

myTaxyService

# Requirements Analysis and Specifications Document

Matteo Farè, Federico Feresini, Thierry Ebali Essomba

November 6, 2015

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Purpose . . . . .	4
1.2	Actual system . . . . .	4
1.3	Scope . . . . .	4
1.4	Acronyms and Definitions . . . . .	5
1.4.1	Acronyms . . . . .	5
1.5	Reference Documents . . . . .	5
<b>2</b>	<b>Overall Description</b>	<b>6</b>
2.1	Product Perspective . . . . .	6
2.2	Product Functions . . . . .	6
2.3	User Characteristics . . . . .	7
2.4	Constraints . . . . .	7
2.5	Assumptions and Dependencies . . . . .	7
2.5.1	Assumptions . . . . .	7
2.5.2	Hardware Dependencies . . . . .	8
2.6	Apportioning of Requirements . . . . .	8
<b>3</b>	<b>Specific Requirements</b>	<b>9</b>
3.1	External Interfaces . . . . .	9
3.1.1	User Interfaces . . . . .	9
3.1.1.1	Registration . . . . .	10
3.1.1.2	Login . . . . .	11
3.1.1.3	Taxi Request . . . . .	13
3.1.1.4	Taxi Reservation . . . . .	14
3.1.1.5	Shared Ride . . . . .	15
3.1.1.6	Taxi Driver Interfaces . . . . .	16
3.1.2	API Interfaces . . . . .	16
3.1.3	Software Interfaces . . . . .	16
3.1.4	Hardware Interfaces . . . . .	17
3.1.5	Communication Interfaces . . . . .	17
3.1.6	Memory Constraints . . . . .	17
3.2	Functional Requirements . . . . .	17

3.3	Scenarios . . . . .	20
3.3.1	Scenario 1 . . . . .	20
3.3.2	Scenario 2 . . . . .	20
3.3.3	Scenario 3 . . . . .	20
3.3.4	Scenario 4 . . . . .	21
3.3.5	Scenario 5 . . . . .	21
3.3.6	Scenario 6 . . . . .	21
3.3.7	Scenario 7 . . . . .	21
3.3.8	Scenario 8 . . . . .	21
3.4	UML Models . . . . .	22
3.4.1	Use Case . . . . .	22
3.4.1.1	Registration . . . . .	23
3.4.1.2	Login . . . . .	24
3.4.1.3	Taxi Request . . . . .	25
3.4.1.4	Taxi Sharing . . . . .	26
3.4.1.5	Payment . . . . .	27
3.4.1.6	Deleting Request . . . . .	28
3.4.1.7	Manage Customer Requests . . . . .	29
3.4.1.8	Set Availability . . . . .	30
3.4.2	Class Diagram . . . . .	31
3.4.3	Sequence Diagrams . . . . .	32
3.4.3.1	Registration . . . . .	32
3.4.3.2	Login . . . . .	33
3.4.3.3	Taxi Request . . . . .	34
3.4.3.4	Taxi Sharing . . . . .	35
3.4.3.5	Status Management . . . . .	35
3.4.3.6	Delete Reservation . . . . .	36
3.4.4	Activity Diagrams . . . . .	37
3.4.4.1	Registration . . . . .	37
3.4.4.2	Login . . . . .	38
3.4.4.3	Taxi Request . . . . .	39
3.4.4.4	Taxi Sharing . . . . .	40
3.4.4.5	Status Management . . . . .	41
3.4.4.6	Deleting Reservation . . . . .	42
3.5	Performance Requirements . . . . .	42
3.6	Logical Database Requirements . . . . .	43
3.7	Design Constraints . . . . .	43
3.8	Software system attributes . . . . .	43
3.8.1	Reliability . . . . .	43
3.8.2	Availability . . . . .	43
3.9	Security . . . . .	43
3.10	Maintainability . . . . .	43
3.11	Portability . . . . .	44

<b>4</b>	<b>Appendix</b>	<b>45</b>
4.1	Alloy . . . . .	45
4.1.1	Signature . . . . .	45
4.1.2	Functions . . . . .	46
4.1.3	Facts . . . . .	46
4.1.4	Generated World . . . . .	47
4.1.4.1	Globe . . . . .	55
4.1.5	Software Used . . . . .	56
4.2	Hours of Work . . . . .	56

# Chapter 1

## Introduction

### 1.1 Purpose

This Requirement Analysis and Specification (RASD) document aims to set forth the system requirements, discerning between functional and non functional ones. The core functionality that must be provided by the system and the possible extensions are described during the following document. The output of this work will drive the design, implementation and evaluation of the myTaxiService system. Identifying the appropriate user groups and individuals is a key step when defining system requirement; different classes of users have been identified together with an analysis of the tasks they perform and the scenarios in which myTaxiService can be expected to operate for these users. The analysis has aimed at addressing questions such as what users typically need from this kind of service, what type of information they need and how they look for it, to enable them to carry out their activities more effectively, and how these users would expect myTaxiService to respond to their requests.

### 1.2 Actual system

Currently the only way for a customer to request or to reserve a taxi ride is to call the telephone number of the company. This method is old and inconvenient. myTaxiService will speed up and make a lot easier those operations.

### 1.3 Scope

The aim of the project is to create a web and a mobile application in order to optimize the taxi service of a large city. Users can request a taxi, by specifying their current position, and the system sends them back the code of the incoming taxi and the awaiting time. Users can also reserve a taxi, at least two hours before the ride, by specifying the origin and the destination

of it. Furthermore, users could decide to share a ride, with passengers that are in the same zone and are going in the same direction.

Taxi drivers use the application to notify their availability and to confirm that they have accepted a call.

The system manages a taxi queue system, in order to offer a better service. The city is divided in ten zones, each one associated to an independent queue of available taxis. When a user asks for a ride, the request is forwarded to the first taxi of the queue of that zone. If the taxi drivers confirms, the system sends all the necessary informations to the passenger; otherwise the request is forwarded to the second available taxi, and the first one is moved in the last position of the queue.

## **1.4 Acronyms**

In this document, we have used these acronyms:

- RASD: Requirements Analysis and Specification Document.
- OS: Operating System.
- GPS: Global Positioning System.
- DBMS: Data Base Management System.
- JVM: Java Virtual Machine

## **1.5 Reference Documents**

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

## Chapter 2

# Overall Description

### 2.1 Product Perspective

The myServiceTaxi project is a new product intended for use on the Android and iOS platform. A web application intended for desktop users will also be developed. Furthermore there is a server-side component which will be responsible for database and synchronization services. The aim of the project encompasses both server and client side functionalities, so both aspects are covered in detail within this document.

### 2.2 Product Functions

The following list provide a brief summary of the main functionalities and features of the myTaxiService system. The functions are split into two categories: core functions and additional functions. The first one are essential to the application's operation, whereas additional functions simply add new functionalities.

Core Functions:

- User registration.
- Ride request.
- Setting status option.

Additional Functions:

- Ride reservation.
- Taxi sharing.

## 2.3 User Characteristics

myTaxiService project is intended for use by five user classes:

- **Visitors** can only see the sign up interface, through which they can register giving all the requested personal informations.
- **Registered Users** can reserve a ride, calculate a ride cost, know the position of the closest taxi, or share the ride with other passengers. It is not required any basic understanding.
- **Taxi Drivers** have to give the identifying code of their taxi in order to successfully log in the application. Moreover they can manage their status and answer to the customers' requests.
- **Developers** who are interested in making the application better, find and correct bugs.
- **Testers** who use the beta versions of the product and test it in many ways for bugs and errors.

## 2.4 Constraints

- GUI is only in English.
- Only registered customers and taxi drivers will be authorized to use the services.
- Limited to HTTP/HTTPS
- Limited screen size and resolution for mobile devices.
- Limited memory and processing power.
- myTaxiService is meant to be quick and responsive, even when dealing with numerous transactions, so each feature must be designed and implemented with efficiency.
- On desktop systems, myTaxiService is implemented in Java. To run the system, JVM is required.

## 2.5 Assumptions and Dependencies

### 2.5.1 Assumptions

We have made these assumptions:

- Customers can't request for a ride starting from a place out of the city.



- Customers can't select a place out of the city as the destination of their ride.
- Only available drivers are present in a queue.
- In each tail there is always at least one taxi driver.
- Customers have always enough available credit to pay the ride.
- Taxi Drivers do not reject many rides consecutively.
- Customers specify their current location with an acceptable margin of error.
- Taxi drivers reach the meeting place with at most a minimal delay.

### **2.5.2 Hardware Dependencies**

The application will use the location sensors (GPS) to record the location of a specific customer or taxi. The GPS functionalities will be achieved using the API provided by the operating system.

## **2.6 Apportioning of Requirements**

myTaxiService could be translated in other different languages in future versions of the system.

## Chapter 3

# Specific Requirements

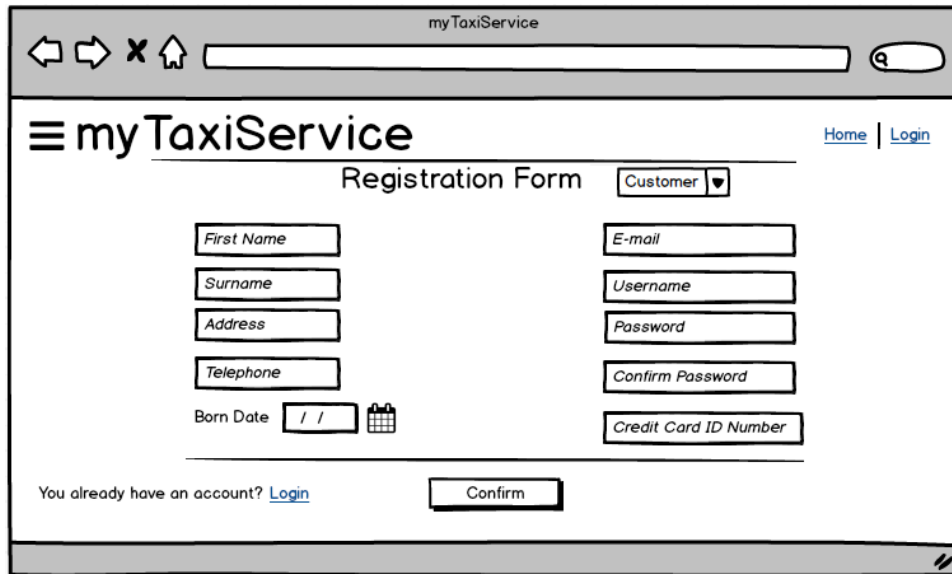
### 3.1 External Interfaces

#### 3.1.1 User Interfaces

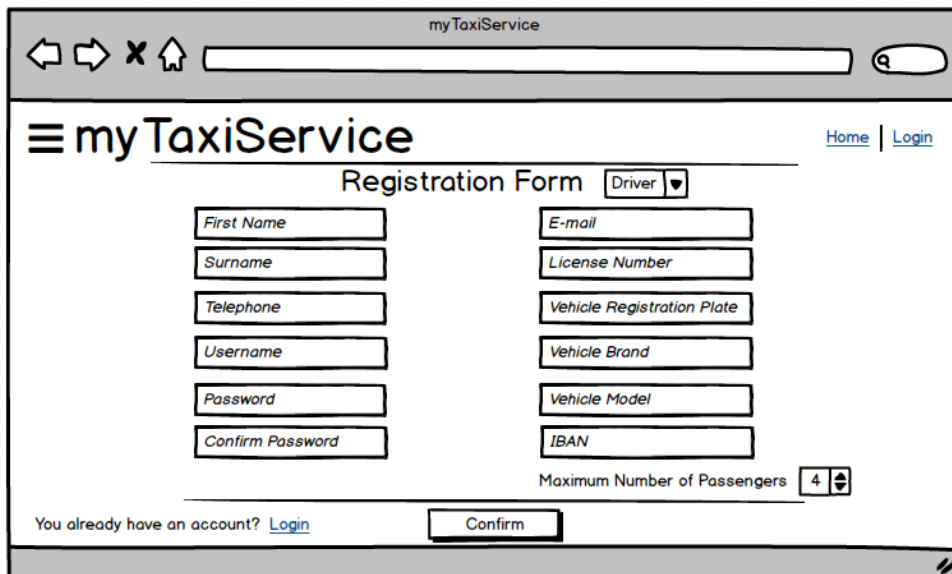
The system offers an interface of signing up asking for all needed credentials to be registered. After the logging in interface which an user can access the application with, he will be able to make a request for an available ride. This interface will ask him for the starting point, the desired destination and if he want to use the sharing ride option. The taxi driver can access his personal area giving his taxi code, username and password. Then he will be able to answer the customers' requests or to set his own status on available or not. Here are some mockups that explain better the structure of either the web-app and the mobile app. myTaxiService will be supported by most of commercial smartphones, tables and laptops. Here are some mockups that explain better the structure of either the web-app and the mobile app.

