Requirement Analysis and Specification Document

Michele Madaschi Lidia Moioli Luca Martinazzi

December 4, 2015

Contents

Introduction

1.1 Purpose

This document aims to describe, specify and analyze the software requirements for *My Taxi Service*.
My Taxi Service is needed to provide a passenger-friendly interface to interact with the city's taxi service, and ensure a fair management of the city-wide taxi deployment.

1.2 Scope

The passengers should be able to use an application (either mobile or browser based) to request a taxi through the system, which in turn should answer with the ETA and identification code of the incoming tax.
The passengers should be compelled to provide their current location to the system, for their request to be accepted.
The taxi drivers should be able to use a mobile application to communicate their availability to the system, and accept or refuse incoming calls.
The system shall manage a queue of taxis for each taxi zone[1].
The system shall receive GPS location data from each taxi, and use that information to assign each taxi to a taxi zone; an available taxi is automatically placed into the taxi queue belonging to the taxi zone it currently occupies.
The system shall remove a taxi from the queue upon receiving a confirmation in which the driver accepts an incoming call. If a taxi (driver) does, on the other hand, refuse an incoming call, the system shall move it to the bottom of its taxi queue.
The system matches a passenger's position to a taxi zone, and uses that information to forward the call to the first taxi available in the relative taxi queue.
The system shall provide an Application Programming Interface, to make room for future improvements.
The system shall also provide the possibility of requesting the reservation of

[1] See Definitions

a taxi; said reservation must occur at least 2 hours before the actual time of the ride; the time of the ride has to be specified by the passenger during the reservation procedure, as well as the passenger's location and destination.
However, the system will actually allocate a taxi (by means of removing it from the queue) only 10 minutes before the requested time of the ride.
On top of that, a "taxi sharing" option shall also be provided. The request of a shared ride shall trigger a process in which the systems looks for other compatible taxi-sharing requests, computes an adequate route, and calculates the amount of money each different passenger has to pay.

1.3 Domain properties

In this section we will analyze the background laying behind My Taxi Service:

- Passengers will pay at the end of the ride the amount of money demanded by the taxi driver
- Passenger reserve a taxi only in a period of three month starting from the current time
- Taxi drivers must own a valid taxi driving license
- A taxi can reach every position within the same zone in less than 10 minutes
- ETA is estimated with a maximum error margin of 5 minutes and it's supplied by the map service our system will interact with
- We assume GPS coordinates reliable
- We assume that, for each zone, if its queue is empty, the system must notify the unavailability of taxis

1.4 Goals

The passengers must be able to:

G1 Transmit its position and the desired destination to the system, thus initiating the Request of a taxi

G2 Receive the code and the ETA of the incoming taxi

G3 Reserve a taxi for a time period, starting at the time specified during the reservation[2], and ending after the ride is complete.

[2]the starting of the reserved ride must occur at least 2 hours after the time of the reservation

G4 Request a shared ride

The taxi driver must be able to:

G5 Answer a passenger's request

G6 Render him/herself available to the scheduler

G7 Receive informations regarding the fee defined for each passenger

The system must be able to:

G8 Offer a programmatic interface to enable the development of additional services

1.5 Definitions, Acronyms and Abbreviations

Passenger: the user who sends a taxi request.

Guest/Visitor : an huma interacting with the system withouth a related account.

User: an human interacting with the system identifiable with a related account. Users are split in 2 classes: 'passengers' and 'taxi drivers'.

System: the automatic part that manages the service.

ETA: estimated time of arrival.

Taxi zones: geographical partitions of the city, non overlapping, with an average size of 2Km^2.

Queue: a list of all available taxis in the corresponding taxi zone. It is managed as a FIFO queue. There is exactly one taxi queue associated to each taxi zone.

GPS: global position system.

Shared ride: a passenger shares the ride with other people that origins from the same zone, and go to the same direction

Active reservation : a reservation is considered active if it is in the reservation scheduler and isn't occured yet.

Same direction : two taxi rdies are considered to be going in "the same direction" if and only if the destination taxi zone is the same.

UI : User interface

Vehicle ID : in this document vehicle ID is a synonymous of "license plate"

1.6 Reference documents

We wrote this document helped by :

"Assignments 1 and 2" : the file that explain the service we need to development

"Requirements2" : the file containing the standard IEE for RASD

"RASD example SWIMv2" : an old RASD wrote for the meteocal project

1.7 Used tools

LaTeX : to create the PDF file

Pencil : to draw the mock up

Alloy modelling IDE : to develop alloy models

Dia : to draw class diagram (UML)

ArgoUML : to drawn use cases diagrams and sequence diagrams

Git : to esaly share files with teammates

1.8 Overview

This document is structured in 6 parts:

Introduction in which it is explained the purpose of the software, and specific project-related terms are explained

Overall description in which more focus is given to constraints and assumptions regarding the project

UI mockup in which are given sample screenshots, preparatory for the prototyping of an actual application

Specific requirements in which requirements, both functional and non-functional, are fully explained. Use cases are also located there.

UML in which is displayed an UML class diagram in aid of the future implementation of the system

Appendix in which is given an alloy modelization of project related aspects of the taxi system

Overall description

2.1 Product perspective

The system must interact with a map service, to retrieve information about the route to send to the taxi driver in case of shared ride.

2.2 Constraints

We will develop a unique mobile application that can be used by both passengers and taxi-drivers.
The web application will include only passengers functions.
The web application must be available for the most popular web browser (Mozilla Firefox, Google Chrome, Safari, Internet Explorer, Microsoft Edge)
The mobile application must be available for Android, Windowsphone and iOS.

2.3 User characteristics

The users must be connected to the network to use the application. Passengers can interact with the service through a web browser or a mobile application; they don't need any particular ability or foreknowledge to use it.
Taxi driver must access to the application with a device provided of GPS; since they must follow a standard procedure they must attend a formation course before starting (2 hours will be enough).

2.4 Assumptions

- A passenger is required to subscribe an account to utilize the taxi services (taxi request, taxi booking, taxi sharing)
- Taxi drivers can create only one account per vehicle ID

- Passengers who reserve a taxi can delete the reservation; if a taxi was allocated for the ride, the system will notify the taxi driver and put him at the top of the queue.
- In order to evade bad behaviour, we decide to implement a report system, that allow the service to disable users
- We also decide to add a master terminal interface, that allow the stakeholder to set up some parameters

