POLITECNICO
MILANO 1863

RASD - myTaxiService

Ciceri, Filippo
filippo.ciceri@mail.polimi.it

Cesaro, Federico
federico.cesaroi@mail.polimi.it

Capecchi, Luca
luca.capecchi@mail.polimi.it

November 30, 2015

Contents

1 Introduction

1.1 Purpose

This RASD is a specification document intended to outline the measures being taken to implement "myTaxiService", a service which will be used by taxis in the city of X in order to facilitate taxi usage for drivers and customers. These specifications are for the government of X, the future provider of this service, and the developers that will create the applications to support the service.

1.2 Scope

1.2.1 Software

This service will require the development of the following:

Client-Side

- A mobile application for customers
- A web interface for customers to access the service
- A mobile application for taxi drivers to interact with calls.

Server-Side

- A server application that handles live queues
- A database that stores user data, including credentials
- Another database that handles Registered user rides, memorizing both booked rides and past itineraries

The service will allow customers to instantly call a taxi, without requiring authentication. In case of advance booking, users need to login first, and can then access authenticated user functions. These include for example being able to book a taxi for a future date.
When booking taxis, a registered user may enable ride-sharing. Ride-sharing queues up a customer's ride into the system, which will handle merging multiple requests into a single unified ride, which shall only marginally alter each single request by a small amount, determined by some predefined parameters.

Authenticated users can also modify their already booked rides, as well as cancel them within a time frame of 2 hours from the start of their ride, and also modify their personal and payment information.
Taxi drivers will have an interface which allows them to communicate when they start and stop working(being available), and the menu to accept or refuse a job. Drivers will also have an option to remove themselves from the queue, for the occasions when they pick up passengers from the street. The system is responsible for sending the appropriate notifications to the customer when their ride has been accepted, along with information on the taxi coming to pick them up, as well as provide an estimate of the time remaining before the start of their ride.

1.2.2 Applications

This service is intended to simplify the process of acquiring a taxi in X. It should remove the need for a human operator

1.2.3 Goals

The service should allow customers to do the following things:

1. Allow a customer to get a taxi instantly, much like dialing the taxi number in any city
2. Allow a customer, after registration, to book a taxi for a specific date and time
3. Allow registered users to automatically share rides with each other, through an automatic queueing system
4. Allow user to view their reservations and cancel them
5. Allow taxi drivers to accept rides both through the application and externally in a seamless manner
6. Assign taxi drivers to customers automatically

1.3 Definitions

Guest A guest to the service, namely someone who may be a driver or a registered user but has not been authenticated.

Registered User/User A user of the service who has gone through the registration process, whose information is stored within the service's database.

Driver A user of the service that provides their taxi for the service. They are registered in a manner different from that of other users.

Customer Any person trying to use a taxi through the application, both a registered users and guests.

Ride Generic term to talk about a taxi trip

Instant Ride A ride ordered directly through the application which instantly assigns a taxi to a customer.

Reservation/Booked Ride A ride booked in advance through the application.

Ride Locking Term used to indicate reservations that have been "locked" from cancellation. A reservation is considered locked when it can no longer be cancelled or modified (2 hours before the start of the trip).

1.4 Overview

This document is composed by the following sections:

1. The introduction, which contains basic information on how the service works
2. Overall description, which describes the service's structure, how the service operates, what each type of user can do and the way the back-end infrastructure operates.
3. Requirements, a section that analyses what is going to be required to make the actual application work.
4. UML, where UML diagrams show the way user functions operate, as well as how the app shall be structured.
5. Alloy, where Alloy Analyzer is used to specify properties and limitations of the application.

2 Overall Descriptions

2.1 Product Perspective

This section looks at all the necessary interfaces needed for the service to work properly.

2.1.1 User Interfaces

A home page, where you can instantly book a taxi, login and sign up. Upon logging in, according to the user type, you can access user-specific functionalities, listed in section 3.

2.1.2 Hardware Interfaces

A device with internet connectivity is required, as well as some input measure to insert user data and other information. A gps is optional but not mandatory, since it can communicate a user's position in order to simplify booking. This can be aided with the use of other localization services such as geolocalization via cell line and wifi.

2.1.3 Software Interfaces

A soft keyboard can be used instead of a hardware one for input purposes. A web browser is required if the application is not available/installed on the device trying to make use of the service.

2.1.4 Communication Interfaces

An internet connection is required for all types of users of the application.

2.1.5 Memory Interfaces

150MB of storage space for the application + cache when using the app.

2.2 Product Functions

2.2.1 User Access and Registration

Customers have the option of registering with the service. This must be done with email and password. Password recovery functions are available. This enables the reservation function. Unregistered users may still make use of the instant call function.
Being a registered user gives access to a personal page. Other than making reservations, user and payment data may be edited here. There should also be an option that lets a user see a list of past reservations.
Taxi drivers on the other hand cannot register through the site. Their information is inserted manually into the system by an administrator and a special set of credentials is sent to them.

2.2.2 Instant Call

This is the most basic function of the application, and is available to all customers. The user supplies their position immediately and the order is sent to all nearby taxi drivers.

2.2.3 Reservation

This function is only available for registered users. The user inputs their desired date and time of departure, the location, their destination and the number of passengers. Reservations have to be made, and eventually modified or cancelled, at least 2 hours before the start of the ride. However, even a reserved ride is treated like all other calls, and therefore is assigned to a taxi driver only 10 minutes before the start of the trip.

2.2.4 Shared Taxi Function

Ride sharing is an advanced option of the regular reservation function. When a user enables sharing, they are inserted into a queue for the shared ride function. As soon as one request in the queue can be locked, the system scans the queue and creates an itinerary inserting as many possible shared requests into a single order. All participants in the order will be notified of the full itinerary via email, and their approximate pick-up time. When this request is sent to taxi drivers, they receive the full itinerary including all stops they have to make.
Note that shared rides can be split into dropping off and picking up different passengers during the same ride, as long as the fee each customer pays is proportional to the length of the trip and the time travelled.

2.2.5 Taxi Driver Status

Due to the way taxis work, a taxi driver is free to set their own schedule. Therefore, the application allows them to set their working status, that is to toggle it between working and not working. When a driver is working, they are inserted into the worker queue, and when they reach the top of the queue they receive the available offers.
When a driver accepts a ride, they are removed from the queue. Once they complete the ride, they notify the system by clicking the "ride complete" button, which automatically reinserts them into the queue. If a rider refuses a ride, they are moved to the end of the queue.
Taxi drivers can also temporarily remove themselves from the queue when they pick up passengers the traditional way. When this happens, they use the "remove from queue" button. This places them into a "ghost" ride within the application and removes them from the queue. Pressing the "ride complete" button reinserts them into the queue.

2.2.6 Taxi Notification System

Notifications for rides are sent to both the taxi driver and the customer. The mode in which a taxi driver interacts with notifications is described above.
Customers on the other hand receive a notification with an estimated arrival time when one of their orders is accepted.

2.2.7 Queue System

The taxi queue is handled automatically by the system. Every time a driver sets their status to working or every time they end a ride, they are automatically added to the end of the queue. Setting the working status to not working or using the remove from queue function removes a driver from the queue. The system uses a single queue for all taxis. The queue is built using a FIFO model, but is filtered by proximity. Every time a customer is found, the system looks for the closest drivers, in increasing distance intervals. As soon as a driver is found, the offer is proposed to them. If a driver refuses, he is moved to the end of the queue and the ride is proposed to the next driver in the queue. If the driver accepts, he is removed from the queue.

2.3 Constraints

2.3.1 Regulatory policies

Since the application is being commissioned by the government of X, there should be no issues concerning regulatory policies, since this application doesn't introduce any competitors, but only acts as an aid to an existing system.

2.3.2 Hardware limitations

The only hardware limitations regard the availability of an internet connection, since the service cannot operate without one. Therefore, devices with no connectivity can't make use of the application.

2.3.3 Interfaces to other applications

myTaxiService doesnt have to interfaces with any other applications.

2.3.4 Parallel operation

The service shall be able to function independent of how many users may be using the application at any time.

2.3.5 Safety and Security Considerations

Due to privacy issues, the identity of customers shall be protected at all stages. Therefore, the identity of customers going on shared rides shall not be known other customers ahead of time. Also, when accessing logs via the web interface, the database shall never return information about customers different from the one requesting it.

2.4 Assumptions

Creating our project we had to assume some facts to clarify some unspecified situations. We assume that:

- If a customer calls a taxi, he will be there when it arrives.
- If a Taxi Driver accepts or refuses a ride, he won't change his mind.
- If a Taxi Driver accepts a ride, he won't miss the appointment.
- A Taxi Driver may pick up passengers who are not using the app. Doing so temporarily removes them from the queue.
- Each Taxi has a unique code assigned to it.
- Each customer will create at most one account.
- Taxi drivers take no longer than 10 minutes to reach the customers.
- All the taxis in the city have the same number of seats. A customer inserts the number of seats he needs, then the system calculates the required number of taxis.
- In the city there are enough taxis to satisfy every call. There won't be any customer that will have to wait for his taxi more than the established time.

