POLITECNICO

MILANO 1863

Software Engineering 2

**Requirements Analysis
and Specification
Document**

Amorosini Francesco
Casali Alice
Fioravanti Tommaso

10 November 2019

Contents

Introduction

This document represents the Requirement Analysis and Specification Document (RASD) for SafeStreets: an application that aims to improve the safety of urban areas by giving its users the possibility to report traffic violations to authorities. The goal of this document is to supply a description of the system in terms of its functional and non-functional requirements, listing all of its goals, discussing the constraints and the limits of the software, and indicating the typical use cases that will occur after the release. This document is addressed to the stakeholders, who will evaluate the correctness of the assumptions and decisions contained in it, and to the developers who will have to implement the requirements.

1.1   Purpose

SafeStreets is a crowd-sourced application that intends to provide users with the possibility to notify authorities when traffic violations occur. In order to report a traffic violation, users have to compile a report containing a picture of the violation, the date, time, position, and type of violation. SafeStreets stores all the information provided by users, completing it with suitable metadata (timestamps, sender's GPS position, etc.). In particular, in case the user had not provided any information regarding the license plate of the car breaking the traffic rules, SafeStreets runs an algorithm to read it from the submitted picture. Moreover, the application allows both citizens and authorities to check some useful statistics: in particular, SafeStreets is able to analyze the recieved information to build statistics and give suggestions to improve the safety of urban areas. If the municipality offers a service that allows users to retrieve the information about the accidents that occur in its territory, SafeStreets can cross this information with its own data. Independently from Safestreets, the municipality could offer a service that takes the information about the violations coming from SafeStreets, and generates traffic tickets from it. In this case, SafeStreets ensures that the chain of custody of information coming from users is never broken, thus information is never altered. Information about issued tickets can be used by SafeStreets to build statistics, for example to have a

feedback on the effectiveness of the application.

1.1.1 Goals

- [G.1] Allow the Users to access the functionalities of the application from different locations and devices.

- [G.2] Allow Guests to authenticate either as Authority or Citizen.

- [G.3] Allow the User to document traffic violations and accidents by compiling a Report.

- [G.4] Store the reports provided by the Citizens.

  - [G.4.1] If the Citizen has not provided any information about the license plate, the System runs an algorithm to read it from the submitted picture.

- [G.5] Allow Authorities to consult the reports submitted by the Citizens.

- [G.6] Analyze the stored information to build statistics.

  - [G.6.1]If allowed by the Municiplaity, SafeStreets can cross their information about accidents with its own in order to improve the analysis.

- [G.7] Detect unsafe areas through statistics.

  - [G.7.1]Suggest possible interventions to the Authorities.

- [G.8]Build statistics on iussued tickets if the Municiplaity allows SafeStreets to access such data.

  - [G.8.1] Ensure that the chain of custody of the information coming from the Citizens is never broken.

- [G.9] Allow the Users to consult statistics.

  - [G.9.1] Offer different levels of visibility to different roles.

1.2 Scope

According to the *World* and *Machine* paradigm, proposed by M. Jackson and P. Zave in 1995, we can distinguish the *Machine*, that is the portion of the system to be developed, from the *World*, that is the portion of the real-world affected by the machine. In this way, we can classify the phenomena in three different types: World, Machine and *Shared* phenomena, where the latter type of phenomena can be controlled by the world and observed by the machine or controlled by the machine and observed by the world:
In this context, the most relevant phenomena are organized as in the following table.

World	Shared	Machine
Traffic violation: circumstance in which someone doesn't respect the traffic rules	**Registration/Login**[1]: a Guest can sign up to the application or log in if already registered.	**DBMS query**: operation performed to retrieve/store data
Accident: event that caused damage to things or people.	**Commit of a report**[1]: action of sending a report documenting a traffic violation or an accident.	**API queries**: request for third-part services.
	Send notifications[2]: to notify the Authorities when a report is submitted.	**Encrypt data**: to compute the hash function to ensure that reports are never altered and that user's credentials are safe.
	Build statistics[2]: computation of statistics to find unsafe areas, to see which veichles commit multiple infractions and also to monitor the trend of issued tickets.	**Role Isolation**: to grant different levels of visibility to Citizen and Authorities
	Safety Suggestionss[2]: to give suggestions for possible safe interventions (e.g add a barrier).	

Table 1.1: In the table above, *1* refers to shared phenomena controlled by the world and observed by the machine, whereas *2* refers to the phenomena controlled by the machine and observed by the world

.

1.3 Definitions, Acronyms and Abbreviations

1.3.1 Definitions

- *System*: the totality of the hardware/software applications that contribute to provide the service concerned.

- *Guest*: someone who has yet to sign up and who is not able to access any feature of the application.

- *User*: a registered user who has logged in.

  - *Citizen*: user who can send Reports of traffic violation. It has limited visibility over the stored information.
  - *Authority*: user who has access to the history of Reports and is notified whenever a new Report is recieved. It has full visibility over the data exposed by the System.

- *Municipality*: a single administrative division with its own local government. It stores sensitive information about the Authorities and their work (g.e. traffic tickets).

- *Traffic violation*: occurrence of drivers that violate laws that regulate vehicle operation on streets and highways(e.g double-way parking).

- *Safety Suggestion*: recommendation to improve the safety of urban areas. It is computed by analyzing the statistics built by the system.

- *Report*: documentation of a traffic violation. It contains a picture of the violation, the date, time, position, and type of violation. Some fields can be omitted in case an accident is being reported.

- *Hash*: a mathematical algorithm that maps data of arbitrary size to a bit string of fixed size. It is a one-way function, that is, a function which is practically infeasible to invert.

- *Chain of custody*: chronological documentation that records the sequence of custody, control, transfer, analysis, and disposition of physical or electronic evidence.

- *Third-party services*: services used by the System in order to provide extra functionalities (e.g. image recognition).

1.3.2 Acronyms

- **API**: *Application Program Interface*, set of routines, protocols and tools for building software applications on top of this one.

- **OCR**: *Optical Character Recognition*, software dedicated to the detection of characters contained in a document and to their transfer to digital text that can be read by a machine. In this context, OCR will be used to read license plates.

- **UML**: *Unified Modeling Language*, is a standard visual modeling language intended to be used for analysis, design, and implementation of software-based systems.

- **GPS**: *Global Positioning System*, technology widely used to get the user's position.

- **DBMS**: *Data Base Management System*, software that provides organized space memory to store information.

- **ID**: *Identification*, a unique key given to each registered User.

1.3.3   Abbreviations

- [G_i]: i-th goal.

- [R_i]: i-th requirement.

- [D_i]: i-th domain assumption.

1.4   Revision History

Date	Version	Changes
10.11.2019	1.0	First release
22.11.2019	1.1	- UML Class Diagram adjustments: added StorageManager and other minor changes. - Goal mapping is now explicit in Use Case Templates. - General review for achieving consistency with the Design Document.

1.5 Reference Documents

- Specification document: *SafeStreets Mandatory Project Assignment.pdf.*

- IEEE std 830-1998: *IEEE Recommended Practice for Software Requirements Specifications.*

- ISO/IEC 9126-1:2001: *Software engineering — Product quality — Part 1: Quality model*

- Alloy docs: `http://alloy.lcs.mit.edu/alloy/documentation.html`.

- UML diagrams: `https://www.uml-diagrams.org`.

1.6 Document Structure

This document is presented as it follows:

1. **Introduction** contains a general description of the system and its goals, presenting the problem in an informal way.

2. **Overall description** gives a general description of the application, focusing on the context of the system and giving further details about shared phenomena. Furthermore, we will provide the specifications of constraints, dependencies and assumptions, that show how the System is integrated with the real world.

3. **Specific requirements** goes into details about functional and non functional requirements and a list of all possible interctions with the System is provided using use cases and sequence diagrams.

4. **Formal analysis using Alloy** contains the Alloy model of some critical aspects of the System with all the related comments. A proof of consistency and an example of the generated world are also provided.

5. **Effort spent** shows the time spent writing this document by each member of the team.

Overall Description

2.1   Product Perspective

We are going to analyze all the shared phenomena that are listed in the previous section. The concepts are clarified through class and state diagrams.

Registration/Login (World controlled, Machine observed)
A Guest can sign up to the application or log in, if already registered. In the case of a new submit, the System provides a form which the Guest have to fill with his data, specifying if he/she is a Citizien or an Authority.

Commit of a report (World controlled, Machine observed)
A Citizen can send a Report with a description of the violation compiling a structured field. He/she has the possibility to attach a picture. The System will check if the compiled fields are consistent and complete, then it will encrypt the Report and send it to the Storage Manager (Figure 2.1).

Send Notification (Machine controlled, World observed)
The System notifies the Authorities whenever a new Report is submitted by a Citizen. We assume that the notified Authorities will accordingly handle the violation reported. (Figure 2.1).

Figure 2.1: SafeStreets' statechart diagram about the collection, notification, and storage of a Report.

