Politecnico di Milano

AA 2019-2020

Computer Science and Engineering

Software Engineering 2 Project

RASD

Requirement Analysis and Specification Document

version 1.0 - 10/11/2019

Cem Uyuk - 942341
Sidem Isil Tuncer - 944006
Ali Ozkaya - 943637

Table of Contents

1. Introduction

1.1. Revision History

- Version 1.0 (10 November 2019)
- Version 1.1 (9 December 2019)
    - Goals, Requirements and Use Cases Updated: Login situation and system manager removed.
- Version 1.2 (15 December 2019)
    - Class Diagram updated, the rest of the document changed to keep consistency
    - Alloy Model reimplemented

1.2. Purpose

SafeStreets is a creative initiative that aims to enable civilians to notify authorities about the traffic violations that occur frequently around them. The purpose of this initiative is, firstly, to encourage civilians to submit traffic violations through a mobile application by taking a photo and, secondly, to allow authorities to create tickets for violations in an easier way. Through the application, each registered user will be able to submit a violation and validate the correctness of other violation submissions. Thanks to the chain of information flow and the validation process completed by users, the authorities will be able to create tickets for submitted violations. The violation data gathered by the application will be available for users to view in different formats.

In order to delve deeper into the details of this application, these documents represents the Requirements Analysis and Specification Document (RASD). The goal of this document is to fully explain the system in terms of functional and non-functional requirements, goals, consider the limitations and indicate use cases both verbally and in diagrams to address the software developers that have to transfer the real-world intuition to the SafeStreets software.

1.3. Scope

1.3.1. Description of the Problem

SafeStreets is a crowd-sourced parking violation reporting service to be which aims to notify authorities about traffic violations. The service should also detect traffic violation hubs and offer possible solutions. The aim is to design an application to facilitate the digital needs for the service which includes gathering crowd-sourced and authority provided data, and sharing the relevant parts of this data with the authorities. Users will easily be able to submit traffic violation reports by taking a picture of the violation and the related car's license plate, instead of going the whole process of contacting the authorities directly. Simplifying the reporting process is also likely to increase the amount of reports coming from the public.

The application will focus on parking violations instead of all kinds of traffic violations because upon checking reviews of similar applications we came to the conclusion that most submissions of non-stationary traffic violations are denied as they either lack a clear display of the violation or the license plate. This creates frustration among the users, which results with low review scores and many users uninstalling the app.

Each user has to verify the submissions of other users while submitting a traffic violation. This system was implemented because according to the studies of similar services, around 75% of the submissions are denied because it is either spam or it provides insufficient information. These faulty submissions will be eliminated in a crowd-sourced way before they are presented to the authorities to decrease the workload of the authority users. A similar applications which is used in a single city boasts 50000+ downloads and it is unfeasible to expect authority users to weed out all improper submissions.

Authorities will be able to view submitted violations in a structured order, verify them and generate traffic tickets accordingly, with the information provided. The regular users and authorities will also be able to view a heatmap of traffic violation hotspots, and authorities will be able to view license plates which commit the most violations and an intervention map which presents unsafe areas and possible solutions. The system should be easy to use and reliable for both types of users and should be scalable in case of future content updates.

1.3.2. Goals

[G1]. Allow a visitor to be registered to the SafeStreets application as a regular user giving their credentials.

[G2]. Allow authorities to be registered to the SafeStreets application as an authority user giving their dedicated authority email address and other credentials.

[G3]. Allow a regular user to submit a violation.

- [G3.1]. Allow a regular user to take a photo of the general scene of the violation.
- [G3.2]. Allow a regular user to register the car’s license plate through a photograph.

[G3.3]. Allow a regular user to select the violation type.

[G3.4]. Allow the application to record the location information when the violation photo is taken.

[G4]. Allow regular users to validate others' violation submissions.

[G5]. Allow authority users to approve or disapprove a pending violation in their city of duty.

[G6]. Allow regular users and authority users to visualize approved violation frequency information on a heatmap.

[G7]. Allow authority users to see a list of cars that committed the highest number of violations in their city of duty.

[G8]. Allow authority users to receive suggestions for traffic regulations that can be applied to potentially unsafe areas.

1.4. Definitions, Acronyms, Abbreviations

1.4.1. Definitions

1. Authority E-mail: The dedicated email address of an authority member.
2. Regular User Interface: The interface a regular uses sees once logged into the application.
3. Authority User Interface: The interface an authority uses sees once logged into the application.
4. Violation: Traffic violation to be reported in the application.
5. Submitted Violation: Violation that is submitted by a regular user which is in the process of validation.
6. Validation: The process where a regular user is asked to confirm the information about the submitted violation type and license plate.
7. Validated/Pending Violation: A submitted violation which has been positively validated by three regular users and was transferred to the pending violation queue.
8. Pending Violation Queue: The queue in which the validated violations are present for an authority user to approve/disapprove.
9. Approved Violation: A pending violation which is approved by an authority user, which means that the violation is ready to be ticketed.
10. City of Duty: The city in which the authority user is on duty.
11. Heatmap: The map of a city which shows the violation frequencies of streets in color temperature.
12. Unsafe Areas: The areas in which there are relatively higher number of traffic violations.

13. Intervention Map: A map which displays potentially unsafe areas and suggestions for intervention.

13. Crossed data: The overlapping locations between SafeStreets application's approved violation data and the data provided by the authorities, used to detect unsafe areas.

14. Security objects: Total of security objects kept in database which keeps photograph information of violations to ensure chain of custody. This information is linked to their associated violation by an encrypted ID.

1.4.2. Acronyms

1. RASD: Requirements and Analysis Specification Document
2. API: Application Programming Interface
3. AI: Artificial Intelligence
4. ID: Identifier

1.4.3. Abbreviations

1. [Gn]: n-th Goal
2. [Dn]: n-th Domain Assumption
3. [Rn]: n-th Functional Requirement

1.5. References

- Specification Document: 'SafeStreets Mandatory Project Assignment'
- Similar application app store statistics: https://play.google.com/store/apps/details?id=com.ichangemycity.publiceye&hl=en
- Paper based on similar application: https://pdfs.semanticscholar.org/a01b/a3cf4a46eb3c195038294c3e0f34fdcb13dd.pdf
- IEEE Recommended Practice for Software Requirements Specifications - IEE Std 830-1998

1.6. Overview

1. Introduction: In the first section is an introduction of the problem is stated. The brief introduction is followed by the list of identified goals and the basic information to understand the rest of the document.

2. Overall Description: The second section consists of an overall description of the system in which the boundaries of the system are declared and actors enlisted. The boundaries are presented with all the assumptions necessary. The assumptions cover both to understand the specific needs of the regular users and the authority users.

3. Specific Requirements: The third section covers the specific requirements for the application to function. Functional and non-functional requirements are identified and listed in this section.

4. Scenarios: The fourth section is covering six scenarios, each of which try to picture particular situations that the system has to go through and how it works in specific cases.

5. UML Modeling: The fifth section is entirely composed of the UML diagrams to model the system in detail.

6. Alloy Modeling: The sixth section embodies the Alloy model of the system. It covers the dynamics of the system in detail with a proof of consistency and an example of a generated world.

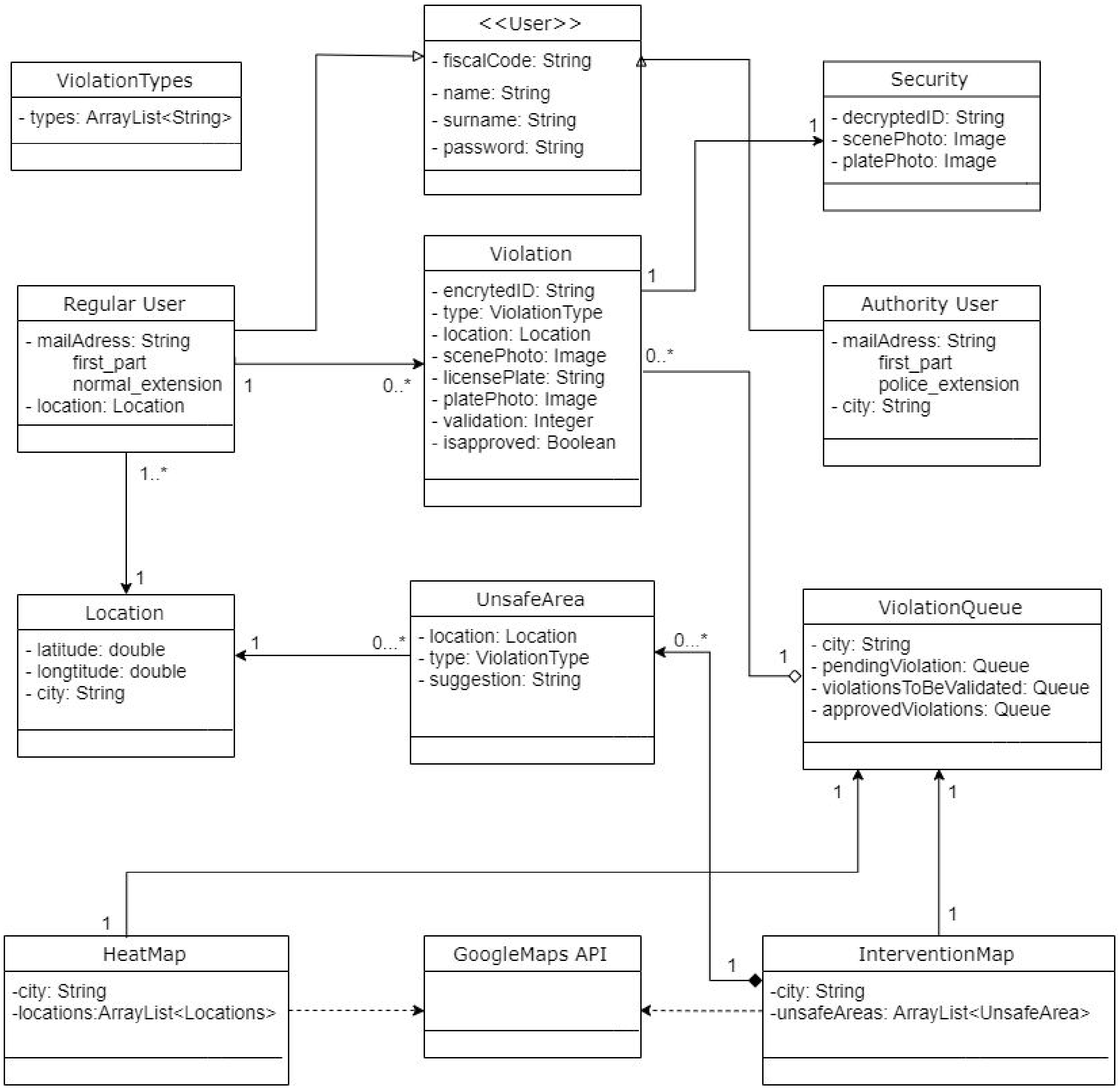
7. Appendix: The last section is accessory and contains a list of the tools used to create this document. Additionally, it includes a table of hours spent.

2. Overall Description

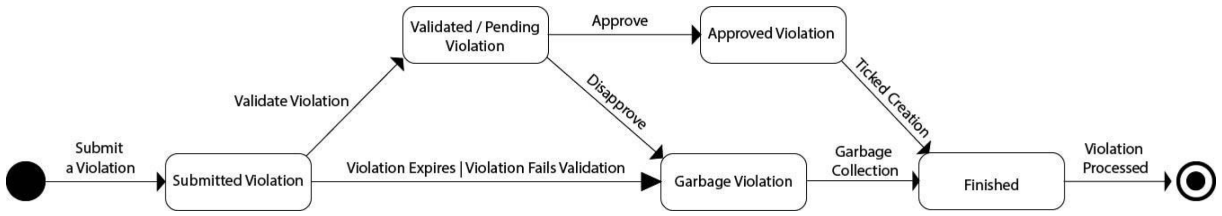
2.1. Product Perspective

SafeStreets is a standalone software service which offers a mobile and web application. The service relies on Google Maps API while displaying the Heatmap and Intervention Map, and it relies on AnyLines API to recognize license plates. While these API's are chosen for the development, they are by no means irreplaceable in the future updates on the project. Regular users can access the mobile application, which supports iOS and Android platforms while authority users can access the application from any modern browser. The class diagram below shows detailed information about the relationships between the classes within the SafeStreets service.

2.1.1. Class Diagram



2.1.2. State Diagram



2.2. Product Functions

In this section a detailed summary of the most important functions of the system are provided. We addressed major requirements involved in the goals like user registration, violation submission and providing intervention suggestions.

2.2.1. Registration Management

Separation of user types in SafeStreets is an important aspect in the sense of the applications usage and levels of visibility of the stored data. In the process of regular user registration, a real fiscal code is going to be required which will be verified by an external system provided by the municipality. If an authority wants to register, a dedicated email address is going to be required additionally to complete the process and direct authority user to another user interface than the regular user interface. This dedicated email addresses will be recognized by the application because of their specific extensions and system is going to register them as authority user. Both regular and authority users will receive an email for verification of their email address and their registration. Also, the users are going to create their passwords after verifying their registration via this email.

2.2.2. Acquiring Violation Details

Submitting a violation will be a serious process for SafeStreets application because checking the certainty and accuracy of the information entered by the users is one of the most important requirements of the application. The application initially brings a camera view to the screen for the user to take a photo of the violation. Afterwards AnyLines API is launched, which detects licence plate in real time from the camera view. Then the application retrieves violation types from the server and allow the user to choose the appropriate one. Since the application will keep records of violations, it will include time of the submission. In addition, SafeStreets is going to utilize GPS data in order to keep a record of the submission location. Last step is required to complete the submission which is going to be the validation of other three regular user’s submitted violations. In the next part, this function is going to be explained in detail.

2.2.3. Validation of Submitted Violations

In the process of violation submission, SafeStreets is also going to ask users to validate three previously submitted violations to achieve more accurate reports to send to the authorities. If there are no unvalidated submissions, the application will skip this step and allow user to submit the violation. Application is going to show the user the pictures of the previous violation submissions and ask them to check the correctness of violation type and the license plate by comparing the pictures and the recorded data. If the application can get three positive validations on a submitted violation it will add it to the pending violations queue, otherwise it will be deleted.

2.2.4. Generating Possible Interventions

Another aim of the application is to use its stored data and create useful solutions with the contribution of the municipality. SafeStreets is going to take the information of accidents that occurs on the territory of the municipality and cross it with its own violation data to identify potentially unsafe areas. Authority users will be able to view unsafe areas on a map and they can click these areas on the map to get intervention suggestions. These suggestions will be selected by the application from a list of possible interventions that were agreed upon with the municipality.

2.3. User Characteristic

1. Visitor: Someone who does not yet have an account on SafeStreets application. The only thing a visitor can do is to register in the application. In order to do so, they are required to have a fiscal code.
2. Regular User: A civilian who has registered to the SafeStreets application. A regular user is able to login to the system, submit a violation, validate other violations, see a heatmap of violation density.
3. Authority User: An authority member (e.g. policeman) who has registered to the SafeStreets application. An authority user is a member of a local authority such as the municipality or police department. An authority user can login, approve violations, see a number of violations a car has committed, visualize a heatmap of violation frequency and see the intervention map.

2.4. Assumptions and Dependencies

2.4.1. Domain Assumptions

[D1]. The correctness of fiscal code, name and surname combination may be verified using an external service.

[D2]. Fiscal code, which is also the username, is assumed to be unique.

[D3]. When the system sends an email to register a regular user, this email is assumed to be delivered correctly.

