SafeStreets RASD

Matteo Secco, Mohammad Rahbari

release date: November 10, 2019
version 1

Contents

1 Introduction

1.1 Purpose

here we include the goals of the project

The required system, called SafeStreets, is a distributed system to allow the citizens to signal parking violations to the competent autorities.

The system must allow the citizen to submit pictures of the violation, attaching data such as date, time and position. The user will have to specify the type of the violation when sending these data.

When reciving such data, the system must store them, toghether with the plate of the car that performed the violation (elictated from the picture the citizen sent), the informations about the violation itself (in particular the type of violation), and the name of the street where the violation occourred (which can be retrieved by the positioning information the user sent).

In addition, the application must allow both autorities and citizens to analyze the stored data, for example highlighting streets or plates with most violations registered. Different levels of security can be offered.

Finally, the application must be enable authorities to automatically generate traffic tickets using its data for people commiting violations, if and only if it can be verified that the data concerning it has not been altered. In this case, SafeStreets could use this informations to build additional statistics.

1.1.1 Goal list

`G.1` Allow citizens to notify parking violations and be acknowledged of the result as soon as possible

`G.2` Force citizens to provide all the needed data about violation, in particular infraction type, picture, date, time and position

`G.3` Prevent the autorities to have to manually address parking tickets

`G.4` Ensure no tickets can be emitted if the notification's data has been modified somehow

`G.5` Ensure no tickets can be emitted if the plate of the car that committed the infringment owns a permission for that infringiment

`G.6` Every notification not covered by `G.4` or `G.5` will be eligible for ticket generation

`G.7` Allow both citizens and authorities retrive informations about previous violations and released tickets, possibly in an aggregated form. Every violation reported to the system and stored will appear in some statistics

1.2 Scope

here we include an analysis of the world and of the shared phenomena
The world where the system must fit is an everday city, with focus on the traffic of moterized veichles.
The events the system aims to influence are the parking of motorized veichles, in particular the ones considered infractions.
In the context of the system, when any user notices an illegal parking, he/she may notify the system and provide any needed informations to the competent authorities. In particular, the notification is composed by a picture of the infraction, a timestamp (date and time), the geographical location of the infraction and the type of infraction wich is to be notified. Some of these informations can be gathered automatically from the user's device. Notice that, for legal reasons, SafeStreets will just make the data available to the auth, and will not generate tickets itself.

In addiction, the user may interrogate the system to gather aggregated informations about the locations with more violation incidence, and the cars which committed more violations.

Table 1: Phenomena

Phenomenon	**Shared**	**Who controls it**
A citizen parks its car	N	W
A citizen spots a car in an illegal parking	N	W
A citizen wants to notify a violation	Y	W
The user fills the data needed to notify a violation	Y	W
The machine revices a notification, analyzes it and stored stores it if it's trusted	N	M
A user requests map statistics	Y	W
A citizen requests top violators statistics	Y	W
The machine analyzes data and builds statistics	N	M
Statistics are organized and presented to the user	Y	M
An authority wants to generate tickets	N	W
An authority requests some violation notified and stored in the machine	Y	W
The machine provides some notification to an authority requesting them	Y	M
The authority decides whether generate a ticket for a violation or not	Y	W
The authority generates a ticket for a violation	N	W
The authority informs the machine she has generated a ticket for a given violation	Y	W
An authority requests informative statistics about its competence area	Y	W

1.3 Definitions, Acronyms,Abbreviations

Person: A person in the real world. Every Citizen is a person, generally an Authorithy is not

User: A person, an organization or a system which somehow uses SafeStreets

Citizen (cit): This term will be used to denote every user not owning particular privileges or permissions. A citizen is only allowed to notify violations and see some aggregated data

Authority (auth): This term will denote every user (phisical or digital) having privileged access to the stored data. An example of Authority is the Local Police.

Notification: Represents a set of data submitted by any user composed by:

- A picture of a parking infraction
- A timestamp of when the notification occourred, containing date and time (may be gathered automatically by the citizen's device)
- A geographical position of where the infraction occurred (may be gathered automatically by the citizen's device)
- The type of infraction notified

Car: The word car will be used to issue every mothorized vehicle

Plate: Identifies a car

Permisison: A document released by a verified authority, granting to a car the permission to park in a set of reserved parkings (ex: permission to park on parking reserved for disabled people).

2 Overall descriprion

2.1 Product perspective

Generally speaking safe street app provides a better collaborating between normal citizs and auths to solve some problem related to traffic of their city.In this case citizs can report car drivers violator to the auths by sendig a photo from the violator's car through the application then if the photo sender has right he/she can be sure that the violator received his/her fines. Overally auths and citizs by mining these data that comes from the app can have some information about the violations that already happend over their city.Bellow we provide some Diagrams to have better understanding and making our goals more explicit.

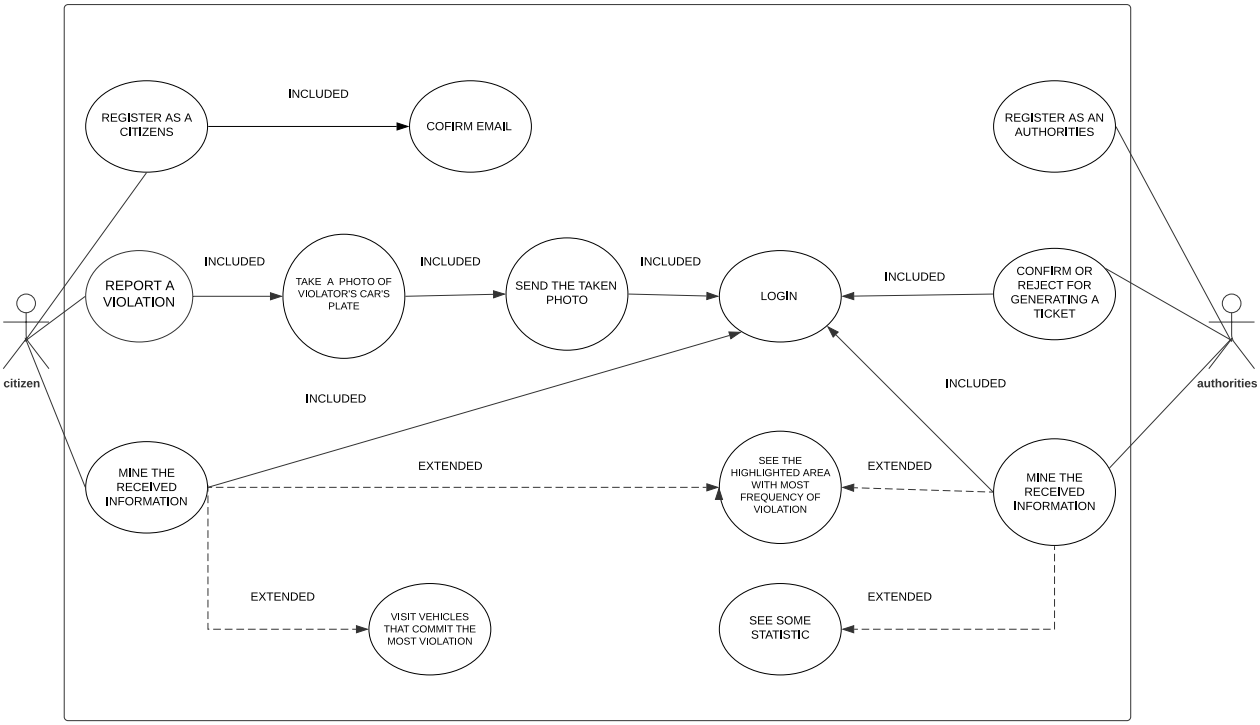


Figure 1: Use Case Diagram

Table 2: Register as a Citizen

Actor	User
Entry condition	• The user has installed the Application • The user has opened the Application
Events flow	1. Click on Sign up button 2. Fill all the mandatory fields and provide the necessary information 3. Click on Confirm button 4. The system saves the data temporarly 5. The user confirm his/her email 6. The system stores the data permanently
Exit condition	The user's Account has register successfully
Exeption	• The user is already signed up. In this case the system warns the user and suggests him/her to do the login. • The username is already taken • The e-mail is already registered • The user didnt fill all of the mandatory fields with valid data • The user doesn't confirm the email in time, and its data are deleted • Server is unreachable