Build statistics (Machine controlled, World observed)
The System analyzes the stored data and computes general statistics on the violations. In this way, the System is able to find unsafe areas and which vehicles commit the most violations. All Users can access general statistics, however only Authorities can request statistics on private information such as traffic tickets. If the System is allowed to access Municipality's data, statistics can be enhanced by crossing it with SafeStreets' information (Figure 2.2).

Safety Suggestionss (Machine controlled, World observed)
The System gives Authorities suggestions for possible safe interventions. His knowledge is based on the statistics previously computed (Figure 2.2).

Figure 2.2: SafeStreets' statechart diagram of the creation of statistics on unsafe areas.

2.1.1 Class Diagram

The class diagram in Figure 2.3 shows a possible representation of the Safe Streets' architecture. Notice that the Municipality's database is only accessible by the System if SafeStreets is authorized by the Municipality Manager. It's also important to outline that the System is not designed to provide a function to generate official traffic tickets, since only real world authorities (g.e. local police) can generate traffic tickets with legal validity.
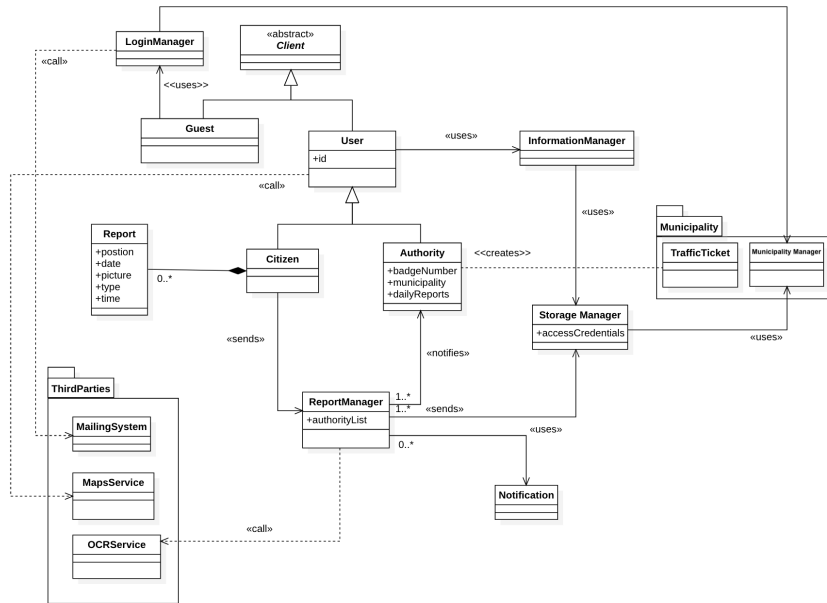
Figure 2.3: SafeStreets' class diagram.

2.2 Product Functions

In this section the most important features of the application are extensively explained:

- **Commit of a Report**: the System allows a Citizen to notify Authorities whenever a traffic violations occur. This function is achieved by compiling a Report with a picture of the violation, the date, time, position, and type of violation. Not all of these fields are always compulsory: for example, in the case the type of violation is *accident*, only date, time and position are required. Once all the information is gathered and the report is sent to SafeStreets, the Report Manager goes in the `Verifying Information` status: it checks if all the obligatory fields have been compiled correctly, and then adds some useful metadata to the Report. Among the additional metadata, the systems takes note of whether or not the position has been provided by GPS, and also if the sender has added any license plate in the `Checking License Plate` status. If no license plate has been inserted, the Report Manager runs an OCR algorithm to retrive it from the picture provided (`Running OCR` status).
  After all the steps above, the Report is validated (`Validating Report` status) and a copy of it is notified to the Authorities, which we assume are going to handle it properly (g.e. dispatching a local police patrol). It

is important to outline that the System is only designed to inform Authorities whenever a new report is confirmed, but by no means it provides a function which generates official traffic tickets or dispatches patrols in case of accidents. Also, it's relevant to notice that the Report Manager contains a list of all the Authorities, so it can notify all and only those Authorities who are on duty in the involved municipality. Finally, the Report is encrypted by means of a hash function (`Encrypting Report` status), thus no one can alter the information after its confirmation (see Section 3.5.3 for more details on security).

Once the Report has been validated and encrypted the System is ready to store it: the Report is sent to the Storage Manager, where it's stored into a local buffer and there awaits to be exported into SafeStreets' database. Each day at a set time (g.e. between day/night shifts of the police) the System permanently stores all the Reports into the database. This mechanism takes advantages of the fact that one does not need fresh data to do general statistics, and in return it gives the possibility to schedule the Report storage when the System is not busy.

- **Build statistics**: the System analyzes the stored data to compute general statistics and returns useful information to the Users. This function is carried out by an Information Manager, which is in charge of retrieving, decrypting and elaborating the information available to the Storage Manager. In particular, the Information Manager periodically executes the following functions:

  1. Compute a statistic which highlights the areas and vehicles with the highest frequency of violations.

  2. Identify potentially unsafe areas by analyzing Reports of accidents, and suggest possible interventions.

  3. Analyze traffic ticket trends to build statistics about the most recurrent offenders and the effectiveness of SafeStreets.

  Every User can request to see the latest result of the functions above as soon as they are available. Since both Citizen and Authorities have access to those results, it's really important to give them different levels of visibility, so that classified information doesn't end up in the hands of civilians (g.e vehicles with the highest frequency of violations and most recurrent offenders). It is also worth noting that some of the above functions are enhanced, or even enabled, only if the Municipality grants the Information Manager access to their database. In particular, the Function 2 can be enhanced by crossing Authorities' accident information with SafeStreets', whereas Function 3 is entirely enabled only if the System has access to traffic tickets data.

- **Safety Suggestionss**: the System is able to give Authorities suggestions to improve the safety of urban areas. This function is enabled by the Information Manager and the statistics previously built. In fact, observing

for each area which kind of violations are more frequent and the trends of the accidents, the Information Manager runs an artificial intelligence to detect which solution is the most suitable for a given problem. Just like for the report storing process, these suggestions are updated when the System is less busy (g.e. at night) and a sufficient amount of fresh data is added to the database. Safety Suggestionss are downloaded by SafeStreets' application togeter with the latest statistics results, whenever they are requested. Despite this, it should be noted that only Authorities have access to such data, as they are assumed to be those who implement the suggestions in order to improve their service.

2.3 User Characteristics

SafeStreets can be used from both Citizens and Authorities. It is recommended for adult users, but there are no limitation of age. We take for granted that the Users have access to Internet and are able to install and use the mobile application.
The character are:

- *Guest*: anyone who downloads and opens the app but still has to sign up or log in. He/she cannot use any of the functions provided by SafeStreets.

- *User*: a registered Guest who has to logged either as Citizen or as Authority. He/She is recognized by the System through an ID and can access any feature of the application granted for his/her role.

- *Citizen*: someone who has logged-in with his credentials, which is recognized by the System through an ID. He/she can access to the personal data menu and add new reports.

- *Authority*: someone who has logged-in with his credentials. He/she is notified every time a new Report regarding its municipality is stored by the System.

2.4 Assumptions, Dependencies and Constraints

2.4.1 Assumptions

- [D.1] The registration mail is assumed to be correctly recieved by the User.

- [D.2] The System internal clock time used to provide notifications is correct.

- [D.3] When a little accident occurs there is no need to contact the Authorities, whereas when a dangerous accident happens Users are supposed to phone call the Authorities for a promptly intervention.

- [D.4] The Authorities are assumed to take measures when they are notified about a new report.

- [D.5] After the first time, the System does not need to request the access to the Municipality's database as long as its permission is valid.

- [D.6] The statistics and suggestions are assumed to be consulted by Authorities periodically, in order to provide a better service.

- [D.7] Only specific components of the System is able to decrypt sensible data.

- [D.8] SafeStreets is not responsible of the traffic tickets generated by Authorities.

2.4.2 Dependencies

- A DBMS is needed in order to store and retrieve Users' Reports.

- Information on issued tickets (and possibly accidents) are provided by the Municipalities' database, which has limited access.

- The OCR tool is provided by external thrid party services.

- An external Mailing Service is used to send confirmation mails.

- The Map service is provided by external an third party (which can be different by the one that provides the OCR tool).

- The system relies on an AI to analyze the data, build statistics and compute suggestions.

2.4.3 Constraints

- Smartphone equipped with an OS compatible with the application.

- Properly working Internet connection.

- The permission to acquire and store personal data must be given by the User.

- Offer the possibility to the Users to delete their account.

- It is impossible to know if the tickets are generated thanks to SafeStreets, or if they are independent from the service.

Specific Requirements

3.1 External Interface Requirement

SafeStreets is a mobile based application, even though it can be also consulted using a web browser. In the following sections a more detailed description of the application is given in terms of hardware, software and communication interfaces. There are also present some prototypes of User Interface for mobile users, by means of mockups.

