



POLITECNICO DI MILANO 1863

SOFTWARE ENGINEERING 2 PROJECT

Requirement Analysis and Specification Document (RASD)

---

# SafeStreets

Version 1.0

---

*Authors*

Tiberio Galbiati  
Saeid Rezaei

*Supervisor*

Dr. Matteo Rossi

**Copyright:** Copyright © 2019, Tiberio Galbiati - Saeid Rezaei – All rights reserved

**Download page:** <https://github.com/TiberioG/GalbiatiRezaei.git>

November 9, 2019

# Contents

<b>Table of Contents</b>	<b>2</b>
<b>List of Figures</b>	<b>4</b>
<b>List of Tables</b>	<b>4</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Purpose	5
1.1.1 Description of the given problem	5
1.1.2 Goals	5
1.2 Scope	6
1.3 World and shared phenomena	6
1.4 Definitions, acronyms, abbreviations	6
1.4.1 Definitions	6
1.4.2 Acronyms	6
1.4.3 Abbreviations	7
1.5 Revision history	7
1.6 Reference Documents	7
1.7 Document Structure	7
<b>2 Overall Description</b>	<b>8</b>
2.1 Product perspective	8
2.1.1 Class Diagram	8
2.2 Product functions	10
2.2.1 Report violation	10
2.2.2 Explore Data	10
2.2.3 Issue a ticket	10
2.2.4 Ticket statistics	13
2.3 User characteristics	13
2.4 Assumptions, dependencies and constraints	13
2.4.1 Domain assumptions	13
2.4.2 Dependencies	14
<b>3 Specific Requirements</b>	<b>15</b>
3.1 External Interface Requirements	15
3.1.1 User Interfaces	15
3.1.2 Hardware Interfaces	21
3.1.3 Software Interfaces	21
3.1.4 Communication Interfaces	21
3.2 Functional Requirements	21
3.2.1 Use Cases diagrams	23
3.2.2 Use Cases Description	24
3.2.3 Sequence diagrams	30
3.2.4 Requirements traceability matrix	32
3.3 Performance Requirements	33
3.4 Design Constraints	33
3.4.1 Standards compliance	33
3.4.2 Hardware limitations	33
3.4.3 Any other constraint	33

3.5	Software System Attributes . . . . .	34
3.5.1	Simple User Interface . . . . .	34
3.5.2	Reliability . . . . .	34
3.5.3	Availability . . . . .	34
3.5.4	Security . . . . .	34
3.5.5	Maintainability . . . . .	34
3.5.6	Portability . . . . .	34
<b>4</b>	<b>Formal Analysis Using Alloy . . . . .</b>	<b>35</b>
4.1	Abstract Entity and Signature . . . . .	35
<b>5</b>	<b>Effort Spent . . . . .</b>	<b>39</b>

## List of Figures

1	High-level Class Diagram . . . . .	9
2	Violation reporting state diagram . . . . .	11
3	Tickets creation and approval state diagram . . . . .	12
4	[GUI] Login screen . . . . .	15
5	[GUI] Report   open camera view . . . . .	16
6	[GUI] Report   picture of violation taken screen . . . . .	17
7	[GUI] Report   violation info form screen . . . . .	18
8	[GUI] Explore data screen   offenders . . . . .	19
9	[GUI] Explore data   heatmap . . . . .	20
10	[GUI] Ticket   approval screen . . . . .	21
11	Basic service use case diagram . . . . .	23
12	Advanced function use case diagram . . . . .	24
13	Sequence Diagram for Violation Reporting . . . . .	30
14	Sequence Diagram for HeatMap visualization . . . . .	31
15	Sequence Diagram for ticket Creation and approval . . . . .	32

## List of Tables

1	World and Machine Table . . . . .	6
2	Traceability matrix . . . . .	33

# 1 Introduction

## 1.1 Purpose

This document represents the Requirement Analysis and Specification Document (RASD). Goals of this document are to completely describe the system in terms of functional and non-functional requirements, analyze the real needs of the customer in order to model the system, show the constraints and the limit of the software and indicate the typical use cases that will occur after the release. This document is addressed to the developers who have to implement the requirements and could be used as a contractual basis.

### 1.1.1 Description of the given problem

SafeStreets is a crowd-sourced application that intends to provide users with the possibility to notify authorities when traffic violations occur, and in particular parking violations. The application allows users to send pictures of violations, including their date, time, and position, to authorities. Examples of violations are vehicles parked in the middle of bike lanes or in places reserved for people with disabilities, double parking etc.

SafeStreets stores the information provided by users, completing it with suitable meta-data every time it receives a picture. In particular it uses an external service which reads the license plate and stores the decoded string of the plate. Also it stores the type of the violation which is input by the user from a provided list. Lastly it stores the name of the street where the violation occurred which is retrieved from the geographical position where the user took the picture. In addition, the application allows both end users and authorities to mine the information that has been received. Two visualizations are offered: the first is an interactive map where are highlighted with a gradient color the streets with the highest frequency of violations. The second is a list of the vehicles that committed the most violations (available only to authority users)

In addition the app offers a service that creates automatically traffic tickets which can be approved and sent to citizens by the local police. This is done using the data crowd-sourced by the users. The application guarantees that every picture used to generate a ticket has't been altered. In addition, the information about issued tickets is used to build statistics. Two kind of statistics are offered: a list of people who received the highest number of tickets and some trends of the issued tickets over time and the ratio of approved tickets over the violations reported.

### 1.1.2 Goals

- [G1] Allow users to notify authorities about traffic violations
- [G2] Allow users to send pictures with metadata of violations
- [G3] Allow users to mine information recorded
- [G4] Have at least two different privilege for mining data
- [G5] Generate traffic tickets
- [G6] Generate statistics about issued tickets
- [G7] Be sure every information uploaded is never altered

## 1.2 Scope

### 1.3 World and shared phenomena

Here are listed the phenomena related to the "machine" which means the software-to-be with the required working hardware and the "world" which is the real environment affected by the "machine". A phenomena can be shared by both machine and world if it's controlled by the world and observed by the machine or controlled by the machine and observed by the world.

Phenomenon	Shared	Who controls it
User wants to report a violation	N	W
User takes a picture	Y	M
The machine decodes the plate from the picture	Y	M
The user knows the reason why the vehicle is in violation	N	W
The machine asks the user for the kind of violation	Y	M
The machine stores the violation reported	N	M
The user wants to see data visualization	N	W
The machine shows the data visualization	Y	M
The machine creates a ticket in the system	N	M
The machine checks any alteration of the picture	N	M
The authority user approves the ticket	Y	W
The authority user doesn't approve the ticket	Y	W
the authority sends the ticket to the offender	N	W

Table 1: World and Machine Table

## 1.4 Definitions, acronyms, abbreviations

### 1.4.1 Definitions

- Heatmap : A heatmap is a graphical representation of data that uses a system of color-coding to represent different values
- Enduser : a regular citizen which will use the app
- Authority user : someone who's working for an authority like (police, municipality etc.) recognized
- Geocoding : the process of converting addresses (like a street address) into geographic coordinates (latitude and longitude)
- Reverse geocoding: the process of converting geographic coordinates into a human-readable address

### 1.4.2 Acronyms

- ALPR : Automated Licence Plate Recognition
- GUI : Graphical User Interface
- GDPR : EU General Data Protection Regulation
- API : Application Programming Interface

### **1.4.3 Abbreviations**

## **1.5 Revision history**

This is the first released version 10/11/2019.

## **1.6 Reference Documents**

### **References**

- [1] OpenALPR Technology Inc. , OpenALPR documentation <http://doc.openalpr.com>
- [2] Ministero delle infrastrutture e dei Trasporti, DECRETO LEGISLATIVO 30 aprile 1992, n. 285 Nuovo codice della strada, <https://www.normattiva.it/uri-res/N2Ls?urn:nir:stato:decreto.legislativo:1992-04-30;285!vig=>
- [3] GOOGLE inc, Google Maps Platform Documentation | Geocoding <https://developers.google.com/maps/documentation/geocoding/start>
- [4] GOOGLE inc, Google Maps Platform Documentation | Heatmap <https://developers.google.com/maps/documentation/javascript/heatmaplayer>

## **1.7 Document Structure**

## 2 Overall Description

### 2.1 Product perspective

#### 2.1.1 Class Diagram

The following class diagram is a high-level class diagram which should be intended as a model of the application structure. During the implementation part more classes and attributes can be created and used.

**User** This class is the father of the two possible kind of users: **EndUser** and **Authority** which are needed because our application is intended to be multi-user and with at least two privileges for data that can be viewed and possible functions accessible.

**Location** Every user is in a **Location** class used to represent the location as latitude and longitude coming from the OS of the smartphone.

