



POLITECNICO DI MILANO  
SCUOLA DI INGEGNERIA INDUSTRIALE E  
DELL'INFORMAZIONE  
M.Sc. IN COMPUTER SCIENCE AND ENGINEERING

---

## myTaxiService

Requirements Analysis and Specification Document

---

*Authors:*

Angelo GALLARELLO

Edoardo LONGO

Giacomo LOCCI

December 2, 2015

Version 2.2

# Contents

<b>Contents</b>	<b>1</b>
<b>1 Introduction</b>	<b>4</b>
1.1 Purpose . . . . .	4
1.2 Scope . . . . .	4
1.2.1 Goals . . . . .	4
1.2.2 Applications . . . . .	4
1.3 Definitions . . . . .	5
1.4 References . . . . .	6
1.5 Document Overview . . . . .	7
<b>2 Overall Description</b>	<b>7</b>
2.1 Product Perspective . . . . .	7
2.1.1 Hardware Interfaces . . . . .	8
2.1.2 Software Interfaces . . . . .	8
2.1.3 Communication Interfaces . . . . .	10
2.1.4 Memory Constrains . . . . .	10
2.1.5 Site Adaptation Requirements . . . . .	10
2.2 Product Functions . . . . .	10
2.2.1 Web Application and Mobile Application ( <i>User</i> ) . . . . .	10
2.2.2 Mobile Application ( <i>Taxi Driver</i> ) . . . . .	11
2.2.3 Back-End Application . . . . .	11
2.3 User Characteristics . . . . .	11
2.4 Constraints . . . . .	12
2.4.1 Regulatory Policies . . . . .	12
2.4.2 Hardware Limitations . . . . .	12
2.4.3 Interfaces to other applications . . . . .	12
2.4.4 Parallel Operations . . . . .	13
2.4.5 Reliability Requirements . . . . .	13
2.4.6 Safety and Security . . . . .	13
2.5 Assumptions And Dependencies . . . . .	14
2.5.1 Domain Assumptions . . . . .	14
2.5.2 Taxi Driver Assumptions . . . . .	14
2.5.3 User Assumptions . . . . .	14
2.5.4 Taxi Driver Assumptions . . . . .	14

<b>3</b>	<b>Specific Requirements</b>	<b>15</b>
3.1	User Interface . . . . .	15
3.1.1	Mobile Application ( <i>User</i> ) . . . . .	15
3.1.2	Web Application . . . . .	18
3.1.3	Mobile Application ( <i>Taxi Driver</i> ) . . . . .	22
3.2	Scenarios . . . . .	24
3.2.1	User Sign-up . . . . .	25
3.2.2	User Log in . . . . .	27
3.2.3	Modify Data . . . . .	29
3.2.4	Taxi Request . . . . .	31
3.2.5	Taxi Driver Registration . . . . .	33
3.2.6	Delete Request . . . . .	35
3.2.7	Report Problem . . . . .	37
3.2.8	Ride Sharing . . . . .	39
3.3	Class Diagram . . . . .	40
3.4	State Charts . . . . .	41
3.5	Functional Requirements . . . . .	45
3.5.1	Web Application . . . . .	45
3.5.2	Mobile Application ( <i>User</i> ) . . . . .	45
3.5.3	Mobile Application ( <i>Taxi Driver</i> ) . . . . .	46
3.5.4	Back-End Application . . . . .	47
3.6	Alloy . . . . .	49
3.6.1	Alloy code . . . . .	50
3.6.2	Alloy world considerations . . . . .	55
	<b>Appendices</b>	<b>57</b>
<b>A</b>	<b>Tools</b>	<b>57</b>
<b>B</b>	<b>Hours of work</b>	<b>57</b>

## Abstract

The current document represents the *Requirement Analysis and Specification Document* (RASD) of *myTaxiService* system. This document is intended to be read and used by everyone involved with the development of the software (i.e. developers, testers, project managers) and the delegating institution (i.e. the government of the large city). Therefore its purpose is to provide the intended reader with a fulfilling guideline, that can be used as the main resource for the development of the system, and with a presentation of the challenges that will have to be addressed throughout the whole process.

An incredible amount of effort has been put into every section of this document; every aspect of the *system-to-be* has been thoroughly explored, meticulously analyzed and accurately studied so that every situation can be addressed in a unique way and can be interpreted without ambiguity.

In the following sections the system is broken down into its components, examining its functional and non-functional requirements. An in depth analysis of the relationship between the system and the external world is conducted throughout the document, stressing the assumptions that have been taken and highlighting the constraints that every requirement has to satisfy in order to have a functional and useful system.

The reader of this document will find UML class diagrams, sequence diagrams and use cases that provide all the needed information required to build the system. Furthermore, every use case is justified by the goals and assumptions that lead to the need of a specific use case.

Moreover, the delegating institution will find a reason for every design choice taken and therefore is able understand the necessity of a specific feature.

This way, each and every reader will have a clear understanding of the *system-to-be*, by finding a justification for every decision taken concerning any aspect that is worth of considering for a successful development of the investigated system.

# 1 Introduction

This section introduces the *system-to-be*, and provides a high level description of its functioning, together with the main functionalities that the system will offer.

## 1.1 Purpose

The system is a software named *myTaxiService*. It is a taxi service that will operate in a big city; the main purpose is to simplify the access of passengers to the service and to guarantee a fair management of the taxi queues.

## 1.2 Scope

### 1.2.1 Goals

The aim of *myTaxiService* presented in the RASD is to:

**[G1] Intuitive service**

Make the process of requesting a service intuitive for the majority of potential users.

**[G2] Fast**

Lower the time required from a user to request or book a taxi. Request time is less than 30 seconds, booking time is less than 1,5 minutes.

**[G3] Availability**

Enable the user to take advantage of the taxi service from a wide range of locations, namely from every location where the user has access to an Internet enabled device that can run the application.

**[G4] Fair queues**

Guarantee a fair management of the taxi queue by minimizing the idle time of every taxi driver using the service.

### 1.2.2 Applications

To accommodate these goals the following pieces of software are to be developed:

**Mobile Application (*User*)** The mobile application is available for all major mobile OS (Android, iOS, Windows Mobile, Blackberry). This one is specifically addressed towards the *User*, giving him a broad range of actions to perform. It gives the *User* the opportunity to manage his profile, request a taxi and make or cancel a reservation.

**Web Application** The web application is thought as an alternate way, for the *User* to manage his profile, request a taxi and make or cancel a reservation.

**Mobile Application (*Taxi Driver*)** The mobile application is available for all major mobile OS (Android, iOS, Windows Mobile, Blackberry). This one is specifically addressed towards the *Taxi Driver*. It enables the *Taxi Driver* to accept a request issued by the system or dismiss it, signal any problem to the *Support Operator*.

**Back-End Application** The back-end is the application that manages the request system and queue system. This also provides a control panel used by the *Support Operator* to have an overall view of the whole system, and enables him to issue warnings to *Users* or *Taxi Drivers*, and to respond to warnings issued by the *Taxi Driver*.

## 1.3 Definitions

Here is a list of the words used in this document:

### **User**

He/She is the end user of the service that, once subscribed, makes use of the taxi service as a passenger.

### **Taxi Driver**

He/She is another kind of customer of the service. He/She is the one that, once subscribed, will be commissioned with a work.

### **Support Operator**

He/She is the person that is in charge of the customer service and interfaces with the customers, both the taxi drivers and the users.

### **Technician**

He/She is responsible for the maintenance.

**Taxi Zone**

Area of the city of  $2\text{km}^2$  .

**Admissible Paths**

Two paths are admissible if and only if they have the same starting zone and the destination of the shortest one is at most *one zone distant* from the longest path.

**Immediate Request**

An immediate request is a request issued by the User in order to request a ride on the spot.

**Reservation Request**

A reservation request is a request issued by the User in order to request a reservation for a ride that will happen in the at least two hours in the future.

**Pending Request**

A pending request is a request issued by the User that has received neither a confirmation nor a refusal.

**Active Ride**

An active ride is a ride that had started (the taxi had departed) but not has not yet finished (the taxi has not arrived at the destination).

**Single Ride**

A single ride is a ride that involves only one *User*.

**Shared Ride**

A shared ride is a ride that involves only more than one *User*.

**1.4 References**

- IEEE Std 830-1998: “IEEE Recommended Practice for Software Requirements Specifications”
- Project description: “Software Engineering 2 Project, AA 2015/2016”
- Alloy Language Reference: <http://alloy.mit.edu/alloy/documentation/book-chapters/alloy-language-reference.pdf>

- UML Language Reference: [https://www.utdallas.edu/~chung/Fujitsu/UML\\_2.0/Rumbaugh--UML\\_2.0\\_Reference\\_CD.pdf](https://www.utdallas.edu/~chung/Fujitsu/UML_2.0/Rumbaugh--UML_2.0_Reference_CD.pdf)

## 1.5 Document Overview

This document provides a detailed description of the system and it is structured in three main sections:

### Introduction

This section gives a high level description of the main topics covered throughout the whole document, that is the system purpose and its scope.

### Overall Description

This section offers the reader an overview over the general factors and functions that affect the product and its requirements.

### Specific Requirements

This offers an insight over every functional and non functional requirement.

## 2 Overall Description

This section highlights all the main functionalities and constraints of the system-to-be. Some of these functionalities and interfaces require a more detailed description that is provided in the *Specific Requirements* section on page 15.

### 2.1 Product Perspective

As stated in the *Scope* section the product we will release is composed by four main software applications.

- A **Web Application** addressed to the *Users* to use our service. This application must interface mainly with the **Back-End Application** and with Google's Maps API.
- A **Mobile Application (*User*)** addressed to the *Users* to use our service and available for Android, iOS, Windows Mobile and Blackberry. This application must interface mainly with our **Back-End Application** and with Google's Maps API



- A **Mobile Application** (*Taxi Driver*) addressed to the *Taxi Drivers* to use our service and available for Android, iOS. This application must interface mainly with our **Back-End Application** and with Google's Maps API
- A **Back-End Application** that will handle all the business logic and that must interface mainly with Google's Maps API and with a MySQL database .

### 2.1.1 Hardware Interfaces

Both the **Mobile Application** (*User*) and the **Back-End Application** must interface with the GPS module and with the Network module.

