POLITECNICO
MILANO 1863

Software Engineering 2
A.Y. 2019/2020

SafeStreets

Requirement Analysis and Specification Document

Alessandro Fulgini — alessandro.fulgini@mail.polimi.it
Federico Di Cesare — federico.dicesare@mail.polimi.it

Version 1.0
November 10, 2019

Deliverable:	RASD
Title:	Requirement analysis and specification document
Authors:	Alessandro Fulgini, Federico di Cesare
Version:	1.0
Date:	November 10, 2019
GitHub Repository:	github.com/fuljo/FulginiDiCesare-SE2
Copyright:	Copyright © 2019 Alessandro Fulgini, Federico di Cesare — All rights reserved

Acknowledgments
Typeset with XeTeX
Sans-serif font: Titillium by Accademia di Belle Arti di Urbino
Monospace font: Consolas
Alloy syntax highlighting: alloy-latex-highlighting
UML diagrams generated with PlantUML
Mockups designed with Adobe XD
Mockup graphics:

- Montserrat font by Julieta Ulanovsky
- Mobile app icon collection by Jessica Gueller (released under CC BY 3.0)
- Isometric car parking design by macrovector_official / Freepik

Contents

1 Introduction

1.1 Purpose

1.1.1 General purpose

The core idea behind SafeStreets is that in order to enforce the traffic laws, the help of everyone is necessary, including common citizens. SafeStreets is a system that allows citizens to report infractions to the authorities, while staying anonymous.

The user experience will be as smooth as possible, a person that spots a traffic law violation, can open the app, take a picture and with a few *taps* send it to the authorities of its municipality. The system will try to automatically collect all the data required to complete a violation report: location, date, time and licence plate number. In case something doesn't work, the user will be asked to fill in the incomplete data. When sending the violation report, there is no need to address a specific authority, the system will route the report to the correct one automatically.

From the side of the authorities, the system will show all the violation reports occurred in the municipalities under that specific authority control.

Both citizens and operators can obtain some statistics from the system. These statistics will show how the different violations are distributed in space, time, and type. It's worth mentioning that citizens won't of course see the specific vehicle statistics, for privacy reasons.

In case a municipality offers data about accidents, SafeStreets will use the *Smart-Suggestions* technology to cross this data with the violation reports and automatically suggest the authority some *actions* that can be carried out to prevent the same accidents/violations from occurring again.

1.1.2 Goals

G1 The system must allow users to notify authorities of parking violations, by uploading pictures of the violation together with metadata (date, time, position, infraction type, licence plate of the vehicle)

G2 The system must be able to extract the licence plate number from the photos uploaded by the users

G3 The system must persistently store the violations uploaded by the users, while keeping the uploader anonymous

G4 The authorithy must be able to visualize, accept and refuse the single violation reports that occurred in its competence area

G5 The authorithy must be able to visualize violation statistics based on their location, time, type or responsible vehicle

G6 The user must be able to visualize anonymized violation statistics only based on their location, time and type

G7 The system must be able to determine unsafe areas by matching data about violations and data about accidents. It must then use such data to suggest actions that the authorithy can take to help preventing accidents

The word *must* in the description of this goals refers to a mandatory objective that the system will be able to accomplish, in the scope defined by the domain assumptions. However it is worth to point out that specifically for G2 and G7, the ability to accomplish the objective may vary depending on the quality of the input data.

1.2 Scope

SafeStreets is a service that can be offered and managed by authorities to willing citizens who want to contribute to their city's public order. In this way the authorities can improve their control over traffic violations.

The service is supposed to be provided in Italy, so we assume that each municipality has one reference authority (e.g. police department) that is responsible for handling traffic violations in that area. The authority may activate the service or not in its area.

First of all, the user must register to the service by providing his/her fiscal code, a username and a password of his choice. By requiring the fiscal code to be unique for each user, we can avoid duplicate accounts for the same citizen.

When the user detects a violation, he can open the software on his mobile device and take a picture. The system will elaborate the picture in order to auto-detect the licence plate number. Furthermore, the app will get from the device the date, time and GPS position. The auto-collected data will be shown to the user who can correct it before being sent to the authority. The only data the user must manually insert is the infraction type, which will be chosen from a finite list of possible types. This process will be designed to be as smooth as possible, to offer a clean user experience.

From the authority point of view, the violation reports are handled by an operator, who is identified by an username (must be unique) and a password. The operator will review the violation and then decide whether to accept it. He may discard the violation when it contains erroneous or incomplete data. For example the photo may simply not show a violation, or it may happen that the licence plate in the picture is partially unreadable and the number inserted by the user is likely to be wrong.

After accepting the violation, the authority might schedule intervention on the ground (e.g. remove the vehicle from the road). Scheduling the intervention, however, is out of the scope of the software to be.

The data collected is of course persistent as the S2B will offer various statistics to citizens and authorities. The authorities will be able to view statistics on the number and

Table 1.1: Table of phenomena. Phenomena controlled by the world are marked with *W*, while the ones controlled by the machine are marked with *M*

Phenomenon	Controlled by	Shared
A user registers to the service	W	✓
A vehicle commits a parking violation	W	✗
A user takes a photo	W	✓
A user reports a violation	W	✓
An operator checks the reported violations	W	✓
An operator accepts/refuses a violation report	W	✓
The Police emits a fine for a violation	W	✗
The system computes statistics based on the stored violations	M	✗
A user/operator views the statistics	W	✓
The system retrieves data about accidents	M	✗
The system computes suggested actions	M	✗
An operator checks for new suggested actions	W	✓
An operator marks a suggested action as *carried out*	W	✓
The system determines whether a carried out action has been useful or not	M	✗

types of violations. The statistics can be filtered by violation type, location, date/time and vehicle. Citizens will also be able to view statistics, but in order to keep privacy they won't be able to see the vehicles involved.

For the data about accidents, we assume that a municipality may provide it or not. Therefore the function which suggest actions to prevent accidents can only be offered in municipalities that provide such data. The data is supposed to contain the location, timestamp and accident type for each accident. We also assume that there are a finite pre-defined set of accident types.

The various phenomena that are relevant or partially relevant for the system, are summarized in table 1.1.

1.3 Definitions, acronyms, abbreviations

Definitions

User a citizen, identified by his fiscal code, who is able to report violations to authorities and view anonymized statistics. In order to access the functionalities the user must first register with an username and a password.

Authority the public organization which enforces traffic laws in one or more municipalities. The authority is said to be *competent* for those municipalities. The authority is the recipient of the violation reports that occurred in its *competence area.*

Violation a *traffic law infraction*, in particular regarding parkin. In the context of the

S2B it refers to the complete set of data sent by users to authorities: photo, date, time, type and licence plate number of the involved vehicle. Also referred to as *infraction*.

Operator an authority employee that reviews the violations sent and decides whether to keep them.

Acronyms

API	Application Program Interface
UI	User Interface
GPS	Global Positioning System
S2B	Software To Be
OCR	Optical Character Recognition
DMV	Department of Motor Vehicles

Abbreviations

Gn	n^{th} goal
Dn	n^{th} domain assumption
Rn	n^{th} requirement

1.4 Revision history

Version 1.0 (10/11/2019) Initial version

1.5 Reference documents

[2] Elisabetta Di Nitto and Matteo Rossi. *Mandatory Project: goal, schedule and rules.* 2019.

[5] "ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering". In: *ISO/IEC/IEEE 29148:2018(E)* (Nov. 2018). DOI: 10.1109/IEEESTD.2018.8559686.

[6] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis.* MIT Press, 2016, pp. 259–294. ISBN: 9780262528900. URL: https://mitpress.mit.edu/books/software-abstractions-revised-edition.

1.6 Document structure

The present document is divided into chapters. Here we give a brief description of each chapter and its intended audience.

Chapter 1 outlines the general purpose of the project, including the goals it has to reach. It illustrates the scope of the applications and provides some definitions that will be used throughout the document. It also contains the document revision history and the list of the specification documents that describe the project and the required analysis approach. It is intended for a *general audience*, as the description is provided at a high and non-technical level.

Chapter 2 provides a description of the intended product. Here we illustrate the domain model by showing the main entities involved, their relations and life cycle. Then we illustrate the functionalities, the types of users and finally we set the boundaries of the domain. This chapter is addressed to the *clients* for which the system is being built for, in order to check that the right functionalities and domain assumptions have been captured, but also to *software engineers* and *developers* that work on this project, so they can get a high-level description.

Chapter 3 provides a detailed description of the requirements. Here we define the various interface requirements, the functional requirements (also illustrated with the help of use cases) and finally the non-functional requirements (performance, reliability, security and other constraints). The intended audience of this chapter are the *designers*, *developers* and *testers* of the project. When possible, we inserted some hints to achieve the desired requirements, which might become handy during the design phase.

Chapter 4 provides a formal analysis of the domain model using the *Alloy* language. In particular it is aimed to show the consistency constraints of the data model. For this reason it is particularly addressed to the *developers* of the data model and to the *testers*, that will need to verify that the implemented product is consistent with the formal representation.

2 Overall Description

2.1 Product perspective

The basic idea upon which the system will be built is that the user is a volunteer, so it has to put as little effort as possible to send information about violations.

Right after the user enters the application, he can take a picture directly from there. Then the taken picture is scanned by an OCR algorithm that detects (or at least tries to) the biggest licence plate and obtains its number. The application proceeds to collect as much data as possible from the built-in functions of the host device: gets the time, date and retrieves location through GPS. The auto-collected data is then showed to the user who can correct or complete it. Finally the user inserts the infraction type and the violation is ready to be sent to the authorities. When sending the violation to the authority the system doesn't save the name of the user on the violation itself, in order for it to remain anonymous.

In figure 2.1 we represent the main entities of interest for the system. A *Violation* is reported by exactly one *User*. It is associated to exactly one *Picture*, *Location* and *Vehicle* that committed it; these have no reason to be considered by the system if they are not associated to at least one violation.

Accidents are also associated to their *Location*, so the system can match them with *Violations* to provide *SuggestedActions* in order order to reduce accidents.

A *Location* where a violation or accident took place belongs to exactly one municipality. In the real world, each municipality has an authority which is responsible for it, however we used the 0..1 cardinality to represent the fact that the authority might not participate to SafeStreets; in this case, it makes no sense for users to report violations in that area.

Each authority might have multiple operators which are allowed to handle the violations reported in its competence area.

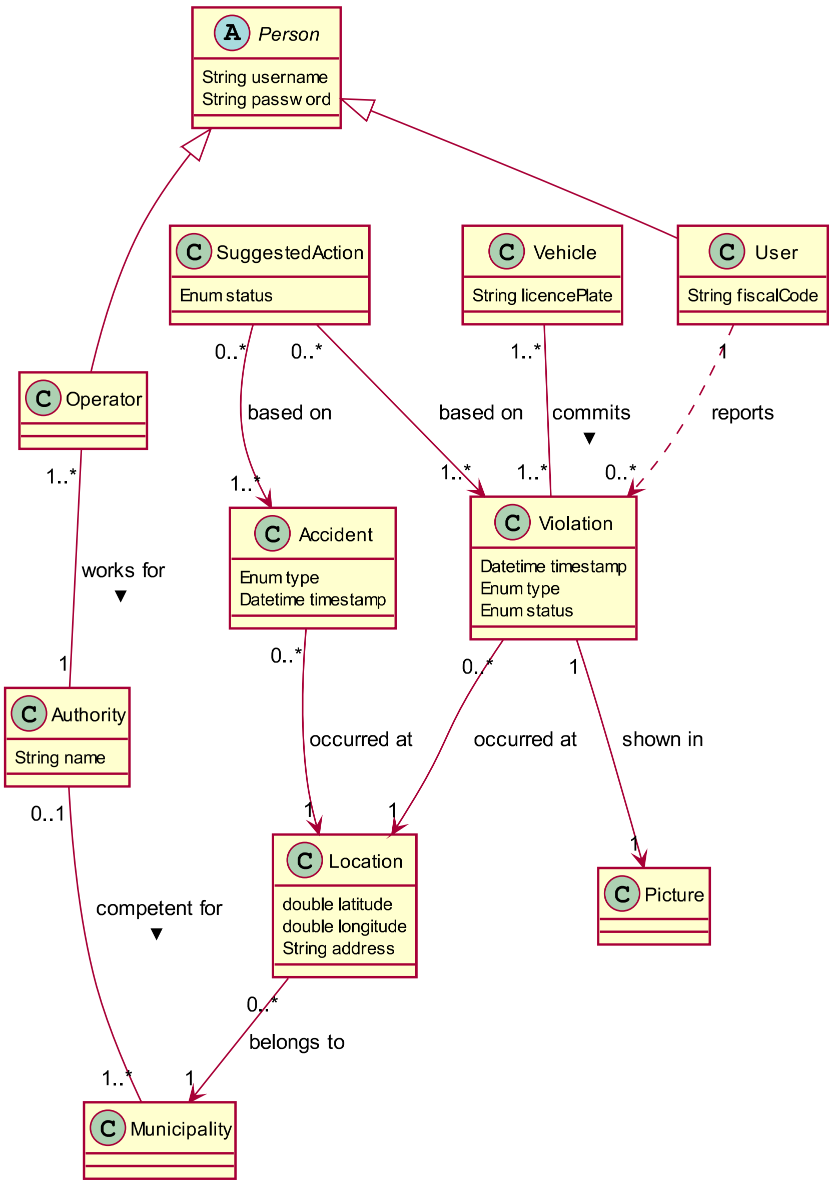


Figure 2.1: Class diagram

Figure 2.2 shows the evolution of a Violation, from when the user takes the picture, to the final review from the authority operator.

