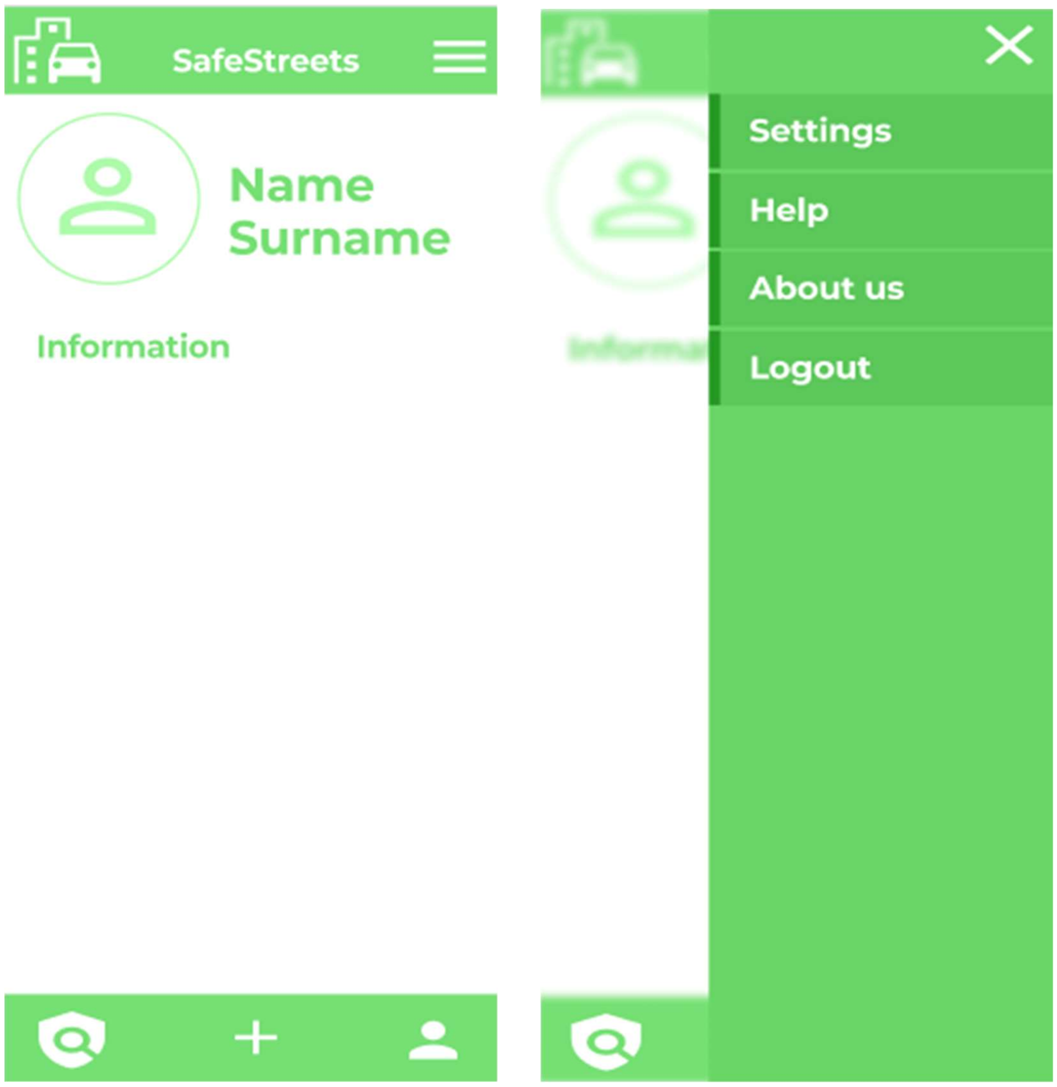


Figure 8. *Mockup – User page*

The figure 8 represents the user interface. Below the page user can see available functions. The first one is highlighting the streets with the highest frequency of violations (it can be more different functions; this one is just an example). The second one is reporting a violation. The last one shows the profile. The button above on the right opens menu which is the same for any customer.

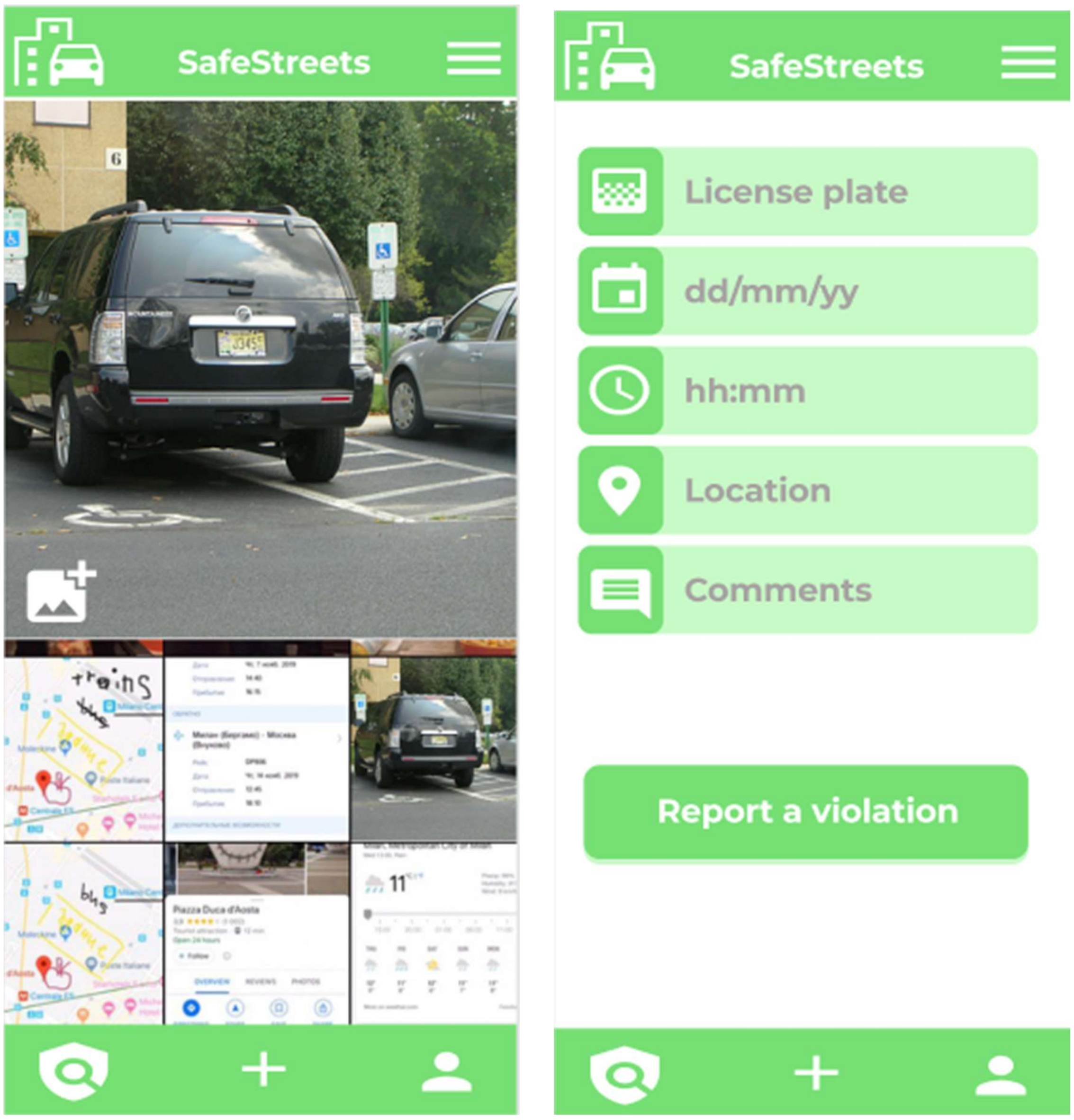


Figure 9. *Mockup – Reporting a violation.*

The most important function about reporting a violation is represented on the figure 9. First the user has to choose a picture with traffic violation and then he sees the page where he has to add an information about the violation, including their date, time and location. He also can add some comments. SS runs the algorithm to read the license plate on the picture, but user can change it as well if something is wrong.

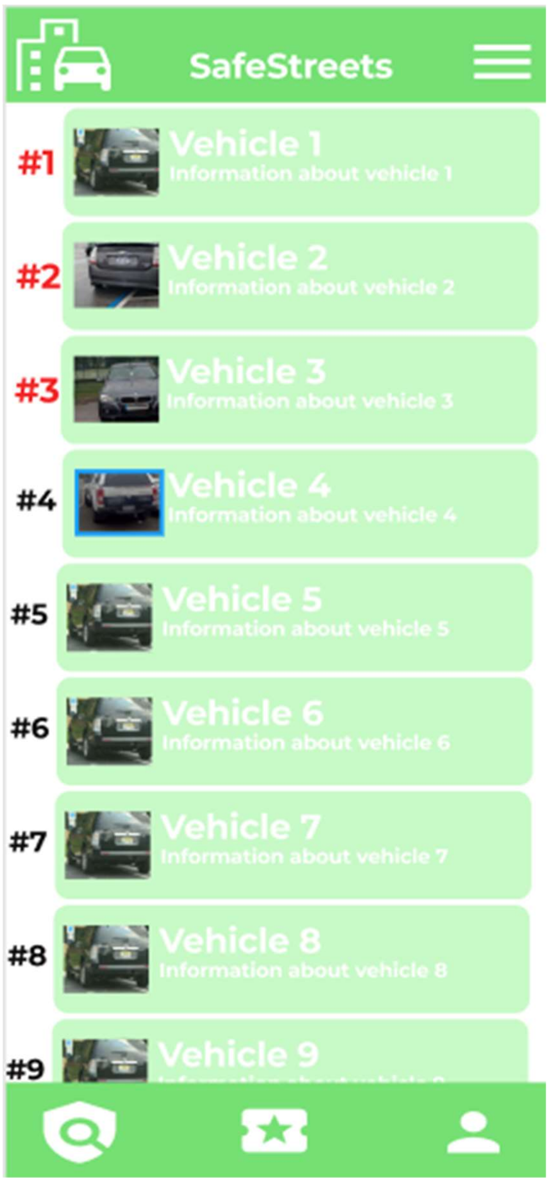

SafeStreets
#1
Vehicle 1
Information about vehicle 1
#2
Vehicle 2
Information about vehicle 2
#3
Vehicle 3
Information about vehicle 3
#4
Vehicle 4
Information about vehicle 4
#5
Vehicle 5
Information about vehicle 5
#6
Vehicle 6
Information about vehicle 6
#7
Vehicle 7
Information about vehicle 7
#8
Vehicle 8
Information about vehicle 8
#9
Vehicle 9

Figure 10. *Mockup – Authority page.*

On the figure 10 we can see that functions below the page are different for authority. The figure shows rating vehicles that commit the most violations. After selecting one of them you can see the entire history of violations about a particular vehicle.