[D4]. The email provided is a local authority email extension. Authorities have distinct e-mail extensions. These extensions are used solely by the members of these institutions.

[D5]. The city of duty of the authority user is assumed to be detected correctly by the external server.

[D6]. Location services are accessible by the application.

[D7]. Phones that use this application are assumed to have camera and location services.

[D8]. The GPS of the phone is accurate.

[D9]. The resolution of the camera is processable by the AI algorithm.

[D10]. There are active regular users who uploaded a certain number of violations.

[D11]. There are active regular users who validated a certain number of violations.

[D12]. There are active authority users who approved a certain number of violations.

[D13]. The municipality of a given city shares the information of accidents that occurs on the territory of the municipality with the application.

3. Specific Requirements

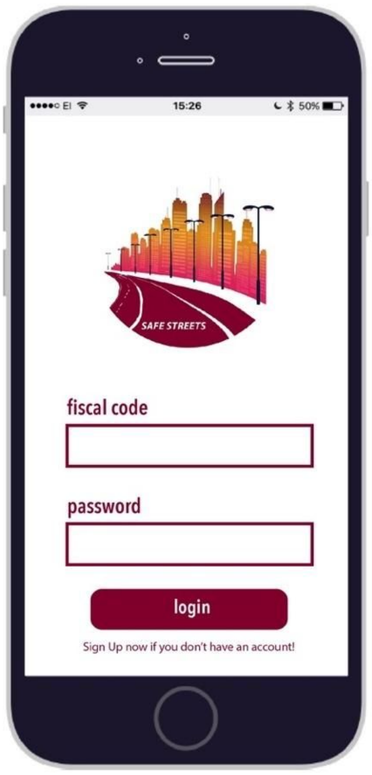
3.1. External Interface Requirements

3.1.1. User Interfaces

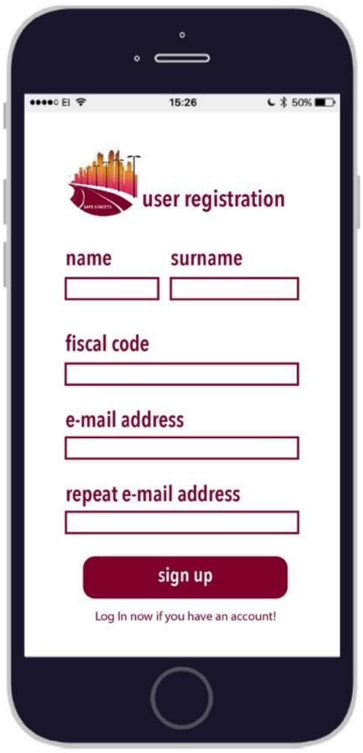
The following illustrations represent what the application will look like in the first release.

3.1.1.1. Mobile Application Interfaces

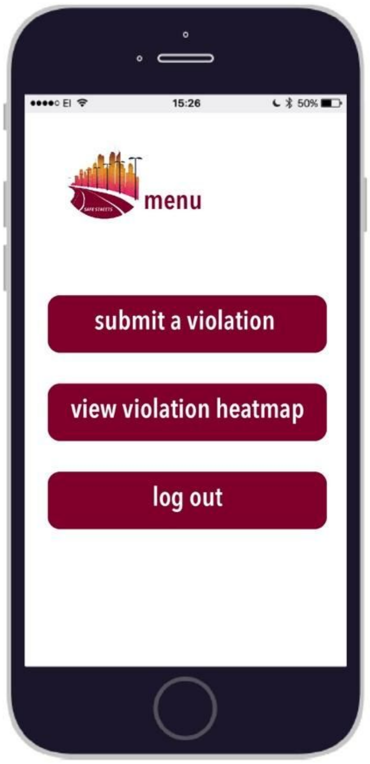
3.1.1.1.1. Login



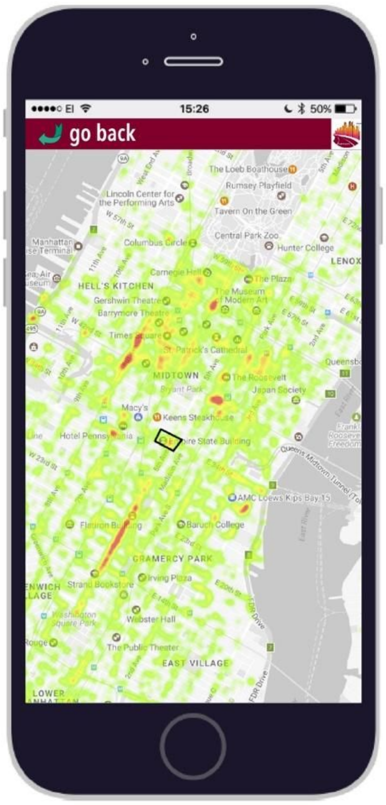
3.1.1.1.2. Registration



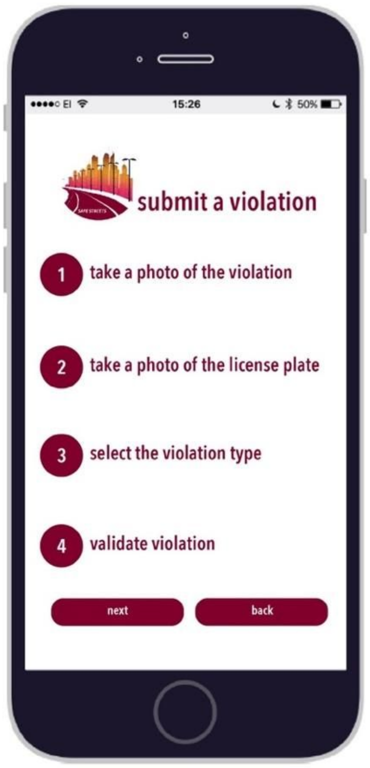
3.1.1.1.3. User Menu



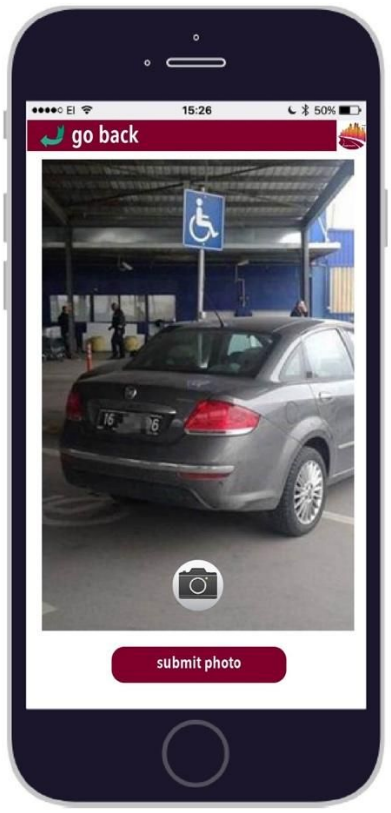
3.1.1.1.4. Heatmap



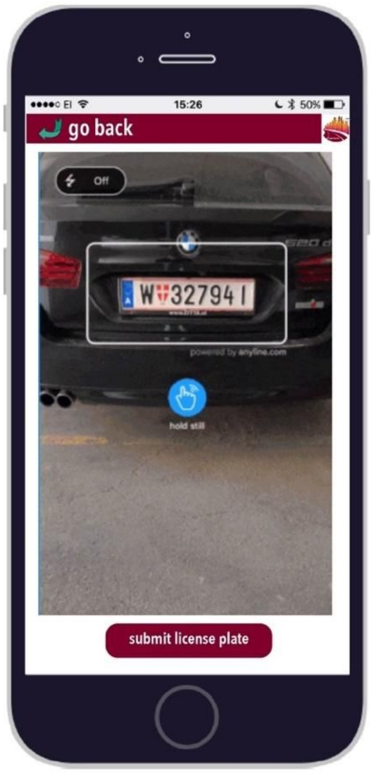
3.1.1.1.5. Submit a Violation


15:26
submit a violation
1 take a photo of the violation
2 take a photo of the license plate
3 select the violation type
4 validate violation
next
back

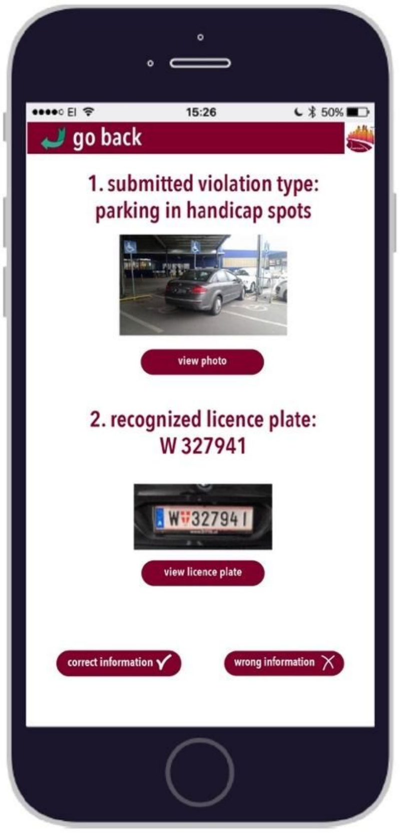
3.1.1.1.6. Taking a Photo General Photo of the Scene



3.1.1.1.7. Photographing the License Plate

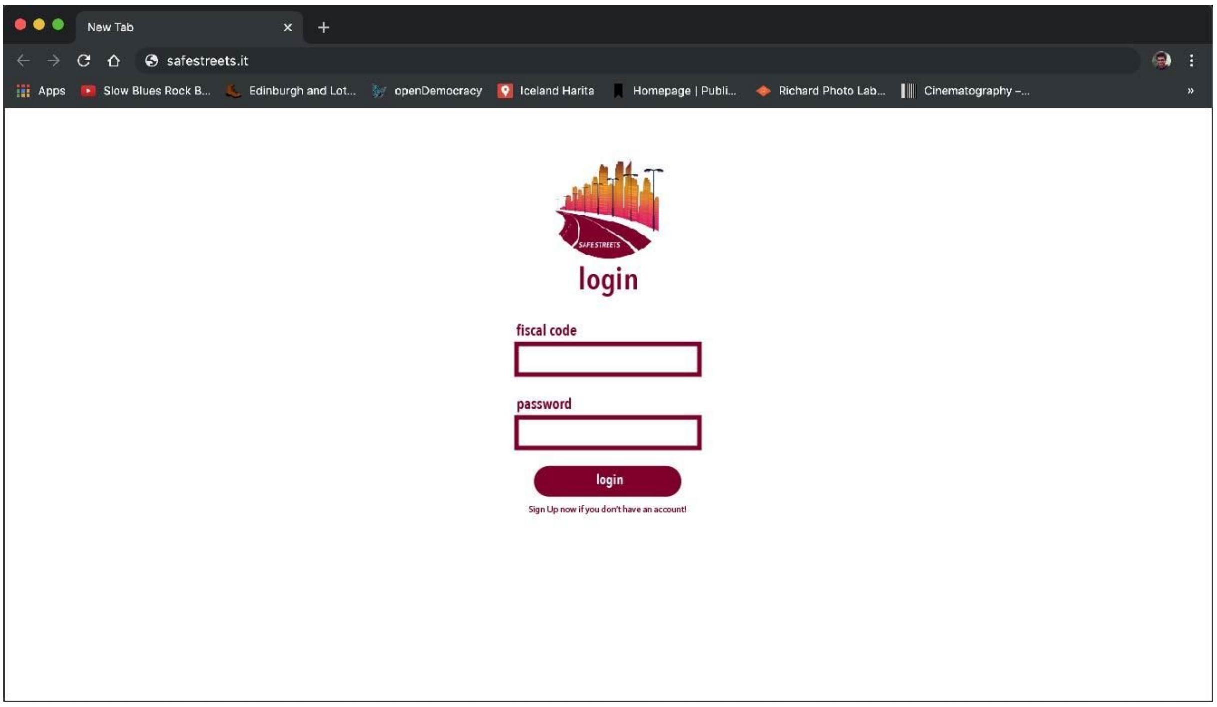


3.1.1.1.8. Validation

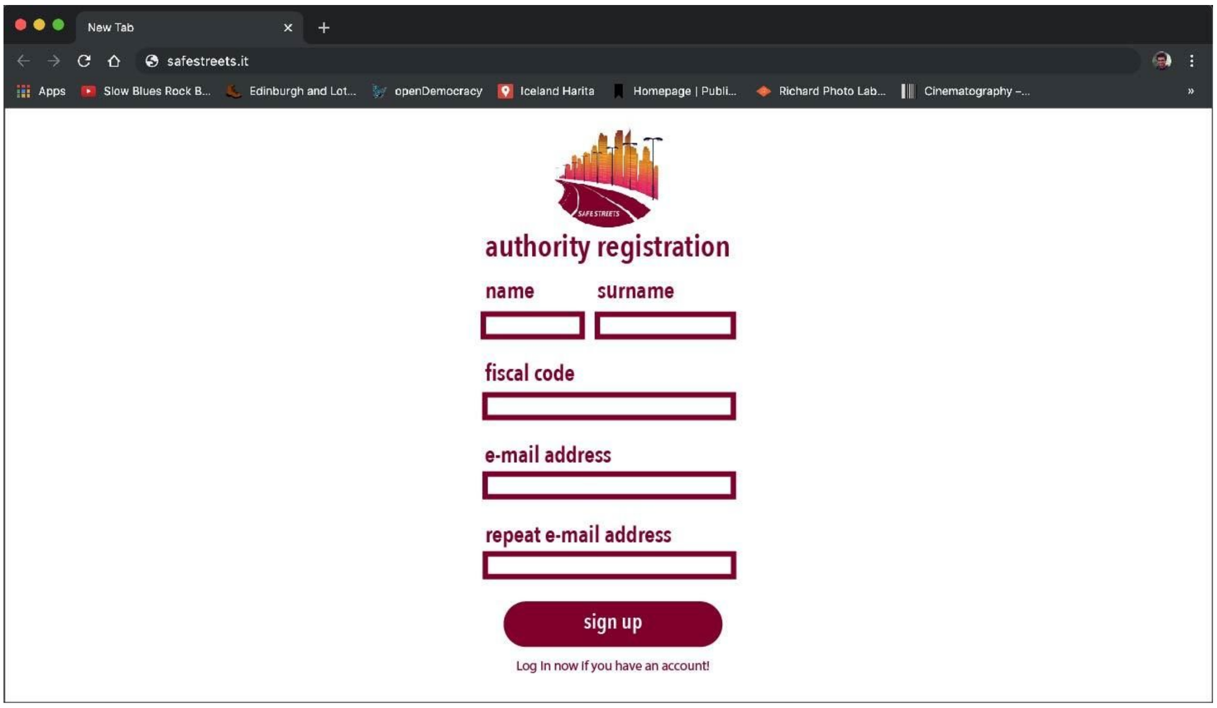

15:26
go back
1. submitted violation type: parking in handicap spots
view photo
2. recognized licence plate: W 327941
view licence plate
correct information
wrong information