3.1.1 User Interface
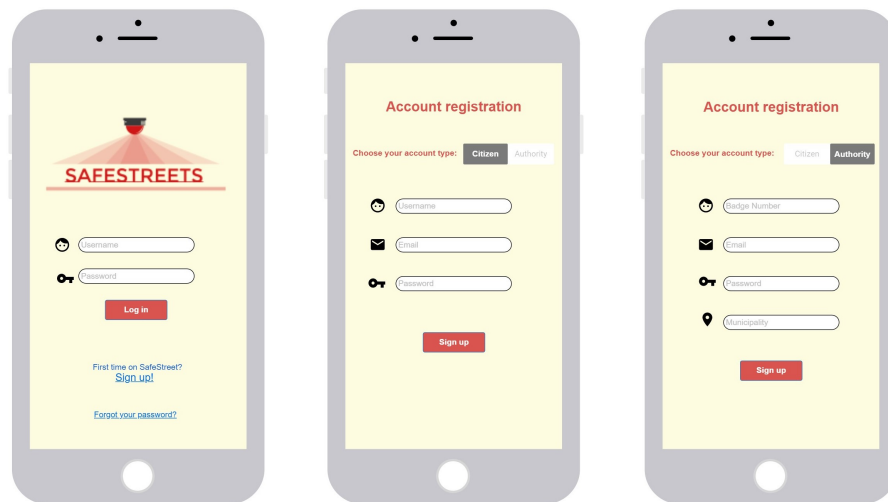
Figure 3.1: Guest signup interface and login interface.
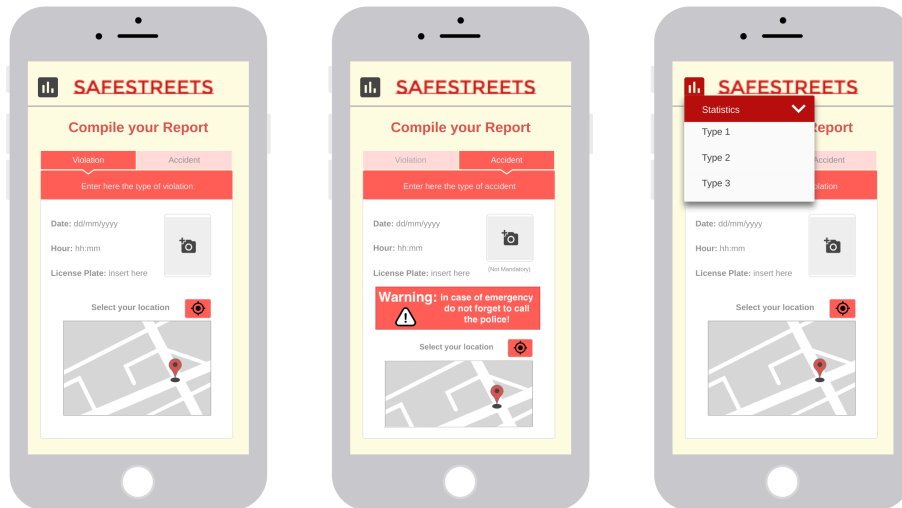
Figure 3.2: Citizens interfaces to make new report (violation or accident), and to consult statistics menu.

Figure 3.3: Authorities interface to see new report, to consult statistics and suggestions menu.

Figure 3.4: Example of statistic and suggestion interfaces.

3.1.2 Hardware Interface

The application is avaiable for mobile devices that guarantee Internet access with a reasonably recent browser and have a camera with a fairly high definition.

3.1.3 Software Interface

- **Operating System**: iOS, Android.

- Web browser.

- Web Server application.

- DBMS.

- **Report System**: use third part services as OCR.

- **Mailing Service**: APIs to send emails to the User.

- **Mapping Service**: APIs to locate the position of the traffic violation.

3.1.4 Communication Interface

The application runs over HTTPS protocol for the communication over the Internet and with the DBMS.

3.2   Functional Requirements

In the following section are explained the functional requirements of the application.

- [R.1] A visitor must be able to register. During the registration the System will ask to provide some credentials.

- [R.2] Check if the Guest credentials are valid:

    - the username is not already taken by another registered User.
    - the email is in the right format.
    - the password has a minimum length.
    - the badge is valid and matches the municipality, if the Guest is an Authority.

- [R.3] If credentials are valid, the System sends a confirmation email.

- [R.4] Allow to log in using personal credentials.

- [R.5] Allow to change username, only if the new username is not already in use by another User, email, only if the new email is in a correct format and password, only if the new password is different from the precedent and respects the minimum length.

- [R.6] Send a confirmation email if username, email or password is changed (similarly to the registration process).

- [R.7] Allow to change password if it has been forgotten, through the personal email.

- [R.8] The Citizen must be allowed to create reports, specifying:

    - The type of report (violation/accident).
    - The location of the violation/accident.
    - The date of the violation/accident.
    - The time of the violation/accident.
    - A picture of the violation/accident (not mandatory).

- [R.9] Check if the Report created by the Citizen is valid.

- [R.10] Notify the Authorities whenever a new Report is submitted.

- [R.11] Provide two different level of visibility regarding the statistics. Only the Authorities can consult statistics with private data.

- [R.12] Safestreets' application must show the most recent version of the suggestions and statistics.

- [R.13] Provide a recognition system of badge number.

- [R.14] Allow the Citizen to consult the history of its personal reports.

- [R.15] Allow Users to see the their position in a map.

- [R.16] All traffic violations must be notified to Authorities according to their municipality.

- [R.17] Provide the Authorities with a list of the most recent reports submitted.

- [R.18] Allow Authorities to also consult older Reports.

- [R.19] Interact with the Municipality's System to get the permission for using their data.

- [R.20] Use third party services to enable some functions.

- [R.21] Statistics and suggestions have to be periodically recomputed.

- [R.22] Provide protection of data using encryption.

- [R.23] Acknowledge whether or not multiple istances of Reports are referred to the same infraction, even if protracted over time.

- [R.24] Cross the data coming from external DBMSs with the System's own data.

- [R.25] Cross data from different Reports in order to establish their truthfulness.

- [R.26] Unsafe areas are detected by means of an artificial intelligence.

- [R.27] Safety Suggestions are computed by means of an artificial intelligence.

- [R.28] After its validation, no one should be able to modify a submitted Report.

3.2.1 Use Cases

Figure 3.5: Guest use case.

Register Account

ID	UC1
Description	The *Guest* wants to create an account for the application.
Achieved Goals	[G.1]
Actors	*Guest*, *Login Manager* and *Mailing Service*
Preconditions	The *Guest* downloads and opens the application.

Flow of Events	1. The **Guest** taps the Sign Up button and and selects a role between Citizen or Authority. 2. Based on the **Guests'** decision, the **Login Manager** provides the appropriate form to enter all the required data: username, email address and password. In the Authority's form the username corresponds to the badge number, and the municipality of belonging must be inserted as well. 3. The **Guest** fills the form with all the required information and then confirms the entries. The same procedure will be used for future logins. • If the **Guest** is an Authority, he/she has to insert his/her badge number as username and provide the municipality of belonging. 4. The **Login Manager** checks if all the informations are correct, in which case it generates a random activation URL and asks the **Mailing Service** to forward the URL to the email address of the **Guest**. 5. The **Guest** receives the mail and clicks on the URL. 6. The **Login Manager** accepts the registration and stores the data provided by the **User** .
Postconditions	The **Guest** is now able to log in.
Exceptions	1. The **Login Manager** realizes that invalid cerdentials have been entered (g.e. the account already exists) and shows an error message. The flow restarts from point 2.

Log in

ID	UC2
Description	The **Guest** wants to log in to the application.
Achieved Goals	[G.1] and [G.2]
Actors	**Guest** and **Login Manager**
Preconditions	A **Guest** wants to log in to the application and is already registered.

Flow of Events	1. The **Guest** opens the application and inserts his/her credentials (username and password). • If the **Guest** is an Authority, he/she has to insert his/her badge number as username. 2. The **Guest** taps the Log In button. 3. The **Login Manager** checks if all the credentials provided are correct.
Postconditions	The **User** is logged in either as Citizen or Authority.
Exceptions	1. The **Login Manager** recognizes invalid credentials than shows an error message. The flow restarts from point 1.
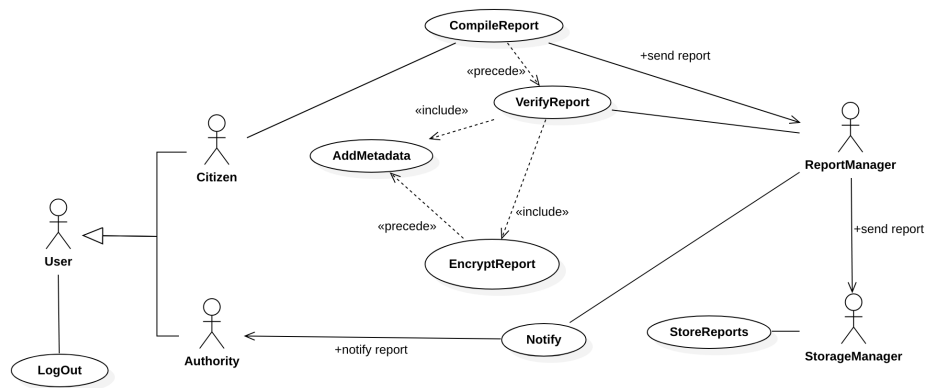
Figure 3.6: User and Report Manager use case.