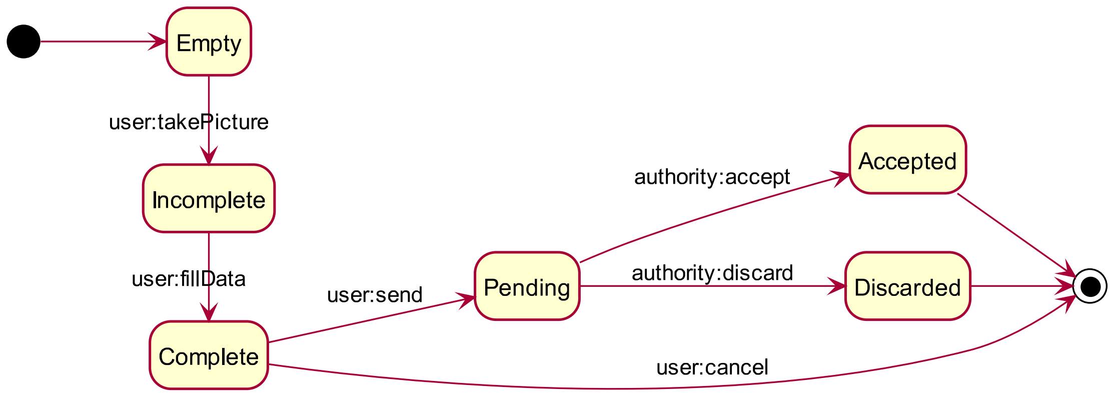


Figure 2.2: Violation state diagram

Figure 2.3 shows the various states of a SuggestedAction generated by *SmartSuggestions*. Whenever an authority actually implements an action, an operator can mark the corresponding SuggestedAction as *carried out*. The system will then monitor the new violation reports and related accidents in order to understand if that specific SuggestedAction has been useful or not.

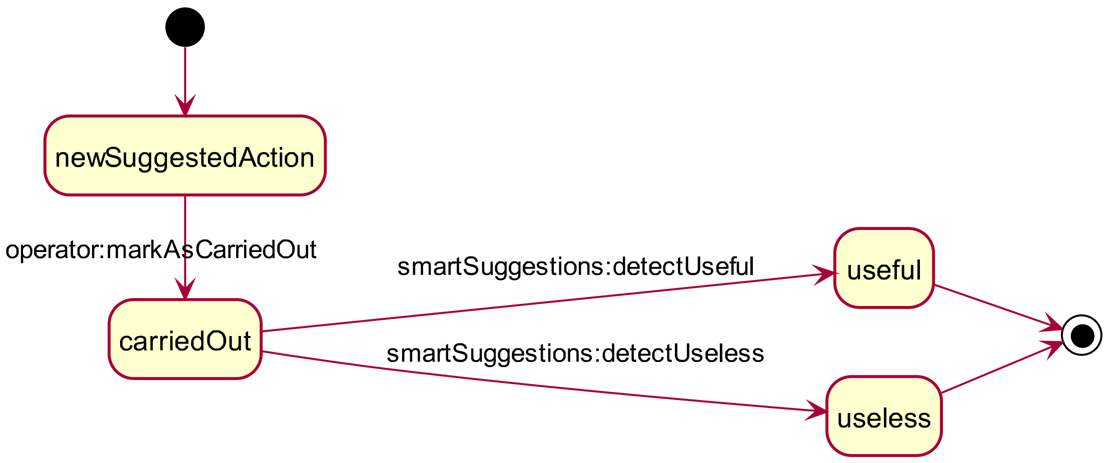

newSuggestedAction
operator:markAsCarriedOut
carriedOut
smartSuggestions:detectUseful
useful
smartSuggestions:detectUseless
useless

Figure 2.3: Suggested Action state diagram

2.2 Product functions

In this section the major functions of the S2B will be summarized.

Report a Violation
This is the core function of the system. When a user spots a violation, it can open the app, log-in using username and password and rapidly take a picture. The system will now run an OCR algorithm to find the biggest licence plate on the picture and extract

the number. Other information are required to complete the violation report: date, time and position. They can be collected through the standard tools of the mobile host device. If one or more of these information are wrong or incomplete, then the user can correct the wrong ones. Finally the user must select the right Violation Type. When a Violation report has been completed, the user can choose if he wants to send it or to discard it.

Violations are automatically routed to the correct authority based on the municipality they occurred in.

An authority operator can see all the violations the authority is responsible for and review them. In particular he can accept the violation, or in case it is considered inappropriate or simply erroneous, discard it.

View Statistics

Both users and operators can ask the system for statistics about the number and type of violations. Authorities and users can see different statistics, for privacy reasons a common user cannot see data about the specific vehicles. Some examples of available statistics are:

- Number of violations of a certain type
- Number of violations divided by area
- Number of violations divided by time of the day
- Number of violations divided by vehicle (*authority only*)

Crossed data statistics are also available (e.g. most popular violation type for each area).

SmartSuggestions

If the municipality allows SafeStreets to collect their data about accidents, the S2B will cross this data with the violations occurred in that area. The system will try to determine a *cause-effect* relation between violations and accidents and finally suggest a smart suggestion to prevent that accident from occurring again.

New suggestions will be displayed to the operator as they become available. Furthermore the operator can mark the suggestions as carried out. Since then the system will monitor the number of accidents of that type in that area to determine if the suggestion was effective or not, and possibly provide better suggestions in the future.

2.3 User characteristics

Here we describe the various users of the applications, distinguishing them by their role and the functions they have access to.

User a citizen, uniquely identified by a fiscal code, who registers to SafeStreets by providing username and password. After logging in he will be able to report violations to authorities and view anonymized statistics about violations. In order for him to be able to use the service he will need a mobile device (i.e. smartphone) with a camera and GPS.

Authority a recognized government organization that enforces traffic laws in a certain area. The authority can ask to be registered to SafeStreets by certifying its status and providing the list of municipalities under its jurisdiction. The authority will receive a set of username/password pairs which will be given to its operators to enable them to access the system. If the municipalities in which the authority operates also provide data about accidents, the authority can request to also activate the *SmartSuggestions* function by providing access to this data.

Operator an employee of the authority, to which the authority provides an username/-password pair to access the system. The *basic service* of SafeStreets enables the operator to manage violations reported in the *competence area* of his organization; this includes: viewing, accepting and discarding violations, viewing detailed statistics about the violations (including the involved vehicles). If the *SmartSuggestions* feature is enabled for that authority he will also be able to view suggestions generated by the system and manage them. Both functionalities can be accessed from a web browser.

2.4 Assumptions, dependencies and constraints

Here we present the domain assumptions that must hold in order for the system to work properly.

D1 A citizen can be uniquely identified by his fiscal code

D2 The users' devices provide date and time with a precision of 1 minute or less, with resepect to UTC

D3 The users' devices provide GPS location with a precison of 10 meters or less

D4 Any vehicle that may commit a violation can be uniquely identified by its licence plate number

D5 Each location where the service can be used is associated to exactly one municipality

D6 Each municipality has exactly one authorithy that enforces traffic rules in its territory

D7 Authorities have demonstrated that the municipalities for which they requested the violations are under their jurisdiction

D8 The internet connection works without problems

D9 The system can access the database of registered licence plates

D10 The data about an accidents contains its location, timestamp and type (within a finite list of types)

Hardware requirements

In order to use the service, the user will need a smartphone with a camera, a GPS sensor and internet connectivity. Both Wi-Fi and data connections are fine as long as they provide a reasonable bandwidth to upload and download data.

The operators of the authorities can access their dedicated functions through a web browser. In this way the service can be accessed both from desktop computers at the authorities' headquarters and from the mobile devices of the on-field agents. In both cases a stable internet connection is required.

External data sources

SafeStreets needs to access the DMV registry to check the validity of a licence plate. We assume the existence of a public REST API that, given a licence plate number, tells whether it is registered to the DMV or not.

In case the *SmartSuggestions* function is to be activated, the authority must also provide access to the data about accidents in the municipality(ies) under its jurisdiction. The assumption is that the data can be retrieved in a structured or semi-structured format (e.g. JSON, XML) through a REST API. The API must allow to retrieve the data about the accidents of a single municipality, indicating the location (e.g. latitude, longitude, address), type of each accident (within a finite set of pre-defined types) and the timestamp of when it took place. Furthermore it must allow to retrieve the violations that occurred after a certain date, in order for SafeStreets to perform incremental updates.

Privacy constraints

According to Article 4 of the General Data Protection Regulation [4] both the fiscal code and the licence plate number are considered *personal data* as they can be directly or indirectly associated to a person. Therefore our system must comply to the GDPR, and in particular to the national law DPR 15/2018 [3] that regulates the treatment of personal data by police forces.

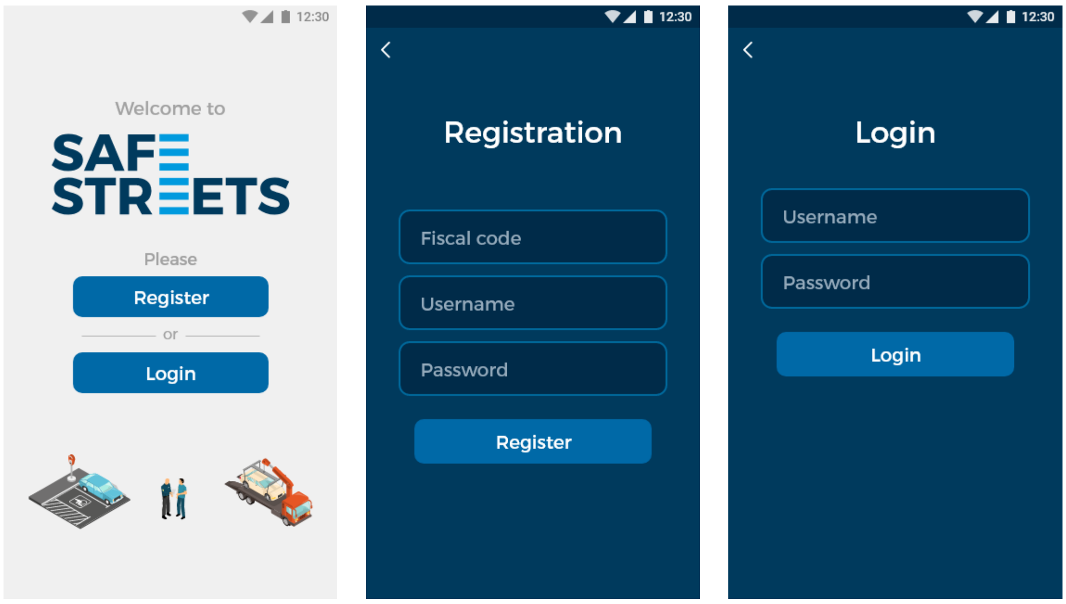
3 Specific requirements

3.1 External interface requirements

3.1.1 User interfaces

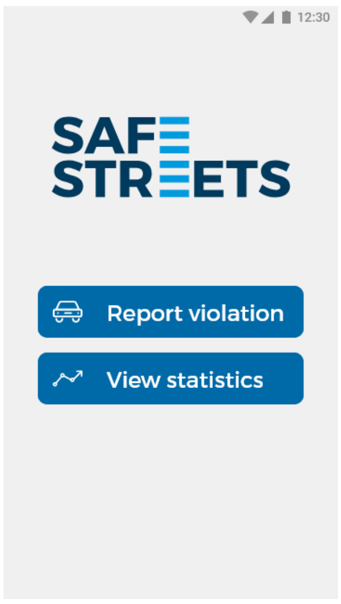
Here we show some mockups of the user interface. These screens are aimed at giving an overview of how the interface for the main functions should look like. However the final implementation may differ from this.