### 3.1.1.1 Registration

These two muckups show the desktop registration form, both the customer and the taxi driver one



The screenshot shows a web browser window titled "myTaxiService". The page has a header with the site name and navigation links for "Home" and "Login". Below the header is a "Registration Form" with a dropdown menu set to "Customer". The form contains two columns of input fields: "First Name", "Surname", "Address", "Telephone", "Born Date" (with a date picker), "E-mail", "Username", "Password", "Confirm Password", and "Credit Card ID Number". At the bottom, there is a link "You already have an account? [Login](#)" and a "Confirm" button.



The screenshot shows a web browser window titled "myTaxiService". The page has a header with the site name and navigation links for "Home" and "Login". Below the header is a "Registration Form" with a dropdown menu set to "Driver". The form contains two columns of input fields: "First Name", "Surname", "Telephone", "Username", "Password", "Confirm Password", "E-mail", "License Number", "Vehicle Registration Plate", "Vehicle Brand", "Vehicle Model", and "IBAN". At the bottom, there is a link "You already have an account? [Login](#)", a "Confirm" button, and a "Maximum Number of Passengers" field with a value of 4 and a dropdown arrow.

### 3.1.1.2 Login

These are the desktop login interfaces.



myTaxiService

Please sign in to access the service  
You don't have an account? [Sign Up](#)

Customer Driver

username  
password

[Forgot Username?](#)  
[Forgot Password?](#)

Sign in

Download our mobile application  
Available on all mobile stores

This is a wireframe of a desktop login interface for 'myTaxiService'. The browser window has a title bar with navigation icons and a search bar. The page header includes a hamburger menu icon, the brand name 'myTaxiService', and a taxi icon. The main content area is divided into two sections. The left section is for user login, featuring a toggle for 'Customer' (selected) and 'Driver'. It contains input fields for 'username' and 'password', links for 'Forgot Username?' and 'Forgot Password?', and a 'Sign in' button. The right section promotes a mobile application, showing a smartphone and a tablet with the text 'Download our mobile application' and 'Available on all mobile stores'.



myTaxiService

Please sign in to access the service  
You don't have an account? [Sign Up](#)

Customer Driver

username  
password  
taxi code

[Forgot Username?](#)  
[Forgot Password?](#)

Sign in

Download our mobile application  
Available on all mobile stores

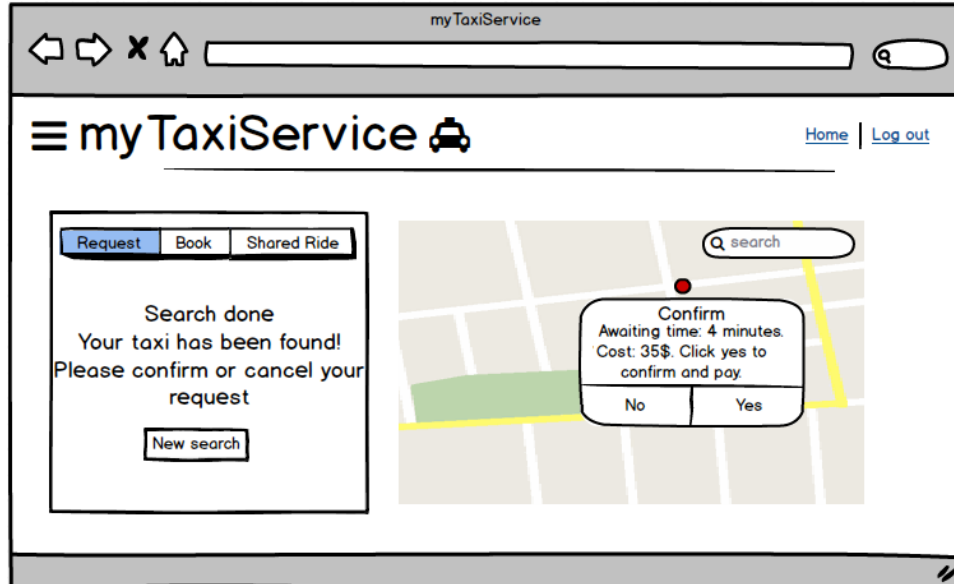
This is a wireframe of a desktop login interface for 'myTaxiService', specifically for drivers. The layout is identical to the customer interface, but the 'Driver' toggle is selected. The login form includes an additional 'taxi code' input field below the password field. The right section remains the same, promoting the mobile application.

And these are the mobile ones.



### 3.1.1.3 Taxi Request

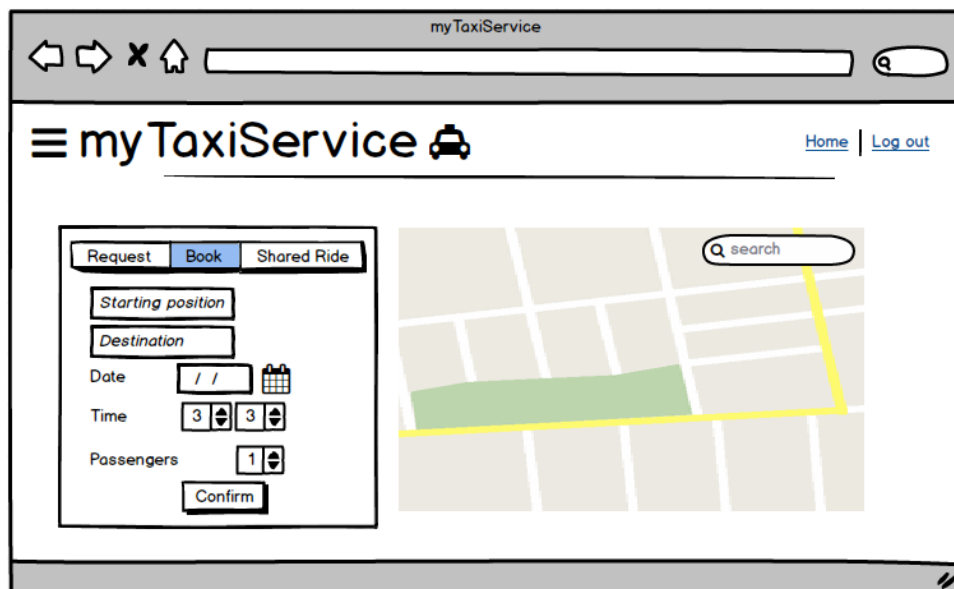
These mockups show the interfaces through which an user can request a taxi ride.





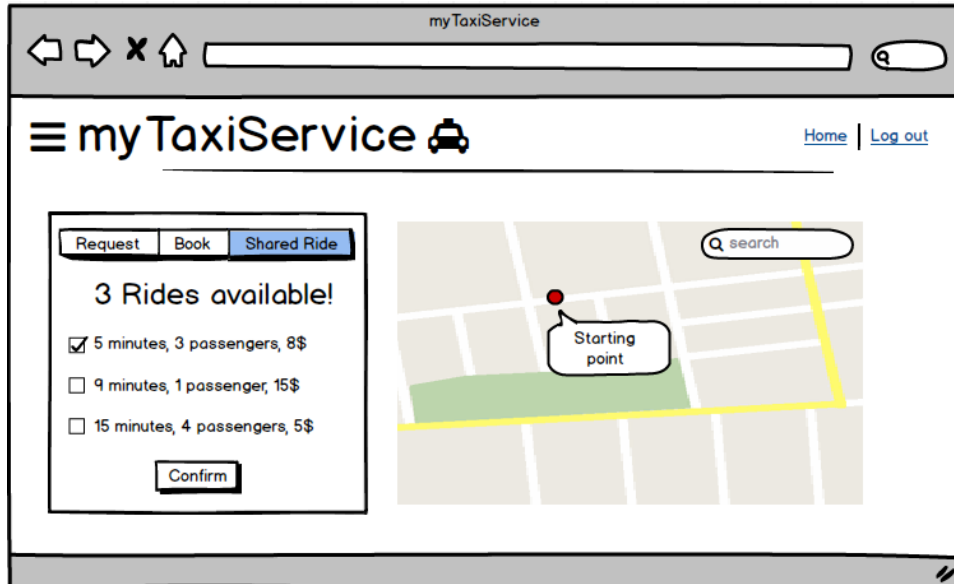
#### 3.1.1.4 Taxi Reservation

This mockup shows the taxi reservation interface.



### 3.1.1.5 Shared Ride

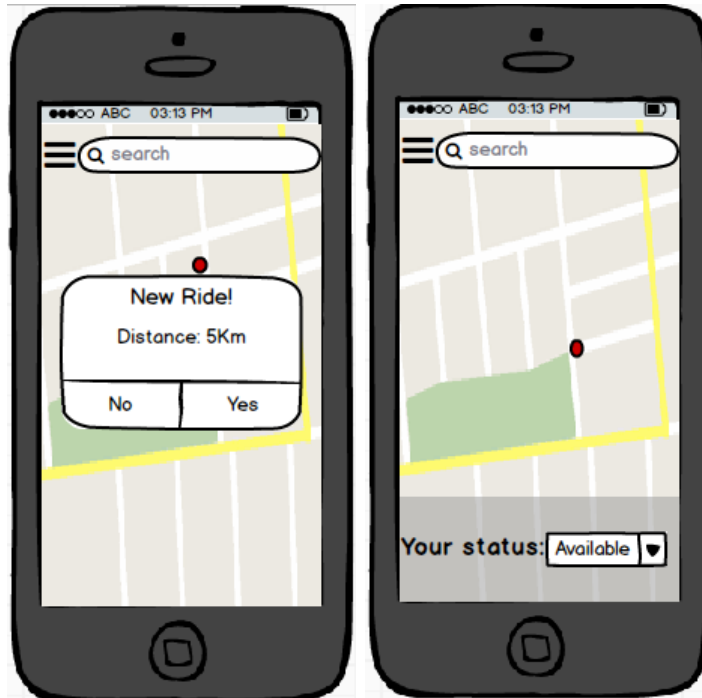
These two mockups show the interface through which is possible to start a shared ride. The second one shows the list of available shared rides.





### 3.1.1.6 Taxi Driver Interfaces

This two mockups show the interfaces through which a taxi driver can manage his status and confirm or reject a ride request.



### 3.1.2 API Interfaces

The system will interface with Google to use the Google Maps API for calculating the path of the rides and to identify the customers and taxi drivers' positions. The system will also interface a database to hold the registered data by users and taxi drivers.

### 3.1.3 Software Interfaces

#### 1. Database Management System (DBMS)

- Name: MySQL
- Version: 5.7
- Source: <http://www.mysql.it/>

#### 2. Java Virtual Machine (JVM)

- Name: JEE
- Version: 7

- Source: <http://www.oracle.com/technetwork/java/javaee/tech/index.html>

### 3. Google Maps

- Name: Google Maps Javascript API
- Source: <https://developers.google.com/maps/documentation/javascript/get-api-key>

#### 3.1.4 Hardware Interfaces

For the desktop version of myTaxiService to run, you will need a PC (regardless of its operating system, it can be Windows or Linux) or Mac. Since it doesn't require so many resources, it will run on most system without problems. The mobile version will be available on all smart-phones and tablets whose operating system are Android or iOS.

#### 3.1.5 Communication Interfaces

Desktop clients will need a standard Wi-Fi connection (IEEE 802.11) to connect to the system. Mobile clients will need at least a 3G connection. All users will connect with HTTP/HTTPS protocol.

#### 3.1.6 Memory Constraints

The system will need a maximum of external storing capacity of 4TB. To install the application on mobile phones or tablets we are going to request a maximum of 30MB.

## 3.2 Functional Requirements

1. A visitor is allowed to register through a user-friendly sign up interface.
  - The system shall provide a sign up interface asking for first name, surname, born date, address, telephone number, username, password and credit card ID for each customer that want to use the service.
  - The system shall be able to recover all personal data of users that register through their social network accounts.
  - The system shall save all registration data of users in his database.
2. A registered user is allowed to login with username and password, or by a social network account.
  - The system shall provide a sign in interface through which a customer can login with his username or e-mail, and his password.

- The system shall provide a function that lets users login with their social network accounts.
3. Taxi drivers are allowed to register through a user-friendly sign up interface.
    - The system shall provide a sign up interface through which a taxi driver can register giving his name, surname, e-mail address, telephone number, licence number, username, password, vehicle number plate, vehicle brand, vehicle model, IBAN, and the maximum number of passengers that he can carry simultaneously.
    - The system shall assign to each driver a unique taxi code.
    - The system shall save all registration data of taxi drivers in his database.
  4. Taxi drivers are allowed to log in by giving the identifying code of their taxi.
    - The system shall provide a sign in interface that lets a taxi driver login with his username, password and taxi code.
  5. An user that has forgotten his password can retrieve it using the given function.
    - The system shall modify the saved data of an user that wants to change his password removing his old one.
    - The system shall send to the user an e-mail with the link through which he can set a new password.
    - The system shall save in the database the new password of the user.
  6. An user is allowed to request a taxi ride giving his current position, the desired destination and the number of passengers.
    - The system shall provide an interface through which an user can specify the starting point and the destination of a ride.
    - The system shall forward users' requests to the first taxi of the queue of the zone that the selected starting point belongs to.
    - The system shall forward a user's request to the next taxi in the queue if the first one rejects it or after a timer expiration (2 minutes).
    - The system shall answer a user request with the awaiting time for his taxi and the ride cost, and it must let him accept or reject that ride.