### 2.1.2 Software Interfaces

#### Web Application

- MyTaxyService API
  - Mnemonic : Back-end
- Google Maps API
  - Mnemonic : Google Maps API
  - Version Number : V3
  - Source : <https://developers.google.com/maps/documentation/javascript/>

#### Mobile Application (*User*) and Mobile Application (*Taxi Driver*)

- MyTaxyService API
  - Mnemonic : Back-end
- Google Maps API
  - Mnemonic : Google Maps API
  - Version Number : V3
  - Source : <https://developers.google.com/maps/>
- Android SDK

- Mnemonic : Android
  - Version Number : 6.0
  - Source : <http://developer.android.com/sdk/index.html>
- iOS SDK
  - Mnemonic : iOS
  - Version Number : 9.2
  - Source : <https://developer.apple.com/ios/download/>
- Windows Mobile SDK
  - Mnemonic : Windows Mobile
  - Version Number : 6.5
  - Source : <http://www.microsoft.com/en-us/download/details.aspx?id=17284>
- BlackBerry SDK
  - Mnemonic : BlackBerry
  - Version Number : 10
  - Source : <https://developer.blackberry.com/>

## **Back-End Application**

- MySQL API
  - Mnemonic : MySQL
  - Version : 5.0
  - Source : <https://dev.mysql.com/doc/refman/5.0/en/c-api.html>
- Java EE API
  - Mnemonic : Java
  - Version : 6
  - Source : <http://docs.oracle.com/javaee/6/api/>
- Google Maps API
  - Mnemonic : Google Maps API

- Version Number : V3
- Source : <https://developers.google.com/maps/documentation/javascript/>

### 2.1.3 Communication Interfaces

#### Web Application

Every application must interface with the Internet network. This interface is handled by the operative systems and not by the applications themselves.

### 2.1.4 Memory Constrains

**Web Application and Mobile Application (*User*) and Mobile Application (*Taxi Driver*)** The Mobile and Web Applications can not exceed 75MB of RAM usage.

**Back-End Application** The Back-End application can not exceed 150GB of total RAM usage.

### 2.1.5 Site Adaptation Requirements

#### Software Adaptation

Every Mobile Application must be developed according to the platforms design guidelines <sup>1 2</sup>.

## 2.2 Product Functions

This section highlights the main product functions sorted by application. Functions are here described in high level language to give a general overview of the products.

Detailed requirements are listed in section 3.5 on page 45.

### 2.2.1 Web Application and Mobile Application (*User*)

[F1] The app must allow the user to register and login

[F2] The app must allow the user to book a taxi on the spot.

[F3] The app must allow the user to reserve a taxi ride

---

<sup>1</sup><http://developer.android.com/design/index.html>

<sup>2</sup><https://developer.apple.com/design/>

- [F4] The app must allow the user to share his ride with other people with the same starting point and destination direction .

### 2.2.2 Mobile Application (*Taxi Driver*)

- [F5] The app must allow the taxi driver to login
- [F6] The app must provide position updates to the Back-End Application
- [F7] The app must allow the taxi driver to handle ride requests
- [F8] The app must allow the taxi driver to report issues that could delay his arrival (traffic jam, car faults, etc ...)
- [F9] The app must allow the taxi driver to overview a history of his last rides.
- [F10] The app must allow the taxi driver to report an absent user.

### 2.2.3 Back-End Application

- [F11] The application must provide an interface for the registration process of users and taxi drivers
- [F12] The application must provide an interface for the data modification process of the users
- [F13] The application must implement a password recovery system
- [F14] The application must handle the request queue using a fair <sup>3</sup> policy
- [F15] The application must provide an accurate estimate of the total cost and time of a ride
- [F16] The application must provide an interface to allow an operator to perform all basics CRUD operations
- [F17] The application must implement an algorithm to check if two users can share a ride

## 2.3 User Characteristics

This section highlights the main characteristics of the actors defined in section 1.3 on page 5.

---

<sup>3</sup>For clarification see requirements in the subsection 3.5.4 on page 47

**User** The user has to be at least 14 years old and own a valid ID.  
The user has to be familiar with standard mobile applications interfaces.

**Taxi Driver** The driver has to be a regularly licensed taxi driver.  
The driver has to be familiar with standard mobile applications interfaces.

**Technician** The technician must be able to debug and identify issues in a full stack environment. The technician must be familiar with the application requirements and the overall infrastructure and design.

**Support Operator** The support operator must be familiar with the overall application design and has to know how to resolve common user's issues.

## 2.4 Constraints

### 2.4.1 Regulatory Policies

- **Privacy Policy** Data should be collected and stored following the privacy policy guidelines provided by Mozilla Foundation [https://developer.mozilla.org/en-US/Marketplace/Publishing/Policies\\_and\\_Guidelines/Privacy\\_policies](https://developer.mozilla.org/en-US/Marketplace/Publishing/Policies_and_Guidelines/Privacy_policies)
- **Taxi Driver Policy** The client application for taxi drivers must be accessible only by registered and regularly licensed taxi driver.

### 2.4.2 Hardware Limitations

**Mobile Application (User)** The Mobile Application (User) must be developed in order to be available for the 80% of the devices currently on the market for each different platform.

**Web Application (User)** The Web Application must be developed in order to be available for the 100% of devices that support the latest version of most used browser <sup>4</sup>.

### 2.4.3 Interfaces to other applications

**Mobile Application (User)** The developer of the **Mobile Application (User)** must follow the design guideline <sup>5</sup> <sup>6</sup> of the platform and has to

---

<sup>4</sup>[http://www.w3schools.com/browsers/browsers\\_stats.asp](http://www.w3schools.com/browsers/browsers_stats.asp)

<sup>5</sup><http://developer.android.com/design/index.html>

<sup>6</sup><https://developer.apple.com/design/>

interface with the respective Developer Console and make sure that the app will be accepted and published.

The application must interface with Google's Maps API <sup>7</sup>.

**Web Application** The application must interface with Google's Maps API.

**Mobile Application (*Taxi Driver*)** The application must interface with Google's Maps API.

**Back-End Application** The application must interface with Google's Maps API.

#### 2.4.4 Parallel Operations

**Back-End Application** The Back-End Application must handle parallel request from multiple users without generating conflicts or inconsistency.

The system must decline an Immediate Request from a user if it finds another pending Immediate Request from the same user or if the user is involved in an Active Ride.

#### 2.4.5 Reliability Requirements

**Back-End Application** This requirement of reliability and availability refers to the total time the system is available for the applications of the end users.

The system must reach a 99.99% uptime (corresponding to a inactivity time of 50 minutes/year).

#### 2.4.6 Safety and Security

**Mobile Application (*Taxi Driver*)** The application must include an automatic night mode and gallery mode in order not to compromise the driver security.

All the network connections must be encrypted.

---

<sup>7</sup>See section 2.1.2 on page 8

## 2.5 Assumptions And Dependencies

### 2.5.1 Domain Assumptions

We assume that the system will be deployed in a city of about 1.5M inhabitants with about 4800 regular licensed taxi drivers.

We assume that at launch time the *Mobile Application (User)* will reach an audience of about 100k users while the *Web Application* will reach an audience of about 10k users.

We assume that the city is divided in *Taxi Zone*.

### 2.5.2 Taxi Driver Assumptions

We assume that every taxi driver will have a compatible device <sup>8</sup> (personal or provided by the project client) with a working GPS module.

We assume that our system and the *Support Operator* have access to an internal database of valid taxi licenses of the city.

We assume that a taxi can have at most 6 passengers.

We assume that a taxi ride always ends and that exceptional problems are solved outside the system.

We assume that every taxi driver has only one car and that every car belongs only to one taxi driver.

### 2.5.3 User Assumptions

We assume that every user will have a device with an enabled Internet connection.

We also assume that the location provided by the user (manually or automatically via GPS) is always correct.

### 2.5.4 Taxi Driver Assumptions

We assume that the taxi driver has to send his personal information (license number, name, surname, C.F.) via email to the support address to get valid login credentials.

We assume that every taxi driver handles all his rides with our application.

---

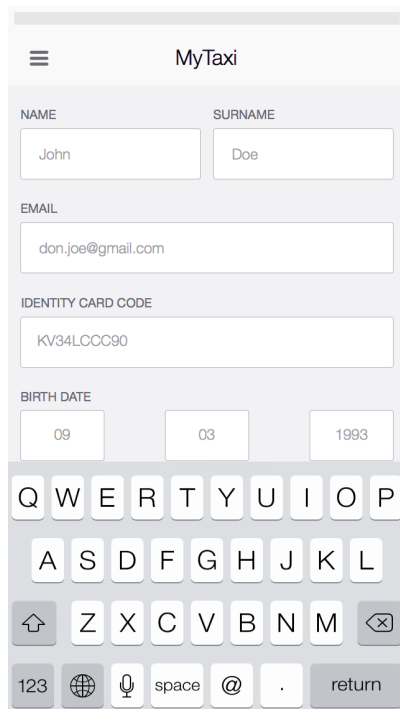
<sup>8</sup>See section 2.4 on page 12

## 3 Specific Requirements

### 3.1 User Interface

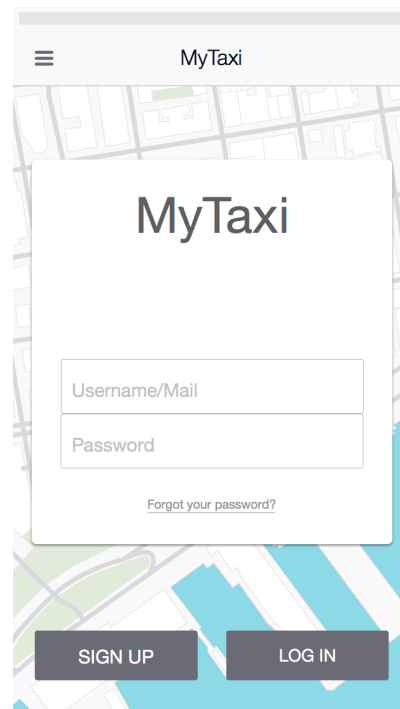
In this section are shown the first mock-ups of the applications of the *system-to-be*.

#### 3.1.1 Mobile Application (*User*)



The Register screen features a header with a hamburger menu icon and the title "MyTaxi". Below the header, there are four input fields: "NAME" (containing "John"), "SURNAME" (containing "Doe"), "EMAIL" (containing "don.joe@gmail.com"), and "IDENTITY CARD CODE" (containing "KV34LCCC90"). Below these fields is a "BIRTH DATE" section with three input boxes for day, month, and year, containing "09", "03", and "1993" respectively. At the bottom of the screen is a full QWERTY keyboard.