3.1.1.2. Web Application Interfaces

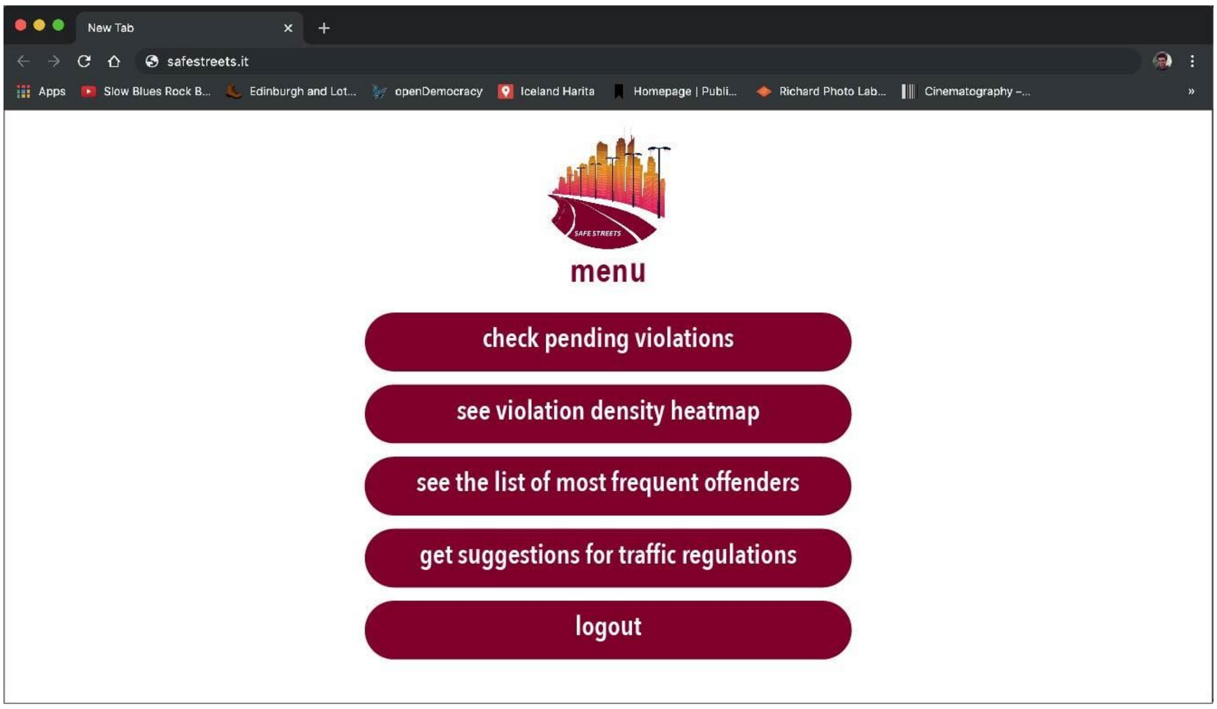
3.1.1.1.1. Login



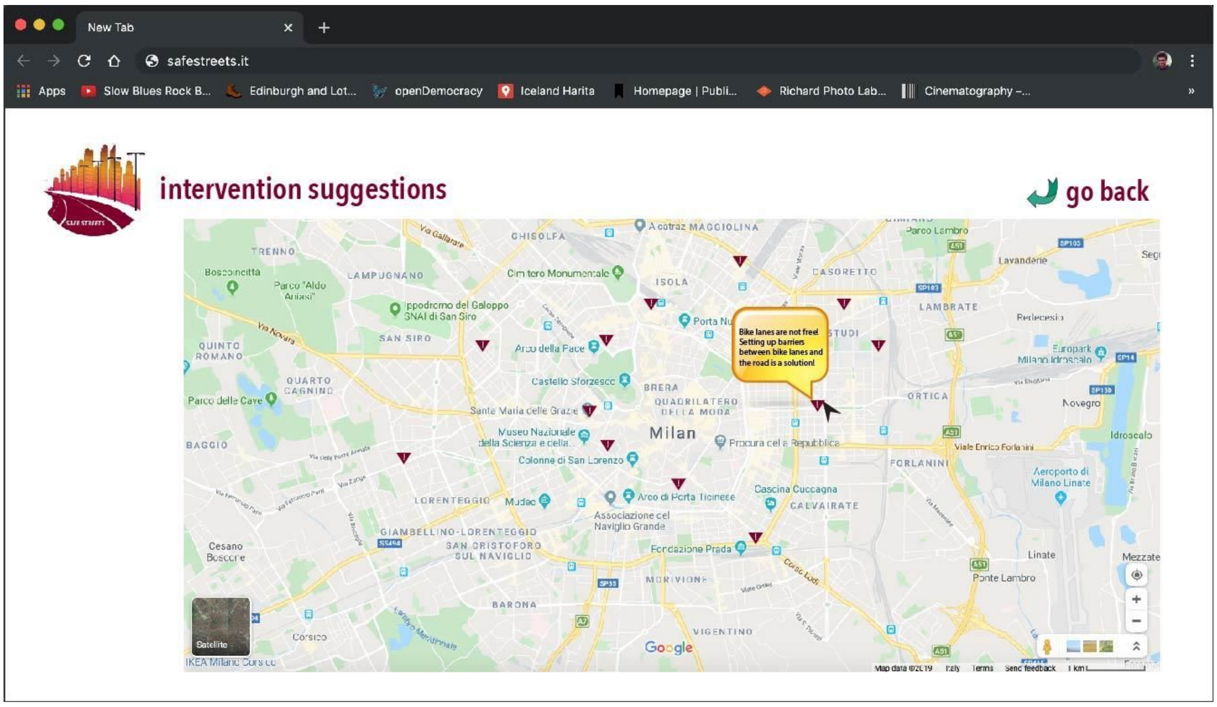
3.1.1.1.2. Registration



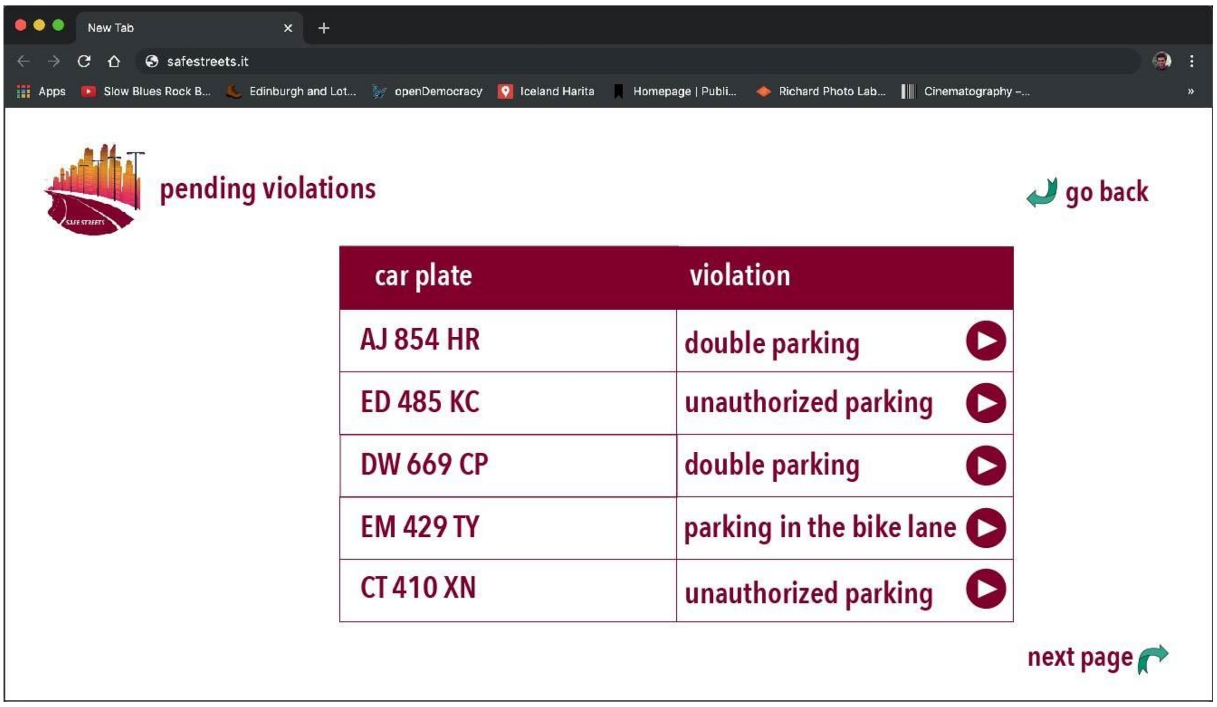
3.1.1.1.3. Menu


safestreets.it
SAFE STREETS
menu
check pending violations
see violation density heatmap
see the list of most frequent offenders
get suggestions for traffic regulations
logout

3.1.1.1.4. Possible Intervention Suggestions


safestreets.it
intervention suggestions
go back
Milan
Bike lanes are not free! Setting up barriers between bike lanes and the road is a solution!
Satellite

3.1.1.1.5. Pending Violations



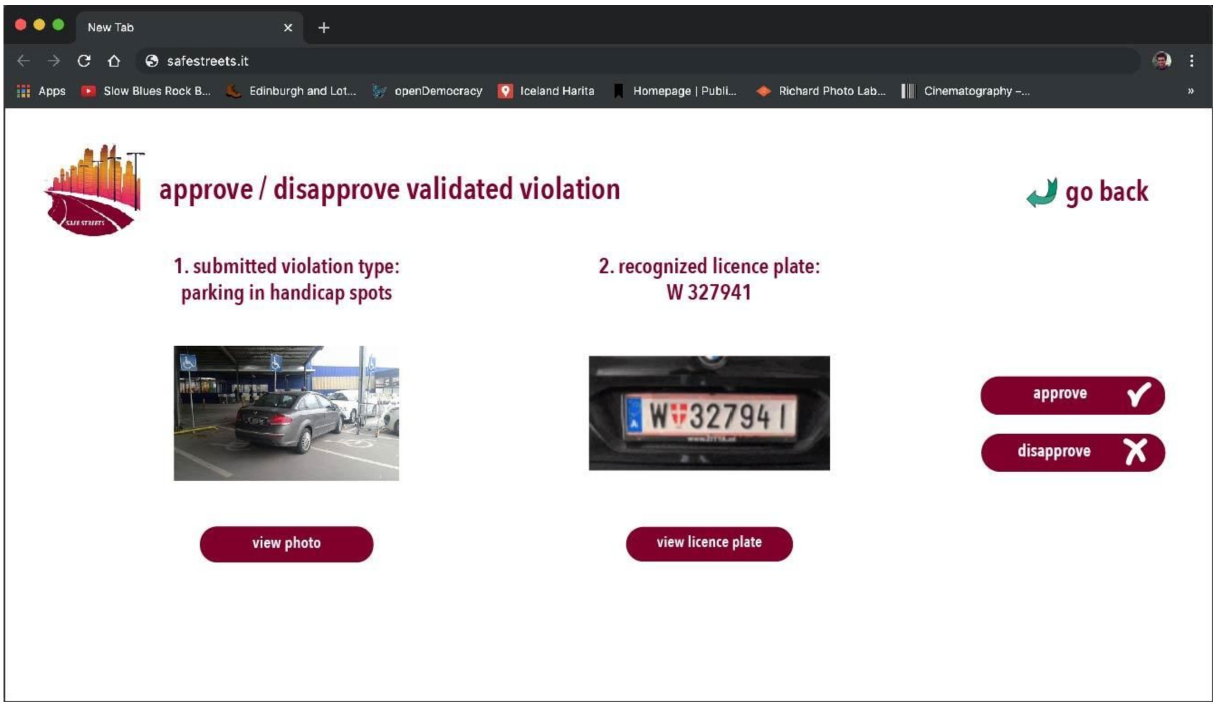
pending violations

go back

car plate	violation
AJ 854 HR	double parking
ED 485 KC	unauthorized parking
DW 669 CP	double parking
EM 429 TY	parking in the bike lane
CT 410 XN	unauthorized parking

next page

3.1.1.1.6. Approve or Disapprove

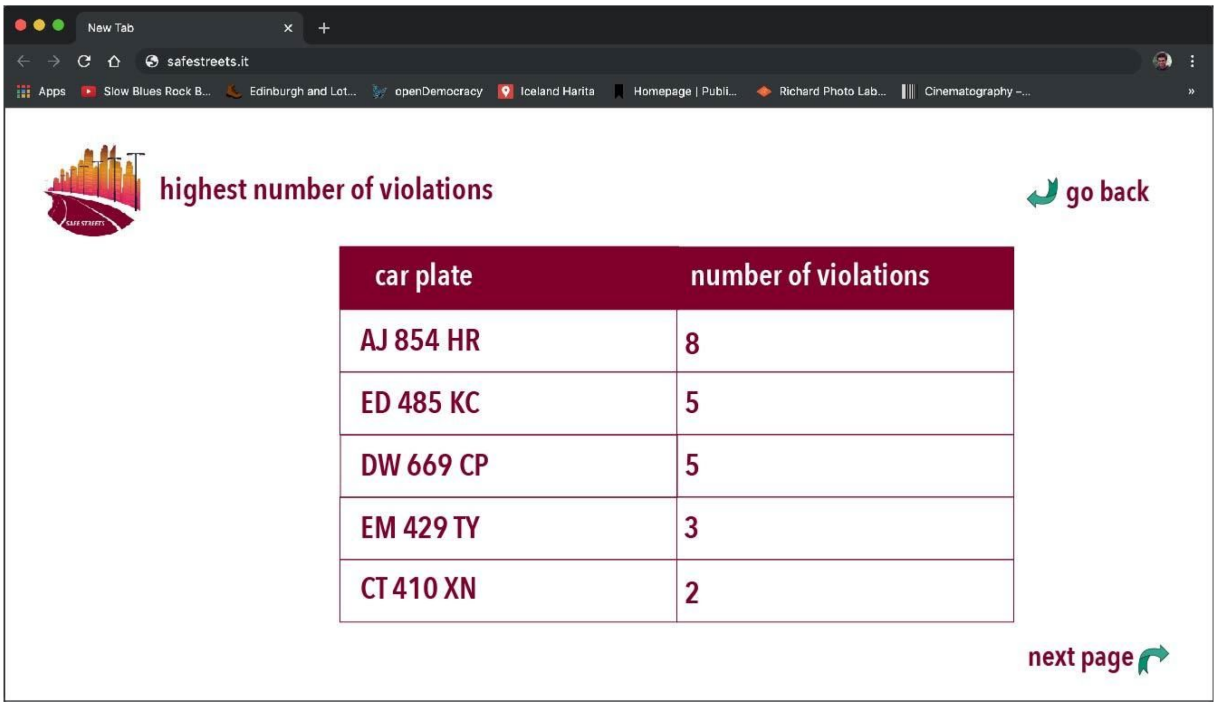


approve / disapprove validated violation

1. submitted violation type:
parking in handicap spots

2. recognized licence plate:
W 327941

3.1.1.1.7. The List of Frequent Offenders



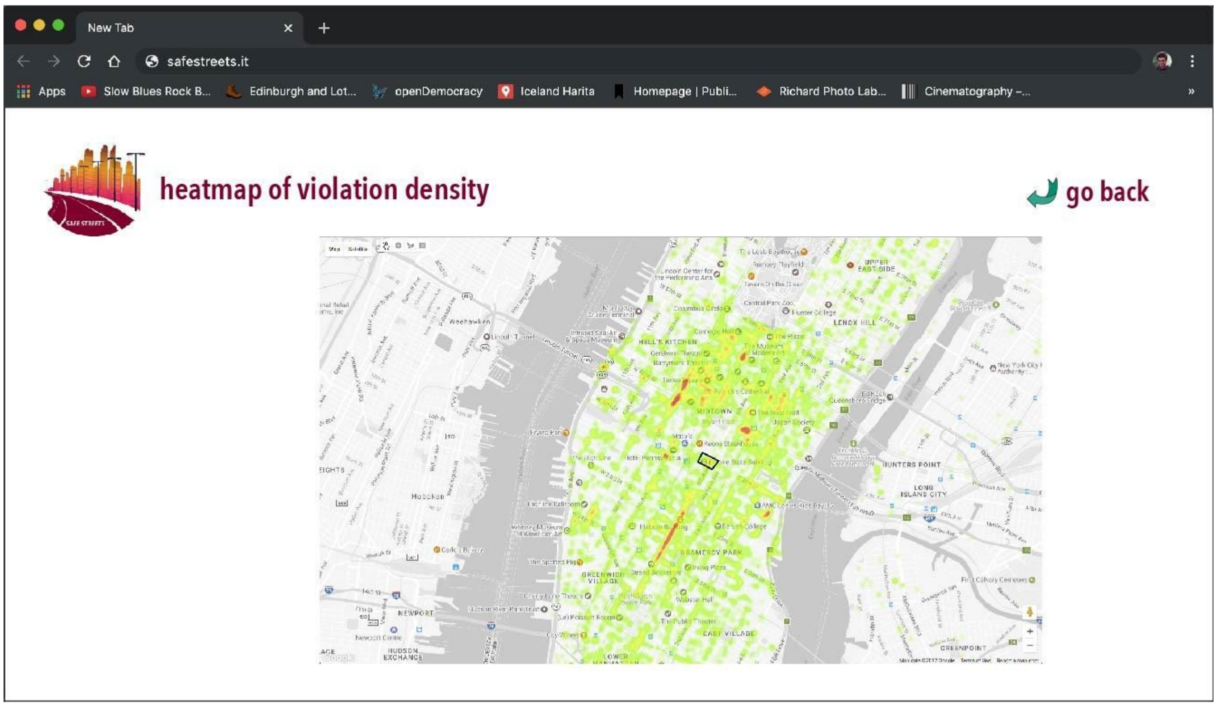
highest number of violations

go back

car plate	number of violations
AJ 854 HR	8
ED 485 KC	5
DW 669 CP	5
EM 429 TY	3
CT 410 XN	2

next page

3.1.1.1.8. Heatmap



3.1.2. Hardware Interfaces

Regular users will be able to reach the service with their iOS or Android mobile devices. For the utilization of the app the devices must have an internet connection, roaming connection is recommended. The devices should also have GPS and camera services, and these services must be made available for the application's access. Authority users will be able to reach the service with modern web browsers, which are supported by most PC operating systems, iOS and Android.