2.5 Scenarios

Scenario 1: User Registration and Login

$USER is using the service for the first time. Wanting to book a taxi, $USER must register first. After clicking the register button, and inserting key information, such as email and password, as well as some basic demographics data, $USER is prompted to select a preferred payment method. After selecting credit card payment, $USER saves credit card information on the site and is now ready to make a reservation.

Scenario 2: Unregistered User Basic Call

$USER has recently installed the application, but has not been registered with the service yet. Needing a taxi immediately, $USER clicks the "instant call" button and inputs his current location. As soon as a driver has accepted the call, $USER sees an estimate of the time remaining until the taxi reaches him. Once the taxi reaches him, $USER's trip starts.

Scenario 3: Basic Reservation

$USER is a registered customer of the service. $USER logs into the service's website, and selects the "reservation" option. $USER inserts the date and time when he wishes to be picked up. The day of the reservation comes, and 2 hours before the specified time, $USER 's order is locked, and a notification is sent to $USER. 10 minutes before the start of $USER 's trip, his reservation is inserted into the queue. As soon as the order is accepted, $USER receives another notification, with information concerning when the taxi driver will reach him, as well as the taxi's code. $USER is picked up by the taxi and the taxi takes $USER to his destination. Payment can be made either via the methods offered by the service or in cash.

Scenario 4: Sharing Reservation

$USER1 wants to make a reservation for a taxi on monday around lunchtime, to go to the airport with a work colleague. $USER1 and his colleague don't mind sharing the taxi with someone else if the fee is reduced by sharing. Being a registered user, $USER1 reserves a taxi for 2 from his workplace to the airport, enabling the sharing option. and setting the time to 12:30, aiming to reach the airport by 13:15.
$USER2 also has to go to the airport on monday around lunchtime. $USER2 lives only a few minutes from the airport, but doesn't want to go by car and pay for expensive parking. $USER2 thinks taking the taxi alone is too expensive, and therefore tries to make a reservation with sharing. $USER2 sets his pick-up time at 1:00 pm.
Monday comes around, and at 10:30, the system locks $USER1's reservation. Seeing that $USER2 has a compatible request, the system automatically locks $USER2's order as well. Both users are notified of their full itinerary. 12:20 rolls around, and the order is automatically queued into the system. A taxi driver accepts the order, knowing the trip he needs to make all the way to the airport including the pick-up at 13:00, and picks up $USER1 and his colleague from their workplace. The taxi then reaches $USER2's home at 13:00 and picks him up as well. Once the three passengers are delivered to the airport, fees are split accordingly; $USER1 pays the full trip from the workplace to $USER2's home and 2/3 of the trip from $USER2's home to the airport, while $USER2 pays for the remainder.

Scenario 5: Taxi Driver Accepting an Offer

$DRIVER1 logs into the application and selects the "start working" option. Immediately, he is inserted into the queue. After a few minutes, $DRIVER1 receives the notification of an available ride nearby. The customer is only 2 minutes from where $DRIVER1 currently is, and therefore he accepts the offer. $DRIVER1 picks up the customer and takes him to his destination. After being paid, $DRIVER1 clicks the "end of ride" button and is reinserted into the queue.

Scenario 6: Taxi Driver in Queue Picking up a Regular Passenger
$DRIVER1 is in the application queue waiting on an offer. As he drives around, he encounters a pedestrian requesting a taxi. $DRIVER1 selects the "remove from queue" option in the application, and therefore leaves the queue. After picking up the passenger, he takes the passenger to his destination. Once the passenger reaches the destination and pays for the ride, $DRIVER1 click the "end of ride" button and is reinserted at the end of the queue.

Scenario 7: Taxi Driver Refusal and Acceptance of a Shared Ride offer
$DRIVER1 and $DRIVER2 are the first two in the queue for the city center. $DRIVER1 receives an offer for a shared ride like that in scenario 4. $DRIVER1 however doesn't want to travel all the way to the airport, and since the queue in the city center is never very long, refuses the offer, and gets placed at the end of the queue.
Moments later $DRIVER2 receives a notification for that same offer. $DRIVER2 accepts the offer and picks up the first two passengers. $DRIVER2 then driver to the pick-up location for the second customer and picks him up as well. After arriving at the airport, the application, basing itself on travel time and distance tells the taxi driver how much each customer has to pay. After payment, the taxi driver selects the "end of ride" function and is reinserted into the queue.

3 Requirements

3.1 Unregisterd User Requiremets

3.1.1 System Registration

The system needs to enable sign up for new users. This will unlock to the user, after a login, the access to advance features.

Figure 1: Those mockup show the registration form via web and mobile app.

3.1.2 User Login

The system must provide an authentication system that enable the access to advanced features.

3.1.3 Instant Calls

The system must let unregistered user call a taxi immediately.

When an instant call is made the system will show a popup with a captcha test in order to complete the request.

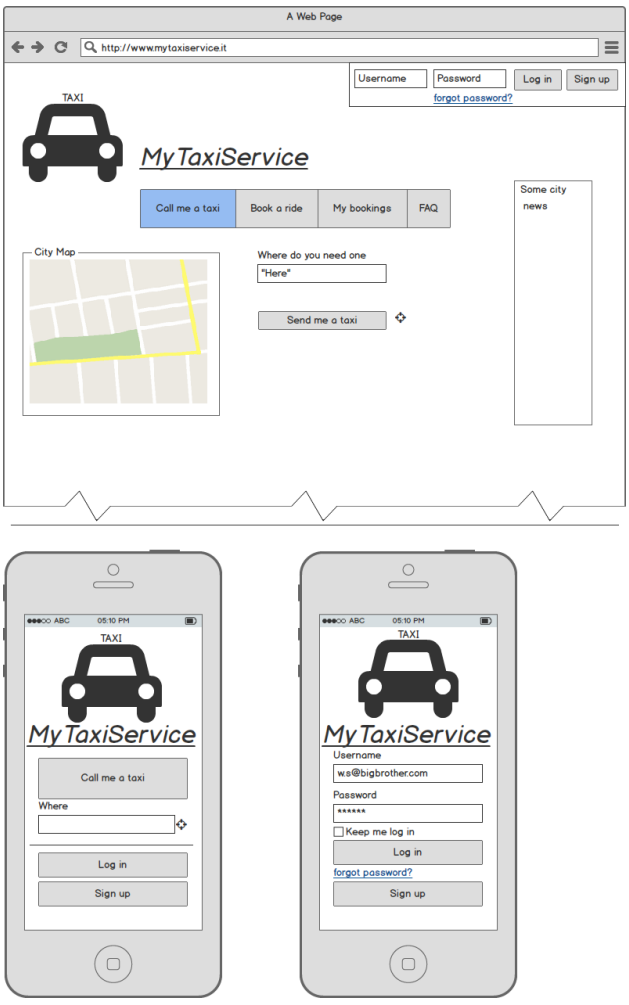


Figure 2: The home page give the acces to both login and instant call.

3.2 Registered User Requirements

3.2.1 User edit information

The system must have a profile editing function, where a user can change personal data, such as their address, password, email and also add new payment options. On top of that, the system shall log all the rides a user makes into a database.

Figure 3: Edit form.

3.2.2 Instant Calls

The system must let registered user call a taxi immediately and give them some advance options.

When an instant call is made the system will show a popup with a captcha test in order to complete the request.

Figure 4: Home page of a logged user.

3.2.3 Booking and Sharing

The system must let registered users book a ride for a specific date and time. On top of that, there has to be an option to share rides with other people who are booking similar routes. Notifications must be sent when a match is found for shared rides.

Figure 5: Web and app pages used for book a ride.

3.2.4 Booking History

The system must let the user to see their booking history and let the acces to the editing of a future ride.