- The system shall entrust the transactions management to an external system, in the event that the customer confirms the ride.
  - The system shall forward user's decision to the taxi driver.
7. An user is allowed to reserve a ride booking it at least two hours before the meeting, specifying the starting and the ending point of the ride, and the number of passengers.
- The system shall provide an interface through which an user could reserve a ride by specifying: starting point, destination, number of passengers, date and time of the ride.
  - The system shall check the time chosen by the user; if it is at least two hours after the reservation, it must save all the data of the ride. Otherwise it must reject the reservation.
8. A taxi driver is allowed to manage his own current status (available or not).
- The system shall provide an interface that lets a taxi driver change his current status between available and not available.
  - The system shall add a taxi driver at the end of the queue of his zone when his status becomes available.
9. A taxi driver is allowed to accept or reject a request of a ride.
- The system shall provide a function that lets each taxi driver accept or reject a ride asked by an user.
  - The system shall update the queue moving in last position an available taxi driver that rejects a ride.
10. An user is allowed to share a ride with other passengers, whose destination is similar.
- The system shall provide an interface that lets an user join an existing shared ride or create a new one, by specifying his position, the desired destination and the number of passengers.
  - The system shall provide to each user that search an existing shared ride a list of all available rides, specifying starting point, leaving time, number of confirmed passengers and cost of the ride (depending on the number of passengers).
  - The system shall provide an interface that lets a user select his favourite shared ride.
  - The system shall keep the client up to every price modification.

- The system shall forward all the new information to the selected taxi driver.
11. The system shall manage transactions redirecting payments towards an external system.
- The system shall manage the possibility to cancel the client's reservation with the following policy:
    - (a) If the reservation is cancelled until 15 minutes before the ride, the full amount is returned.
    - (b) If the reservation is cancelled into the 15 minutes before the ride, 50% of the amount is returned.
    - (c) If the reservation is not cancelled, the client hasn't the right to claim the return of the full amount paid.

### **3.3 Scenarios**

#### **3.3.1 Scenario 1**

Matteo has just come back from a business trip. When he arrives at the airport he needs to find an efficient and quickly way to get home. Searching on the web he reads about the new application myTaxiService and after being informed on the type of service, he decides to download the mobile application and to sign up. He fills in the registration form, submitting all his personal informations. The system accepts his registration and Matteo can visualize his personal page and request a taxi ride.

#### **3.3.2 Scenario 2**

Federico has just obtained his taxi license. Searching on the web he reads about a new application called myTaxiService. Federico decides to start his new activity downloading the mobile application on his smart-phone and signing up. He completes the registration form, giving all his personal data and other information about his vehicle. The system accepts his registration so Federico can visualize his personal page and wait for customer to request rides.

#### **3.3.3 Scenario 3**

Thierry, a taxi driver, wants to set his status on available and start receiving customer's requests for rides. But when he enters his email, password and taxi code the system displays an error message in order to notify him that the password doesn't match with the email or with the taxi code. The system automatically cancels all the inputs of the sign in form but the email. Now

Thierry can enter his data once more to log in, or he can decide to retrieve the forgotten password.

#### **3.3.4 Scenario 4**

Paolo is a taxi driver and a member of myTaxiService. He has just completed a ride leading the customer to his destination. In order to receive a new customer he sets his status on "available" through his personal account so that the system can add him to the right queue related to his current position.

#### **3.3.5 Scenario 5**

Mauro has just send a taxi request with his account on myTaxiService. Thanks to the partnership with PayPal he doesn't have to worry about the payment that will be carried out automatically with the data entered during the registration.

#### **3.3.6 Scenario 6**

Paolo is waiting for a customer request. MyTaxiService receives a request and redirects it to Paolo, that is the first driver in his queue. Paolo receives a notification of the new request but a mishap does not allow him to accept it, then he rejects the request.

#### **3.3.7 Scenario 7**

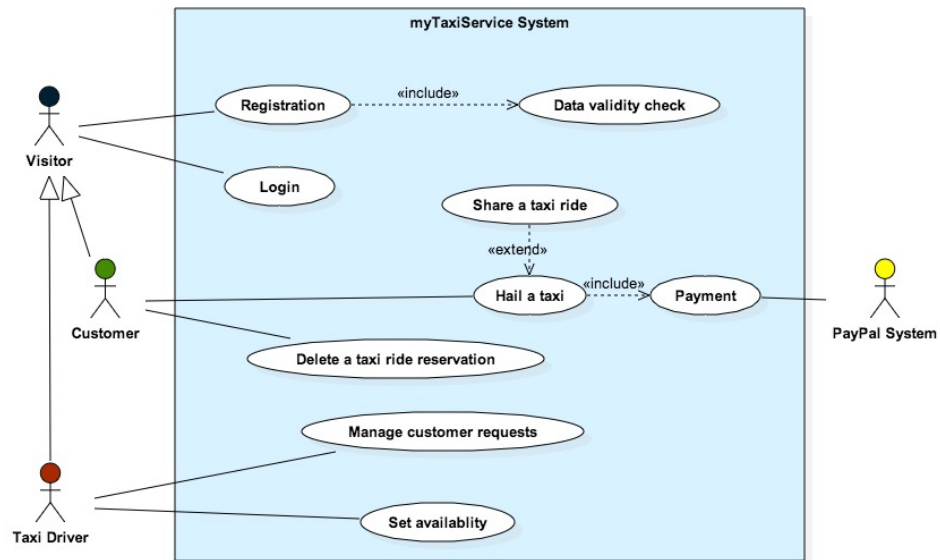
Davide is looking for a shared taxi ride in order to save money. After that he has entered all the ride data, the application returns a list of possible solutions sorted by awaiting time so that Davide can choose the most convenient for his needs.

#### **3.3.8 Scenario 8**

Due to a logistic problem Pietro failed to show up at the meeting with the taxi driver. He hasn't even been able to cancel the request, then he will not be entitled to any refund of the payment.

## 3.4 UML Models

### 3.4.1 Use Case



### 3.4.1.1 Registration

Use case name	Registration
Related Requirement	The visitor specifies his personal data during the registration
Goal in context	A visitor requests to be registered by the system
Precondition	No precondition is needed
Successful end condition	A containing user data record is saved into the database
Failed end condition	No record is created
Primary actor	Visitor
Trigger	The visitor fills the registration form through a web page
Main Flow	Step and Action:
	1- The visitor accesses to the registration web page
	2- The visitor fills the registration form
	3- The visitor sends the registration request
Extensions	Step and Branching Action:
	2.1- The visitor specifies if he is a customer or a taxi driver



### 3.4.1.2 Login

Use case name	Log-in
Related Requirement	The registered user fills the sign-in form by specifying email address and related password to access
Goal in context	The user accesses to myTaxiService
Precondition	The user needs to be registered
Successful end condition	The user accesses to the service
Failed end condition	The user can't access to the service
Primary actor	Visitor
Trigger	The user fills the sign-in form
Main Flow	Step and Action:
	1- The user accesses to the sign in form
	2- The user enters his data used during the registration
	3- The user sends the sign in request
Extensions	Step and Branching Action:
	2.1- If the user is a Taxi Driver, he has to specify the taxi code

### 3.4.1.3 Taxi Request

Use case name	Hail a taxi
Related Requirement	The customer fills the form by specifying the starting point and the destination of the ride.
Goal in context	A customer hails a taxi
Precondition	The customer needs to be registered
Successful end condition	The customer sends a ride request to a taxi driver
Failed end condition	No request is sent by the customer
Primary actor	Customer
Trigger	The customer fills the taxi ride reservation form
Main Flow	Step and Action:
	1- The customer fills the taxi ride reservation form
	2- A taxi ride request is sent to the first free taxi driver in queue
	3- The customer gets back an offer specifying the cost of the ride
	4- The customer manages the offer
	5- The system notifies the taxi driver about the choice of the customer
Extensions	Step and Branching Action:
	1.1- The customer specifies the starting point and the destination of the ride
	1.2- The customer can choose the sharing ride option
	4.1- The customer can choose to accept the offer or to reject it

#### 3.4.1.4 Taxi Sharing

Use case name	Manage customer requests
Related Requirement	The taxi driver receives a list of requests by the customers according to his priority in the queue. He can confirm or reject the ride
Goal in context	The taxi driver manages customer requests
Precondition	The taxi driver needs to be registered and he has to have at least one related ride request
Successful end condition	The taxi driver manages customer requests and the system notifies the costumer about the taxi driver decision
Failed end condition	The customer request is not managed
Primary actor	Taxi Driver
Trigger	The taxi driver clicks the button to confirm or reject the request
Main Flow	Step and Action:
	1- The taxi driver accesses to the list of the requests provided by the system according to his priority in the queue
	2- The taxi driver manages the request
	3- The system notifies the customer about the taxi driver choice
Extensions	Step and Branching Action:
	2.1-The taxi driver can choose to confirm a request or reject it

#### 3.4.1.5 Payment

Use case name	Payment
Related Requirement	At the moment of paying the customer is redirected to an external system (PayPal)
Goal in context	The customer confirms his reservation with the payment
Precondition	The customer needs to be registered and he needs to have a related taxi ride offer to confirm
Successful end condition	The reservation is confirmed and the priority taxi driver queue is recalculated
Failed end condition	The reservation fails
Primary actor	Customer
Secondary actor	PayPal's system
Trigger	The customer completes the reservation operation
Main Flow	Step and Action:
	1- The customer is redirected to an external system
	2- PayPal's system charges the customer the amount to pay
	3- The system notifies the customer that the operation succeeded
Extensions	Step and Branching Action:
	2.1- The system calculates the quota which the customer has to pay for the chosen ride

### 3.4.1.6 Deleting Request

Use case name	Delete taxi ride reservation
Related Requirement	The customer can choose to delete his ride before it takes place
Goal in context	The customer deletes the ride
Precondition	The customer needs to be registered and he has to have a related reservation
Successful end condition	The customer deletes the ride and he is refunded according to the policy of the system
Failed end condition	The customer doesn't delete the ride
Primary actor	Customer
Trigger	The customer clicks the deleting button
Main Flow	Step and Action:
	1- The customer accesses to his taken reservation
	2- The customer selects a ride
	3- The customer deletes the reservation
	4- The system notifies the taxi driver
	5- The priority queue is recalculated
Extensions	Step and Branching Action:
	3.1-The customer specifies the ride he wants to delete
	3.2- The customer is refunded