As stated earlier in section 2.4 on page 14 the user (i.e. citizen) will access the service through a mobile application. The mockups of the app screens are shown in figure 3.1. The operators, on the other hand, will access the service through a web portal, whose pages are represented in figure 3.2 on page 18. Note that the suggestion function will only be displayed if *SmartSuggestions* is active for that particular authority.

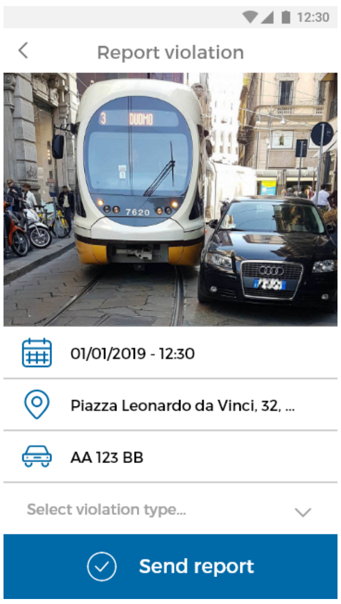


(a) Landing page **(b)** Registration page **(c)** Login page

Figure 3.1: Mockups of the mobile interface, used by the citizens



(d) Main menu

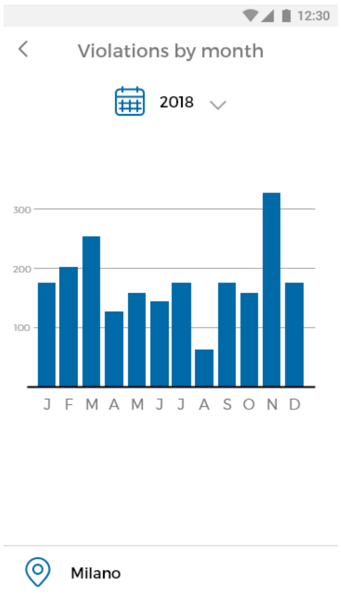


01/01/2019 - 12:30

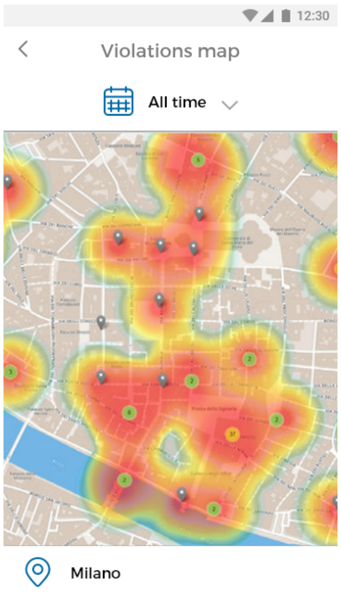
Piazza Leonardo da Vinci, 32, ...