3.1.3. Software Interfaces

- Google Maps JavaScript API v3.39:

Information Link: https://developers.google.com/maps/documentation/javascript/tutorial
Use: Google Maps API will be used to display SafeStreets application's data as a heatmap to regular and authority users. The heatmap will be weekly updated by feeding SafeStrees data into Google Maps API, and the current week's heatmap will be available for view to all users. Additionally Google Maps API will also be used to create a "get suggestions" map, which can be accessed by the authority users. SafeStreets application will cross its data with the traffic accident data provided by the municipality to designate unsafe areas. These unsafe areas and their proposed solutions will be fed into Google Maps API to create the "get suggestions" map, which will be updated monthly. The map will have pointers at the unsafe areas and clicking on these pointers will reveal the proposed solutions to the user.

- AnyLines License Plate Scanner (ALPS) v15:

Information Link: https://anyline.com/products/scan-license-plates/
Use: SafeStreets will use AnyLines API during a regular user's violation creation. The API will be used to scan the license plate of the car committing the violation. API offers real time scanning from camera and is supported for iOS and Android devices. API can identify the license plate and the country it belongs to. The identified license plate will be saved along with other violation information during the creation of a submitted violation.

3.1.4. Interfaces Provided by the Municipalities

SafeStreets is designed to be an application which assists the municipalities, and this requires exchange of data between the application and the databases of the municipality. SafeStreets relies on municipality interfaces to check the validity of fiscal codes and requires traffic accident data from the municipalities for the process of designating potentially unsafe areas.

3.1.5. Communication Interfaces

The application will use the internet to communicate between user devices and the server. Wi-fi connection is enough to access the application but because the application is designed to be used en-route by regular users, cellular internet connection (2G/3G/4G) is highly recommended.

3.2. Functional Requirements

3.2.1. Requirements

[G1]. Allow a visitor to be registered to the SafeStreets application as a regular user giving their credentials:

- [D1]. The correctness of fiscal code, name and surname combination may be verified using an external service.
- [D2]. Fiscal code, which is also the username, is assumed to be unique.
- [D3]. When the system sends an email to register a regular user, this email is assumed to be delivered correctly.

- [R1.1]. A visitor must start with the registration process. The system asks for fiscal code, name, surname, email and phone number to register.
- [R2]. A user verifies the account and creates a password through a received email.
- [R3]. Signing up to the mobile application results in the creation of a regular user's account.

[G2]. Allow authorities to be registered to the SafeStreets application as an authority user giving their dedicated authority email address and other credentials:

- [D1]. The correctness of fiscal code, name and surname combination may be verified using an external service.
- [D2]. Fiscal code, which is also the username, is assumed to be unique.
- [D4]. The email provided is a local authority email extension. Authorities have distinct e-mail extensions. These extensions are used solely by the members of these institutions.

- [D5]. The city of duty of the authority user is assumed to be detected correctly by the external server.
- [R1.2]. A visitor must start with the registration process. The system asks for fiscal code, name, surname, email, phone number and the associated city of duty to register.
- [R2]. A user verifies the account and creates a password through a received email.
- [R4]. Signing up to the web application results in the creation of an authority user's account.

[G3]. Allow a regular user to submit a violation:

- [D5]. Camera access is enabled for the application.
- [D6]. Location services are accessible by the application.
- [D7]. Phones that use this application are assumed to have camera and location services.
- [D8]. The GPS of the phone is accurate.
- [D9]. The resolution of the camera is processable by the AI algorithm.

- [R5]. A registered regular user must be able to login to the mobile application and utilize the regular user interface.
- [R6]. An Artificial Intelligence (AI) API is used to recognize the license plate number during taking a photo of the plate.
- [R7]. The system presents a list of violation types for the user to choose.
- [R8]. The application records the time and date of violation submission.
- [R9]. The application utilizes GPS data to record the location of the submission.
- [R10]. The application asks a regular user to validate, if exists, three random violations previously submitted by other regular users.

[G4]. Allow regular users to validate others' submitted violation:

- [D10]. There are active regular users who uploaded a certain number of violations

- [R5]. A registered regular user must be able to login to the mobile application and utilize the regular user interface.

- [R11]. Application asks user to validate, if present, three random submitted violations before their submission gets to the validation queue to be validated.
- [R12]. Application shows the pictures of the submitted violation in question and presents buttons for the user to validate the correctness of the car plate number and violation type.
- [R13]. A submitted violation goes through three validation processes, if all of them are positive, then the submitted violation becomes a pending violation. Else, the submitted violation is deleted by the application.

[G5]. Allow authority users to approve or disapprove a pending violation in their city of duty:

- [D11]. There are active regular users who validated a certain number of violations.

- [R14]. A registered authority user must be able to login to the system and utilize the authority user interface.
- [R15]. Application shows a queue of pending violations to an authority user.
- [R16]. Application presents approve/disapprove buttons for each pending violation.
- [R17]. Ensure the chain of custody from a regular user's submission to the view of an authority user.

[G6]. Allow regular and authority users to visualize approved violation frequency information on a heatmap:

- [D12]. There are active authority users who approved a certain number of violations.

- [R5]. A registered regular user must be able to login to the mobile application and utilize the regular user interface.
- [R14]. A registered authority user must be able to login to the system and utilize the authority user interface.
- [R18]. The system records the violations that are approved previously by an authority user.
- [R19]. The system uses the recorded violations and their locations to create a heatmap.
- [R20]. The system uses an existing map interface.

- [R21]. The system updates the heatmap weekly.

[G7]. Allow authority users to see a list of cars that committed the highest number of violations in their city of duty:

- [D12]. There are active authority users who approved a certain number of violations.

- [R14]. A registered authority user must be able to login to the system and utilize the authority user interface.
- [R18]. The system records the violations that are approved previously by an authority user.

[G8]. Allow authority users to receive possible intervention suggestions for traffic regulations for that can be applied to potentially unsafe areas:

- [D13]. The municipality of a given city shares the information of accidents that occurs on the territory of the municipality with the application.

- [R14]. A registered authority user must be able to login to the system and utilize the authority user interface.
- [R20]. The system uses an existing map interface.
- [R22]. The application crosses its data with the information that comes from the municipality.
- [R23]. The application makes meaning out of the crossed data in order to suggest possible interventions.
- [R24]. Intervention map is updated monthly.

3.2.2. Scenarios

3.2.2.1. Scenario 1

Ricardo meets his friend Alessandra for dinner at an Amalfitana restaurant in Milano. Throughout their dinner they talk about various things and catch up with each other. At some point, Alessandra tells Ricardo about the parking violations that occur very frequently on her street. She tells Ricardo that there has always been parking problems on her street but with the emergence of these recent wrong parking and a lot of cars parking without the permit on her street has been troubling her. She mentions that she is missing an easy way to report such illegal matters, upon which Ricardo mentions the application SafeStreets. Alessandra downloads the application at the moment and registers giving the necessary information such as her name, surname fiscal code and e-mail address. She receives an e-mail upon which she completes the registration and gets her password. All this happens very quick so Ricardo doesn't get bored but instead feels happy for being helpful. As the dinner ends, Alessandra goes back home but again has hard time parking her car. After a while she finds a spot and parks her car. This time instead of going directly home, she looks for cars that don't have permit to park in her street. She logs in to SafeStreets and reports three cars that have parked on her street without a permit to park in there. In order to submit each application, she is asked to validate three violation submissions that are reported previously by other people. She gladly validates upcoming violation reports and submits the violations on her street.

3.2.2.2. Scenario 2

Matteo is invited to an art exhibition opening near Duomo, Milano. He also wants to take his mother who needs a wheelchair to move. For this reason, Matteo has to find a street where people don't park on disabled spots and with the lowest number of violations happening

where it is also close to the art gallery that will have the opening. He logs into SafeStreets application (which he uses frequently for cars that park in disabled spots) and checks the violation density heatmap near Duomo. He locates a street that looks rather greener. He decides to drive there to park the car.

3.2.2.3. Scenario 3

Silvia is a new starter at the police department of Milano. Upon arrival, her boss tells her to check traffic violations and bring him a report. Silvia is supposed to focus on parking violations until she can understand how the different branches of the police department work in real life. Using her police e-mail address, she registers to the SafeStreets web application to see the traffic violations as her boss asked. She starts browsing through validated violations that occurred. First thing she does is to view the highest number of violations a car has committed, then she looks at the violation density heatmap map to see where the most violations occur. Crossing these two sources of information, she prepares a possible list of interventions. After creating her own list, she extracts the list of possible interventions from the web application. She creates a report that crosses and compares the two lists of intervention to present in the next meeting with her boss.

3.2.2.4. Scenario 4

Andrea works at the police department for ticketing of traffic violations. For certain personal reasons, he can't go to work for a week and someone else has to take responsibility of his workload. He asks his friend Marco. Marco is ok with following up the task. He creates his own SafeStreets authority user account and goes into the pending violations list. He views the validated violations that need require his approval. He approves the first two violations, the third one breaks the chain of custody, so he moves to the next violations. After completing a dozen of them, he returns to his own tasks. He decides to continue with SafeStreets after lunch that day.

3.2.2.5. Scenario 5

Angela has a personal parking spot at her work that has a sign showing her own license plate number. However, when she gets to work on Monday morning, she sees another car parking on her place. She takes out her phone and takes a photo of the general scene. Then the program asks her to point her phone camera to the license plate. The algorithm is not able to read the plate and asks her to get closer. She takes a few steps and gets next to the car and finally the program reads the license plate.

3.2.2.6. Scenario 6

Carlo wants to submit a parking violation that the same car commits near his father's house. He takes a photo of the general scene and the license plate too. As he presses submit, SafeStreets application asks him to validate a violation. The first violation has the necessary information correctly. Carlo marks it as correct, the next violation that comes up has a different car in the general scene than the car plate that was presented. He marks it incorrect. The information on the third one is correct, so he follows with the correct information button. After controlling three violations, his violation submission is recorded.

3.2.3. Use Case Descriptions

3.2.3.1. User Sign Up

Actors: Visitor

Goals: [G1] [G2]

Input Conditions: -

Event Flow:

1. The Visitor clicks the "Sign Up now if you don't have an account!" button on the main page of the application to begin registration.
2. The Visitor fills their credentials into the presented fields.
3. The Visitor clicks the "sign up" button.
4. The system uses an external server to confirm credentials, email is checked to be associated to the authority if the user is signing-up on the web application.
5. An email is sent to the address provided by the Visitor, which is used to activate the account and set a password.
6. The user enters a password.
7. The user is saved into the SafeStreets database as a regular user if they completed their sign-up on the mobile application, else if they completed their sign-up on the web application they are registered as an authority user.

Output Conditions:

The Visitor successfully ends registration and becomes a Regular User/Authority. The user can now login to the associated application with his/her fiscal code and designated password.

Exceptions:

1. The fiscal code, name and surname combination is not verified by the external system of municipality. An error message is shown to the visitor and event flow returns to 2.
2. The password is filled improperly. Password field is highlighted, and the user is notified. Event flow returns to 6.
3. The email addresses in "email address" and "repeat email address" fields do not match. Both fields are highlighted, and the user is notified. Event flow returns to 2.
4. For web application sign-up, the email cannot be matched to an authority user by the external server. An error message is shown to the visitor and event flow returns to 2.
5. The internet connection is lost. The visitor is notified, event flow returns to 2.

3.2.3.2. Regular User Violation Submission

Actors: Regular User

Goals: [G3] [G4]

Input Conditions: Regular User is logged in to the mobile application.

Event Flow:

1. The Regular User clicks the "submit a violation" button on the Regular User Menu.
2. The system presents the information page.
3. The Regular User views the information page and clicks the "next" button.
4. The system presents the violation photo page.
5. The Regular User takes the photo of the violation using the camera view on the screen.
6. The Regular User clicks "submit photo" button.
7. The system presents the submit license plate page.
8. The Regular User point the camera on the license plate such that the license plate falls in the highlighted location on the camera view.
9. The Regular User clicks "submit license plate" button after the license plate is recognized.
10. The system presents the select violation type page.
11. The Regular User selects the violation type by clicking on the corresponding button.
12. The Regular User clicks the "select violation type" button.
13. The system presents the validation page.
14. The Regular User checks the presented unvalidated violation submission's information and selects "correct information" or "wrong information" accordingly.
15. The system saves the user's validation.
16. Even flow from 13 to 15 repeats two more times.
17. The system creates a new unvalidated violation submission with the information the Regular User provided.

Output Conditions:

The Regular User successfully created an unvalidated violation submission. The submission is added to the validation queue. The user is returned to the main menu.

Exceptions:

1. The Regular User's camera is unavailable. The user is notified of the problem and the application returns to the Regular User Menu

2. The internet connection is lost. The Regular User is notified and application halts until the connection is re-established, the user can cancel this wait by clicking the "back" button on the top right of the screen to return to the main menu.

3.2.3.3. Regular and Authority User Heatmap View

Actors: Regular User/Authority User

Goals: [G6]

Input Conditions: User is logged in to the application.

Event Flow:

1. The user clicks the “view violation heatmap” or “see violation density heatmap” button on the main menu.
2. The system presents the heatmap of the city they are located in.
3. The user moves around the map by swiping on the screen.
4. The user clicks "go back" button to return to the main menu.
5. The system presents the main menu.