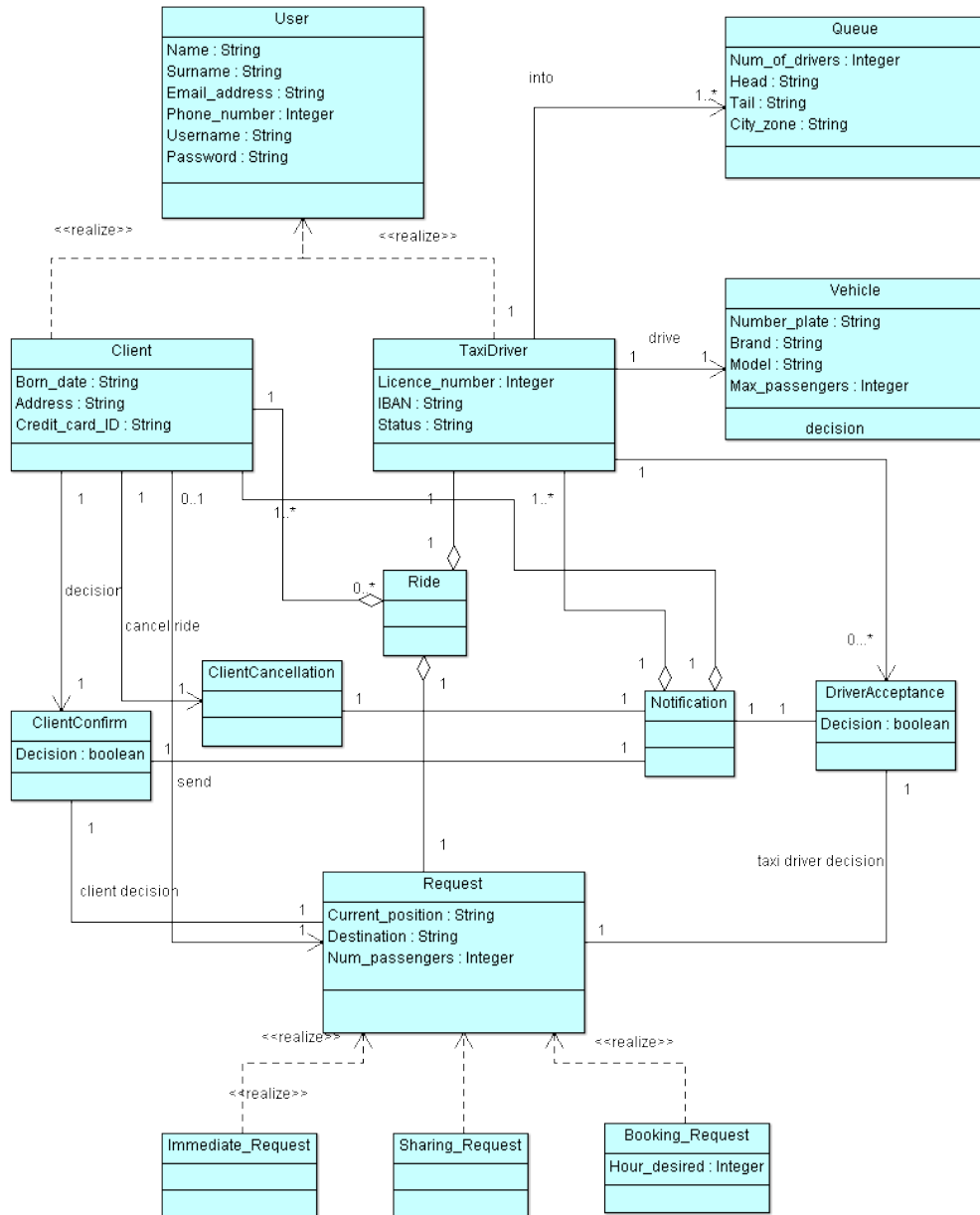
### 3.4.1.7 Manage Customer Requests

Use case name	Manage customer requests
Related Requirement	The taxi driver receives a list of requests by the customers according to his priority in the queue. He can confirm or reject the ride
Goal in context	The taxi driver manages customer requests
Precondition	The taxi driver needs to be registered and he has to have at least one related ride request
Successful end condition	The taxi driver manages customer requests and the system notifies the costumer about the taxi driver decision
Failed end condition	The customer request is not managed
Primary actor	Taxi Driver
Trigger	The taxi driver clicks the button to confirm or reject the request
Main Flow	Step and Action:
	1- The taxi driver accesses to the list of the requests provided by the system according to his priority in the queue
	2- The taxi driver manages the request
	3- The system notifies the customer about the taxi driver choice
Extensions	Step and Branching Action:
	2.1-The taxi driver can choose to confirm a request or reject it

### 3.4.1.8 Set Availability

Use case name	Set Availability
Related Requirement	Taxi driver handles his status by setting the availability of his taxi
Goal in context	Taxi driver sets his availability
Precondition	Taxi drivers needs to sign in before managing his availability status
Successful end condition	Taxi driver status changes availability
Failed end condition	Taxi drivers status doesn't change availability
Primary actor	Taxi Driver
Trigger	Taxi driver clicks on the changing status button
Main Flow	Step and Action:
	1- Taxi driver accesses to his personal area
	2- Taxi driver clicks on the button to change the status
	3- The system recalculates the taxi driver queue