UI mockups

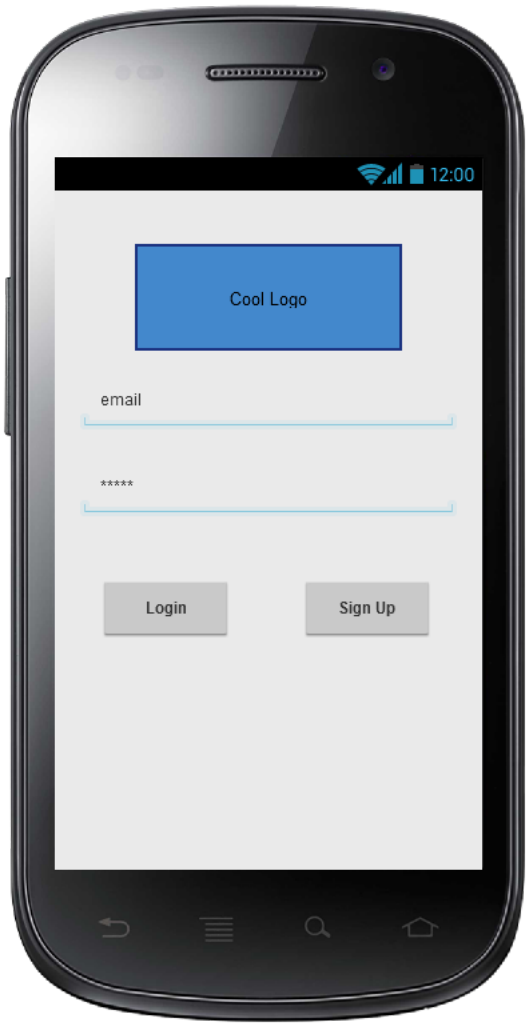


Figure 3.1: Login

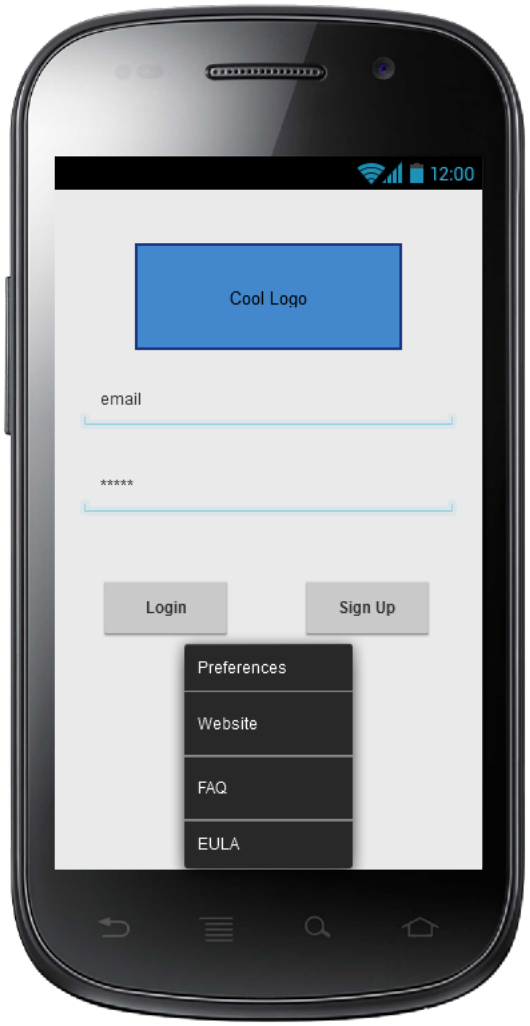


Figure 3.2: Login+menu

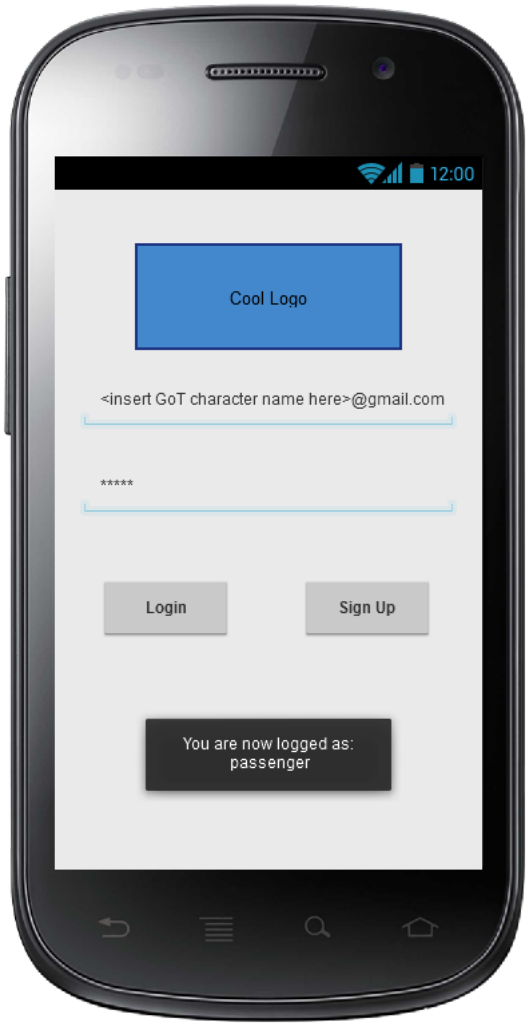


Figure 3.3: Login+toast

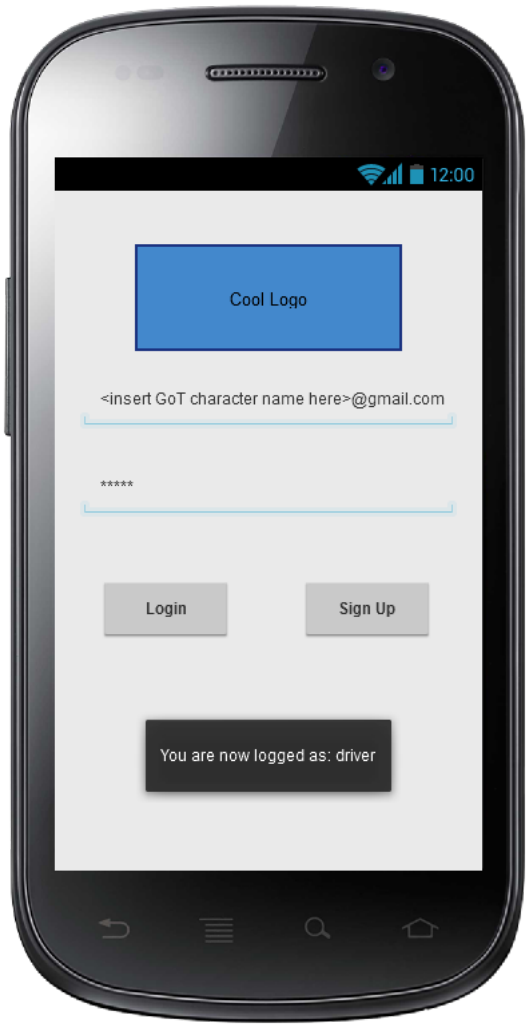


Figure 3.4: Login+toast (driver

Figure 3.5: Passenger map screen

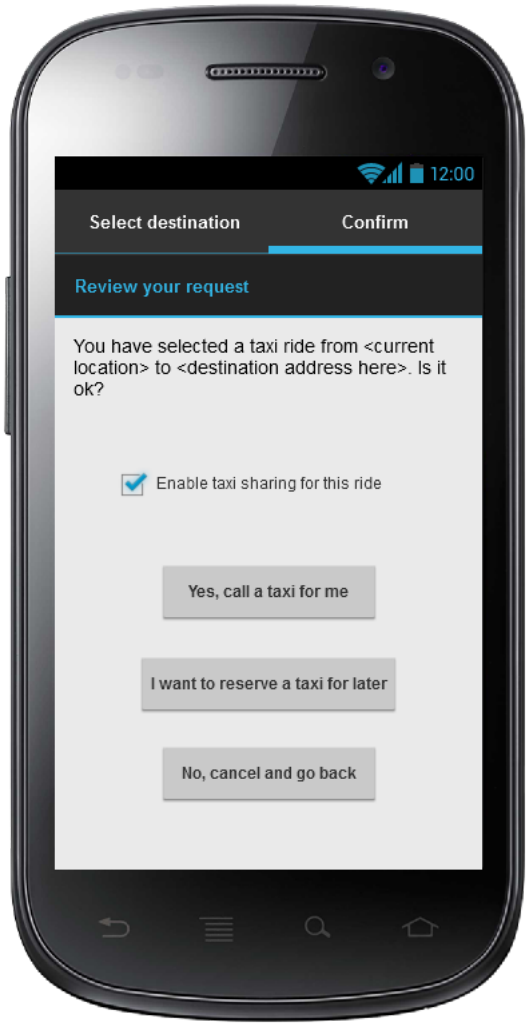


Figure 3.6: Passenger confirm screen

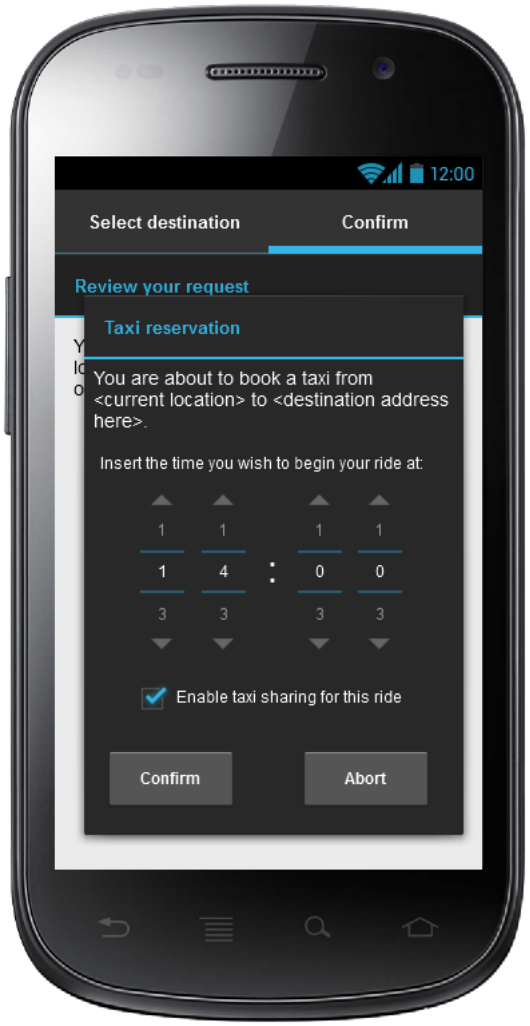


Figure 3.7: Passenger confirm screen - reserve

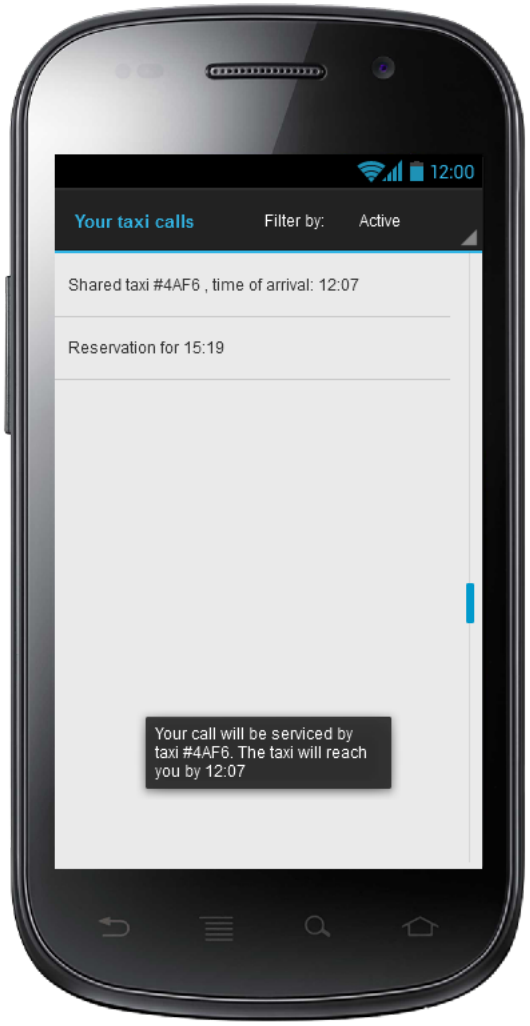


Figure 3.8: Passenger call list screen + toast

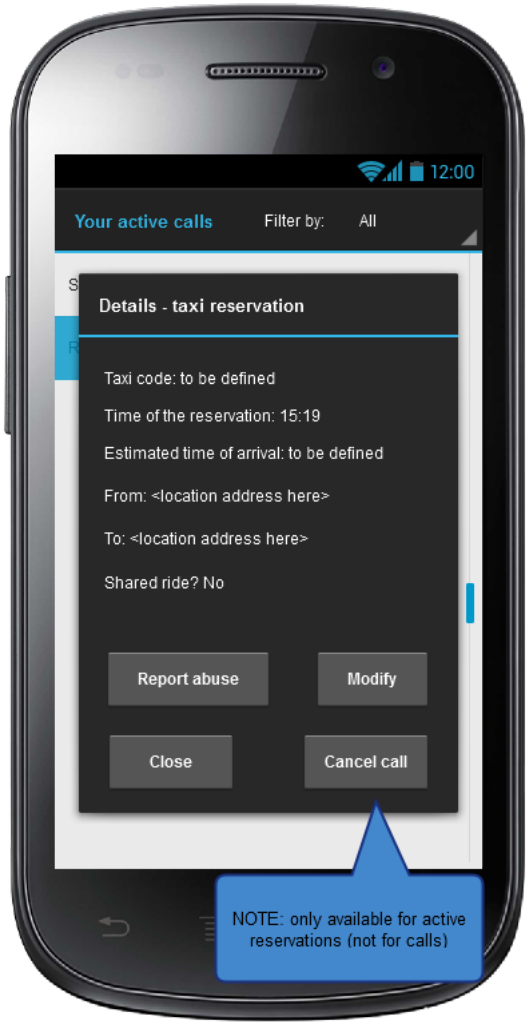


Figure 3.9: Passenger call list detail

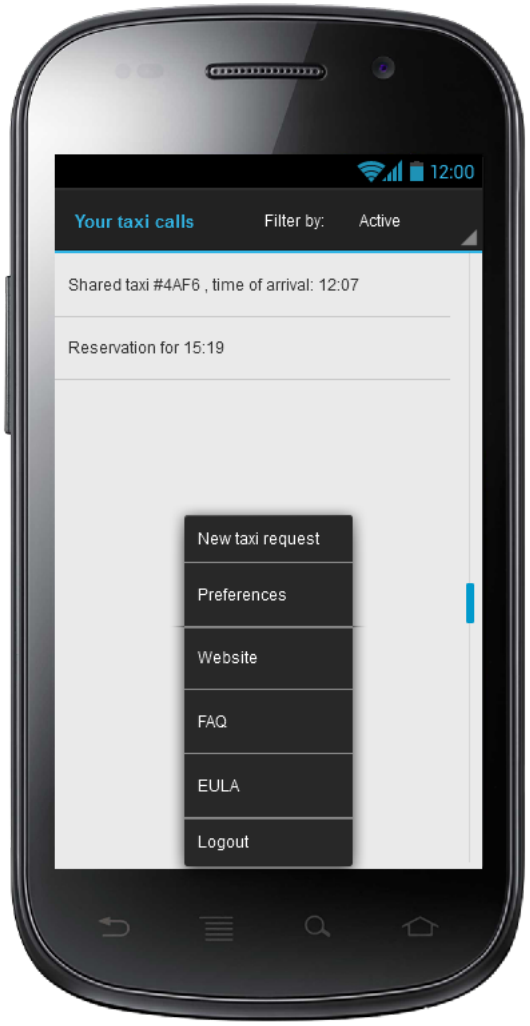


Figure 3.10: Passenger call list + menu

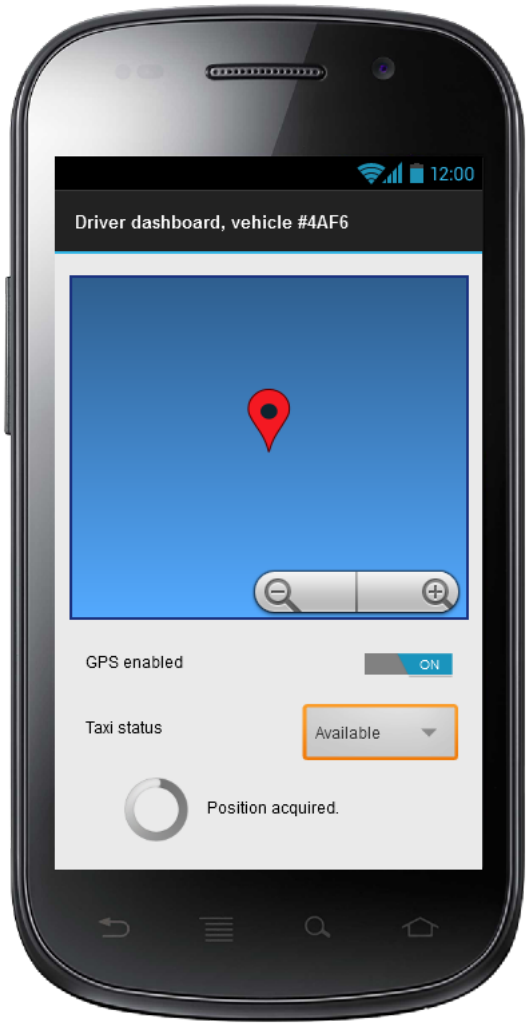


Figure 3.11: Driver Idle screen

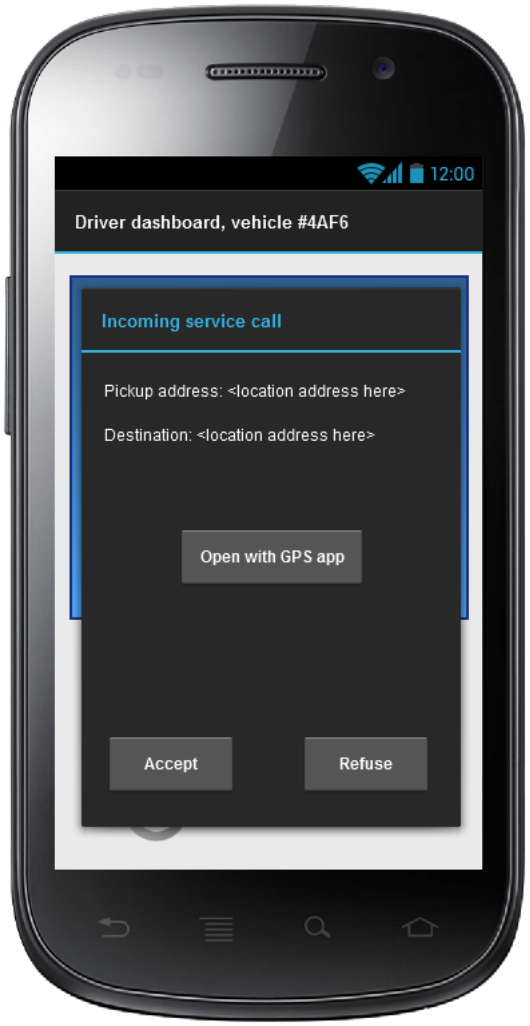


Figure 3.12: Driver new call dialog

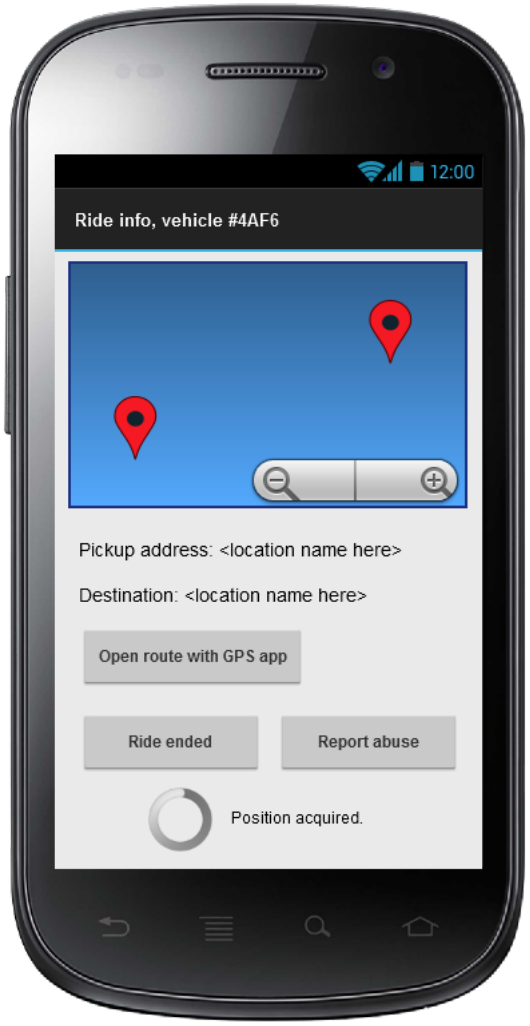


Figure 3.13: Driver ride info screen

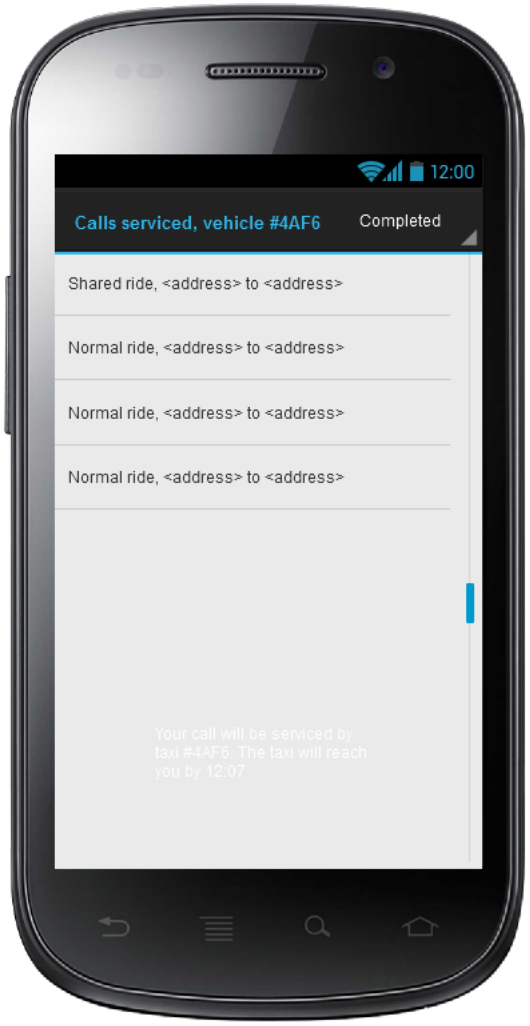


Figure 3.14: Driver call list

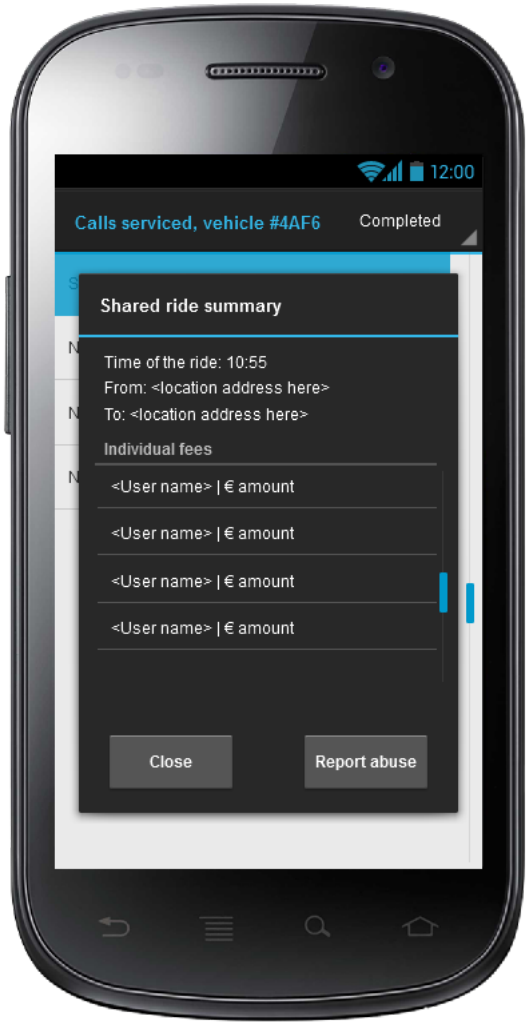


Figure 3.15: Driver shared fee dialog

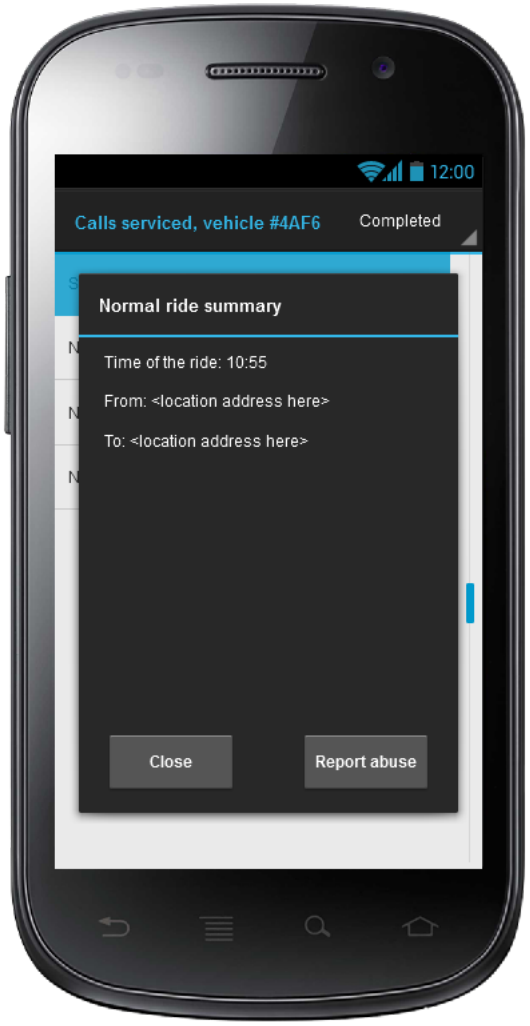


Figure 3.16: Driver normal ride dialog

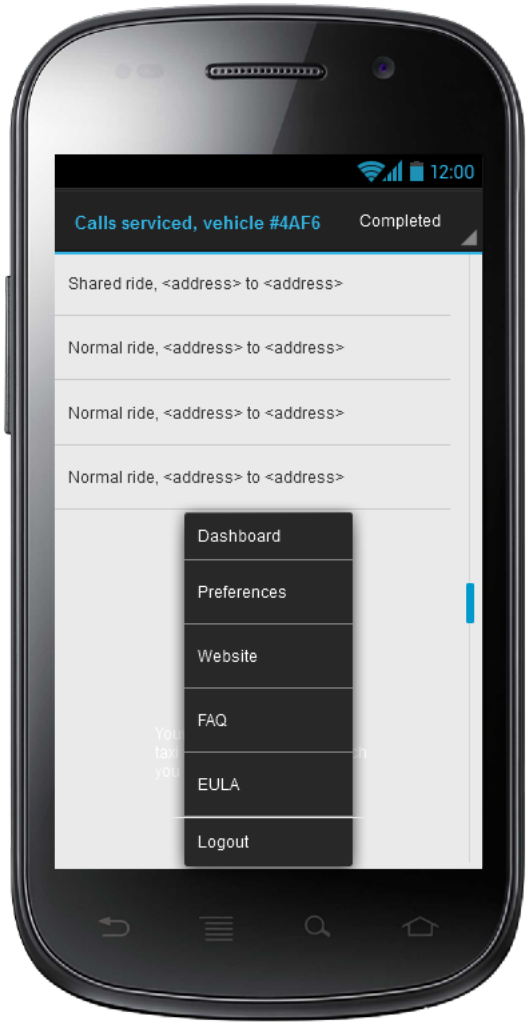


Figure 3.17: Driver call list + menu

Specific requirements

4.1 External interface requirements

The system must be able to comunicate with a map service, in order to retrieve information about the users' position, send the best route to the taxi drivers ,in case of shared ride, calculate the ETA for the passengers that requested a taxi.
Also the application must be able to interact with a map service, due to allow passengers to select the meeting position.

4.2 Functional Requirements

For each goal, we define the specific function that we will have to implement;