Output Conditions:

Theuser successfully viewed the traffic violation heatmap.

Exceptions:

1. The internet connection is lost. The user is notified, and application returns to the main menu.

3.2.3.4. Authority User Approve or Disapprove Violation

Actors: Authority User

Goals: [G5]

Input Conditions: Authority User is logged in to the web application.

Event Flow:

1. The Authority User clicks the "check pending violations" button on the Authority User Menu.
2. The system presents the pending violation queue.
3. The Authority User selects a violation from the pending violation queue.
4. The system confirms the chain of custody and presents the violation details.
5. The Authority User selects approve or disapprove based on the credibility of the information.
6. The system records the input.

Output Conditions:

The Authority User successfully approved or disapproved a violation. If approved, violation information is stored in the database.

Exceptions:

1. The internet connection is lost. The Authority User is notified, and application returns to the Authority User Menu.

2. Chain of custody is not confirmed. The Authority User is notified, and the violation is removed from the pending queue. Application returns to the Authority User Menu.
3. There are no pending violations. The list that shows the pending queue is empty, the Authority User can go back to the main menu using the "go back" button.

3.2.3.5. Authority User Check Most Frequent Offenders

Actors: Authority User

Goals: [G7]

Input Conditions: Authority User is logged in to the web application.

Event Flow:

1. The Authority User clicks the "see the list of most frequent offenders" button on the Authority User Menu.

2. The system presents the most frequent offenders page.

3. The Authority User moves around the list using the buttons.

4. The Authority User clicks "go back" button to return to the Authority User Menu.

5. The system presents the Authority User Menu.

Output Conditions:

The Authority User successfully viewed the most frequent offenders.

Exceptions:

1. The internet connection is lost. The Authority User is notified, and application returns to the Authority User Menu.

3.2.3.6. Authority User Get Suggestions for Traffic Regulations

Actors: Authority User

Goals: [G8]

Input Conditions: Authority User is logged in to the web application.

Event Flow:

1. The Authority User clicks the "get suggestions for traffic regulations" button on the Authority User Menu.

2. The system presents the intervention map of their city of duty

3. The Authority User moves around the map by dragging the screen with their mouse.

4. The Authority User clicks on highlighted unsafe areas to view the intervention suggestions.

5. The Authority User clicks "go back" button to return to the Authority User Menu.

6. The system presents the Authority User Menu.

Output Conditions:

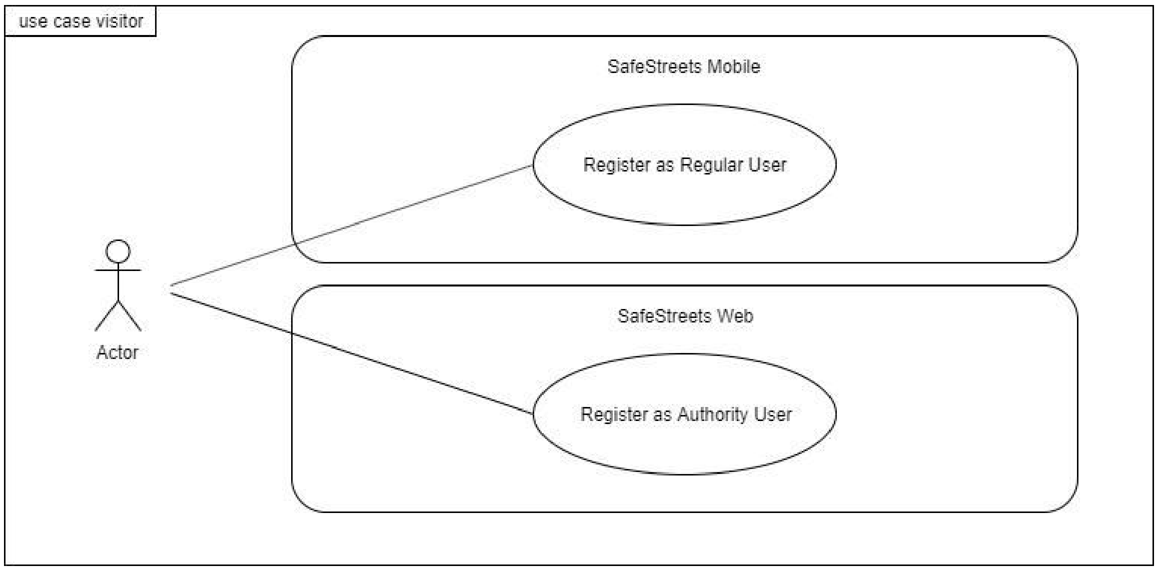
The Authority User successfully viewed the intervention map.

Exceptions:

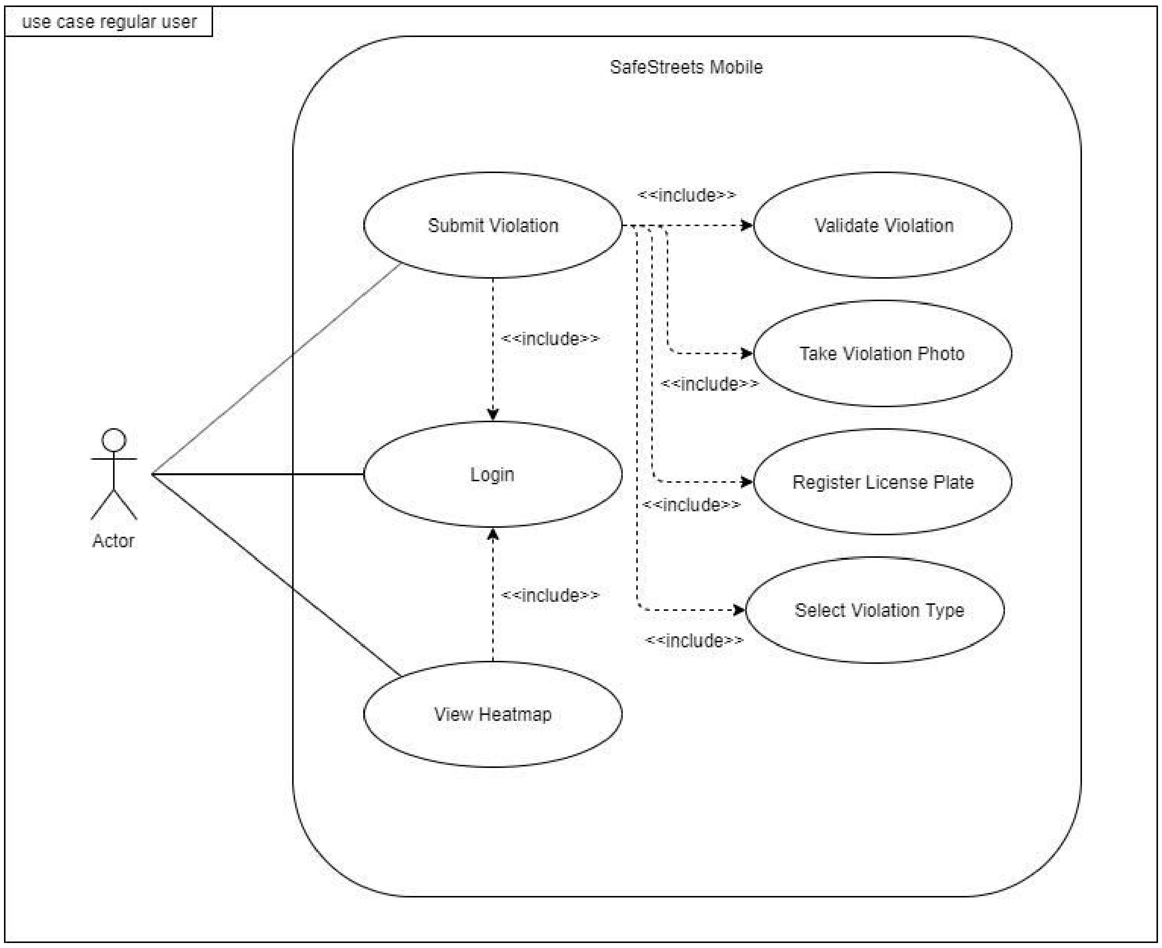
1. The internet connection is lost. The Authority User is notified, and application returns to the Authority User Menu.