3.1.2 Hardware interfaces

The system has no hardware interfaces.

3.1.3 Software interfaces

First, it is worth paying attention to the structure of the connection of the municipality and the application. The full picture can be seen in the Sequence diagrams section.

From the beginning, the SS sends a request to the municipality about the accidents. The municipality processes this request and sends the list to the application. If you look from the other side, the SS can send a request to the municipality about the list of suggested interventions and get the necessary information.

Our assumption is the municipality will connect to the system via our own API, so it will get the violations via an API that we will offer to it. besides this, the own municipality needs to offer an API for us to retrieve the information about the accidents. With this, we don't need the municipality to create an account.

Further it is worth considering the interaction of the application with authority. Because authorities have a certain level of access, they request information about ordinary users.

3.2 Functional requirements

3.2.1 User

Scenarios

Scenario 1

Marshall is a 50-year-old man. He lives alone and loves to walk the streets of his city. Totally aw-abiding citizen. He does not like offenses, because every time he walks the streets, he notices small violations. He wants to change the situation in his city, so he downloaded the application to somehow influence the situation by sending photos. He believes that if everyone uses this application, the situation will become better.

Scenario 2

Laura, 20 years old. She is studying at university. Recently I passed on the rights and is trying to get comfortable with my new car in the city. It is difficult for her to give parking and driving rules. He uses the navigator exclusively to find the route of travel since he does not understand the roads. Laura does not know the subtleties of parking in various streets. But does not want to break them. To do this, she uses applications to find suitable places, so that the probability of violation is less.

Use case diagram

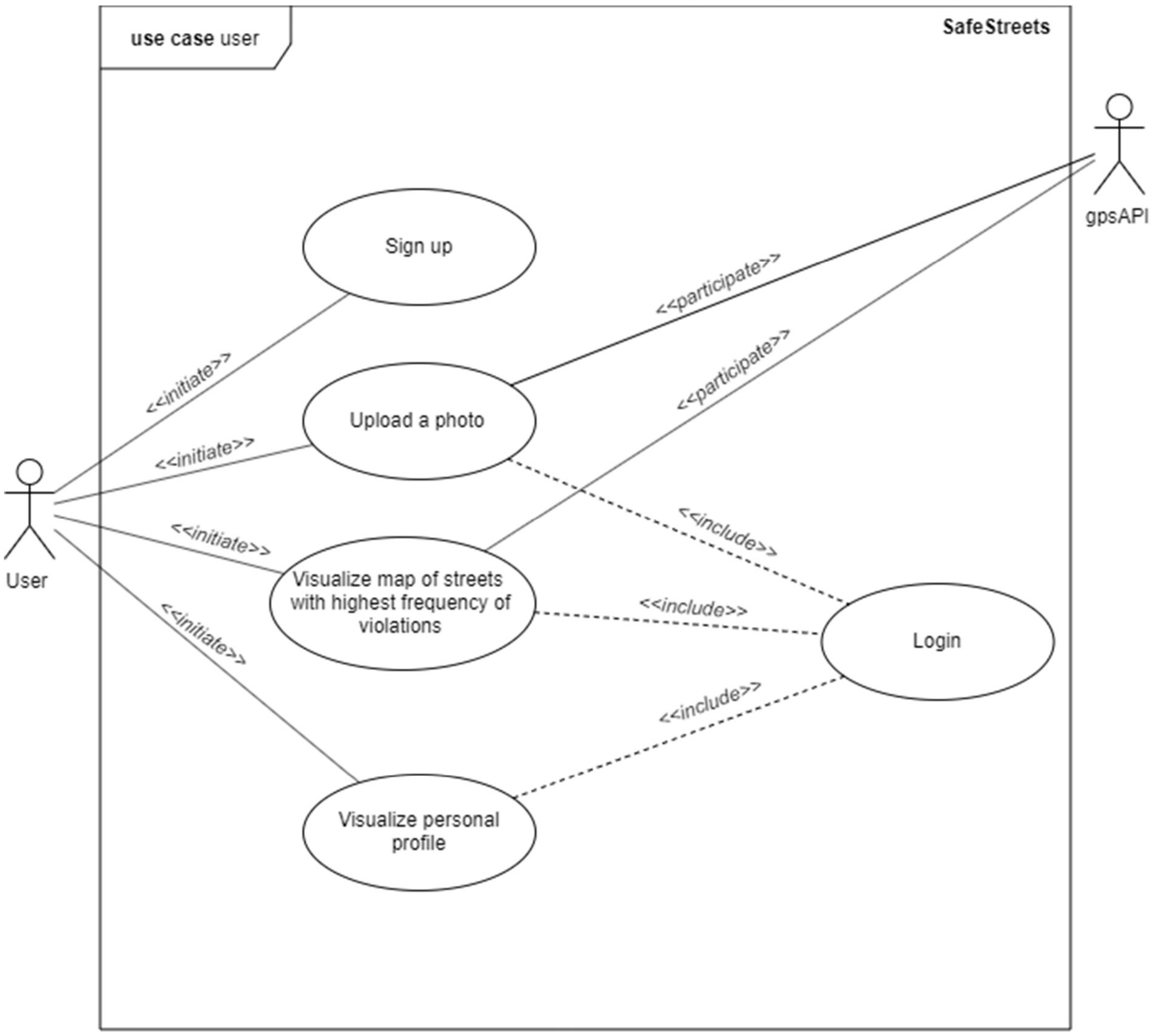


Figure 11. *Use case diagram for user*

Use case description

Name	Login
Actor	User
Entry conditions	1. The user has opened the application in his device. 2. The user has already done the sing up procedure. 3. The user is not signed in yet.
Events flow	1. The user opens the application.

	2. The user enters his username and password correctly. 3. The user selects the "Login" button.
Exit conditions	The user is logged in and can interact with the system, visualizing the available data and managing his account details.
Exceptions	1. The user put a wrong username. 2. The user put a wrong password. The system warns the user about one of the fields is incorrect, without specifying which one.

Name	Sign up
Actor	User
Entry conditions	The user opens the application on his device.
Events flow	1. The user chooses the "Register" option. 2. The user fills the mandatory personal details options. 3. The user presses the button of confirmation. 4. The system processes the data.
Exit conditions	The user is registered and the account is created by the system.
Exceptions	1. The user has already registered. In this case, the system warns the user about the divergency and doesn't continue the process. 2. The username already exists. The system warns the user about the already used username, and suggests a new available one next to the written one. 3. The password is too weak. In this case, the system warns the user to get a stronger password.

Name	Upload a photo
Actor	User
Entry conditions	1. The user has already logged in the application. 2. The user has already taken a photo with his device. 3. The user is with his application opened.
Events flow	1. The user selects the “+” option. 2. The user selects a photo from the device gallery. 3. The user fills the mandatory information about the taken photo. 4. The user fills the non-mandatory information with optional data. 5. The user presses the button “Report a violation” 6. The system collects the information. 7. The system gives a feedback about the uploaded photo and information.
Exit conditions	The system recognizes the credibility and stores the picture and the information successfully in the system.
Exceptions	1. The user didn’t put all the mandatory information. In this case, the upload is not done and the system warns the user about the missing information. 2. The photo didn’t have credibility enough to be considered. The user is warned about his last upload, and the photo is deleted in his profile and in the system.

Name	Visualize map of streets with highest frequency of violations
Actor	User
Entry conditions	1. The user has already logged in the application.

	2. The user is with the application open on his device.
Events flow	1. The user clicks in the respective button for the map page. 2. The user is able to see the updated map about the violations occurred in the area visualized.
Exit conditions	The user can see the map with the highlighted streets whose are the most violation ones.
Exceptions	There are no exceptions.

Name	Visualize personal profile
Actor	User
Entry conditions	1. The user has already logged in the application.
Events flow	1. The user selects the respective button for the profile. 2. The system provides all the data about his personal profile in the app.
Exit conditions	The user can see all his personal profile details.
Exceptions	There are no exceptions.