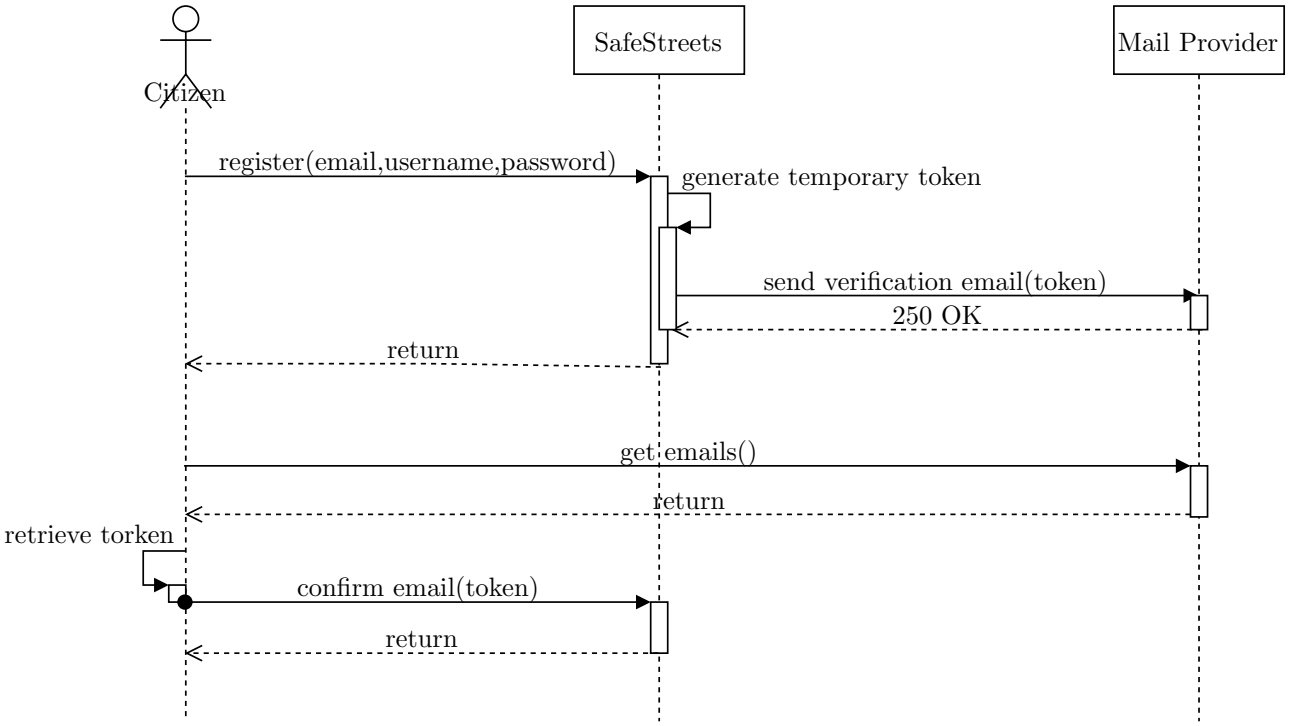


Figure 2: Register as a Citizen: sequence diagram

Table 4: Login

Actor	User
Entry condition	• The user has already signed up and has the application on own device • The user has opened the Application
Events flow	1. Fill the Username and Password fields 2. Click on "sign in" button 3. The user is successfully logged in
Exit condition	The user is signed in and can access the services for its user class
Exeption	• The user enters invalid Username • The user enters invalid Password • All the exceptions are handled by notifying the user which fieldis are wrong , suggesting to correct it

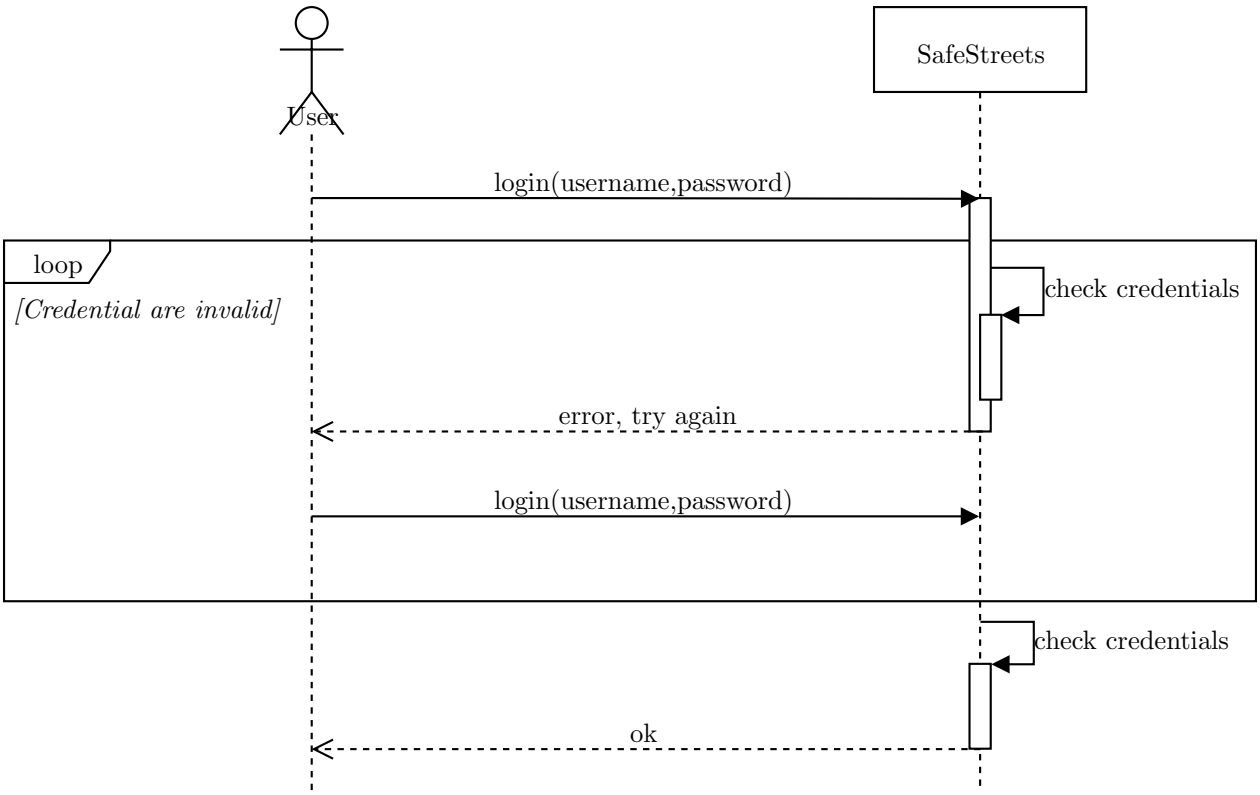


Figure 4: Login: sequence diagram

Table 3: Register as an authority

Actor	Authorities
Entry condition	• The user has installed the Application • The user has opened the Application
Events flow	1. Click on Sign up button 2. Fill all the mandatory fields and provide the necessary information(different for authorities such as there is no field for enter their Email and their username should be their personal code as an officer ...) 3. Click on Confirm button 4. The system saves the data
Exit condition	The user's Account has register successfully
Exeption	• The user is already signed up. In this case the system warns the user and suggests him/her to do the login. • The username is already taken • The user didnt fill all of the mandatory fields with valid data • Server is unreachable

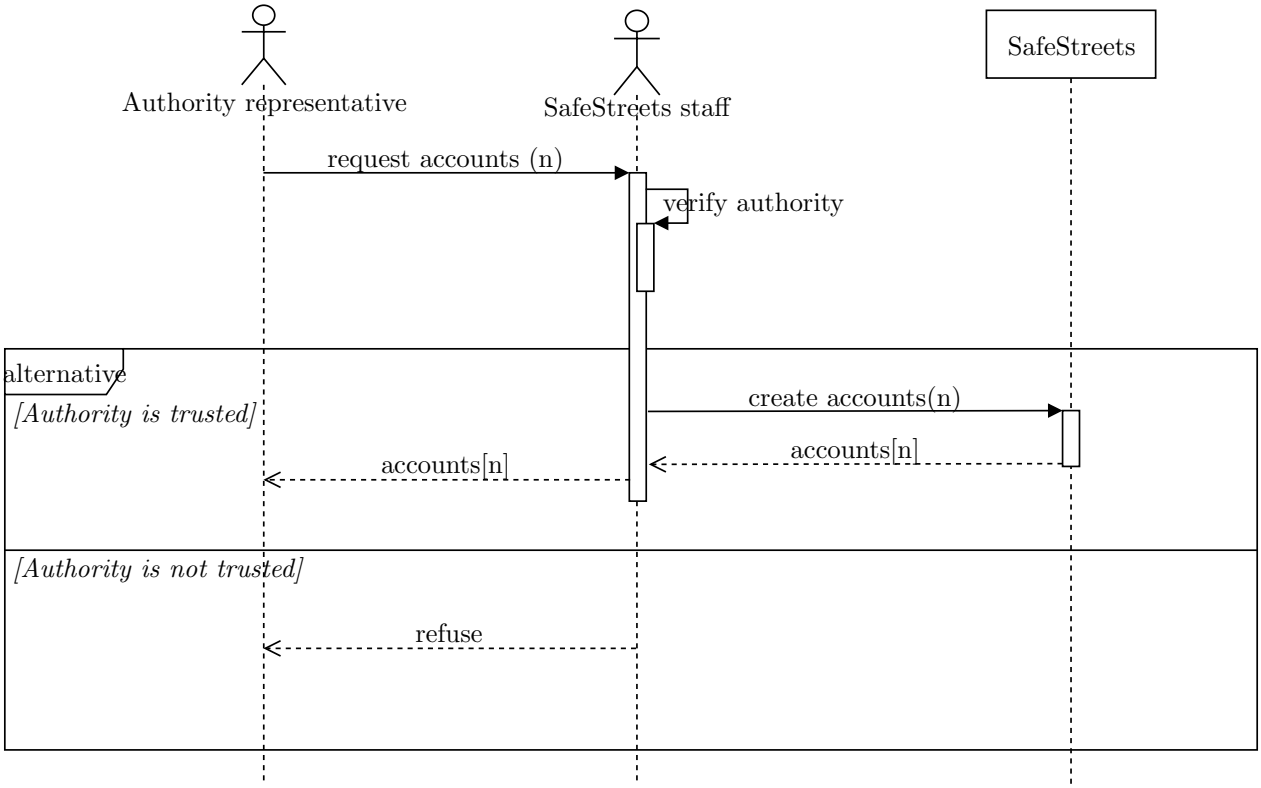


Figure 3: Register as an Authority: sequence diagram

Table 5: Report a violation

Actor	citizens
Entry condition	The user is signed in into the application
Events flow	1. User open the app and observe that the first interface is like a camera App for faster access to taking the photo 2. After focus on the violator's car's license plate push the Red Circle Button to taking photo 3. If the photo have all the well captured photo (vivid picture) push the send button 4. Inform the user that his/her photo successfully recieved by representing a notification 5. App after some seconds send the result whether it is accepted as a violation or this is Reserved parking place for some Special citizens(Such as under cover police or people with disabilities and...)
Exit condition	The violation is stored and the user is notified of the result
Exeption	• Server is unreachable • The notification is discarded for some reason, such as a permission for the car to park in the reserved parking. In that case, the user is notified • Server is unreachable