G1 :

- Persons can create an account;
- Persons can log into their account;
- Passengers can select from a menu the option of requesting a taxi as soon as possible;
- Passengers can insert their position filling an input form and confirm it;
- The system will receive the request and identify the zone in which the passenger is in;
- The system will forward the request to the first taxi in the selected zone queue and wait for an answer;
- If the taxist accepts, the system will remove him from the queue; otherwise it will append the taxist to the last position and scan the list for a taxist to accept.

G2 :

- As soon as a taxist accepts a request, the system invokes the support system to calculate the ETA giving the position of the taxi and the position of the passenger;
- The system will communicate the taxi code and the ETA.

G3 :

- Passengers can select from a menu the option of reserving a taxi for a chosen ride and date;
- Passengers can insert the initial and final position, time and date, their email and confirm it;
- The system will receive the reservation and if it respects the 2 hour constraint it will send a confirmation;
- Ten minutes before the ride starts, the system allocates a taxi for it.

G4 :

- The application must have a selectable option labled:"share your ride", that allows passengers to enable the shared ride service. In case of non reserved ride, the application will ask passengers the amount of time they can wait for others people;
- When the system receive a request of a shared ride, it will search for others shared ride requests starting from the same taxi zone, and going in the same direction;
- When a new passenger is added to a shared ride, the system will interact with the map service, in order to retrieve a new route for the taxi driver, and to calculate new fees;