Sequence diagrams

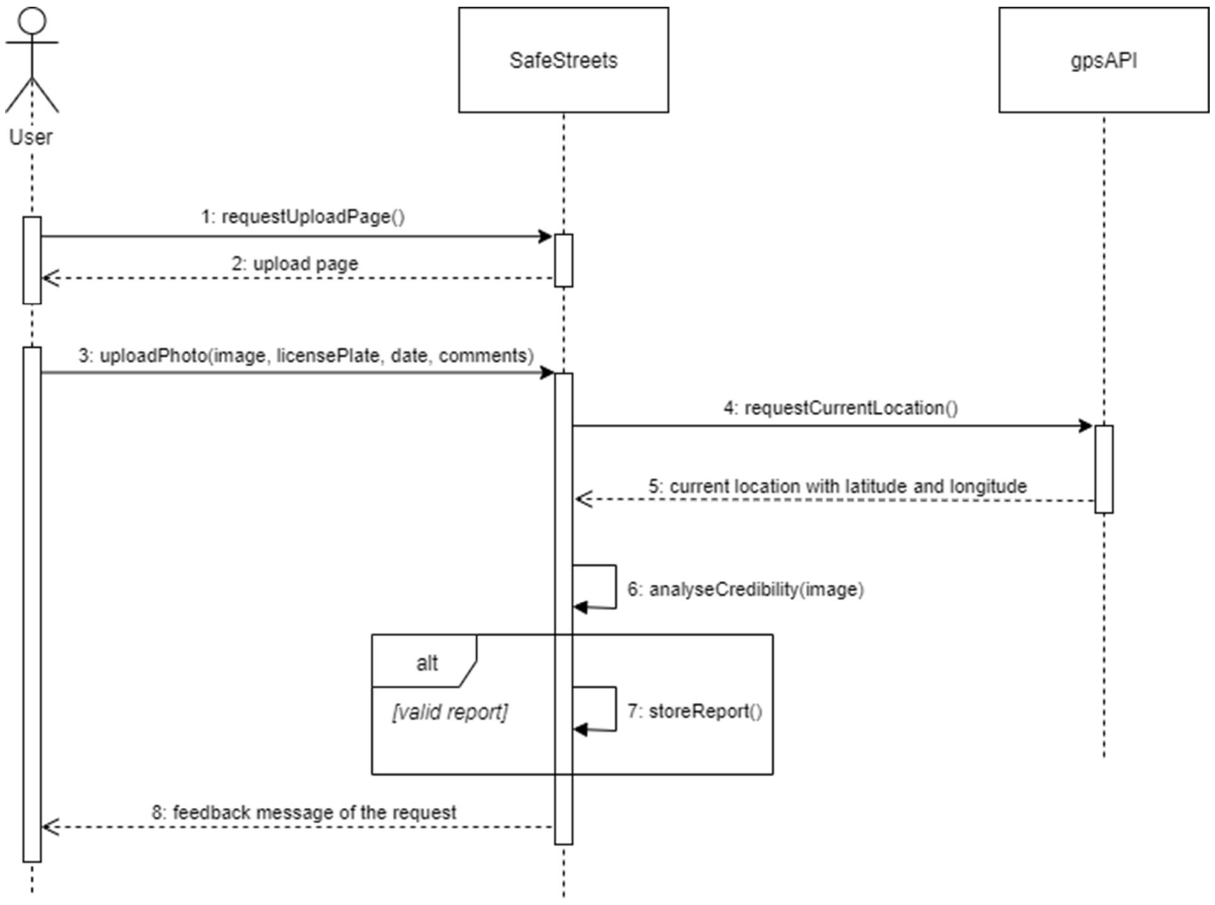

User
SafeStreets
gpsAPI
1: requestUploadPage()
2: upload page
3: uploadPhoto(image, licensePlate, date, comments)
4: requestCurrentLocation()
5: current location with latitude and longitude
6: analyseCredibility(image)
alt
[valid report]
7: storeReport()
8: feedback message of the request

Figure 12. *Sequence diagram: Upload a photo*

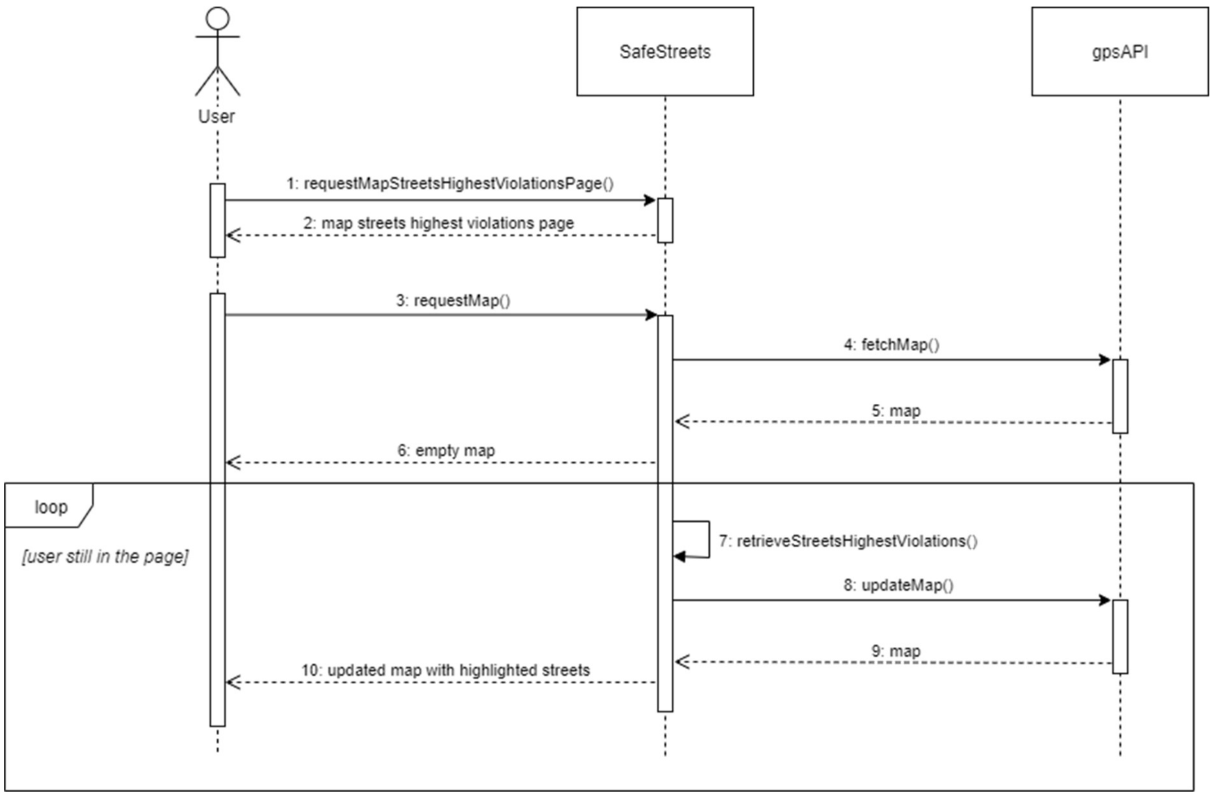

SafeStreets
gpsAPI
User
1: requestMapStreetsHighestViolationsPage()
2: map streets highest violations page
3: requestMap()
4: fetchMap()
5: map
6: empty map
loop
[user still in the page]
7: retrieveStreetsHighestViolations()
8: updateMap()
9: map
10: updated map with highlighted streets

Figure 13. *Sequence diagram: Visualize map of streets with highest frequency of violations*

3.2.2 Authorities

Scenarios

Scenario 3

Ivan, 45 years old, works in the Department of Health. Very concerned about the situation on the roads, in particular violations. To solve this problem, it has an enough level of access to ss to request the vehicles that commit the most violations. In the future, he can analyze this data, look for patterns and solutions to problems with many violations.

Use case diagram

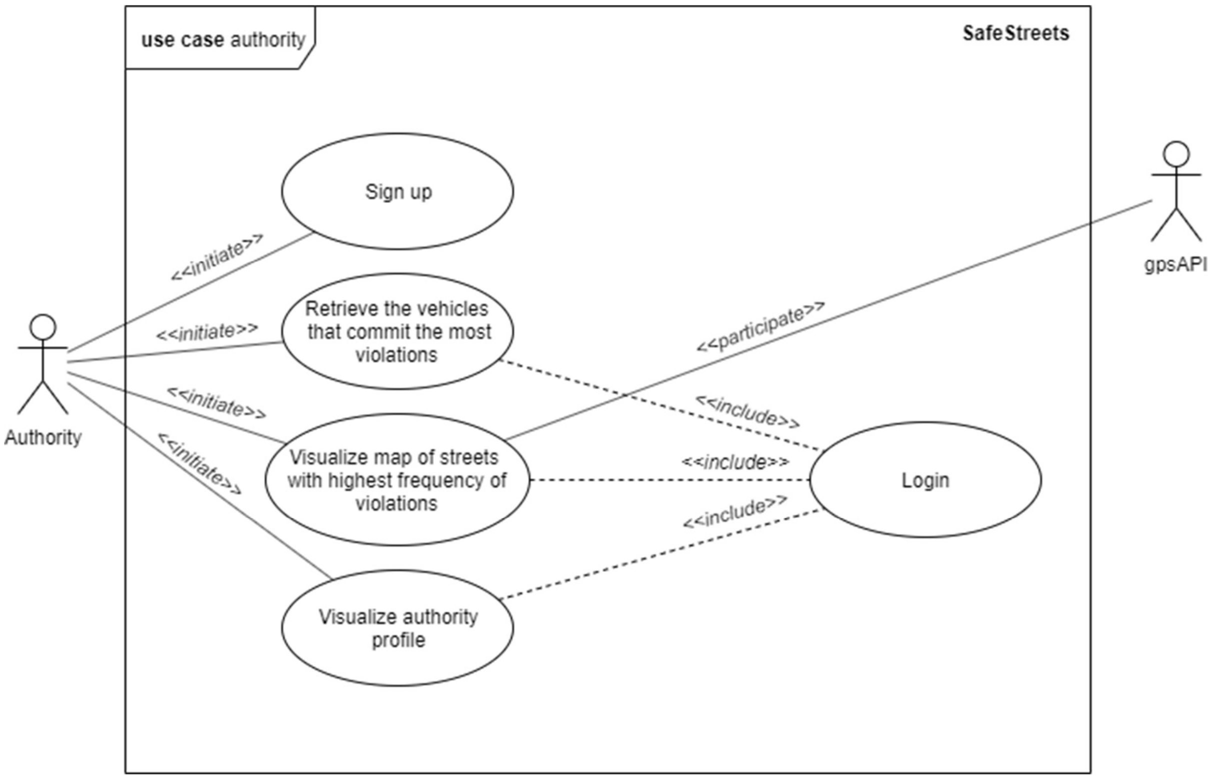


Figure 14. *Use case diagram for authority*

Use case description

Name	Login
Actor	Authority
Entry conditions	1. The authority has opened the application in his device. 2. The authority has already done the sing up procedure. 3. The authority is not signed in yet.
Events flow	1. The authority opens the application. 2. The authority enters his username and password correctly. 3. The authority selects the "Login" button.
Exit conditions	The authority is logged in and can interact with the system, visualizing the available data and managing his account details.

Exceptions	1. The authority put a wrong username. 2. The authority put a wrong password. The system warns the authority about one of the fields is incorrect, without specifying which one.