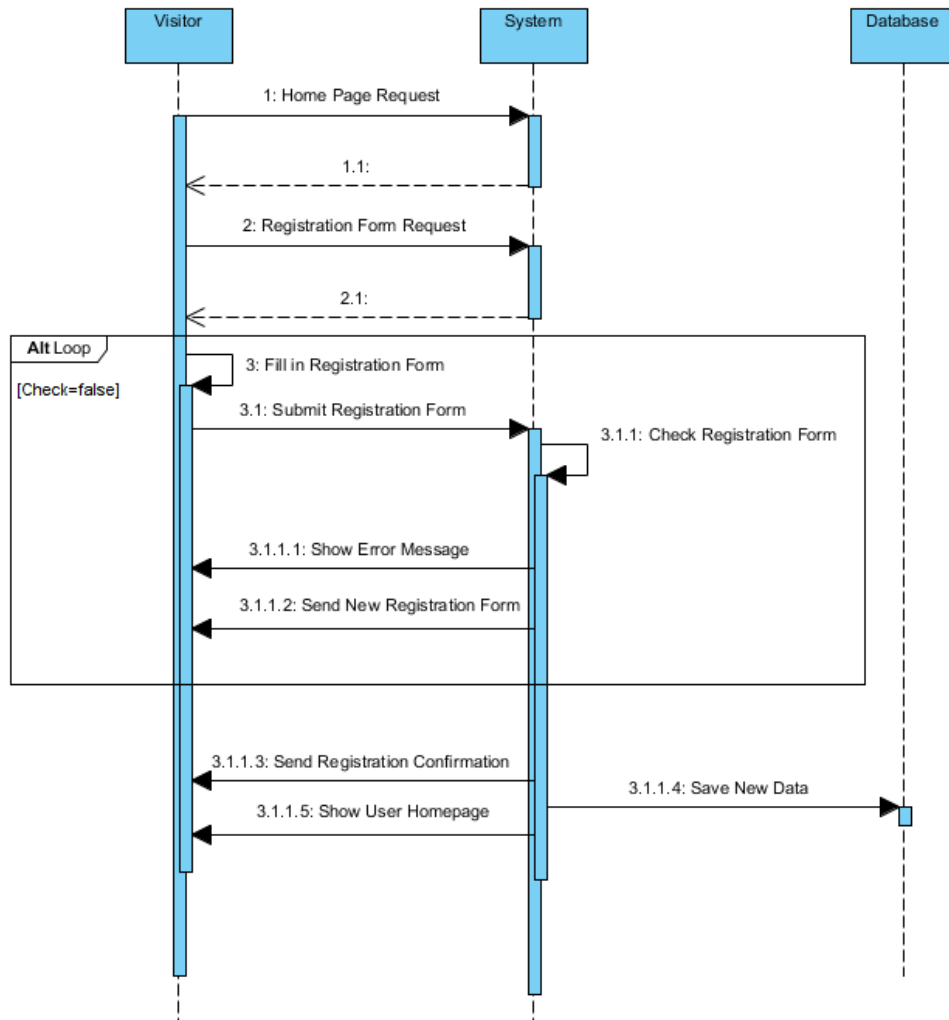
### 3.4.2 Class Diagram



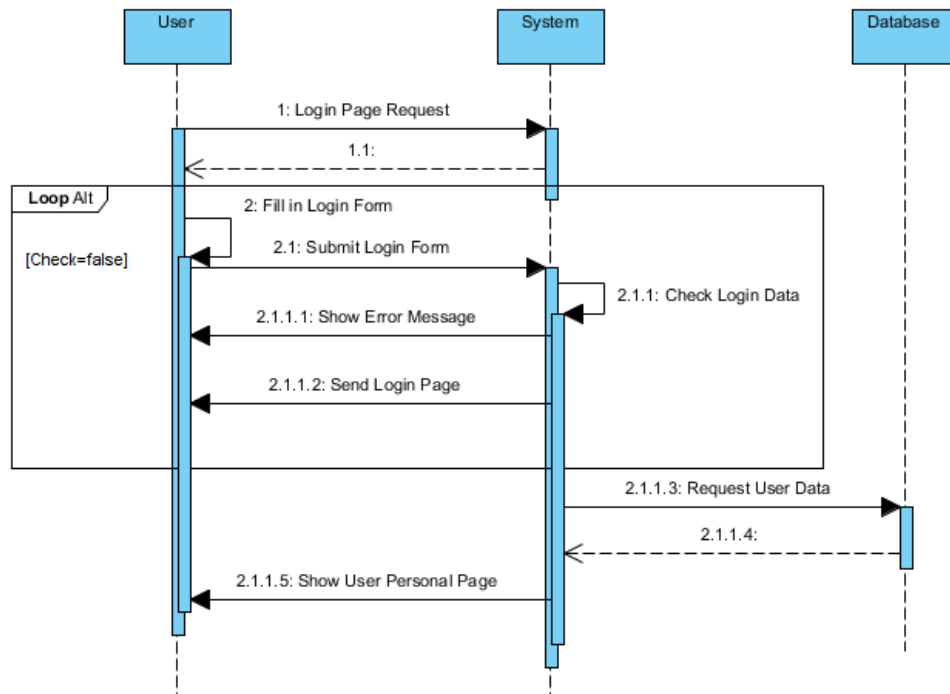


### 3.4.3 Sequence Diagrams

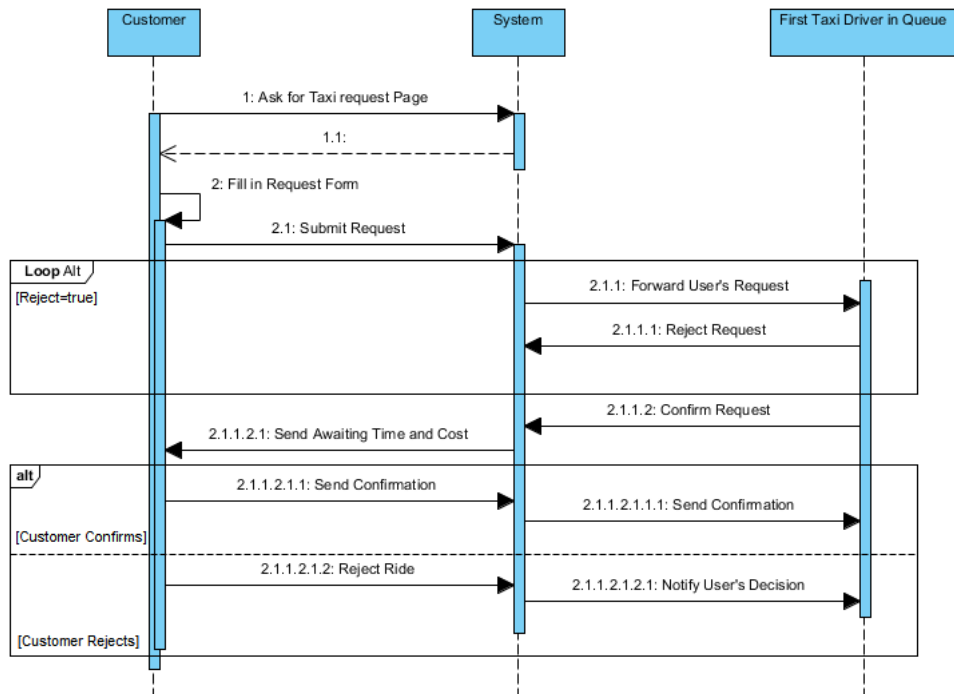
#### 3.4.3.1 Registration



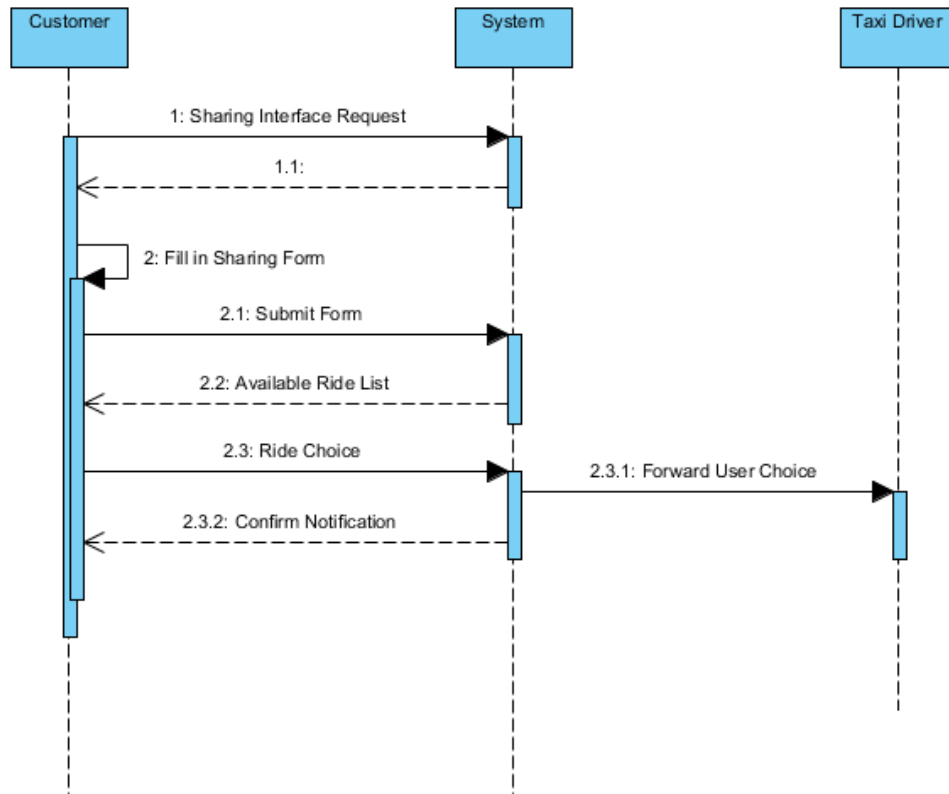
### 3.4.3.2 Login



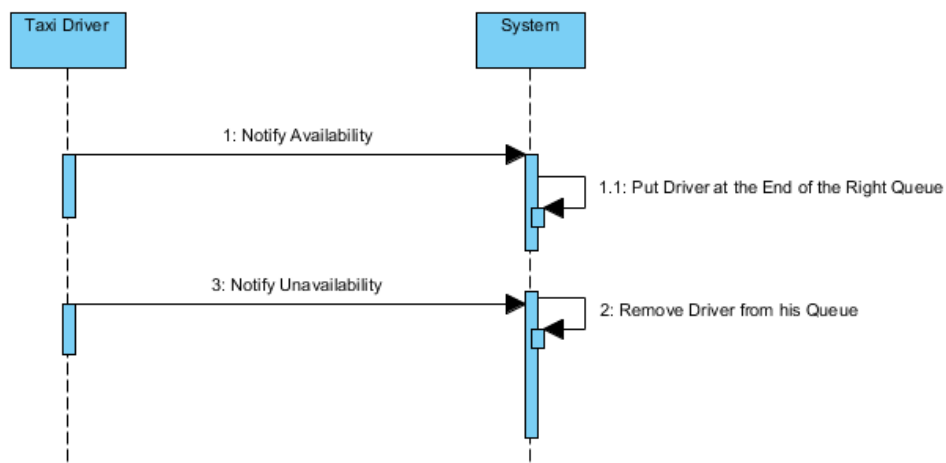
### 3.4.3.3 Taxi Request



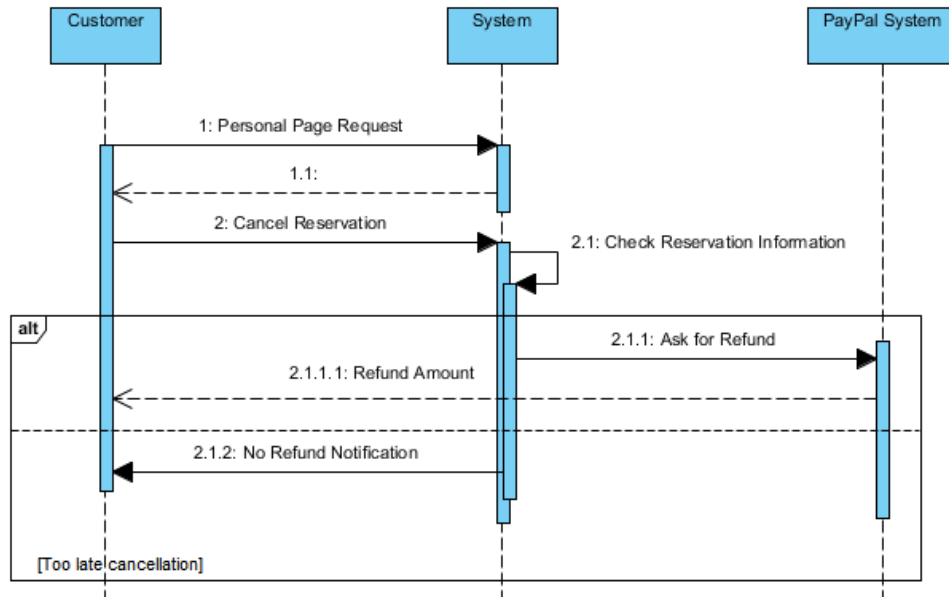
#### 3.4.3.4 Taxi Sharing



#### 3.4.3.5 Status Management

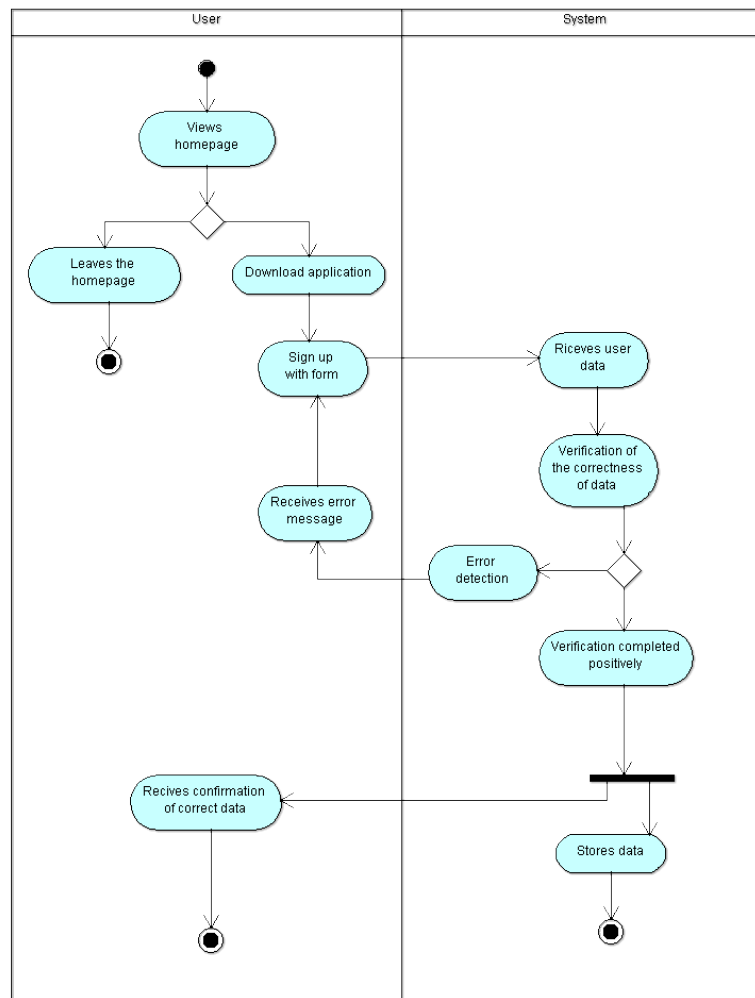


### 3.4.3.6 Delete Reservation

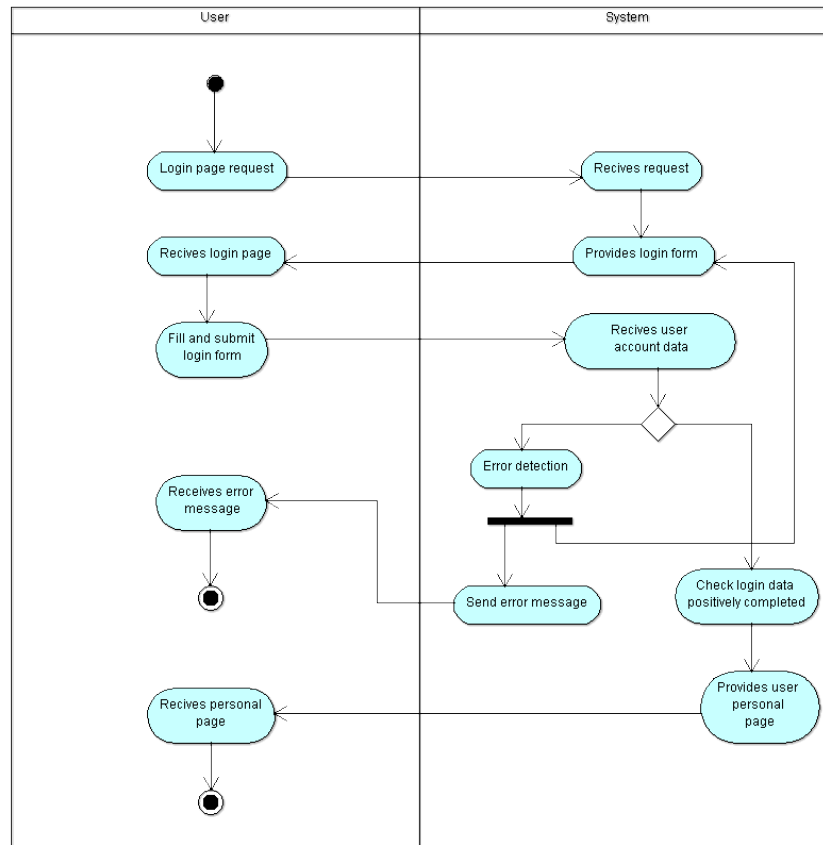


### 3.4.4 Activity Diagrams

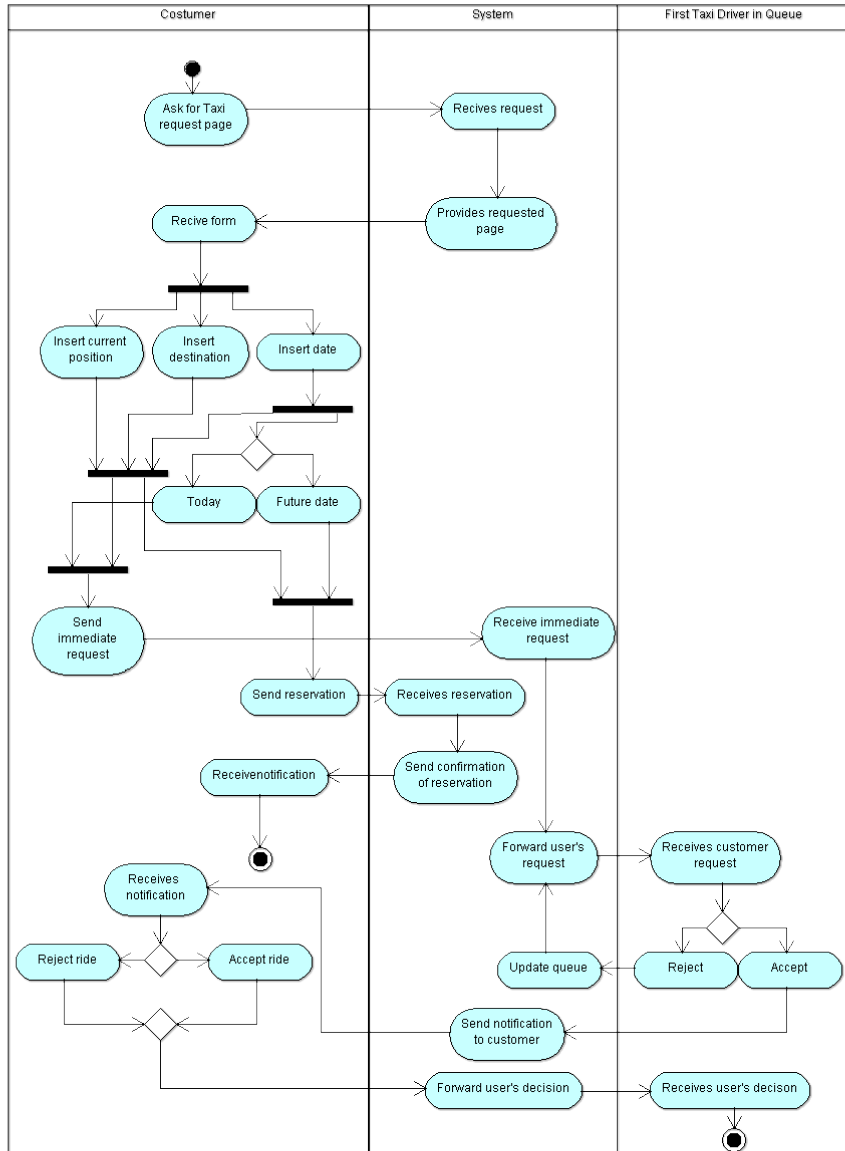
#### 3.4.4.1 Registration



### 3.4.4.2 Login

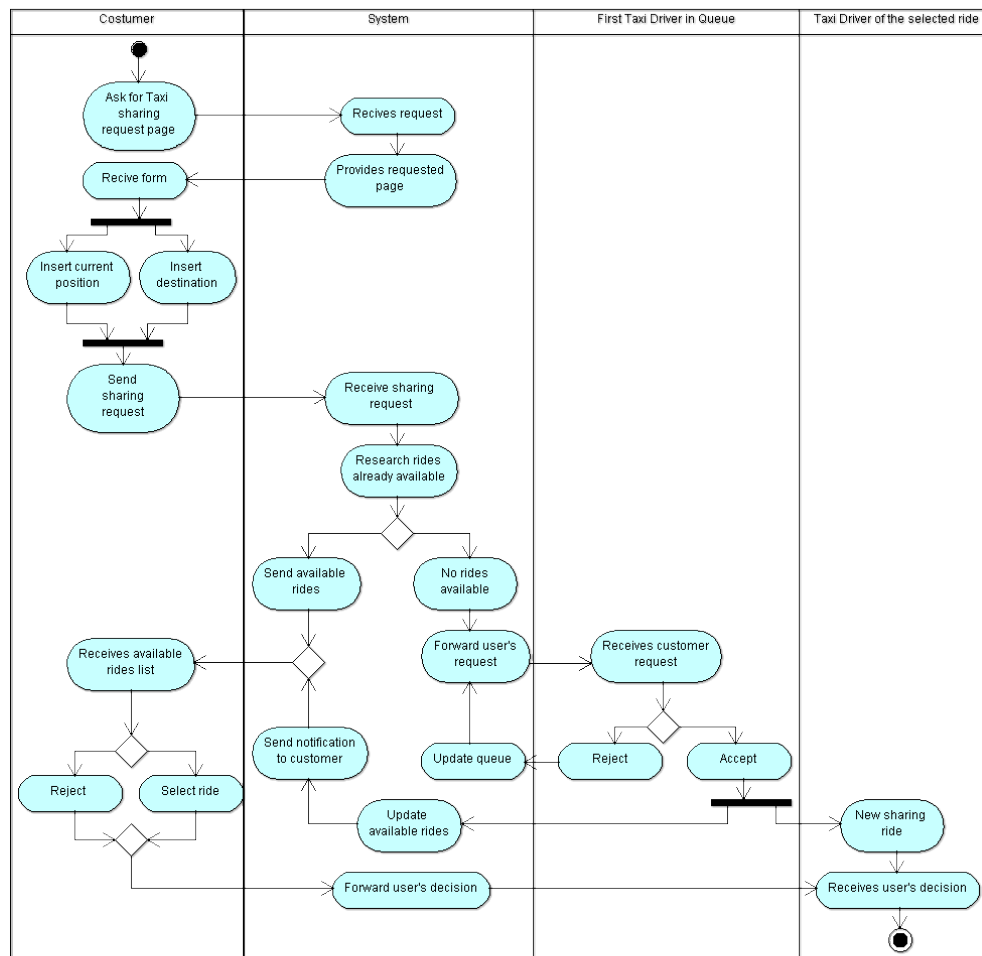


### 3.4.4.3 Taxi Request

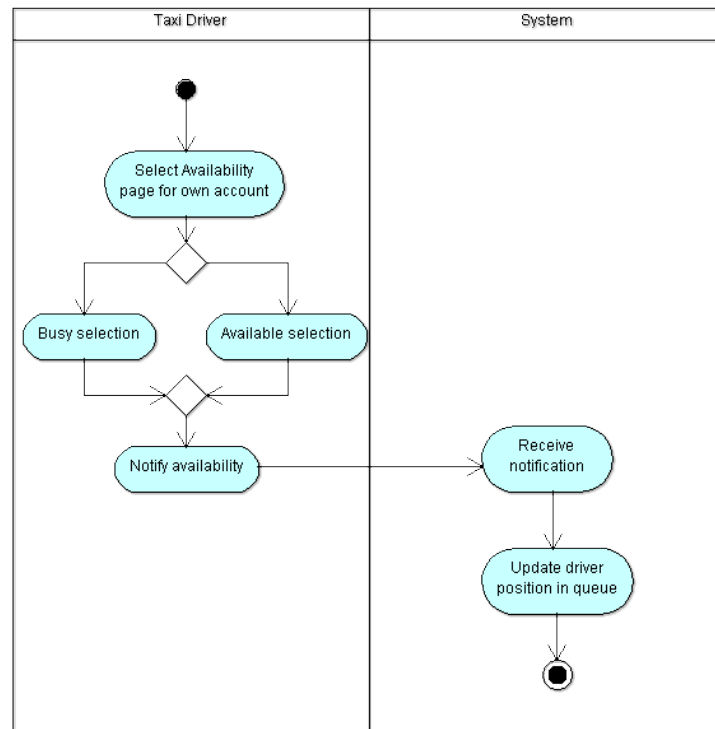




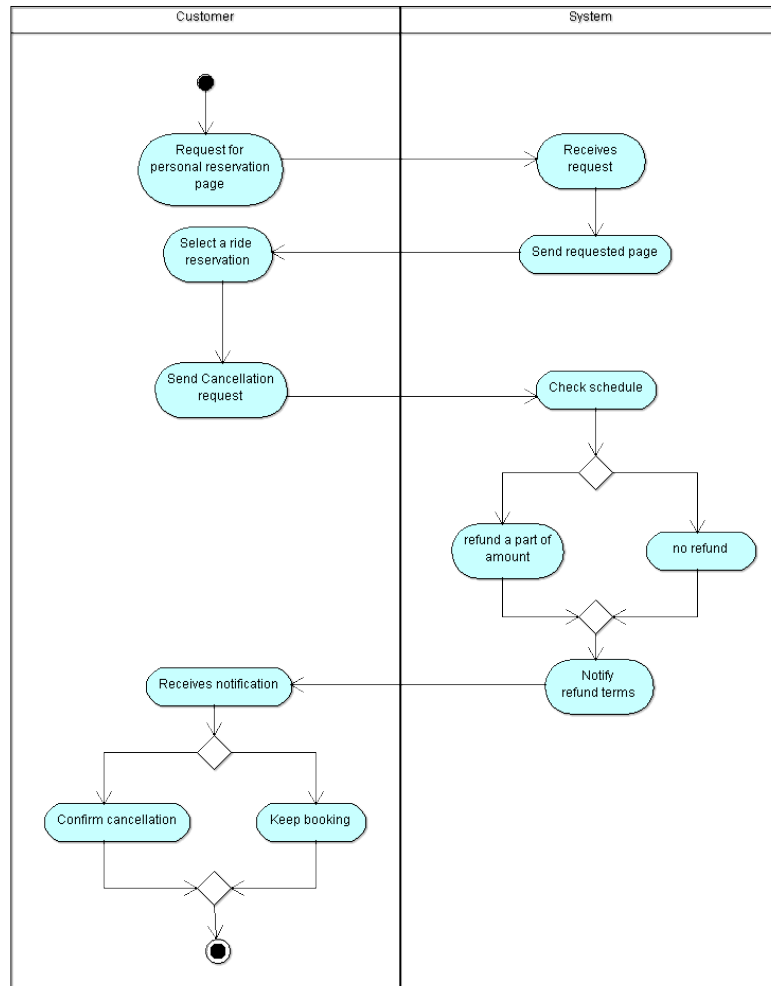
#### 3.4.4.4 Taxi Sharing



### 3.4.4.5 Status Management



### 3.4.4.6 Deleting Reservation



## 3.5 Performance Requirements

- 95% of the operations must respond within 5 seconds.
- The systems has to support 200 concurrent users.
- The system has to support the registration of 10.000 users' data.

## **3.6 Logical Database Requirements**

The system must store all the user account informations as well as the taxi ride records. For each user account, all the informations shall be stored on an external supporting SQL database.

## **3.7 Design Constraints**

Software Language: All coding will be done in Java (Swift for the iOS version). Fault Tolerance: Data should not become corrupted in case of system crash or power failure.

## **3.8 Software system attributes**

### **3.8.1 Reliability**

The system will not crash on invalid data or input. The System will check the validity of the user inputs when they are loaded. If there is an error the system will ask the user to repeat the operation. If an error occurs during the payments, the system displays an error message and stops the reservation. If there is a fatal error, the system shuts down without crashing the device it is running on.

### **3.8.2 Availability**

The system will allow the user to restart the application after a crash. The user will be able to use myTaxiService after the system restarts.

## **3.9 Security**

The system will use default operating system security on his device or computer. The system will check all the user inputs. If they are not in compliance with the expected input, the user will be compelled to repeat the procedure. In case only one of multiple inputs is wrong, the system right will save the right ones and allows the user not to insert them twice. All sensitive data will be encrypt before being saved and stored in the database. All of the security checks needed to ensure that no fraudulent transactions occur will be granted by the external payment service.

## **3.10 Maintainability**

All code is fully documented. Each function is commented with pre and post-conditions. All program files include comments and date of the last

change. Modularity has a main importance to permit future modifications.

### **3.11 Portability**

myTaxiService will be designed to run almost on every commercial platform which support the following browsers: Google Chrome, Safari, Firefox, Opera and Internet Explorer. The code will be written using Java for android devices or Swift for iOS devices. About the 70% of the code is host dependent.