- When the timeout of one passengers ,added to the current ride, occur, the system will procede with the allocation of the taxi;
- After the taxi allocation, the passengers who requested the shared ride will receive, not only the taxi ID, but also the fee they have to pay.

G5 :

- The system must forward a taxi request in the following cases:
  1: A passenger has requested a ride.
  2: A taxi reservation is sheduled to begin in 10 minutes.
- If a taxi driver refuses to take care about a call, the system will move him at the end of the queue,and forward the request to the next taxi driver in the queue. If a queue is empty, the system will notify the passenger that there are no taxi available.

– If a taxe driver accepts to take care of the call, the system shall remove him from the queue.

G6 :

– A taxi driver logged in into the system can select the button " Ready ", then the system will notify the system that the logged user is ready to accept some passengr's call. The application also send the taxi driver's position detected with a GPS

– If the application needs to retrieve data from a GPS and this isn't available, it will remind the user to turn it on.

– When the system receive a notification , by a taxi driver, informing that he is ready to take care of some passengers, it will append the user in the queue corresponding to the taxi zone that include the position retrieved by the application.

G7 – When a taxi driver is assigned to a shared ride, the system will send him the route he needs to follow, and the fee amount every passenger have to pay

– When a driver is assigned to a non-shared ride, the system will send him the route he needs to follow, and the fee amount the passenger has to pay

G8 :

– It is also neccessary to develop programmatic interfaces that allow to customize the system, adding new features.