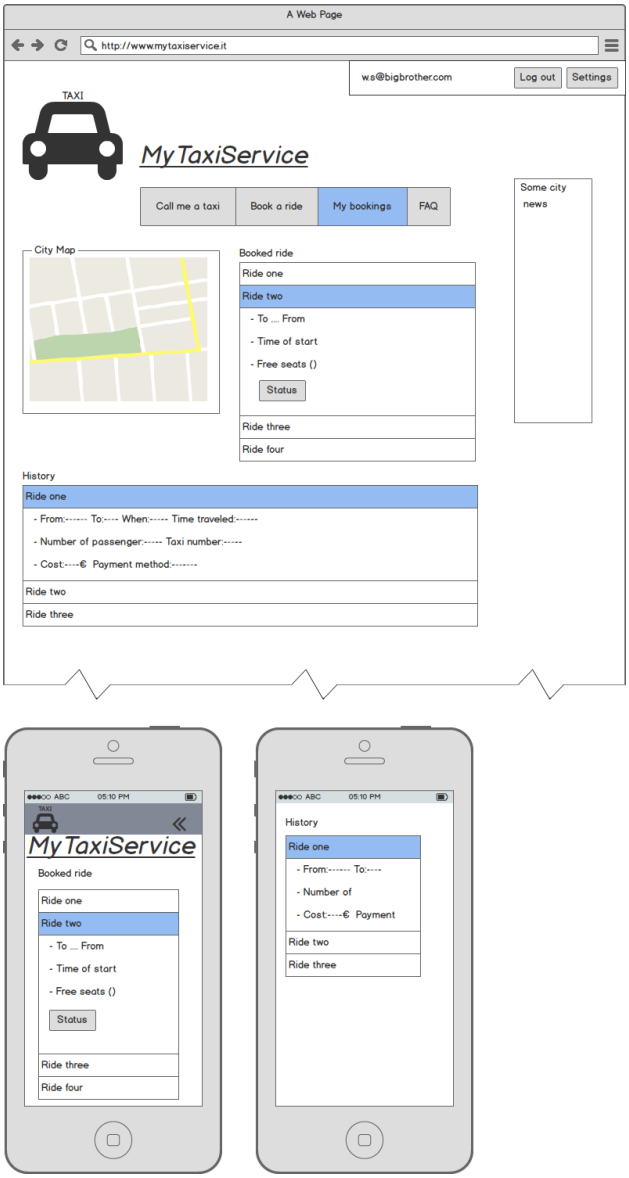


Figure 6: Booked ride history.

3.2.5 Booking Editing

The system must let the user to modify or cancel a booked ride (if the ride has not been locked).

Figure 7: Ride editing form.

3.3 Taxi Driver Requirements

The worker queue must be handled automatically by the system. This includes building a queue dynamically using a FIFO model.

3.3.1 Driver Login

The system must provide a driver's authentication system that enable the driver to acces his feature.

Figure 8: Driver's log in interface.

3.3.2 Driver Work Settings

The system must provide the driver the acces to his working condition.

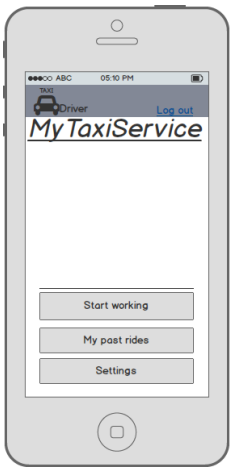


Figure 9: Driver's home page.

3.3.3 Driver Ride Acceptance

The system must provide the driver to accept or refuse a ride.

Figure 10: Driver's ride information page.

3.3.4 Driver Ride Settings

The system must provide the driver information on the ride and a command to end it, and a function that automatically reinserts into the queue.

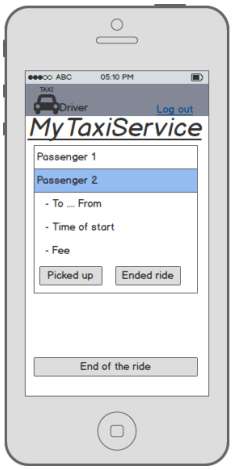


Figure 11: Driver's ride information page.

3.4 Functional Requirements

After having stated the effective requirements for the system, we can start listing what a user can do.

Unregistered user/Guest

- Sign up
- Log in
- Instant call

Registered User

- Instant call
- Change user data, including payment options
- View ride history
- Advance booking(with sharing options)
- Booking modifications
- View booking status(including sharing status)
- Logout

Taxi Driver

- Set work status
- Start a ride
- Accept or refuse a ride
- Notify end of ride
- Logout

3.5 Logical Database Requirements

A database should be put in place server-side to handle credential storage, ride booking and ride history. The database responsible for ride history shall include where the each ride started and ended, the taxi carrying the passengers and the corresponding driver, and the user/s that made use of the taxi(multiple users, up to 4, in case of a shared ride). Note that rides made by unregistered users should also be logged, so drivers may view their own history.

3.6 Non Functional Requirements

3.6.1 Performance Requirements

The system should be fast in order to garantee a good service to the final user. We assume that our system response to the users need is close to zero.

3.6.2 Design Constraint

The main system will be developed using Java EE, we also have to develope two mobile applications in Objective-C and Java.

3.6.3 System Avaibility

The service will be accesible online both via web and mobile-app. To achieve this we will use a Cloud Virtual Server provider. This will guarantee more scalability for the system and can also reduce the cost that we would have with dedicated server. On top of that we can maintain high level performances even in busy days

3.7 Security

My Taxi Service implements a login authentication in order to protect user's sensible informations. Password must be 8 character long and must contain both letters and numbers. Password will be encrypted and stored in the database with all user's data.

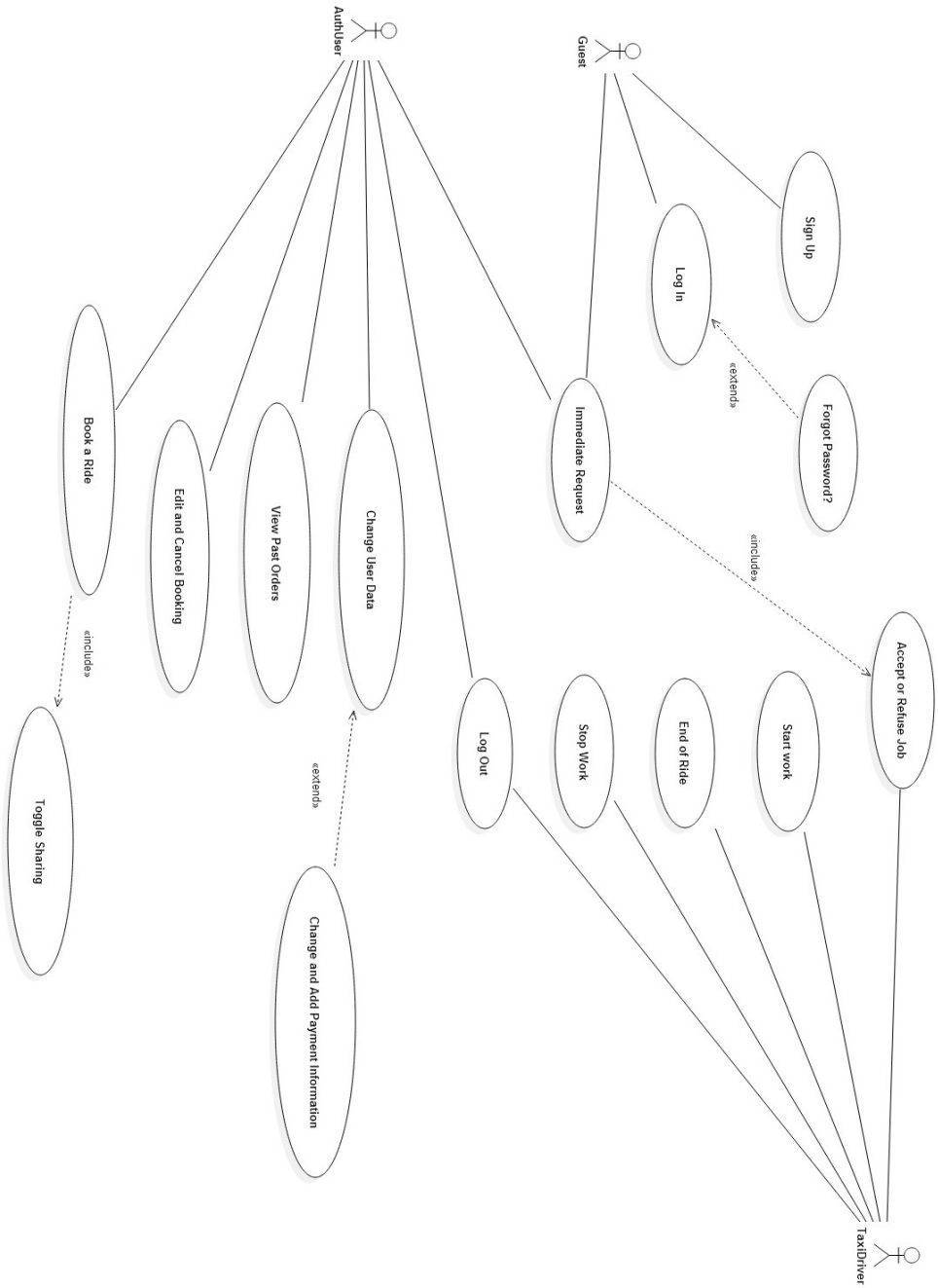
We guarantee that the user can't acces to other user sensible information.