Send report

ID	UC3
Description	A **Citizen** reports a traffic violation.
Achieved Goals	[G.3] and [G.4]
Actors	**Citizen** and **Report Manager**
Preconditions	The **Citizen** witnesses a traffic violation and wants to report it.

Flow of Events	1. The **Citizen** logs in to the application.
	
	2. The **Citizen** taps on *Report* button.
	
	3. The **Citizen** compiles a Report.
	
	    • He/She takes a picture of the violation, if possible.
	    • He/She also enters the date, time, location and the type of the violation.
	
	4. The **Citizen** taps the *Send* button.
	
	5. The **Report Manager** checks if the report has been correctly compiled.

Postconditions	The report is sent and the **Citizen** recieves a message that confirms the sending was successful.
Exceptions	1. The **Report Manager** rejects the report and shows an error message saying what fields are wrongly compiled. The flow restarts from point 2.

Store and Notify report

ID	UC4
Description	The **Report Manager** recieves a report from a Citizen and notifies the Authorities about it. Then the report is stored in a database.
Achieved Goals	[G.4] and [G.5]
Actors	**Report Manager** and **Storage Manager**
Preconditions	The **Report Manager** recieves a valid report (see UC3).

Flow of Events	1. The ***Report Manager*** computes metadata (g.e. whether or not license plate has been entered, if position has been provided by GPS, etc.) and attaches them to the Report. 2. The ***Report Manager*** runs an API call to retrieve any information about the license plate by means of an external OCR. 3. The ***Report Manager*** notifies the Authorities of the municipality where the violation occurred about the report. 4. The ***Report Manager*** encrypts the report. 5. The ***Report Manager*** sends the report to the ***Storage Manager***. 6. The ***Storage Manager*** stores the reports in the DBMS according to its schedule.
Postconditions	The ***Authorities*** are notified about the Report, which is now stored into the DBMS and ready to be analyzed for building statistics.
Exceptions	1. If the type of the Report recived by the ***Report Manager*** is *Accident* the report is not notified to the authorities, but it is only stored in the database.
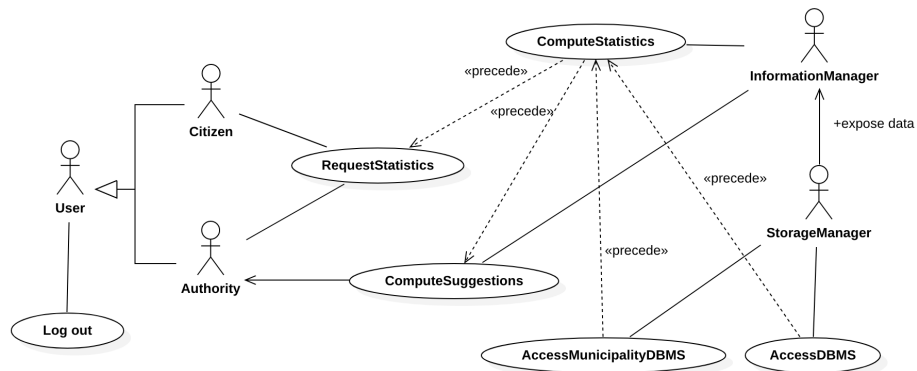
Figure 3.7: Information Manager use case.

Compute Statistics

ID	UC5
Description	A ***User*** wants to see some statistics (e.g the safety of a certain area) and ask the ***Information Manager*** about them.
Achieved Goals	[G.6], [G.7], [G.8], and [G.9]
Actors	***User*** and ***Information Manager***
Preconditions	The ***Information Manager*** has already computed the statistics that the user intends to consult. At set times, the ***Information Manager*** updates all the statistics due to possible new stored reports.

Flow of Events	1. The **User** logs in to the application and taps on the *Statistics* button.
	
	2. The **System** displays a menu with all the possible statistic options to compute.
	
	3. The **User** chooses the statistics in which he/she is interested in.
	
	4. The **Information Manager** loads the statistics that the user selected and the application shows them.
	
	    • The application may use external services to show some statistics on a map.
	
	    • If the **User** who requested to see the statistics is a Citizen, he/she should not be able to see statistics on sensible data.
	
	    • If the **User** who requested to see the statistics is an Authority, the **Information Manager** will also show him/her the latest safety suggestions (if available).

Postconditions	The **User** sees the results of the statistics request.
Exceptions	1. If the Municipality doesn't allow the Storage Manager to access their information, the **Information Manager** use those data to build statistics. In this case, a message saying *Statistic Unavailable* is shown to the **User** who requests to see statistics about iussued traffic tickets. The flow restart from point 3.