- Passengers can access a section, in which they be able to check the ID of the taxi assigned to their ride and manage (delete or modify) an active reservation.
- When a passenger delete a reservation , the system will remove it from the reservation scheduler and, if a taxi driver is already assigned, notify the taxist.
- A passenger can modify an active reservation changing position, date and time.
- The system will accept modification only if sent before the taxi allocation.
- The system will accept date and time modification if it occur at least two hours after the request or/and after the previous reservation.
- A taxi driver have the possibility to remove himself from the queue by clicking the:"Disable" button.

- The system will remove a taxi from the list if receive the corresponding request by the taxi driver, or if the taxist logged out.
- A master terminal interface must be implemented in order to allow, the stakeholder, to configure some parameters (the number of the taxi zones, the set of positions belonging to each zone, the number of reports (per day, month and year) needed to automatically ban a user and the maximum number of reports (per hour) a user can insert).
- Every time a report is added to a user, the system will check if the constraints inserted by the master terminal are satified, otherwise the system must automatically ban the user.
- The master terminal interface allows to manually ban users, or enable banned users.
- The system must refuse reports added by a user if the user has already reached the maximum number of reports (per hour) decided by the master terminal.
- When the system refuse a report, a notification appear on the user screen, reminding him that he has already exceeded the maximum number of reports for that hour.

4.3 The World and the Machine

Figure 4.18 shows the model of the system, according to Jackson and Zave's model "The world and the machine"

4.4 Scenarios

1. Jon is driving back home while he notices a strange noise coming from his car. He decides to stop at the first garage on the road and then call a taxi using an app he downloaded some weeks ago, MyTaxiService. The mechanic checks the car and tells him that the problem is quite serious and the car must stay there for some maintenance. He opens MyTaxiService app and he accesses his account with his email and password. Then he inserts his position and selects "call a taxi". The app informs him that he has to wait 5 minutes for taxi 13C to come over.
2. Brandon is going to Moscow in three days and must be at the airport at 3:00 pm. Since the parking fee at the airport is very high he decides to reserve a taxi that will lead him right near the airport

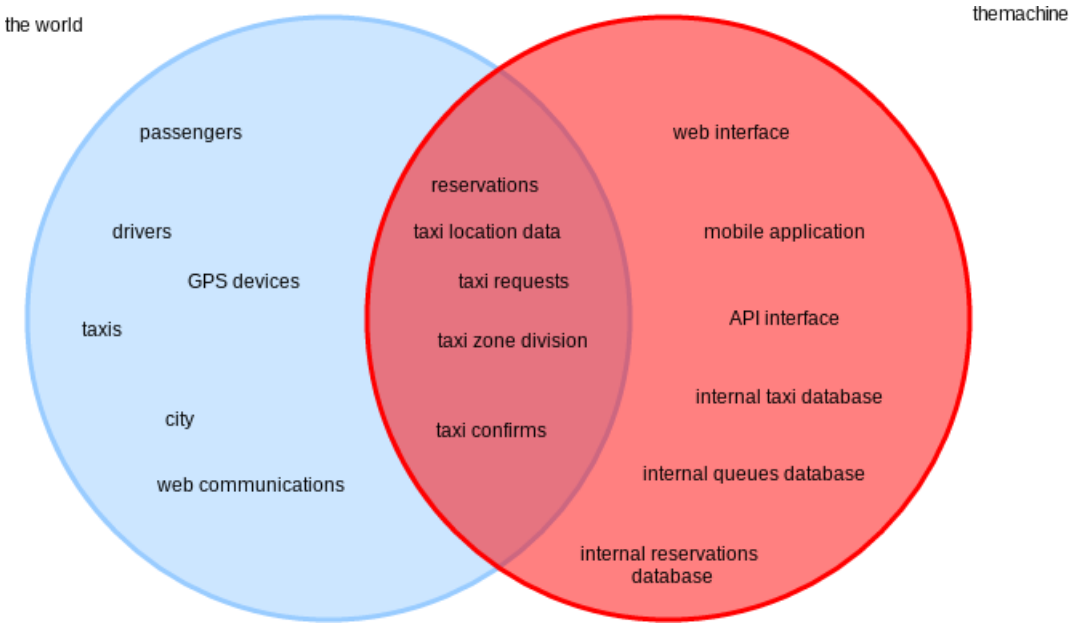


Figure 4.18: the world and the machine

entrance. Brandon searches on his personal computer for a taxi service and finds MyTaxiService, creates an account inserting his email address and inventing a password, he waits untill he receives a mail from MyTaxiService at the same email account he gave while signing in. He clicks the link in the mail and now his account is valid. He can reserve a taxi from his house to the airport for that day and see the details in his "active taxi list".

3. Eddard, a taxist, has just started a new day of work. As soon as he gets in his vehicle he logs into his account, inserting his code and password. Now he is connected with the system, a notification pop up informs him to turn on the gps. He activates the gps and he is sending correctly his position. Now he awaits for incoming calls.

4. Robb, a taxist, receives a notification of an incoming call. The request comes from Mario Street, not far from his position. Robb looks at the time, it's 12.58 meaning that his turn is over in two minutes, so he declines the request.

5. Arya wants to go to an exibition downtown. She has heard that the city centre will be closed at traffic but taxis will still be able to access. So she decides to book one, and in order to save some money she reserves a ride and enables the sharing option, hoping that also others will use the taxi and join her. She receives a notification from the taxi

service in which they confirm the taxi and that 3 more people will use the same ride so the cost will be only of 5 dollars.

6. Rickon, a taxist, receives a notification of an incoming call. It's a shared ride, the system provides him the route he has to follow, pick up a person in Golgi street then one in Grossich Street and leading them to Piazza Duomo, and the fees every passenger has to pay.

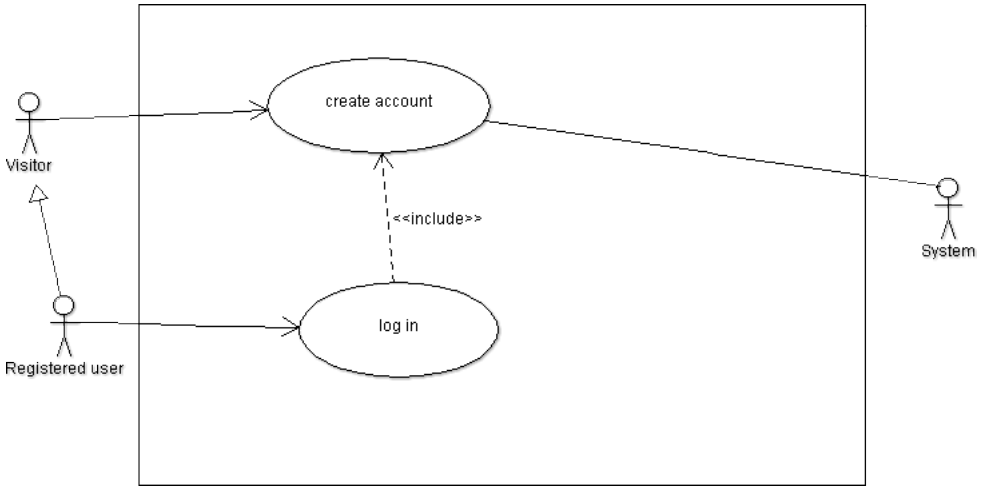
4.5 Use cases

- Create an account

USE CASE	A user can create an account into myTaxiService
ACTORS	Visitor
Entry condition	
Flow events	Visitor creates an account inserting his email address and choosing a password (longer than 8 characters). If he is willing to create an account as taxi driver, he must specify also his license number. The system processes the submission (in particular for a taxi driver checks the validity of the license given) and sends a confirmation mail to the given address in which there's a link that the user must click to validate his account.
Exit condition	A new account is created
Exception	If the email address is not correct or the user doesn't confirm his account, the system will not consider any request from it. If the email address already belongs to an existing account or the password is shorter than 8 characters, the system gives an error before the user can continue.

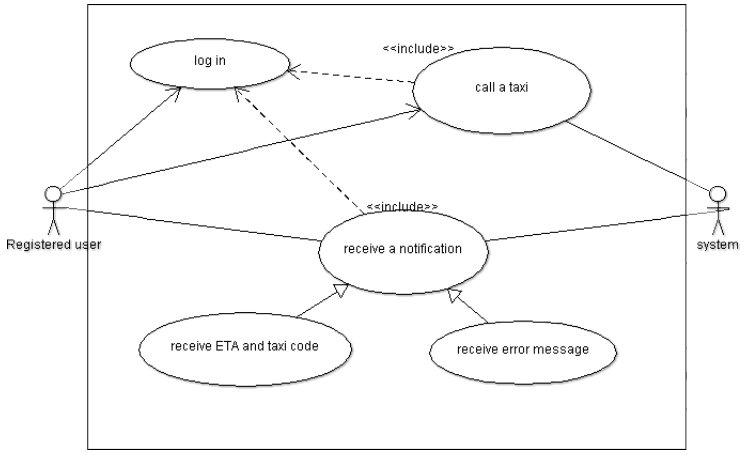
- Log in

USE CASE	A user can access his account
ACTORS	User
Entry condition	User must have already created an account
Flow events	User inserts his email and password.
Exit condition	The user sees his homepage, the taxist interface if he is a taxi driver or a passenger interface if he is not.
Exception	If the email doesn't correspond to an account the system gives an error message; if email is correct but the password isn't, the system gives an error message;



- Request a taxi

USE CASE	A passenger can request a taxi
ACTORS	Logged passenger
Entry condition	Passenger must have already logged into his account
Flow events	Passenger presses the button “call a taxi”, then inserts his adress (street/square, number). When he has done he clicks “send” button. The system checks if the location exists and then contacts the first taxist in the queue. When a taxist positively responds, the system look up for his position using his code and then calculates the ETA. Send both information, the taxi’s code and ETA to the user via a notification.
Exit condition	A taxi, no more in the queue of available taxis, takes charge of the call and it’s moving towards the passenger.
Exception	If the location doesn’t exists or the system doesn’t find this address in the city map: the system sends an error message; If there are no taxis in the queue: the system sends a notification saying there are no available taxis;.