At sign up the user will recive an email with a confirmation link to enable the acces to the advanced feature of our system.

We also implement a capcha control when you make an instant call, in order to prevent missclick or botnet attack.

4 UML Models

4.1 Use Case Diagram



4.2 Use Cases

These are the use cases which are part of the service

- Sign up
- Log in
- Edit user data
- Instant call
- Booking ride
- Edit booking
- Cancel booking
- View past rides
- Set working condition
- Notify end of a ride
- Accept or refuse a job
- Logout

4.3 Use Case Description

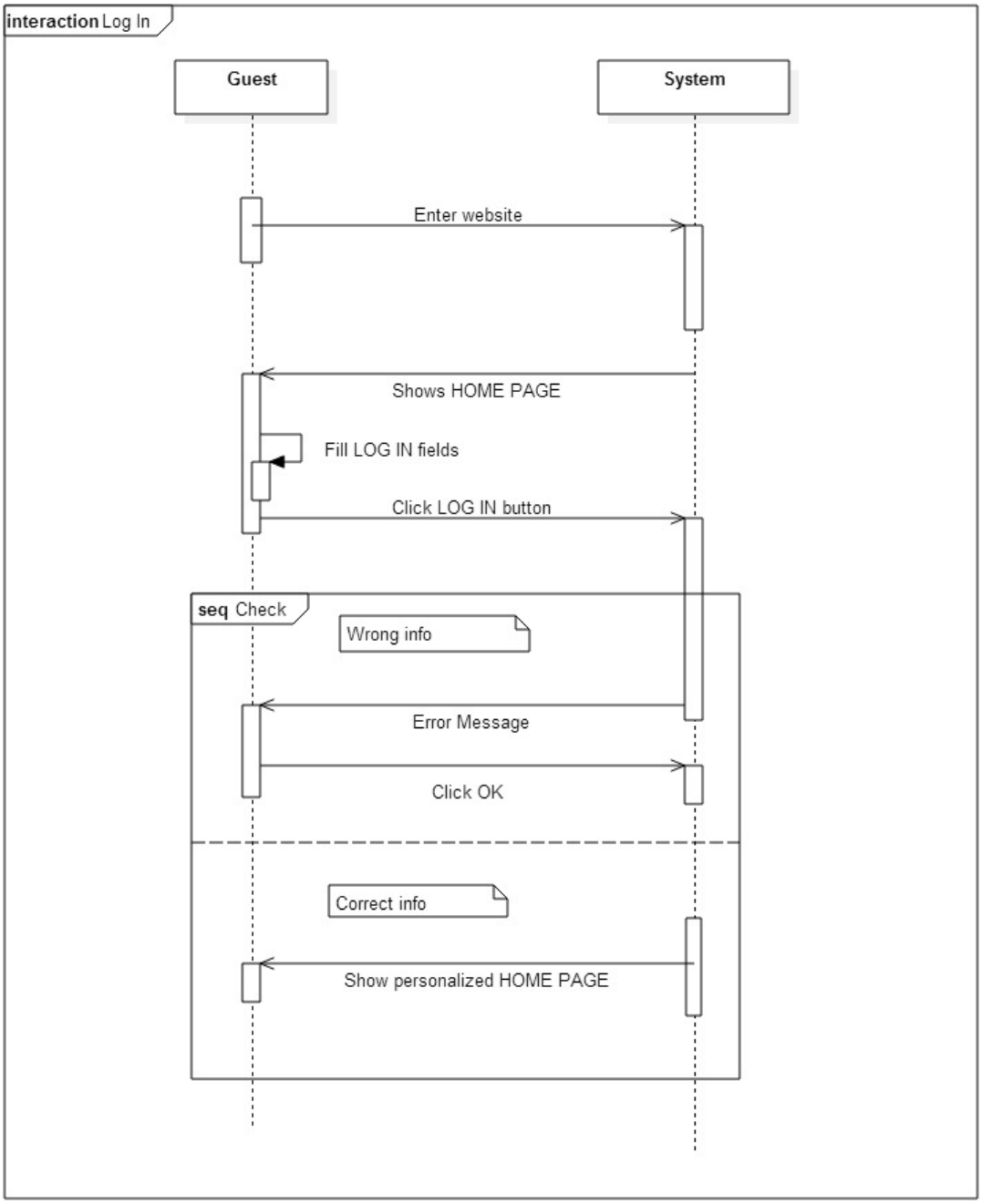
These are the detailed descriptions of the Use Cases listed and shown before

4.3.1 "Sign up" case:

Name	Sign up
Actor	Guest
Entry conditions	None
Flow of events	• The guest enters the website • The guest clicks the SIGN UP button • The guest fills the following fields: – Name – Surname – E-mail – Date of birth – Address – Password – Payament system (optional) • The Guest clicks on CONFIRM button
Exit conditions	The system shows the home page personalized with the user credential and updates its database.
Exceptions	The guest inserts the wrong email/password combination, the email is already registered in the system or some fields are empthy.

4.3.2 "Log in" case:

Name	Log in
Actor	Guest
Entry conditions	None
Flow of events	• The guest enters the website • The guest fills the text field requested (email and password) • The guest clicks on LOG IN button
Exit conditions	The system shows the home page personalized with the user credential
Exceptions	The guest inserts the wrong email/password combination or the email is not registered in the system

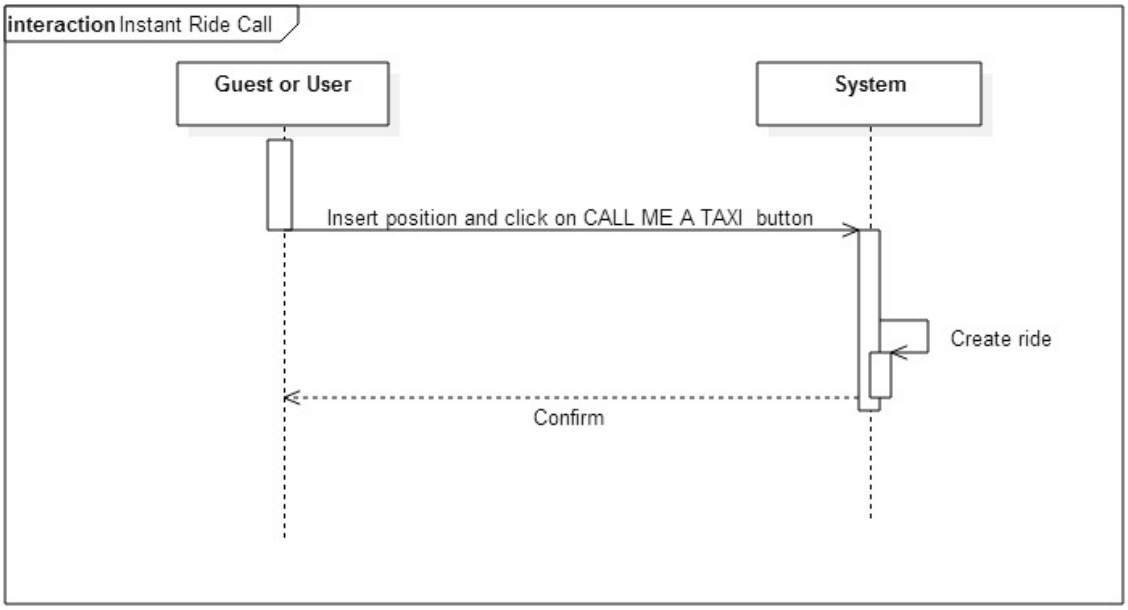

interaction Log In
Guest
System
Enter website
Shows HOME PAGE
Fill LOG IN fields
Click LOG IN button
seq Check
Wrong info
Error Message
Click OK
Correct info
Show personalized HOME PAGE

4.3.3 "Edit user's data" case:

Name	Edit user's data
Actor	User
Entry conditions	None
Flow of events	• The User clicks on SETTINGS button • The system shows the list of data that the User filled when he signed up • The User modifies the data • The User clicks on SAVE button • The system shows the modified data list • The User click on OK button
Exit conditions	The system shows the home page personalized with the user credential
Exceptions	The guest inserts wrong new data or doesn't fill all the required fields