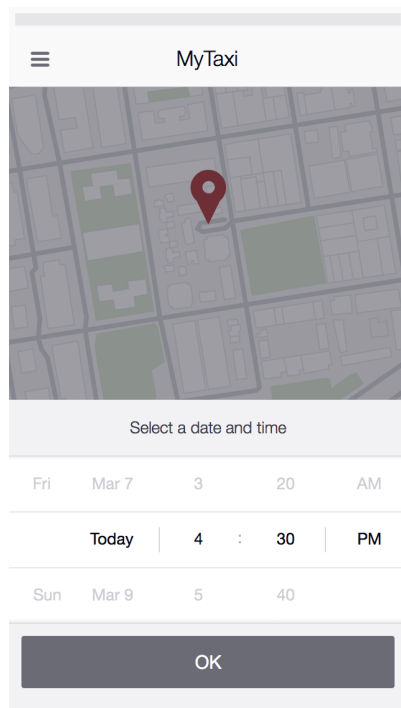
(a) Register



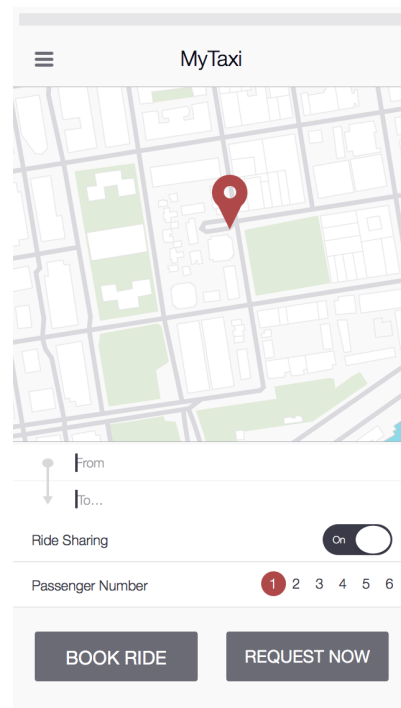
The Log In screen features a header with a hamburger menu icon and the title "MyTaxi". Below the header is a large white card with the "MyTaxi" logo at the top. Inside the card are two input fields: "Username/Mail" and "Password". Below the "Password" field is a link that says "Forgot your password?". At the bottom of the screen are two buttons: "SIGN UP" and "LOG IN". The background of the screen shows a stylized map.

(b) Log In

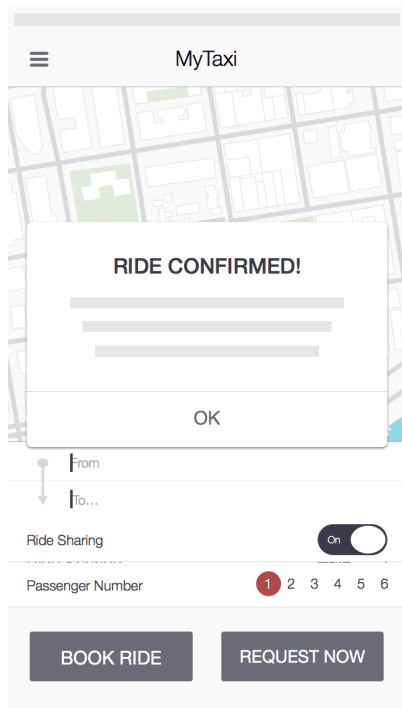




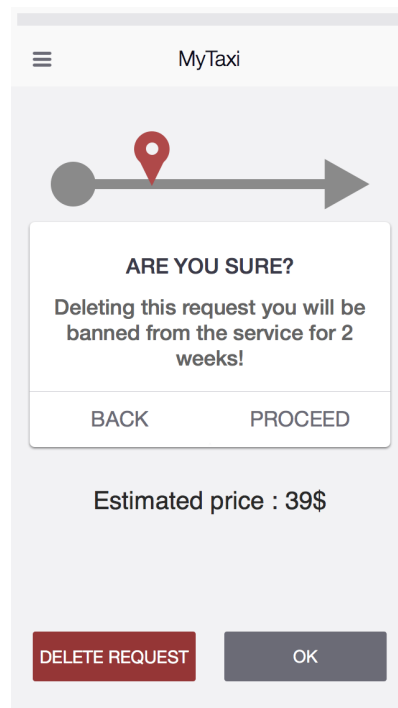
(a) Book



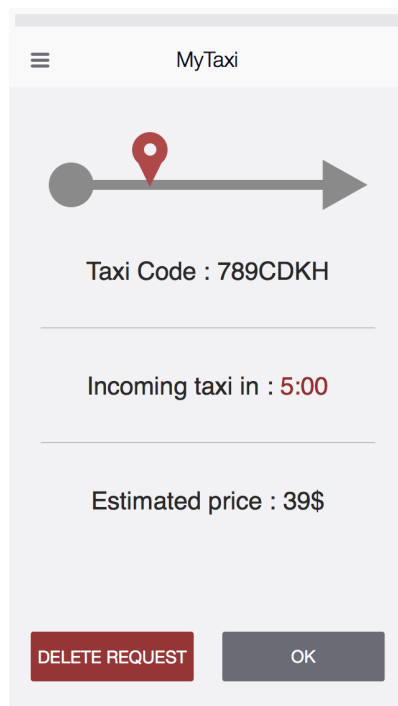
(b) Call



(a) Confirm



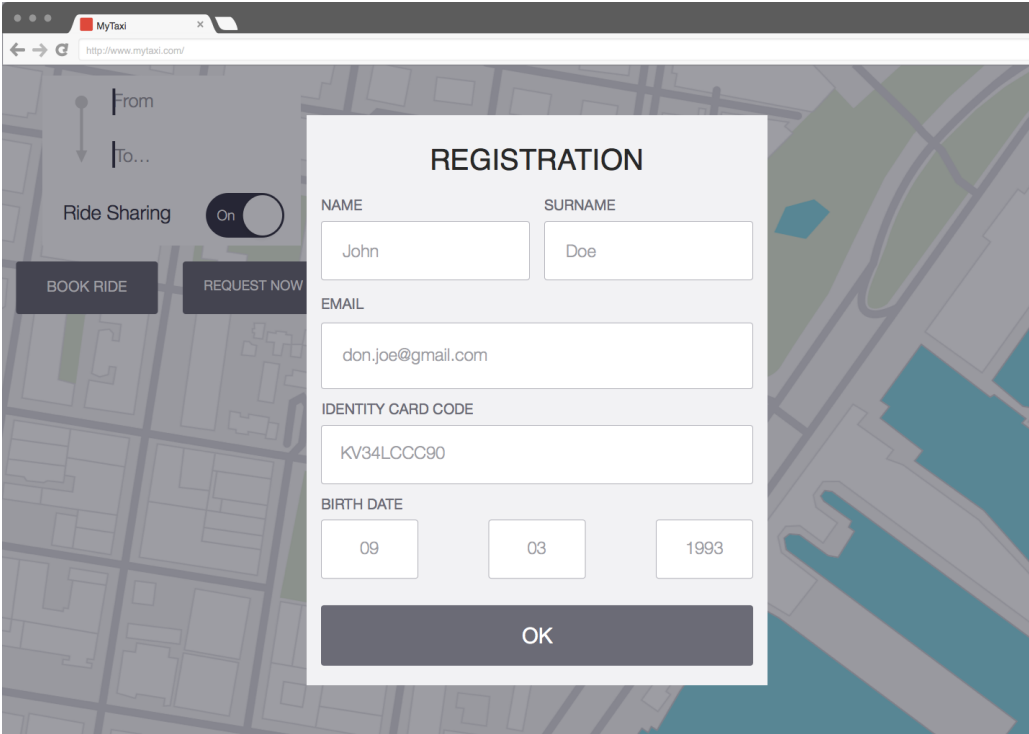
(b) Delete



(c) Pending

### 3.1.2 Web Application

Figure 4: Register

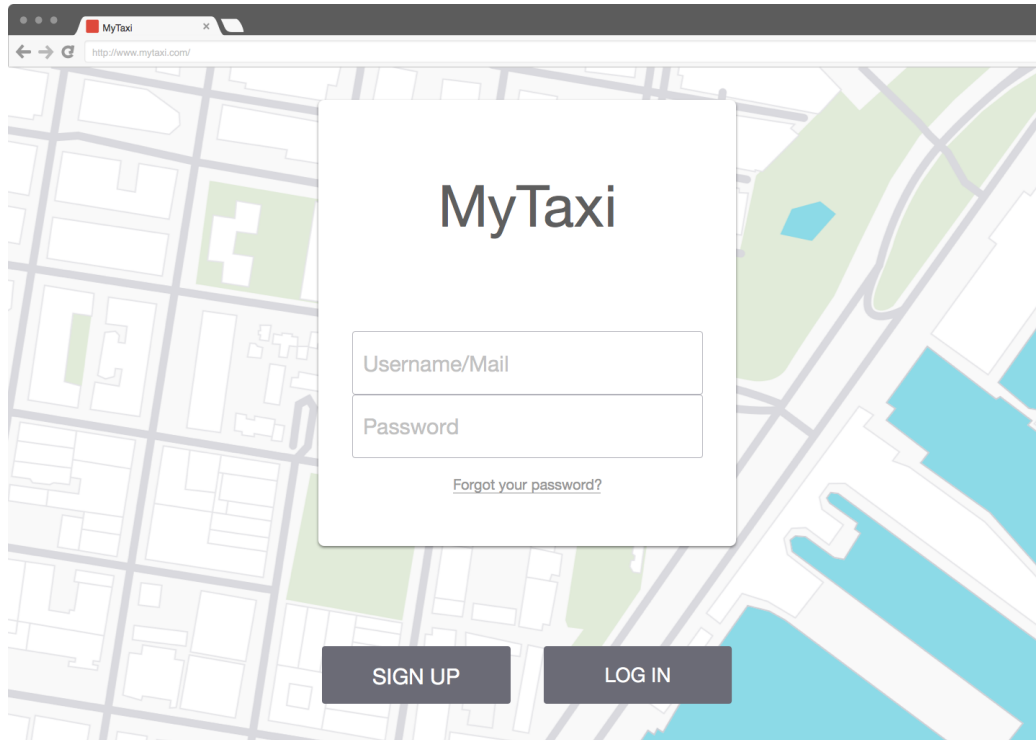


The image shows a web browser window with the address bar displaying "http://www.mytaxi.com/". The browser's title bar says "MyTaxi". The background of the page is a map. Overlaid on the map is a registration form titled "REGISTRATION". The form contains the following fields and controls:

- NAME**: A text input field containing "John".
- SURNAME**: A text input field containing "Doe".
- EMAIL**: A text input field containing "don.joe@gmail.com".
- IDENTITY CARD CODE**: A text input field containing "KV34LCCC90".
- BIRTH DATE**: Three separate text input fields containing "09", "03", and "1993".
- OK**: A large dark button at the bottom of the form.