3.2.4. Use Case Diagrams

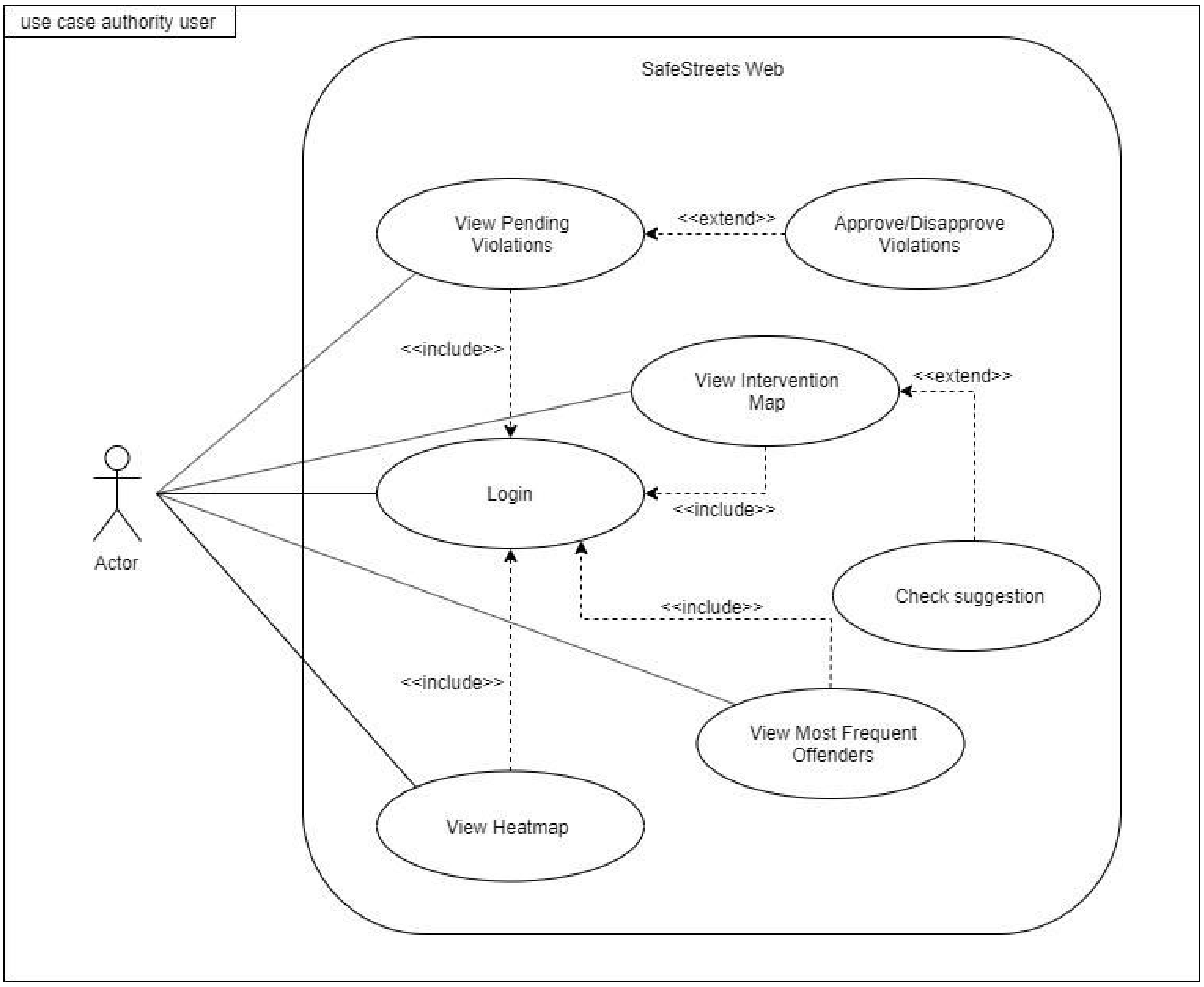
3.2.4.1. Visitor



3.2.4.2. Regular User

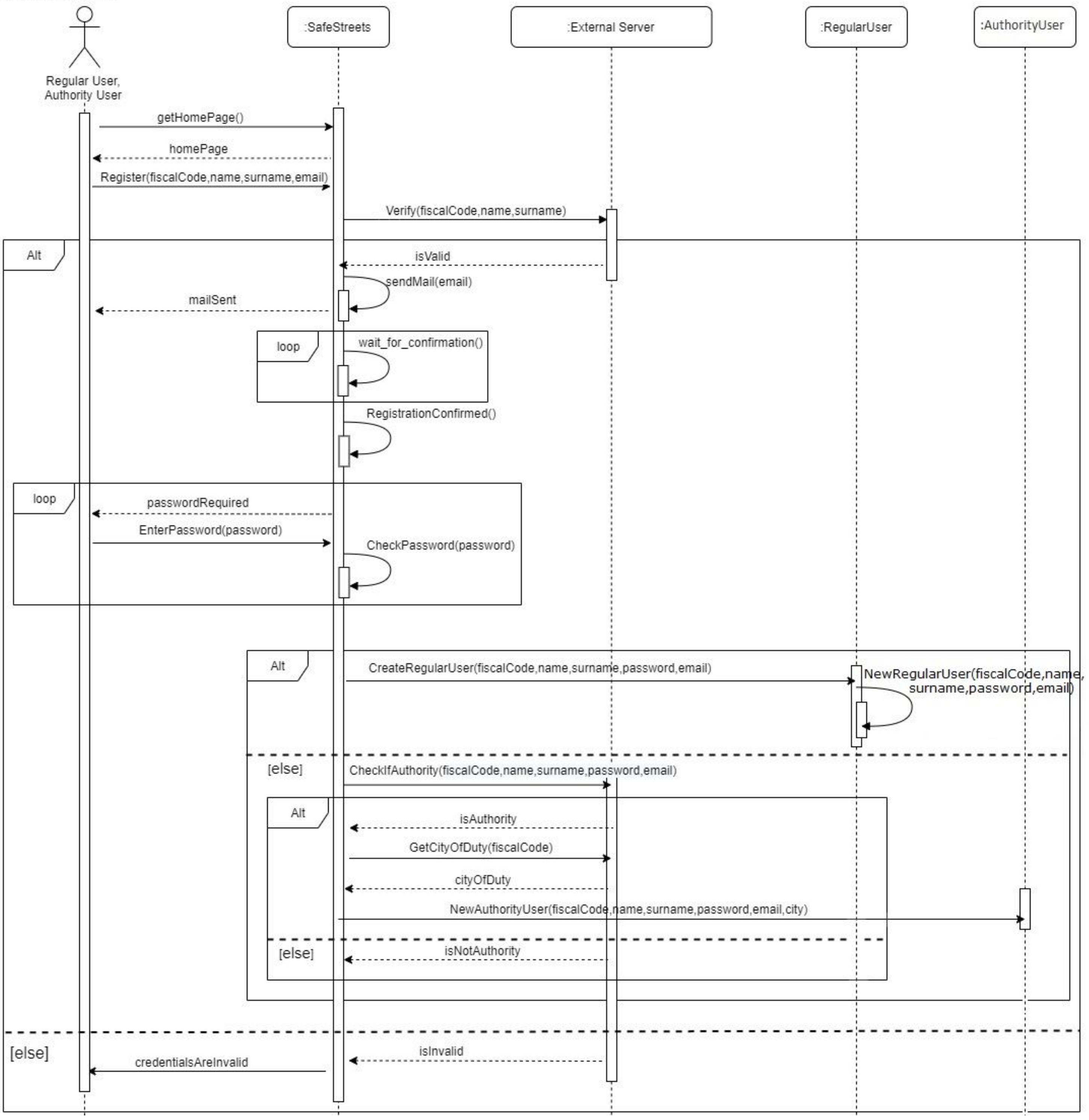


3.2.4.3. Authority User



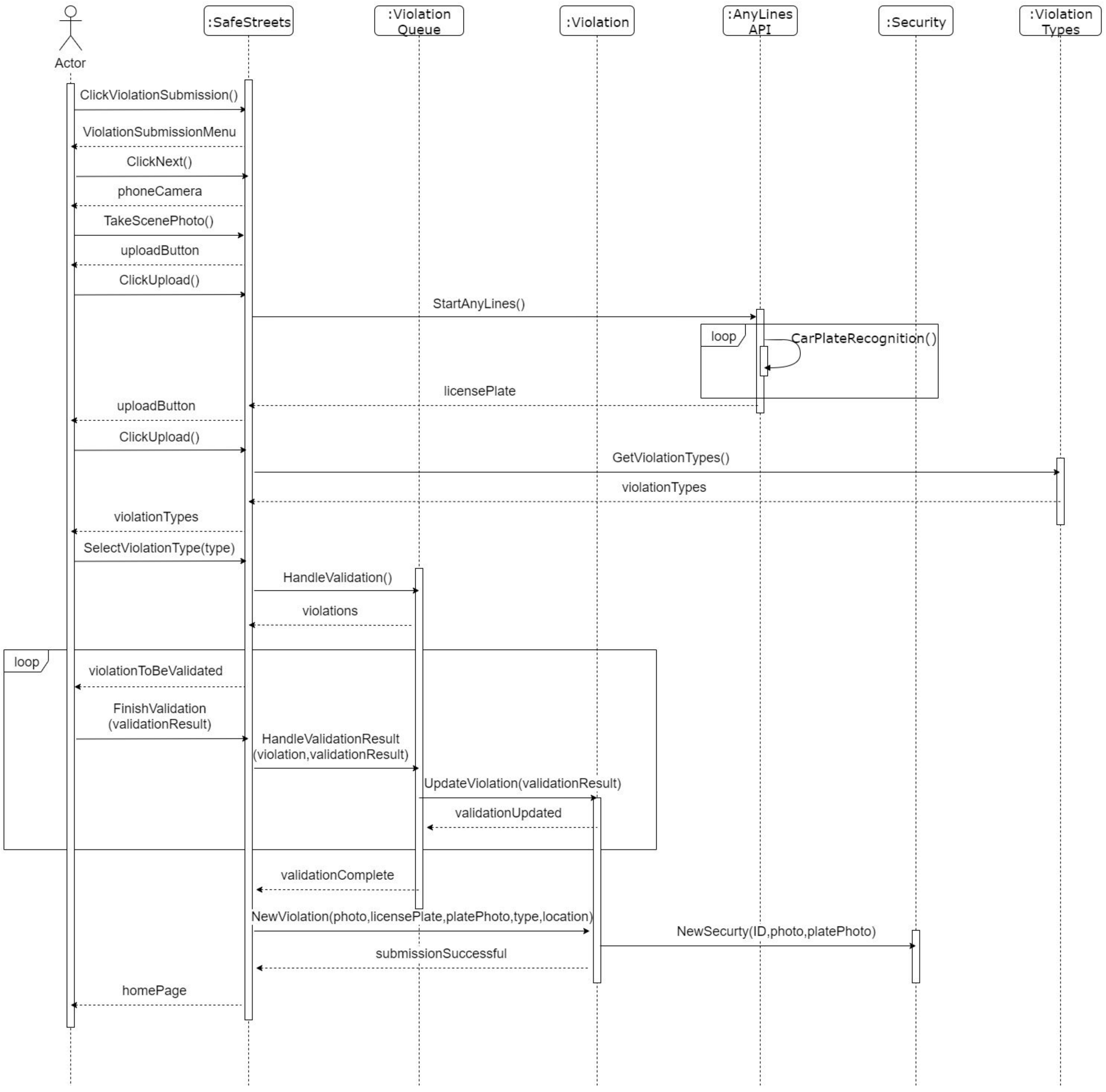
3.2.5. Sequence Diagrams

3.2.5.1. Registration

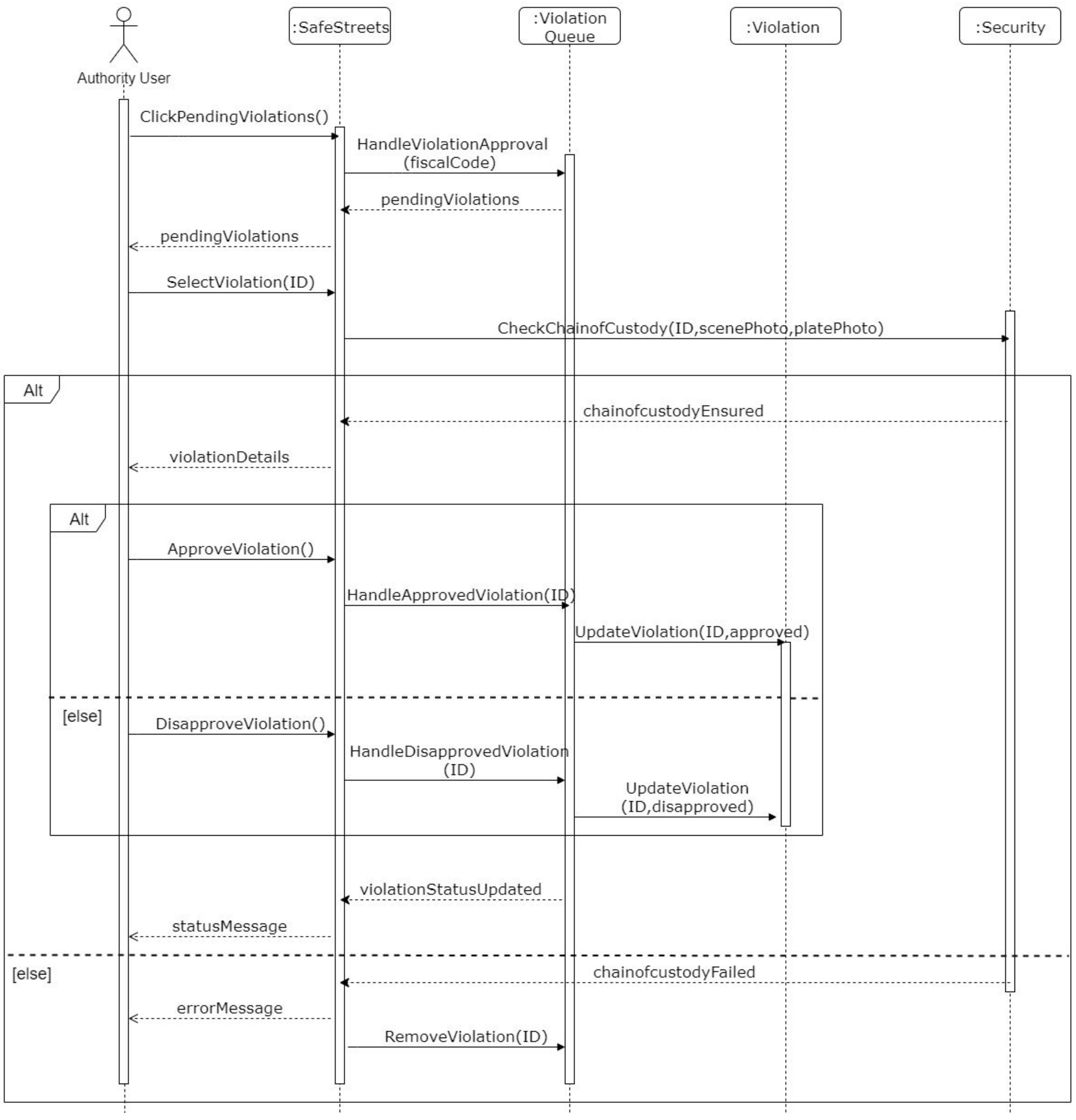


Regular User, Authority User
:SafeStreets
:External Server
:RegularUser
:AuthorityUser
getHomePage()
homePage
Register(fiscalCode,name,surname,email)
Verify(fiscalCode,name,surname)
Alt
isValid
sendMail(email)
mailSent
loop
wait_for_confirmation()
RegistrationConfirmed()
loop
passwordRequired
EnterPassword(password)
CheckPassword(password)
Alt
CreateRegularUser(fiscalCode,name,surname,password,email)
NewRegularUser(fiscalCode,name, surname,password,email)
[else]
CheckIfAuthority(fiscalCode,name,surname,password,email)
Alt
isAuthority
GetCityOfDuty(fiscalCode)
cityOfDuty
NewAuthorityUser(fiscalCode,name,surname,password,email,city)
[else]
isNotAuthority
[else]
credentialsAreInvalid
isInvalid

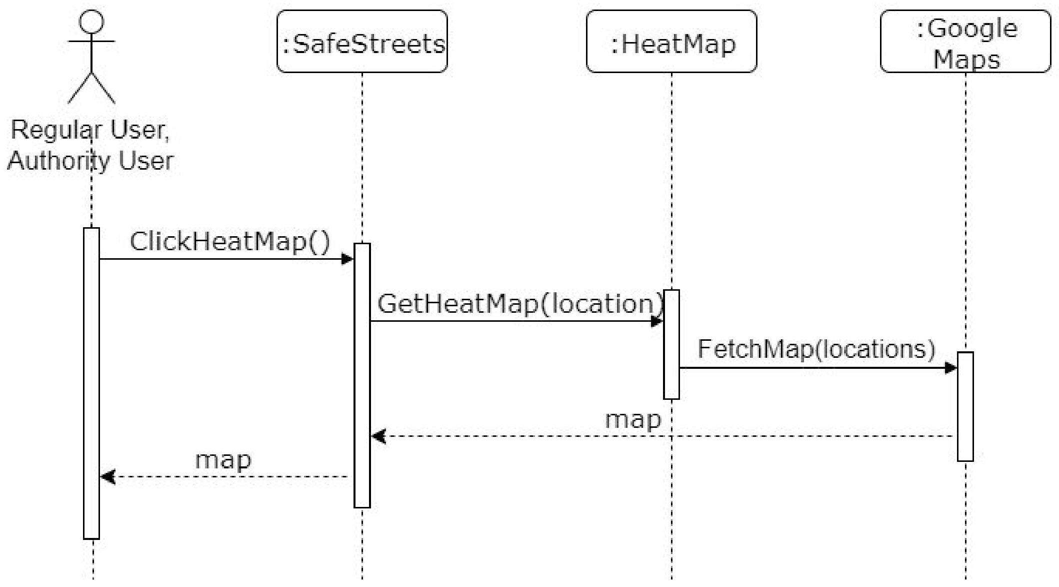
3.2.5.2. Submit Violation



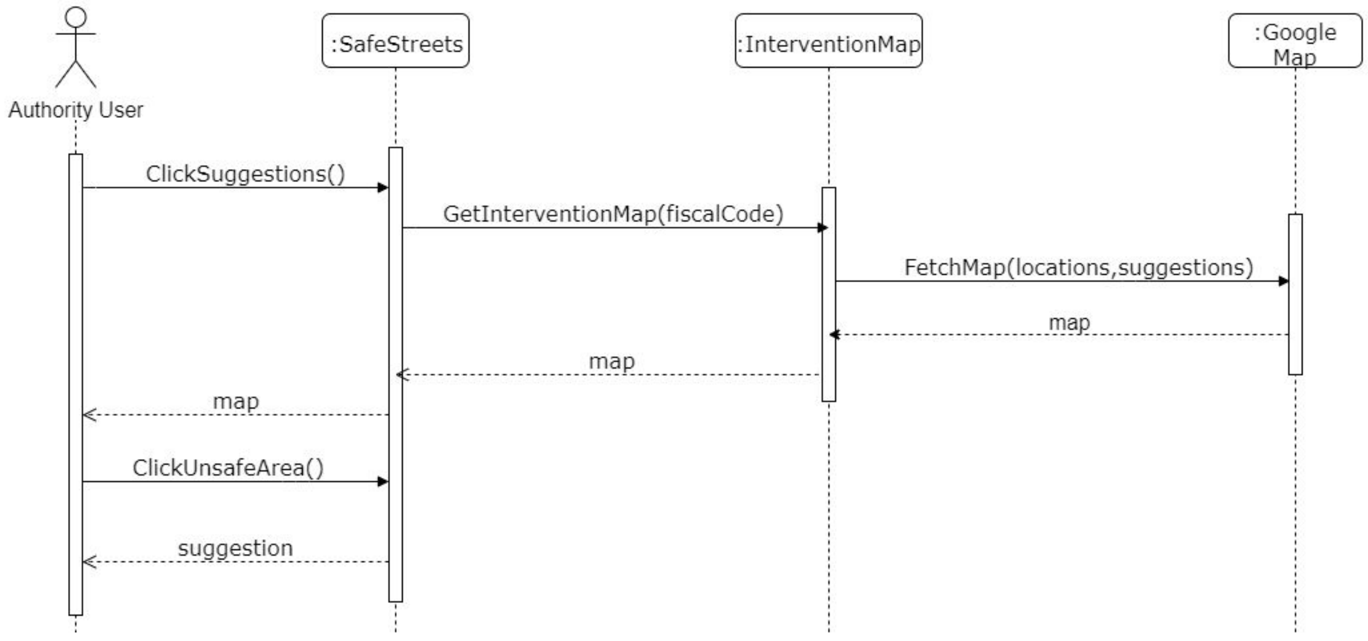
3.2.5.3. Approve or Disapprove Violation



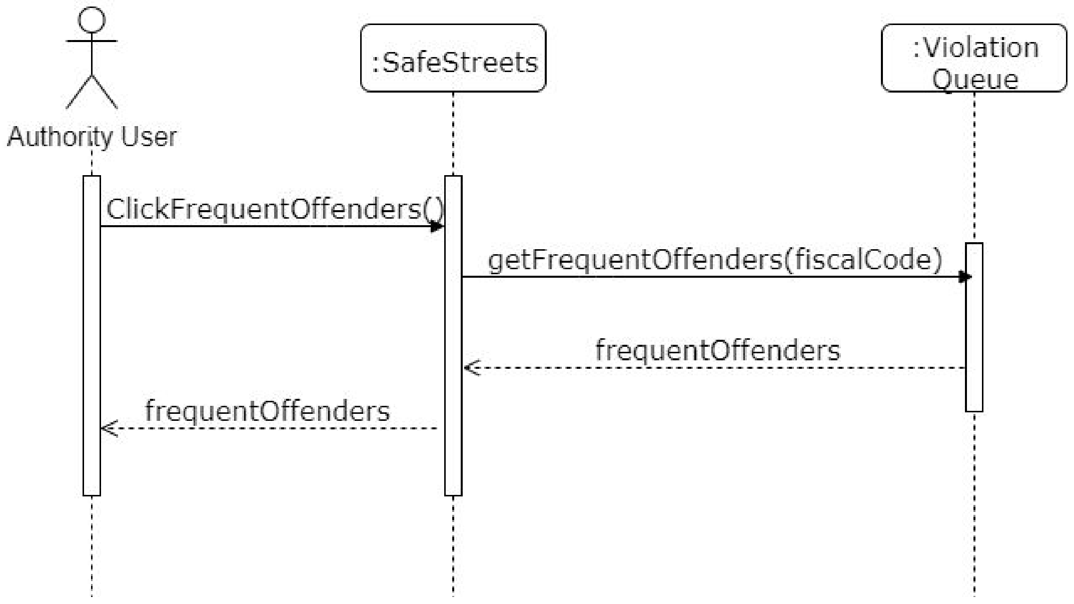
3.2.5.4. View Heatmap



3.2.5.5. View Traffic Intervention Suggestions



3.2.5.6. View the List of Frequent Offender



3.3. Non-Functional Requirements

3.3.1. Performance Requirements

The system has to be able to receive violation submissions simultaneously, while also updating the validation results of the existing submissions. Databases of the SafeStreets have to keep all violations along with their current states and should keep a pending queue for the view of authorities. Servers will be shared between the web application and the mobile application. On a study for a similar app, 53 submissions were received from 50 users in the span of one month. The system is expected to have around 50000 downloads for a city of approximately 8.5 million population and according to studies, applications of this download range have around 40% active users, which makes 20000 active users. When combined, data suggests an estimate around 21000 submissions for a single city per month which should be correctly handled by the application. Among these submissions around 20% is expected to be valid according to the studies, and with the user validation system of SafeStreets, application is expected to eliminate 80% of the faulty violations. This means around 36% of the submissions will pass to pending queue, so the queue has to be able to keep more than 7600 violations, which is the estimated monthly input.