Name	Sign up
Actor	Authority
Entry conditions	The authority opens the application on his device.
Events flow	1. The authority chooses the “Register” option. 2. The authority fills the mandatory professional details options. 3. The authority presses the button of confirmation. 4. The system processes the data. 5. After manual analyzing, a feedback is sent to the authority, allowing the authority to log in the application with his username and password.
Exit conditions	The authority is registered, and the account is created by the system.
Exceptions	1. The authority has already registered. In this case, the system warns the authority about the divergency and doesn’t continue the process. 2. The username already exists. The system warns the authority about the already used username and suggests a new available one next to the written one. 3. The password is too weak. In this case, the system warns the authority to get a stronger password. 4. The manual analyzing doesn’t accept the registration of the authority. In this case, the authority will receive a notification about the situation and will

	need to register again, putting all the details one more time.

Name	Retrieve the vehicles that commit the most violations
Actor	Authority
Entry conditions	1. The authority has already logged in the application. 2. The authority is with the application open on his device.
Events flow	1. The authority clicks in the respective button for the page in question 2. The authority is able to see the ranking of vehicles that committed more violations and your details
Exit conditions	The authority can see the ranking of vehicles that commit the most violations in the application
Exceptions	There are no exceptions.

Name	Visualize map of streets with highest frequency of violations
Actor	Authority
Entry conditions	1. The authority has already logged in the application. 2. The authority is with the application open on his device.
Events flow	1. The authority clicks in the respective button for the map page. 2. The authority is able to see the updated map about the violations occurred in the area visualized.
Exit conditions	The authority can see the map with the highlighted streets whose are the most violation ones.

Exceptions	There are no exceptions.

Name	Visualize personal profile
Actor	Authority
Entry conditions	The authority has already logged in the application.
Events flow	1. The authority selects the respective button for the profile. 2. The system provides all the data about his personal profile in the app.
Exit conditions	The authority can see all his personal profile details.
Exceptions	There are no exceptions.

Sequence diagrams

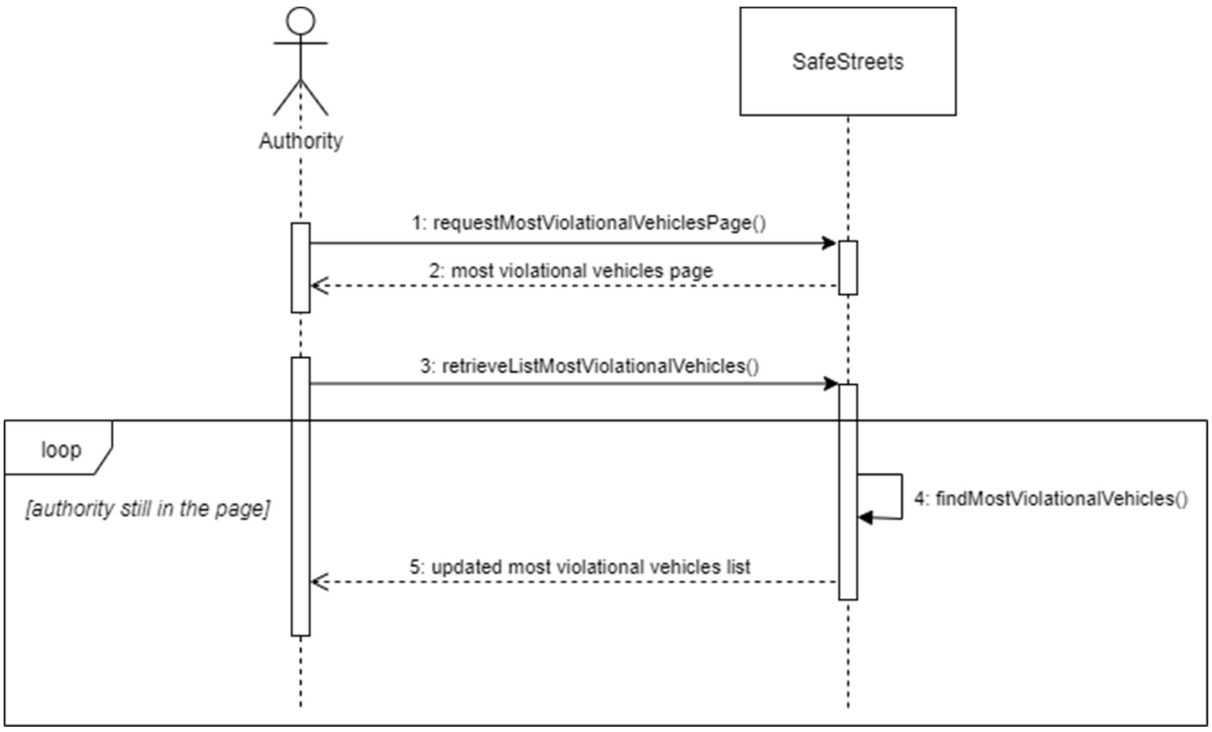

Authority
SafeStreets
1: requestMostViolationalVehiclesPage()
2: most violational vehicles page
3: retrieveListMostViolationalVehicles()
loop
[authority still in the page]
4: findMostViolationalVehicles()
5: updated most violational vehicles list

Figure 15. *Sequence diagram: Retrieve the vehicles that commit the most violations*

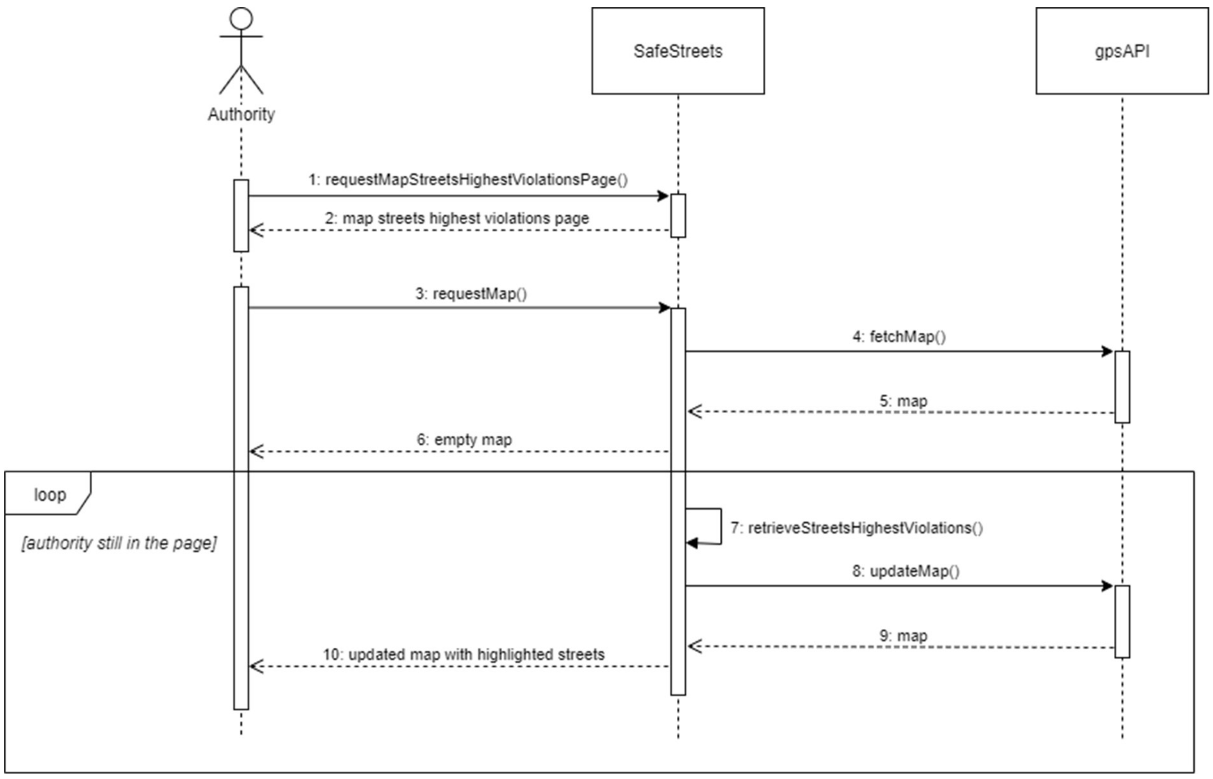

Authority
SafeStreets
gpsAPI
1: requestMapStreetsHighestViolationsPage()
2: map streets highest violations page
3: requestMap()
4: fetchMap()
5: map
6: empty map
loop
[authority still in the page]
7: retrieveStreetsHighestViolations()
8: updateMap()
9: map
10: updated map with highlighted streets

Figure 16. *Sequence diagram: Visualize map of streets with highest frequency of violations*

3.2.3 Municipality

Scenarios
Scenario 4

Peter. 30 years old, work in the police. Before the application, he made a detour in search of parking violations several times a week and spent a lot of time on it. Thanks to the cooperation with the application, he can receive reliable photographs of violations for issuing penalty tickets. It has an access level to query the intruder database.

Scenario 5

The municipality department is notified of the accident on Seventh Street. The scale of the accident is such that the traffic is completely blocked. This situation can create big problems for the residents of the city who built their route along this road. They will be stopped for at least two hours because they will have to wait for the solution to the accident and everyone will be able to disperse extremely slowly. Thus, the department sends information about this accident to the application, warning users about traffic obstruction. Users can see this and significantly save time by moving along another road.