**Reverse Geocoding** This interface is used to communicate with the external Geocoding service to get a readable address from the coordinates as explained in section 2.4.2.

**Violation** This class is used to store all the data related to the reported violation. The *kind* attribute is selected by the user from a list of possible kind of violations in the form section as explained in [UC4b]. In the **Violation** class we store also the raw latitude and longitude in case there will be need of those data later, as an example if it's impossible get precise location using reverse geocoding.

**Photo** This class is needed to represent the photo of the violation. It's mandatory that one and only one picture is associated to every violation.

**ALPR** This interface is needed to interact with the external ALPR service which receives a picture and returns a string containing every licence plate found in the picture. This interface is used to complete the attribute *licence plate* of every violation.

**MinedInfo** Classes **MinedStreets** and **MinedOffenders** are used to represent the data coming from the database of all violation and processed to offer different kind of visualizations.

**Heat Map visualizer** This interface is used to communicate with the external service providing a map of streets with an overlay highlighting the spots where violation occurred.

**Ticket** This class is used to represent the ticket with the fine for the owners of violating vehicles. Every instance will be automatically created by the system, using data coming from the instances of the **Violation** class. This data has to be combined with data from some authority database eg. the database of all registered vehicles plates, the database of violation of the traffic law. The attribute *valid* is a boolean value, set by the class **CheckValidity** TRUE if the picture associated to the corresponding violation has't been altered. Otherwise the ticket is considered not valid, and the attribute is set to FALSE. The attributes *kind violation*, *violation street*, *violation street numb*, *licence plate*, *date violation* are copied from the instances of **Violation**. The attributes *amount* and *date payment due* are filled by the connection with the external authority database containing the amount of money to be paid for every fine and the standard deadline for payment. The attribute *Offender name* and *Offender surname* are used to store the identity of the owner of the



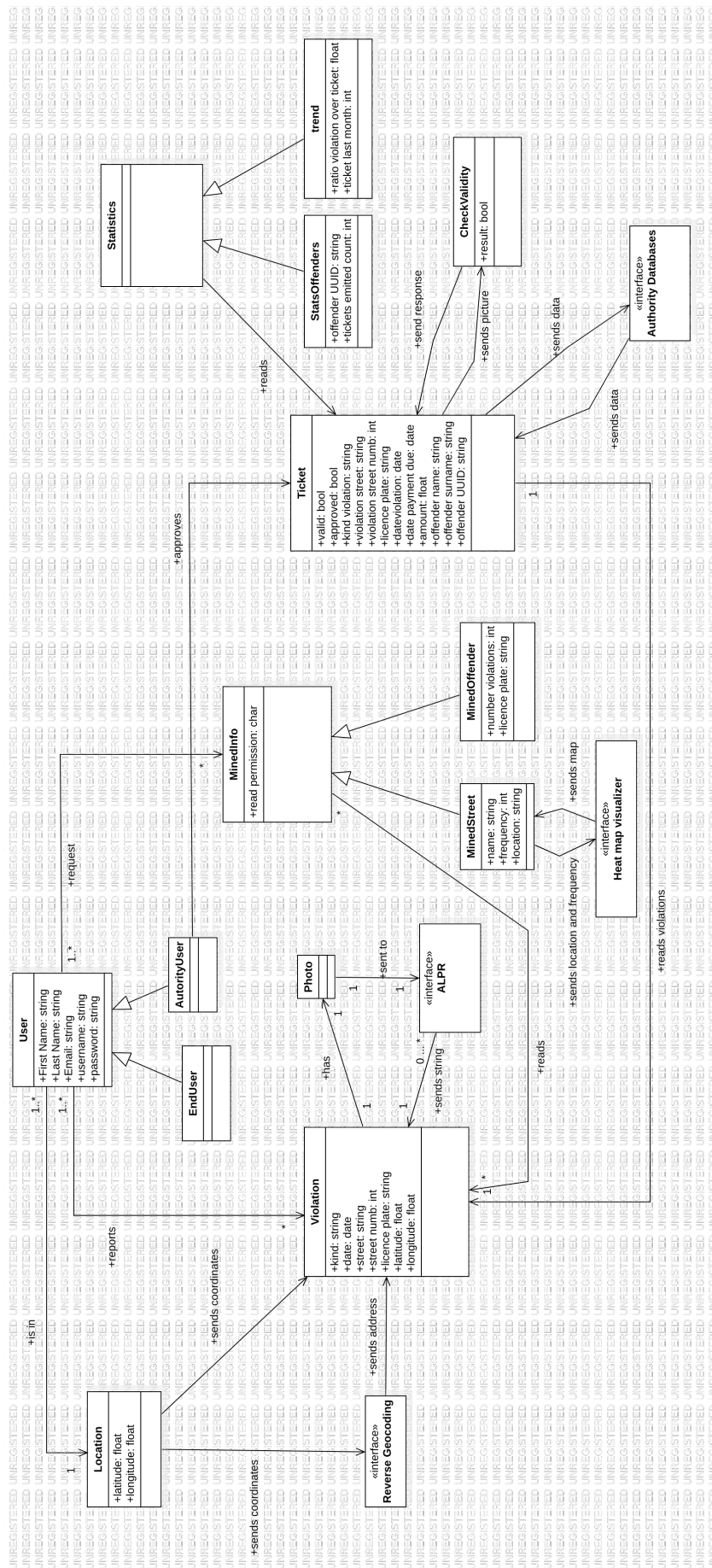


Figure 1: High-level Class Diagram

veichle, these should be filled knowing the plate and quering the external licence plate registration database.

**Statistics** The classes **StatsOffenders** and **StatsTrends** are used to represent the data about the issued ticket and used for visualizations.

**CheckValidity** Is a class used to check if the picture taken has been altered or not. It's either possible to directly implement this functionality or use an external service.

## 2.2 Product functions

In this section are listed and explained the main functionalities of our application.

### 2.2.1 Report violation

The main function of SafeStreets is allowing users to report traffic violations, in particular when parking violations occur. User will be required to take a picture of the vehicle responsible of the violation and select the kind of violation. Once opened, the app will show on display the camera recording mode in order to start reporting the violation by taking the picture. Some alerts will appear reminding the user he has to include the plate of the vehicle and the violation must be visible. After the picture has been taken it can be possible there are other plates visible in the picture, not related to the vehicle the user is reporting. So the app will give the user a tool to cover using his finger these plates. In case no plates are found the system will ask the user to take again the picture. After the picture has successfully recorded, the system will automatically decode the licence plate and the name of the street where the user is. The user will be required to fill in a form where are listed the possible descriptions of violations.

### 2.2.2 Explore Data

The app will offer the possibility to the users to visualize the data collected. Two kinds of visualizations are offered:

1. Heatmap of streets where most violations occurred
2. Vehicles that committed the most violations

In order to get those data the system will periodically query the database of violations in order to create a table where the count of violation is stored, both for streets and vehicles. There will be a section in the app called "Explore Data" where will be able to choose which kind of data to visualize. Only A

### 2.2.3 Issue a ticket

This function is used to create tickets to send fines to the owners of vehicles which have been reported by SafeStreets. Every time a new violation is inserted in the database the System will use the new data available to generate a proposal of ticket, combining the data from violations with data coming from Municipality databases.

A ticket has the following structure:

1. Place where violation occurred
2. Date when violation occurred
3. Plate of vehicle

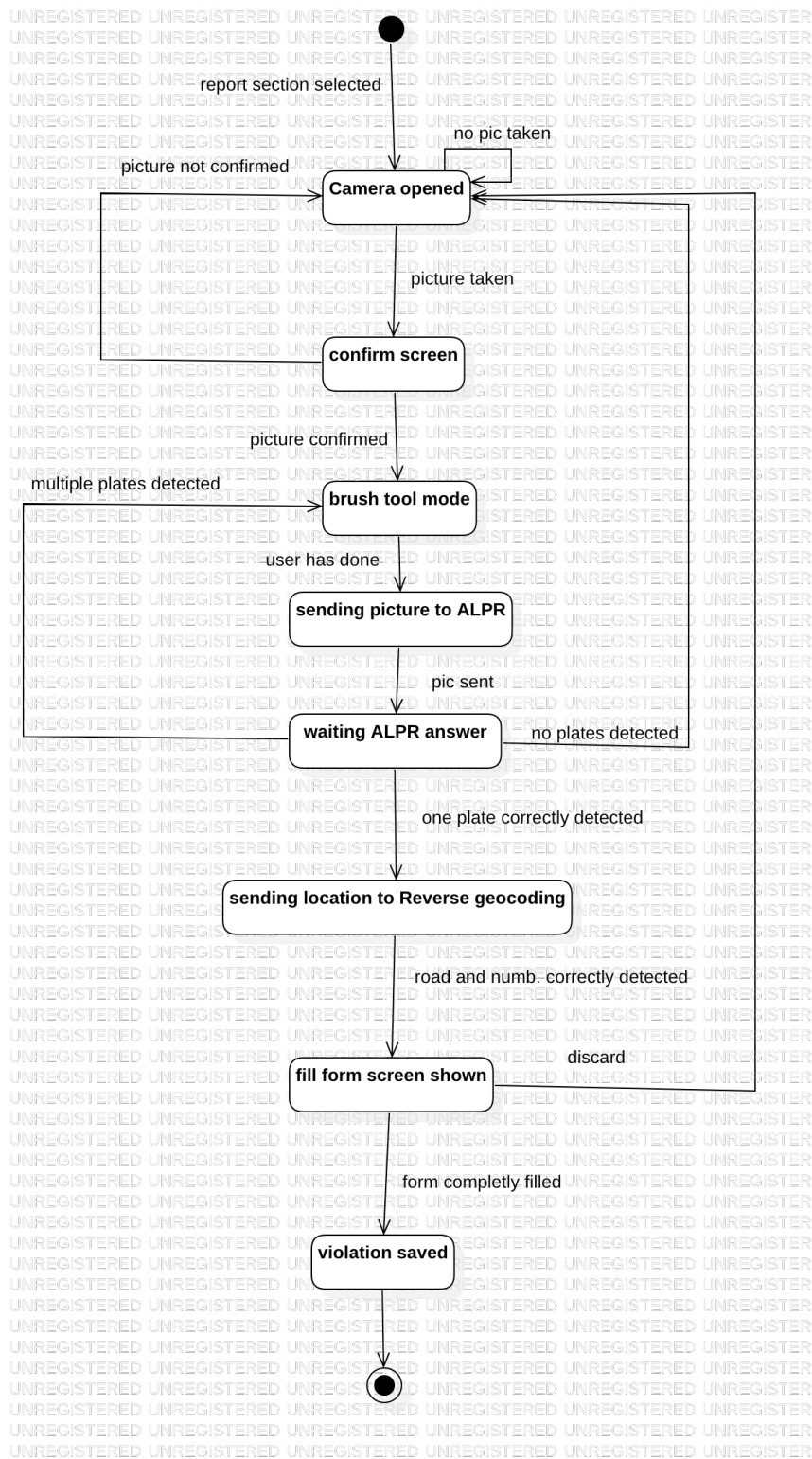


Figure 2: Violation reporting state diagram

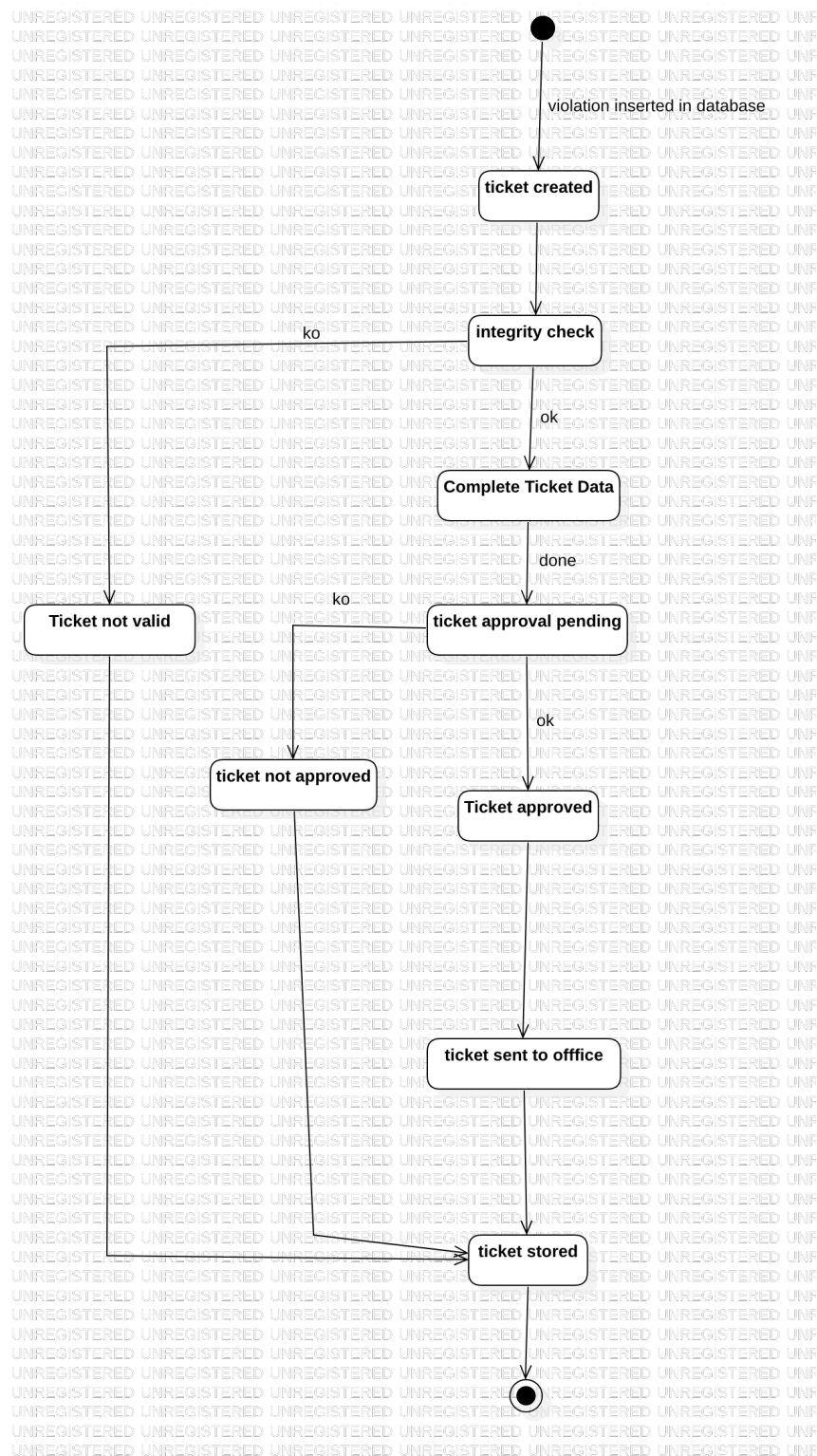


Figure 3: Tickets creation and approval state diagram

4. Article and code of violation
5. Amount to be paid
6. Date when the payment is due

Place, date, plate are data coming from the instances of Violation class. To create a complete ticket we need to associate the kind of violation to an article and code of the traffic legislation.

An external service or a code writted ad hoc will be used to check if the picture has been modified. If the result of this check is positive the ticket just created will be flagged as *valid* and will go in ticket approval state. In any other case, if the picture has been modified, the ticket is stored as *not valid* for debug purposes. Exaples of possible uses can be: bulding statistics or investigate if there are users who are trying to cheat or create spam violations.

If a ticket is considered valid, the next state is pending-approval status. Authority users (e.g. policemen) will check manually the pending-approval tickets reading all data before the approval. We have chosen to add this human control before sending the fine because every ticket should be signed by authorities. If ticket is not approved it will go in approval-denied status and will be stored for debug purposes and for statistics.

If ticket has been approved it will have to be sent to the offender. The system will connect to the external vehicle registration database in order to retrieve the name, surname, address of the offender knowing the licence plate of his/her vehicle. Now we have all the data to print the ticket and send it via regular mail. There will be an office of police-station which will do the job.

#### **2.2.4 Ticket statistics**

This function has the purpose to show

Two kind of visualizations are offered:

1. List of offenders with highest number of tickets emitted
2. Trends of emitted tickets

The first visulaization will be available only to authorities. In a section of the app it will be possible to see for each citizen who has ever received a ticket, the count of those ticket received, in descending order. Note that this is different compared to the other visulization "veichels that committed the highest number of vioations" explained in section 2. In this ticket statistic view we are considering only emitted tickets and we are aggregating all tickets of a specific citizen. Actually in case a citizen he has multiple veichles, here we are countin all the tickets emitted for all of his veichles.

