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

Download page: https://github.com/TiberioG/GalbiatiRezaei.git

November 10, 2019

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This is the Requirement Analysis and Specification Document (RASD) of SafeStreet application. 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 to authorities pictures of violations, including their date, time and position. Examples of violations are: vehicles parked in the middle of bike lanes, in places reserved for people with disabilities, on foothpats, double parking etc.

SafeStreets stores the information provided by users, completing it with suitable meta-data every time it recieves a picture. In paricular it is able to read automatically the license plate of a vehicle and store it without asking the user to type it. 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, rereiving it automatically from the geographical position where the user took the picture. Then the application allows both end users and authorities to mine the information crowd-souced. Two visualizations are offfered: the first is an interactive map where are high-lighted 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-soucred 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 priviledge 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

1.2.1 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

1.3 Definitions, acronyms, abbreviations

1.3.1 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.
- 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

1.3.2 Acronyms

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

1.3.3 Abbreviations

1.4 Revision history

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

1.5 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

1.6 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 imposssible get a precise address 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. If we made possible to add more pictures then there would be the need to chech if all the pictures are reporting the same vehicle for the same violation, but just from different perspectives. We believe that with only one picture is still possible to determine the 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 **MinedSteets** 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 highliting the spots where violation occurred.

Ticket This class is used to represent the ticket with the fine for the owners of violationg vehicles. Every instance will be automatically creadted 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, like the database of all registered vehicles plates and the database of violation of the traffic code. The attribute *valid* is a boolean value, set by the class **CheckValidity** TRUE if the picture associated to the correspinding 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*, *dateviolation* are copied from the instnces of **Violation**. The attributes

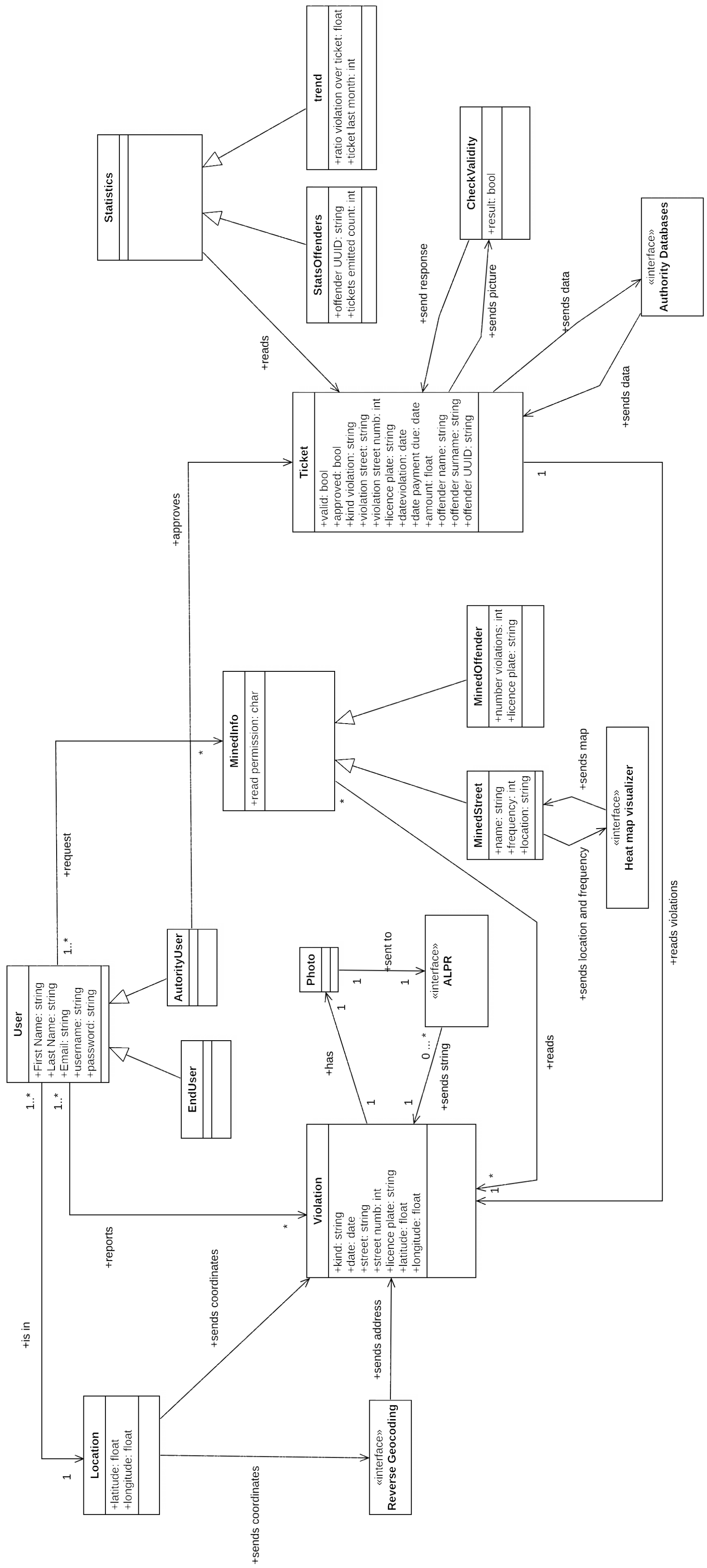

Figure 1: High-level Class Diagram

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 vehicle, 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 occour. User will be required to take a picture of the vehicle responsible of the violation ad select the kind of violation. Once opened, the app the 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 those plates using his finger. In case no plates are found, the system will ask the user to take again the picture. After the picture has succesfully 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 ad submit it.

2.2.2 Explore Data

The app will offer the possibility to the users to visualize the data collected. Two kind 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 Authority Users will have access to the table of plates which committed the most violations.