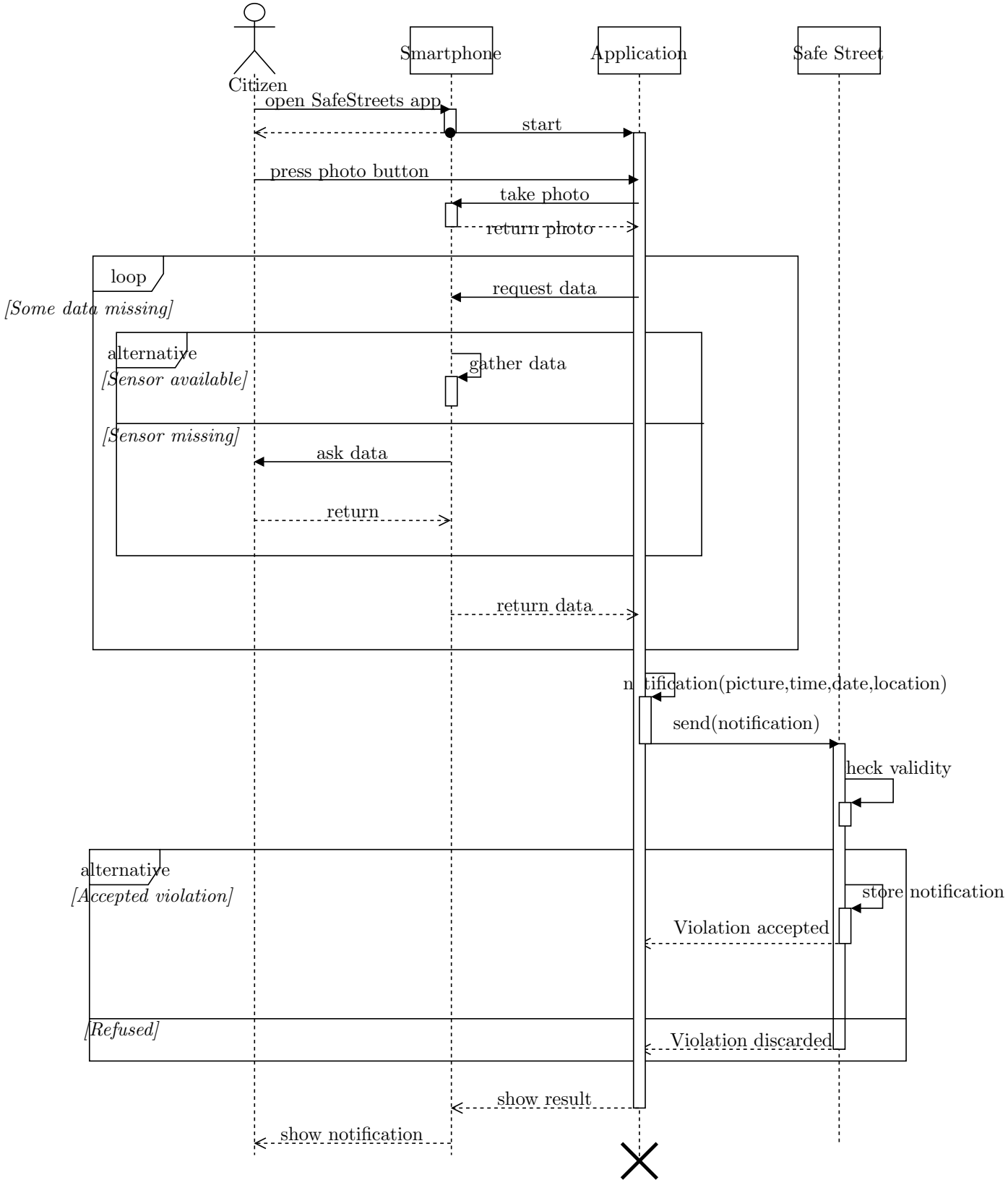


Figure 5: Login: sequence diagram

Table 6: See the level of risk of each street on a map

Actor	User
Entry condition	• The user is signed in into the application
Events flow	1. User Click on MAP button 2. A request is sent to the server 3. The server aggregates the data in classes of risk and sends them to the User Interface 4. The User Interface sends these data to the Map API, which returns the map to display 5. SafeStreets' application displays the map to the user, who can navigate it thanks to the direct channel with the API
Exit condition	The user can see the requested data presented on its device
Exeption	• Server is unreachable • The service for map creation is not available • The needed data cannot be retrived from the database

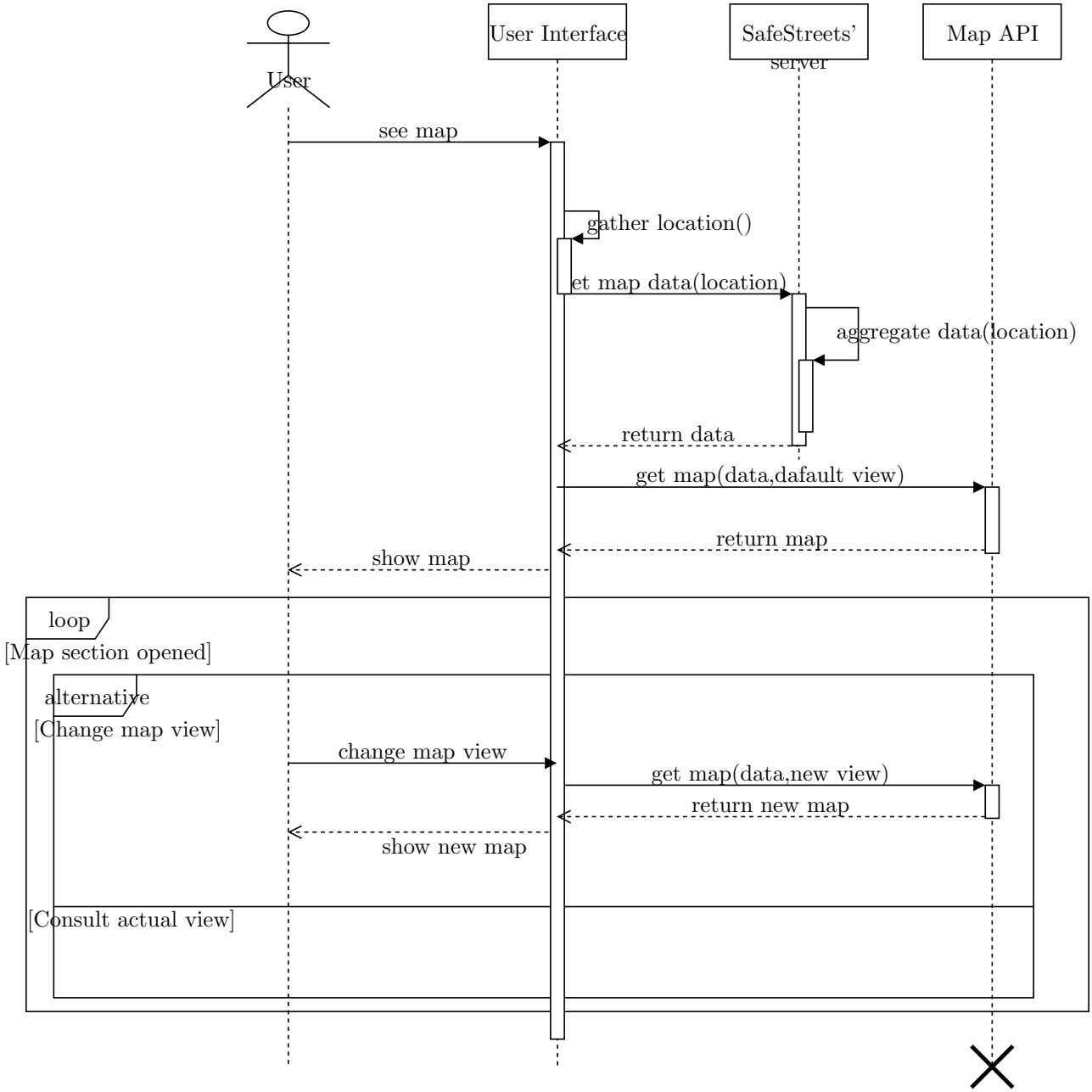


Figure 6: See streets' risk: sequence diagram

Table 7: See the ranking of top violators

Actor	Citizen
Entry condition	• The user is signed in into the application
Events flow	1. User Click on RANKING button 2. A request is sent to the server 3. The server computes the ranking, takes a picture for each ranked violator, hides the plates and sends this data back to the User's application 4. The application displays the ranking to the user
Exit condition	The user can see the requested data presented on its device
Exeption	• Server is unreachable • The service for map creation is not available • The needed data cannot be retrived from the database