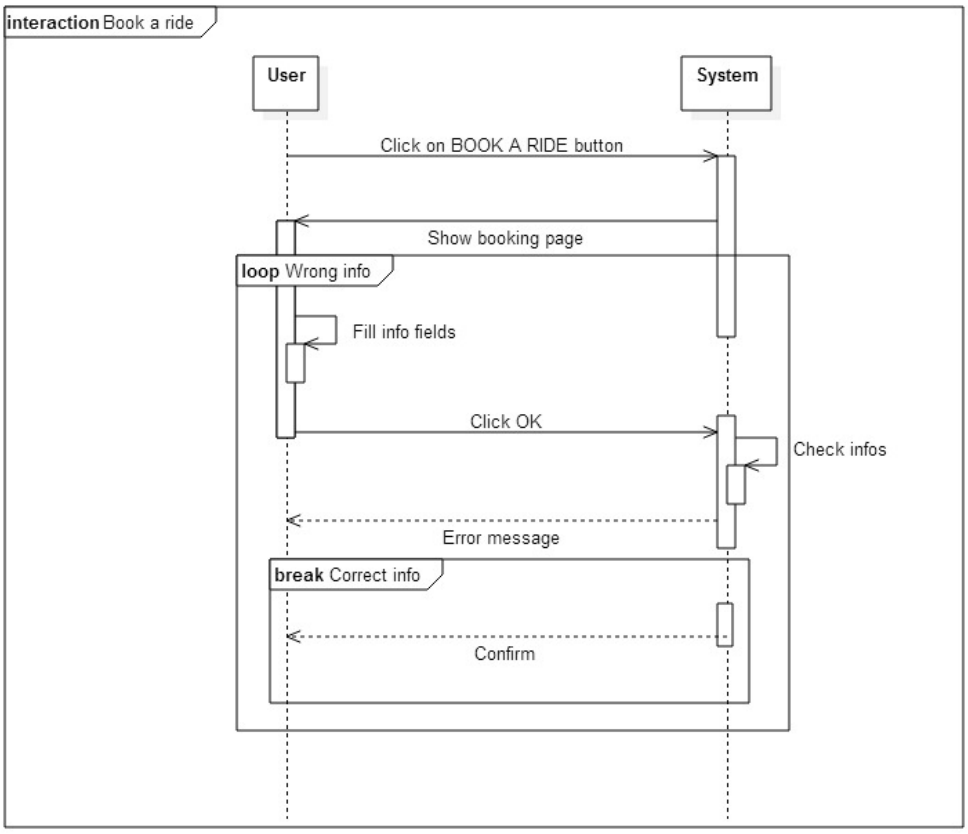
4.3.4 "Instant call" case:

Name	Instant ride
Actor	Guest and User
Entry conditions	None
Flow of events	• The User either clicks on CALL ME A TAXI button or HOME button • The system shows the page used to call a taxi • The User chooses if insert an address or use his own position (via GPS) • The User clicks on SEND ME A TAXI button • The User compile the captcha and confirm the call
Exit conditions	The system shows a confirmation message
Exceptions	None



4.3.5 "Book a ride" case:

Name	Book a ride
Actor	User
Entry conditions	None
Flow of events	• The User clicks on BOOK A RIDE button • The system shows a page with the field that the User has to fill: – Origin position (can be usd the GPS) – Destination – Number of people traveling – Departuring Time – An option to enable the Taxi Sharing • The User fills the fields and clicks BOOK IT button
Exit conditions	The system shows the confirmation page with the data inserted by the User and updares its booking list
Exceptions	The guest inserts wrong new data like an impossible hour


interaction Book a ride
User
System
Click on BOOK A RIDE button
Show booking page
loop Wrong info
Fill info fields
Click OK
Check infos
Error message
break Correct info
Confirm

4.3.6 "View status" case:

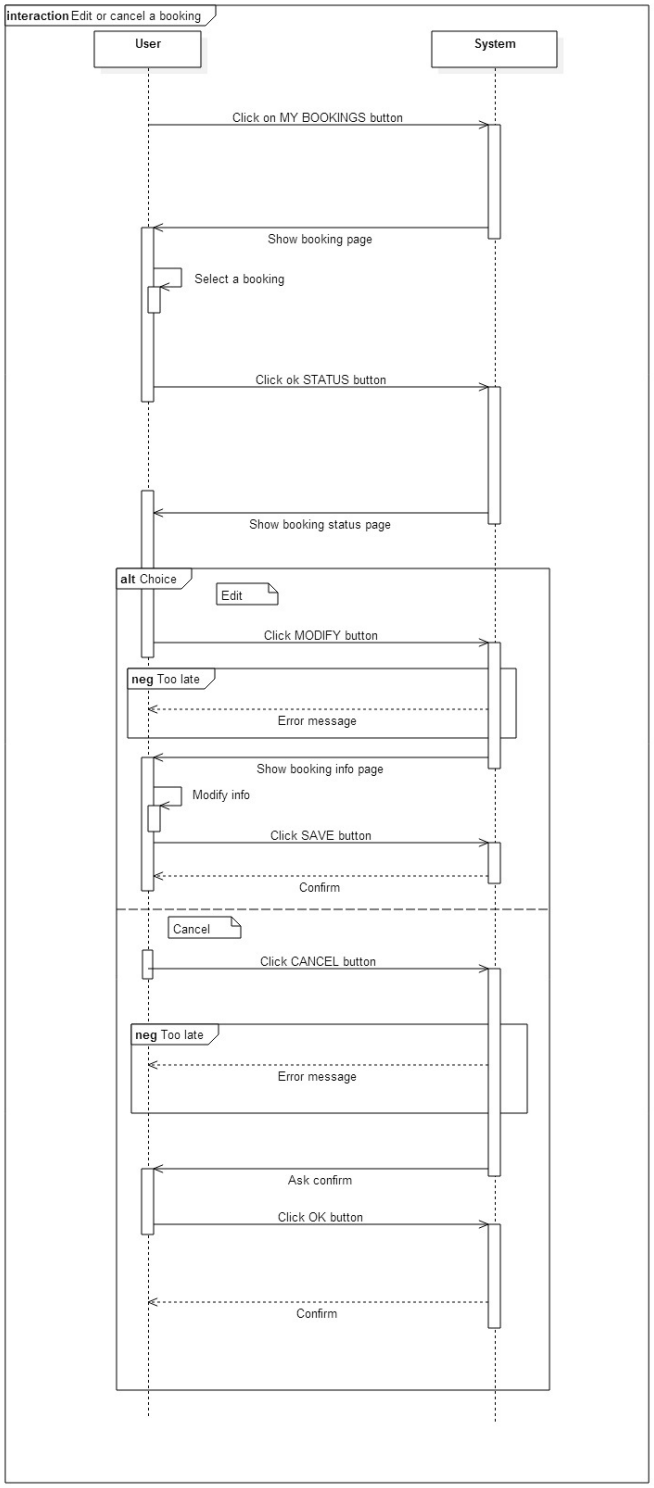
Name	View status
Actor	User
Entry conditions	The User must have booked at least one ride
Flow of events	• The User clicks on MY BOOKINGS button • The system shows a page with all the User's bookings • The User chooses a shared ride and clicks on STATUS button • The system shows a page with the sharing situation
Exit conditions	The system shows again the User's booking page
Exceptions	None

4.3.7 "Edit Booking" case:

Name	Edit Booking
Actor	User
Entry conditions	The User must have booked at least one ride
Flow of events	• The User clicks on MY BOOKINGS button • The system shows a page with all the User's bookings • The User chooses a booking and clicks on STATUS button • The system shows a page with the reservation's information • The User modifies the data and clicks on MODIFY button
Exit conditions	The system shows the confirmation page with the new data of the booking and updates its booking list
Exceptions	The User inserts wrong data like an hour to close to that moment The User tries to modify a ride less that ten minutes before its departuring time

4.3.8 "Cancel Booking" case:

Name	Cancel Booking
Actor	User
Entry conditions	The User must have booked at least one ride
Flow of events	• The User clicks on MY BOOKINGS button • The system shows a page with all the User's bookings • The User chooses a booking and clicks on STATUS button • The system shows a page with the reservation's information • The User clicks on CANCEL button • The System shows a page that ask the user to confirm the cancellation • The User clicks the OK button
Exit conditions	The system shows the confirmation page and updates its booking list
Exceptions	The User tries to cancel a booking that has a departure hour that is less than 10 minutes after that moment


interaction Edit or cancel a booking
User
System
Click on MY BOOKINGS button
Show booking page
Select a booking
Click ok STATUS button
Show booking status page
alt Choice
Edit
Click MODIFY button
neg Too late
Error message
Show booking info page
Modify info
Click SAVE button
Confirm
Cancel
Click CANCEL button
neg Too late
Error message
Ask confirm
Click OK button
Confirm

4.3.9 "View past rides" case:

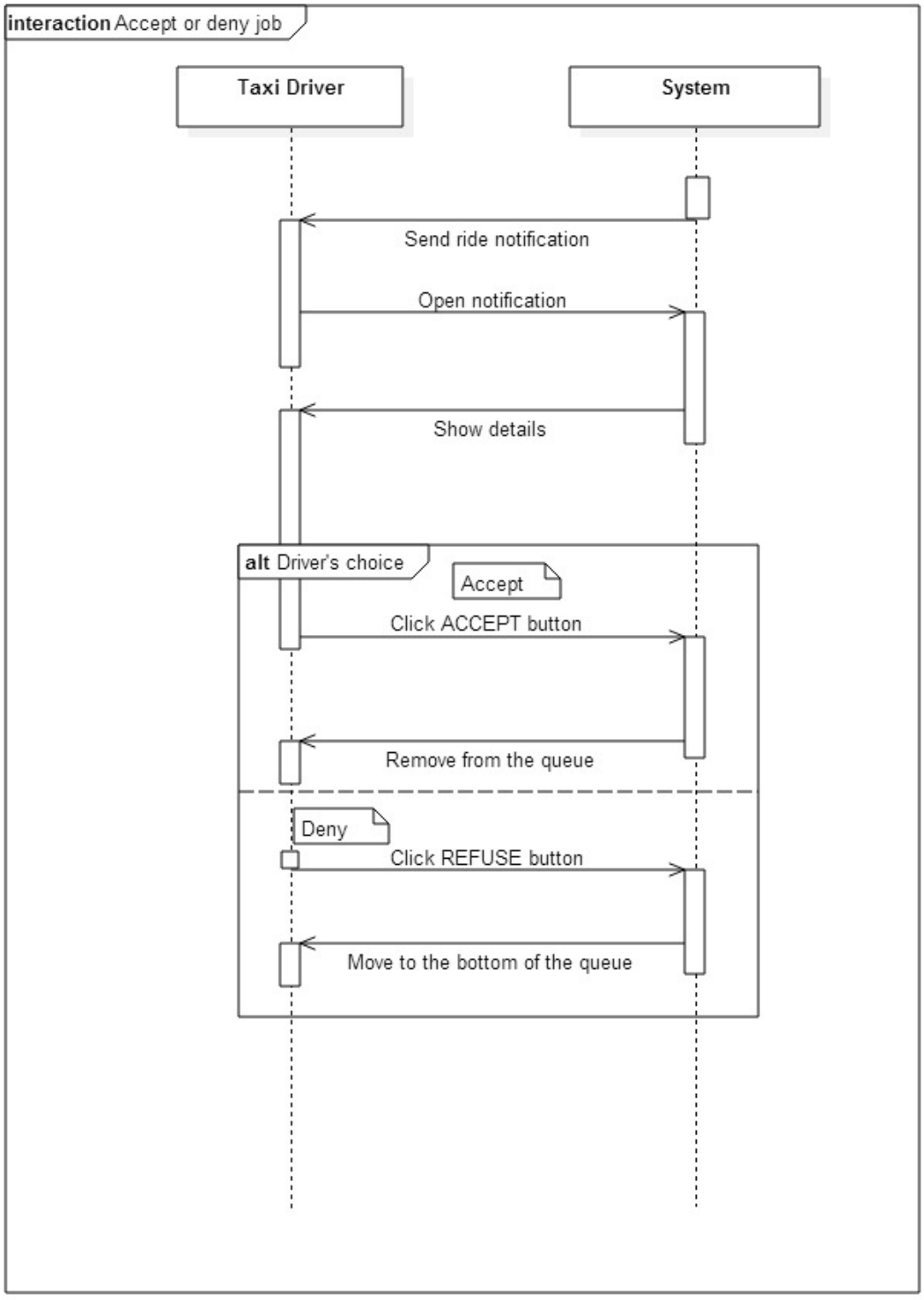
Name	View past rides
Actor	User
Entry conditions	The User must have ordered at least one ride
Flow of events	• The User clicks on MY BOOKINGS button • The system shows a page with all the User's bookings (if there are any) and the booking history • The User can see the details of a particular booking by selecting one
Exit conditions	The system shows again the User's booking page
Exceptions	None

4.3.10 "Set Working Condition" case:

Name	Set Working Condition
Actor	Taxi Driver
Entry conditions	None
Flow of events	• The Driver log in via app • The system shows a page with the current working state and, if the state is on "working", the amount of time the Driver has been working • The Driver clicks on START WORKING or STOP WORKING button to set his situation • The system shows the working state page updated
Exit conditions	The system updates the Driver's condition
Exceptions	None

4.3.11 "Accept or refuse a job" case:

Name	Accept or refuse a job
Actor	Taxi Driver
Entry conditions	The Driver must have been notified of the presence of a possibile ride
Flow of events	• The Driver opens the notification • The system shows a page the data of the ride • The Driver clicks on ACCEPT or REFUSE button • The system shows a confirmation page
Exit conditions	The system either notifies the asker and removes the Driver from the queue if he accepted, or moves the Driver to the bottom of the queue if he refused
Exceptions	None


interaction Accept or deny job
Taxi Driver
System
Send ride notification
Open notification
Show details
alt Driver's choice
Accept
Click ACCEPT button
Remove from the queue
Deny
Click REFUSE button
Move to the bottom of the queue

4.3.12 "Notify the end of a ride" case:

Name	Notify the end of a ride
Actor	Taxi Driver
Entry conditions	The Driver must have already accepted a ride
Flow of events	• The Driver clicks on the RIDE COMPLETED button in his homepage • The system shows a page to ask for confirmation • The Driver clicks on the OK or CANCEL button to confirm or not • The system shows the homepage again
Exit conditions	The system add the Driver to the queue
Exceptions	None

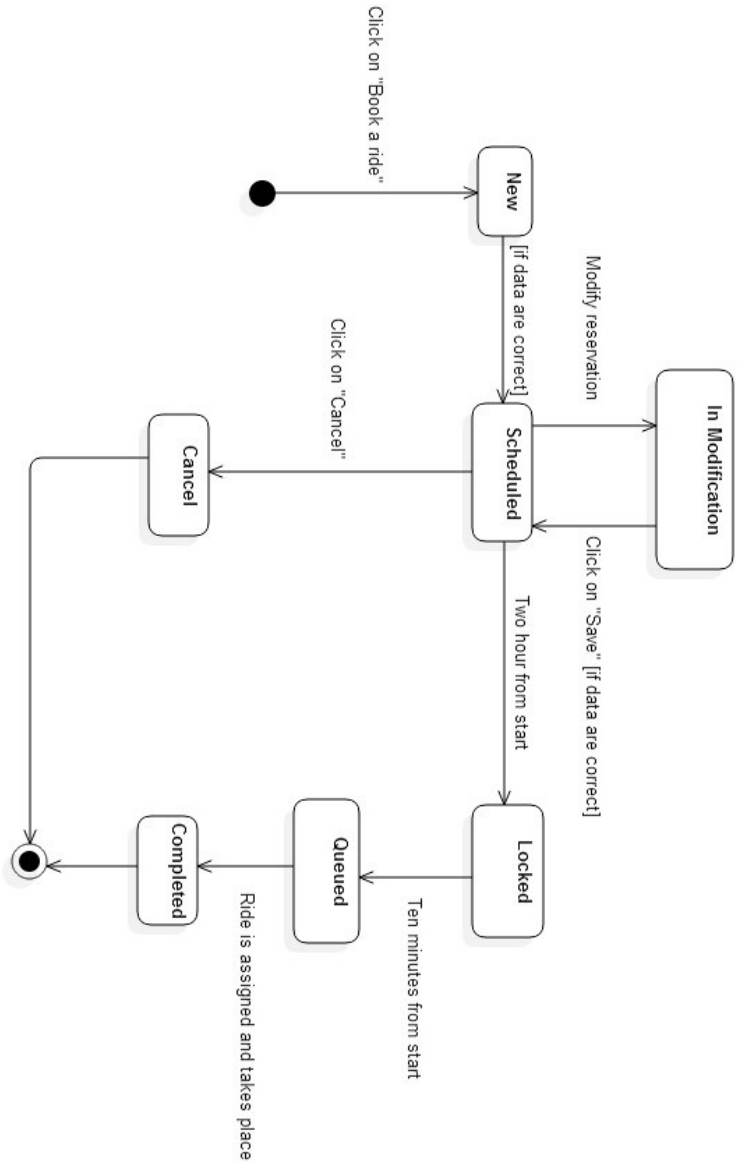
4.3.13 "Log Out" case:

Name	Log Out
Actor	User and Taxi Driver
Entry conditions	None
Flow of events	• The user clicks on LOG OUT button • The system shows the Guest's homepage
Exit conditions	None
Exceptions	None