In the background, to the left of the form, there is a "Ride Sharing" toggle switch set to "On", and two buttons labeled "BOOK RIDE" and "REQUEST NOW".

Figure 5: Login



A screenshot of a web browser displaying the MyTaxi login page. The browser's address bar shows the URL <http://www.mytaxi.com/>. The page features a background map of a city. In the center, there is a white login form with the title "MyTaxi". Below the title are two input fields: "Username/Mail" and "Password". A link labeled "Forgot your password?" is positioned below the password field. At the bottom of the form, there are two buttons: "SIGN UP" and "LOG IN".

MyTaxi

Username/Mail

Password

[Forgot your password?](#)

SIGN UP LOG IN

Figure 6: Book

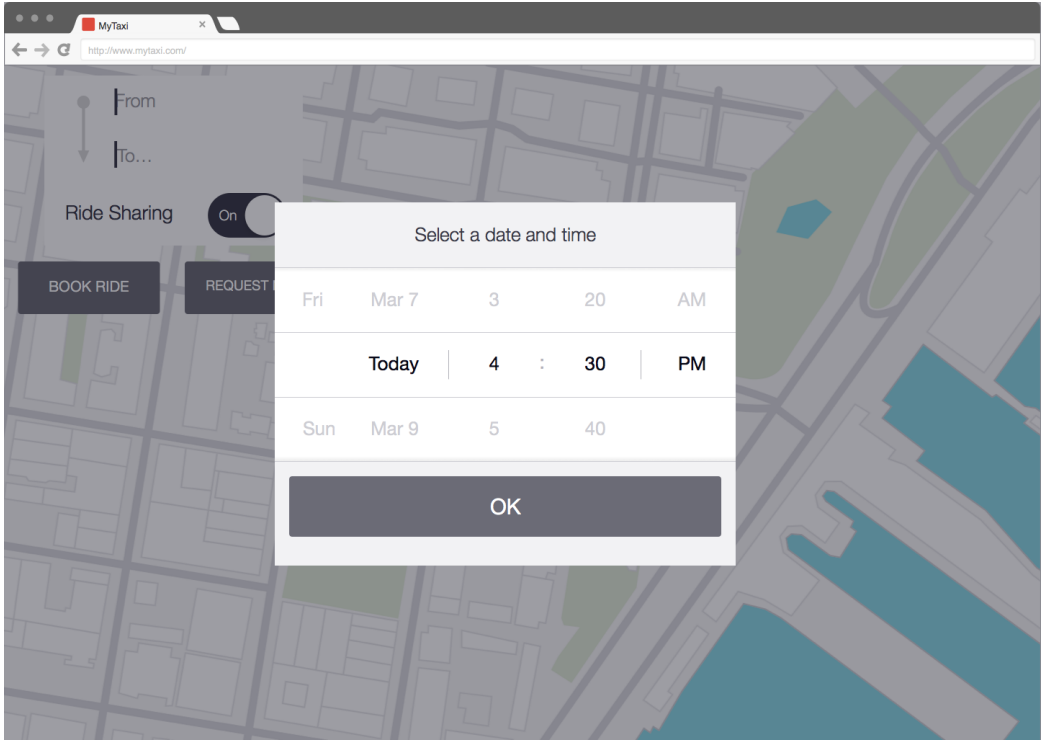
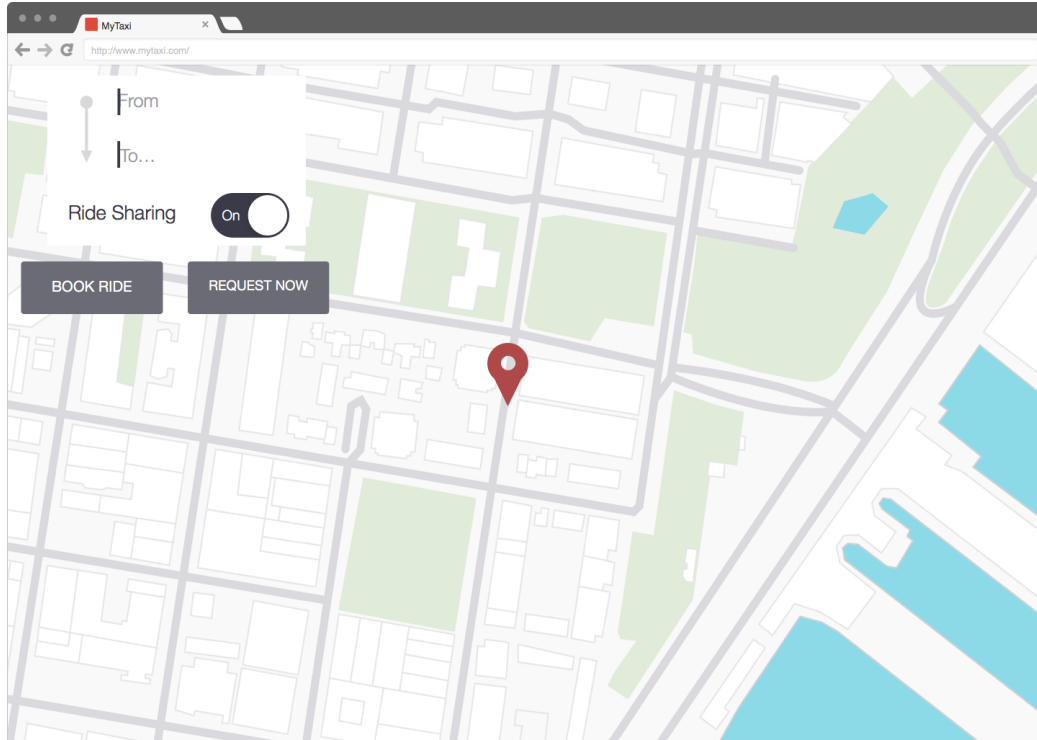
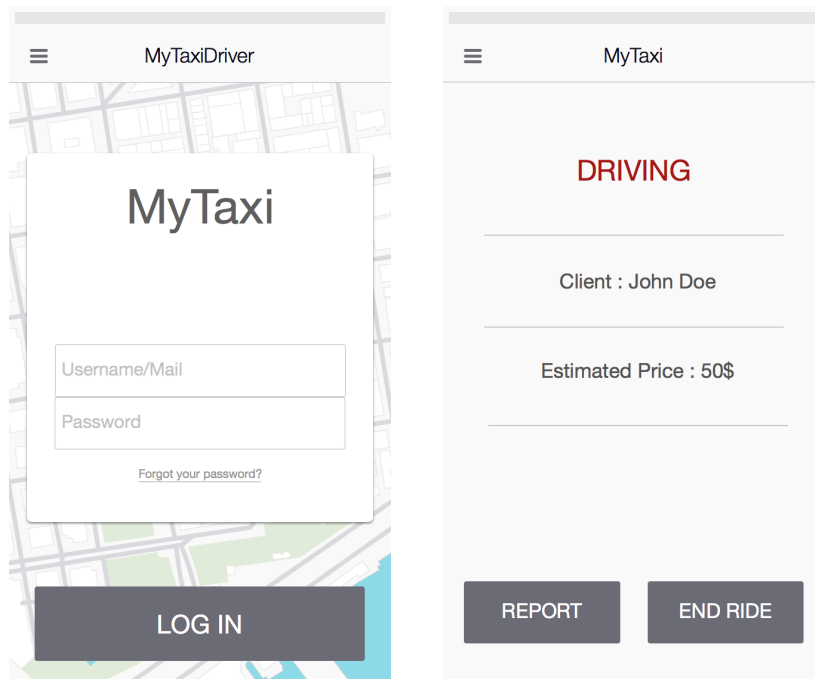


Figure 7: Request



### 3.1.3 Mobile Application (*Taxi Driver*)

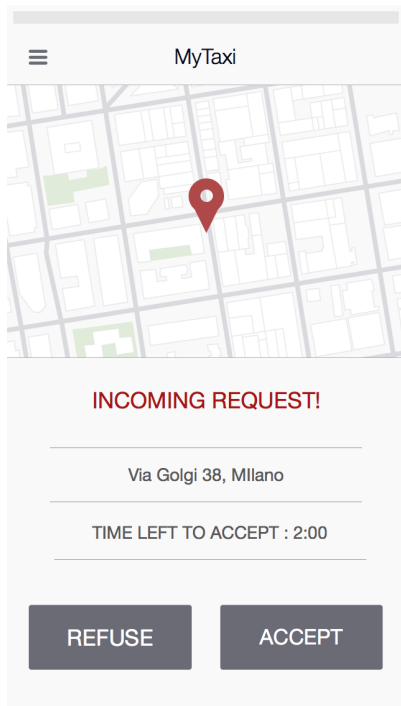


(a) Log in

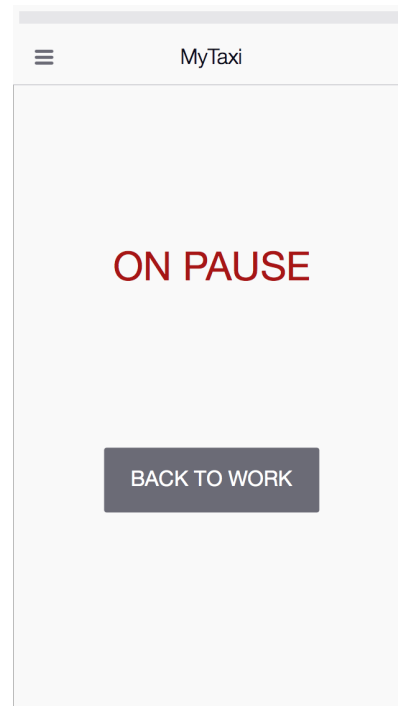
(b) On Ride

MyTaxi
Client : John Doe Duration : 50 min Price : 35\$ Date : 27/12/2015
Client : Samantha Meaning Duration : 34 min Price : 22\$ Date : 27/12/2015
Client : Aim Sokul Duration : 12 min Price : 5\$ Date : 27/12/2015
Client : Ketel Muibien Duration : 34 min Price : 22\$ Date : 27/12/2015
Client : John Cena Duration : 93 min Price : 90\$ Date : 27/12/2015
Client : Pingu Statico Duration : 50 min Price : 35\$ Date : 27/12/2015
Client : John Doe Duration : 50 min