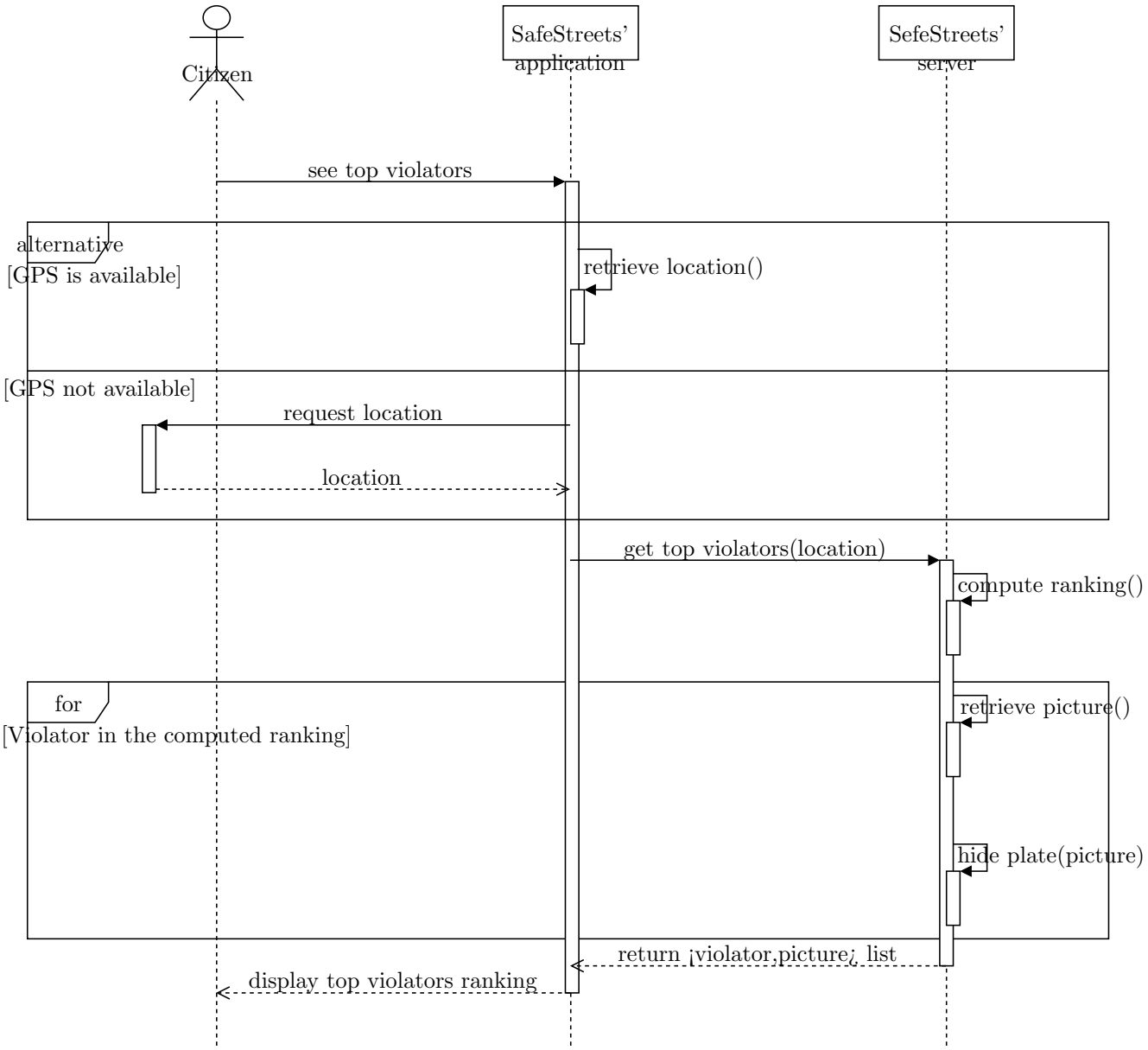


Figure 7: See streets' risk: sequence diagram

Table 8: Automated data mining

Actor	Authorities
Entry condition	• The user is signed in into the application
Events flow	1. (SafeStreets' server will periodically run algorithms for data mining and data analysis, to identify and report useful informations to the authorities of the city) 2. An authority clicks on MINE DATA button 3. A request is sent to the server 4. If the background computation spotted some useful informations about the city of competence of the authority, these are sent to the authority's device 5. The informations are presented to the user
Exit condition	The user can see the requested data presented on its device
Exeption	• Server is unreachable • The service for map creation is not available • The needed data cannot be retrived from the database

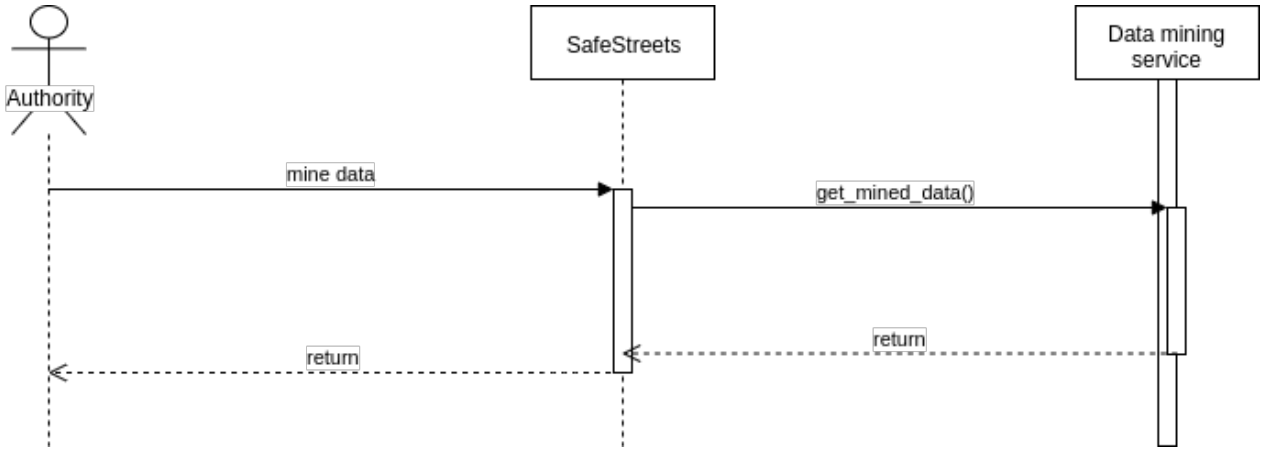


Figure 8: Data mining: sequence diagram

Table 9: Issuing tickets

Actor	Authorities
Entry condition	• The user is signed in into the application
Events flow	1. User see all the recieved photos from citizens by order 2. User clicks on one of them to analize the details 3. Decide to choose Issuing the ticket or Ignore it (for some reason such as this licence plate belongs to some justified people) 4. **Provide infos about the ticket (ex: how much money**
Exit condition	• The authority releases a ticket for the choosen violation • If the owner of the plate which recived the ticket is a SefeStreets user, he's notified of the ticket
Exeption	• Server is unreachable • Violation is discarded

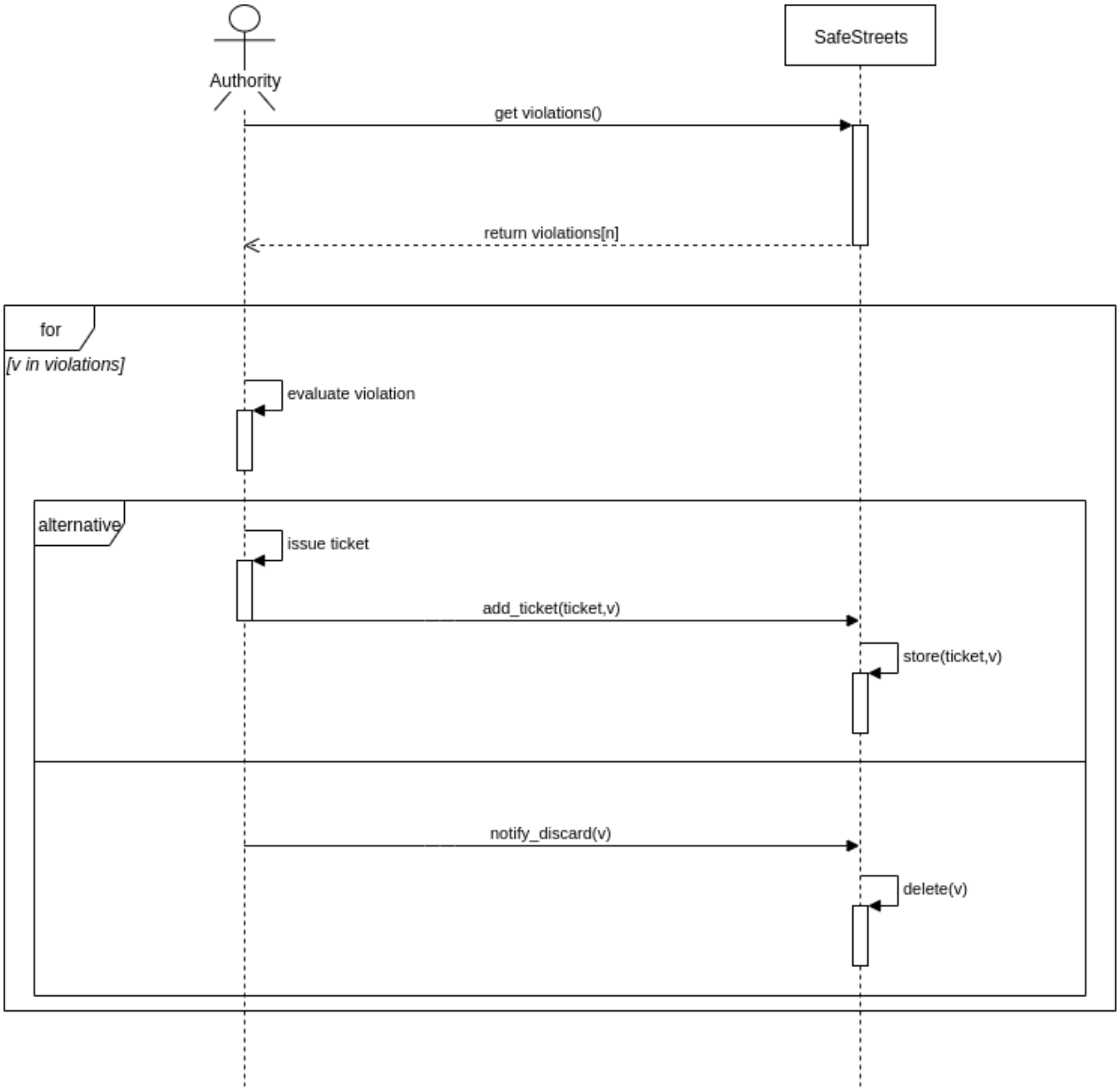


Figure 9: Issue ticket: sequence diagram

2.2 Product functions

The main functions of the system follows (lines in parenthesis are outside the system domain but are inserted for completeness):

Violation notification This is the core function of the system, exploited by the citizens. This function can be divided in 3 parts:

- **Data gathering** When the user spots the violation and with the help of its phone fills all the data (hopefully many will be gathered automatically) and sends them to the server
- **Elaboration** When the data are recived by our server the integrity of the data is checked; then a computer vision algorithm elaborates the plate. Finally, all the encoded data are stored to the database.
- **Acknowledgment** After the elaboration, the user recives a message confirming everything was ok, or asking more data if something went wrong during the elaboration.