4.4 State Charts

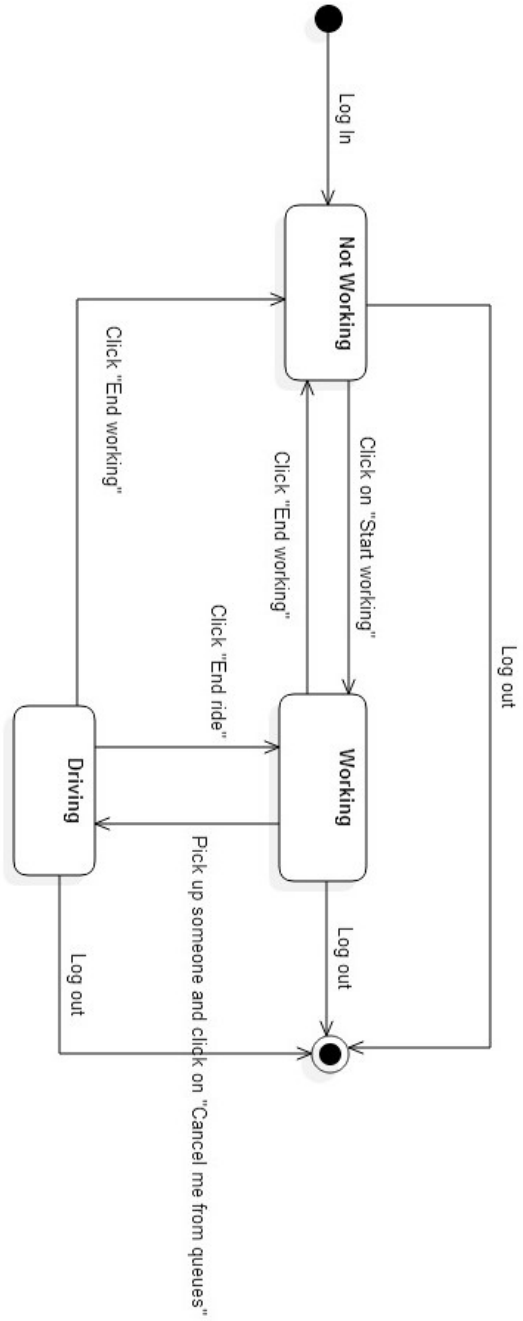
4.4.1 Reservation

This statechart describes the status of a reservation from creation to completion

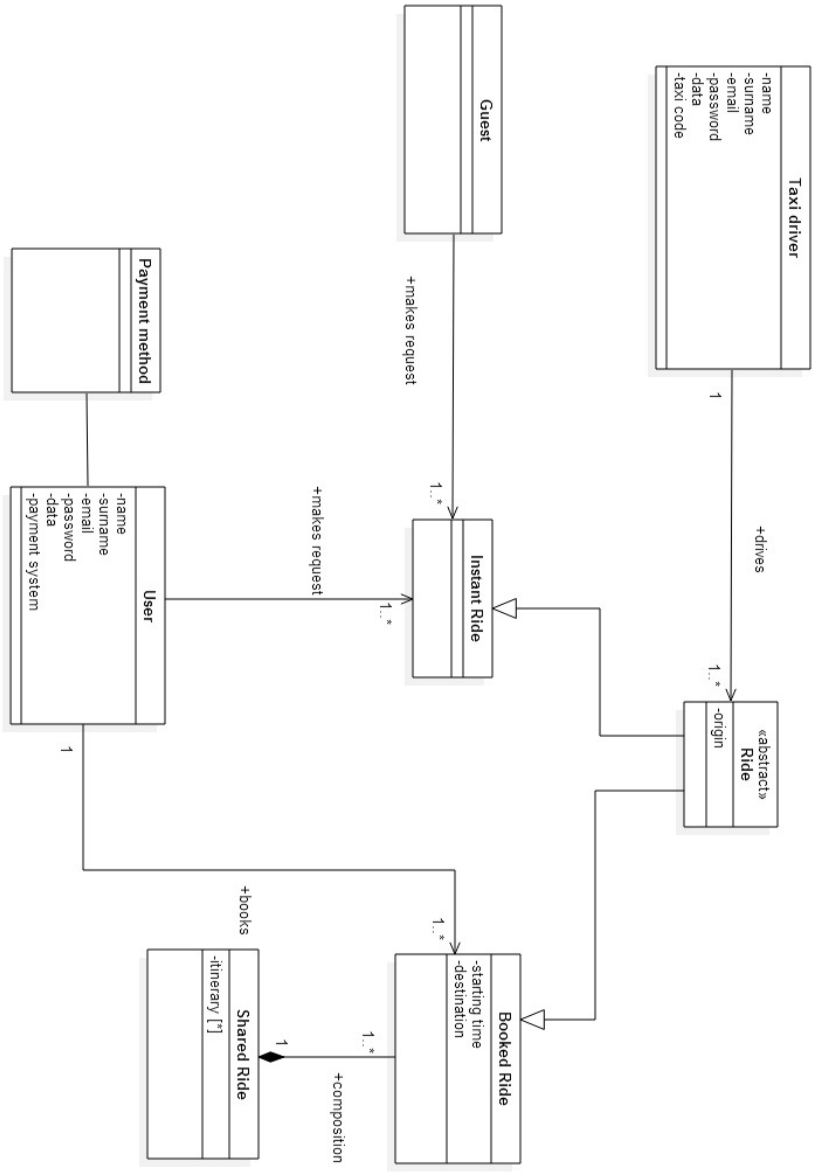


4.4.2 Driver Status

This statechart follows the status of a Taxi Driver throughout the workday



4.5 UML Class Diagram



5 Alloy

5.1 Analysis

The following alloy model is obtained by referring to our class diagram. Our code file has been structured by first inserting signatures, followed by facts, assertions and predicates. The complete code file is available on github. The results of our analysis, namely the generated worlds, are available at the end of this section.

5.2 Signatures

```
//Signatures

abstract sig Ride{
        start: one Place,
        driver: one TaxiDriver
}

sig InstantRide extends Ride{
}

sig BookedRide extends Ride{
        mainPassenger: one User,
        occupation: one Int,
        destination: one Place,
        depTime: one Int
}

sig SharedRide {
        components: set BookedRide,
        itinerary: set Place
}

sig Place{
}

sig User{
}

sig TaxiDriver{
}
```

5.3 Facts

```
//FACTS

//depTime can't be negative
fact NoNegativeTime{
all br: BookedRide | br.depTime >=0
}

//occupation must be at least one
fact NoNullPassenger{
all br: BookedRide | br.occupation > 0
}

fact DriverBounds{
//Every TaxiDriver can't have more than one ride assigned, unless they are componing the same shared ride
all td: TaxiDriver | all disj r1, r2: Ride | assignRide [r1, td] implies (r2.driver !=td or one sr: SharedRide | r1 in sr.components and r2 in sr.components)
//In shared rides, the driver must be same for every part of the itinerary
all sr: SharedRide | all disj br1, br2: BookedRide | (br1 in sr.components and br2 in sr.components) implies br1.driver = br2.driver
}

fact RideBounds{
//
//The starting position and the arriving position can't be the same
all br: BookedRide | !(br.start=br.destination)
//In shared rides, the itinerary must be composed by all the places that are either a starting or a destination point of a booked ride that composes the sharing
all sr: SharedRide | all br: BookedRide | br in sr.components implies (br.start in sr.itinerary and br.destination in sr.itinerary)
//In shared rides, the itinerary must be composed by only the places that are either a starting or a destination point of a booked ride that composes the sharing
all sr: SharedRide | (all p: Place | p in sr.itinerary implies (one rp: Place | (rp in sr.components.start and rp = p) or (rp in sr.components.destination and rp = p)))
//In shared rides,all the components must have a different mainPassenger
all sr: SharedRide | no disj br1, br2: BookedRide | ( br1 in sr.components and br2 in sr.components and br1.mainPassenger = br2.mainPassenger)
//Every shared rides must have at least one component
all sr: SharedRide | #sr.components>0
//Two shared rides can't share the same booked ride as a component
all sr: SharedRide | all br: BookedRide | br in sr.components implies all sr2: SharedRide | br in sr2.components implies sr2=sr
//In shared rides, two components can start at the same place if and only if they start at the same time
all sr: SharedRide | all disj br1, br2: BookedRide | (br1 in sr.components and br2 in sr.components) implies (br1.depTime = br2.depTime iff br1.start = br2.start)
}
```

5.4 Assertions

```
//ASSERTIONS

//Check that each ride has only one Taxi Driver
assert NoMoreThanOneDriver {
all disj td1, td2: TaxiDriver | all r: Ride | (r.driver = td1 implies r.driver != td2) and (r.driver = td2 implies r.driver != td1)
}
check NoMoreThanOneDriver for 5

//Check that when a ride is added to a Shared Ride, it is contained in the components
assert CorrectCompose {
all br: BookedRide | all sr: SharedRide | (composeRide [br,sr] implies br in sr.components)
}
check CorrectCompose for 5

//Check that there are not two ride with the same main passenger in the same shared ride
assert NoSameMainPassenger{
all disj br1, br2: BookedRide | br1.mainPassenger = br2.mainPassenger implies no sr: SharedRide | br1 in sr.components and br2 in sr.components
}
check NoSameMainPassenger for 5


//Check that if a booked ride composes a shared ride, its starting and arriving place are in the shared's itinerary
assert ConsistentItinerary{
all sr: SharedRide | all br: BookedRide | (br in sr.components implies (br.start in sr.itinerary and br.destination in sr.itinerary))
}
check ConsistentItinerary for 5

//Check that a booked ride can't be in the composition of more that of shared ride
assert NoMoreThanOneSharedComponent {
all br: BookedRide | all sr: SharedRide | (br in sr.components implies (no sr2: SharedRide | sr!=sr2 and br in sr2.components))
}
check NoMoreThanOneSharedComponent for 5
```