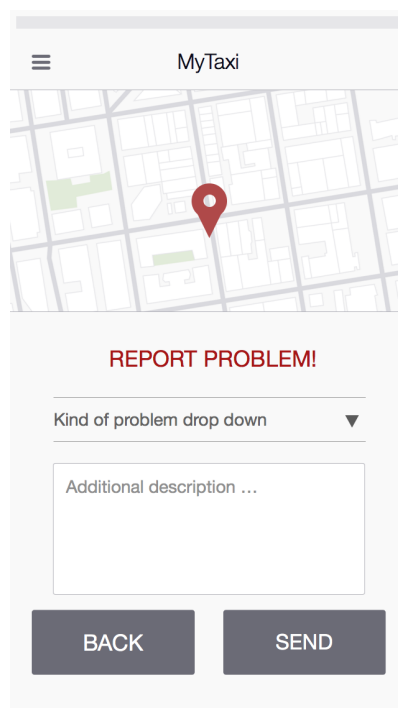
(c) Ride History



(a) Incoming Request



(b) Idle State



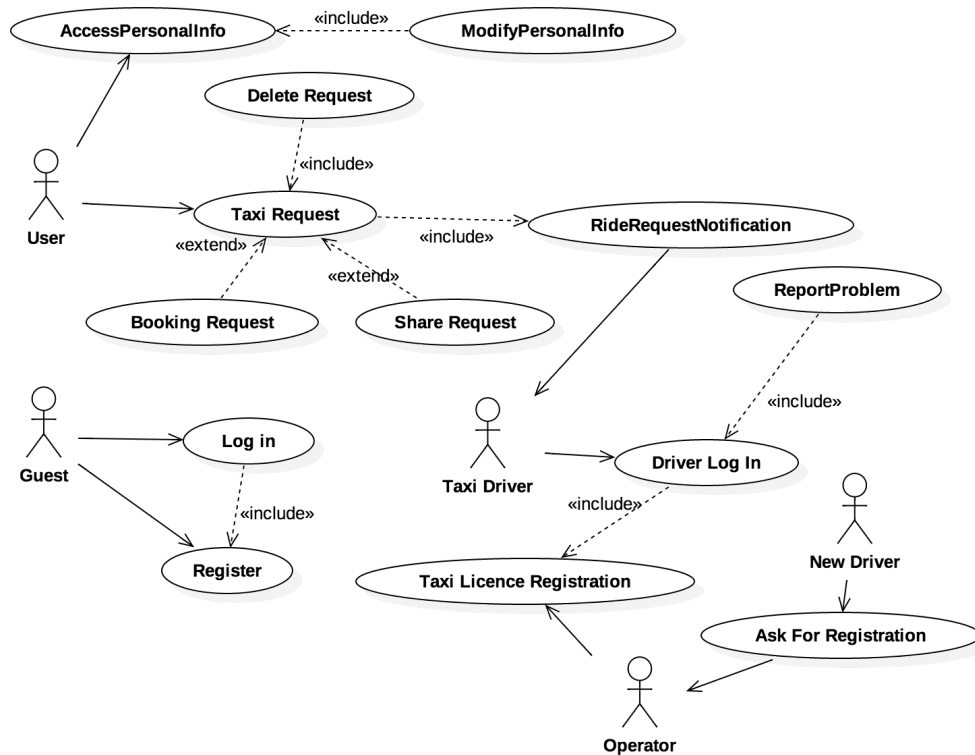
(c) Report Problem



## 3.2 Scenarios

This section describes the most common scenarios of the *system-to-be* highlighting all the interactions between the *User* and the system. The image below gives a general overview of the use cases.

Figure 10: Use Cases



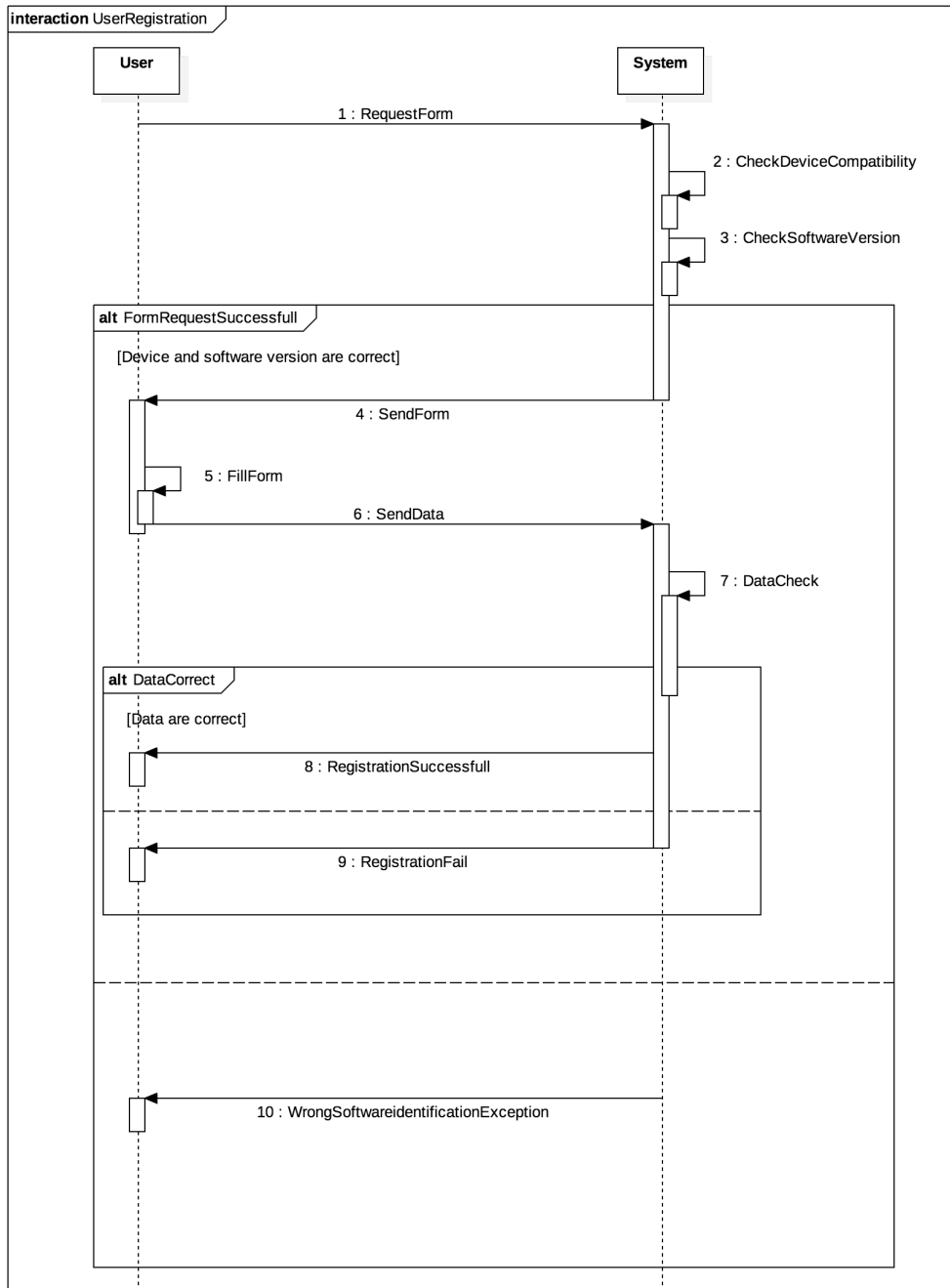
### 3.2.1 User Sign-up

**Scenario** Alice is tired of calling a taxi by phone. So she decides to download a new app called *MyTaxiService*. Once the download is completed she opens the app and signs-up providing her email, name, surname, password, date of birth and ID.

Table 1: User Registration Use Case

Actor	Guest
Goal	Register guest
Input condition	NULL
Event flow	<ol style="list-style-type: none"><li>1. Guest clicks on the sign up button</li><li>2. Guest fill the fields</li><li>3. Guest press confirm button</li><li>4. Backend check the data and confirm registration</li></ol>
Output condition	Guest successfully ends registration process. Now he/she can log in using the credential specified.
Exception	<ol style="list-style-type: none"><li>1. Data inserted are not consistent with the requirements</li><li>2. User is already registered</li><li>3. Connection not available</li></ol>

Figure 11: Sequence Diagram



### 3.2.2 User Log in

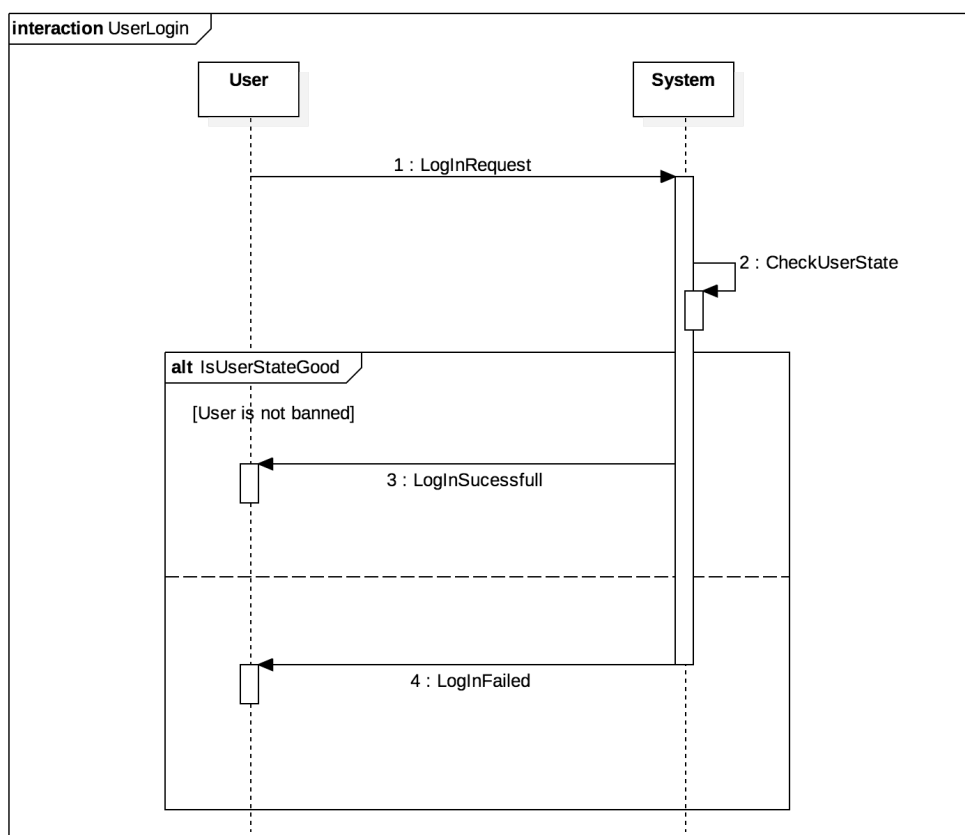
#### Scenario

Bob is a registered user. Bob wants to book a taxi, he opens the mobile app, he provides his email and his password and after the verification he can access the service.