## Chapter 4

# Appendix

### 4.1 Alloy

Here we are showing our Alloy model created using the Class Diagram.

#### 4.1.1 Signature

```
//-----SIGNATURE-----
abstract sig Boolean{}
one sig True extends Boolean{}
one sig False extends Boolean{}
sig Date{}
sig Location {}
sig NumberPlate{}
sig Customer {}

sig TaxiDriver {
  taxi : Vehicle,
  available : one Boolean
}

sig Reservation {
  customer : lone Customer,
  sharingCustomer : some Customer,
  driver : one TaxiDriver,
  date : one Date,
  start : one Location,
  end : one Location
}

one sig Queue {
  tail : one TaxiDriver,
  head : one TaxiDriver,
  inQueue : some TaxiDriver
}

sig Vehicle {
  numberPlate : one NumberPlate,
  maxPassengers : one Int,
}
//-----END SIGNATURE-----
```

### 4.1.2 Functions

```
//-----FUNCTIONS-----  
//This function gets all the taxi drivers who are available  
fun availableTaxiDrivers [] : set TaxiDriver {  
  { t : TaxiDriver | t.available = True}  
}  
//This function gets all the taxi drivers who are not available  
fun notAvailableTaxiDrivers [] : set TaxiDriver {  
  { t : TaxiDriver | t.available = False}  
}  
//-----END FUNCTIONS-----
```

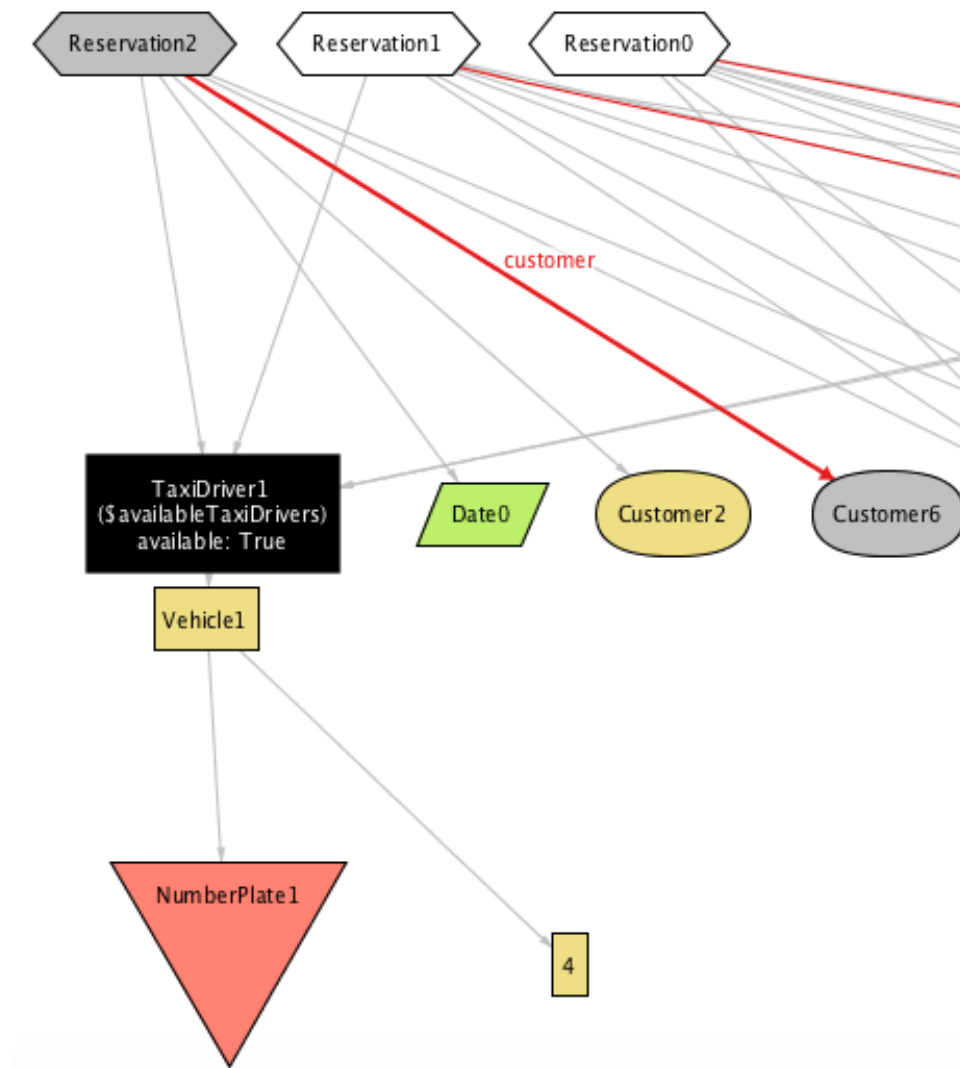
### 4.1.3 Facts

```
// -----FACTS-----  
//A customer can not be one of the sharing customers of the same ride  
fact ACustomerNotSharingCustomer{  
  no r : Reservation | r.customer in r.sharingCustomer  
}  
// A taxi driver can have only one reservation at time  
fact TaxiDriverCanHaveOnlyOneReservationAtTime{  
  all r1,r2 : Reservation | (r1.driver = r2.driver and r1.date = r2.date) => (r1 = r2)  
}  
//A customer can not have more then one reservation in the same moment  
fact onlyOneRideAtTimeForTheCustomer {  
  all r1, r2 : Reservation | (r1.customer = r2.customer and r1.date = r2.date) => (r1 = r2)  
}  
//A reservation can not have the starting point equals to the destination  
fact notCircularRide {  
  no r : Reservation | r.start = r.end  
}  
//A vehicle is related to a single number plate  
fact uniqueNumberPlate{  
  all v1,v2 : Vehicle | (v1.numberPlate = v2.numberPlate) => (v1=v2)  
}  
//Taxi Max passengers are from 1 to 6  
fact maxPassengersForTaxi {  
  all v : Vehicle | v.maxPassengers > 0 and v.maxPassengers < 7  
}  
//Each taxi can not transport more passengers than his max capacity  
fact limitOfPassengersForRide{  
  all r : Reservation | #r.sharingCustomer < r.driver.taxi.maxPassengers  
}  
//A taxi can not be shared by different taxi drivers  
fact oneTaxiForEachDriver{  
  all t1, t2 : TaxiDriver | (t1.taxi = t2.taxi) => (t1 = t2)  
}  
//It can not exist a taxi without driver  
fact noGhostTaxi{  
  #TaxiDriver = #Vehicle  
}  
//If available a taxi driver is into the queue  
fact availableTaxiIntoTheQueue {  
  all q : Queue, t : TaxiDriver | ((t.available = True) => (t in q.inQueue)) and ((q.tail.available = True) and (q.head.available = True))  
}  
//Tail and head of the queue are the same only if there is only one element in the queue  
fact tailAndHeadQueue {  
  all q : Queue | ((q.tail = q.head) <=> (#q.inQueue = 1))  
}  
//It can not exist a vehicle without number plate  
fact noUnnumberedVehicle{  
  #NumberPlate = #Vehicle  
}  
//-----END FACTS-----
```