- Book a taxi

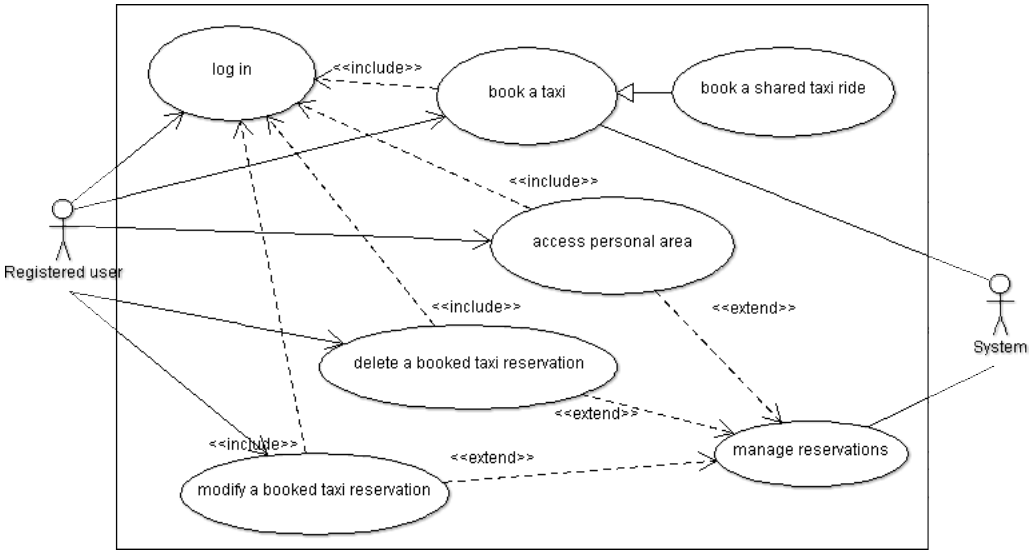
USE CASE	A Passenger can book a taxi
ACTORS	Logged passenger
Entry condition	Passenger must have already logged into his account
Flow events	Passenger presses the button “book a taxi”, then inserts the departure adress (street/square, number), the destination address (street/square, number), the date (dd/mm/yyyy 00.00). He can select the shared ride option. When he has filled all fields, he clicks “send” button. The system checks if the location exists and if the date respects the constraints then sends a confirmation back to the user and saves the reservation in the scheduler system for the day it occurs and in the queue where the ride will start.
Exit condition	A taxi is allocated ten minutes before the ride begins, and the passenger can see in his active taxi list the reservation.
Exception	If the location doesn’t exists or the system doesn”t find this address in the city map: the system sends an error message; If the date doesn’t respect the constraints an error message is sent.

- Book a shared taxi ride

USE CASE	A Passenger can book a taxi with shared ride option
ACTORS	Logged passenger
Entry condition	Passenger must have already logged into his account
Flow events	Passenger performs the same operation as booking a traditional ride but selects also the share ride option. Now the input form will also ask him which destinations he wants to share. When he has done he clicks "send" button. The system checks if the location exists and if the date respects the constraints then sends a confirmation back to the user and saves the reservation for the day it occurs and in the queue where the ride will start, in addition it will search for other reservations with shared option enabled that could have parts of the way in common and arrange a special route the taxist will follow.
Exit condition	A taxi is allocated ten minutes before the ride begins, and the passenger can see in his active taxi list the reservation. The system will send a notification with the price the user must pay.
Exception	If the location doesn't exists or the system doesn't find this address in the city map: the system sends an error message; If the date doesn't respect the constraints an error message is sent

- Modify a booked ride

USE CASE	A Passenger can modify a previous reservation
ACTORS	Logged passenger
Entry condition	Passenger must have already logged into his account and booked a ride, shared or not
Flow events	Passenger enters his taxi calls and visualizes all the active rides he booked. He can modify all the parameters (start, destination, date). In addition he can also delete a reservation.
Exit condition	The modification/deletion is effective.
Exception	If the location doesn't exists or the system doesn't find this address in the city map: the system sends an error message; If the date doesn't respect the constraints an error message is sent

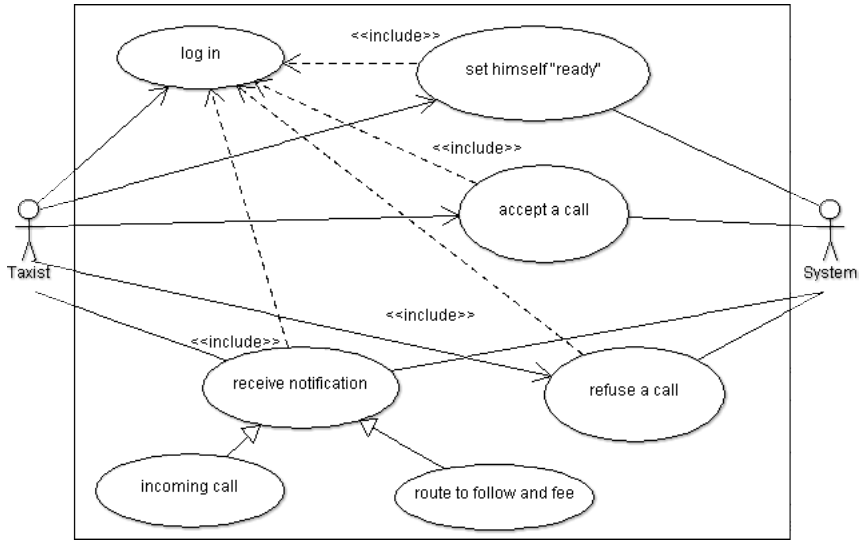


- Answer a call

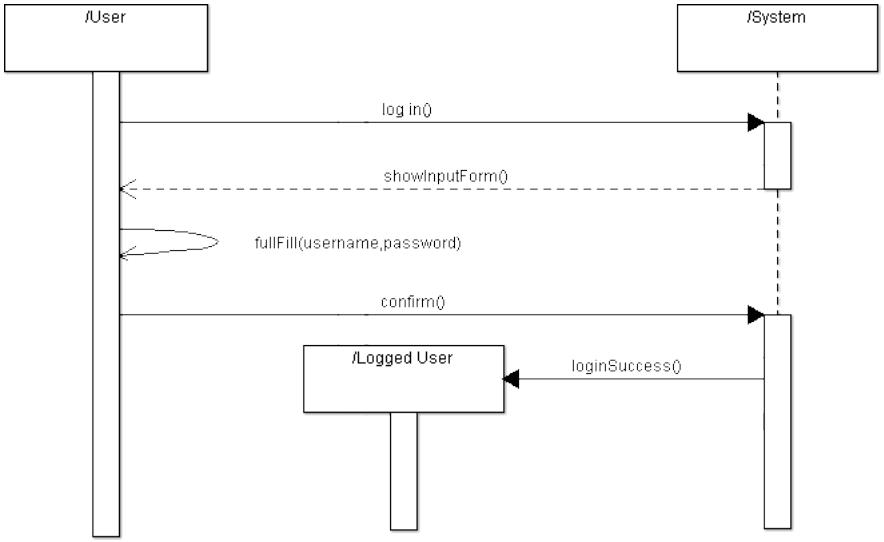
USE CASE	A taxist can answer a call
ACTORS	Logged taxist
Entry condition	Taxist must be working and available, the system must forward a valid request
Flow events	Taxist receives a notification on his mobile, with information about the ride (route to follow, time). He displays two option, accept or refuse. If he accepts he takes charge of the call otherwise he waits for another call. In the first case his code is sent to the system.
Exit condition	If the taxist accepts, he is no more visible in that queue to the system. If he refuses, he will be moved from first position in the queue to the last one.
Exception	