5.5 Predicates

```
//PREDICATES

pred show(){
#InstantRide>1
#BookedRide>1
#SharedRide>1
#User>1
#TaxiDriver>1
#Place>1 }
run show for 5

//Pred to show the relationship between bookied rides, shared rides, places and drivers
pred showSharing(){
#BookedRide > 1
#components >3
#SharedRide = 1
#User >3
#TaxiDriver > 1
#InstantRide = 0
#mainPassenger > 1
#Place > 1 }
run showSharing for 5

//Pred to show the relationship and the differencies between booked and instant rides
pred showRides(){
#BookedRide >1
#InstantRide >1
#SharedRide = 0 }
run showRides for 5

//Pred to assing a ride to a taxi driver
pred assignRide (r: Ride, td: TaxiDriver){r.driver = td}
run assignRide for 5

//Pred to add a ride to a shared ride
pred composeRide (br: BookedRide, sr: SharedRide) {br in sr.components and br.start in sr.itinerary and br.destination in sr.itinerary}
run composeRide for 5
```

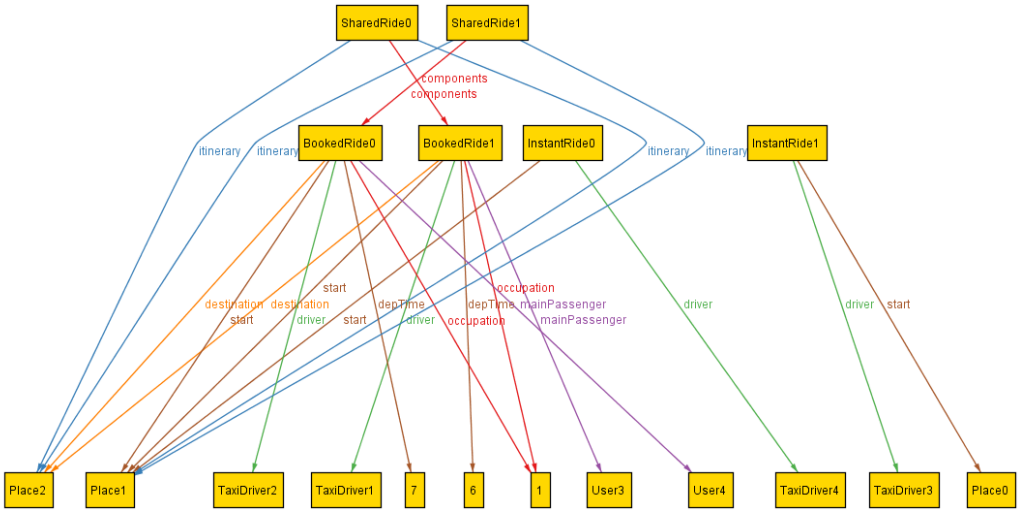
5.6 Review

```
11 commands were executed. The results are:
  #1: No counterexample found. NoMoreThanOneDriver may be valid.
  #2: No counterexample found. CorrectCompose may be valid.
  #3: No counterexample found. NoSameMainPassenger may be valid.
  #4: No counterexample found. NoTooCloseRide may be valid.
  #5: No counterexample found. ConsistentItinerary may be valid.
  #6: No counterexample found. NoMoreThanOneSharedComponent may be valid.
  #7: Instance found. show is consistent.
  #8: Instance found. showSharing is consistent.
  #9: Instance found. showRides is consistent.
  #10: Instance found. assignRide is consistent.
  #11: Instance found. composeRide is consistent.
```

5.7 Generated Worlds

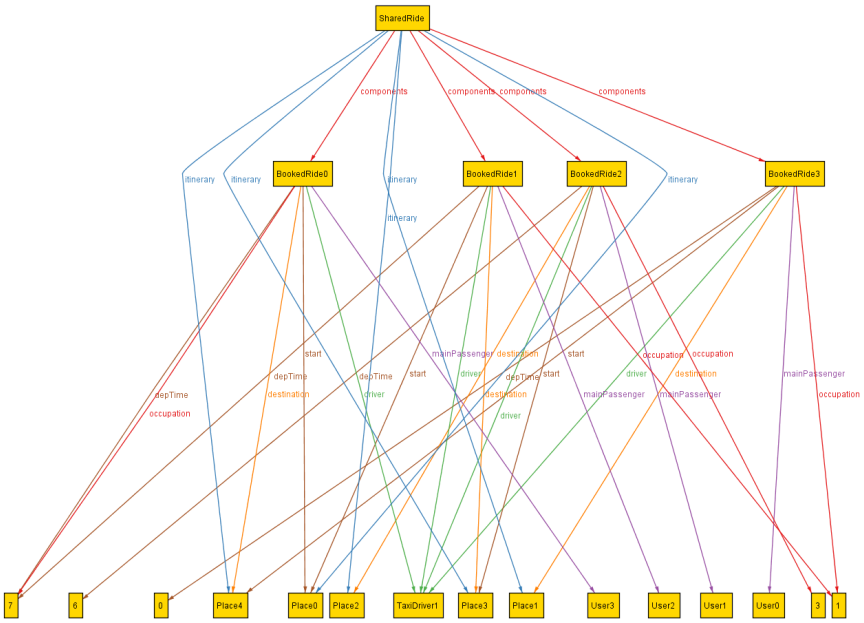
5.7.1 Basic show() predicate

This diagram is the result of "run show() for 5". It was generated to represent the differences between shared, booked and instant rides.



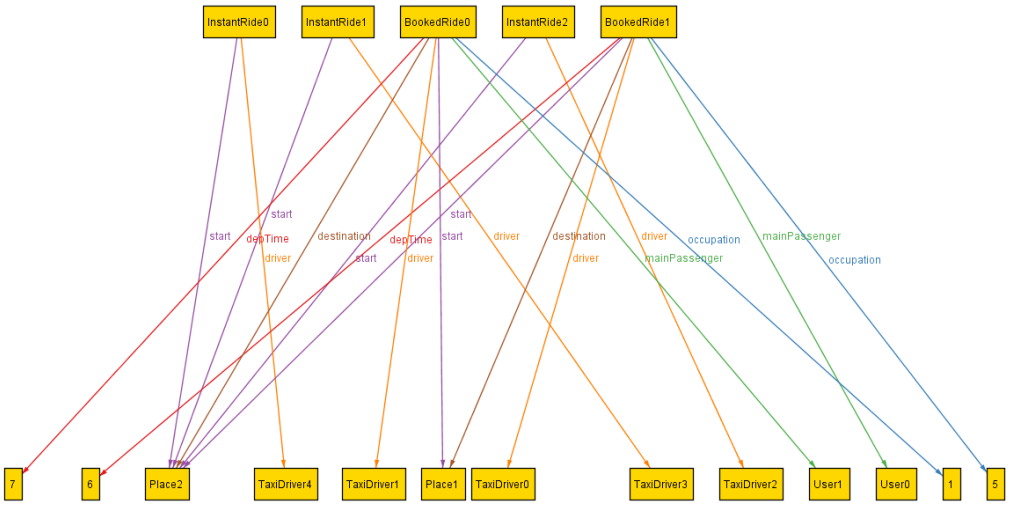
5.7.2 showSharing() predicate

This diagram is the result of "run showSharing() for 5". It was generated to represent the relationship between shared and booked rides.



5.7.3 showRides() predicate

This diagram is the result of "run showRides() for 5". It was generated to show the differences between booked and instant rides.



6 Appendix

6.1 Software and Tools Used

- MiKTeX (http://www.miktex.org/) to format and create this document.
- StarUML (http://www.staruml.io/) used to create UML diagrams: Use Case, Sequence and Class Diagrams as well as State Charts.
- Balsamiq Mockups (http://www.balsamiq.com/products/mockups/) used to create mockups for mobile and desktop webpages.
- Alloy Analazyer (http://alloy.mit.edu/) to verify the validity of our model.

6.2 Work Hours

Time spent on the creation of this RASD:

- Filippo Ciceri: 28 hours
- Federico Cesaro: 28 hours
- Luca Capecchi: 28 hours