3.3.2. Design Constraints

3.3.2.1. Standards Compliance

SafeStreets application's access to camera and location of the user's device will be solely used during traffic violation creation process. Private information of the users will be protected abiding the General Data Protection Regulation(2016), because the application is intended to be used within the European Union and European Economic Area.

3.3.2.1. Design Constraints

For regular users a mobile device which supports iOS or Android is necessary to access the application. iOS devices need to support iOS 10.0 or higher and require a camera with 1080p resolution. Android devices need to support Android SDK 19 or higher and require a camera with at least 720p resolution. The application also needs at least 2 GB RAM to function. The devices should also have an internet connection, preferably cellular connection (2G/3G/4G), and GPS. Authority users can access the application using any device which supports modern web browsers (e.g. Mozilla Firefox, Google Chrome, Opera, Microsoft Edge, Safari).

3.3.3. Software System Attributes

3.3.3.1. Reliability

In the case that the system is down, the data will be preserved. If there is an unexpected disconnection in the server, the Database will be read reachable only during the period that the system is down.

3.3.3.2. Availability

The system must guarantee 24/7 service. Very small variations can be acceptable beyond this requirement. The application will be available as a web application for authority users and it will be available as a mobile application for regular users.

3.3.3.3. Security

The system stores regular and authority user credentials, so the security of this information is a primary concern. The chain of custody between the time a regular user submits a violation and the time submitted violation reaches an authority user should also be protected. For this reason we store violation information in two separate arrays and these violations are associated using a violation ID and an encrypted violation ID. The security objects stores the pictures associated with each violation, namely violation photo and license plate photo, and information in this array can be linked to the associated violations by decrypting the encrypted key. When an authority user selects a pending violation from the pending queue, the submission on the main array's photos are compared with the security object's photos to ensure chain of custody.

3.3.3.4. Maintainability

The weekly and monthly updates of the maps instead of creating a new one for each access is designed to lighten the load on the system. Pending queue for each city will be designed to hold more than the estimated one month worth of traffic violation data to prevent overflows. API's which are used in the system are regularly updated following the new trends in technology, which will ease the SafeStreets application's maintainability.

3.3.3.5. Portability

The mobile application is designed for mobile phones, but it can be ported for any device that has internet access, GPS and a camera. The mobile application will be developed for iOS and Android platforms, and these platforms cover most if not all modern portable devices with the mentioned features. License plate scanning API relies on iOS and Android platforms, but it can be replaced with any other state of the art license plate recognition API in case the system to be ported does not support it. The web application can be accessed from any device which supports modern web browsers, and can be ported to work on upcoming browsers or updates easily.

4. Alloy Modeling

4.1. Signatures

```
open util/time
open util/integer

abstract sig Bool {}
one sig True extends Bool {}
one sig False extends Bool {}
one sig Uncertain extends Bool {}

sig Location {
	latitude: one Int,
	longitude: one Int
}
```

```
sig Image{}

sig ViolationType{
        typeName: one String
}

abstract sig User {
        fiscale: one String
}

sig RegularUser extends User {
        mailAddress: one String,
        location: Location one -> Time
}

sig AuthorityUser extends User {
        mailAddress: one String
}

sig Violation {
        id: one String,
        type : one ViolationType,
        location: one Location,
        scenePhoto: one Image,
        platePhoto: one Image,
        carPlate: one String,
        validation: Int one -> Time,
        isApproved: Bool one -> Time
}{
        all val : validation.Time | val =< 3 and val >= 0
}

sig SecurityViolation {
        id: one String,
        scenePhoto: one Image,
        platePhoto: one Image
}

sig UnsafeArea {
        location: one Location,
        type: one ViolationType,
        intervention: one String
}

one sig HeatMap {
        locations: set Location -> Time
}
```

```
one sig InterventionMap {
	unsafeAreas: set UnsafeArea
}

one sig OffendersMap {
	carPlates: set String -> Time,
}

one sig PendingViolations{
	violations: set Violation -> Time
}

one sig ViolationsToBeValidated{
	violations: set Violation -> Time
}

one sig ApprovedViolations{
	violations: set Violation -> Time
}
```

4.2. Facts

```
fact fiscaleUniqueness {
	no disjoint u1,u2: User | u1.fiscale = u2.fiscale
}

fact idUniqueness {
	no disjoint v1,v2: Violation | v1.id = v2.id
	no disjoint s1,s2: SecurityViolation | s1.id = s2.id
	all v: Violation, s: SecurityViolation | v.id != s.id
}

fact photoDifference {
	all v: Violation | v.scenePhoto != v.platePhoto
}

fact imageUniqueness {
	no disjoint v1,v2: Violation | v1.scenePhoto = v2.scenePhoto or v1.scenePhoto =
v2.platePhoto or v1.platePhoto = v2.platePhoto
}

fact unsafeAreaUniqueness {
	no disjoint u1,u2: UnsafeArea | u1.location = u2.location
}

fact typeUniqueness {
	no disjoint t1,t2: ViolationType | t1.typeName = t2.typeName
}
```

```
fact areaToViolation {
	all ua: UnsafeArea | some v: Violation | ua.type = v.type
}

fact mailDifference {
	all au: AuthorityUser, ru: RegularUser | au.mailAddress != ru.mailAddress
}

fact ViolationState{
	all v: Violation | one t': Time | v.validation.t' = 0 and v.isApproved.t' = Uncertain
	all v: Violation, t: Time | (v.validation.t = 1 => all t': Time | gte[t',t] => v.validation.t' !=
0)
	all v: Violation, t: Time | (v.validation.t = 2 => all t': Time | gte[t',t] => (v.validation.t' =
2 or v.validation.t' = 3))
	all v: Violation, t: Time | (v.validation.t = 3 => all t': Time | gte[t',t] => v.validation.t' =
3)
	all v: Violation, t: Time | (v.isApproved.t = True => (v. validation.t = 3 and (all t': Time
| gte[t',t] =>v.isApproved.t' = True)))
	all v: Violation, t: Time | (v.isApproved.t = False => all t': Time | gte[t',t]
=>(v.isApproved.t' = False and v.validation.t' = v.validation.t))
}

fact Security{
	all s: SecurityViolation | one v:Violation | s.scenePhoto = v.scenePhoto and
s.platePhoto = v.platePhoto
	all v:Violation | one s: SecurityViolation | s.scenePhoto = v.scenePhoto and
s.platePhoto = v.platePhoto
}

fact HeatMapState{
	all t: Time, v: Violation, h: HeatMap |  v.location in h.locations.t => v.isApproved.t =
True
	all t: Time, h: HeatMap | all loc: h.locations.t | some v:Violation | loc = v.location
}

fact OffendersState{
	all t: Time, v: Violation, o: OffendersMap | v.carPlate in o.carPlates.t <=>
v.isApproved.t = True
}

fact ValidationSet{
	all t: Time, v: Violation, m: ViolationsToBeValidated | v in m.violations.t <=>
v.validation.t < 3 and v.isApproved.t = Uncertain
}

fact PendingSet{
	all t: Time, v: Violation, m: PendingViolations | v in m.violations.t <=> v.validation.t =
3 and v.isApproved.t = Uncertain
}
```

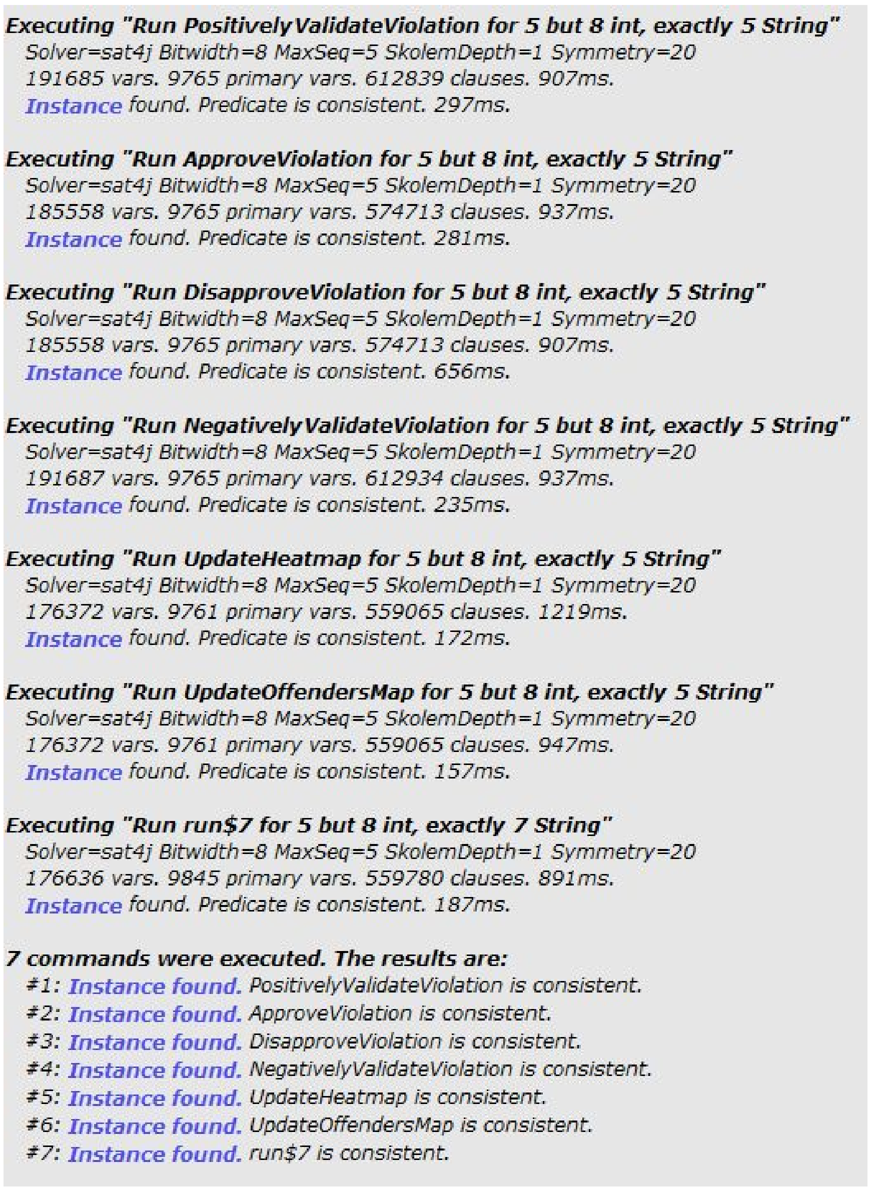
```
fact ApprovedSet{
	all t: Time, v: Violation, m: ApprovedViolations | v in m.violations.t <=>
v.isApproved.t = True
}
```

4.3. Predicates

```
pred isViolationBeingValidated[v: Violation, t: Time]{
	//v.validation.t < 3 and v.isApproved.t = Uncertain
	one m: ViolationsToBeValidated | v in m.violations.t
}

pred isViolationPending[v: Violation, t: Time]{
	//v.validation.t = 3 and v.isApproved.t = Uncertain
	one m: PendingViolations | v in m.violations.t
}

pred isViolationApproved[v: Violation, t: Time]{
	//v.isApproved.t = True
	one m: ApprovedViolations | v in m.violations.t
}

pred PositivelyValidateViolation[v: Violation, t: Time]{
	isViolationBeingValidated[v,t]
	v.validation.(t.next) = v.validation.t + 1
	v.isApproved.(t.next) = Uncertain
}

pred NegativelyValidateViolation[v: Violation, t: Time]{
	isViolationBeingValidated[v,t]
	v.validation.(t.next) = v.validation.t
	v.isApproved.(t.next) = False
}

pred ApproveViolation[v: Violation, t: Time]{
	isViolationPending[v,t]
	v.isApproved.(t.next) = True
}

pred DisapproveViolation[v: Violation, t: Time]{
	isViolationPending[v,t]
	v.isApproved.(t.next) = False
}

pred UpdateHeatmap[h:HeatMap, t: Time]{
	all v: Violation | isViolationApproved[v,t] => v.location in h.locations.t
}
```

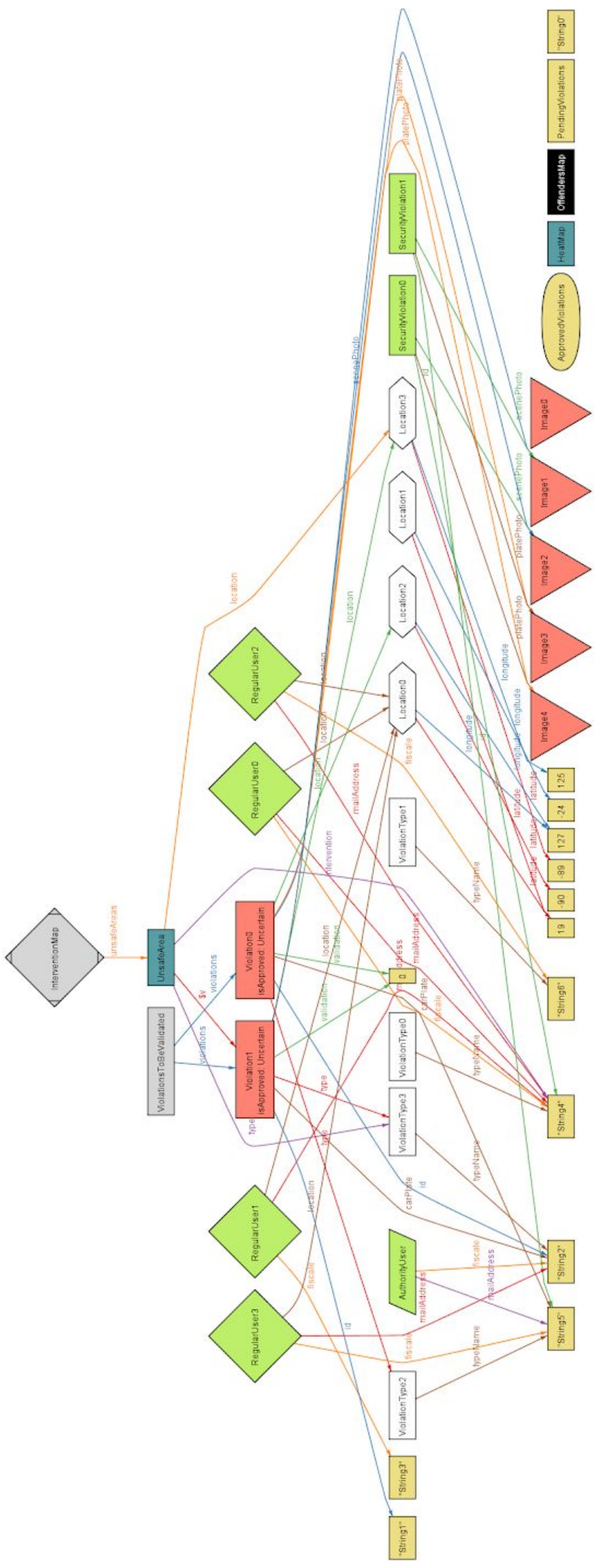
```
pred UpdateOffendersMap[o:OffendersMap, t: Time]{
	all v: Violation | isViolationApproved[v,t] => v.carPlate in o.carPlates.t
}

run PositivelyValidateViolation for 5 but 8 Int, exactly 5 String
run ApproveViolation for 5 but 8 Int, exactly 5 String
run DisapproveViolation for 5 but 8 Int, exactly 5 String
run NegativelyValidateViolation for 5 but 8 Int, exactly 5 String
run UpdateHeatmap for 5 but 8 Int, exactly 5 String
run UpdateOffendersMap for 5 but 8 Int, exactly 5 String
run {#Violation = 2} for 5 but 8 Int, exactly 7 String
```