AA 123 BB

(e) Violation reporting page

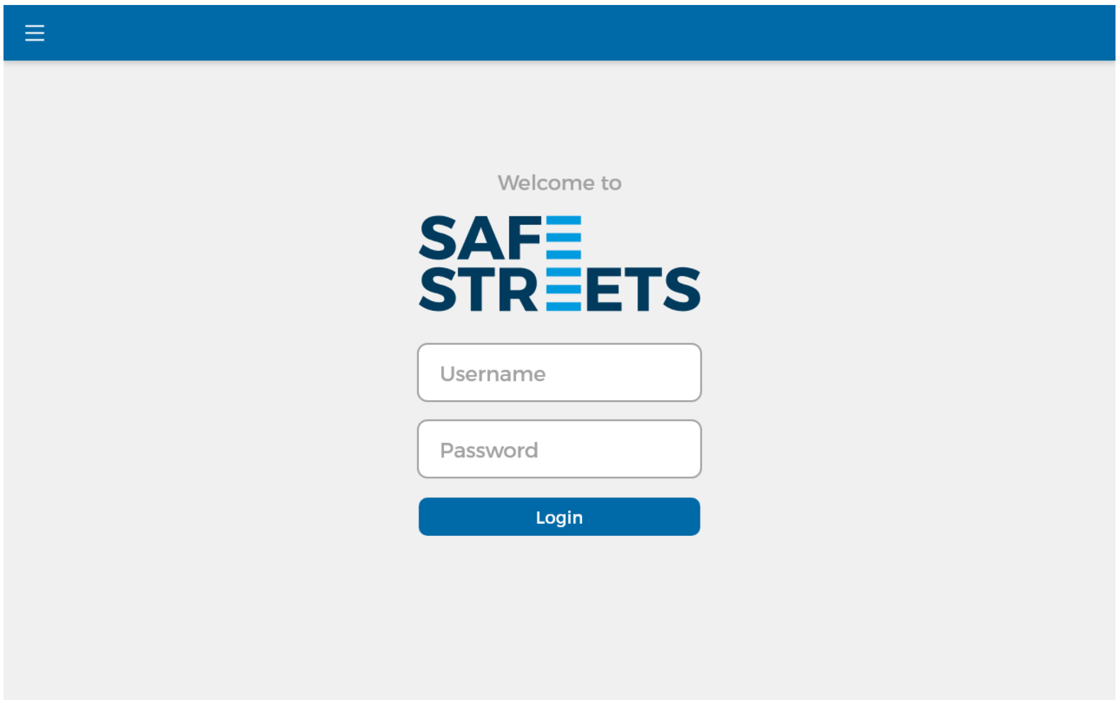


(f) Statistics page: histogram

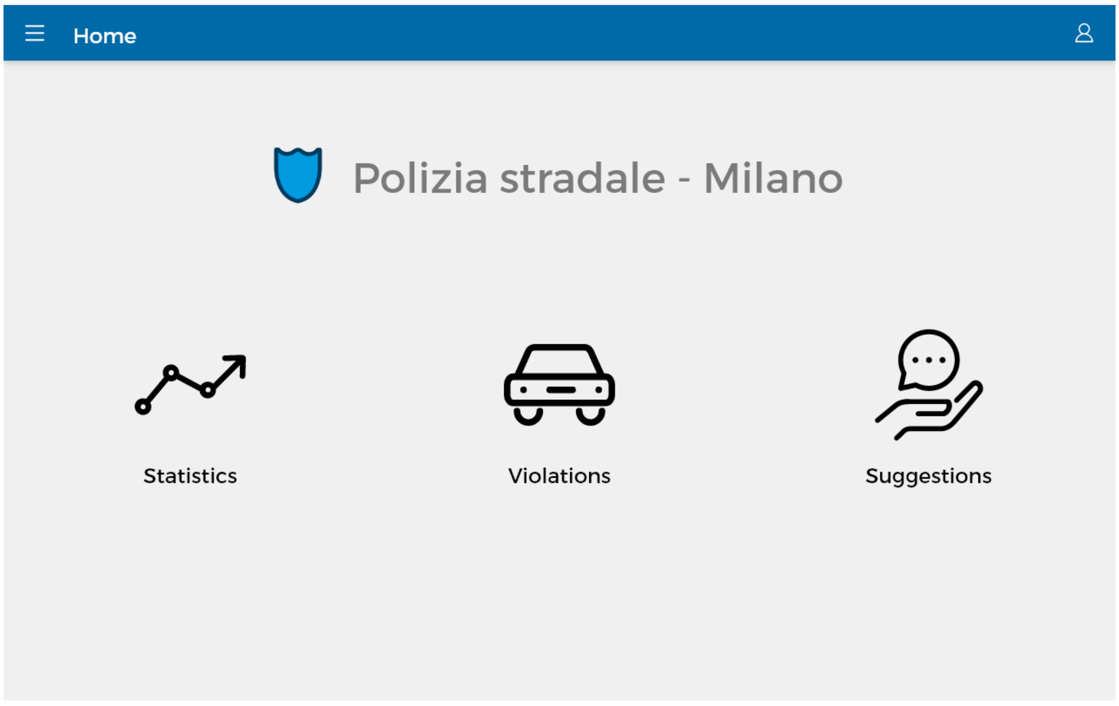


(g) Statistics page: heat map

Figure 3.1: Mockups of the mobile interface, used by the citizens

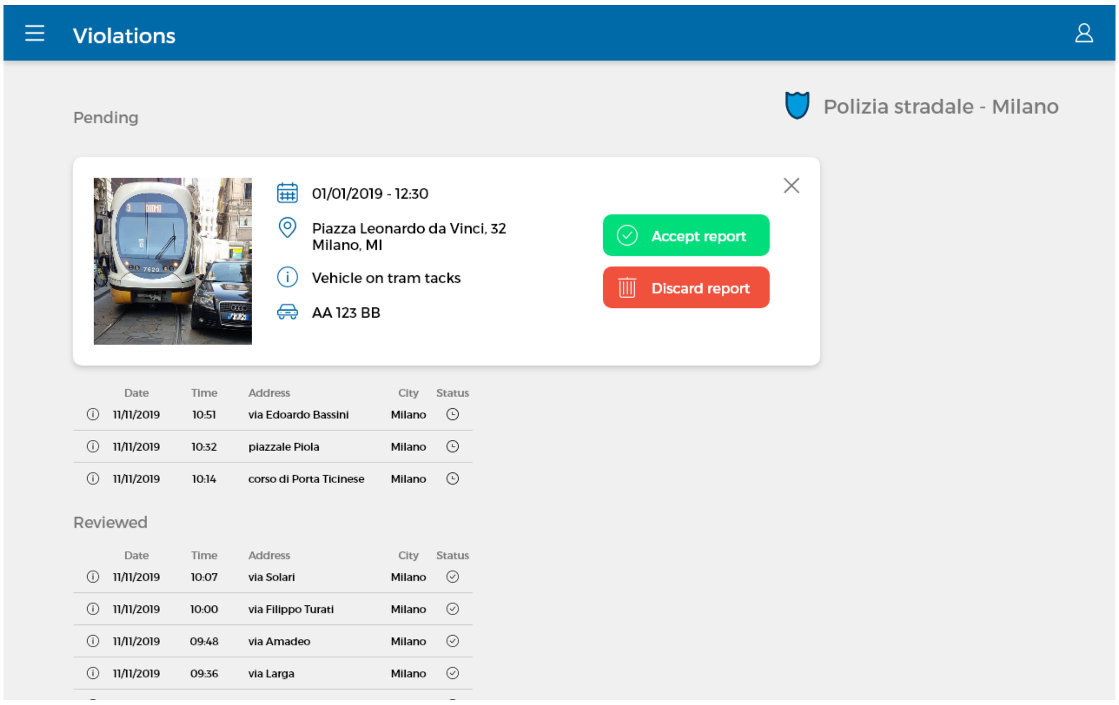


(a) Landing page



(b) Main menu

Figure 3.2: Mockups of the web interface, used by the authority operators

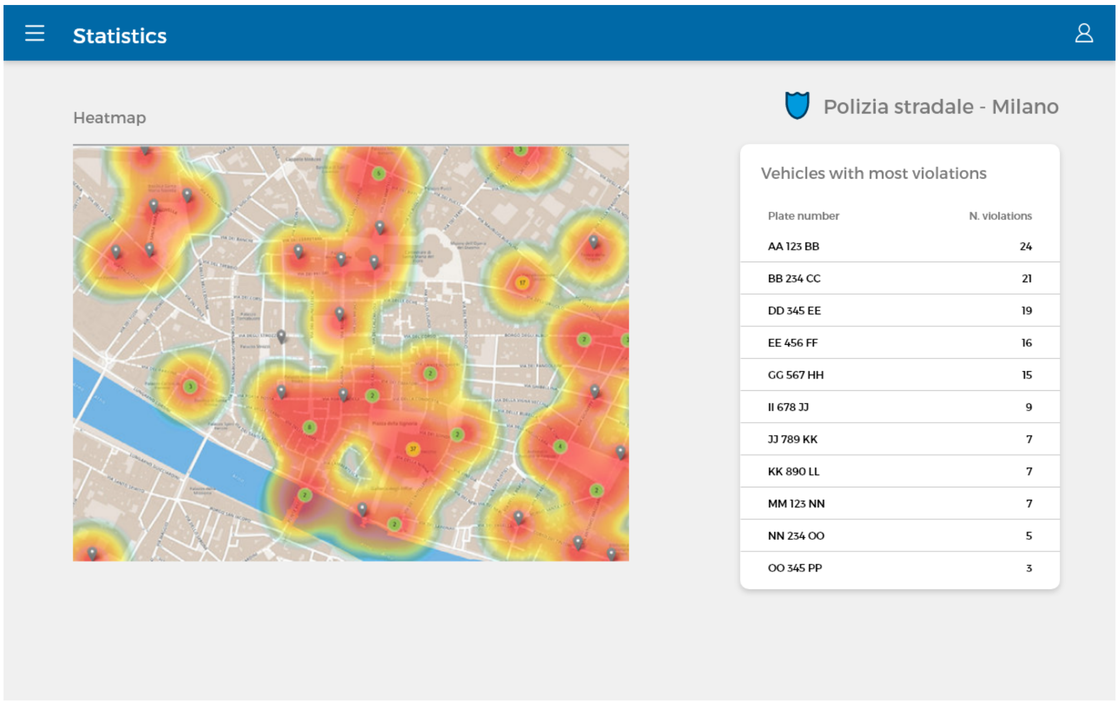


01/01/2019 - 12:30
Piazza Leonardo da Vinci, 32
Milano, MI
Vehicle on tram tacks
AA 123 BB

	Date	Time	Address	City	Status
ⓘ	11/11/2019	10:51	via Edoardo Bassini	Milano	🕒
ⓘ	11/11/2019	10:32	piazzale Piola	Milano	🕒
ⓘ	11/11/2019	10:14	corso di Porta Ticinese	Milano	🕒

	Date	Time	Address	City	Status
ⓘ	11/11/2019	10:07	via Solari	Milano	✓
ⓘ	11/11/2019	10:00	via Filippo Turati	Milano	✓
ⓘ	11/11/2019	09:48	via Amadeo	Milano	✓
ⓘ	11/11/2019	09:36	via Larga	Milano	✓

(c) Violations list

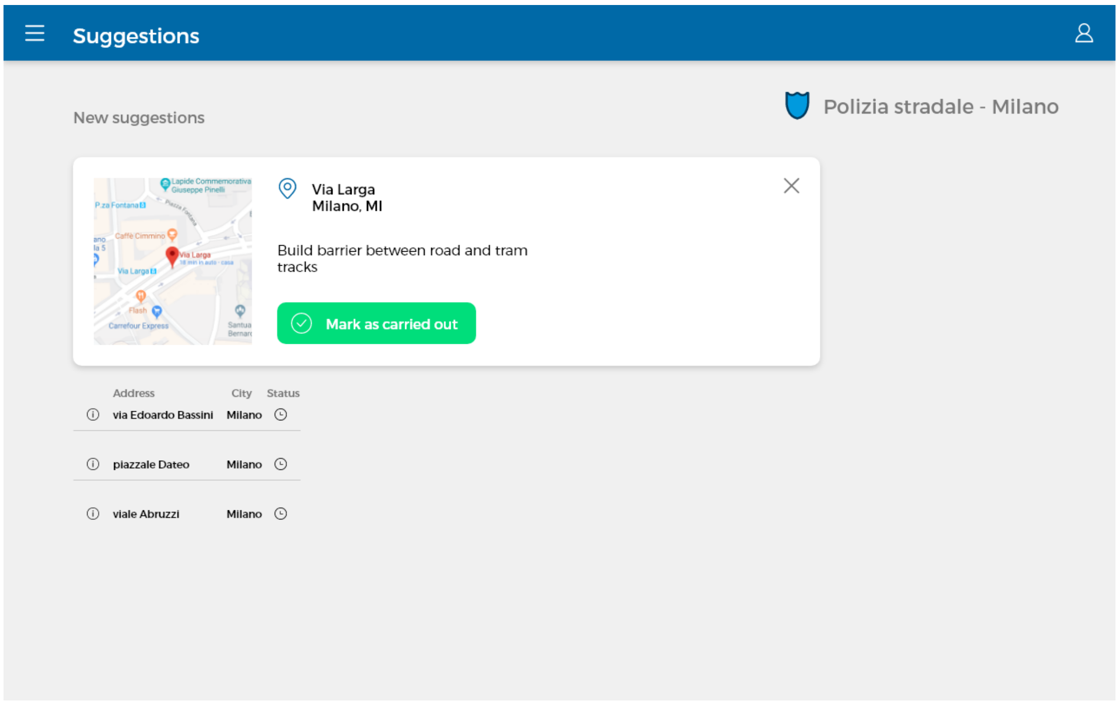


Vehicles with most violations

Plate number	N. violations
AA 123 BB	24
BB 234 CC	21
DD 345 EE	19
EE 456 FF	16
GG 567 HH	15
II 678 JJ	9
JJ 789 KK	7
KK 890 LL	7
MM 123 NN	7
NN 234 OO	5
OO 345 PP	3

(d) Statistics page

Figure 3.2: Mockups of the web interface, used by the authority operators



Via Larga
Milano, MI

Build barrier between road and tram tracks

Address	City	Status
via Edoardo Bassini	Milano	
piazzale Dateo	Milano	
viale Abruzzi	Milano	

(e) Suggestions page

Figure 3.2: Mockups of the web interface, used by the authority operators

3.1.2 Hardware interfaces

The system doesn't provide any hardware interfaces.

3.1.3 Software interfaces

The S2B doesn't provide any external API: the service is accessible only from the user interface. However, the implementation will most likely use an internal API to allow interaction between components. In case the authority will need to integrate the S2B with other systems, some time in the future, this internal API might be standardized and made available to the authority.

3.1.4 Communication interfaces

The S2B uses the internet as the standard communication facility. No other communication interfaces are planned.

3.2 Functional requirements

In this section we start by analyzing the *scenarios* and defining the *use cases* (and related sequence diagrams) for each user class (citizens and authorities). Then we programmatically show the requirements that the S2B must satisfy in order to reach the goals.

3.2.1 User

Scenarios

Scenario 1.1 Davide is a Professor Assistant at his University and lives five minutes away from his office. He is a very healthy person, everyday he goes to his office by bike, and most of the days he finds a car parked on the bicycle lot, so he has to leave his bike in an unsafe place, where it can be stolen. Davide is sick of this situation and wants to to something about it. He knows that the Municipality of Milan recently deployed a new service: *Safe Streets*, that can be used to report situations like this. He downloads the App, registers, logs in and then takes a picture of the car illegally parked on the bike lot. The morning after Davide goes to his office as always by bike and to his surprise the bike lot was free, so the new *Safe Streets* works for real!

Scenario 1.2 Carmelo is a man who lives in a small town in Sicily, but he's currently planning to move to Catania for work reasons. In his hometown Carmelo has to live with a very annoying problem: his neighbor often parks his car in front of Carmelo's garage. So when Carmelo needs to use the car he usually finds the exit blocked. While looking for a new apartment in Catania, Carmelo finds out that the Police has activated a new service called *SafeStreets* that enables him to see statistics of the reported parking violations. Since he plans to use the car almost everyday to go to work, he doesn't want to have the same problem he had in his hometown. So he decides to subscribe to the service and looks for statistics about the *parking in front of driveway* violations. In this way he can identify the areas of the city where he's most unlikely to find his garage entrance blocked by another car and look for apartments there.

Scenario 1.3 It's 8:15 AM in Milan and the tram 9, packed with commuters, is moving in its reserved lane when it suddenly comes to an halt. The driver turns to the passengers and clearly states: "another guy has parked his car on the tram tracks, so we can't go on". The passengers become very angry, because they know that they'll be late at work. But luckily *Vincenzo Brambilla*, a true milanese who is always up to date with the latest technologies, exclaims "*ghe pensi mi*" (i'll handle this); Vincenzo takes out his smartphone, opens the SafeStreets app, takes a photo at the parked car, selects the *vehicle on tram tracks* violation type and sends the report to the local Police. Within minutes, a tow truck arrives and takes the car away from the tracks, so the tram can continue on its route.

Use cases

In figure 3.3 the various use cases in which the user participates are shown. After that we describe each use case in further detail and provide a sequence diagram which shows the various steps.

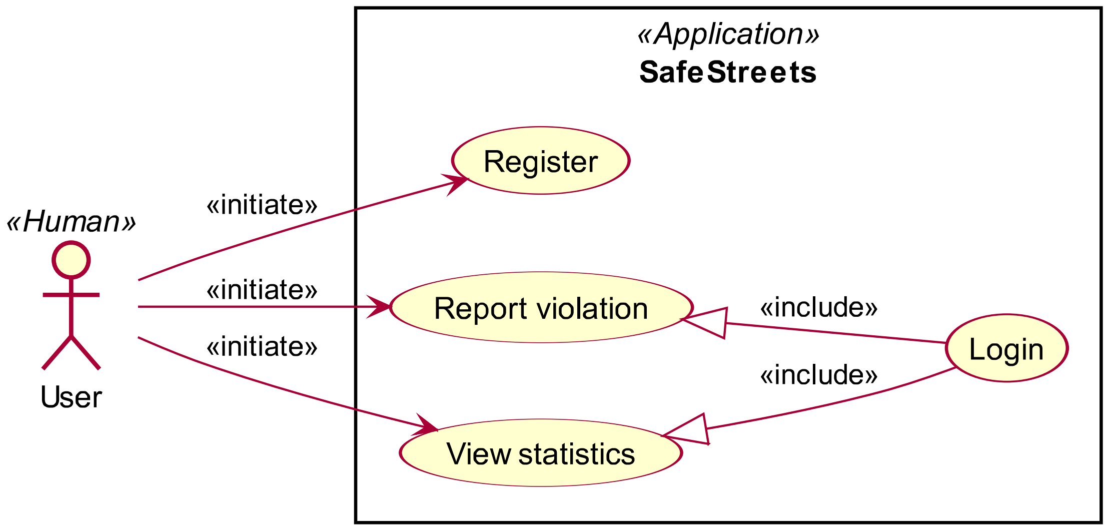


Figure 3.3: Use case diagram for the user

Use case 1.1

Name: Register
Actors: User
Entry conditions:

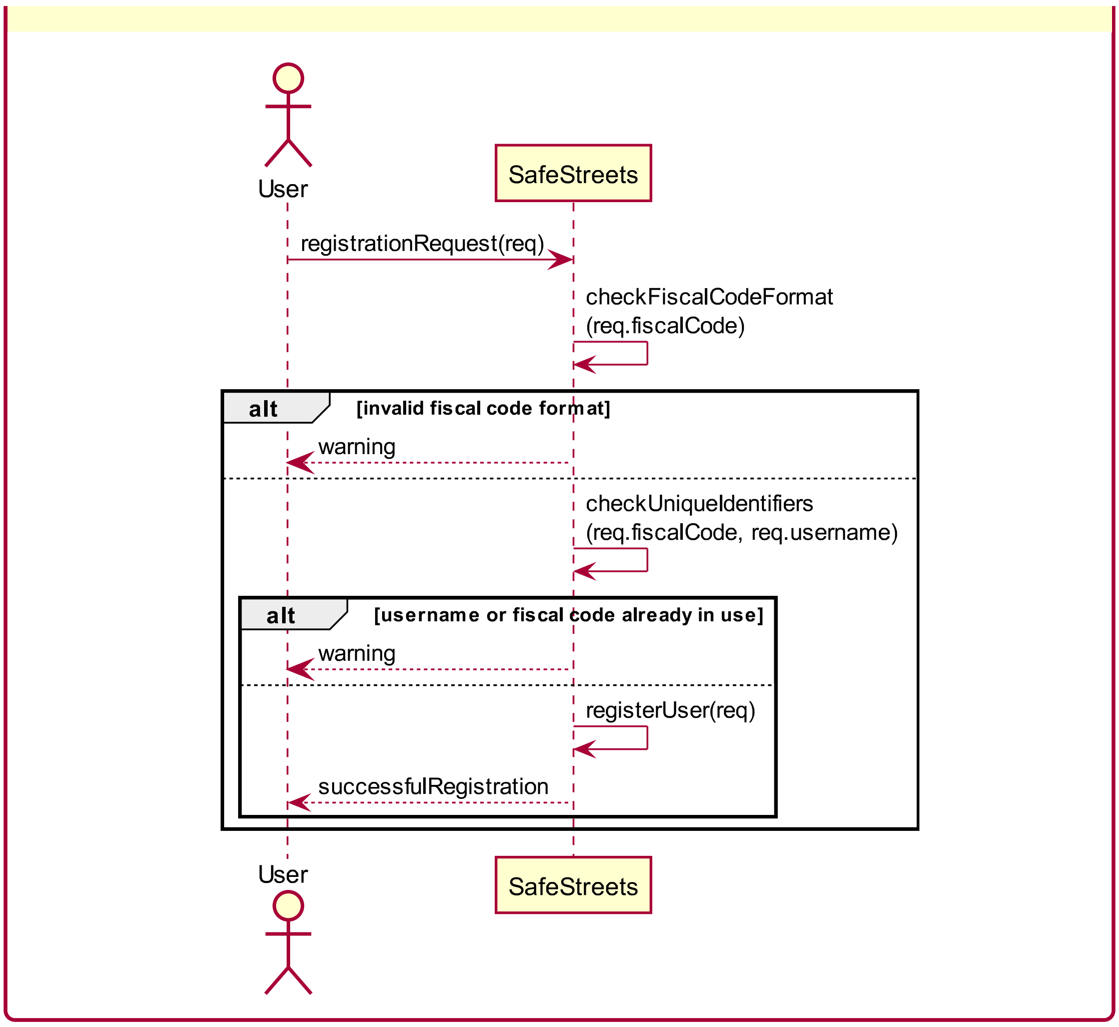
1. The user has opened the application on his device (he's in the *landing page*).
2. The user hasn't already logged in.
3. The device is connected to the internet.

Event flow:

1. The user taps on the *Register* button.
2. The user is presented with a form, in which he inserts his fiscal code, username and desired password.
3. The user taps on the *Register* button to request the registration.
4. The system receives the registration and stores the information entered by the user.
5. The user receives a confirmation in the app that the registration terminated successfully.

Exit conditions: The user is registered to the system and is now able to log in.
Exceptions:

1. **The fiscal code or the username already belongs to another user:** a warning is shown and the registration process is aborted.
2. **The user inserted an invalid fiscal code:** a warning is shown.
3. **The user didn't fill all the fields:** he's prompted to fill all the fields before tapping *Register* again.


User
SafeStreets
registrationRequest(req)
checkFiscalCodeFormat
(req.fiscalCode)
alt
[invalid fiscal code format]
warning
checkUniqueIdentifiers
(req.fiscalCode, req.username)
alt
[username or fiscal code already in use]
warning
registerUser(req)
successfulRegistration
User
SafeStreets

Use case 1.2

Name: Login
Actors: User
Entry conditions:

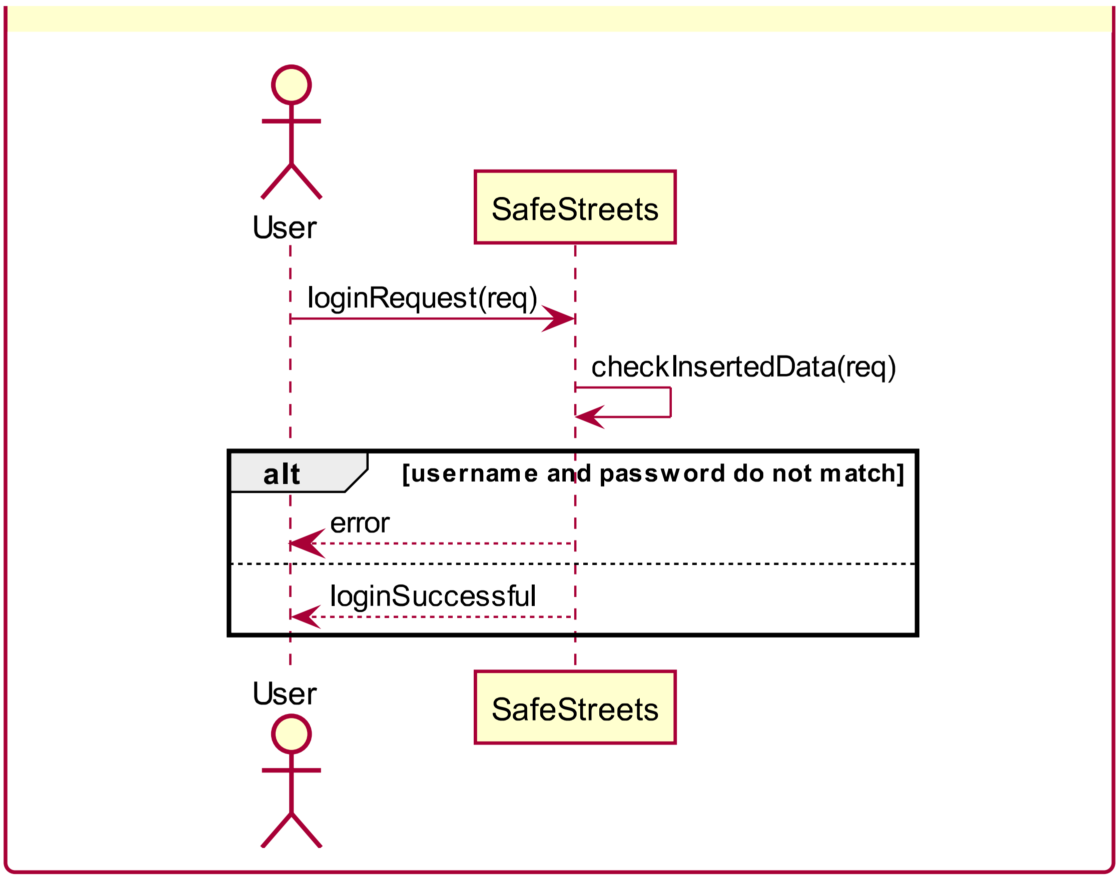
1. The user has opened the application on his device.
2. The user hasn't already logged in.
3. The device is connected to the internet.

Event flow:

1. The user taps on the *Login* button.
2. The user is presented with a form in which he inserts his username and password
3. The user taps on the *Login* button to send the login request.
4. The system receives the login request and checks the inserted data.
5. The user is now logged in and lands on the *Main Menu* page.

Exit conditions: The user is logged in and he's able to access all the application services.
Exceptions:

1. **The combination of username and password does not exist:** the user is shown an error pop-up and he's invited to insert the data again.
2. **The user didn't fill all the fields:** he's prompted to fill all the fields before tapping the *Login* button again


User
SafeStreets
loginRequest(req)
checkInsertedData(req)
alt
[username and password do not match]
error
loginSuccessful
User
SafeStreets

Use case 1.3

Name: Report Violation
Actors: User
Entry Conditions:

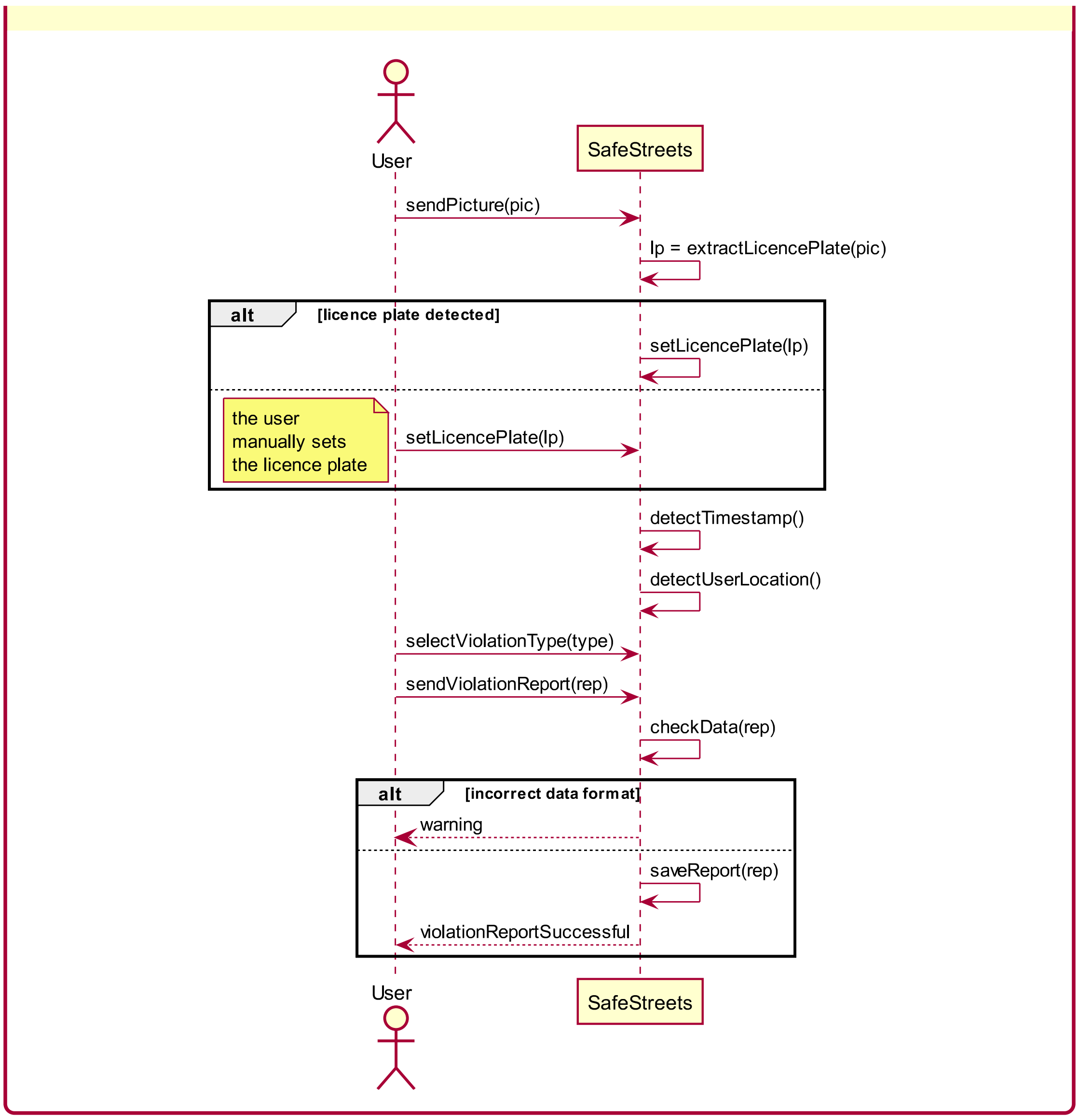
1. The user has opened the application on his device.
2. The user has successfully logged in.
3. The device is connected to the internet.