Table 2: User Log In Use Case

Actor	User
Goal	User is logged in([F1])
Input condition	User is correctly registered. Internet connection is available.
Event flow	1. User enters his credentials 2. User presses the login button 3. The backend checks the credential
Output condition	User can access the service.
Exception	1. Wrong email or password 2. No connection available 3. App has to be updated

Figure 12: Sequence Diagram



### 3.2.3 Modify Data

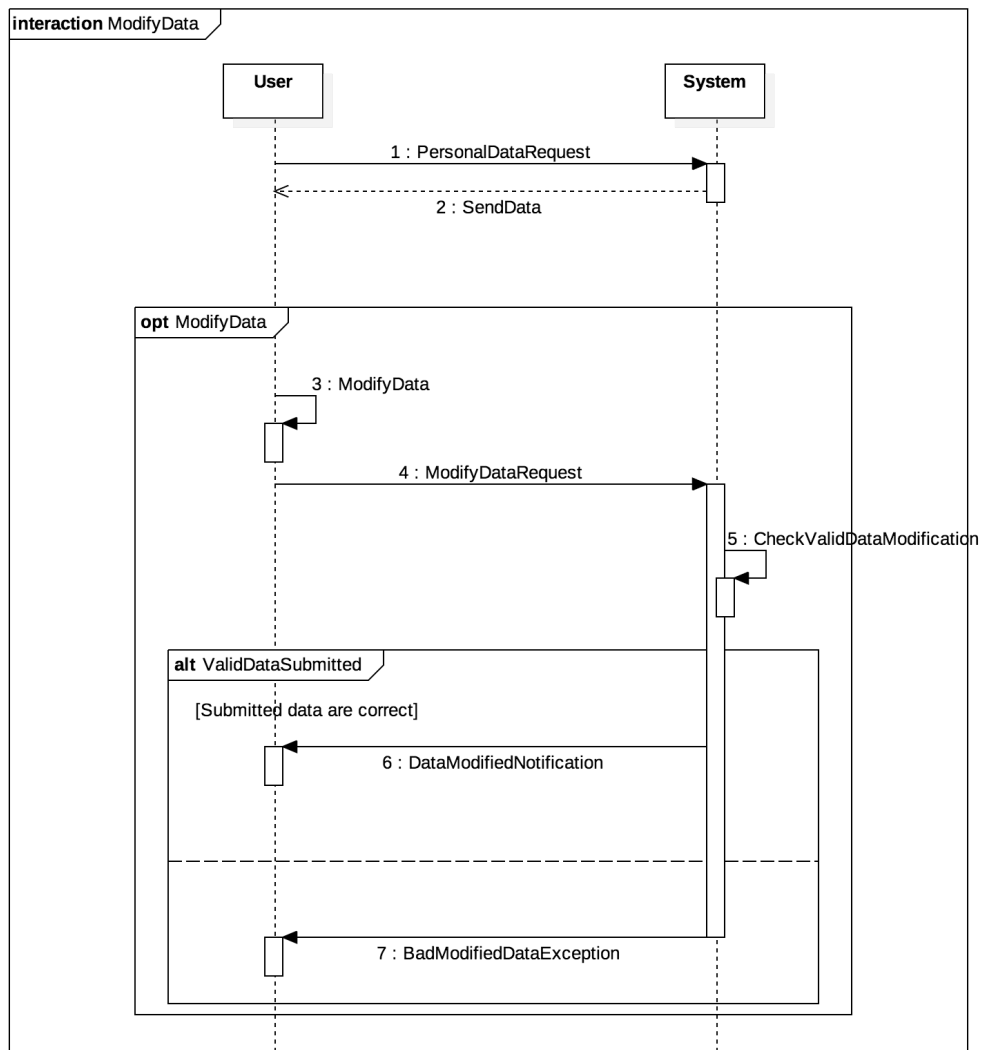
#### Scenario

Alice is logged in. She wants to change some data she entered wrong.

Table 3: User Modify Data Request

Actor	User
Goal	Modify personal data
Input condition	User is correctly logged in.
Event flow	<ol style="list-style-type: none"><li>1. User press the modify data button</li><li>2. User modify data and press confirm</li><li>3. Backend notify the user of correct modification</li></ol>
Output condition	User can consult the new data
Exception	<ol style="list-style-type: none"><li>1. Connection unavailable</li><li>2. Bad written data</li></ol>

Figure 13: Sequence Diagram



### 3.2.4 Taxi Request

#### Scenario

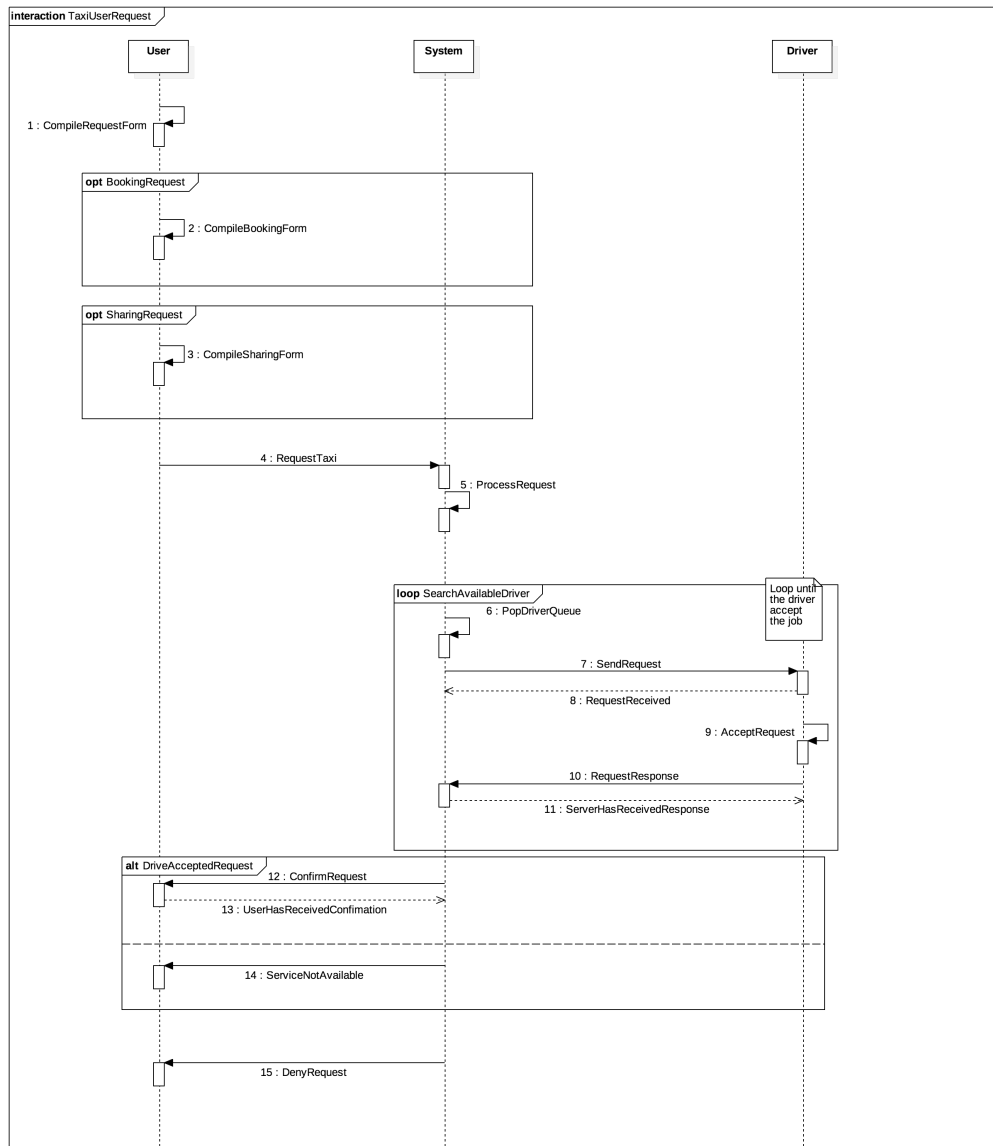
Alice is logged in. She wants to book a taxi. She opens the app, she fills all the fields, she decides if she wants the taxi now or later and waits for the response.