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:

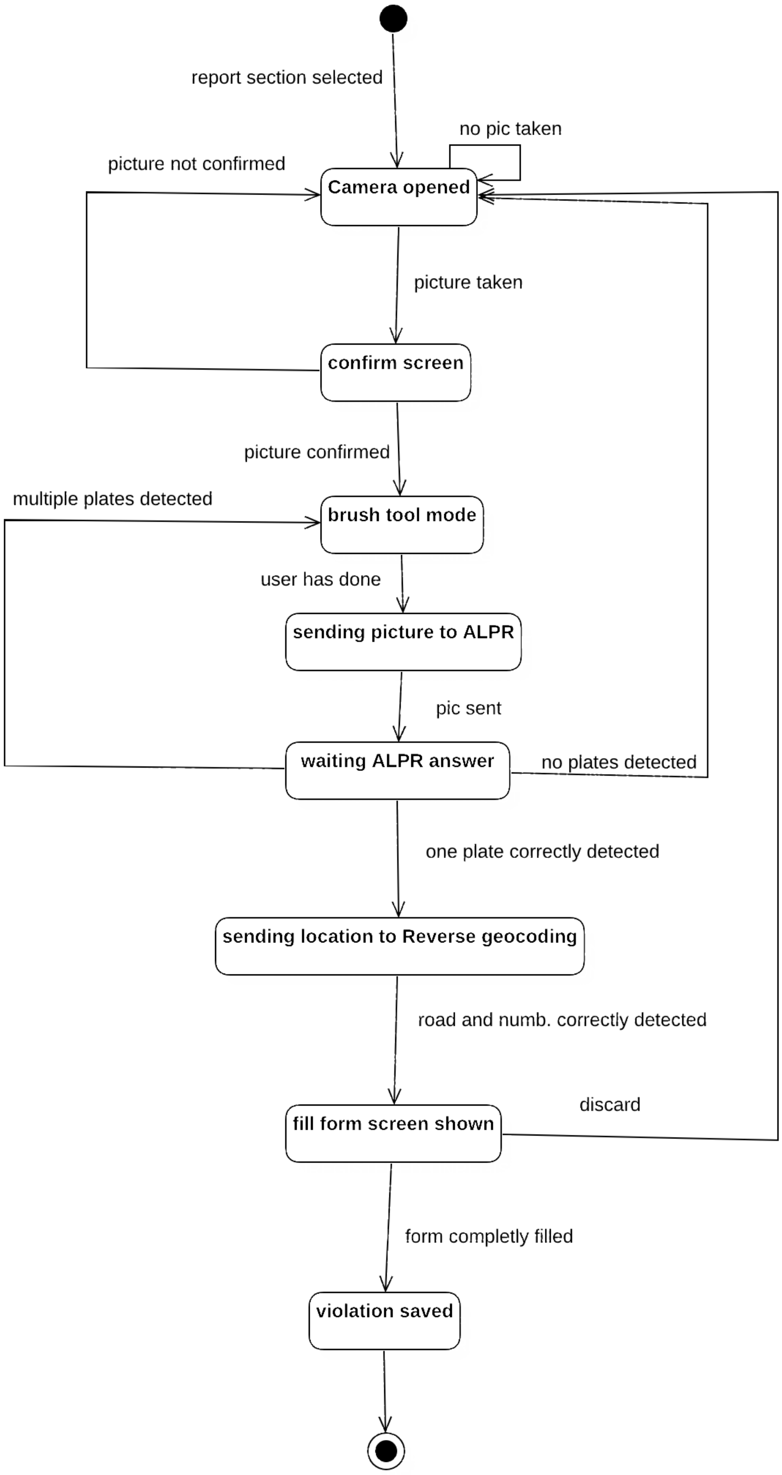

Figure 2: Violation reporting state diagram

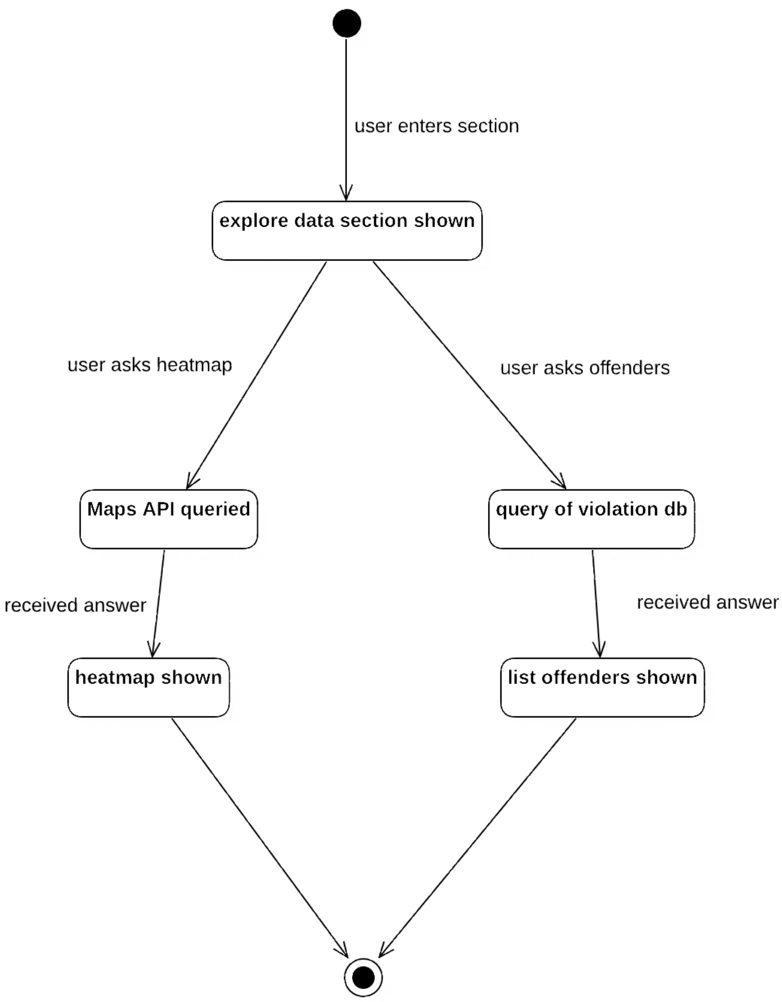

Figure 3: Mining information state diagram

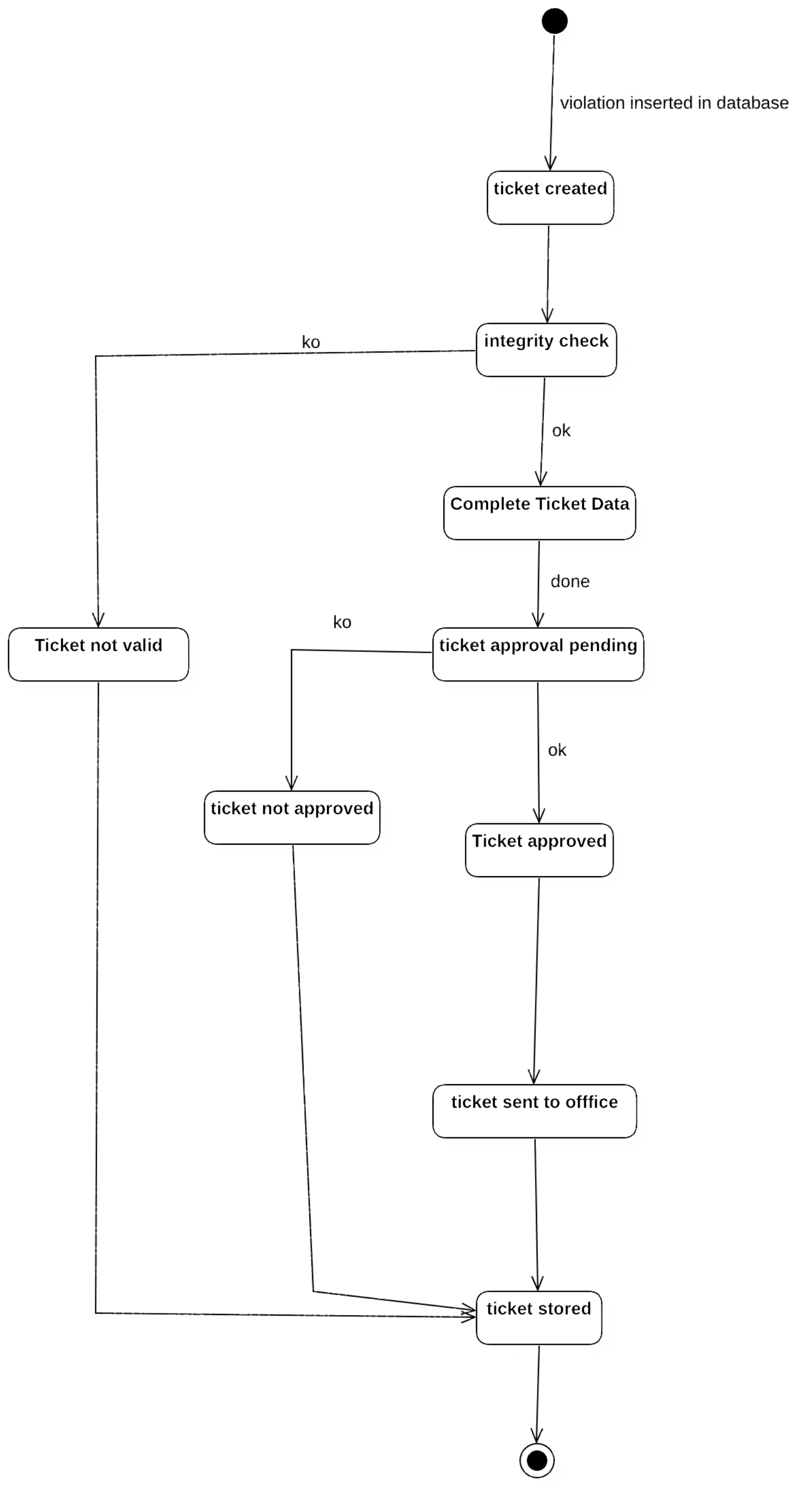

Figure 4: Tickets creation and approval state diagram

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

An external service or a piece of software 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, which means the picture has been modified, the ticket is stored as *not valid* for debug purposes. Examples of possible uses of those not-valid tickets 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 a 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. Maybe in the future tickets will be sent via a Certified E-Mail.

2.2.4 Ticket statistics

This function has the purpose to show some statistics about the issued tickets. Here we will take into account only valid and approved tickets.

Two kind of visualizations are offered:

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

Both visualizations 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 ”vehicles that committed the highest number of violations” 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 vehicles, here we are counting all the tickets emitted for all of his vehicles.

2.3 User characteristics

Here we focus on the needs of our users:

For End Users we want to offer a easy and fast way to report violations which must be discrete and effective. For Authority users we want to offer a service that will help to create tickets, making this process easier. Moreover the ststistics can be used by the municipality to know where there is need to buld more parking areas or improve the local public transport.

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 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 them with 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. Each vehicle has one and only one main owner

[D11] We have access to a database where are stored all the codes of violations and the amount of fine for the violation. Every violation has one and only one amount of fine required

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

[D13] Each licence plate is unique, there are no vehicles with the same plate

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 app will be used "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 vehicles 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 with 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 5 shows the login page where it's possible to access the registration form for new users.