3.2.2 Sequence Diagrams
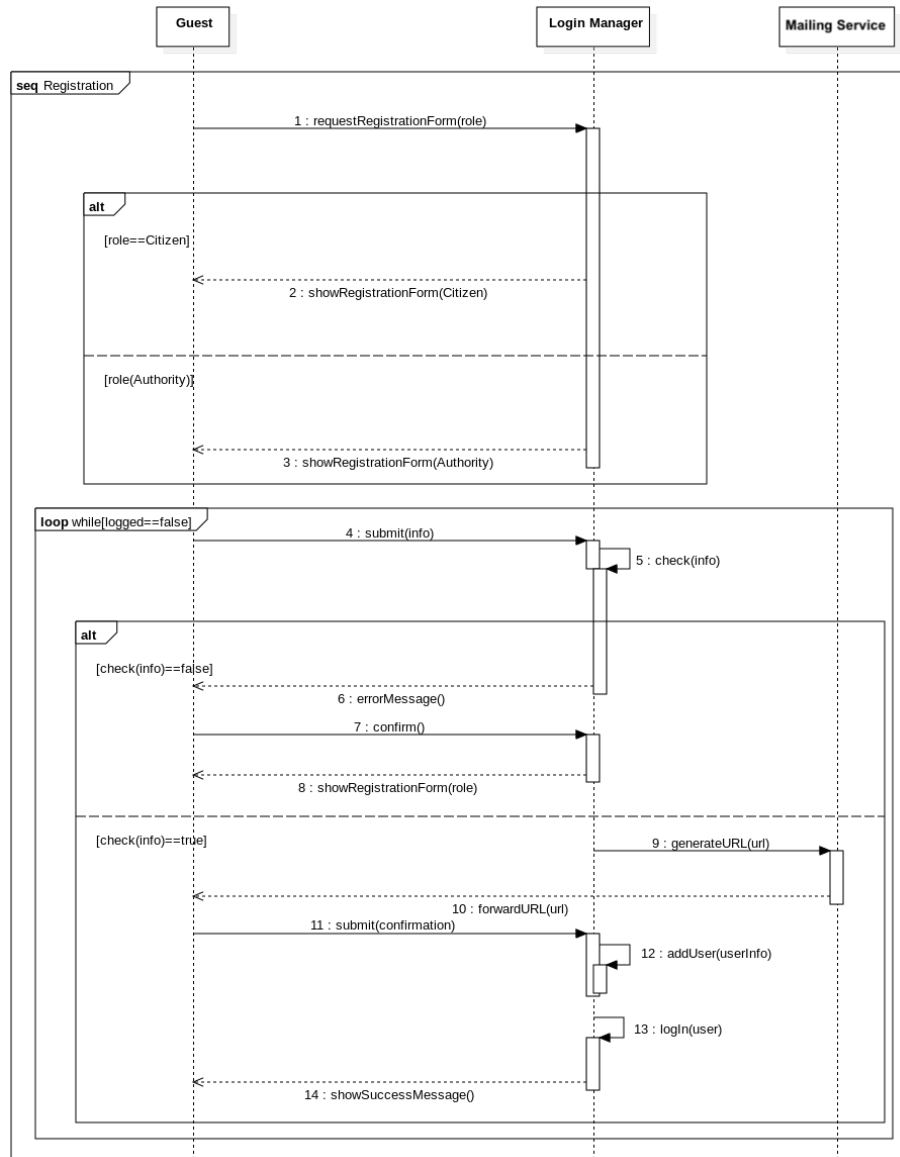
Figure 3.8: Sign up Sequence Diagram.

Figure 3.9: Login Sequence Diagram.
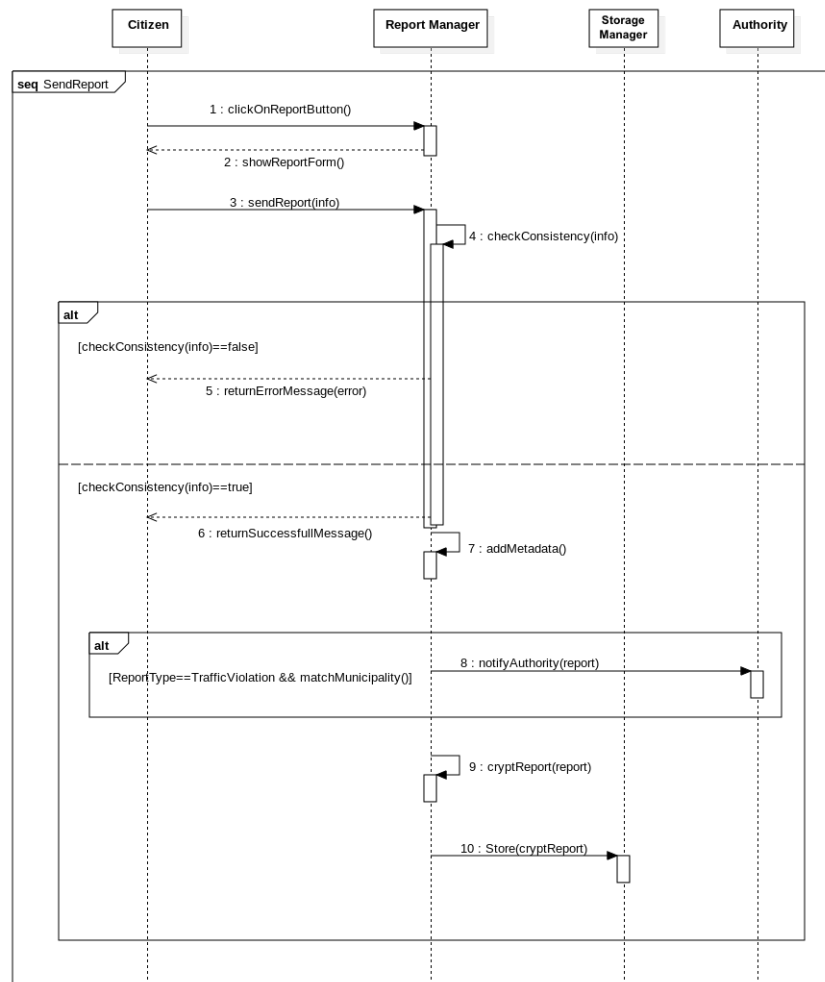
Figure 3.10: Send Report, notify and Store report Sequence Diagram.
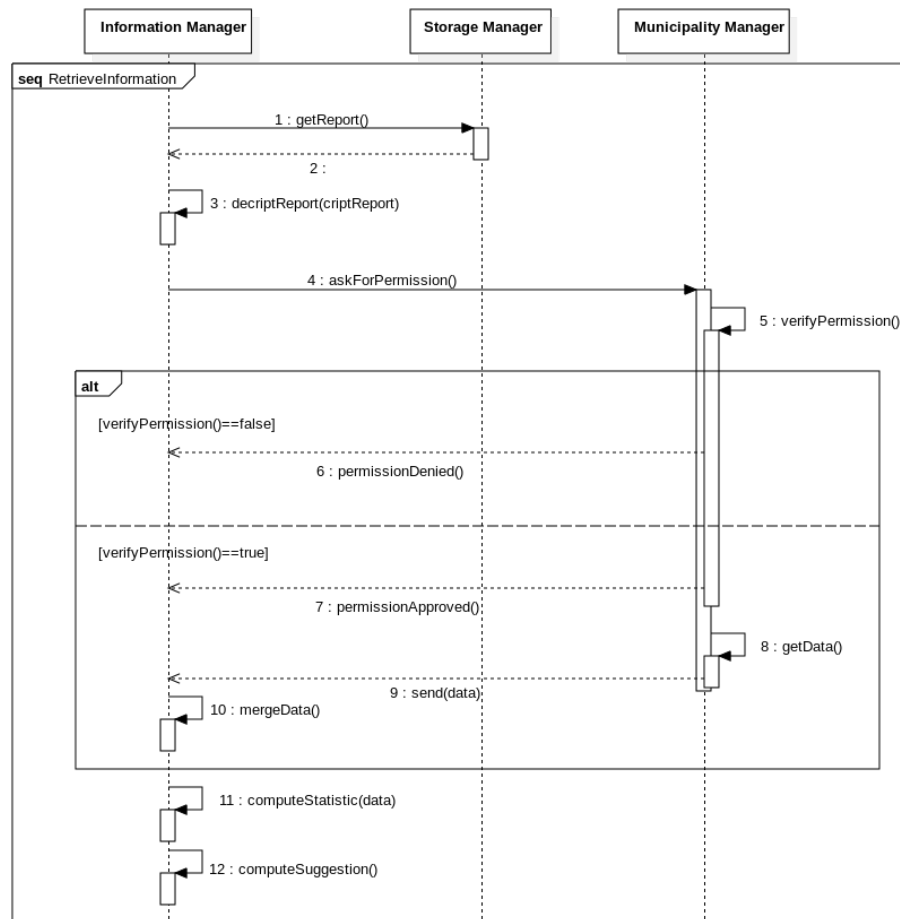
Figure 3.11: Retrieve Information Sequence Diagram.

3.2.3   Goals Mapping on Requirements and Assumptions

- **[G.1] Allow the Users to access the functionalities of the application from different locations and devices.**

  Requirements

  – [R.1] A visitor must be able to register. During the registration the System will ask to provide some credentials.
  – [R.2] Check if the Guest credentials are valid:
    * the email is in the right format.
    * the password has a minimum length.
    * the badge is valid and matches the municipality, if the Guest is an Authority.
    * the username is not already taken by another registered User.
  – [R.3] If credentials are valid, the System sends a confirmation email..
  – [R.5] Allow to change username, but only if the new username is not already in use by another User, email, but only if the new email is in a correct format, and password, only if the new password is different from the precedent and respects the minimum length.
  – [R.6] Send a confirmation email if username, email or password is changed (similarly to the registration process).
  – [R.7] Allow to change password if it has been forgotten, through the personal email.
  – [R.20] Use third party services to enable some functions (Mail Service).

  Domain Assumptions

  – [D.1] The registration mail is assumed to be correctly recieved by the User.

- **[G.2] Allow Guests to authenticate either as Authority or Citizen.**

  Requirements

  – [R.4] Allow to log in using personal credentials.
  – [R.13] Provide a recognition system of badge number.
  – [R.15] Allow Users to see the their position in a map.

– [R.20] Use third party services to enable some functions (Mapping System).

- **[G.3] Allow the User to document traffic violations and accidents by compiling a Report.**

  Requirements

  – [R.8] The Citizen must be allowed to create reports, specifying:
    * The type of report (violation/accident).
    * The location of the violation/accident.
    * The date of the violation/accident.
    * The time of the violation/accident.
    * A picture of the violation/accident (not mandatory).
  – [R.9] Check if the Report created by the Citizen is valid.
  – [R.14] Allow the Citizen to consult the history of its personal reports.

- **[G.4] Store the reports provided by the Citizens.**
  – **[G.4.1] If the Citizen has not provided any information about the license plate, the System runs an algorithm to read it from the submitted picture.**

  Requirements

  – [R.10] Notify the Authorities whenever a new Report is submitted.
  – [R.16] All traffic violations must be notified to Authorities according to their municipality.
  – [R.20] Use third party services to enable some functions (OCR).

  Domain Assumptions

  – [D.2] The System's internal clock time used to provide notifications is correct.
  – [D.3] When a little accident occurs there is no need to contact the Authorities, whereas when a dangerous accident happens Users are supposed to phone call the Authorities for a promptly intervention.

- **[G.5] Allow Authorities to consult the reports submitted by the Citizens.**

Requirements

- [R.17] Provide the Authorities with a list of the most recent reports submitted.
- [R.18] Allow Authorities to also consult older Reports.

Domain Assumptions

- [D.4] The Authorities are assumed to take measures when they are notified about a new report.

- **[G.6] Analyze the stored information to build statistics.**
  - **[G.6.1]If allowed by the Municiplaity, SafeStreets can cross their information about accidents with its own in order to improve the analysis.**

    Requirements

  - [R.19] Interact with the Municipality's System to get the permission for using their data.
  - [R.21] Statistics have to be periodically recomputed.
  - [R.24] Cross the data coming from external DBMSs with the System's own data.
  - [R.25] Cross data from different Reports in order to establish their truthfulness

    Domain Assumptions

  - [D.5] After the first time, the System does not need to request the access to the Municipality's database as long as its permission is valid.
  - [D.6] The statistics and suggestions are assumed to be consulted by Authorities periodically, in order to provide a better service.
  - [R.23] Acknowledge whether or not multiple istances of Reports are referred to the same infraction, even if protracted over time.

- **[G.7] Detect unsafe areas through statistics.**
  - **[G.7.1] Suggest possible interventions to the Authorities.**

    Requirements

- [R.21] Suggestions have to be periodically recomputed.
- [R.26] Unsafe areas are detected by means of an artificial intelligence.
- [R.27] Safety Suggestionss are computed by means of an artificial intelligence.

Domain Assumptions

- [D.6] The statistics and suggestions are assumed to be consult by Authorities periodically, in order to provide a better service.

- **[G.8] Build statistics on iussued tickets if the Municiplaity allows SafeStreets to access such data.**

  - **[G.8.1] Ensure that the chain of custody of the information coming from the Citizens is never broken.**

Requirements

- [R.22] Provide protection of data using encryption.
- [R.28] After its validation, no one should be able to modify a submitted Report.