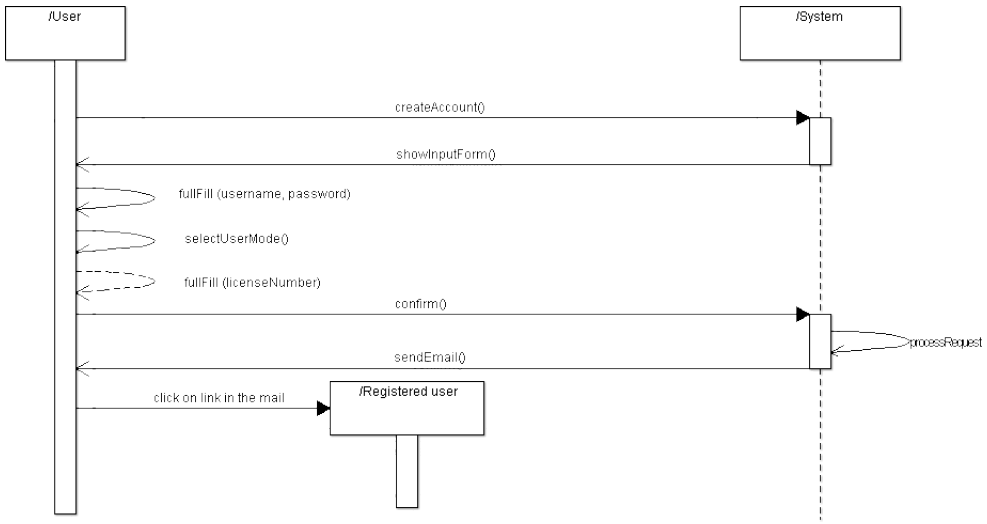
- Set availability

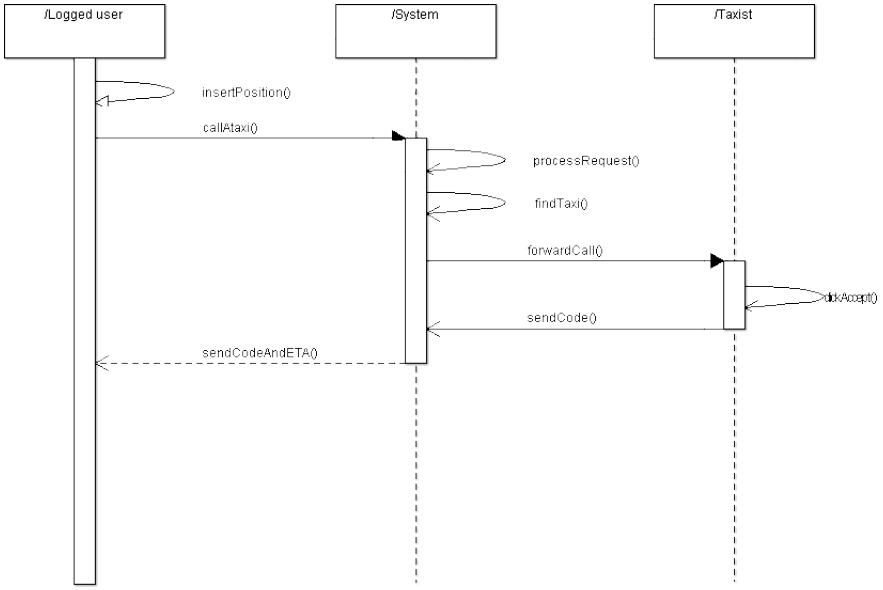
USE CASE	A taxist can render himself availablel
ACTORS	Logged taxist
Entry condition	Taxist must be working
Flow events	Taxist sends his availability to the system pushing the “ready” button.
Exit condition	The system receives his position and puts him in the queue in which he is in.
Exception	If the taxist hasn’t abilitate the gps, the app notifies him to turn it on.



4.5.1 Sequence diagrams




/User
/System
createAccount()
showInputForm()
fullFill (username, password)
selectUserMode()
fullFill (licenseNumber)
confirm()
processRequest
sendEmail()
click on link in the mail
/Registered user


/Logged user
/System
/Taxist
insertPosition()
callAtaxi()
processRequest()
findTaxi()
forwardCall()
tkAccept()
sendCode()
sendCodeAndETA()


/Taxist
/System
clickReady()
sendPosition()
updateQueue()
forwardCall()
RefuseRequest()
refuse()
updateQueue()

4.6 Software system attribute

4.6.1 Reliability

In order to easily react against failure, the system will make a backup of all the server and save it on a cloud service.

4.6.2 Availability

The system is completely automatized, so it will be available every day at every time, except for the first Wednesday of every month from 20:00 to 23:00 , when the server will be disconnected for maintenance, update and backup.

4.6.3 Security

The servers will be protected from external attack by adding two firewalls, located: between the network and the application server, and between the application server and a the data server. Furthermore, all communications between user and server will be protected via Transport Layer Security.

4.6.4 Maintainability

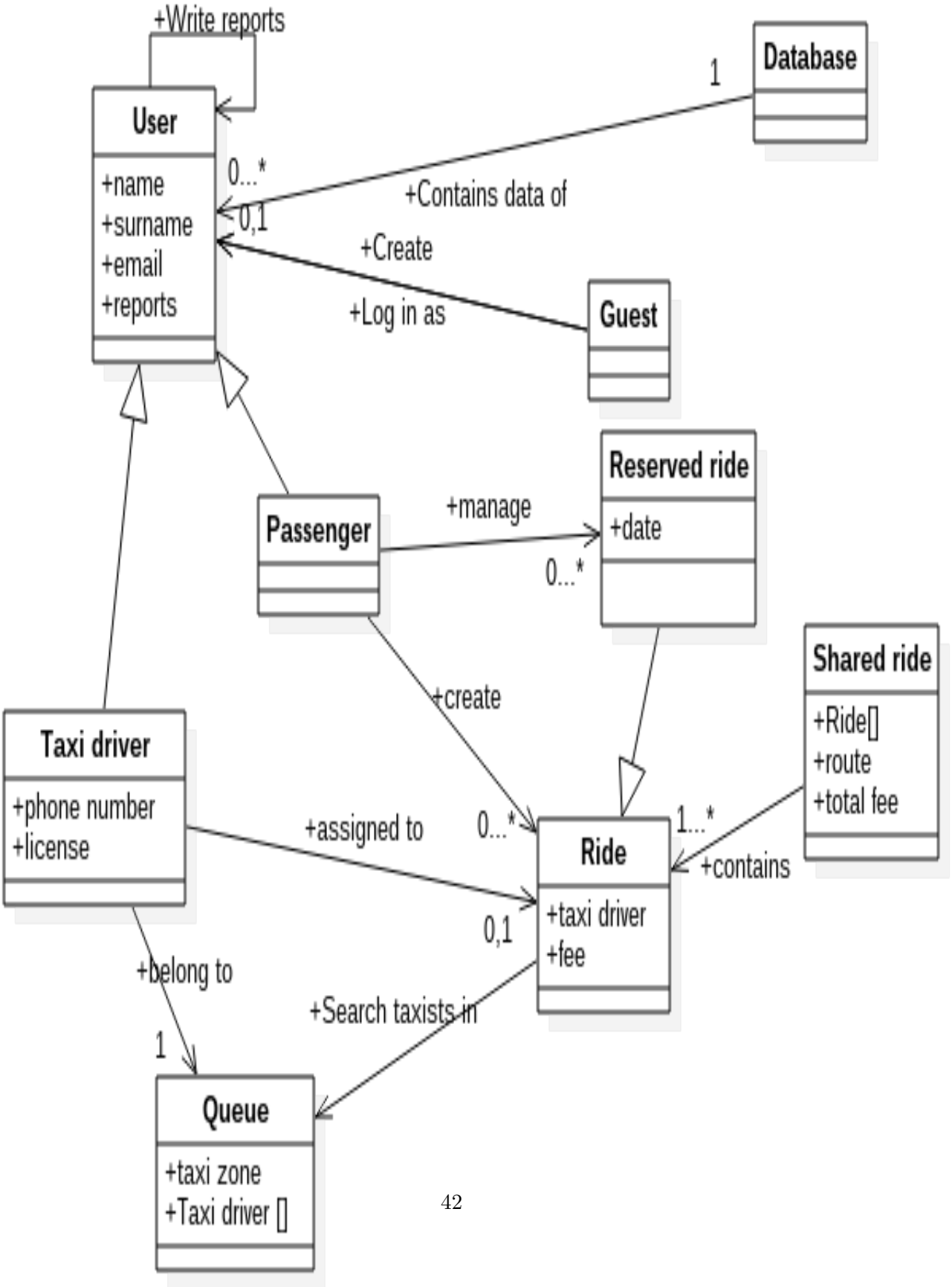
The application must provide set of API for the purpose of adding new features in future time. Those API must be thoroughly documented; the core system must be documented as well.

4.6.5 Portability

The server side application could be deployed on any platform supporting JRE-7.
The mobile application shall be developed for the major mobile operating systems (Android , iOS, Windows phone). The web application shall be compatible with the most widely used browsers (Mozilla Firefox, Google Chrome, Safari, Internet Explorer, Microsoft Edge)

UML

The UML printed below represent an idea of how the system must be developed, in order to make it easier to be read, we decide to insert only few example of activities and actions. An activity is a set of object that must be shown to the users (also the guests). An action is something that can be activated by the user, or also by a guest, interacting with the application.