Use case diagram

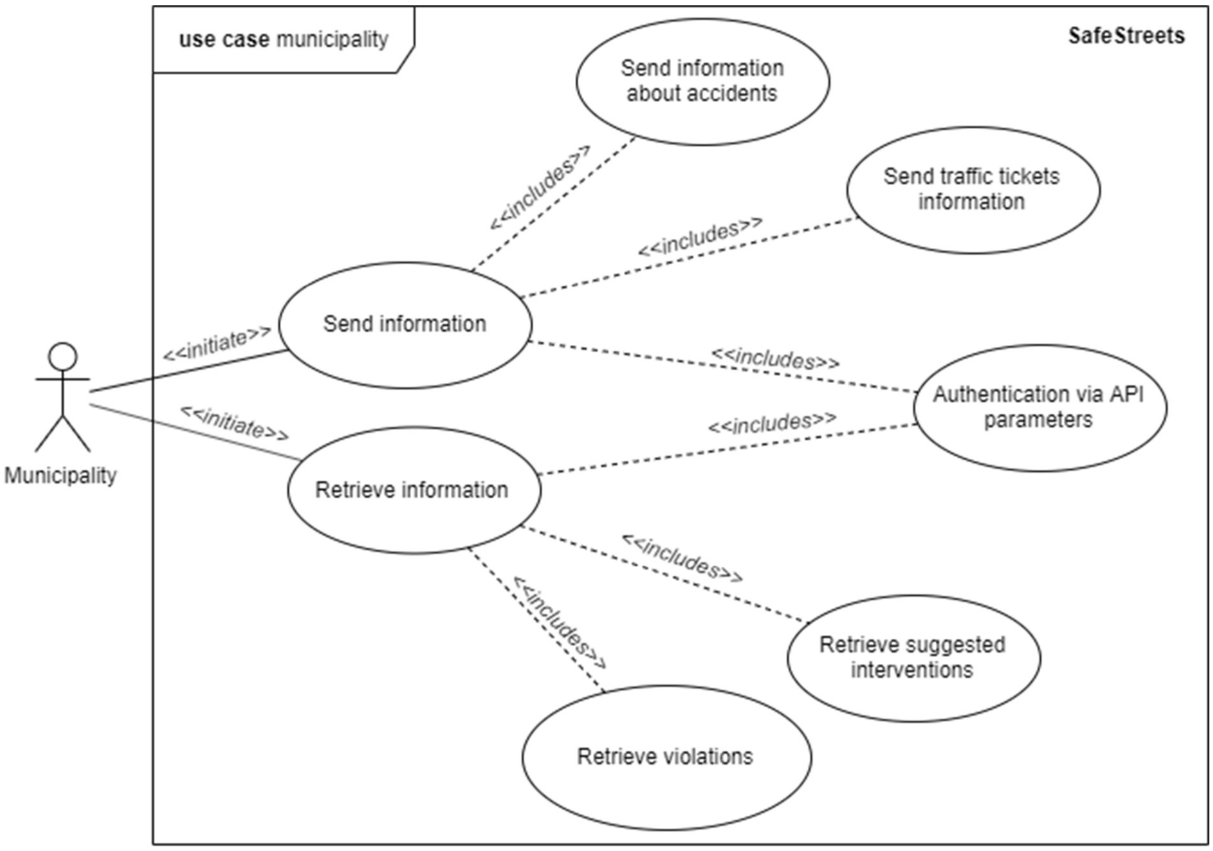


Figure 17. *Use case diagram for municipality*

Use case description

Name	Authentication via API parameters
Actor	Municipality
Entry conditions	1. The municipality has a manual registration in the system 2. The municipality has an own service that can interact with the system 3. The municipality is endowed with a username and password as parameters for the API connection
Events flow	1. The municipality requests a permission to access the system with specific parameters

	2. The municipality has access to provide or retrieve information from the system
Exit conditions	The system provides the access to the municipality and it is able to exchange information.
Exceptions	1. The parameters are invalid. In this case, the access is denied, and the municipality needs to restart the request. 2. A forbidden request is made, even with right parameters The request is rejected, and a warning is sent as response.

Name	Send information
Actor	Municipality
Entry conditions	1. The municipality has access to the system
Events flow	1. The municipality requests the permission to send data 2. The system returns a message allowing the data dispatch 3. The municipality sends the data to the system 4. The system processes the data and stores it
Exit conditions	All the data is sent successfully, and it is stored in the system.
Exceptions	1. The data is not fully sent In this case, the system warns the municipality and the request needs to be restarted. 2. The system is not ready to receive the data In this case, the message returned after the request is warning to do not send the data.

Name	Retrieve information
Actor	Municipality
Entry conditions	1. The municipality has access to the system
Events flow	1. The municipality requests the permission to retrieve data from the system 2. The system returns a message allowing the operation 3. The municipality specify the data to be retrieved 4. The system sends the data to the municipality
Exit conditions	The municipality has received all the data.
Exceptions	1. The data is not fully sent 2. The system is not ready to send the data. In both cases, the municipality is warned to restart the request.

Sequence diagrams

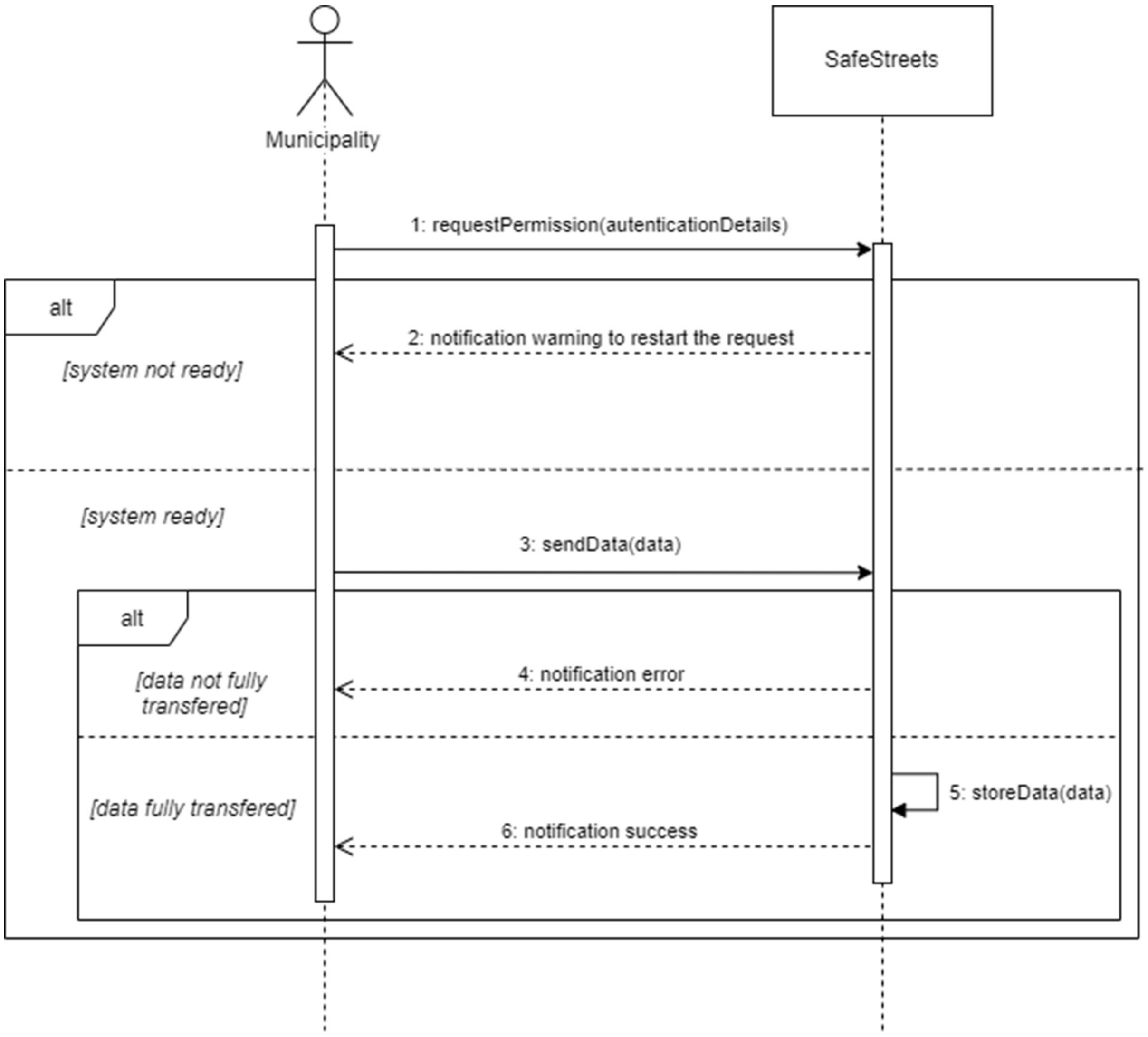
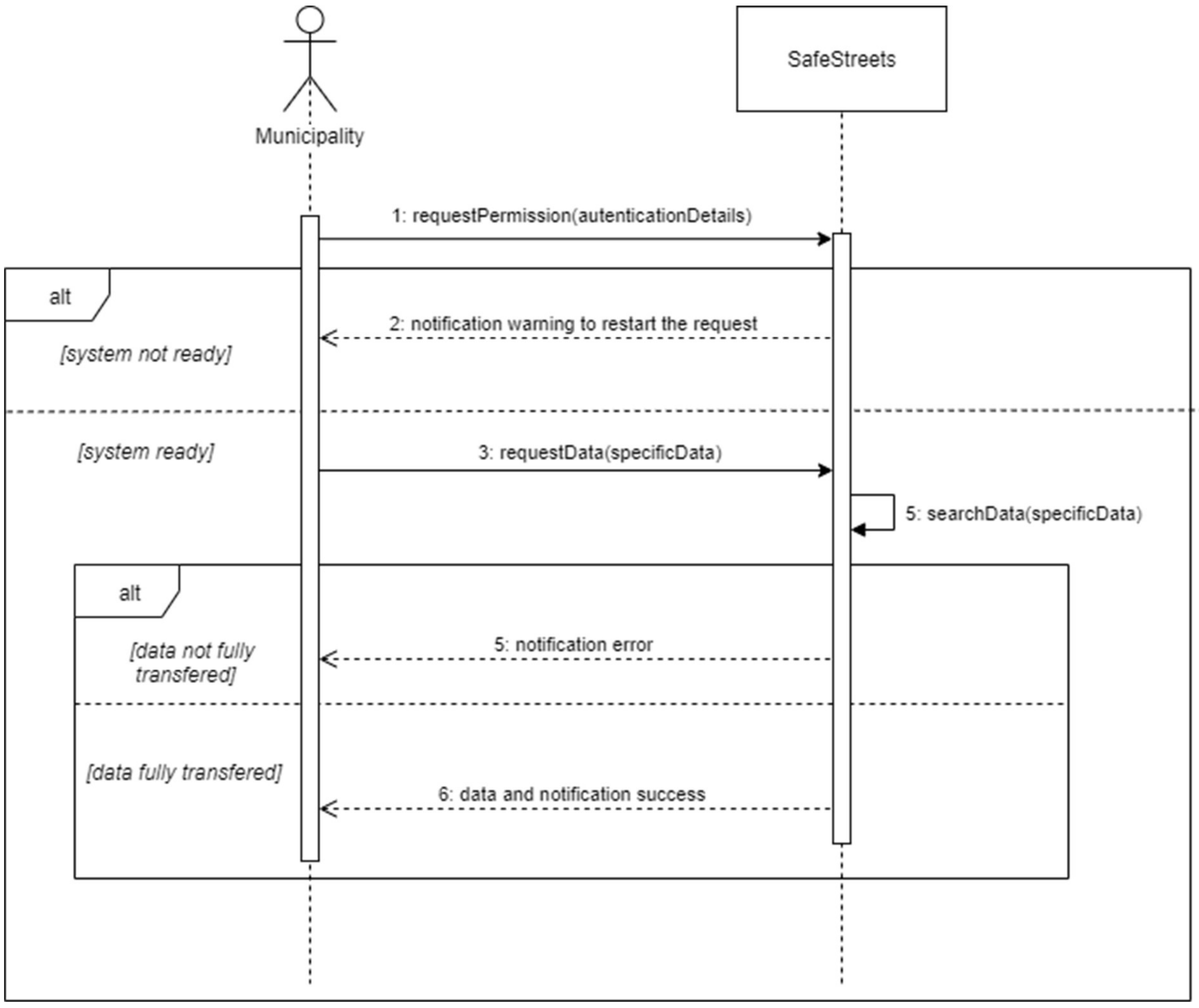