After login the user will enter by default into the first tab "Report", see Figure 6, where is shown what the camera is recording so 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.

After the picture has been taken, the apps shows it as in Figure 7. 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 8 shows how appears the form where the user has to select the kind of violation he is rporting from a scroll-down list. Each row has an info button, when pressed the GUI will show a verbal description of the violation.

Figure 9 shows the view of the plates which have committed the highest number of violations.

Figure 10 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 11 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

subparagraph name own: picture, location, plate, code of the violation.

Figure 5: [GUI] Login screen

Figure 6: [GUI] Report | open camera view

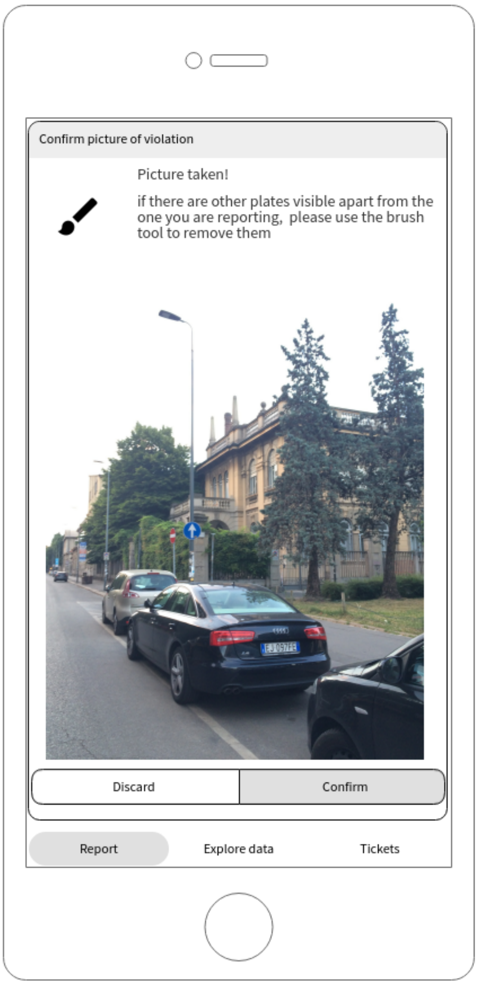

Figure 7: [GUI] Report | picture of violation taken screen

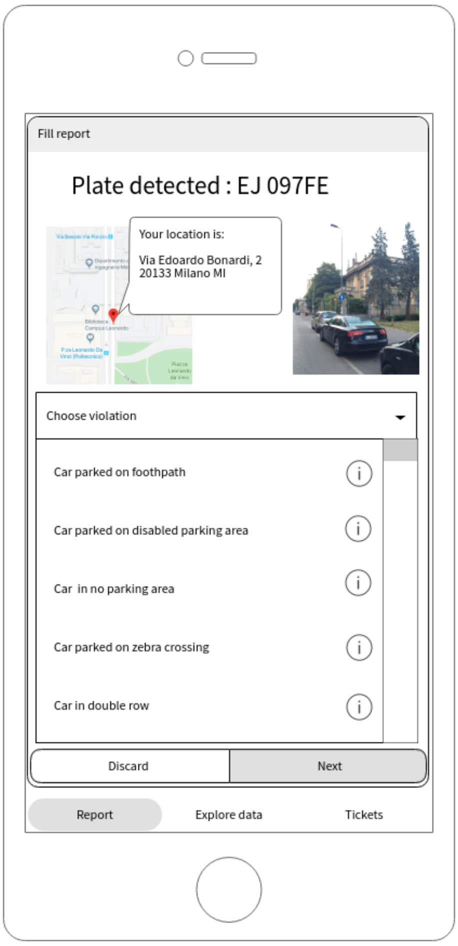

Figure 8: [GUI] Report | violation info form screen

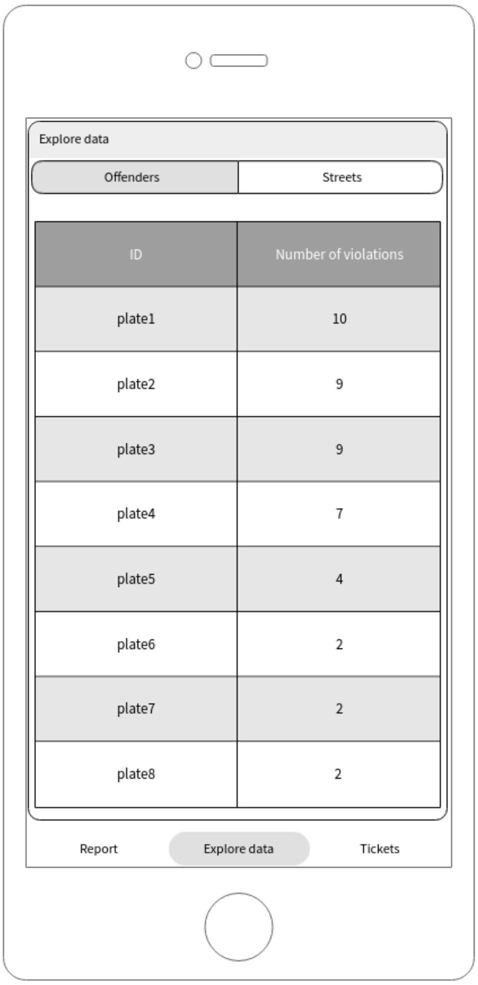

Figure 9: [GUI] Explore data screen | offenders

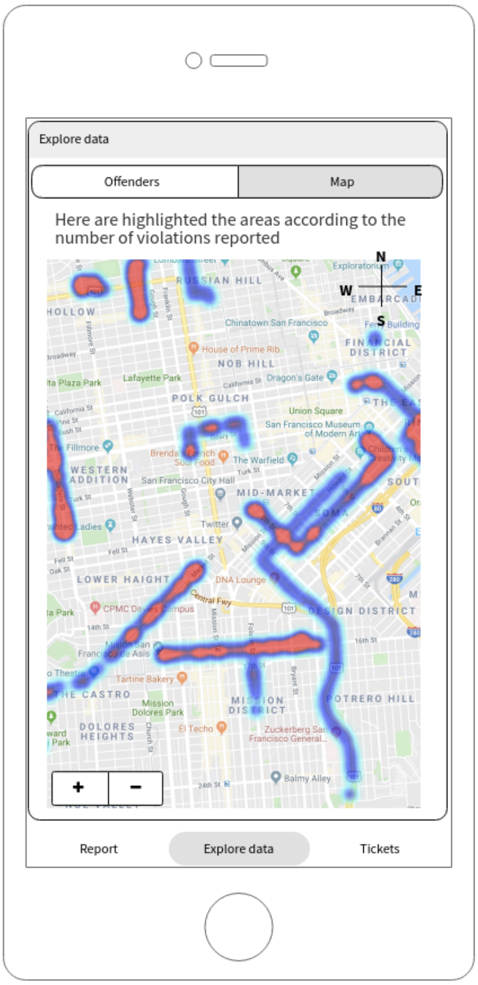

Figure 10: [GUI] Explore data | heatmap

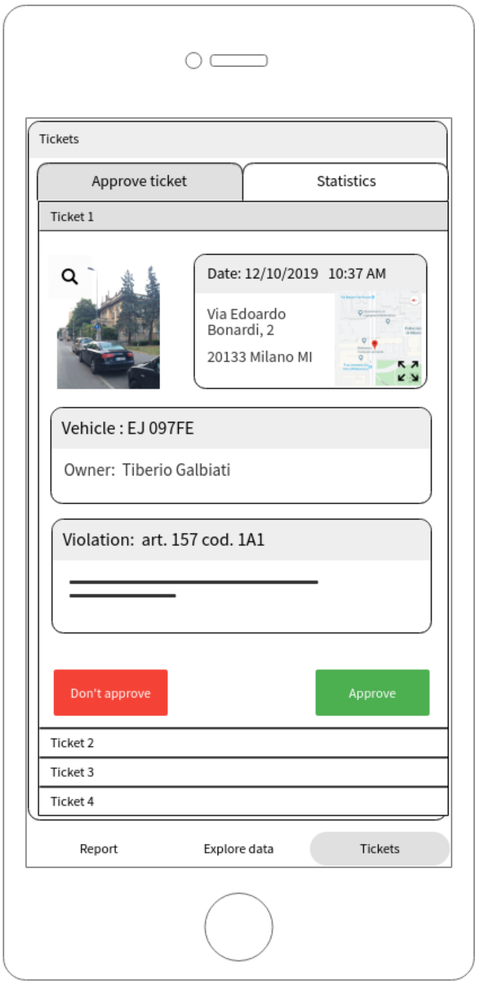

Figure 11: [GUI] Ticket | approval screen

3.1.2 Hardware Interfaces

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

3.1.3 Software Interfaces

Our system will offer a mobile app which works togheter with a server.

3.1.4 Communication Interfaces

The communication between different parts of the system is important since they depend on each other. However, the way communication is achieved is not the key focus of this specific application beacuse we can use some already existing frameworks.

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 shoud work only after succesful 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 them with 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 occurrency of violations

  [R8] Application must be able to count violation for each vehicle

  [R9] Application should show 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 occurrencies 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. Each vehicle has one and only one main owner

[D11] We have access to a database where are stored all the codes of violations and the amount of fine for the violation. Every violation has one and only one amount of fine required

[D13] Each licence plate is unique, there are no vehicles with the same plate

[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 applcation 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.