### **2.3 User characteristics**

## **2.4 Assumptions, dependencies and constraints**

### **2.4.1 Domain assumptions**

- [D1] Device has a working internet connection
- [D2] Device has a camera accessible via software
- [D3] The device should acquire position with an accuracy of enouth meters in order to univocally determine the road (e.g. 5 meters)
- [D4] We have access to an ALPR service which is able to read every licence plate in a picture and return each of them as a string

- [D5] ALPR service has an accuracy of more than 95%
- [D6] The device should take pictures with enough resolution to be able to read by the ALPR service
- [D7] Every vehicle that can be reported should have a licence plate visible
- [D8] The number and kind of violations should be finite (defined by the law)
- [D9] Every authority account is verified and it's not possible to be created using the front end
- [D10] We have access to the vehicle registration database where are stored licence plates, names and the addresses of the owners of every vehicle registered
- [D11] We have access to a database where are stored all the codes of violations and the amount of fine for the violation
- [D12] The only way to upload pictures of violation is through the application

### **2.4.2 Dependencies**

Since we're creating a mobile app, the main dependency is to have a smartphone, which has to provide the following features:

1. Internet connection, possibly using 2G/3G/4G in order to be available where there is no WiFi, considering the use case "on the road"
2. A camera with good resolution
3. GPS sensor

Also there is need to use some external software or APIs :

- ALPR service : the app will be dependent on a third-party service to read the licence plate of the cars like the open source OpenALPR [1]
- Reverse Geocoding: the app will be dependent on some maps API to get the full address, knowing the coordinates of location coming from the GPS of the device. An example of this service is Google Maps API [3]
- Map and Heatmap : The app will be dependent to some Maps API used to show the map and an overlay. An example of this service is Google Maps API [4]



## 3 Specific Requirements

### 3.1 External Interface Requirements

This section provides a detailed description of all inputs and outputs from the system. It also gives a description of the hardware, software and communication interfaces and provides basic prototypes of the user interface.

#### 3.1.1 User Interfaces

In this section we present the mockups of the GUI.

Figure 4 shows the login page where it's possible to access the registration form for new users.



Figure 4: [GUI] Login screen

After login the user will enter by default into the first tab "Report", see Figure 5, where is shown what the camera is recording and the user can start taking the picture of the violation. For the first time a user logs in some banner can appear showing each function of the app. Also we show on the screen some useful reminders about how the picture to be submitted should be.



Figure 5: [GUI] Report | open camera view

After the picture has been taken, the app shows it as in Figure 6 and a "brush tool" will appear, which can be used to cover any other plate appearing in the picture, not related to the vehicle being reported.





Figure 6: [GUI] Report | picture of violation taken screen

Figure 7 shows how appears the form where the user has to select the kind of violation he is reporting from a scroll-down list. Each row has an info button, when pressed the GUI will show a verbal description of the violation related.

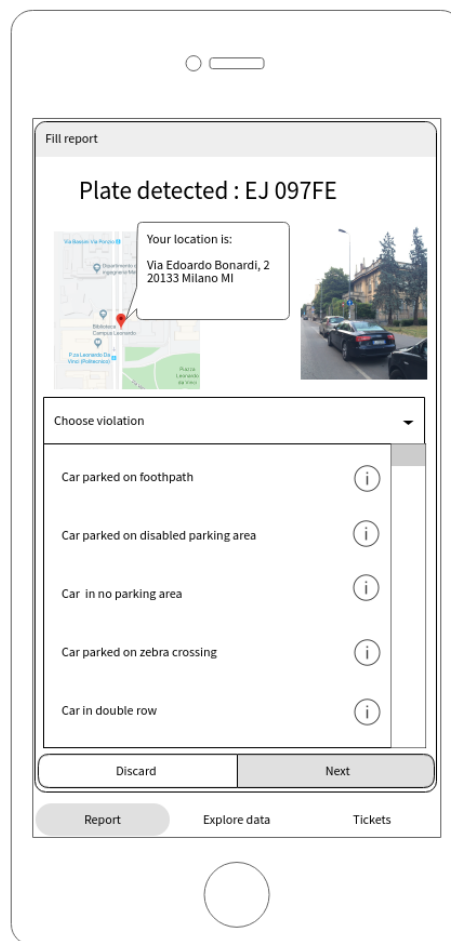
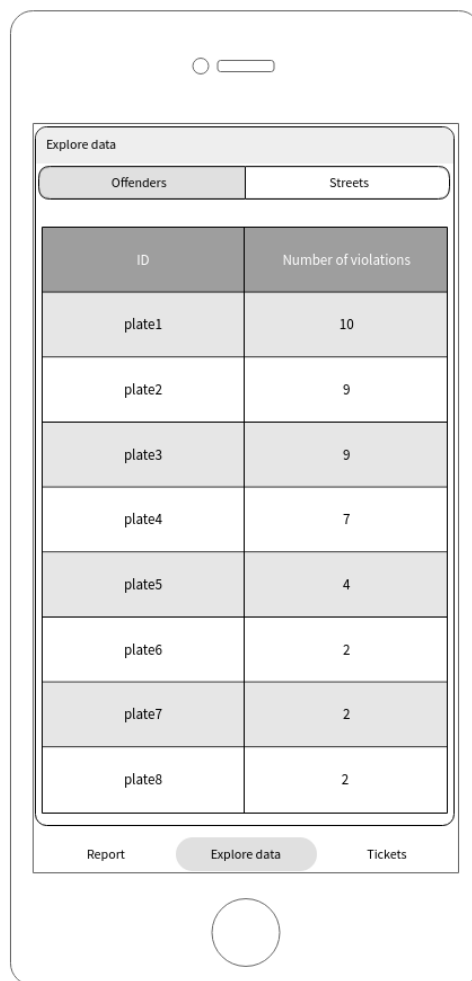


Figure 7: [GUI] Report | violation info form screen

Figure 8 shows the view of the plates which have committed the highest number of violations.



The image shows a mobile application interface for the 'Explore data' screen, specifically for offenders. At the top, there is a header 'Explore data' with two tabs: 'Offenders' (selected) and 'Streets'. Below the tabs is a table with two columns: 'ID' and 'Number of violations'. The table contains eight rows of data, alternating between light and dark gray backgrounds. At the bottom of the screen, there is a navigation bar with three buttons: 'Report', 'Explore data' (highlighted), and 'Tickets'.

ID	Number of violations
plate1	10
plate2	9
plate3	9
plate4	7
plate5	4
plate6	2
plate7	2
plate8	2

Figure 8: [GUI] Explore data screen | offenders

Figure 9 shows the view of the heatmap, with a map generated by the external API with the colored overlay which represents the number of violation occurred.

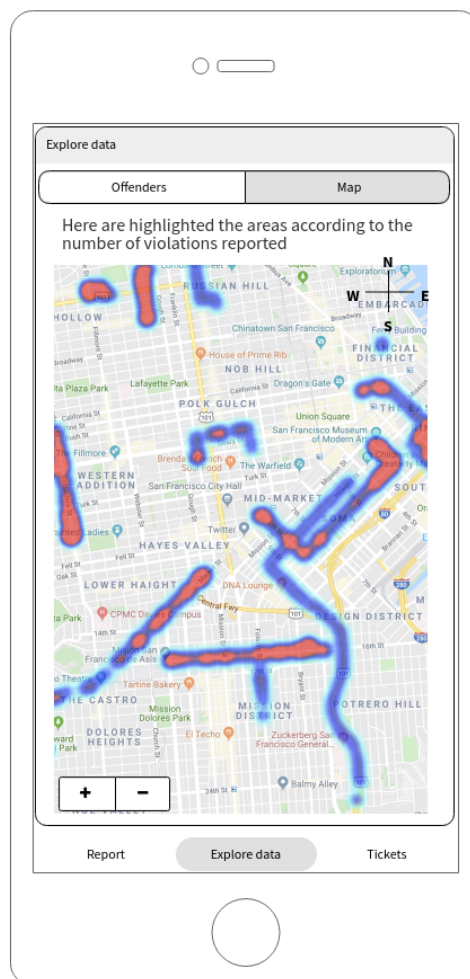


Figure 9: [GUI] Explore data | heatmap

Figure 10 shows the interface only available to Authority Users where are listed all the tickets that are pending for approval. For each ticket all the key informations are shown: picture, location, plate, code of the violation.