Domain Assumptions

- [D.7] Only specific component of the System is able to decrypt sensible data.

- **[G.9] Allow the Users to consult statistics.**

  - **[G.9.1] Offer different levels of visibility to different roles.**

Requirements

- [R.11] Provide two different level of visibility regarding the statistics. Only the Authorities can consult statistics with private data.
- [R.12] Safestreets' application must show the most recent version of the suggestions and statistics.
- [R.21] Statistics and suggestions have to be periodically recomputed.

3.3   Performance Requirements

SafeStreets is an application who's main job is to compute statistics and show the results to its Users. The collection of the data and the computation of the statistics are operations carried out by two distinct modules (respectively the Report Manager and the Information Manager) which write and read the database independently and asynchronously. This design is thought to benefit the performances: in conditions of high traffic the Report Manager can store the recieved Reports into a buffer and export later into secondary memory, whereas the Information Manager can suspend the computation of statistics and present to the requesting Users older results previously computed.

3.4   Design Constraint

As already mentioned in Section 2.4.3, since the System is not designed to provide a function to generate official traffic tickets, we can only assume that the Authorities will handle the reported traffic violations properly. For this reason, the System is not aware of whether or not the violation is being handled, at least as long as a corresponding traffic ticket is generated by an Authority (and the System has the authorization to access such data). Even then, the System cannot distinguish a traffic ticket generated with the help of SafeStreets by one that is not.
Moreover, the System is not designed to communicate with any registry office, thus every user is identified by their username, mail and associated ID. This limitation is one of the reasons for which the System does not notify authorities when an accident is notified: in fact, small accidents are not technically traffic violations and often there's no need to call Authorities, whereas in case of serious accidents the Citizen is encouraged to call the police via phone call. In the latter case, the local police will collect the Citizen's testimony and generalities (which are not available in SafeStreets' user profiles) and hopefully will intervene as soon as possible.

3.4.1   Standards Compliance

- **GDPR** with regard to the processing of the User's personal data.

- **Latitude and Longidute degrees** for all the information regarding locations.

3.4.2   Hardware Limitations

The application should be able to run, at least, under the following conditions:

- Mobile application:

    - iOS or Android smartphone

- 3G connections at 2Mb/s
- 50 MB of space
- 2 GB of RAM
- GPS

- Web application:

  - Updated web browser

3.5 Software System Attributes

This section will cover the non-functional requirements that might be used to evaluate the software performances. All of the covered requirements are part of the ISO/IEC 9126-1:2001 (see Section 1.5).

3.5.1 Reliability

The System (and all its components) should guarantees a 99% of reliability. In order to guarantee the continuity of the service, it must be ensured that the System is fault tolerant.

3.5.2 Availability

The System must guarantee 99.9% (*"three-nines"*) of availability, so that only 10.08 minutes of downtime per week are allowed.

3.5.3 Security

User credentials and data will be stored in a DBMS that should guarantee an high level of security. To reach this goal the passwords stored in the database are salted and hashed, so that a sequence of generic characters of any length is concatenated to the password. Hash is also used to store the report so that the information cannot be altered by anyone except specific system components. Thus as soon as a report arrives to the Report Manager and it's validated, it gets encrypted by means of the hash function to form a digital signature and then it is stored. In this way we create a snapshot of the report ensuring the identity, authenticity, and the non-tampered state of the information. As mentioned before, the System uses HTTP over SSL protocol (HTTPS) to comunicate with all the services in order to guarantee privacy and protection of the exchanged information.

If someone breaks into the OCR tool or the Map service, SafeStreets is not responsible in case of damage, due to the fact that its functionalities are provided by third-part services.

3.5.4 Maintainability

As explained in the Design Document, the System will have a modular architecture: Report Manager, Login Manager and Information Manager are designed to work separately in an asynchronous way, thus the maintenance process will be sped up in case of failure of one module.

3.5.5 Portability

SafeStreets is developed in terms of a native application on Android and iOS, but any device with a web browser can access its features.

Formal Analysis Using Alloy

In this section a description of the model is given by using the Alloy modeling language.

4.1   Worlds Generated

The main aspects of the System and its relationships with the World are represented in the Figure 4.1. However, for a better understanding, the System will be divided into sub-systems according to its functionalities, so that each component can be described thoroughly.
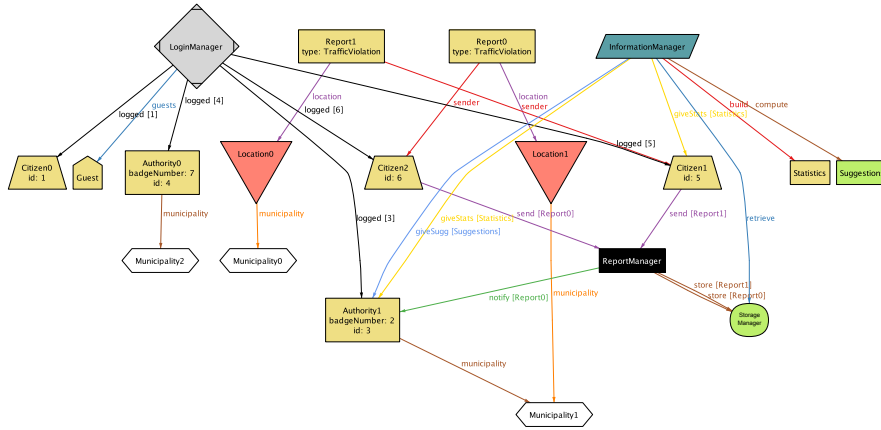
Figure 4.1: Alloy-generated world that shows all the possible interactions.

In such world the Users are modeled either as Citizen or Authority, and each of them has a unique ID (and badge number, if Authority) for which the following facts hold:

```
-- All IDs and badges are unique, different and positive integers --
fact idsAndBadgesDifferent{
        all disj u1,u2:User | u1.id≠u2.id
        all disj a1,a2:Authority | a1.badgeNumber≠a2.badgeNumber
        all u1:User, a1:Authority | (u1.id≠a1.badgeNumber) and (a1.id
            ↪ ≠a1.badgeNumber)
}
fact positiveId{
        all i:Int, u:User | (u.id = i implies i > 0) and (u.
            ↪ badgeNumber ≠ none implies u.badgeNumber > 0)
}
```

4.1.1 Login Manager

The Login Manager is in charge of providing the login functionality, as well as keeping a list of all the Users and the Guests who wants to log in to the application. In the Figure 4.1.1 it is shown the result of a login procedure: the Guest who requested the login in gets disconnected from the Login Manager and a new User makes its appearance in the set of the logged Users. The login manager also has to provide an ID to the new User, which has to be different from all the IDs and the badge numbers already existent.
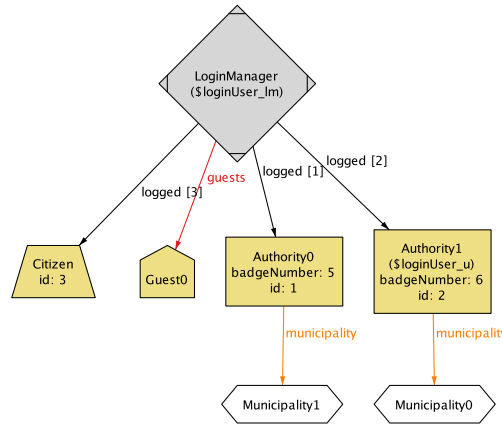
Figure 4.2: Alloy-generated world that shows how the login works.

Below is the predicate which led to the Figure 4.1.1 and all the facts involved:

```
-- Predicates used for the login procedure --
pred disconnectedGuest{
        one g:Guest | one  lm:LoginManager | !(g in lm.guests)
}
pred loginUser[g:Guest, u:User, id:Int, lm:LoginManager]{
        disconnectedGuest
        lm.logged = lm.logged + (id -> u)
}

-- Login Manager keeps track of all Users and Guests (except the one
    ↪ who logs in) --
```

```
fact allUsersInLM{
        lone g:Guest | one  lm:LoginManager | !(g in lm.guests)
        no u:User| one lm:LoginManager | !u.id -> u in lm.logged
        all c:Citizen | (#LoginManager = 0 and #ReportManager = 1)
            ↪ implies #c.send > 0
        #LoginManager = 0 implies #Guest = 0
}
-- Each logged User is linked to a unique integer (which is his ID)
    ↪ --
fact singleId{
        no disj i1,i2:Int, u:User, lm:LoginManager | i1->u in lm.
            ↪ logged and i2->u in lm.logged
}

run loginUser for 6 but 4 Int
```

4.1.2 Report Manager

The Report Manager's job is to collect all the Reports send by the Users, store them into the database, and notify the Authorities of the same Municipality in which the violation took place. As previously mentioned, unlike the other violations, accidents are not notified to Authorities.
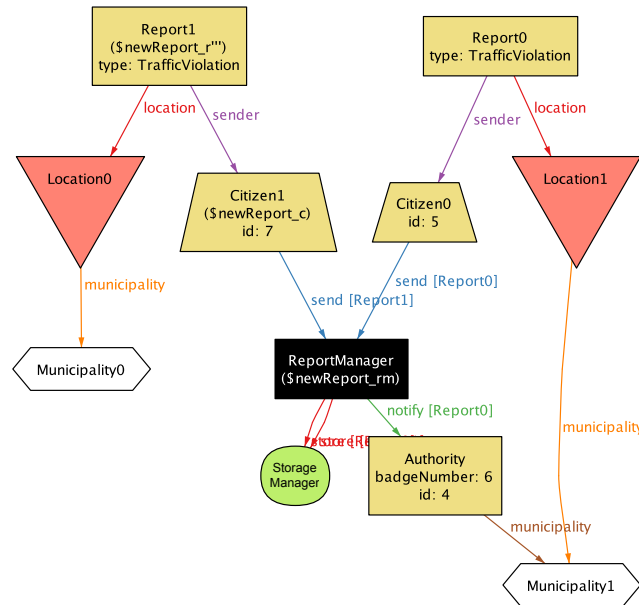
Figure 4.3: Alloy-generated world that shows what happens when a new report is submitted.