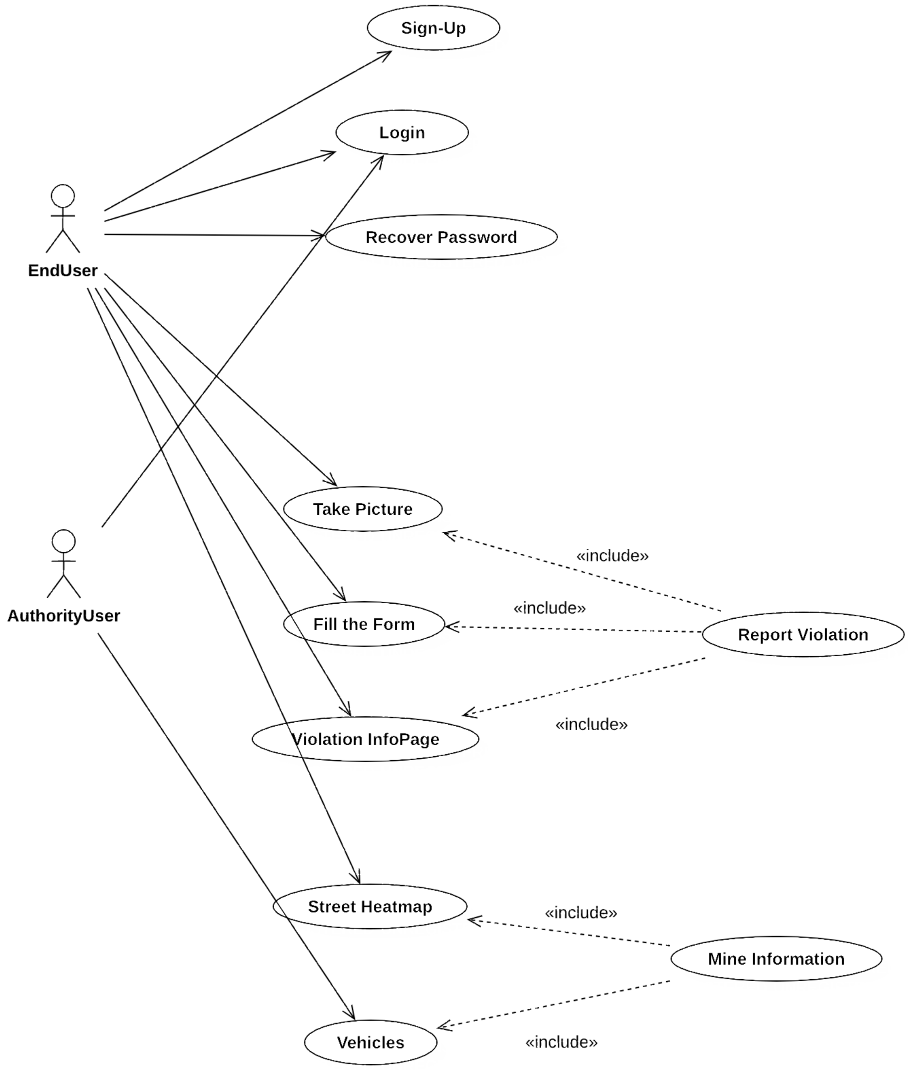

Figure 12: Basic service use case diagram

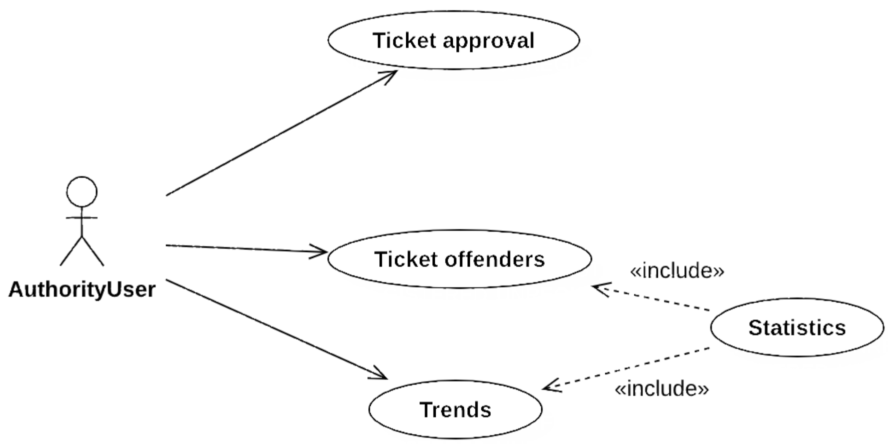

Figure 13: Advanced function use case diagram

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. So 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 clicks 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 whith 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 not 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 infopage
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 decriprion 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 occurrencies

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 higlights the areas with a gradient color according to the number of violations occurred

Exit conditions:

3.2.3 Sequence diagrams

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

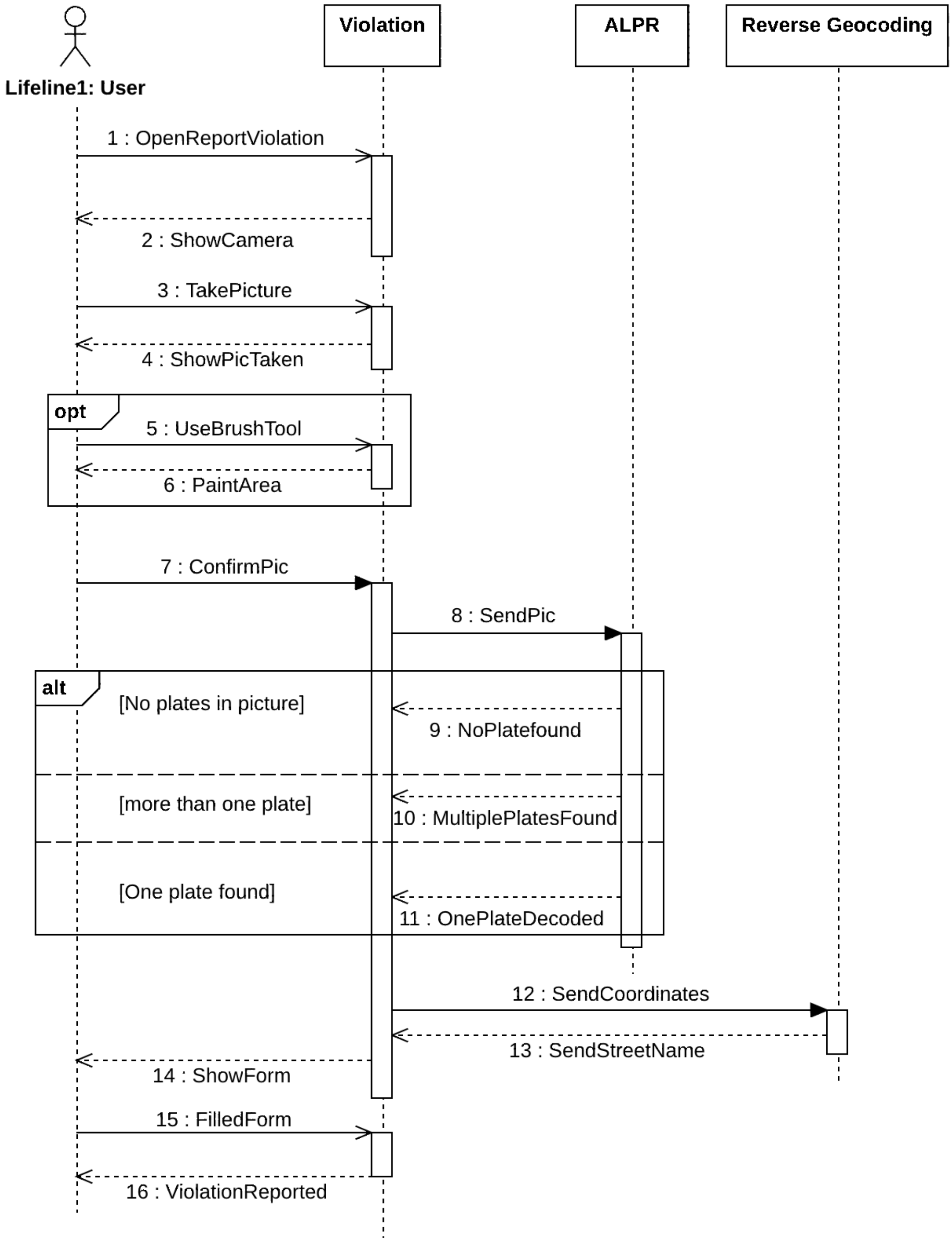

Figure 14: Sequence Diagram for Violation Reporting

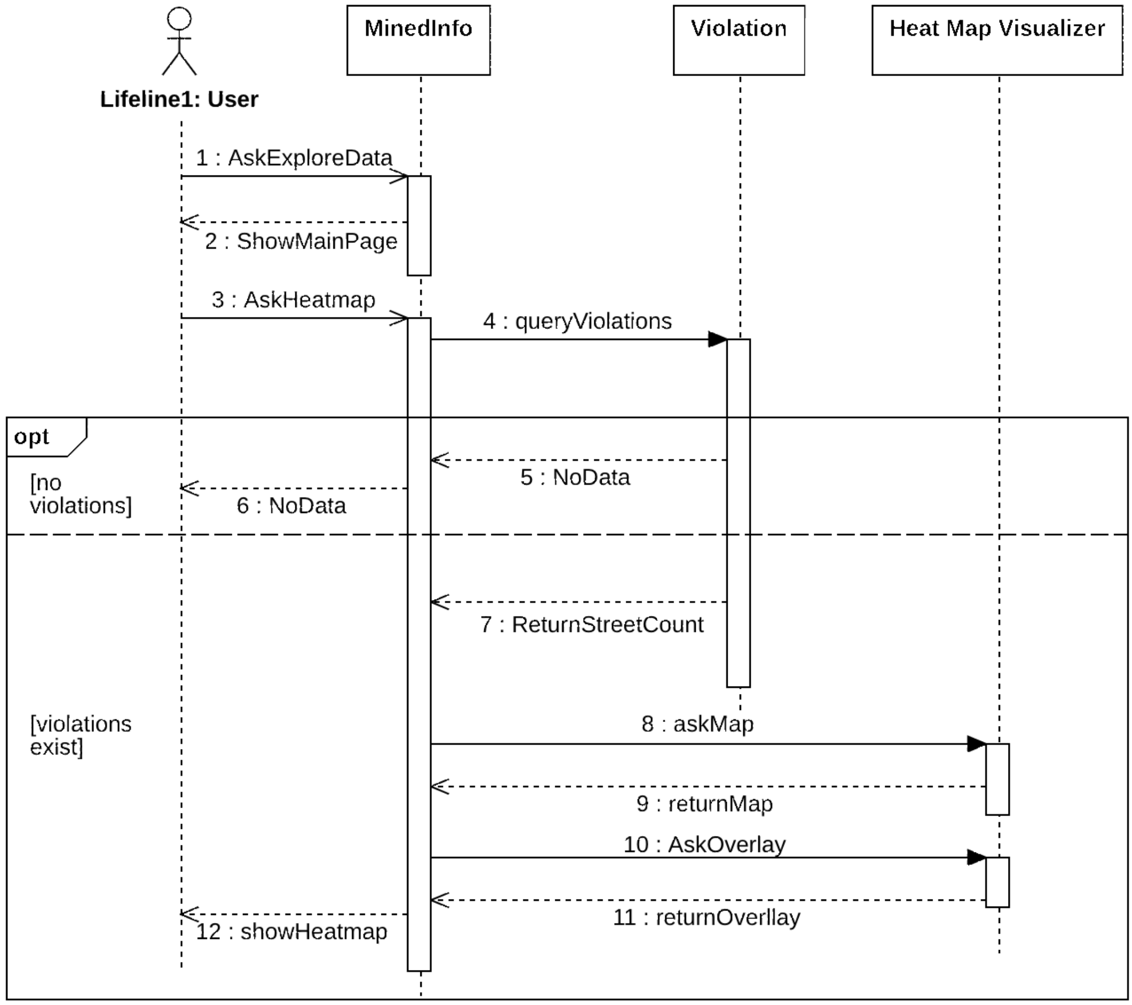

Figure 15: Sequence Diagram for HeatMap visualization

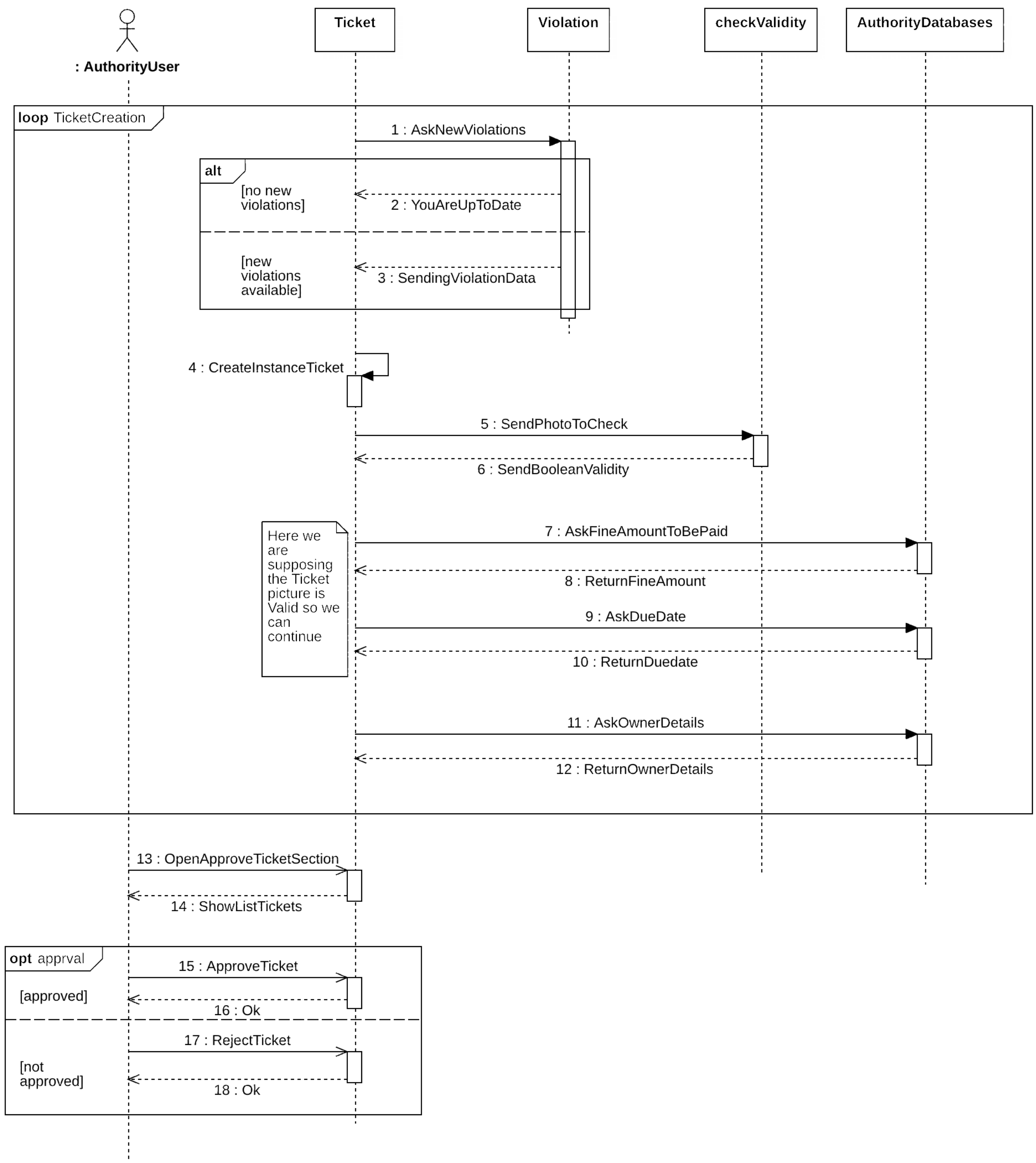

Figure 16: Sequence Diagram for ticket Creation and approval

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 shoudld 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 shoud 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 make 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 services 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 should’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 prevous 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 {}

sig Name, Surname{}
//sig Email, Password{}


abstract sig User {

//  name: one Name,
//  surname:  one Surname,

// email: one Email,
 // password: one Password,
  accessLevel: one Bool,

        minedInfo: some MinedInfo,
}

sig EndUser extends User{

 userLocation: one Location,

}{

accessLevel = False

}

sig Authority extends User{

tickets:  Ticket,

}{

#tickets ≥ 1
accessLevel = True

}

sig Location {

  latitude: one Int ,
```

```
  longitude:one Int

  }



sig Photo {}

sig Violation {

        location: one Location,
        addr: some ReverseGioCoding,
         reporter: one EndUser,
  //type: one string,
         photo: one Photo,
         licensePlate: one ALPR,
  //     date: one Date

}

sig ReverseGioCoding {

  loc: some 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: some 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: one 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 EachTicketOneAuthority {