Figure 18. *Sequence diagram: Send information*



Municipality
SafeStreets
1: requestPermission(autenticationDetails)
alt
2: notification warning to restart the request
[system not ready]
[system ready]
3: requestData(specificData)
5: searchData(specificData)
alt
[data not fully transfered]
5: notification error
[data fully transfered]
6: data and notification success

Figure 19. *Sequence diagram: Retrieve information*

3.2.4 Requirements

In this section, we show the requirements to ensure the satisfaction of goals in the context of a domain assumption. A list of requirements and assumptions of the domain for each goal is for this purpose.

- G1: Users should be allowed to report traffic violations, and in particular parking violations
  - R1: The system must save the collected data of users registered to SS in real time.
  - D1: The devices that acquire users' position provide location with an error of 5 meters at most.
  - D4: Users upload pictures of violations, including their date, time, and position, to the application.
  - D8: The internet connection works properly without failure.

- G2: The picture sent by a user should have credibility automatically calculated
    - R1: The system must save the collected data of users registered to SS in real time.
    - R2: The system must automatically check photos from manual editions.
    - R3: The photo data (date, time) needs to correspondence with the time of sending.
    - D3: The internet connection works properly without failure.

- G3: Users and authorities should be allowed to mine the information about violations
    - R1: The system must save the sent data from users registered to SS in real time.
    - R4: The authorities can see all the information about violations.
    - R5: Users and authorities have the right to view the map of highlights.
    - D1: Different levels of visibility are offered to different roles.
    - D3: The municipality uploads accident information.
    - D6: The application sorts sections of the city by rating of violations and creates a highlighted map.
    - D7: The devices on which the services are exploited can provide real-time information.
    - D8: The internet connection works properly without failure.
    - D9: The devices' GPS can get a location with a precision of up to 5 meters.

- G4: The interventions should be created from the crossing information of accidents and violations
    - R1: The system must save the collected data of users registered to SS in real time.
    - R6: The municipality sends information about current accidents to the application.
    - R7: The system combines data and issues a highlight map.
    - D3: The municipality uploads accident information.
    - D4: Users upload pictures of violations, including their date, time, and position, to the application.
    - D5: SS controls the accuracy of the information.
    - D8: The internet connection works properly without failure.

- G5: Interventions should be allocated to an unsafe area
    - R8: The system must process the accidents information provided by the municipality.
    - R9: The system must cross the information from the violations to the information from the accidents

  - R10: The system must send to the municipality the intervention generated from the crossing information
  - D3: The municipality uploads accident information.
  - D5: SS controls the accuracy of the information.
  - D7: The devices on which the services are exploited can provide real-time information.
  - D9: The devices' GPS can get a location with a precision of up to 5 meters.
  - D10: The local police belong to the municipality actor.

- G6: Users and authorities should see truthful information about the registered violations
  - R1: The system must save the collected data of users registered to SS in real time.
  - R2: The system must automatically check photos from manual editions.
  - R3: The photo data (date, time) needs to correspondence with the time of sending.
  - D4: Users upload pictures of violations, including their date, time, and position, to the application.
  - D5: SS controls the accuracy of the information.
  - D6: The application sorts sections of the city by rating of violations and creates a highlighted map.

- G7: Statistics from the issued tickets should be built
  - R1: The system must save the collected data of users registered to SS in real time.
  - R11: The application must build statistics from the traffic tickets generated by the municipality.
  - R12: The system must retrieve the information of traffic tickets from the municipality.
  - D2: The municipality (and the local police) offers a service that takes the information about the violations coming from SS and generates traffic tickets from it.
  - D3: The municipality uploads accident information.
  - D10: The local police belong to the municipality actor.

Traceability Matrix

Requirements	Use case
R1	Sign up
R2	Visualize personal data
R3	Upload a photo
R4	Retrieve the vehicles that commit the most violations
R5	Visualize map of streets with highest frequency of violations
R6	Send information
R7	Visualize map of streets with highest frequency of violations
R8	Send information
R9	Send information
R10	Retrieve information
R11	Send information
R12	Send information

3.3 Performance requirements

The system must be able to serve a great number of users and system retrieving from municipalities simultaneously. In case of supporting 300.000 customers, the performance shall not fall below 95% of all visible pages for “normal” customers respond (8 seconds or less), including infrastructure, excluding backends.
The system must guarantee a 24/7 service. Very small deviations from this requirement will be obviously acceptable.

Accuracy of the information provided (positions of accidents, violations) has to be the best possible. All the sensors used must provide positions' data with an error lower than 10 meters (GPS technology is theoretically able to provide locations with an error less or equal to 6 meters).

3.4 Design constraints

3.4.1 Standards compliance

The system adopts the precise units of measure distance [km].

With regard to the privacy of data, since the application processes sensitive ones, the entire project is subject to the General Data Protection Regulation (GDPR), a regulation in EU law on data protection and privacy for all individuals within the European Union (EU) and the European Economic Area (EEA).

3.4.2 Hardware limitations

Hardware requirements are present only for specific functions. For example, each device with an Internet connection is good to register or log in, instead GPS and other sensors are needed. In this case, all requirements are reported, it is necessary to use all the functionality:

- Connection to internet (Wi-Fi/4G/3G/2G)
- GPS

3.4.3 Any other constraint

The system must comply with the privacy policy, in particular the privacy of users. When a user uploads information about violations, he must be sure that the data about him will not be transmitted to random people, including the person whose violation he sent. It is also worth remembering that ordinary users can only see the highlight map. But the databases of users are open only to authorities and the municipality.

3.5 Software system attributes

3.5.1 Reliability

The system must be able to operate continuously without any interruptions or failures. In order to do this, the system must be fault tolerant. For example, a central server that contains data should duplicate it, as well as running processes that provide services. An important point of sustainability is the uploading of large amounts of data at one time by different organizations (for example, the role of the municipality makes penalty tickets) and at this moment the system should work without delay for other ordinary users.

3.5.2 Availability

As mentioned earlier, a fault tolerant architecture is needed. The system may be needed by users at any time, this is an important aspect of the success of this program. The system must guarantee a 24/7 service. Very small deviations from this requirement will be obviously acceptable.

3.5.3 Security

Security of the data provided by users and of the communications customer-system is a primary concern.
The data provided by the user contains confidential information, so the security aspect is of utmost importance. The central database in which the data is located must be protected by all necessary measures to prevent any external and internal attacks, as well as to troubleshoot equipment malfunctions.
To send data, you must use the encryption method to ensure confidentiality and consistency.
It is also worth clarifying that different user levels have different access levels. Therefore, the databases used by authorities must be protected from ordinary users.

3.5.4 Maintainability

The development of the application must be done so that in the future it will be easy to fix and modify it, according to the circumstances, and also in order to let cost of these operations be cheap.
It should be borne in mind that the main costs are incurred during the creation of the application, further changes are possible, but inefficient.

3.5.5 Portability

The system can be said to be portable the effort required for porting it proves significantly less than the effort necessary for a new implementation.

Due to the use of the application by different structures, ease of use during the migration also matters.

4 Formal analysis with Alloy modeling

This part of the document is destinated to make an analysis of the critical questions using the Alloy tool. For this, it's necessary to take into account the following aspects:

- It's forbidden to have more than a customer with the same username;
- The municipality needs to provide a service that offers the accidents information inside its territory, retrieves the information of violations from the system and offers the generated traffic tickets information;
- If a request created by the municipality is not completed, it needs to be notified properly by the system and only in this case, with a clear explanation of the reason;
- The photo uploaded by the user needs to be recognized by the system to make sure that the information is real and has credibility to be used by the municipality.