Violation examination This is the way the system could deeply influence the world. This function, exploited by recognized authorities, is the first step for automatically releasing tickets (which is not doable as pointed out by `C.2`. In particular, the authority will:

- Access the server from its interface
- Retrieve data about violation
- Check the data
- (If data are correct, decide if a ticket can be created and in case create it)

In case a ticket is issued to a SafeStreets user, he/she will be notified about it, and will recive informations about the violation and the fee of the ticket itself.

Statistics This funciton can be exploited by any user. Basically, it concerns the displaying of statistics about violations in a graphic, map-based **structure**. It is composed by the following steps:

- A data-mining service runs on our servers, mining the database
- Data are aggregated by some rule
- Aggregated data are ordered
- The ordered aggregated data is displayed in a propter way

The way data are aggregated, ordered and displayed are now listed:

Map-based Data are displayed in a map marking the streets of the city with different colors, one for each identified "class of risk". A street is identified with a class of risk based on the number of violations in that streets in the last 3 months.

List of vehicles Veichles are sorted by number of violations, the ones with the highest number comes first. Depending on privacy laws, a picture of the vehicle taken from the pictures of violations, with the plate hidden, could be displayed

City data General statistics about the violations in the city can be viewed from authorities to decide about **SOMETHING TO DO**

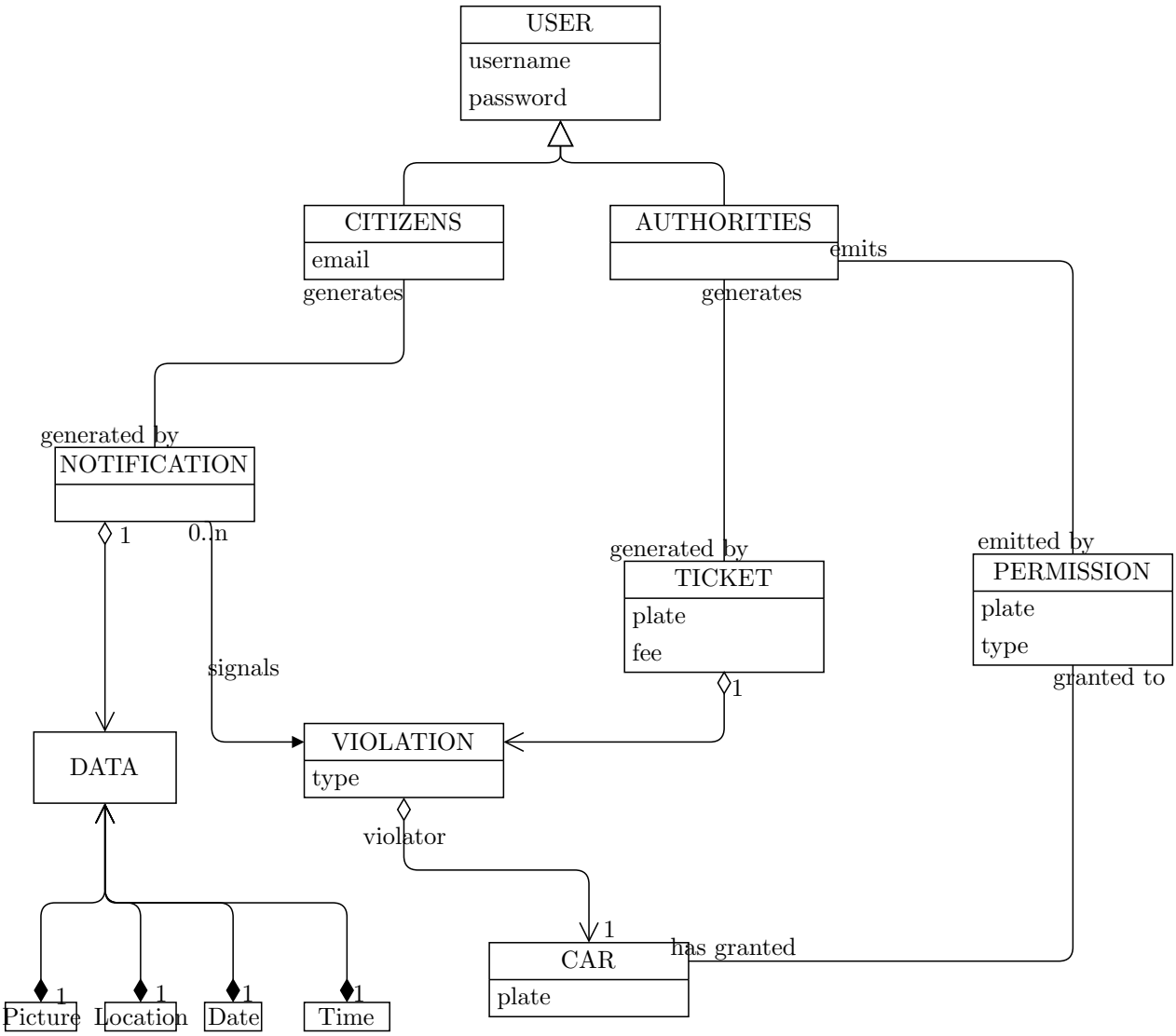


Figure 10: High level class diagram for violation notification and examination

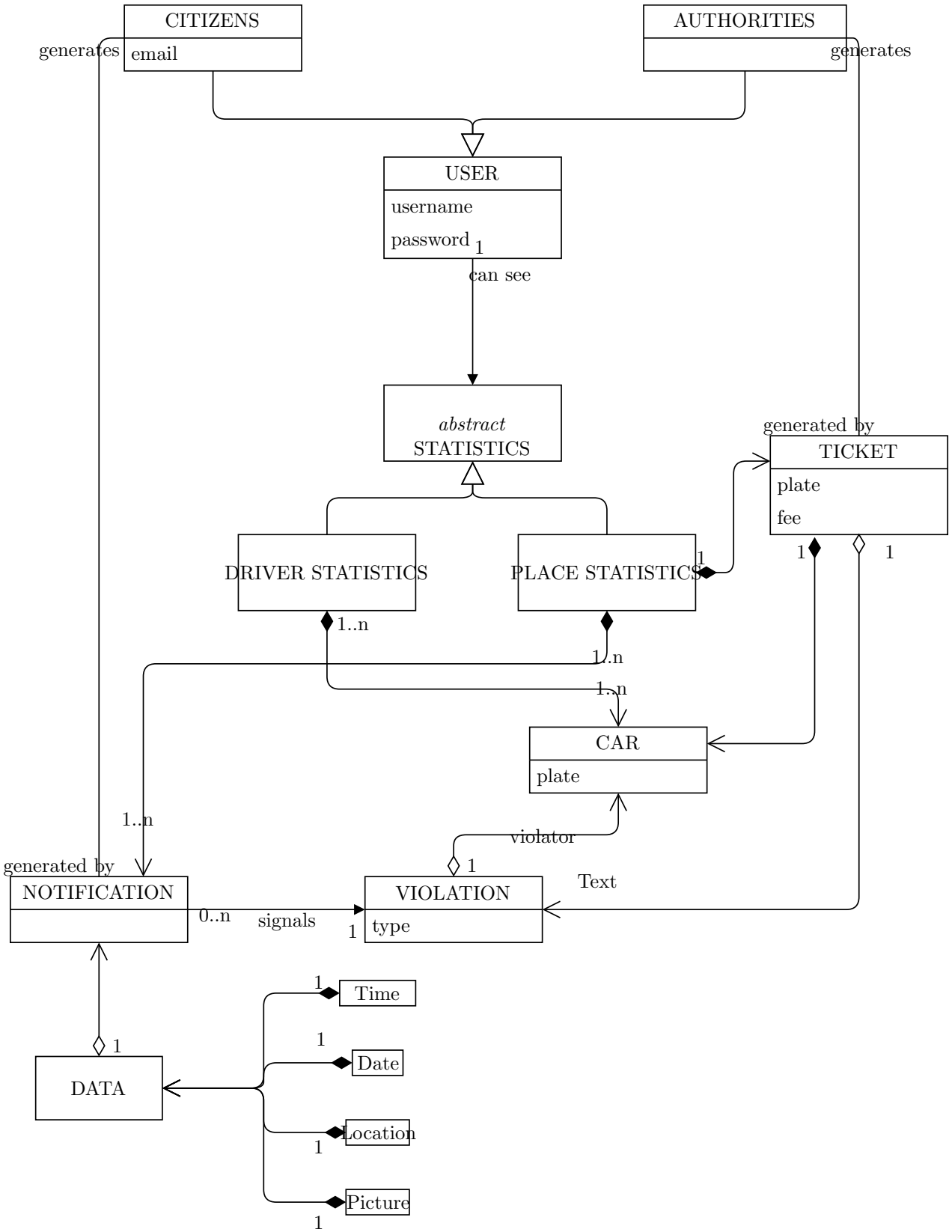


Figure 11: High level class diagram for Statistic production and presentation

2.3 Assumptions, dependencies and constraints

Assumptions:

`A.1` Different cars always have different plates

`A.2` Each car exactly has 1 plate

`A.3` No car is owned by more than 1 person

`A.4` Every auth will have access to any permission released by any other auth. SafeStreets will have access too.