Event flow:

1. The user taps on the *Report violation* button.
2. The application accesses the device's camera.
3. The user takes a picture of the violation.
4. The system extracts the licence plate number from the picture.
5. The application automatically detects the current date, time and location of the device and fills the corresponding fields.
6. The user selects the type of violation from a finite list and can also correct other fields.
7. The user taps on the *Send report* button.
8. The system receives the Violation report, checks data correctness and shows the user a confirmation message.

Exit conditions: The user correctly sends the Violation report
Exceptions:

1. **The system hasn't been able to automatically detect the licence plate number from the photo:** the number will be manually inserted by the user. This doesn't prevent the procedure to complete.
2. **The user didn't fill all the fields:** he's prompted to fill all the fields before tapping *Send report* again.
3. **The inserted data are in an invalid format:** a warning is shown and the data must be filled again before sending the report.


User
SafeStreets
sendPicture(pic)
lp = extractLicencePlate(pic)
alt
[licence plate detected]
setLicencePlate(lp)
the user manually sets the licence plate
setLicencePlate(lp)
detectTimestamp()
detectUserLocation()
selectViolationType(type)
sendViolationReport(rep)
checkData(rep)
alt
[incorrect data format]
warning
saveReport(rep)
violationReportSuccessful
User
SafeStreets

Use case 1.4

Name: View Statistics
Actors: User
Entry Conditions:

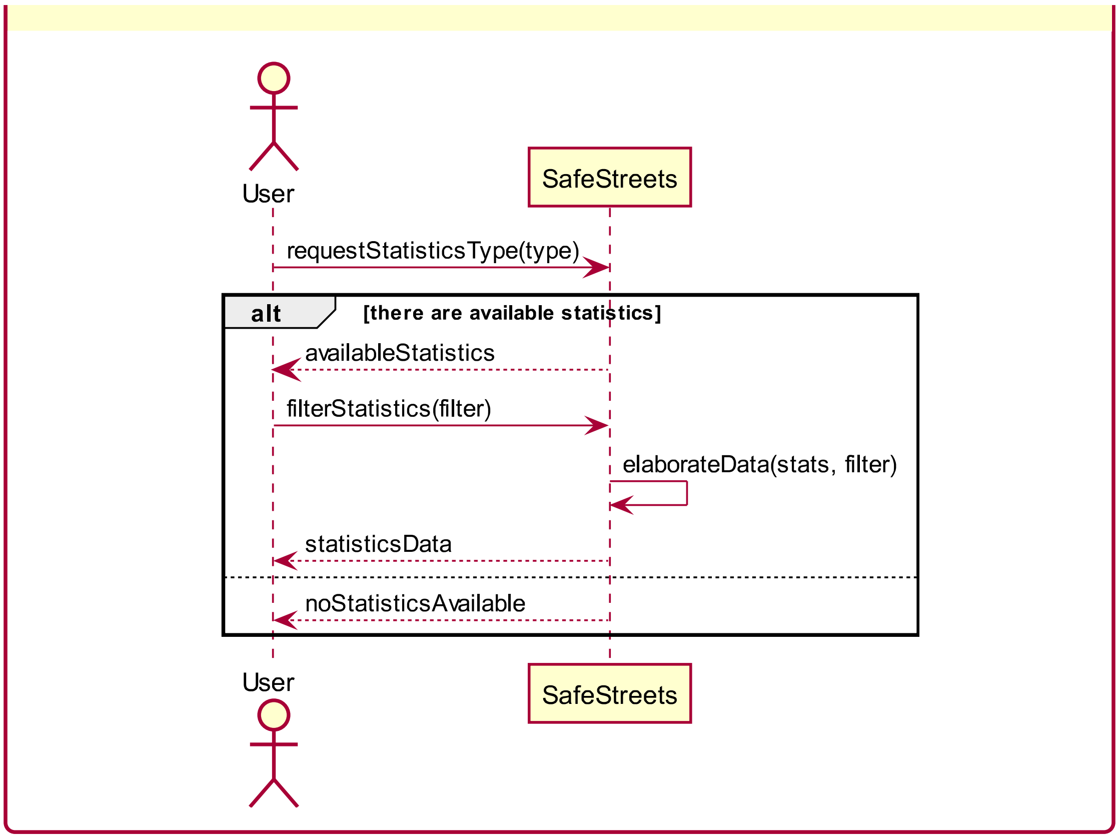
1. The user has opened the application on his device.
2. The user has successfully logged in.
3. The device is connected to the internet.

Event flow:

1. The user taps on the *View statistics* button.
2. The user is shown some options that describe the various statistics available.
3. The system receives the request and answers with the available statistics type.
4. The user selects one of the available statistics type.
5. The system receives the request and answers with the statistics data, which are shown to the user.

Exit conditions: The user visualizes the statistics.
Exceptions:

1. **There are no statistics to be shown:** a warning is shown to the user.


User
SafeStreets
requestStatisticsType(type)
alt
[there are available statistics]
availableStatistics
filterStatistics(filter)
elaborateData(stats, filter)
statisticsData
noStatisticsAvailable
User
SafeStreets

3.2.2 Authority

Scenarios

Scenario 2.1 Milan's Local PD cannot afford to send every morning several traffic cops to prevent people from parking on the crosswalks, there are more urgent works to do! But at the same time this type of infraction cannot be backed off. To fix this problem, the Local PD can use *Safe Streets*. A Local PD operator can see from his screen all the reports regarding that specific type of violation and then take action against them.

Scenario 2.2 The city of Pesaro has activated *SafeStreets* in its area two years ago and it has also installed a system to keep track of traffic-related accidents, in order to take advantage of the *SmartSuggestions* feature. Recently a lot of car crashes have been registered in the intersection between via Venezia and via Milano, mainly because the cars from via Venezia didn't give the other ones right of way. In the same period, *SafeStreets* has received numerous reports of cars parked at the corners of that intersections. *SmartSuggestions* is able to cross the data and infer that the parked cars might have obstructed the view of the drivers coming from via Venezia, who failed to see the other cars approaching. So it emits a suggestion to build *bollards* to physically block illegal parking at the corners of the intersection. An employee of the local police logs into SafeStreets on his office computer and check if there are any suggestions and the system shows the newly created one. After brief considerations, the Police orders to build the suggested bollards, so another operator comes back to SafeStreets and marks the action as *carried out*. After a few weeks no more accidents have occurred at the intersection, so *SmartSuggestions* marks the suggested action as useful, and it will likely propose it again in similar situations.

Use cases

In figure 3.4 are shown the various use cases in which the authority participates. Then each use case is described in a detailed way with also a sequence diagram that shows the interaction steps.

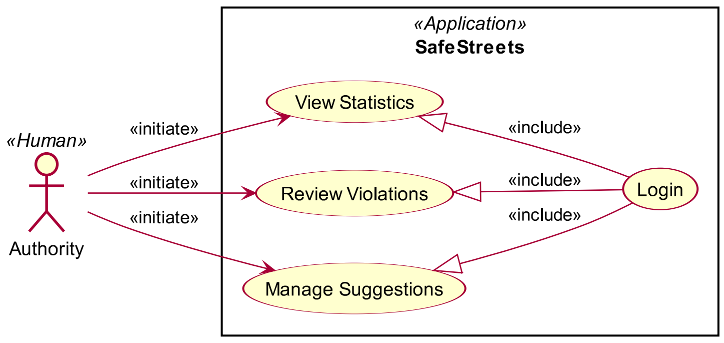


Figure 3.4: Use case diagram for the authority

Use case 2.1

Name: Review Violations
Actors: Authority
Entry Conditions:

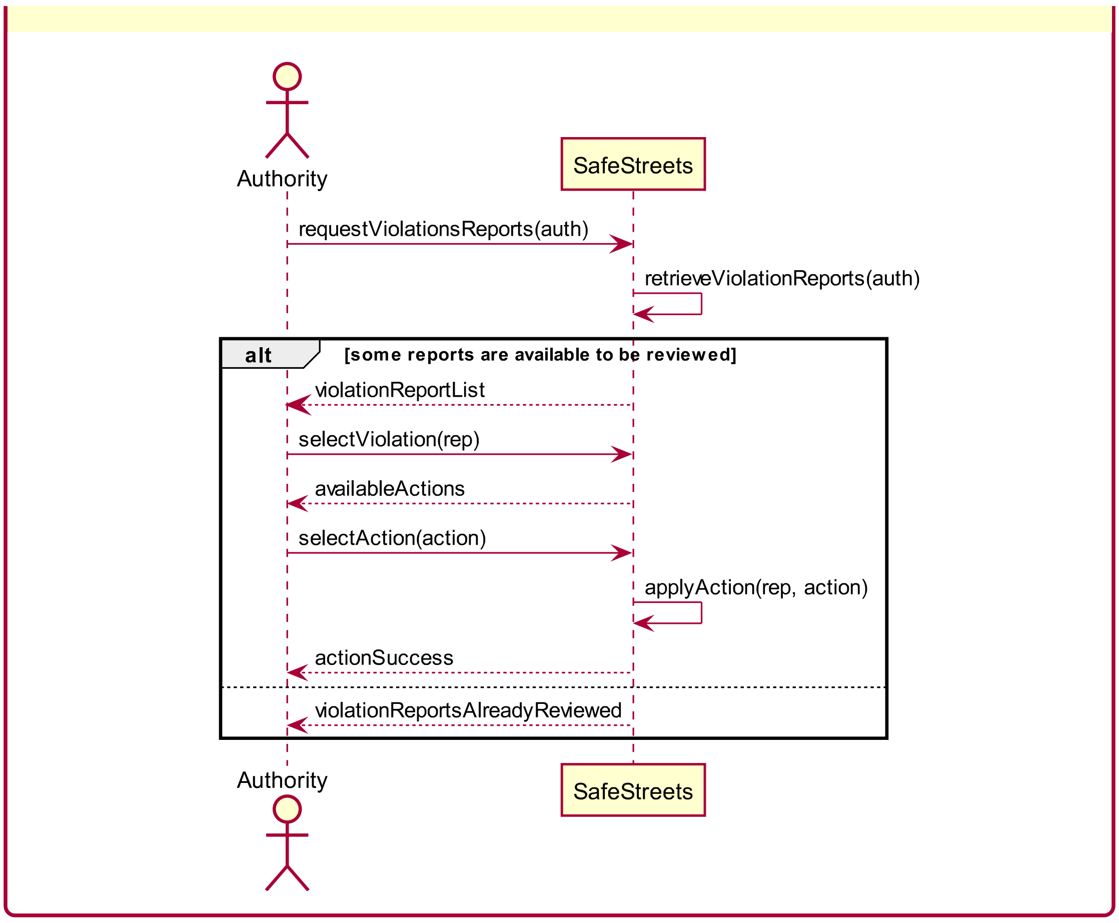
1. The authority operator has opened the application on the device.
2. The authority operator has already logged in.
3. The device is connected to the internet.

Event flow:

1. The authority operator clicks on the *Violations* button.
2. The authority operator is shown all the Violation Reports send by the users.
3. The authority operator selects the Violation Report he wants to review.
4. The system shows the options available for the Report: Accept or Discard.
5. The authority operator selects the action he wants to take.
6. The system elaborates the action and sends the operator a confirmation message.

Exit conditions: The authority operator selects an action to be taken on the specific Violation Report
Exceptions:

1. **There are no violation report to review:** the authority operator is shown a message stating that all the reports has been reviewed.