Below, is the Alloy code for the software project.

```
open util/boolean

//First, all the entities necessary to describe the objects from the class dia
gram
sig Username{}

sig Password{}

sig Date{}

sig Bitmap{}

sig LicensePlate{}

//-------------------------------------------------------------------//
//Now, the description of the objects from the class diagram

//For the position, the values of latitude and longitude are scaled
//Latitude >= -90 and Latitude <= 90
//Longitude >= -180 and Longitude <= 180
sig Position{
    latitude: one Int,
    longitude: one Int
}
{latitude >= -3 and latitude <= 3 and longitude >= -6 and longitude <= 6}

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

```
abstract sig Customer{
    registration: one Registration
}

sig User extends Customer{
    pictures: set Picture,
    violationsSent: set Violation,
    streetsMap: set Street
}

sig Authority extends Customer{
    allViolations: set Violation,
    streetsMap: set Street
}

sig Municipality extends Customer{
    trafficTickets: set TrafficTicket,
    interventions: set Intervention,
    accidents: set Accident,
    allViolations: set Violation,
    warningsReceived: set Warning
}

sig Picture{
    date: one Date,
    image: one Bitmap,
    position: one Position
}

//In each recognition, it's needed to the picture be related to the violation
sig Recognition{
    picture: one Picture,
    violation: lone Violation
}
{picture in violation.pictures}

sig Violation{
    licencePlate: one LicensePlate,
    street: one Street,
    pictures: some Picture,
    violationDate: one Date
}

sig Street{
    safe: one Bool,
    accidentsStreet: set Accident
}
{safe = False iff #accidentsStreet > 0}
```

```
sig Accident{
    accidentDate: one Date
}

sig Intervention{
    targetMunicipality: one Municipality
}

sig TrafficTicket{
    licensePlateTrafficTicket: one LicensePlate,
    dateEmission: one Date
}

sig Statistic{
    data: set TrafficTicket
}

//----------------------------------------------------------------------//
//Errors handling

abstract sig Request{}
abstract sig Warning{}

sig Data{}

one sig TransferWarning extends Warning{}
one sig SystemWarning extends Warning{}

sig SingleRequest{
    authorOfRequest: one Municipality,
    subjectOfRequest: one Data
}

abstract sig SingleRequestResult{
    request: set SingleRequest
}

one sig SingleRequestAccepted extends SingleRequestResult{}
one sig SingleRequestRefused extends SingleRequestResult{}

//Transfer request generates a warning if the data is not fully sent
//The values were scaled for simplicity
sig TransferRequest{
    transferData: set Data,
    authorOfRequestTransfer: one Municipality,
    transferWarning: lone TransferWarning
}
{#transferWarning = 1 iff #transferData < 10}
```

```
//System request generates a warning if the system is not ready
//The values were scaled for simplicity
sig SystemRequest{
    ready: one Bool,
    authorOfRequestSystem: one Municipality,
    systemWarning: lone SystemWarning
}
{#systemWarning = 1 iff ready = False}

//---------------------------------------------------------------------//

//All usernames have to be associated to a Registration
fact UsernameRegistrationConnection{
    all u: Username | some r: Registration | u in r.username
}

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

//All customers have to be associated to a Registration
fact RegistrationCostumerConnection{
    all r: Registration | some c: Customer | r in c.registration
}

//Every customer has a unique username
fact UniqueUsername{
    no disj c1, c2: Customer | c1.registration.username = c2.registration.user
name
}

//The municipality has made a request that has not been accepted if it
//has been notified with the warning
fact TransferWarningOnlyIfNeeded{
    all m: Municipality | TransferWarning in m.warningsReceived implies
    (
        some sr: SingleRequest | m in sr.authorOfRequest and
        (one r: SingleRequestRefused | sr in r.request)
    )
}
fact SystemWarningOnlyIfNeeded{
    all m: Municipality | SystemWarning in m.warningsReceived implies
    (
        some sr: SingleRequest | m in sr.authorOfRequest and
        (one r: SingleRequestRefused | sr in r.request)
    )
}
```

```
//The municipality is notified when a transfer is not concluded
fact TransferWarningIfFailed{
    all sr: SingleRequest | one r: SingleRequestRefused | sr in r.request
    implies
    (
        TransferWarning in sr.authorOfRequest.warningsReceived
    )
}

//The municipality is notified when the system is not ready
fact SystemWarningIfFailed{
    all sr: SingleRequest | one r: SingleRequestRefused | sr in r.request
    implies
    (
        SystemWarning in sr.authorOfRequest.warningsReceived
    )
}

//The municipality is notified if had transfer problems
fact NotFullyTransfered{
    all tr: TransferRequest | #tr.transferData < 10
    implies
    (
        TransferWarning in tr.authorOfRequestTransfer.warningsReceived
    )
}

//The municipality is notified if had system problems
fact NotSystemAvailable{
    all sr: SystemRequest | sr.ready = False
    implies
    (
        SystemWarning in sr.authorOfRequestSystem.warningsReceived
    )
}

//Every single request can be either accepted or refused, not both
fact SingleRequestAcceptedOrRefused{
    all sr: SingleRequest | (some srr: SingleRequestResult | sr in srr.request
)
    and (no disj srr1, srr2: SingleRequestResult | sr in srr1.request and sr i
n srr2.request)
}

//Every picture has an author
fact PictureUserConnection{
    all p: Picture | some u: User | p in u.pictures
}
```

```
//There are no two different violations with the same license plate and the sa
me date
fact DateViolationConnection{
    no disj v1, v2: Violation | v1.licencePlate = v2.licencePlate and v1.viola
tionDate = v2.violationDate
}

//All positions have to be associated to a Picture
fact PositionPictureConnection{
    all pos: Position | some p: Picture | pos = p.position
}

//All accidents have to be associated to a Street
fact AccidentStreetConnection{
    all acc: Accident | some s: Street | acc in s.accidentsStreet
}

//All accidents have to belong to a municipality
fact AccidentMunicipalityConnection{
    all acc: Accident | some m: Municipality | acc in m.accidents
}

//All the inverventions have to belong to a municipality
fact InterventionMunicipalityConnection{
    all i: Intervention | some m: Municipality | i in m.interventions
}

//All the traffic tickets have to belong to a municipality
fact TrafficTicketMunicipalityConnection{
    all tt: TrafficTicket | some m: Municipality | tt in m.trafficTickets
}

//Every bitmap belongs to a picture
fact BitMapPictureConnection{
    all bm: Bitmap | some p: Picture | bm = p.image
}

//Every license plate belongs to a violation
fact LicensePlateViolationConnection{
    all lp: LicensePlate | some v: Violation | lp = v.licencePlate
}

//Every traffic ticket belongs to a license plate
fact TrafficTicketLicensePlateConnection{
    all tt: TrafficTicket | some lp: LicensePlate | lp = tt.licensePlateTraffi
cTicket
}
```

```
//Every date needs to be related to a picture or to an accident or to a traffi
c ticket
fact DateConnection{
    all d: Date | (some p: Picture | d in p.date)
    or
    (some acc: Accident | d = acc.accidentDate)
    or
    (some tt: TrafficTicket | d = tt.dateEmission)
}

//----------------------------------------------------------------------//
//Testing

assert TransferRequestHasWarningCorrectly{
    all tr: TransferRequest | #tr.transferData = 10 iff #tr.transferWarning =
0
}

assert SystemRequestHasWarningCorrectly{
    all sr: SystemRequest | sr.ready = True iff #sr.systemWarning = 0
}

assert NoSingleRequestBothAcceptedAndRefused{
    no sr: SingleRequest | (one a: SingleRequestAccepted | sr in a.request)
    and (one r: SingleRequestRefused | sr in r.request)
}

check TransferRequestHasWarningCorrectly for 5

check SystemRequestHasWarningCorrectly for 5

check NoSingleRequestBothAcceptedAndRefused for 5

//----------------------------------------------------------------------//
//Predicates to generate worlds
//World 1: test requests from municipality
pred world1{
    #SingleRequest = 2
    #SingleRequestAccepted.request = 1
    #SingleRequestRefused.request = 1
    #Municipality = 2
    (some disj m1, m2: Municipality | some disj s1, s2: SingleRequest |
    m1 in s1.authorOfRequest and m2 in s2.authorOfRequest and
    s1 in SingleRequestAccepted.request and s2 in SingleRequestRefused.request
)
}

run world1 for 3 but 0 Intervention, 0 Street, 0 TrafficTicket, 0 Statistic
```

```
//World 2: test transfer request
pred world2{
    #TransferRequest = 1
    #Municipality = 1
    (some t1: TransferRequest | #t1.transferData = 3)
}

run world2 for 3 but 0 Intervention, 0 Street, 0 TrafficTicket, 0 Statistic

//World 3: test system request
pred world3{
    #SystemRequest = 2
    #SingleRequestAccepted.request = 1
    #SingleRequestRefused.request = 1
    #Municipality = 2
    (some disj s1, s2: SystemRequest | s1.authorOfRequestSystem != s2.authorOf
RequestSystem
    and s1.ready = False and s2.ready = True)
}