`A.5` Every modified notification has a corresponding non-modified one (even only theoretical)

`A.6` When a notification is created it is unmodified. Modifications can happen later, generating a new notification (see `A.5`)

`A.7` If a notification is not stored then no auth can see it and therefore no ticket can be released for that notification

Constraints:

`C.1` The authorities will not be able to register automatically to the service. For authentication, they'll be verified and added by an administrator of the system

`C.2` Due to italian law, the system won't be able to automatically produce tickets

3 Specific Requirements

Here we include more details on all the aspects in Section 2 if they can be used for the development team

3.0.1 Scenarios

`S.1` Mario is walking down the street when he comes to a cross. He would like to **cross the cross** staying on the pedestrian lines, but a monstertruck decided to park over there. So Mario needs to walk around the truck and through the street (cars are driving fast), but at least he can take a little revenge by taking out his mobile phone, opening the SafeStreets application and take a picture of the plate of the truck. The application automatically acquires data on time and position using the gps and sends the data to SafeStreets' server. In a second, Mario is notified that the violation has been saved and will be avilable for the police. Mario now knows the crazy-minded driver will pay what he deserves! (like some fines)

`S.2` Luigi is late for work. He cannot find any parking, and is thinking about parking on the bycicle line. He's about to do that, when he suddently remembers that nowaday anyone could signal his infractions with SafeStreets. So he decides to keep searching for a proper parking, and granny Maria can take her daily ride on her bycicle!

`S.3` Veronica is a driver with moving disabilities. She must go for therapy to the Clinic 3times per week, today the specifiec car park for people with disabilities is occupied by a car with normal driver so she cannot park near the clinic and there is no other car park near and she decides to notify this violation citizenship right by Safe Street App. next time She parks her car the right place but a stranger think she was a violator and tries to notify by Safe Street App but stranger gives a notification by the App that this "Parking" is completely right becuase this car's plate is saved in the data base of Safe Street as having permission to park in this area.

`S.4` Antonio wants to take a revenge from one of his rommate(john) so he decides to take a photo of a violator car that the car is similar to john's car and he sticks a prepared paper with the number of john's car's plate on the violator car and send the photo but the Safe Street notify Antonio that his notification is rejected.

`S.5` Larry is a police Officer who wants to issued ticket for someone by his hands but The App make force him that just tickets are confirm through the application are validated

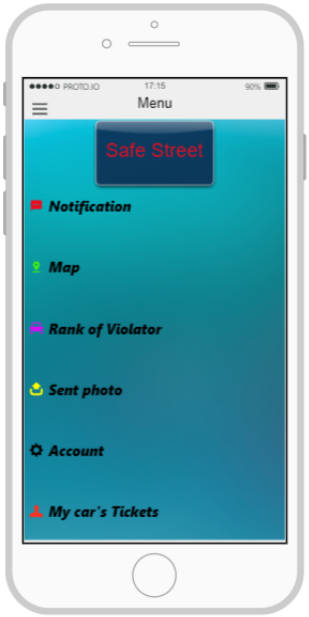
3.1 External Interface Requirements

These requirements come from the needing to communicate to the local police to submit notifications and gather ticket statistics, and to reduce the project cost by outsourcing.

3.1.1 User Interfaces

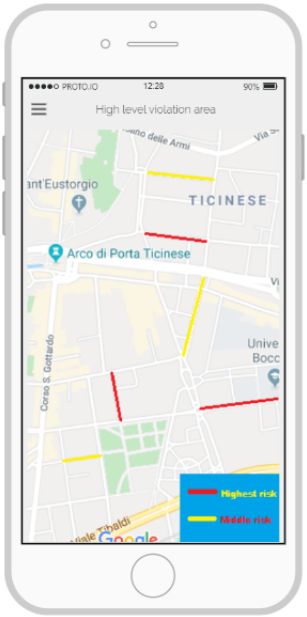
(a) Sign in

(b) Sign up



(c) Menu

(d) Menu



(e) Menu

(f) Menu

UI.1 A graphical user interface should be provided to the auths to allow them to retrive data manually. It could be web-based or desktop-based.

UI.2 The citizen's user interface should be a mobile application to both be able to ensure mobility and security, being able to control the dataflow.

3.1.2 Software Interfaces

SW.1 An api should be produced to allow auth's servers to access data programmatically

SW.2 An external api should be used to manage the map-based displaying of statistics

3.2 Functional Requirements

Definition of use case diagrams, usa cases and associated sequence/activity diagrams, and mapping on requirements

3.2.1 Requirements

R.1 Requires the authorities to give SafeStreets access to the tickets they emitted using SafeStreets' data

R.2 Automatically extract the plate number of the car from the photo, ignoring cars in the background if present

R.3 Ensure no data is altered from the insertion to the eventual storage

R.4 For each data to insert, if the user's device has a sensor to collect thatkind of data, that sensor should be used instead of manual insertion (example: GPS). Otherwise it will be possible to insert it by hand.

R.5 The system should notify the user when his notification has been processed correctly, or ask for more datailed data (example: a more focused picture) if needed.

R.6 If a notification is missing any needed data, the client application will prevent the user from sending it

R.7 No modified notification will be stored by SafeStreets

R.8 If a violation is invalidated by a permission, it won't be stored

R.9 All notifications which are not modified and neither covered by a permission will be saved

3.3 Performance requirements

P.1 The notification should be sent to the server as soon as possible. It should be recived and elaborated as soon as internet and the server's speed allows to match almost-real-time speed

P.2 The quality of the photo should be at most XYZ (to reduce overload on network and server) but at least PQR (to enable automatic recognition).

3.4 Design Constraints

3.4.1 Hardware limitations

HW.1 The auth's hardware will require an internet connection to access the data.

HW.2 The cit's device will have to be a smartphone for the reasons explained in UI.2

3.4.2 Any other constraint

OT.1 The citizen's device should be checked to be trustable, and rooted devices should be blocked (as they could change the app behaviour and submit altered photos)

3.5 Software system attributes

3.5.1 Reliability

The only critical part of the system is the one concerning data mining, as people could exploit it to decide about it's daily routine. For this reason, it's recommended to make the result of the mining (old ones if needed) more available than the rest of the system. In addiction, the system must avoid to generate fake notifications because of system faults to prevent innocent people from getting an unfair ticket.

3.5.2 Availability

As the system does not concern life-critical events, we think an availability of 99% of the time should be enought to ensure a good experience for both cits and auths.

3.5.3 Security

- Transmit data encrypted to prevent alteration in the network
- Only allow auths to access non-aggregated data

3.5.4 Maintainability

Maintainability is not a crucial issue as no updates are planned for the system.

3.5.5 Portability

Portability is a crucial aspect of the system, at least on the citizen's side: it is needed to allow them to submit a notification as they spot a violation. For this reason, the cit's client should be avaiable on as most mobile devices as possible.

3.6 Traceability matrix

Table 10: Traceability matrix

Goal ID	Req ID	Usecase ID
G.1	R.2 R.4 R.5 UI.2 P.1 HW.2	S.1
G.2	R.4 R.6 UI.2 P.2 HW.2 OT.1	
G.3	A.1 A.2 A.3 A.4 A.5 A.6 R.2 R.3 R.6 UI.1 SW.1 HW.1 OT.1	
G.4	A.5 A.6 R.3 R.4 R.6 HW.2 OT.1	
G.5	A.1 A.2 A.4 R.2 SW.1 HW.1	
G.6	A.1 A.2 A.4 P.2 HW.2	
G.7	R.1 R.4 R.6 UI.1 UI.2 SW.1 SW.2 HW.1 HW.2	

4 Formal analysis using alloy

We tryed to build a model using alloy to verify the satisfaction of the goals. As the reader can see in 4.2 our model is successful in doing this. As it can be seen from the alloy code, to enforce the goals we needed to introduce 5 new facts (which is the way requirements and assumptions are modelled in Alloy). The first two ensure unicity of Notifications and Permissions, which can be a reasonable domain assumption. The third one ensures that every time a notification is stored, it generates a ticket. This property was used to avoid to take into account the time between storage and ticket issue, but should **not** be enforced in the final system. The fourth fact enforces the system to store or discard every notification, and notify the user about what happened (so it's again to ignore waiting/processing phases). The last fact is probably the most important, is a new requirement we found out from the formal analysis which is explained better in section 4.3