+Write reports
User
+name
+surname
+email
+reports
1
Database
0...*
0,1
+Contains data of
+Create
+Log in as
Guest
Passenger
+manage
Reserved ride
+date
0...*
+create
Shared ride
+Ride[]
+route
+total fee
Taxi driver
+phone number
+license
+assigned to
0...*
Ride
+taxi driver
+fee
1...*
+contains
0,1
+belong to
+Search taxists in
1
Queue
+taxi zone
+Taxi driver []

Appendix

6.1 Alloy

Here is displayed the fully commented alloy code used to model the portion of city our system will interact with

```
sig GeographicalPosition{
}

sig Taxi{
        location: one GeographicalPosition
}

sig TaxiQueue{  // just a set of taxis
        taxi: set Taxi
}

sig TaxiZone{  // taxi-zone as defined in RASD
        queue: one TaxiQueue,
        positions: set GeographicalPosition
}
{
        #positions > 0 // an empty taxi zone would be
           useless
        all t: queue.taxi | t.location in positions
           // if a taxi is the queue, then its
           location must belong to the zone as well
}

sig City{
        // note: the model itself does not impose that
            there is only one city
        // there could be many overlapping cities
        // the limitation #City = 1 is applied
           elsewhere
```

```
        taxis: set Taxi,
        zones: set TaxiZone,
        positions: set GeographicalPosition
}
{
        all z: TaxiZone | z.positions in positions  //
            taxi zones are made of the same positions
           which make up the city
        all z: TaxiZone | z.queue.taxi in taxis  //
           taxi zones contain only taxis which are in
           the city
        all t1, t2: TaxiZone | !(t1 = t2) => t1.
           positions & t2.positions = none  // taxi
           zones do not overlap
        all t1, t2: TaxiZone | !(t1 = t2) => t1.queue
           & t2.queue = none // different zones have
           different queues
}

// other important constraints
fact {
        all q: TaxiQueue | q in TaxiZone.queue  //
           every taxi-queue must belong to a taxi-zone
        all t: TaxiZone | t in City.zones  // every
           taxizone must belong to a city
        all p: GeographicalPosition | p in TaxiZone.
           positions  // every point in the city must
           belong to a taxi-zone (in other words: the
           city is fully covered by taxi-zones)
        // the following constraint is incorrect (
           commented away), since a taxi servicing a
           passenger resides in a zone, without
           appearing in the queue
        //all t: Taxi, z: TaxiZone | t.location in z.
           positions => t in z.queue.taxi  // the dual
            of what specified in TaxiZone's appended
           fact: if a taxi belongs to a zone's queue,
           then its location must be in the zone as
           well
        all q: TaxiQueue | q in TaxiZone.queue  //
           every taxiqueue must be attached to a
           taxizone
        all t: Taxi | t in City.taxis
        #City = 1  // the scope of our project
```

```
}

// arguments:   t: target taxi, dest: its destination,
    fromZone, fromZone': pre and post state of the
   taxi's initial zone
// toZone, toZone': pre and post state of the taxi's
   final zone
pred taxiMove[t: Taxi, start, dest:
   GeographicalPosition, fromZone, toZone, fromZone',
   toZone': TaxiZone] {
        start in fromZone.positions and start in
           fromZone'.positions
        dest in toZone.positions and dest in toZone'.
           positions
        fromZone'.queue.taxi = fromZone.queue.taxi - t
        toZone'.queue.taxi = toZone.queue.taxi + t
}

// helper predicates, used to describe the state of a
   taxi
pred taxiIsUnavailable[t: Taxi, z: TaxiZone] {
        t.location in z.positions
        t not in z.queue.taxi
}
pred taxiIsAvailable[t: Taxi, z: TaxiZone] {
        t.location in z.positions
        t in z.queue.taxi
}

// arguments: t: the target taxi, c: city in the pre-
   state, c': city in the post state
// we need a pre/post state city (and not just a pre/
   post state taxizone)
// because of vincula imposed on the taxi zones by
   city's append-fact
pred taxiBecomeUnavailable[t: Taxi, c, c': City] {  //
    TODO? unsure
        t in c.taxis and t in c'.taxis  // t always
           belongs to the city
        c.taxis = c'.taxis  // the taxi set is
           unchanged
        all z: c.zones | t not in z.queue.taxi => z in
            c'.zones  // all zones the same except the
            one with the changed taxi
```

```
        // the availability of t in the pre-state
           implies the unavailability of t in the post
           -state
        taxiIsAvailable[t, c.zones] =>
           taxiIsUnavailable[t, c'.zones]
        c.positions = c'.positions  // the pre/post
           state are physycally the same city
}

// sister function of the above
pred taxiBecomeAvailable[t: Taxi, c, c': City] {
        t in c.taxis and t in c'.taxis  // t always
           belongs to the city
        c.taxis = c'.taxis  // the taxi set is
           unchanged
        all z: c.zones | t not in z.queue.taxi => z in
            c'.zones  // all zones the same except the
            one with the changed taxi
        // the availability of t in the pre-state
           implies the unavailability of t in the post
           -state
        taxiIsUnavailable[t, c.zones] =>
           taxiIsAvailable[t, c'.zones] // changed
           from the other function
        c.positions = c'.positions  // the pre/post
           state are physycally the same city
}

pred show {
        #TaxiZone >= 3 // just to make things
           interesting
}

run taxiMove for 1 City, 1 Taxi, 2
   GeographicalPosition, 5 TaxiZone, 5 TaxiQueue
run taxiIsUnavailable for 1 City, 4 Taxi, 4
   GeographicalPosition, 4 TaxiZone, 4 TaxiQueue
run taxiIsAvailable for 1 City, 4 Taxi, 4
   GeographicalPosition, 4 TaxiZone, 4 TaxiQueue
run taxiBecomeUnavailable for 1 City, 4 Taxi, 4
   GeographicalPosition, 4 TaxiZone, 4 TaxiQueue
run taxiBecomeAvailable for 1 City, 4 Taxi, 4
   GeographicalPosition, 4 TaxiZone, 4 TaxiQueue
run show for 15
```

6.2 Generated world

Diagram 6.19 shows one of the instances generate by alloy, running the *show* predicate

6.3 Solver output

```
6 commands were executed. The results are:
  #1: .taxiMove is consistent.
  #2: .taxiIsUnavailable is consistent.
  #3: .taxiIsAvailable is consistent.
  #4: .taxiBecomeUnavailable is consistent.
  #5: .taxiBecomeAvailable is consistent.
  #6: .show is consistent.
```

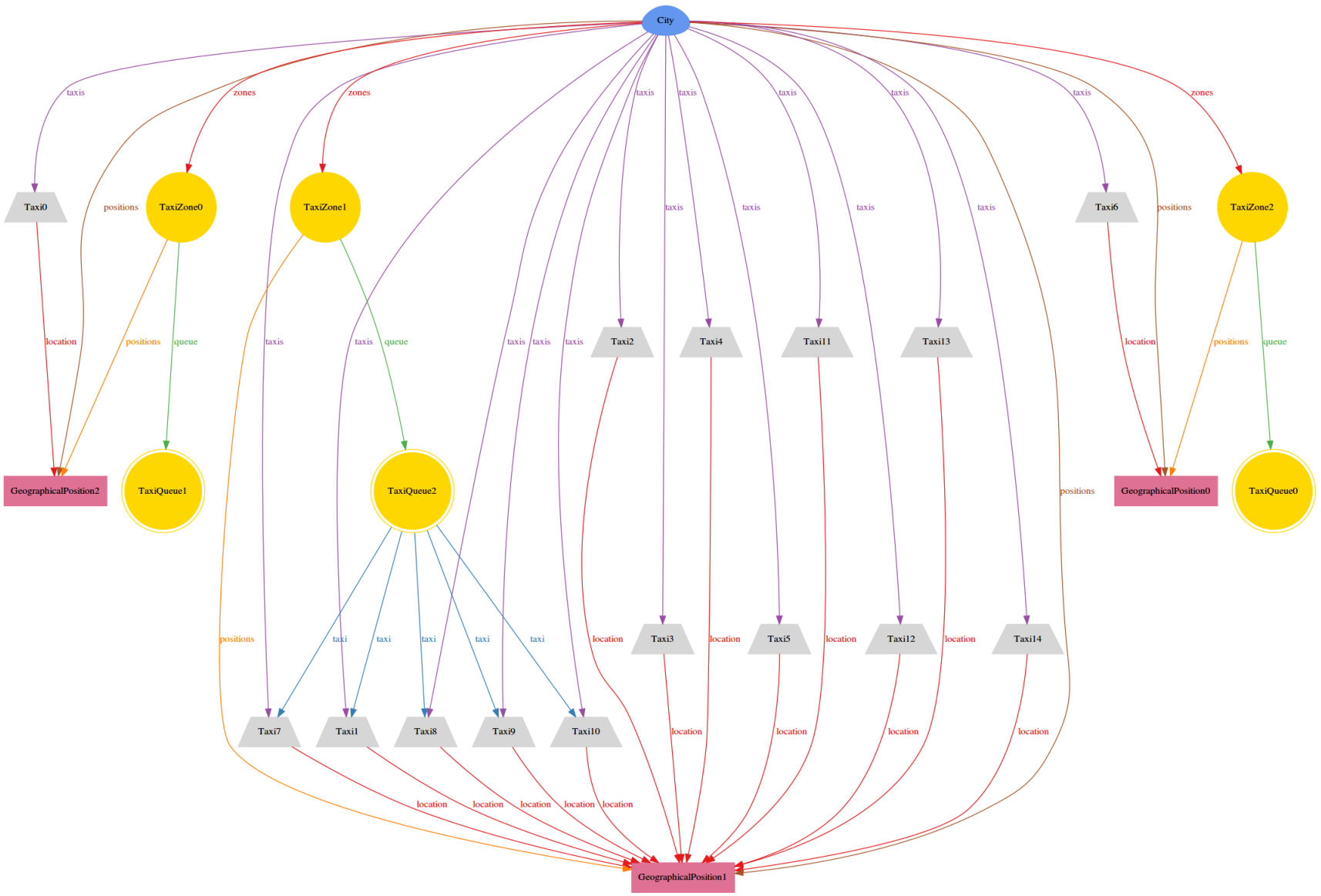

City
Taxi0
TaxiZone0
TaxiZone1
Taxi6
TaxiZone2
Taxi2
Taxi4
Taxi11
Taxi13
GeographicalPosition2
TaxiQueue1
TaxiQueue2
GeographicalPosition0
TaxiQueue0
Taxi3
Taxi5
Taxi12
Taxi14
Taxi7
Taxi1
Taxi8
Taxi9
Taxi10
GeographicalPosition1
taxis
zones
positions
queue
location
taxi

Figure 6.19:

Working hours

To complete this document we spent more or less 33 hours per person