run world3 for 3 but 0 Intervention, 0 Street, 0 TrafficTicket, 0 Statistic

//World 4: everything
pred world4{
    #Violation = 2
}

run world4 for 2 but 0 SingleRequest, 0 TransferRequest, 0 SystemRequest
```

Below are the worlds generated by the code, after executing all the four predicates.

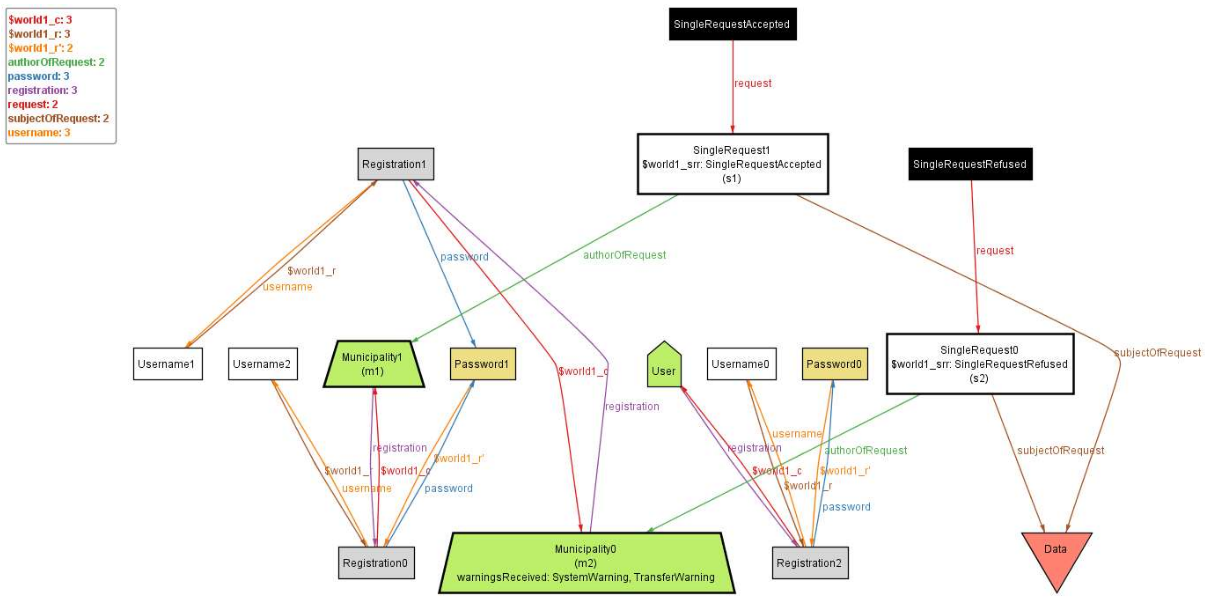


Figure 19. World 1 generated by the Alloy Analyzer

In this world, we can certify that the Municipality is receiving the right warning according to the type of error happened, receiving as well the “accepted” or “refused” feedback from the system. Besides this, we can see that each user has your username and password, not mixing in each other.

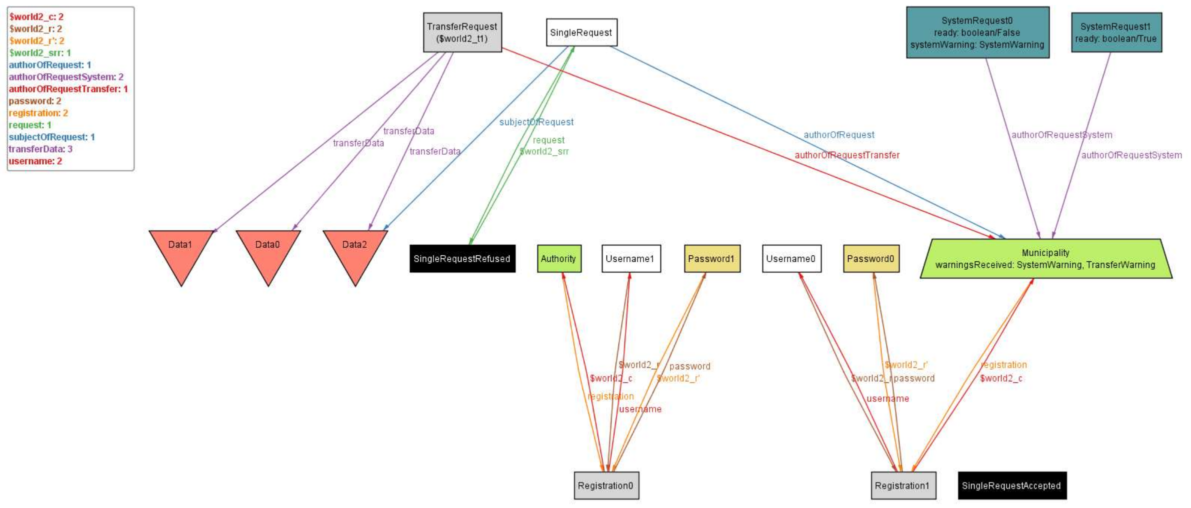


Figure 20. World 2 generated by the Alloy Analyzer

From this world, we can attest that the Municipality created there receives a warning from the system, since that, in the code, it has the condition of “not receiving/sending all the data to/from the system” being, in this case, represented by the number 3 but meaning 30%.

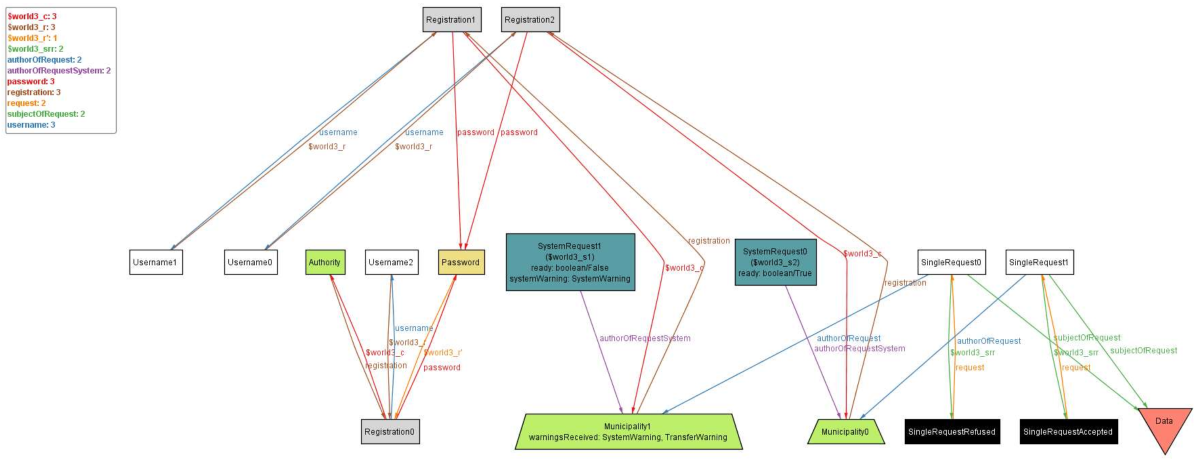


Figure 21. World 3 generated by the Alloy Analyzer

The world 3 represents the different situations for the system: if it is ready or not. For Municipality0, it’s possible to attest that the system is ready from the SystemRequest0, so the feedback for the request is “accepted”. On the other hand, for the Municipality1, it’s the inverse case: it has the “SystemWarning” in its SystemRequest1, and has the feedback “rejected”.

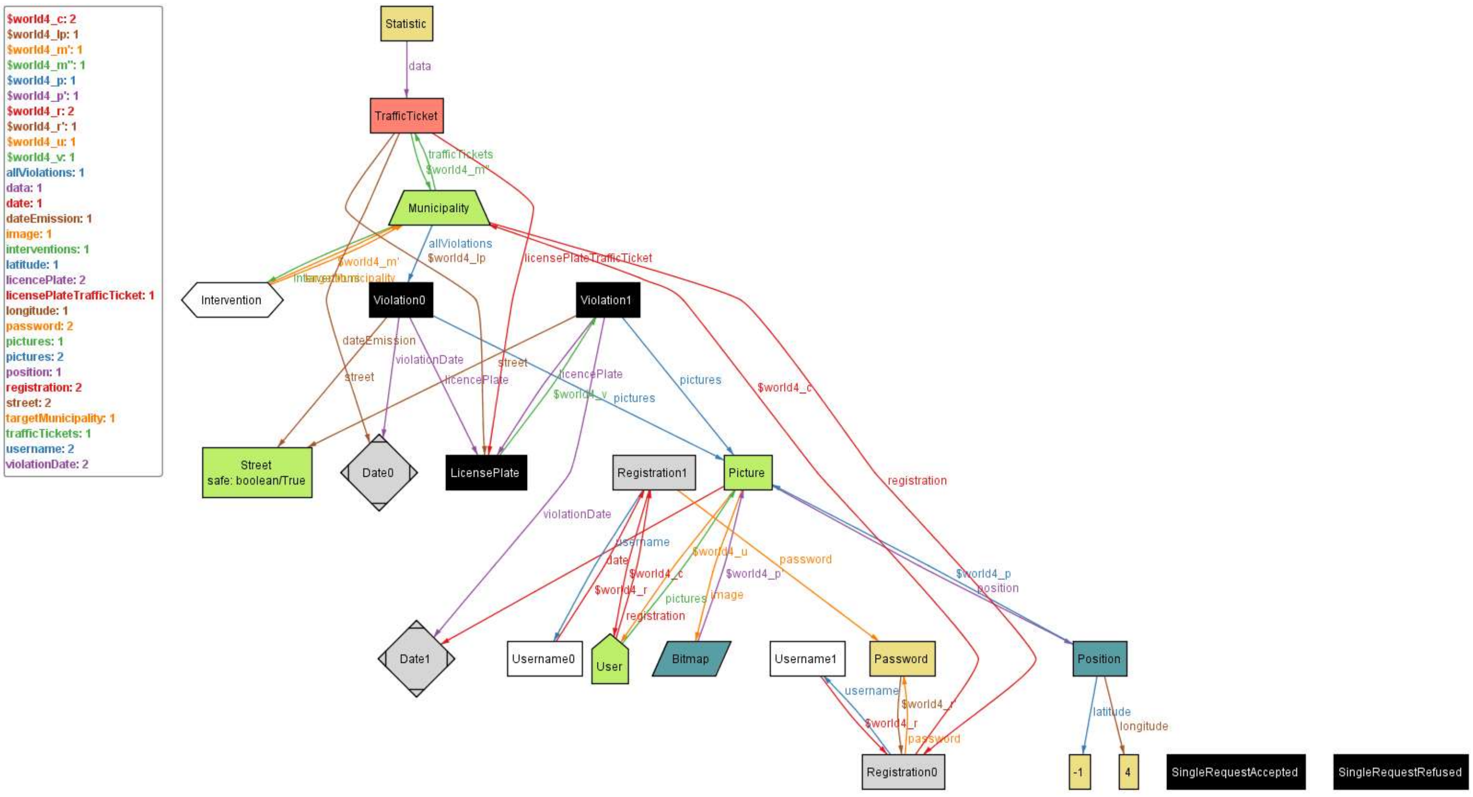


Figure 22. *World 4 generated by Alloy Analyzer*

Lastly, for the world 4, we can see that two violations can be made by only one license plate, and also since there are no accidents liked to a street, it is safe. The municipality, as well, generates traffic tickets and it is used for generate statistics in the system. Finally, the picture has a position connected to it, with values inside the normalized range defined that represents the real values.

5 Appendices

5.1 Used tools

The tools used for the development of this document were those ones listed below.

- Microsoft Office Word Professional Plus 2016
- draw.io
- GitHub
- Alloy Analyser 4.2

5.2 Hours of effort spent

The hours spent by the group are listed below, differentiating for each participant.

Task	**Hours spent**		
	Aida Gasanova	**Alexandre Batistella Bellas**	**Ekaterina Efremova**
Introduction	4	4	1.5
Product perspective	5	4	0.5
Product functions	0.5	3	0.5
Domain assumptions	1	1	3
External interface requirements	6	1	6
Functional requirements	4	14	10
Non-functional requirements	3	2	4
Formal analysis using Alloy	3	10	5