Here are shown all the facts that involve Reports, as well as the predicate which generates a new one:

```
-- Predicate used to add a new Report --
pred newReport[r:Report, c:Citizen, rm:ReportManager]{
        !disconnectedGuest
        c.send=c.send + (r -> rm)
}
-- All reports are sent to the RM and then stored --
fact reportSend{
        all r:Report, c:Citizen |one rm:ReportManager | (r.sender=c)
          ↪ <>( r->rm in c.send)
}
fact reportStore{
        all rm:ReportManager, r:Report |one db:StorageManager | r->db
          ↪  in rm.store
}
-- The Report Manager notifies Authorities with only traffic
   ↪ violations that involve their municipality --
        fact notifyReportBasedOnTypeAndMunicipality{
        all a:Authority,r:Report |one rm:ReportManager | ((r.type =
          ↪ TrafficViolation) and (r.location.municipality =a.
          ↪ municipality)) <> (r->a in rm.notify)
}
-- All municipalities must be involved in a violation or are
   ↪ supervised by at least an Authority --
fact noReportNoMunicipality{
        all m:Municipality | (some r:Report | getMunicipality[r] = m
          ↪ ) or (some a:Authority | a.municipality = m )
        no l:Location | (no r:Report | l = r.location)
}
-- Violations and Accidents exists only if reported --
fact noTypesWithoutReport{
        no v:TrafficViolation| ( no r: Report | r.type = v)
        no i:Incident| ( no r: Report | r.type = i)
}

run newReport for 6 but 4 Int
```

4.1.3 Information Manager

The Information manager retrives from the Storage Manager the information it needs, it builds statistics, computes suggestions, and makes them all available to the Users. Only Authorities have the permission to see safety suggestions, as they are the ones who are supposed to apply them.
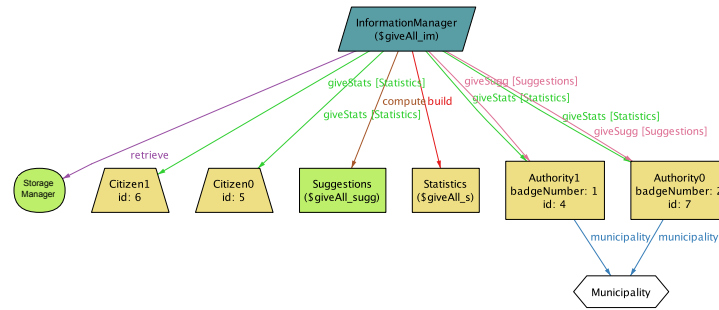

Figure 4.4: Alloy-generated world that shows how the statistics and suggestions are distributed.

Here are shown the facts involving statistics, suggestions and the predicates that makes them available to the Users:

```
-- Predicate used to give Suggestions and Statistics --
pred giveStatistics[u:User, im:InformationManager, s:Statistics, sugg
    ↪ :Suggestions]{
        im.giveStats = im.giveStats + (s -> u)
}
pred giveAll[s:Statistics, sugg:Suggestions, im:InformationManager]{
        all u:User | giveStatistics[u, im, s, sugg]
}
-- Statistics and Suggestions are only generated by the Information
    ↪ Manager --
fact noInfoManagerNoStatistics{
        all s:Statistics| one i:InformationManager | s in i.build
        all s:Suggestions| one i:InformationManager | s in i.compute
}
-- Visibility constraints --
fact giveStatsAndSuggs{
        all u:User, im:InformationManager | (im.build ≠ none) implies
            ↪ (im.build -> u in im.giveStats)
        all im:InformationManager, u:User, s:Statistics, su:
            ↪ Suggestions | (u.badgeNumber ≠ none and s->u in im.
            ↪ giveStats) <> (su -> u in im.giveSugg)
        no c:Citizen, s:Suggestions, im:InformationManager | s->c in
            ↪ im.giveSugg
}

run giveAll for 6 but 4 Int
```

4.2 Model Check

```
assert allDifferendIdsBadges{
no disj u1,u2:User| (u1.id = u2.id) or (u2.badgeNumber ≠ none and u1.
    ↪ id = u2.badgeNumber) or (u1.badgeNumber ≠ none and u2.
    ↪ badgeNumber ≠ none and  u1.badgeNumber = u2.badgeNumber)
}
assert oneMunicipalityOneAuthority{
        no disj m1,m2:Municipality, a:Authority | a.municipality=m1
            ↪ and a.municipality=m2
}
assert writerReportDifferent{
        no disj c1,c2:Citizen, r:Report | r.sender=c1 and r.sender=c2
}
assert allReportHasWriter{
        no r:Report | no c:Citizen | r.sender = c
}
assert notifyReportBasedOnMunicipality{
        no r :Report, a: Authority, rm: ReportManager | (r->a in rm.
            ↪ notify) and (r.location.municipality ≠ a.municipality
            ↪ )
}
assert notifyReportBasedOnType{
        no r :Report, a: Authority, rm: ReportManager | (r->a in rm.
            ↪ notify) and (r.type = Incident)
}
assert suggestionsVisibility{
        no c:Citizen, im:InformationManager |(im.compute ≠ none) and
            ↪ (im.compute -> c in im.giveSugg)
}
assert allUsersCanSeeStatistics{
        no u:User, im:InformationManager | (im.build ≠ none) and !(im
            ↪ .build -> u in im.giveStats)
}
```

```
assert noWanderingClient{
        all c:Client | all r:Report , im:InformationManager , lm:
        ↪ LoginManager| (r.sender = c) or (c in lm.guests or c.
        ↪ id->c in lm.logged) or (im.build ->c in im.giveStats)
        ↪  or (c.municipality ≠ none)
}
assert allUsersLoggedWithID{
        no u:User , lm:LoginManager , i:Int | (i->u in lm.logged) and (
        ↪ i ≠ u.id)
}
assert onlyRMandIMaccessStorageManager{
        no d:StorageManager , rm:ReportManager , im:InformationManager
        ↪ | rm = none and im = none and d ≠ none
}
```

Executing "Check allDifferendIdsBadges"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
4279 vars. 274 primary vars. 8677 clauses. 216ms.
No counterexample found. Assertion may be valid. 34ms.

Executing "Check oneMunicipalityOneAuthority"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
3962 vars. 277 primary vars. 7737 clauses. 70ms.
No counterexample found. Assertion may be valid. 3ms.

Executing "Check writerReportDifferent"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
3938 vars. 277 primary vars. 7695 clauses. 38ms.
No counterexample found. Assertion may be valid. 5ms.

Executing "Check notifyReportBasedOnType"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
3928 vars. 275 primary vars. 7673 clauses. 47ms.
No counterexample found. Assertion may be valid. 3ms.

Executing "Check notifyReportBasedOnMunicipality"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
3958 vars. 275 primary vars. 7769 clauses. 45ms.
No counterexample found. Assertion may be valid. 7ms.

Executing "Check allUsersCanSeeStatistics"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
3900 vars. 272 primary vars. 7621 clauses. 44ms.
No counterexample found. Assertion may be valid. 4ms.

Executing "Check suggestionsVisibility"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
3900 vars. 272 primary vars. 7621 clauses. 51ms.
No counterexample found. Assertion may be valid. 5ms.

Executing "Check allUsersLoggedWithID"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
4180 vars. 288 primary vars. 8193 clauses. 60ms.
No counterexample found. Assertion may be valid. 5ms.

Executing "Check noWanderingClient"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
4156 vars. 276 primary vars. 8062 clauses. 48ms.
No counterexample found. Assertion may be valid. 2ms.

Executing "Check onlyRMandIMaccessDBMS"
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
3898 vars. 271 primary vars. 7622 clauses. 47ms.
No counterexample found. Assertion may be valid. 1ms.

Figure 4.5: Results for running all the assertions above. No counterexample is found.