Authority
SafeStreets
requestViolationsReports(auth)
retrieveViolationReports(auth)
alt
[some reports are available to be reviewed]
violationReportList
selectViolation(rep)
availableActions
selectAction(action)
applyAction(rep, action)
actionSuccess
violationReportsAlreadyReviewed
Authority
SafeStreets

Use case 2.2

Name: View Statistics
Actors: Authority
Entry Conditions:

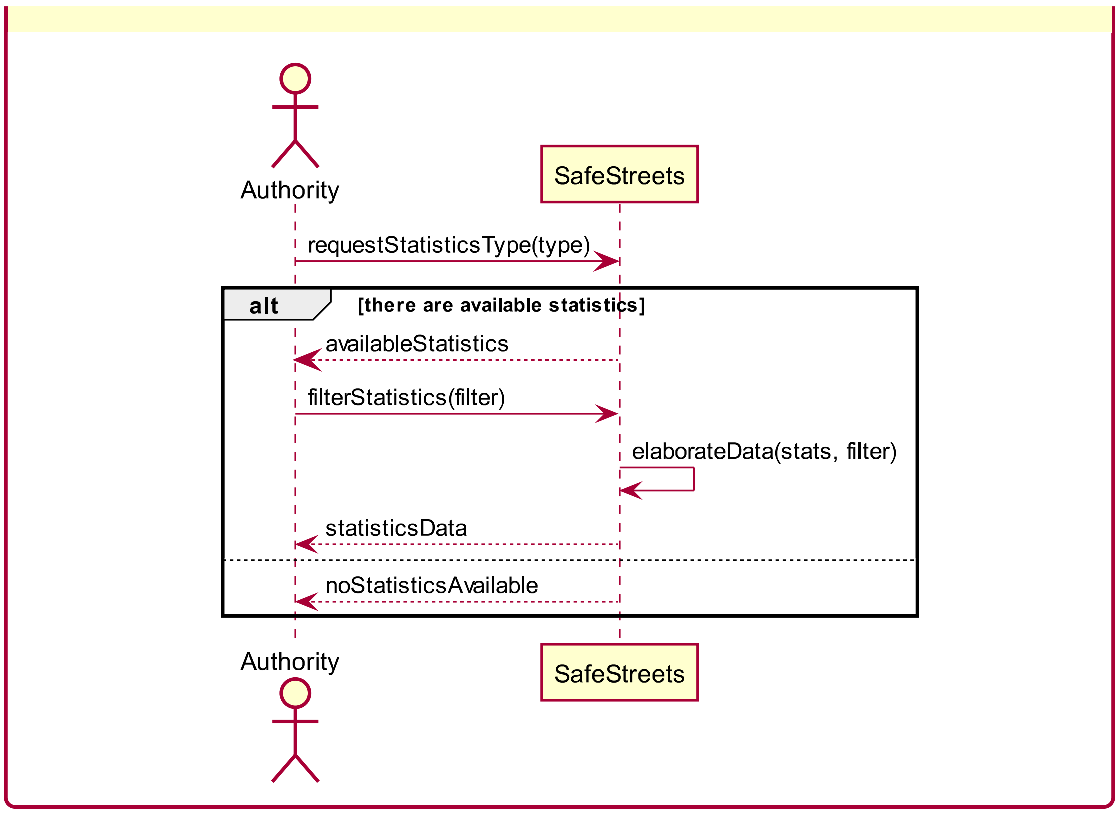
1. The authority operator has opened the application on the device.
2. The authority operator has already logged in.
3. The device is connected to the internet.

Event flow:

1. The authority operator clicks on the *Statistics* button.
2. The authority operator is shown all the available statistics type.
3. The authority operator selects the statistics he wants to see.
4. The system receives the request and elaborates the data.
5. The authority operator is shown a graph that visualizes the statistics he chose.

Exit conditions: The authority operator visualizes the statistics.
Exceptions:

1. **There are no statistics to be shown:** a warning is shown to the authority operator.


Authority
SafeStreets
requestStatisticsType(type)
alt
[there are available statistics]
availableStatistics
filterStatistics(filter)
elaborateData(stats, filter)
statisticsData
noStatisticsAvailable
Authority
SafeStreets

Use case 2.3

Name: Manage Suggestions
Actors: Authority
Entry Conditions:

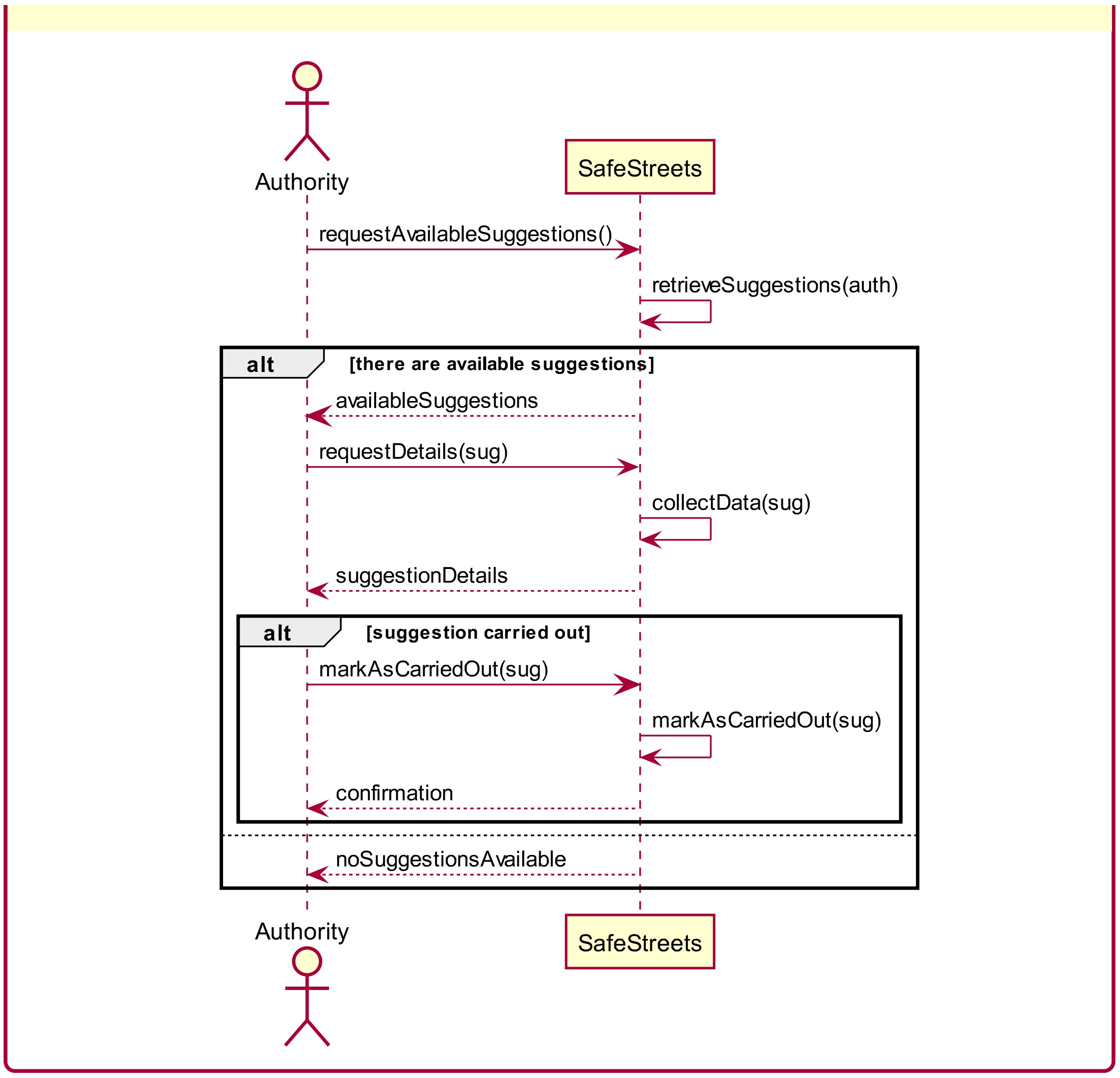
1. The authority operator has opened the application on his device.
2. The authority operator has already logged in.
3. The device is connected to the internet.

Event flow:

1. The authority operator clicks on the *Suggestions* button.
2. The authority operator is shown all the available suggestions, generated by *Smart Suggestions*.
3. The authority operator clicks on the suggestion he wants to see in detail.
4. The authority operator is shown all the details.
5. The authority operator can click con *Mark as carried out* or simply exit.

Exit conditions: The authority operator visualizes the suggestion.
Exceptions:

1. **There are no suggestions to be shown:** the authority operator is shown a message stating that there are no suggestions available.



3.2.3 Requirements

Here we programmatically introduce the requirements needed to satisfy our goals. In order to make apparent that the requirements satisfy the goals in the context of domain assumptions, for each goal we will list the requirements and domain assumptions that are needed to satisfy it. In table 3.1 on page 41 we provide a mapping between each goal and the most relevant use case related to it. It follows that the requirements that need to fulfill the goal are also strongly related to the use case.

G1 The system must allow users to notify authorities of parking violations, by uploading pictures of the violation together with metadata (date, time, position, infraction type, licence plate of the vehicle)

- **R1** The system must allow citizens to register by providing fiscal code, username and password
- **R2** The system must not allow two people (user or operator) to have the same username. Also it must not allow two users (i.e. citizens) to have the same fiscal code
- **R3** The users must be able to upload photos from their mobile devices
- **R4** The system must be able to detect the date, time and location of the users' devices
- **R5** The system must allow the user to select the type of violation he wants to report from a finite list of types
- **R6** The system must allow the user to send violation reports
- **R7** The system must allow the user to edit each field of the report before sending it
- **R8** The system must be able to determine whether a plate number is valid (i.e. registered to the DMV) or not
- **D1** A citizen can be uniquely identified by his fiscal code
- **D2** The users' devices provide date and time with a precision of 1 minute or less, with resepect to UTC
- **D3** The users' devices provide GPS location with a precison of 10 meters or less
- **D4** Any vehicle that may commit a violation can be uniquely identified by its licence plate number
- **D8** The internet connection works without problems
- **D9** The system can access the database of registered licence plates

G2 The system must be able to extract the licence plate number from the photos uploaded by the users

- **R3** The users must be able to upload photos from their mobile devices
- **R8** The system must be able to determine whether a plate number is valid (i.e. registered to the DMV) or not
- **R9** The system must be able to detect the biggest licence plate in a photo and read its number
- **D9** The system can access the database of registered licence plates

G3 The system must persistently store the violations uploaded by the users, while keeping the uploader anonymous

- **R10** The system must be able to persistently store the violation reports, including their attached photos
- **R11** Violations with malformed data (invalid date/time, inexistent licence plate number, missing fields) must not be stored by the system

- **R12** When storing a violation report, the system mustn't save the user who uploaded it. In particular it must not store any identifier (e.g. fiscal code, username) that can provide such associtation

G4 The authorithy must be able to visualize, accept and refuse the single violation reports that occurred in its competence area

- **R2** The system must not allow two people (user or operator) to have the same username. Also it must not allow two users (i.e. citizens) to have the same fiscal code
- **R10** The system must be able to persistently store the violation reports, including their attached photos
- **R13** The system must allow the operators of authorities to login with username and password
- **R14** The system must be able to determine the municipality in which a violation occurred based on the location saved in the report. It must be able to assign the violation to the authorithy responsible for that municipality, if any
- **R15** An operator must be able to access only the violation reports assigned to its authorithy. When a new report has been stored, he must be able to mark it as accepted or discarded
- **R16** Discarded reports are eventually deleted from the system
- **D3** The users' devices provide GPS location with a precison of 10 meters or less
- **D5** Each location where the service can be used is associated to exactly one municipality
- **D6** Each municipality has exactly one authorithy that enforces traffic rules in its territory
- **D7** Authorities have demonstrated that the municipalities for which they requested the violations are under their jurisdiction
- **D8** The internet connection works without problems

G5 The authorithy must be able to visualize violation statistics based on their location, time, type or responsible vehicle

- **R2** The system must not allow two people (user or operator) to have the same username. Also it must not allow two users (i.e. citizens) to have the same fiscal code
- **R10** The system must be able to persistently store the violation reports, including their attached photos
- **R13** The system must allow the operators of authorities to login with username and password
- **R17** The system must be able to filter and aggregate violation reports by location, date, time, type and licence plate number

- **R18** The system must be able to compute statistics based on the number of violations
- **R19** The system must allow an authority operator to view statistics, including filtering for the licence plate of the vehicles
- **D8** The internet connection works without problems

G6 The user must be able to visualize anonymized violation statistics only based on their location, time and type

- **R10** The system must be able to persistently store the violation reports, including their attached photos
- **R17** The system must be able to filter and aggregate violation reports by location, date, time, type and licence plate number
- **R18** The system must be able to compute statistics based on the number of violations
- **R20** The system must allow an user to view statistics which do not include the licence plate of the vehicles involved
- **D8** The internet connection works without problems

G7 The system must be able to determine unsafe areas by matching data about violations and data about accidents. It must then use such data to suggest actions that the authorithy can take to help preventing accidents

- **R10** The system must be able to persistently store the violation reports, including their attached photos
- **R21** The system must be able to retrieve data about accidents from municipalities that provide it (through an API)
- **R22** The system must be able to determine if there is a causal relation between accidents and violations based on their location, types and timestamps
- **R23** The system must be able to determine possible actions that can be taken by the authorithy to reduce the detected accidents, also using the detected causal relations
- **R24** An authorithy operator must be able to view the suggested actions for the zone in which the authorithy operates. He must also be able to mark an action as *carried out*, meaning that it has actually been executed
- **R25** The system must be able to detect if a *carried out action* has been useful or not (i.e. the number of accidents and/or violations that it was supposed to prevent has dropped or not) and possibly provide better suggestions based on this knowledge
- **D5** Each location where the service can be used is associated to exactly one municipality

Table 3.1: Traceability matrix

Goal ID	Use case ID
G1	UC1.3
G2	UC1.3
G3	UC1.4
G4	UC2.1
G5	UC2.2
G6	UC1.4
G7	UC2.3

D6 Each municipality has exactly one authorithy that enforces traffic rules in its territory

D7 Authorities have demonstrated that the municipalities for which they requested the violations are under their jurisdiction

D8 The internet connection works without problems

D10 The data about an accidents contains its location, timestamp and type (within a finite list of types)

3.3 Performance Requirements

The aim is to build a unique system that is able to serve an entire state. Since authorities can opt-in and out of the service, it is required for the system to be *scalable* from tens of thousands of citizens to few millions and up to thousands of authorities. In particular the system must be able to react to a large number of simultaneous requests from both citizens and authorities.