4.1 Code

```
open util/boolean

sig Citizen{
        sent: set Notification,
        acked: set Notification,
        nacked: set Notification,
        plates: set Plate
}{
        acked in sent and nacked in sent and acked&nacked=none
}

sig Authority{
        releasedPermissions: set Permission,
        knownPermissions: set Permission
}{
        releasedPermissions in knownPermissions
}

sig Permission{
        type: one ViolType,
        plate: one Plate
}{
        some a: Authority | this in a.releasedPermissions
}

one sig SafeStreets{
        knownPermissions: set Permission,
        knownTickets: set Ticket,
        storedNotifications: set Notification
}

sig Violation{
        pic: one Picture,
        date: one Date,
        t: one Time,
        loc: one Location,
        type: one ViolType
}{
        (no disj v1,v2: Violation | extractData[v1] = extractData[v2])  //No duplicated violations
and     (some n:Notification | n.viol = this)   //all violations are notified (useless to consider
    ↪ unnoticed ones)
}

sig Notification{
        viol: Violation,
        data: set Data,
        modified: Bool
}{
        (modified = True implies (some n:Notification | this≠n and modify[n,this]) else (data = extractData
            ↪ [viol])) //A5,A6: no self-generated modified tasks, new notifications are generated by a
            ↪ violation
and     fullData[this]
and     (one c: Citizen | this in c.sent)
}

sig Ticket{
        viol: one Violation,
        notif: set Notification,
        plate: one Plate,
        releasedBy: one Authority
```

```
}{
        some n:Notification | this=generateTicket[n] and
        notif in SafeStreets.storedNotifications //A7: If a notification is not stored then no auth can
            ↪ see it and therefore no ticket can be released for that notification
        and all disj n1,n2: Notification | (n1.viol = n2.viol and n1 in notif) implies (n2 in notif)
}

sig Statistic{
        violations: set Violation
}{
        #Violation>0 implies #violations>0
}

sig Plate{}

abstract sig Data{}
sig Picture extends Data{plate: lone Plate}     //empty if plate cannot be retrived
sig Date extends Data{}
sig Time extends Data{}
sig Location extends Data{}
enum ViolType{t1,t2,t3}

//GOALS:
//G1: Allow citizens to notify parking violations and be acknowledged of the result as soon as possible
//For the functional part, enforce this function:
assert G1{
        (#Notification>0) implies (some n: Notification, v: Violation | n.modified=False and n.viol = v and
            ↪  n.data = extractData[v])
}

//G2: check every notification has full data
assert G2{
        all n: Notification | fullData[n]
}

//G3: check if there's a violation it is stored
assert G3{
        #Violation>0 implies #SafeStreets.storedNotifications>0
}


//G4: Ensure no tickets can be emitted if the notification's data has been modified somehow
assert G4{
        no t: Ticket, n: Notification | t = generateTicket[n] and n.modified = True
}

//G5: Ensure no tickets can be emitted if the plate of the car that committed the infringment owns a
    ↪ permission for that infringiment
assert G5{
        no t: Ticket | some per: Permission | per.type = t.viol.type and per.plate = t.plate
}

//G6: Every notification not covered by G4 or G5 will be eligible for ticket generation
assert G6{
        #Authority>0 implies all n:Notification | (n.modified=False and no p:Permission | (p.type=n.viol.
            ↪ type and (some pic:Picture | pic in n.data and pic.plate=p.plate))) implies
                (some t:Ticket | t=generateTicket[n])
}

//G7: Allow both citizens and authorities retrive informations about previous violations and released
    ↪ tickets, possibly in an aggregated form. Every violation reported to the system and stored will
    ↪ appear in some statistics
assert G7{
        all v: Violation | some s: Statistic | (v in s.violations <=> some n:Notification | n in
            ↪ SafeStreets.storedNotifications and n.viol = v)
}

//REQUIREMENTS
fact R1{//Requires the authorities to give safestreets informations about the ticket they release using
    ↪ the system
        all t:Ticket | t in SafeStreets.knownTickets
}

fact R2{//Automatically extract the plate number of the car from the photo, ignoring cars in the
    ↪ background if present
        no n:Notification | some p:Picture | (p in n.data and p.plate=none)
}

fact R3{//Ensure no data is altered from the insertion to the eventual storage
        no n:Notification | (n in SafeStreets.storedNotifications and n.modified = True)
}
```

```
//R4 helps but is not strictly needed for meeting the goals

//R7 R8 R9 are enforced by the "store" function
fact enforceStore{
        all n:Notification | store[n]    and ack[n]
}

fact enforceGenerateStat{
        (all s:Statistic | some l:Location | s=generateStats[l]) and
        all l:Location | some s:Statistic | s=generateStats[l]
}

//ASSUMPTION
//Every "valid" violation generates a ticket (only used for formal analysis)
fact everyNotificationStoredGeneratesATicket{
        all n:Notification | n in SafeStreets.storedNotifications implies some t:Ticket | t=
            ↪ generateTicket[n]
}

fact noDuplicatedNotifications{
        no disj n1,n2: Notification | n1.viol = n2.viol and some c:Citizen | n1 in c.sent and n2 in c.
            ↪ sent
}

fact noDuplicatedPermissions{
        no disj p1,p2: Permission | p1.type=p2.type and p1.plate=p2.plate
}

fact A1{
        no p:Plate | some disj c1,c2:Citizen | p in c1.plates and p in c2.plates
}

//A1, A2, A3 allows to ignore the definition of Car (as it is uniquely identified from its plate)
fact A4{//SafeStreets will have access to any permission released by the auths
        (no p: Permission | some a: Authority | not p in a.knownPermissions)
        and
        (all p: Permission | p in SafeStreets.knownPermissions)
}

//PREDICATES
pred modify[n,n':Notification]{ //Models the function of modifiyng a notification
        n.viol = n'.viol and not n.data=n'.data and n'.modified=True and n.modified = False
}

pred fullData[n:Notification]{  //true iff the notification contains all the required data (true even if
    ↪ more data are inserted)
        (some p: Picture | p in n.data) and
        (some d: Date | d in n.data) and
        (some t: Time | t in n.data) and
        (some l: Location | l in n.data)
}

pred notify[c,c': Citizen,v: Violation,n:Notification]{
        //c' is c after notification is sent, n is the notification generated by v
        n.viol = v and n.data = extractData[v] and n in c'.sent and not n in c.sent
}

pred store[n:Notification]{//models the checking of a notification and the eventual storage
        (n in SafeStreets.storedNotifications) <=> n.modified=False
                                                    and (no p:Permission |  (p.type=n.viol.type and
                                                                                (some pic:Picture
                                                                                    ↪  | pic in
                                                                                    ↪ n.data and
                                                                                    ↪  pic.plate
                                                                                    ↪ =p.plate))
                                                                                    ↪ )
}

pred ack[n:Notification]{
        all c:Citizen, n:Notification | n in c.sent implies n in SafeStreets.storedNotifications implies n in
            ↪  c.acked else n in c.nacked
}

pred isValid[n:Notification]{
        n.modified=False and no p:Permission |  (p.type=n.viol.type and some pic:Picture | pic in n.data
            ↪ and pic.plate=p.plate)
}

//FUNCTIONS
fun extractData[v:Violation]: set Data{
        v.pic + v.date + v.t + v.loc
```

```
}

fun generateTicket[n: Notification]: lone Ticket{
        {t: Ticket | t.viol = n.viol and t.notif = n and t.plate = n.viol.pic.plate}
}

fun generateStats[l: Location]: one Statistic{
        {s: Statistic | all v: Violation | (v in s.violations <=> (v.loc = l and some n:Notification | (n.
            ↪ viol = v and n in SafeStreets.storedNotifications))) }
}

//WORLDS
pred W1{
        #Authority = 3
and     #Citizen = 5
and     #Violation = 4
and     #Plate = 4
and     #Data <= 8
and     #Permission = 1
and #Ticket = 2
and no disj n,n': Notification | some p:Picture | p in n.data and p in n'.data  //Theoretically possible
    ↪ to have identical picture, but messes the diagrams
}

pred W2{
        #Authority = 1
and     #Citizen = 2
and     #Plate = 4
and     #Permission = 0
and     #Violation = 4
and     #Ticket = 4
and     #Location = 4
and     all d:Data | some n:Notification | d in n.data
and no disj n,n': Notification | some p:Picture | p in n.data and p in n'.data  //Theoretically possible
    ↪ to have identical picture, but messes the diagrams
}

check G1 for 10
check G2 for 10
check G3 for 10
check G4 for 10
check G5 for 10
check G6 for 10
check G7 for 10

run W1 for 10
run W2 for 10
```

4.2 Satisfied goals

```
Executing "Check G1 for 10"
    Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
    71928 vars. 3030 primary vars. 180513 clauses. 961ms.
    No counterexample found. Assertion may be valid. 1307ms.
Executing "Check G2 for 10"
    Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
    69405 vars. 3040 primary vars. 175918 clauses. 935ms.
    No counterexample found. Assertion may be valid. 354ms.
Executing "Check G3 for 10"
    Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
    69226 vars. 3030 primary vars. 175732 clauses. 430ms.
    No counterexample found. Assertion may be valid. 225ms.
Executing "Check G4 for 10"
    Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
    70421 vars. 3050 primary vars. 179721 clauses. 354ms.
    No counterexample found. Assertion may be valid. 16ms.
Executing "Check G5 for 10"
    Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
    69755 vars. 3050 primary vars. 177300 clauses. 350ms.
    No counterexample found. Assertion may be valid. 4581ms.
Executing "Check G6 for 10"
    Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
    70865 vars. 3040 primary vars. 180594 clauses. 346ms.
    No counterexample found. Assertion may be valid. 50851ms.
Executing "Check G7 for 10"
    Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
    69630 vars. 3040 primary vars. 177016 clauses. 366ms.
    No counterexample found. Assertion may be valid. 1064ms.
```

4.3 Generated worlds

The first 3 images referes to the same instance, where the theme has been changed in order to highlight different properties. These instance was generated using the predicate W1.
In particular, 14 has the purpouse of showing the basic relationships used in notifiying a violation and the encoded data.

15 focuses on storage and information sharing: the permission in the picture is known by every authority in the instance, despite being released only from Authority2. Every ticket is known by SafeStreets (all the tickets in the model are intended as generated using SafeStreets, as manually addressed ones are out of the application domain), and the system only stores notifications that may (and do in this case, as the model is enforced to do so) produce a ticket. One by one: N0 was modified and then is not stored, N1 and N2 notify a violation of type 3, but the plate of the car which committed it owns a permission for these kind of infractions. N3 a was not modified, and its violation is not covered by a permission, therefore it can produce a ticket and is stored. N4 issues the same plate of N1 and N2, but for a violation of type 2, and then it is stored too and produces a ticket.