4.3   Whole Alloy Model

```
-- SIGNATURES --
abstract sig ReportType{}
lone abstract sig Incident extends ReportType{}
lone abstract sig TrafficViolation extends ReportType{}
lone sig Statistics{}
lone sig Suggestions{}
lone sig StorageManager{}

sig Municipality{}
sig Location{
        municipality: one Municipality
}

abstract sig Client {}
sig Guest extends Client {}
abstract sig User extends Client {
        id: one Int
}
sig Citizen extends User {
        send: Report -> ReportManager
}
sig Authority extends User {
        badgeNumber: one Int,
        municipality: one Municipality
}

sig Report {
        type: one ReportType,
        location: one Location,
        sender: one Citizen,
}

lone sig ReportManager{
        notify: Report-> Authority,
        store: Report -> StorageManager
}
lone sig InformationManager{
        retrieve: StorageManager,
        build: lone Statistics,
        compute: lone Suggestions,
        giveStats: Statistics -> User,
        giveSugg: Suggestions -> User
}
lone sig LoginManager{
        guests : set Guest,
        logged : Int -> User
}

-- FUNCTIONS --
fun getMunicipality[r:Report] : Municipality{
        r.(location.municipality)
}

-- FACTS --
fact reportSend{
        all r:Report, c:Citizen |one rm:ReportManager | (r.sender=c)
        ↪ ≤>( r->rm in c.send)
}
fact reportStore{
        all rm:ReportManager, r:Report |one db:StorageManager | r->db
        ↪  in rm.store
}
fact notifyReportBasedOnTypeAndMunicipality{
```

```
                all a:Authority,r:Report |one rm:ReportManager | ((r.type =
                    ↪ TrafficViolation) and (r.location.municipality =a.
                    ↪ municipality)) <> (r->a in rm.notify)
}
fact idsAndBadgesDifferent{
        all disj u1,u2:User | u1.id≠u2.id
        all disj a1,a2:Authority | a1.badgeNumber≠a2.badgeNumber
        all u1:User, a1:Authority | (u1.id≠a1.badgeNumber) and (a1.id
            ↪ ≠a1.badgeNumber)
}
fact noReportNoMunicipality{
        all m:Municipality | (some r:Report | getMunicipality[r] = m
            ↪ ) or (some a:Authority | a.municipality = m )
        no l:Location | (no r:Report | l = r.location)
}
fact noInfoManagerNoStatistics{
        all s:Statistics| one i:InformationManager | s in i.build
        all s:Suggestions| one i:InformationManager | s in i.compute
}
fact giveStatsAndSuggs{
        all u:User, im:InformationManager | (im.build ≠ none) implies
            ↪  (im.build -> u in im.giveStats)
        all im:InformationManager, u:User, s:Statistics, su:
            ↪ Suggestions | (u.badgeNumber ≠ none and s->u in im.
            ↪ giveStats) <> (su -> u in im.giveSugg)
        no c:Citizen, s:Suggestions, im:InformationManager | s->c in
            ↪ im.giveSugg
}
fact allUsersInLM{
        lone g:Guest | one  lm:LoginManager | !(g in lm.guests)
        no u:User| one lm:LoginManager | !u.id -> u in lm.logged
        all c:Citizen | (#LoginManager = 0 and #ReportManager = 1)
            ↪ implies #c.send > 0
        #LoginManager = 0 implies #Guest = 0

}
fact positiveId{
        all i:Int, u:User | (u.id = i implies i > 0) and (u.
            ↪ badgeNumber ≠ none implies u.badgeNumber > 0)
        all i:Int, u:User, lm:LoginManager |  i->u in lm.logged
            ↪ implies i>0
}
fact singleId{
        no disj i1,i2:Int, u:User, lm:LoginManager | i1->u in lm.
            ↪ logged and i2->u in lm.logged
}
fact noTypesWithoutReport{
        no v:TrafficViolation| ( no r: Report | r.type = v)
        no i:Incident| ( no r: Report | r.type = i)
}
fact aloneStorageManager{
        (#InformationManager = 0 and #ReportManager = 0) implies #
            ↪ StorageManager = 0
}

-- ASSERTIONS --
assert allDifferendIdsBadges{
no disj u1,u2:User| (u1.id = u2.id) or (u2.badgeNumber ≠ none and u1.
    ↪ id = u2.badgeNumber) or (u1.badgeNumber ≠ none and u2.
    ↪ badgeNumber ≠ none and  u1.badgeNumber = u2.badgeNumber)
}
assert oneMunicipalityOneAuthority{
        no disj m1,m2:Municipality, a:Authority | a.municipality=m1
            ↪ and a.municipality=m2
}
assert writerReportDifferent{
        no disj c1,c2:Citizen, r:Report | r.sender=c1 and r.sender=c2
}
```

```
        assert allReportHasWriter{
                no r:Report | no c:Citizen | r.sender = c
        }
        assert notifyReportBasedOnMunicipality{
                no r :Report, a: Authority, rm: ReportManager | (r->a in rm.
                    ↪ notify) and (r.location.municipality ≠ a.municipality
                    ↪ )
        }
        assert notifyReportBasedOnType{
                no r :Report, a: Authority, rm: ReportManager | (r->a in rm.
                    ↪ notify) and (r.type = Incident)
        }
        assert suggestionsVisibility{
                no c:Citizen, im:InformationManager |(im.compute ≠ none) and
                    ↪ (im.compute -> c in im.giveSugg)
        }
        assert allUsersCanSeeStatistics{
                no u:User, im:InformationManager | (im.build ≠ none) and !(im
                    ↪ .build -> u in im.giveStats)
        }
        assert noWanderingClient{
                all c:Client | all r:Report, im:InformationManager, lm:
                    ↪ LoginManager| (r.sender = c) or (c in lm.guests or c.
                    ↪ id->c in lm.logged) or (im.build ->c in im.giveStats)
                    ↪  or (c.municipality ≠ none)
        }
        assert allUsersLoggedWithID{
                no u:User, lm:LoginManager, i:Int | (i->u in lm.logged) and (
                    ↪ i ≠ u.id)
        }
        assert onlyRMandIMaccessStorageManager{
                no d:StorageManager, rm:ReportManager, im:InformationManager
                    ↪ | rm = none and im = none and d ≠ none
        }

        -- PREDICATES --
        pred newReport[r:Report, c:Citizen, rm:ReportManager]{
                !disconnectedGuest
                c.send=c.send + (r -> rm)
        }
        pred loginUser[g:Guest, u:User, id:Int, lm:LoginManager]{
                disconnectedGuest
                lm.logged = lm.logged + (id -> u)
        }
        pred disconnectedGuest{
                one g:Guest | one  lm:LoginManager | !(g in lm.guests)
        }
        pred showUser{
                !disconnectedGuest
        }
        pred giveStatistics[u:User, im:InformationManager, s:Statistics, sugg
            ↪ :Suggestions]{
                im.giveStats = im.giveStats + (s -> u)
        }
        pred giveAll[s:Statistics, sugg:Suggestions, im:InformationManager]{
                all u:User | giveStatistics[u, im, s, sugg]
        }

-- EXECUTION --
run showUser for 6 but 4 Int
run newReport for 6 but 4 Int
run giveAll for 6 but 4 Int
run loginUser for 6 but 4 Int
```

Effort Spent

Francesco Amorosini

Task		Hours
Meeting for analysis of task	‖	4
Tools and Github setup for the project	‖	2
Purpose and Scope	‖	3.5
Product Functions	‖	3.5
Mid-Phase Call: • Functional requirements and Assumptions • Goal Mapping on Requirements	‖	 3 3
Performance requirements	‖	1.5
Design constraints	‖	3.5
Software System Attributes	‖	1
Alloy modeling	‖	2.5
Alloy chapter	‖	4.5
Images and tables polishing	‖	2.5
Grammar correction and global review	‖	5.5
Upgrade to v1.1	‖	3
	‖	**Total:** 43

Alice Casali

Task		Hours
Meeting for analysis of task	‖	4
Tools and Github setup for the project	‖	2
Goals discussion	‖	2
Product Perspective	‖	3
Statechart Diagrams	‖	3.5
First approach on assumptions, dependencies and constraints	‖	1.5
Functional Requirements	‖	3.5
Mid-Phase Call: • Functional requirements and Assumptions • Goal Mapping on Requirements	‖	 3 3
Added new UseCase Templates	‖	2
Statecharts revision	‖	1
External Interface Requirement	‖	1
User interface with Mockup	‖	3.5
Final revision	‖	2
	‖	**Total:** 35

Tommaso Fioravanti

Task	Hours
Meeting for analysis of task	4
Tools and Github setup for the project	2
Purpose and Scope	2,5
Overview, Definitions and Acronyms	1,5
Goal Analysis	2
Class Diagram	2
Use Case Diagrams and Template	3,5
Sequence Diagrams	4
Software System Attributes	2
Functional Requirement and Assumptions review	1,5
Alloy modeling	5,5
Sequence Diagrams review	2
Global review	3
Further refinements	1,5
	Total: 37