The response time for simple operations like reporting and reviewing violations should be less than 7 seconds; for more complex operations, like computing statistics over large data sets, the response time can also be longer (in the order of tens of seconds). However, since this is not a critical system, greater response times can also be tolerated, but only sporadically.

Regarding the *SmartSuggestions* feature, we cannot set a precise boundary on the time necessary to provide a suggested action, also because the involved computations might be very heavy. The aim is that the newly inserted data is processed within a week, but the exact time is left unspecified so processing can be efficiently scheduled, for example, when the overall load of the system is low.

3.4 Design Constraints

3.4.1 Standards compliance

Here we set some standards for data representation, that match the ones of the planned development country: Italy.

- Fiscal codes are represented as capitalized strings. (e.g. `MRTMTT25D09F205Z`)
- Licence plates are represented as capitalized strings without spaces (e.g. `AA123BB`).
- Latitude and longitude are represented as decimal degrees (DD), which is the most common representation format in geographical information systems [1]; in this way they can be conveniently stored as floating point numbers.

Since the system deals with personal data, in the form of fiscal codes and licence plate numbers, it must comply to the General Data Protection Regulation, which applies to all individual citizens of the European Union (EU) and the European Economic Area (EEA). The treatment of data is done by police forces, so the system is also subject to the Italian law DPR-15/2018 [3] that regulates this matter.

3.4.2 Hardware limitations

As described in section 2.4 the service is accessed from a smartphone app on the user side and from the web browser from the authority side. In both cases the Internet is used to communicate between the end user devices and the core of the system.

The smartphones are therefore required to have an internet connection, a camera (needed to take pictures of violations) and a GPS sensor (to locate the violation).

3.5 Software System Attributes

Here we provide a brief description of the desired attributes that the software to be should have.

3.5.1 Reliability

The system should be able to run continuously without any interruptions. To achieve this it should follow some basic principles of fault tolerance; the preferred approach is active replications, since it also provides scalability.

3.5.2 Availability

The system must be perceived as stable and must be there when users need it. However it is not a life-critical system, so a limited downtime (e.g. for maintenance) can be accepted. To sum up, the system must be available 99% of the time.

3.5.3 Security

Since the system will deal with personal data, security both in data storage and in communication are key aspects.

All the *communication* between the clients and servers, and also between different data centers, must provide confidentiality, authentication and integrity; this can be achieved, for example, by using the Transport Layer Security protocol (TLS).

The *data* stored by the system must be protected against unauthorized access and passwords must not be stored as plaintext; a suggestion is to use hashes and salt.

3.5.4 Maintainability

Another key aspect of the system is maintainability. In particular the design must provide the flexibility to add new features on top of the existing ones. It must also be easy to adapt the system to operate in other countries, that may have different formats for personal and vehicle identifiers.

3.5.5 Portability

The system must provide a high level of compatibility with different devices and technologies both on the client and the server sides.

On the *client side* this means that the mobile application can run on the major mobile operating systems (iOS and Android), without particular restrictions on the OS version or specific device models. For the web part, this means that the website must work in all the major web browsers.

On the *server side* this means that the whole system must be broken up into loosely coupled components that interact between themselves and with the clients. The components should also be easy to substitute or move to a different location (e.g. hosting provider). These practices also increase the maintainability and scalability of the system.

4 Formal Analysis

In this chapter we will formally analyze our representation of the world as described in Figure 2.1 using the Alloy language. Firstly we define all the entities that populate the world, then we specify a set of constraints that keep the representation coherent. Lastly we define some functions that allow to see some properties of the model.

4.1 Entities definition

```
sig Username, Password {}

abstract sig Person {
        username : one Username,
        password : one Password
}

sig FiscalCode{}

sig User extends Person {
        id : one FiscalCode
}

sig Operator extends Person {}

sig Municipality {}

sig Authority {
        employs : some Operator,
        zone : some Municipality
}

sig Vehicle {
        plate : one LicensePlate
}

sig LicensePlate {}

sig Violation {
        where : one Location,
        vehicle : one Vehicle,
        picture : one Picture
}

sig Picture{}

sig Location {
        belongsTo : one Municipality,
        latitude : one Int,
        longitude : one Int
}{
        latitude > -90 and latitude < 90 and
        longitude > -180 and longitude < 180
}

sig Accident{
        crashWhere : one Location
}

sig SuggestedAction {
        possibleCause : some Violation,
        prevents : some Accident
}
```

4.2 Constraints

Here are defined all the constraints needed to keep the model coherent. The majority of these are called *noFoster[...]* and the name self describes what the constraint does: we do no want usernames or passwords without their own owner, same for pictures without their own violation, license plate without vehicle and so on.

The second set of constraint defines the uniqueness of some entities, for example there cannot be two different people with the same fiscal code or two vehicles with the same license plate. These are constraints that are needed due to real world limitations, but similar constraints are defined for entities not present in the real world, for example *Username*, which has to be unique only due to our model design.

```
fact noFosterFiscalCode {
        all fisc : FiscalCode | fisc in User.id
}

fact UniqueUsername {
        no disj a, b: Person | (a.username = b.username)
}
```

```
fact noFosterPassword {
        all p : Password | p in Person.password
}

fact noFosterUsername {
        all u : Username | u in Person.username
}

fact noFosterOperator {
        all op : Operator | op in Authority.employs
}

fact noFosterMunicipality {
        all mun : Municipality | mun in Authority.zone
}

fact noFosterVehicles {
        all v : Vehicle | v in Violation.vehicle
}

fact noFosterLocation {
        all loc : Location | loc in Violation.where or loc in
           ↪  Accident.crashWhere
}

fact noFosterLicensePlate{
        all lp : LicensePlate | lp in Vehicle.plate
}

fact noFosterPicture{
        all pic : Picture | pic in Violation.picture
}

fact noMultiJobs {
        all disj a, b : Authority | (no op : Operator |
                op in a.employs and op in b.employs)
}

fact onlyOneAuthorityPerPlace {
        all disj a, b : Authority | (no mun : Municipality |
                mun in a.zone and mun in b.zone)
}

fact UniqueLicensePlates {
        no disj v1, v2 : Vehicle | v1.plate = v2.plate
}

fact NoDuplicateLocations {
        no disj loc1, loc2 : Location | (loc1.latitude = loc2
            ↪ .latitude) or
                (loc1.longitude = loc2.longitude)
}

fact UniquePictures {
        no disj v1, v2 : Violation | v1.picture = v2.picture
}

fact UniqueFiscalCode{
        no disj u1, u2 : User | u1.id = u2.id
}
```

4.3 Functions

Here we describe a function that shows which *Violations* are visible to a certain *Operator*. We make use of two auxiliary functions in order to increase readability.

```
fun getAuthorityFromOperator[op : Operator]: set Authority {
        employs.op
}

fun getViolationsFromMunicipality[mun : Municipality]: set
    ↪ Violation{
        where.belongsTo.mun
}

fun visibleViolations[op : Operator]: set Violation{
        getViolationsFromMunicipality[
            ↪ getAuthorityFromOperator[op].zone]
}
```

4.4 Generated Worlds

With the following command we can generate some examples of the model we formalized.

```
run show {
        #Operator > 1
        #User > 1
        #Location > 1
        #Violation > 1
} for 4 but 9 Int
```

In particular we are asking the solver to have in our sample world at least 2 *Operators*, 2 *Users*, 2 *Locations* and 2 *Violations*. Furthermore we are limiting the maximum cardinality for each entity at 4, and the Integers in our model have a bitwidth of 9.

The output of the Alloy Analyzer for the previous command is the following:

```
Executing "Run show for 4 but 9 int"
    Solver=sat4j Bitwidth=9 MaxSeq=4 SkolemDepth=1 Symmetry
       ↪ =20
    108896 vars. 4360 primary vars. 376105 clauses. 574ms.
    Instance found. Predicate is consistent. 1736ms.
```

and it shows that all the facts are consistent. The world generated with the previous command is shown in figure 4.1.

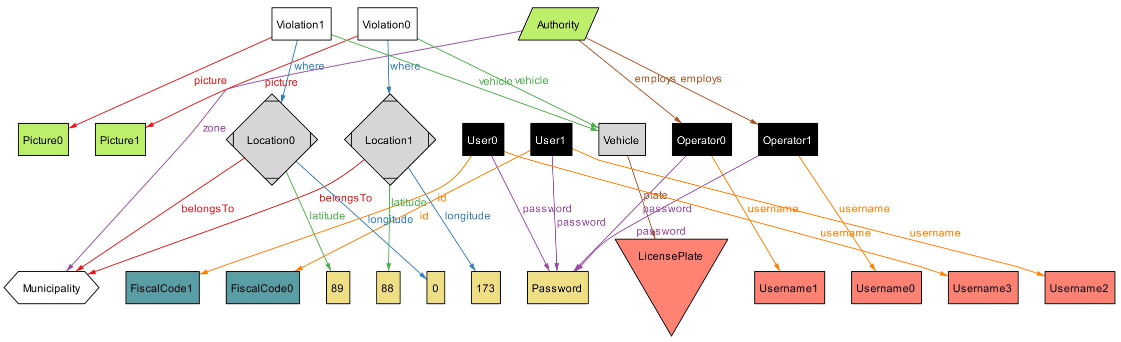


Figure 4.1: World generated from the Alloy model

5 Effort spent

Here we show the hours that each of the authors has spent on specific tasks for creating this document.

Alessandro Fulgini

Task	Time spent
Introduction	12h 30m
Overall description	5h 30m
Mockups	7h
External interface requirements	3h
Functional requirements	6h 30m
Non-functional requirements	5h
Total	**39h 30m**

Federico Di Cesare

Task	Time spent
Introduction	6h
Product perspective	3h
Product functions	2h
User characteristics	2h
Scenarios	1h
Use cases	6h
Formal analysis	7h
Total	**27h**

6 References

[1] *Decimal Degrees on Wikipedia*. 2019. URL: https://en.wikipedia.org/wiki/Decimal_degrees.

[3] *Decreto del Presidente della Repubblica 15 gennaio 2018, n. 15*. 2018. URL: https://www.gazzettaufficiale.it/eli/id/2018/3/14/18G00040/sg.

[4] *General Data Protection Regulation (GDPR)- Article 4 - Definitions*. 2016. URL: https://gdpr.eu/article-4-definitions/.