Table 4: User Taxi Request Use Case

Actor	User
Goal	Request a taxi
Input condition	User is correctly logged in.
Event flow	<ol style="list-style-type: none"><li>1. User press the request taxi button</li><li>2. User fill the form</li><li>3. User decides if he want to perform an Immediate Request or an Reservation Request</li><li>4. User press confirm button</li><li>5. Backend confirm the reservation</li></ol>
Output condition	NULL
Exception	<ol style="list-style-type: none"><li>1. Connection unavailable</li><li>2. Data in the form are not consistent with requirements</li></ol>



Figure 14: Sequence Diagram



### 3.2.5 Taxi Driver Registration

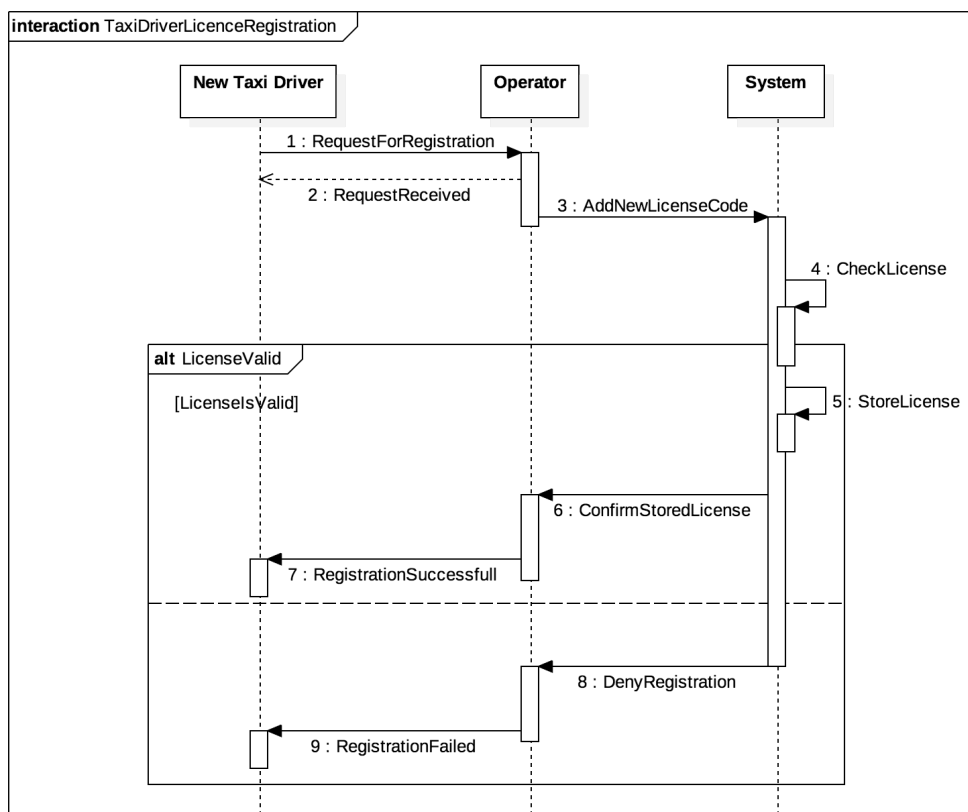
#### Scenario

Bob, a support operator receives a new mail from Alice, a taxi driver who wants to join the service. Bob checks the validity of the license and reply to Bob.

Table 5: Operator Register A New License

Actor	Operator
Goal	Taxi driver registration
Input condition	Operator is correctly logged in.
Event flow	1. Operator receive the request 2. Check the taxi license validity 3. Insert the new driver in the database 4. Notify the driver
Output condition	Now the driver can accept calls.
Exception	1. No connection available

Figure 15: Sequence Diagram



### 3.2.6 Delete Request

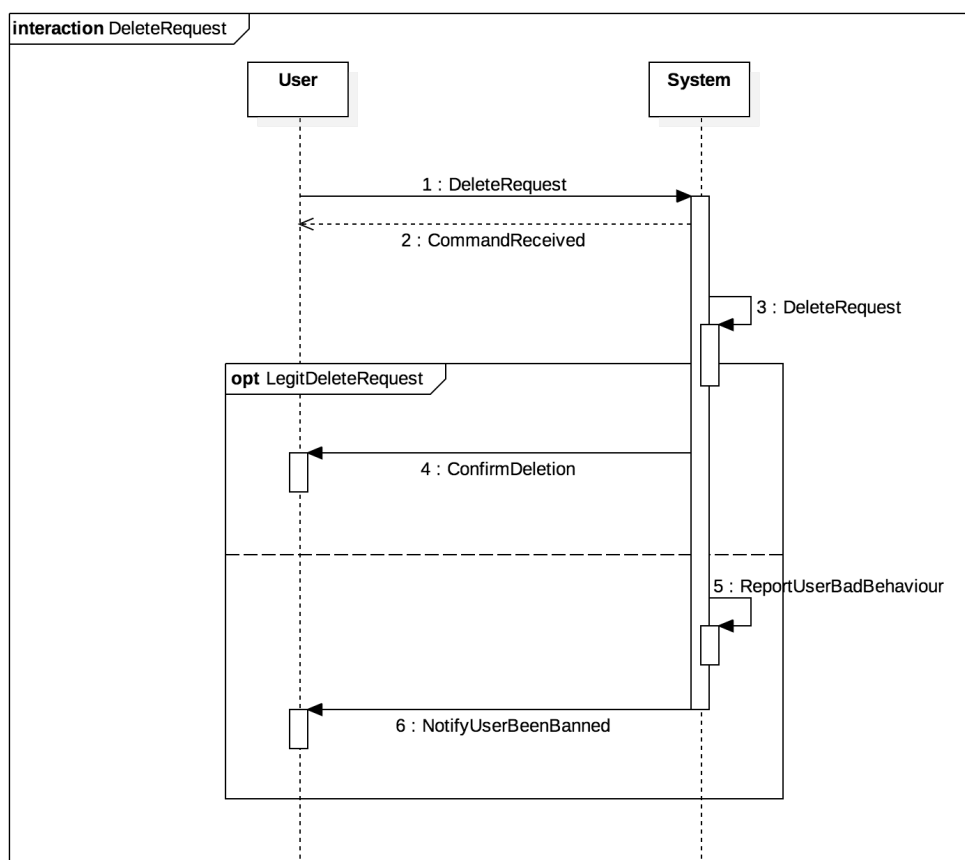
#### Scenario

Bob asked for a taxi 2 minutes ago. Bob finds out he does not need a taxi ride anymore. Bob tries to delete his ride request. The system decides if the delete request is on time and legit.

Table 6: User Delete Request Use Case

Actor	User
Goal	Delete a taxi request
Input condition	User is correctly logged in. User has a pending request.
Event flow	1. User press the delete button 2. User confirm on the pop up 3. Backend notify the user of the deletion and the possible consequent ban
Output condition	User is logged out if the system ban him/her
Exception	1. Connection unavailable 2. Data in the form are not consistent with requirements

Figure 16: Sequence Diagram



### 3.2.7 Report Problem

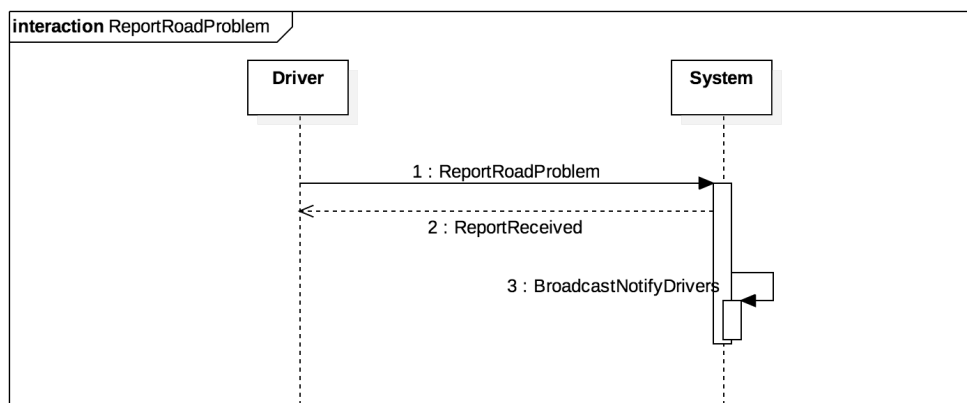
#### Scenario

Bob is a taxi driver. Bob is riding his cab while a incident occurs and he will have to wait at least 30 minutes. Bob report the incident to the system using the app.

Table 7: Report Problem

Actor	Driver
Goal	Report a problem
Input condition	Driver is correctly logged in.
Event flow	1. Driver press the report problem button 2. Driver insert the data of the problem 3. Driver press confirm button 4. Backend notify the driver
Output condition	NULL
Exception	1. No connection available

Figure 17: Sequence Diagram



### 3.2.8 Ride Sharing

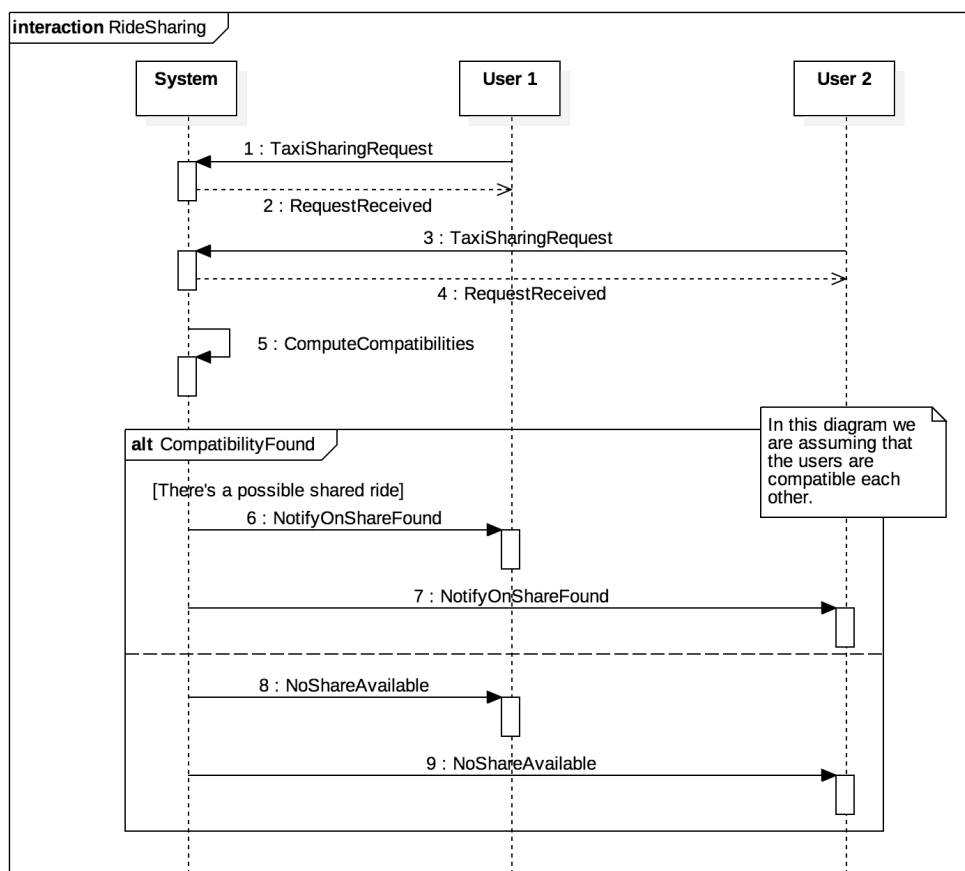
#### Scenario

Bob wants to cut the price of the taxi ride he is booking. He decides to enable the sharing option on the app.

Alice wants to cut the price too.