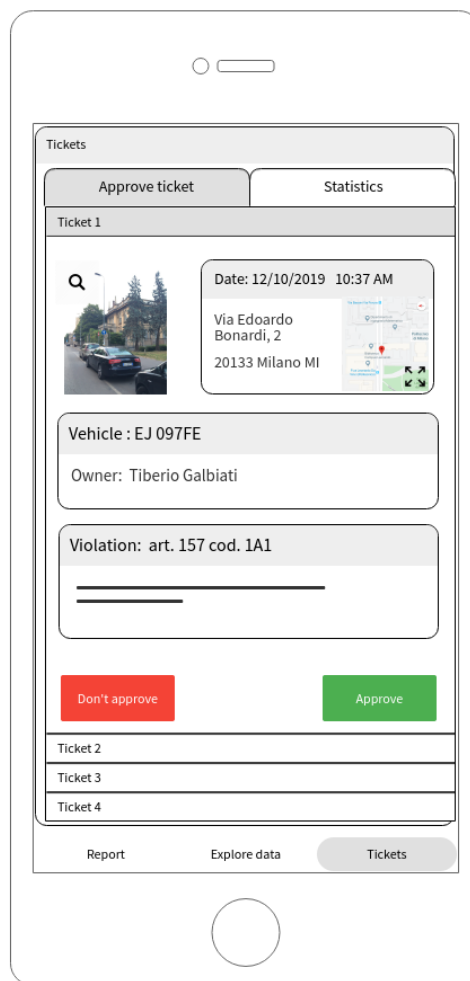


Figure 10: [GUI] Ticket | approval screen

### 3.1.2 Hardware Interfaces

there is no need to have hardware interfaces since we are developing a mobile application with a server side. Internet connection, GPS, and camera are all managed by the OS of the smartphone where the application will run.

### 3.1.3 Software Interfaces

### 3.1.4 Communication Interfaces

## 3.2 Functional Requirements

The functional requirements are those which are the fundamental actions of the system. Here we present for every goal a list of related requirements and domain assumption. Every function should work only after successful login.

- **[G1] Allow users to notify authorities about traffic violations**

[D1] Device has a working internet connection

- [D8] The number and kind of violations should be finite (defined by the law)
- [R1] User must be able to choose the kind of violation from a list
- [R2] User must be able to read detailed information about each kind of violation he can report

- **[G2] Allow users to send pictures with metadata of violations**

- [D2] Device has a camera accessible via software
- [D3] The device should acquire position with an accuracy of enough meters in order to univocally determine the road (e.g. 5 meters)
- [D4] We have access to an ALPR service which is able to read every licence plate in a picture and return each of them as a string
- [D5] ALPR service has an accuracy of more than 95%
- [D6] The device should take pictures with enough resolution to be able to read by the ALPR service
- [D7] Every vehicle that can be reported should have a licence plate visible
- [R3] Date, time and position should be automatically added to the violation reported
- [R4] We should require the user to send again a picture in case the plate is not visible
- [R5] The user must be able to select the vehicle to report in case there are other vehicles in picture
- [R6] Application must automatically determine the street name where User is

- **[G3] Allow users to mine information recorded**

- [R7] Application must be able to count occurrence of violations
- [R8] Application must be able to count violation for each vehicle
- [R9] Application should show the all the vehicles ordered with the number of violations
- [R10] Application should visualize the areas where violation occurred
- [R11] Application must use a gradient of color to show the occurrences of violations as an overlay of a interactive map

- **[G4] Have at least two different privilege for mining data**

- [D9] Every authority account is verified and it's not possible to be created using the front end
- [R12] Regular users cannot mine data about plates of the offenders
- [R13] Authority users can know the exact licence plate when mining data about offenders
- [R14] Only authority users can access the ticket approval section

- **[G5] Generate traffic tickets**

- [D10] We have access to the vehicle registration database where are stored licence plates, names and the addresses of the owners of every vehicle registered
- [D11] We have access to a database where are stored all the codes of violations and the amount of fine for the violation
- [R15] Application must be able to read every violation stored and automatically generate a ticket
- [R16] application should offer to authorities the possibility to approve tickets or not

- **[G6] Generate statistics about issued tickets**

[R17] Application must store all the tickets created

[R18] Application must read all the history of tickets created

- **[G7] Be sure every information uploaded is never altered**

[D12] The only way to upload pictures of violation is through the application

[R19] The application must be able to know if a picture has been altered

[R20] If a picture has been altered the application must automatically flag as not valid the corresponding ticket

### 3.2.1 Use Cases diagrams

Here are presented the use case diagrams for each main function. In the next section each use case will be verbally presented.

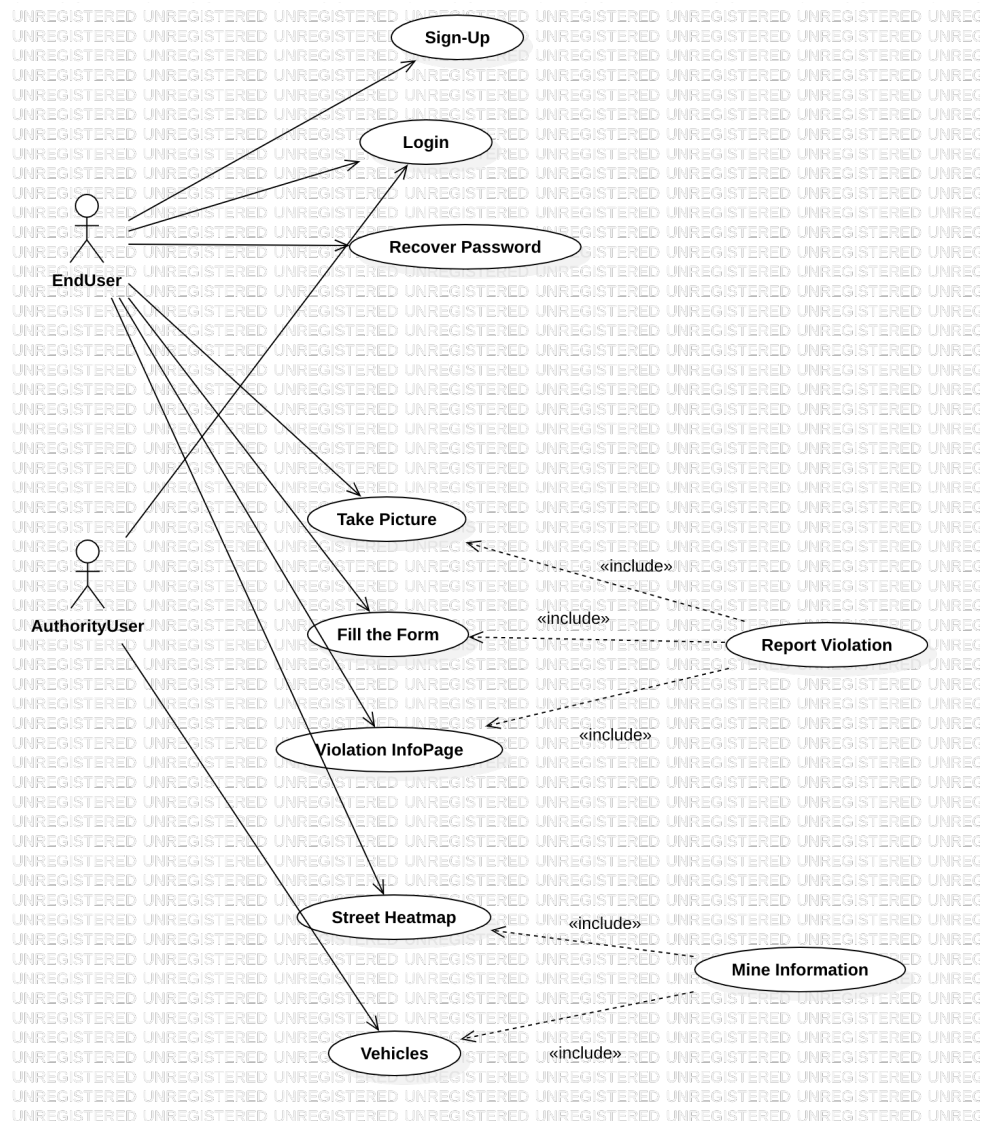


Figure 11: Basic service use case diagram

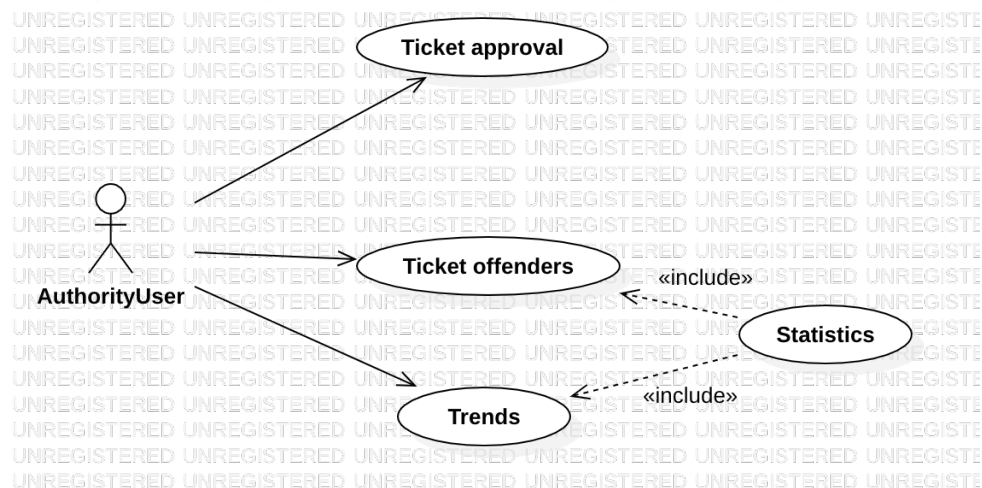


Figure 12: Advanced function use case diagram

### 3.2.2 Use Cases Description

In the following section a description of each use case is provided. For every use case is reported: an ID defining each case, the entry conditions, the steps to accomplish the exit condition and any exception that may occur.