#### 4.1.4 Generated World

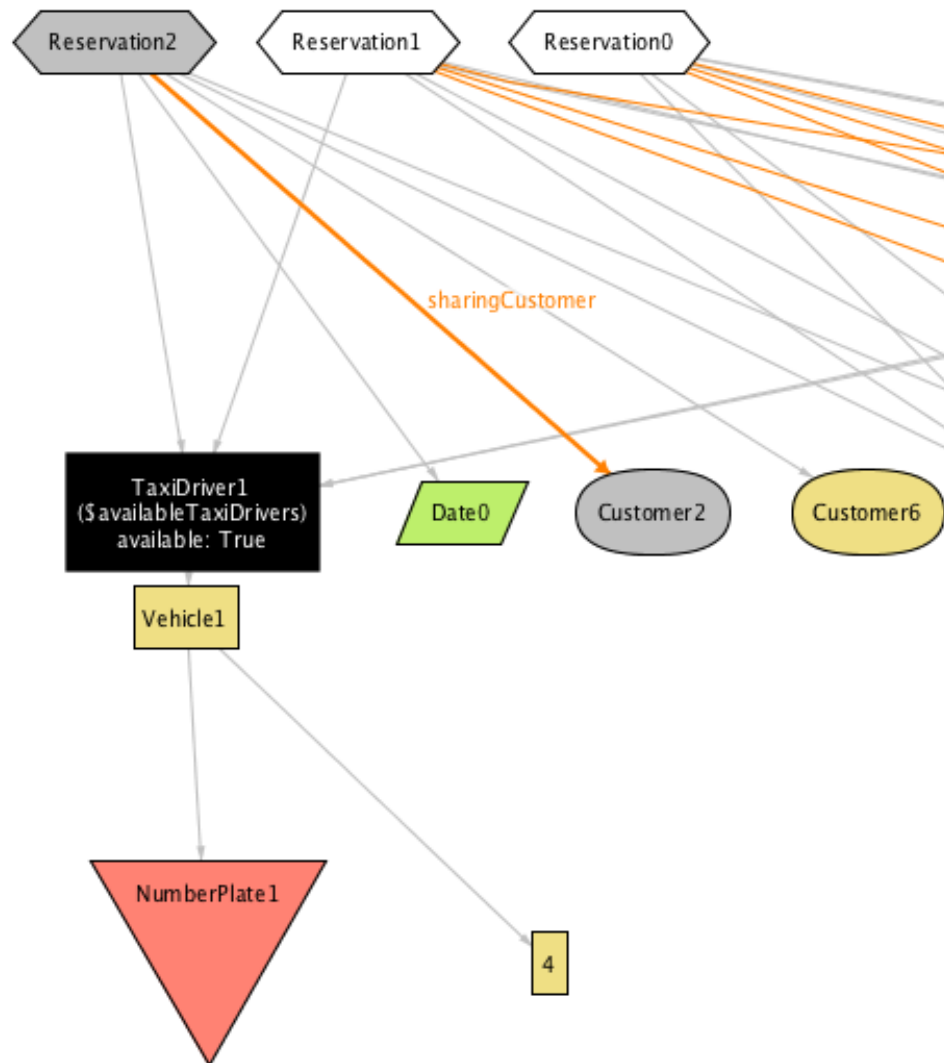
This is the world generated using Alloy system. It's possible to observe that:

- Each reservation is related only to a single customer

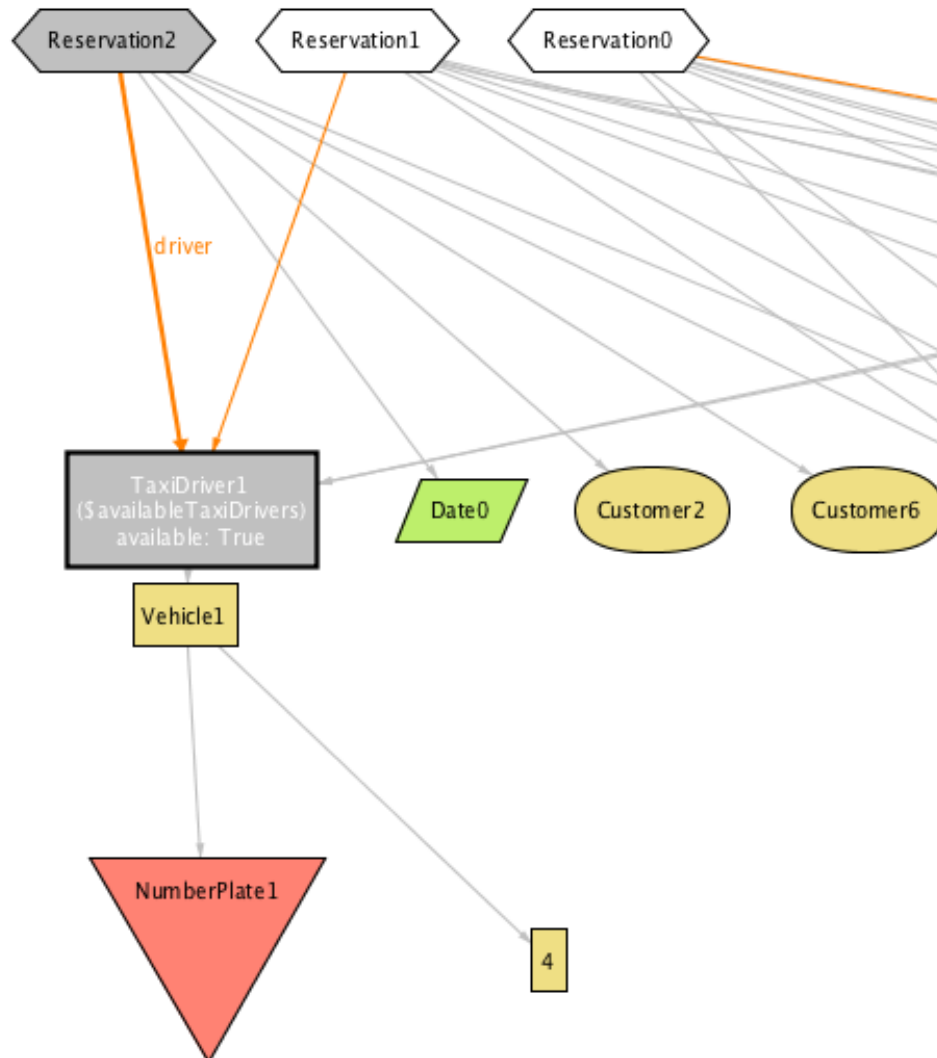




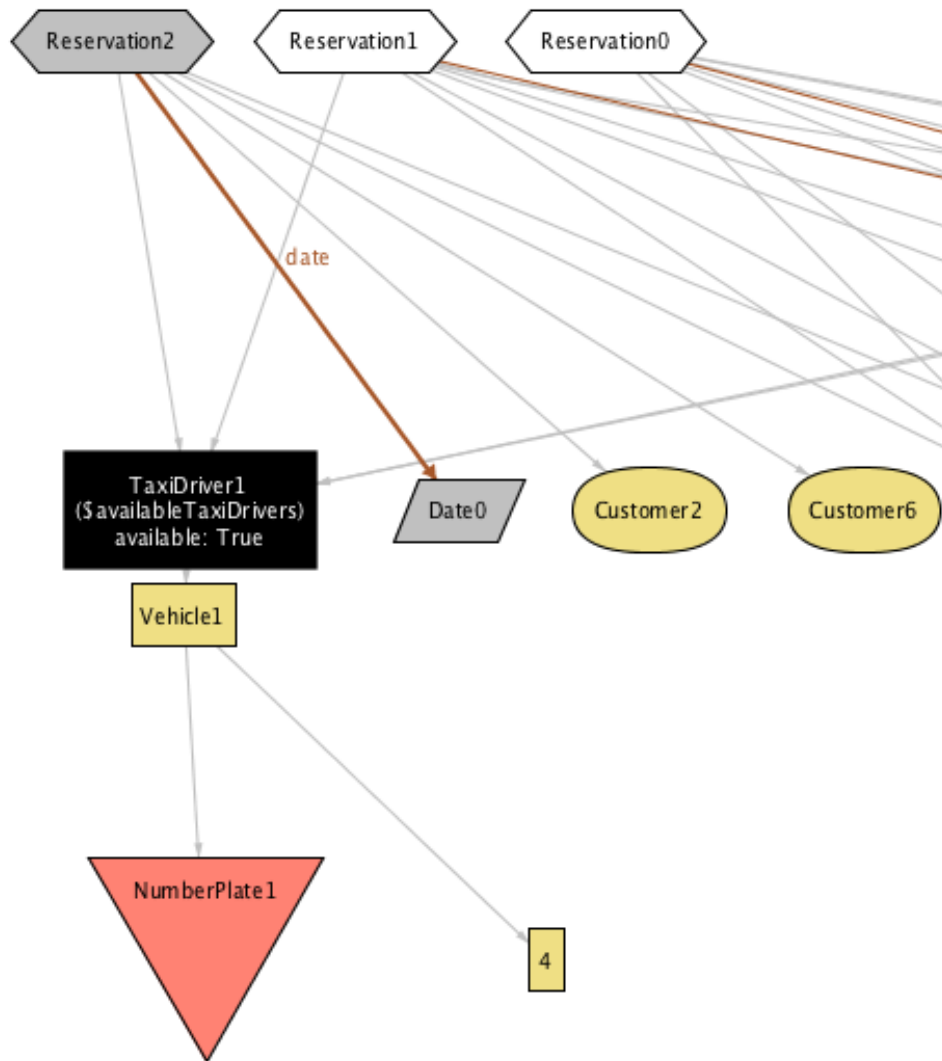
- Each reservation can also be related to other customers, who are those who are sharing the taxi ride with the first customer.