The main focus of 16 is to show how a Statistic only uses violations having a related stored notification, and so a ticket issued on them.

17 has the puprouse of showing another important property: every Statistic is related (transitively) to a Location, andevery Location "is covered" by some statistics. This property, which will need to be enforced, allows a clean and correct management of map-based statistics

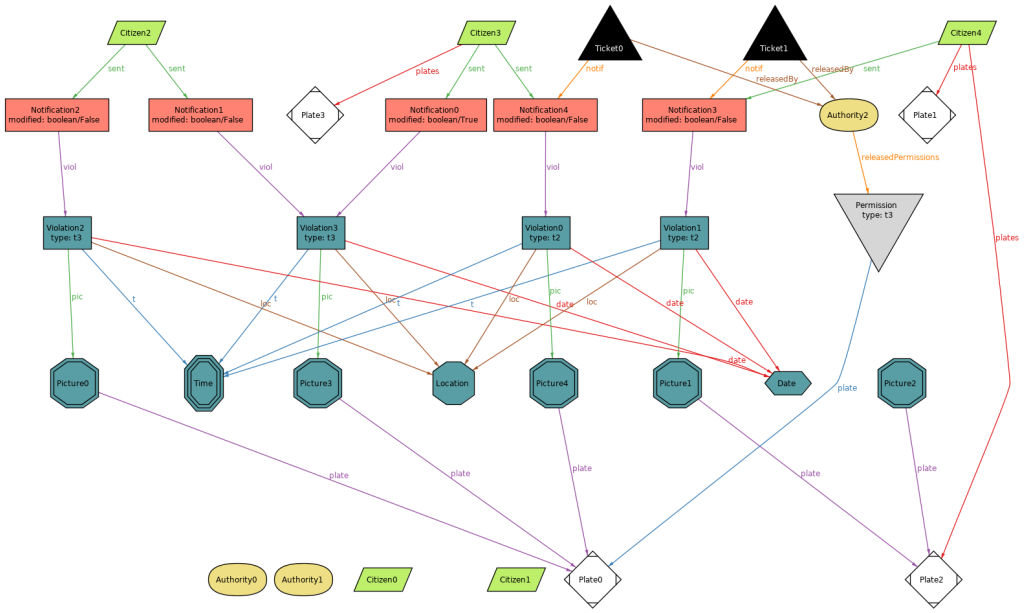


Figure 14: World 1, view 1

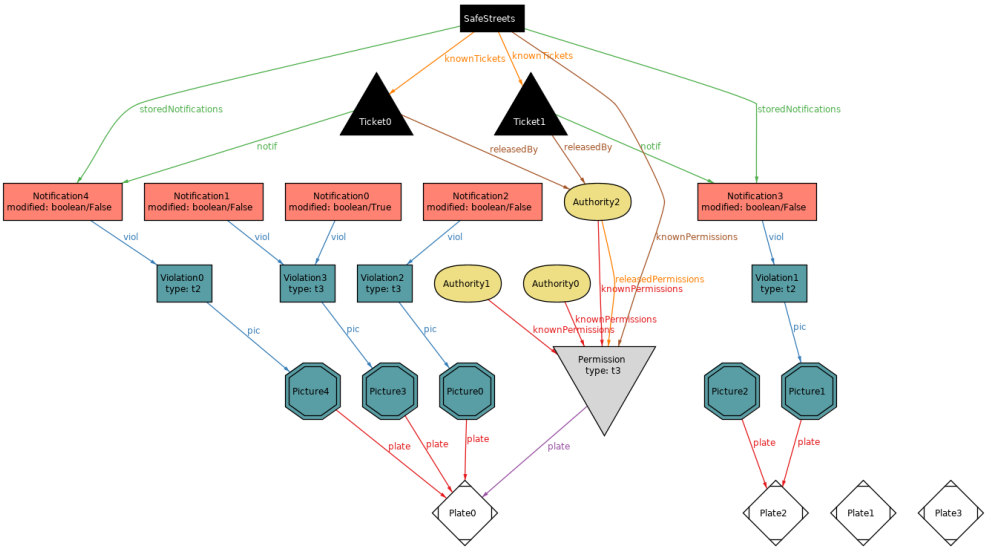


Figure 15: World 1, view 2

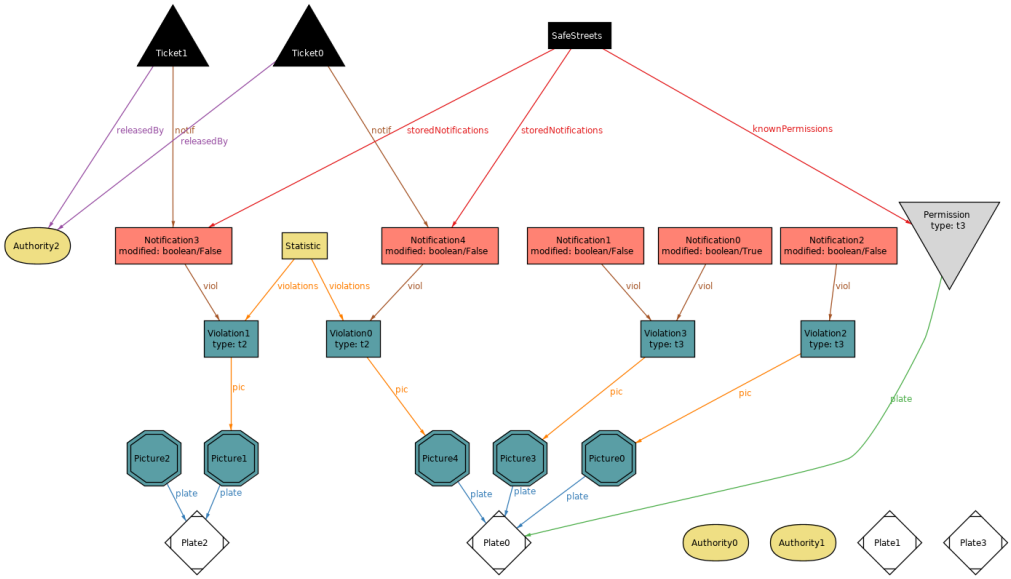


Figure 16: World 1, view 3

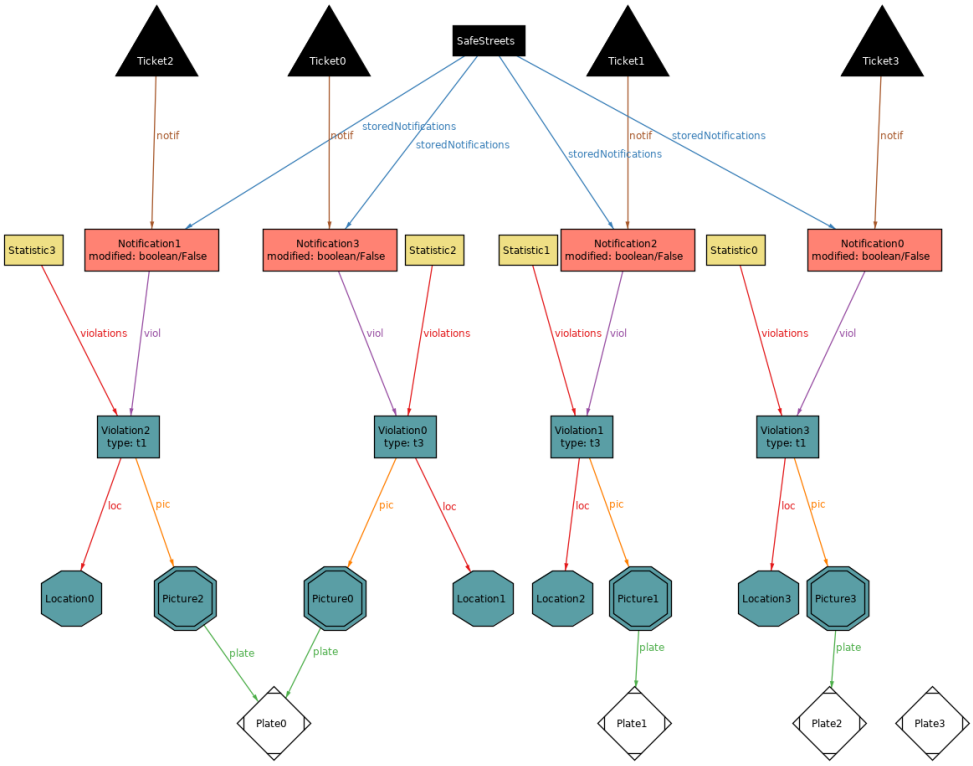


Figure 17: World 2

5 Effort spent

In this section you will include information about the number of hours each group member has worked for this document

Matteo Secco

- October 16: Introduction and Scope. 30m
- October 16: Definitions. 15m.
- October 23: Scenarios and goals. toghether. 1h
- October 24: Update goals, assumptions, alloy skeleton. 2h
- October 24: Teamwork. 2h
- October 27: Traceabiity matrix and related. 1.30h
- October 28: modify some Tasks toghether. 1h
- October 30: Design fixes, product functions. 1.30h
- October 30: Alloy. 1h
- November 2: Alloy. 1h
- November 3: Use-cases. 1h
- November 3: Alloy. 2h
- November 6: Alloy: 1h
- November 6: UML: 1h
- November 7: Phenomena. 30m

Rahbari

- October 23: Scenarios and goals. toghether. 1h
- October 24: Teamwork. 2h
- October 27: Use case diagram,product prospective 2h
- October 28: Modify some Tasks toghether. 1h
- October 30: use case table. 1h
- November 3: finishing all the use case tables 2h
- November8: working on Sequence Diagram 3h

6 References