**ID:** [UC1]

**Name:** Sign-Up

**Actor:** Guest

**Entry conditions:**

1. A citizen who wants to use the service

**Event flow:**

1. The guest reaches the registration page containing the relative form
2. The guest fills up the form and clicks on "Sign up" to complete the process
3. The system redirects the user to his profile page and sends a confirmation email

**Exit conditions:**

- The guest has successfully registered in the system

**Exceptions:**

1. The guest left an empty field or typed something wrong an error message is displayed and the user is asked to fill the form again.

---

**ID:** [UC2]

**Name:** Login

**Actor:** User

**Entry conditions:**

1. The user has already registered

**Event flow:**



1. The user reaches the login page containing the relative form
2. The user types the username and password in the login form and click on "Login" button
3. The system redirects the user to the application homepage

**Exit conditions:**

- The user has access to the application functionalities

**Exceptions:**

1. Username and password didn't correspond or the username didn't exist so an error message is displayed and the user is asked to fill the login form again

---

**ID:** [UC3]

**Name:** Recover Password

**Actor:** User

**Entry conditions:**

1. The user has already registered

**Event flow:**

1. The user reaches the login page containing the relative form
2. The user clicks on "Password recovery" button and is redirected to the password recovery page.
3. The user inserts his email and clicks on "Reset password"
4. The system sends an email to the user with a link and instruction to reset the password
5. The user chooses and types a new password and confirms
6. The application check whether the entered password is strong enough or not
7. The system redirects the user to the login page

**Exit conditions:**

- The user has changed his password

**Exceptions:**

1. The inserted email doesn't match any user in the database so it is displayed an error message and the user is asked to retype a valid email

---

**ID:** [UC4a]

**Name:** Report a violation - taking picture

**Actor:** User

**Entry conditions:**

1. User is logged in

**Event flow:**

1. User enters the section "Report a violation"

2. System opens the camera of smartphone and ask user to take a picture of the violation
3. The system reminds the user: the violation and the licence plate of the vehicle which is in violation must be visible
4. The user takes the picture
5. The system shows the picture just taken
6. The system asks the user: if there are other plates visible in the picture, which are not the one of the vehicle to be reported, use the finger to delete them
7. The system shows "brush tool mode" and the user can cover the other licence plates with his finger. User can skip this part if there are no other plates
8. When done, user press continue button.
9. The system sends the picture to the ALPR service which returns the string containing the plate decoded. User just waits
10. The system shows now on the screen the "Report violation form"

**Exit conditions:**

1. User must continue to next [UC4b]

**Exceptions:**

1. If no plate is found, the user has to repeat this use case, starting from taking the picture again
2. If the ALPR service returns more than one plate, the user is informed that must delete the no required plates and the system goes again to the "brush tool mode"
3. If user doesn't continue to the next use case: e.g. presses exit button, or closes the app for more than 10 minutes, the picture taken is discarded

---

**ID:** [UC4b]

**Name:** Report a violation - fill the form

**Actor:** User

**Entry conditions:**

1. User has successfully completed the precedent [UC4a]
2. User is in the fill-form section of the app

**Event flow:**

1. The system sends GPS location to the external service to get the complete address of the user
2. The form is pre-filled with the address that is given by the external service
3. The user must choose from a list of violations the one referred to the picture taken which wants to report. In the UI every row contains the name of the violation and a "info" button
4. The user choose to send the form

**Exit conditions:**

1. The violation is correctly inserted and stored

**Exceptions:**

1. If user doesn't send the form: e.g. presses exit button, or closes the app for more than 10 minutes, the violation reported is discarded

---

**ID:** [UC4b1]

**Name:** Report a violation - fill the form - violation info page

**Actor:** User

**Entry conditions:**

1. User is in Use case [UC4b]
2. User has pressed the "info" button of a violation from the list

**Event flow:**

1. System shows a brief description of the selected violation

**Exit conditions:**

1. User goes back to Use case [UC4b]

**Exceptions:**

---

**ID:** [UC5a]

**Name:** Mine information - street heatmap

**Actor:** User

**Entry conditions:**

1. User is logged in

**Event flow:**

1. User enters the section "Explore data"
2. The user chooses to get the map about streets with the highest frequency of violations
3. The system retrieves data from the database of violations, counting for each street the number of occurrences
4. The system sends to the external maps API the count of violation and the road name/coordinates
5. The app shows the map with an overlay which highlights the areas with a gradient color according to the number of violations occurred

**Exit conditions:**

User wants to go back to "Explore data" area **Exceptions:**

1. If there are no records the app will report no data available message

---

**ID:** [UC5b]

**Name:** Mine information - offenders

**Actor:** AuthorityUser

**Entry conditions:**

1. AuthorityUser is logged in

**Event flow:**

1. AuthorityUser enters the section "Explore data"
2. The system asks which kind of data the AuthorityUser wants to know
3. The AuthorityUser chooses to get the data about vehicles that committed the highest number of violations
4. The system queries the table where for each licence plate is associated the count of violations
5. The system will report in a tabular way the plate of the vehicle and the count of violations committed
6. If the AuthorityUser scrolls down, the system will offer the chance to load more rows

**Exit conditions:**

1. AuthorityUser wants to go back to "Explore data" area

**Exceptions:**

1. If there are no records the app will report no data available message

---

Advanced function

**ID:** [UC6]

**Name:** Ticket approval

**Actor:** AuthorityUser

**Entry conditions:**

1. A new violation is inserted in database
2. AuthorityUser logged in

**Event flow:**

1. Every time a new violation is created by a EndUser the system will create automatically a ticket to be approved
2. AuthorityUser enters the section "Tickets"
3. AuthorityUser enters the section "Approve Tickets"
4. The System will show the list of tickets available for approval
5. AuthorityUser selects one ticket and system will show the related details
6. System will ask the AuthorityUser if he wants to approve or not the ticket

**Exit conditions:**

1. User wants to go back to "Ticket" area
2. AuthorityUser approves the ticket
3. AuthorityUser doesn't approve the ticket

**Exceptions:**

1. If there aren't tickets pending, the app will report a message saying that there aren't new tickets to approve

---

**ID:** [UC]

**Name:** Statistics - tickets offenders

**Actor:** AuthorityUser

**Entry conditions:**

1. AuthorityUser logged in

**Event flow:**

1. AuthorityUser enters the section "ticket statistics"
2. The system will show the available ticket statistics options available to show
3. AuthorityUser selects to see the statistics about offenders
4. The system queries the table about all tickets, getting the count of tickets associated to every citizen present in the database of ticket created
5. The system will report in a tabular way the name of the citizen and the count of approved tickets he has received
6. If the AuthorityUser scrolls down, the system will offer the chance to load more rows

**Exit conditions:**

1. The AuthorityUser wants to go back to other sections

**Exceptions:**

---

**ID:** [UC5]

**Name:** Statistics - trends

**Actor:** AuthorityUser

**Entry conditions:**

1. AuthorityUser logged in

**Event flow:**

1. AuthorityUser enters the section "ticket statistics"
2. The system will show the available ticket statistics options available to show
3. AuthorityUser has chosen to see the Statistics - trend option
4. The system shows the count of the approved tickets of last month

**Exit conditions:**

1. the AuthorityUser wants to go back to other sections

**Exceptions:**

---

### 3.2.3 Sequence diagrams

In this subsection are shown the sequence diagrams which shows the interactions between the user, our software and the external APIs.