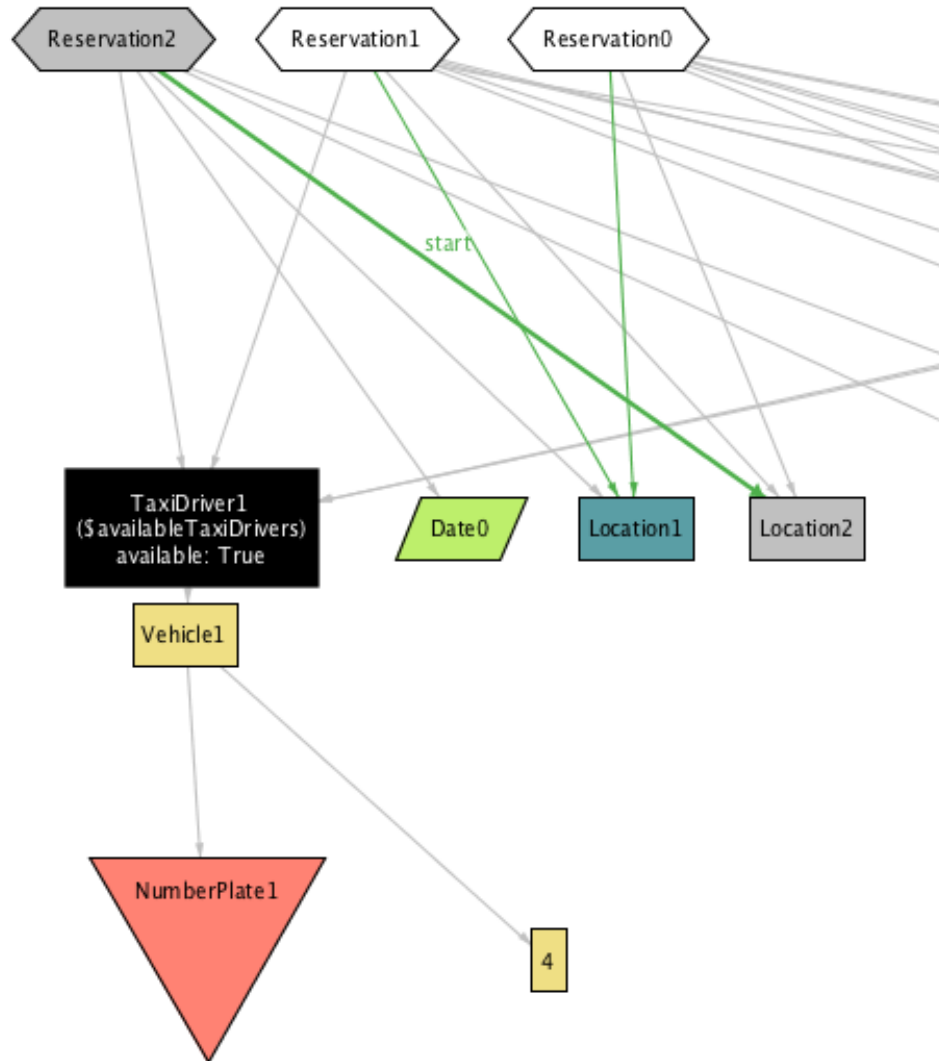
- Each reservation has a related taxi driver

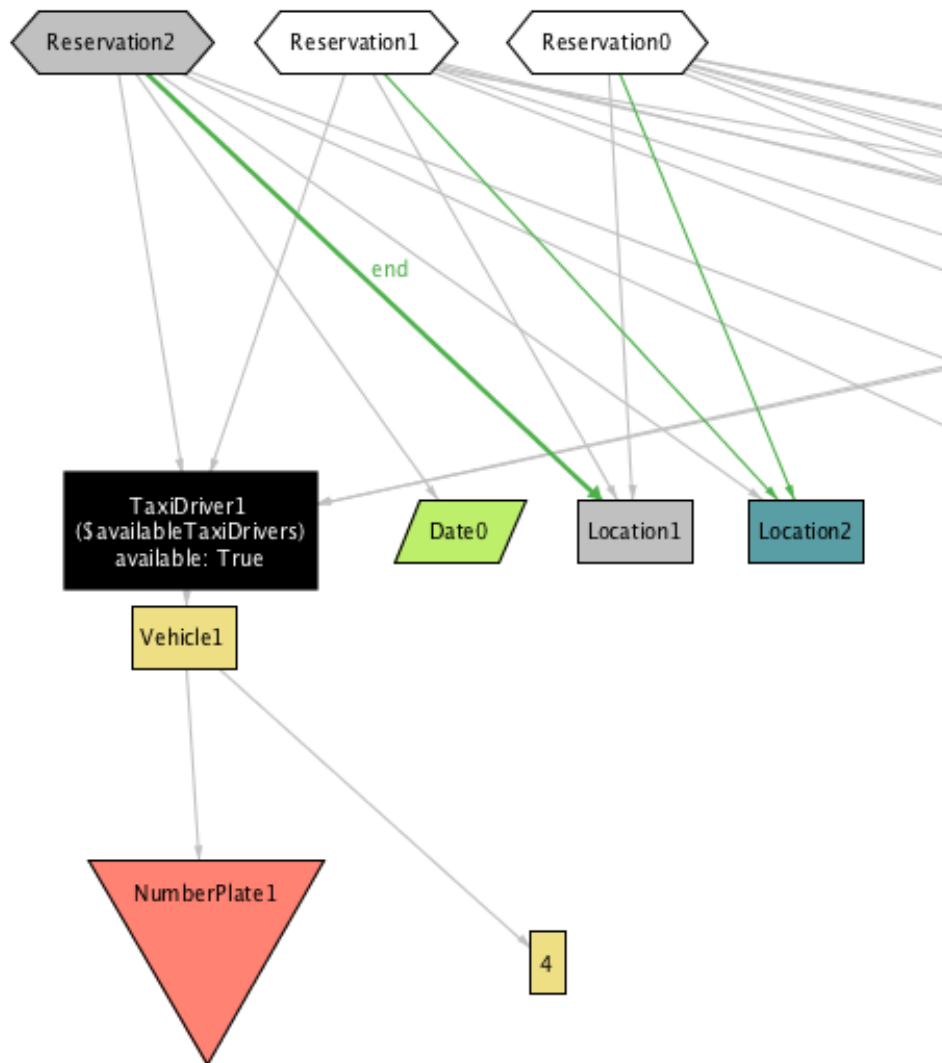


- Each reservation has a data which determines when the ride will take place

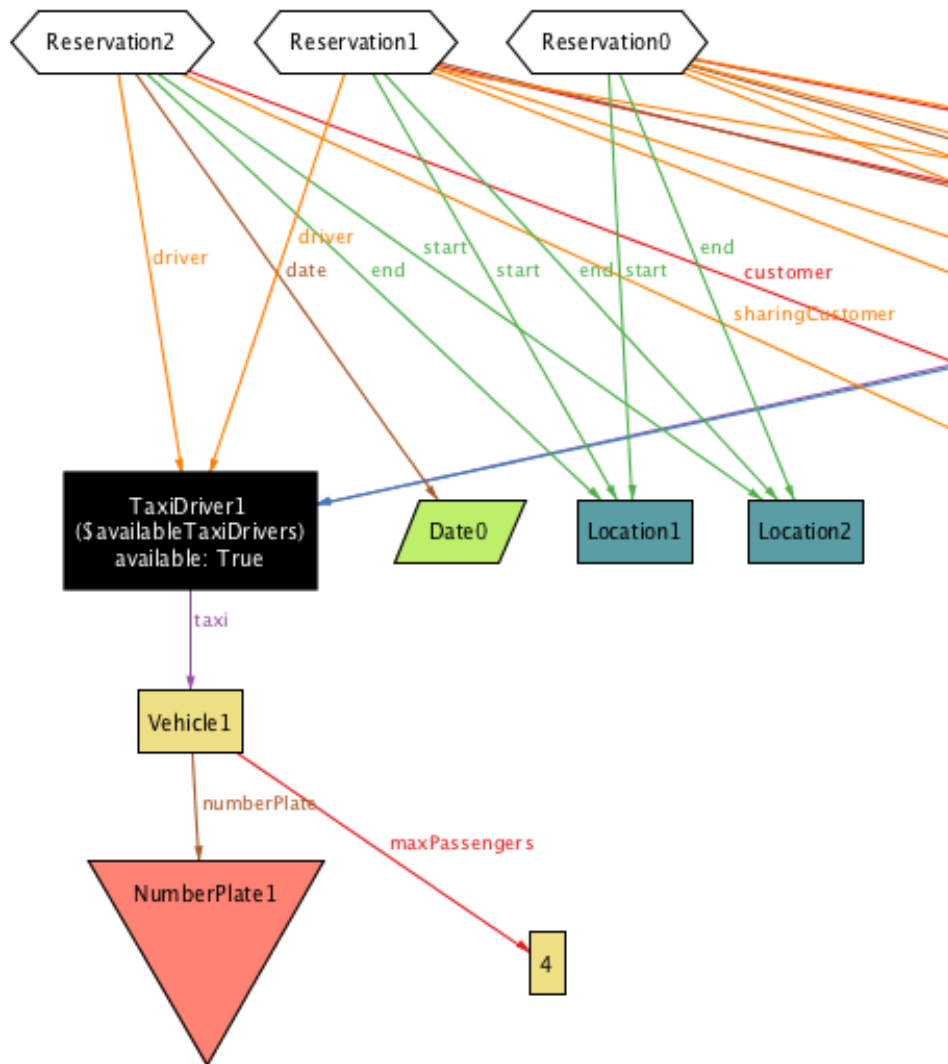


- Each ride has a starting point and a destination which can not be the same place.

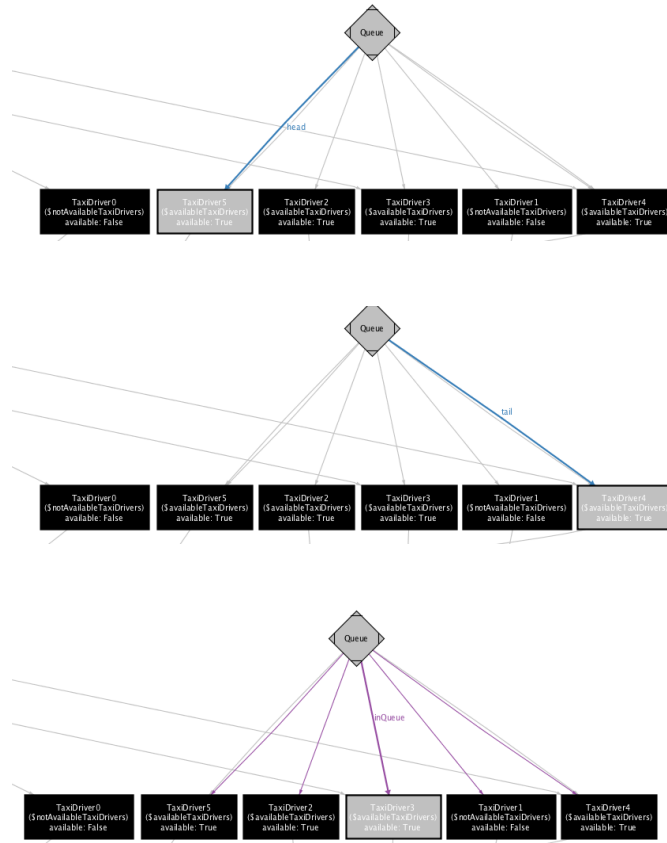




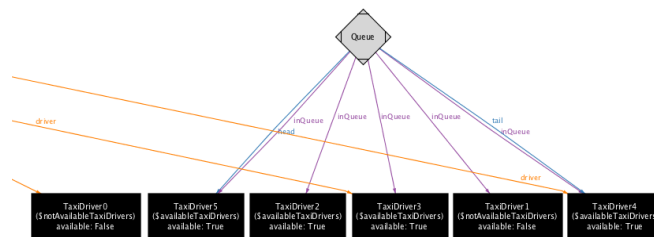
- Other relations between other classes, such as the taxi driver and his vehicle or the vehicle and its number plate



- The queue of the available taxi drivers has a tail, a head and a possible body.

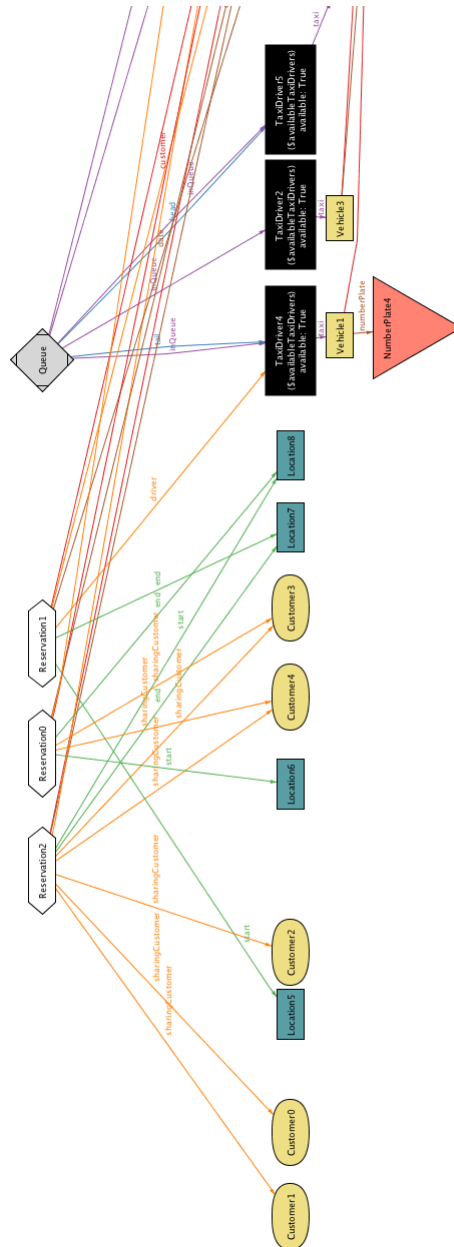


- And a complete vision of the queue.



#### 4.1.4.1 Globe

Finally, here it is a complete view of the generated world.





#### **4.1.5 Software Used**

- TeXstudio: to redact this document.
- Balsamiq Mockups 3: to create all the mockups.
- StarUML: to create Use Cases Diagrams.
- ArgoUML: to create Class Diagram and Activity Diagrams.
- Visual Paradigm Community Edition 12.2: to create Sequence Diagrams.
- Alloy Analyzer: to write and run Alloy code.
- Gimp: to manipulate some images.

### **4.2 Hours of Work**

In order to complete this document, each member of the group has spent about 20 hours.