4.4. Proof of Consistency

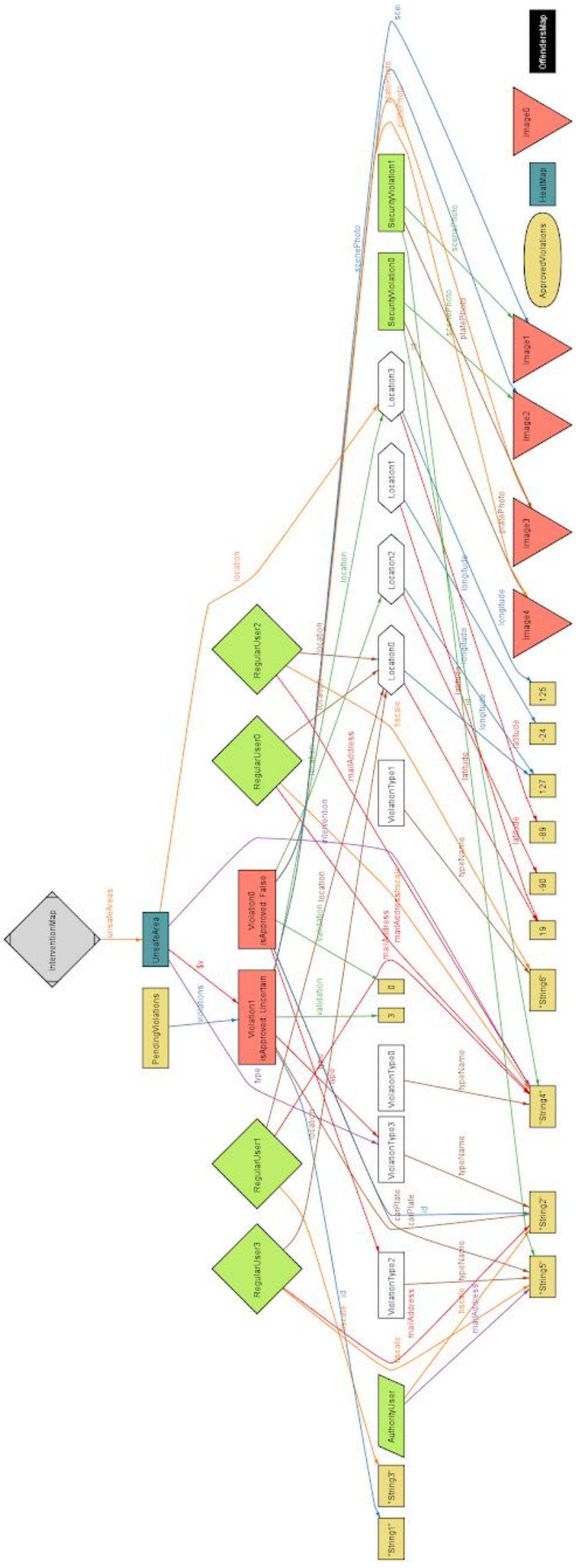


```
Executing "Run PositivelyValidateViolation for 5 but 8 int, exactly 5 String"
   Solver=sat4j Bitwidth=8 MaxSeq=5 SkolemDepth=1 Symmetry=20
   191685 vars. 9765 primary vars. 612839 clauses. 907ms.
   Instance found. Predicate is consistent. 297ms.

Executing "Run ApproveViolation for 5 but 8 int, exactly 5 String"
   Solver=sat4j Bitwidth=8 MaxSeq=5 SkolemDepth=1 Symmetry=20
   185558 vars. 9765 primary vars. 574713 clauses. 937ms.
   Instance found. Predicate is consistent. 281ms.

Executing "Run DisapproveViolation for 5 but 8 int, exactly 5 String"
   Solver=sat4j Bitwidth=8 MaxSeq=5 SkolemDepth=1 Symmetry=20
   185558 vars. 9765 primary vars. 574713 clauses. 907ms.
   Instance found. Predicate is consistent. 656ms.

Executing "Run NegativelyValidateViolation for 5 but 8 int, exactly 5 String"
   Solver=sat4j Bitwidth=8 MaxSeq=5 SkolemDepth=1 Symmetry=20
   191687 vars. 9765 primary vars. 612934 clauses. 937ms.
   Instance found. Predicate is consistent. 235ms.

Executing "Run UpdateHeatmap for 5 but 8 int, exactly 5 String"
   Solver=sat4j Bitwidth=8 MaxSeq=5 SkolemDepth=1 Symmetry=20
   176372 vars. 9761 primary vars. 559065 clauses. 1219ms.
   Instance found. Predicate is consistent. 172ms.

Executing "Run UpdateOffendersMap for 5 but 8 int, exactly 5 String"
   Solver=sat4j Bitwidth=8 MaxSeq=5 SkolemDepth=1 Symmetry=20
   176372 vars. 9761 primary vars. 559065 clauses. 947ms.
   Instance found. Predicate is consistent. 157ms.

Executing "Run run$7 for 5 but 8 int, exactly 7 String"
   Solver=sat4j Bitwidth=8 MaxSeq=5 SkolemDepth=1 Symmetry=20
   176636 vars. 9845 primary vars. 559780 clauses. 891ms.
   Instance found. Predicate is consistent. 187ms.

7 commands were executed. The results are:
   #1: Instance found. PositivelyValidateViolation is consistent.
   #2: Instance found. ApproveViolation is consistent.
   #3: Instance found. DisapproveViolation is consistent.
   #4: Instance found. NegativelyValidateViolation is consistent.
   #5: Instance found. UpdateHeatmap is consistent.
   #6: Instance found. UpdateOffendersMap is consistent.
   #7: Instance found. run$7 is consistent.
```

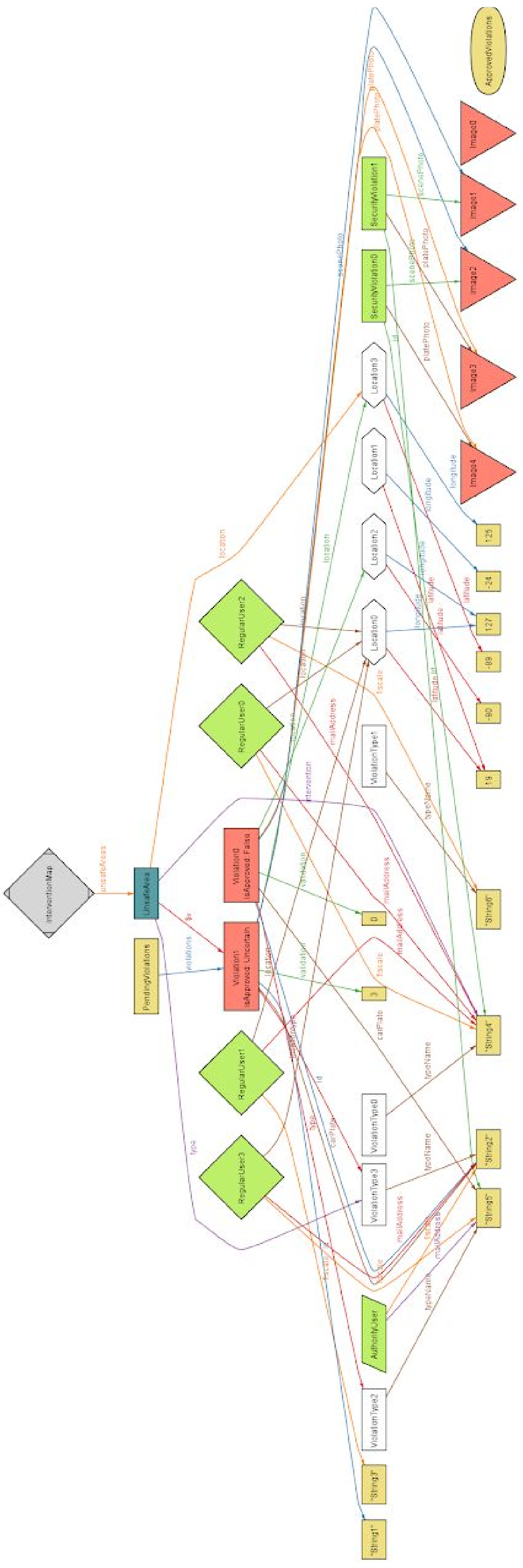
4.5. Generated Dynamic World



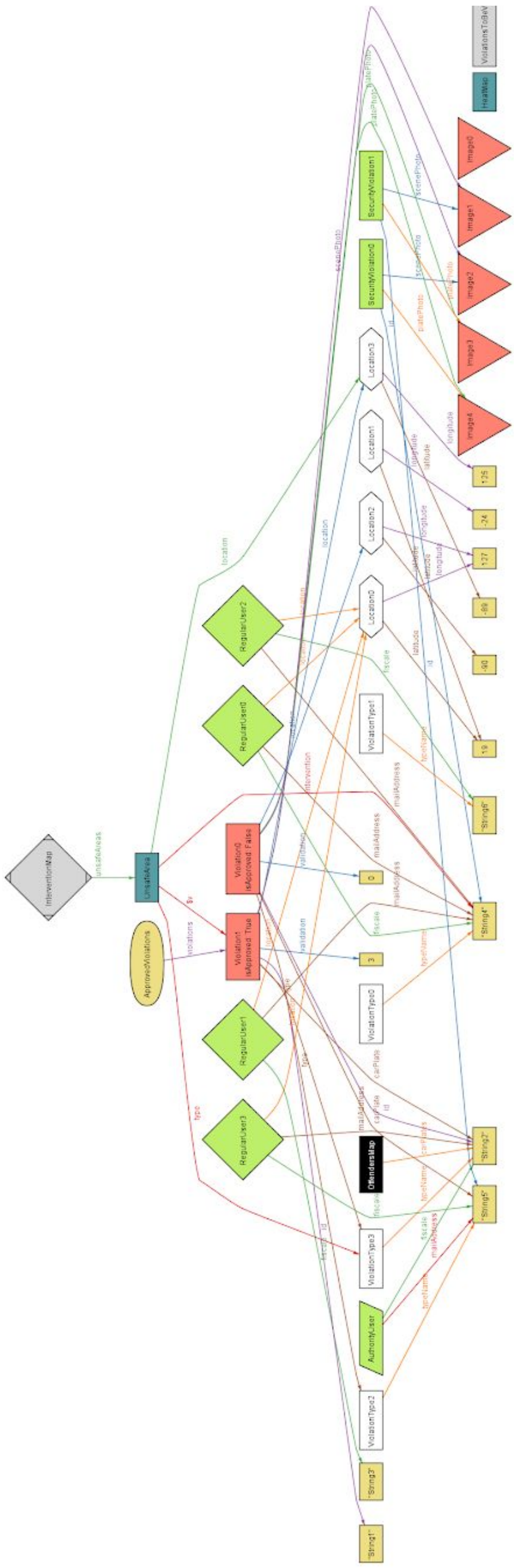
Dynamic World at time 0



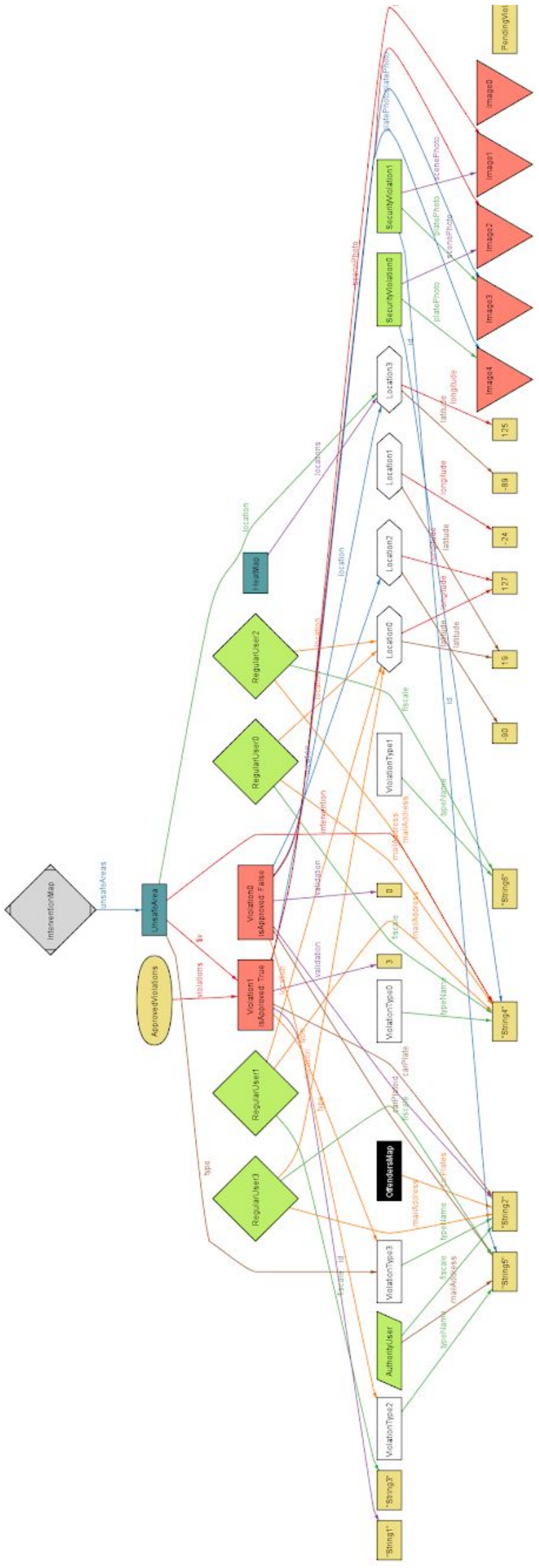
Dynamic World at time 1



Dynamic World at time 2



Dynamic World at time 3



Dynamic World at time 4

5. Appendix

5.1. Tools Used

- Alloy Analyzer 4.2
- Adobe InDesign
- Draw IO Application: www.draw.io

5.2. Effort Spent

Description of the task	**Hours**
Introduction	6
Product perspective	6
Product functions	4
Domain assumptions	8
External interface requirements	8
Functional requirements	16
Non-functional requirements	4
Formal analysis using Alloy	16