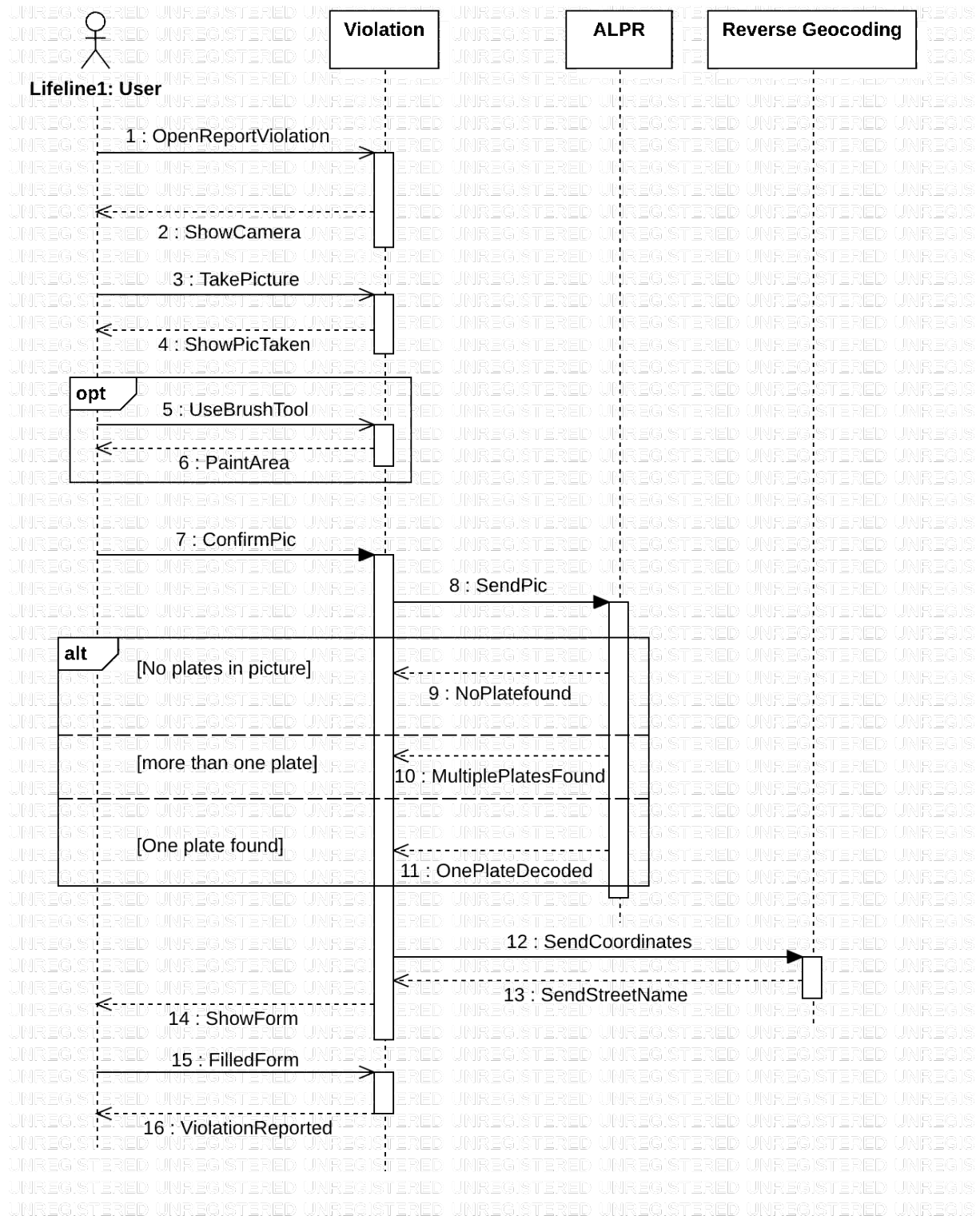


Figure 13: Sequence Diagram for Violation Reporting

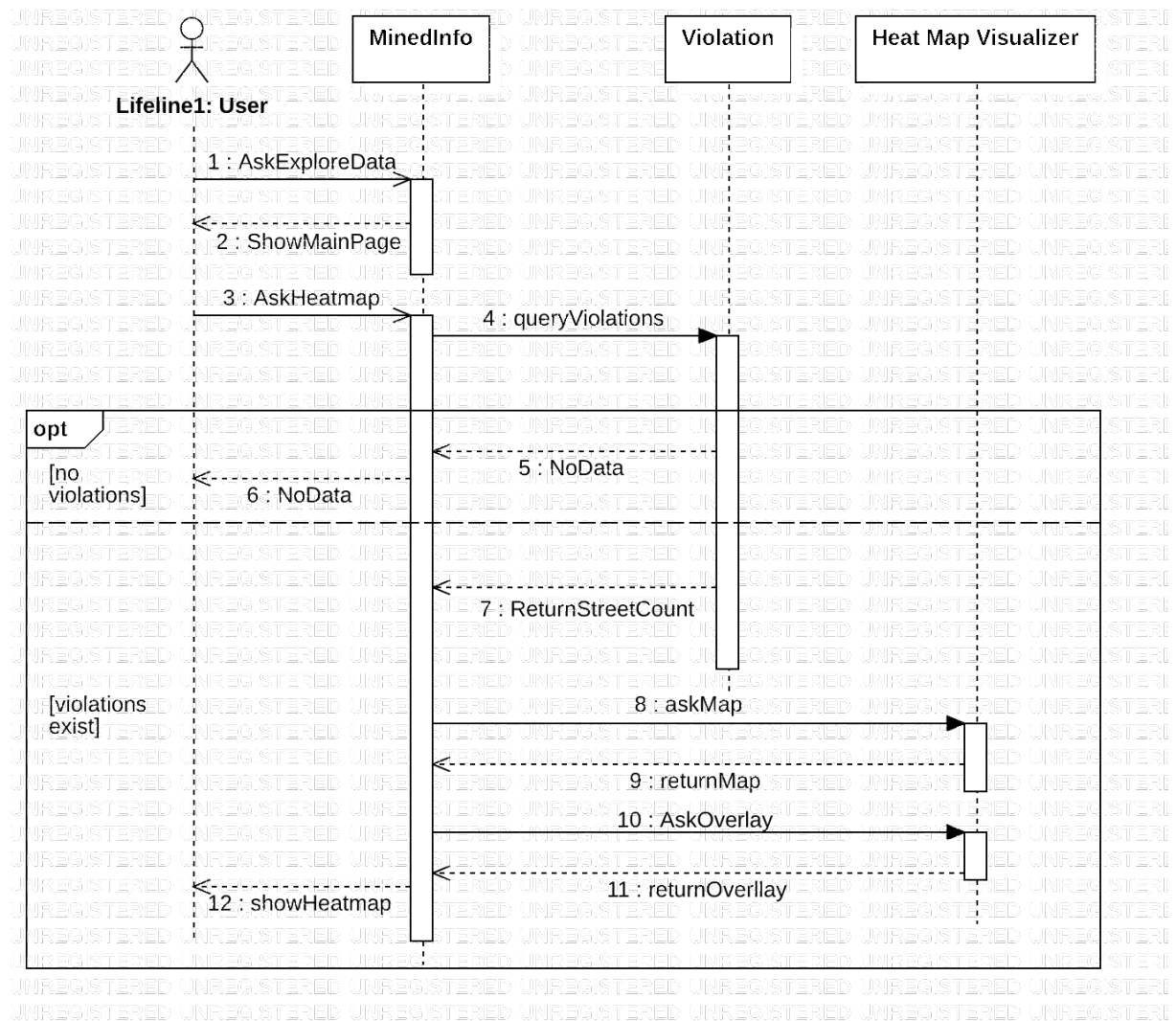


Figure 14: Sequence Diagram for HeatMap visualization



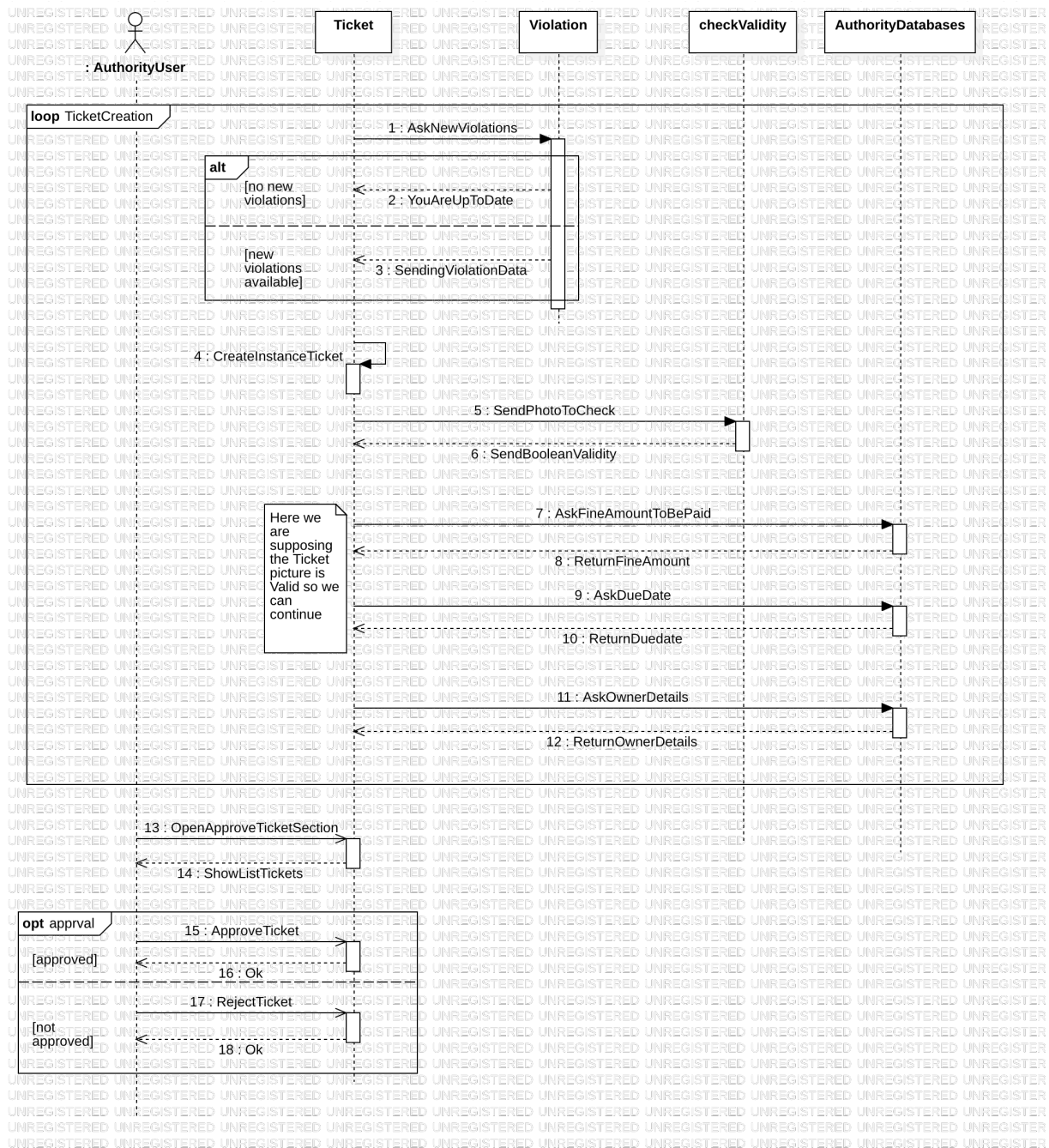


Figure 15: Sequence Diagram for ticket Creation and approval

### 3.2.4 Requirements traceability matrix

In the following table we have listed for each requirement all the use cases related.



Requirement	Use Case
[R1]	[UC4b]
[R2]	[UC4b1]
[R3]	[UC4b]
[R4]	[UC4a]
[R5]	[UC4a]
[R6]	[UC4b]
[R7]	[UC5a]
[R8]	[UC5b]
[R9]	[UC5b]
[R10]	[UC5a]
[R11]	[UC5a]
[R12]	[UC5b]
[R13]	[UC5b]
[R14]	[UC6]
[R15]	[UC6]
[R16]	[UC6]
[R17]	[UC7]
[R18]	[UC5a]
[R19]	[UC5a]
[R20]	[UC5a]

Table 2: Traceability matrix

### 3.3 Performance Requirements

The system must be able to work with simultaneous requests, we can estimate in the worst case more than 10 reported violation at time if we consider a city of more than half a million of inhabitants.

### 3.4 Design Constraints

#### 3.4.1 Standards compliance

The app should be available for the two main operating systems of smartphones: Android Os and Apple iOS.

#### 3.4.2 Hardware limitations

The app will have a server side and a client side (smartphone). On server side limitations can be the size of available storage and the bandwidth. On smartphone side we have the network connectivity (3G/4G connection) and GPS limitations in some areas (very rare case since we will use the app in urban environment).

#### 3.4.3 Any other constraint

- Application should be compliant to European GDPR for the EndUsers
- The traffic violations which can be reported should be compliant to the local traffic code where the app will be used

- For an use in Italy the app should be compliant to the "Codice della Strada" [2], in particular parking violations are reported in Art. 157. The tickets created should have the same structure as required by the law

## **3.5 Software System Attributes**

### **3.5.1 Simple User Interface**

The user interface has to be as simple and intuitive as possible, the application should allow an average user to set up an account and start using the application understanding its functionality in no more than a dozen minutes. In addition there should be a complete tutorial to makes it easy using the application.

### **3.5.2 Reliability**

The application provides a reliable service in which individual users can easily log in and report the violations in the most optimal way. Furthermore it warranties that the chain of custody of the information coming from the users is never broken, and the information is never altered. This would provide a secure and reliable system.

### **3.5.3 Availability**

The application must offer availability in the order of 99.99% granting its service every day at any time (24/7). The lack of service must be minimal. Reporting violation and taking the information about the violation coming from SafeStreets must be active every day at any time. The lack of service is acceptable only if it is due to maintenance. In this case, users must receive a warning 48 hours before.

### **3.5.4 Security**

It's important that the Authority Users cannot be created using a public sign-up form. Those special accounts must be created with the backend on the application. There should be no breaks that would make possible to add pictures and violations not using the built-in camera via the app, for example pictures taken with a different device or already stored in the mobile phone shouldn't be accepted. Moreover, the means of communication must be encrypted to save the confidentiality of information sent and generated, especially the one related to the issue of tickets.