        all t: Ticket | one au: Authority |
        au.tickets = t
}
```

```
//Two dierent users 'cant have the same email
/*fact NoSameEmailForDierentUsers {
no disjoint u1, u2 : User |
u1.email = u2.email
}*/

fact NoSameMinedInfoForDierentUsers {

all disjoint u1,u2: User |      u1.minedInfo ≠ u2.minedInfo

/*      no disjoint u1,u2: User |
        u1.minedInfo = u2.minedInfo
*/
}


fact EqualUserAndLocation{
#EndUser = #Location
}

//fact {all u: User | some n: Name | u.name = n}
//fact {all u: User | all n: Surname | u.surname = n}

fact EndUserRelateOnlyMinedStreet {
         all user: EndUser |
         user.minedInfo = MinedStreet
}


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

//fact {  all us: User | some mined: MinedInfo | us.minedInfo = mined
   ↪ }

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


////
fact EqualLocationForEndUserAndGio  {
        one revGio: ReverseGioCoding|
        one u: EndUser |
         revGio.loc = u.userLocation
}

fact EqualLocationForViolationAndEndUser {
        all viol: Violation |
```

```
        some u:EndUser|
         viol.location = u.userLocation
}

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

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

pred show {
}
run show for 4
```


5 Effort Spent

Tiberio	
task	**Time**
first meeting	1h 30min
abbott analysis	45 min
second meeting	2h
class diagram	1h
requirements	1h
third meeting	1h 30 min
use cases and class diagrams	2h 30 min
meeting	30 min
use cases	2h 30 min
GUI, requirements, class description	4h
GUI, functions, statecharts	4h
ucases traciability matrix	2h
introduction, world and machine	2h 30 min
sequence diagrams	2h
meeting alloy	1h 30 min
sequence and fixes	1h 30 min
check and fix everithing	6h
Total	**x h**