### **3.5.5 Maintainability**

The application must keep a service log in order to fix bugs more easily. The app should be developed using a modular approach, so adding new functions shouldn't require to change the previous code, bust just add the new module. An example could be adding more statistic functions in the Trend section of issued tickets or violations.

### **3.5.6 Portability**

Portability of user data from a device to another is possible by entering personal login data. Also the application will be able to run for devices with different operating systems.

## 4 Formal Analysis Using Alloy

Here's the complete code for our Alloy model. The .als file can be found in our GitHub repository. For clarity we separate the code in Signatures, Facts and Asserts and Predicates.

### 4.1 Abstract Entity and Signature

```
open util/boolean
//sig string {}

abstract sig User {

    //name: one string,
    //surname: one string,
    // address: one string,
    // email: one string,
    //password: one string,
    accessLevel: one Bool,

    minedInfo: one MinedInfo,
}

sig EndUser extends User{

    userLocation: one Location,
}{}

accessLevel = False

}

sig Authority extends User{

    tickets: Ticket,
}{}

#tickets ≥ 1
accessLevel = True

}

sig Location {

    latitude: Int ,
    longitude: Int

}
```

```

sig Photo {}

sig Violation {
    location: one Location,
    addr: one ReverseGioCoding,
    reporter: one EndUser,
    //type: one string,
    photo: one Photo,
    alpr: one ALPR,
    date: one Date
}

sig ReverseGioCoding {
    loc: Location,
    //addr: one string
}{#ReverseGioCoding = 1}

sig ALPR { //remember to add somethin to tell it's only one
    picture: some Photo ,
    // licenseP: one string
}{ #ALPR = 1}

sig Date {}

abstract sig MinedInfo {
    violations: set Violation,
}

sig MinedStreet extends MinedInfo{
/*
    // name: some string,
    frequency: some Int,
    location : one Location,*/
}

```

```

sig MinedOffender extends MinedInfo{
  /*
    n_Violations: one Int,
    licensePlate: one Int,
    uuid: one Int*/
}

sig Ticket {

    violations: set Violation,

}

fact NoSameGPSForDifferentUsers {
    no disjoint u1, u2 : EndUser |
    u1.userLocation = u2.userLocation
}

fact NoSameGPSForDifferentReverseGio {
    no disjoint revGio1, revGio2: ReverseGioCoding |
    revGio1.loc = revGio2.loc
}

fact NoSamePhotoForDifferentViolation {
    no disjoint v1,v2 : Violation |
    v1.photo = v2.photo
}

fact NoSameViolationForDiffReporter{
    no disjoint v1, v2 : Violation |
    v1.reporter = v2.reporter
}

/*
fact NoSameTicketForDiffAu {
    no disjoint a1, a2 : Authority/
    a1.tickets = a2.tickets
}*/

fact { #Ticket ≥ #Authority}

fact { all t: Ticket | one au: Authority | au.tickets = t}

```

```

fact { some user: User | user.accessLevel = False and user.
  ↪ minedInfo = MinedStreet}

fact { all mined: MinedInfo | one us: User | us.minedInfo = mined}

fact { all ph: Photo, alpr: ALPR | alpr.picture = ph}

fact {all revGio: ReverseGioCoding, u: EndUser | revGio.loc = u.
  ↪ userLocation}

fact {all loc: Location |one viol: Violation | loc = viol.location}

fact {all d: Date, viol: Violation | viol.date = d}

fact {all t: Ticket , v:Violation | t.violations = v}

pred show {

  //one Location

}

run show for 5

```

## **5 Effort Spent**