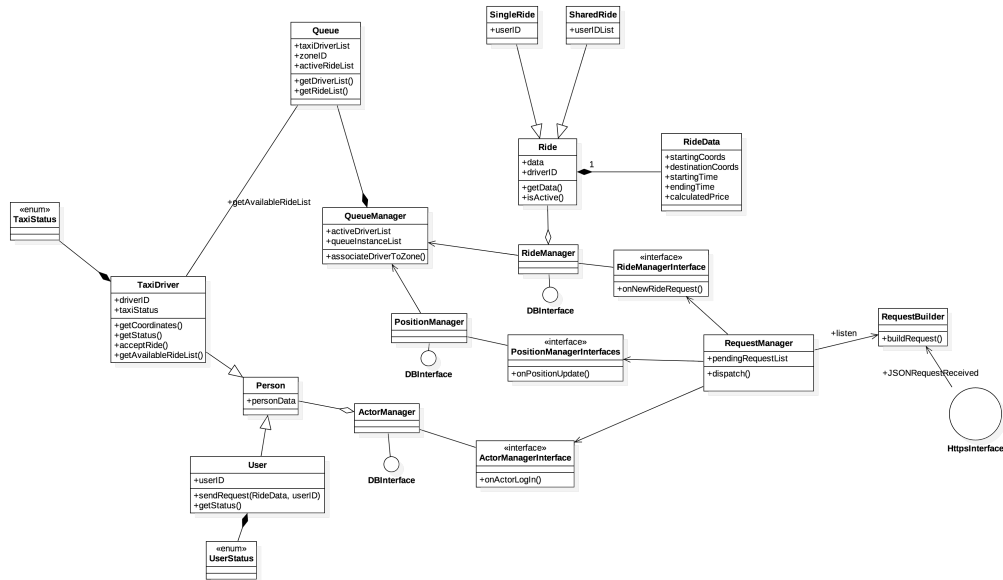
She decides to enable the sharing option on the app and if their paths are *Admissible Paths*, they will share the same ride.

Figure 18: Sequence Diagram





### 3.3 Class Diagram



### 3.4 State Charts

Here are presented the state charts of the main objects of the *system-to-be*

Figure 19: User State Chart

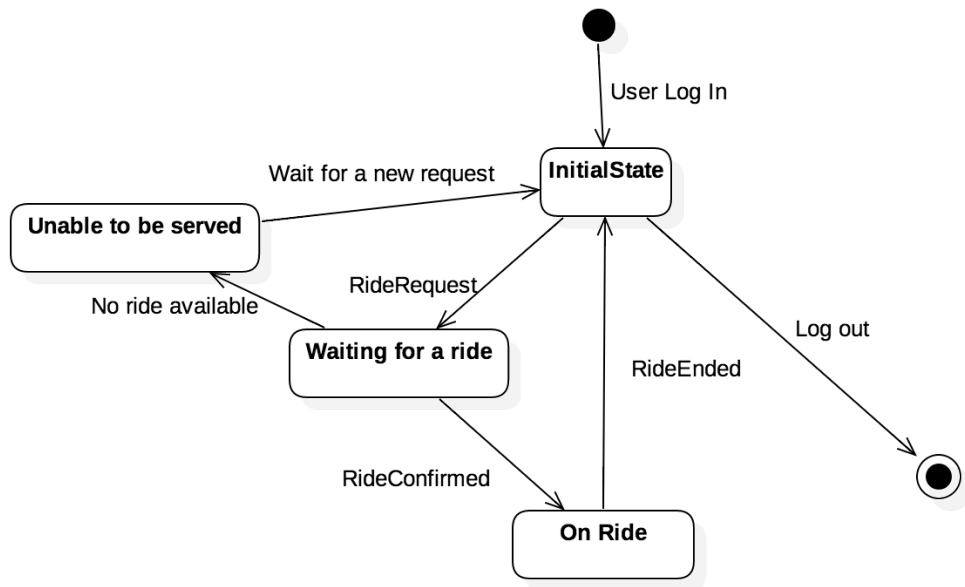


Figure 20: Ride State Chart

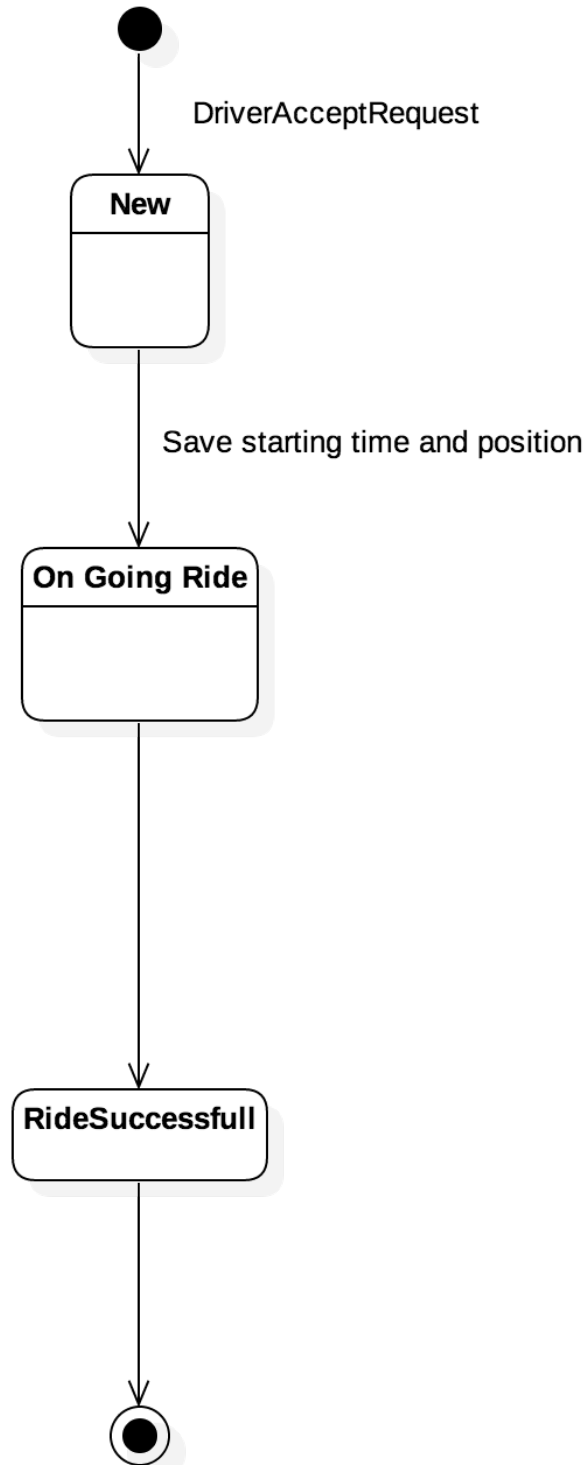


Figure 21: Driver State Chart

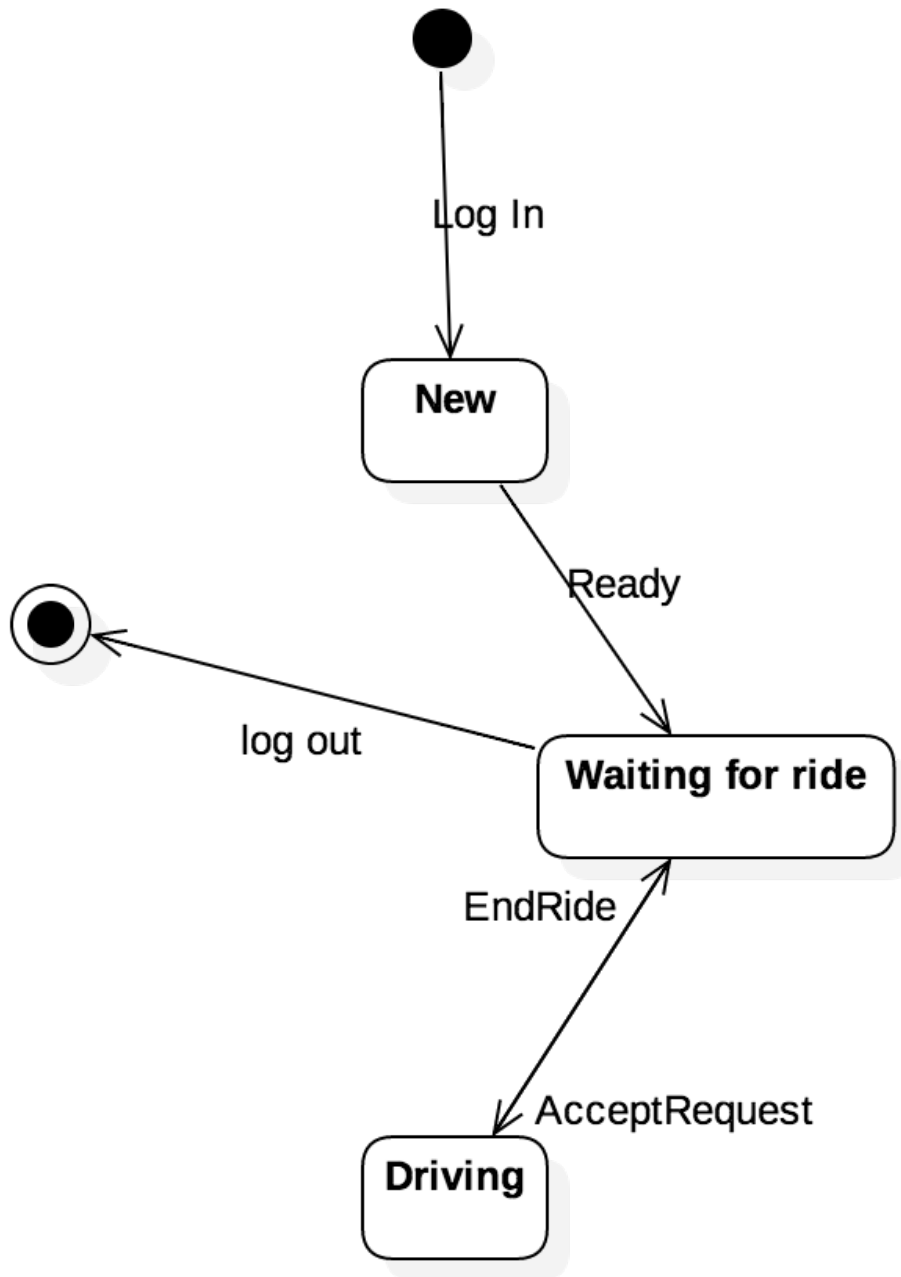
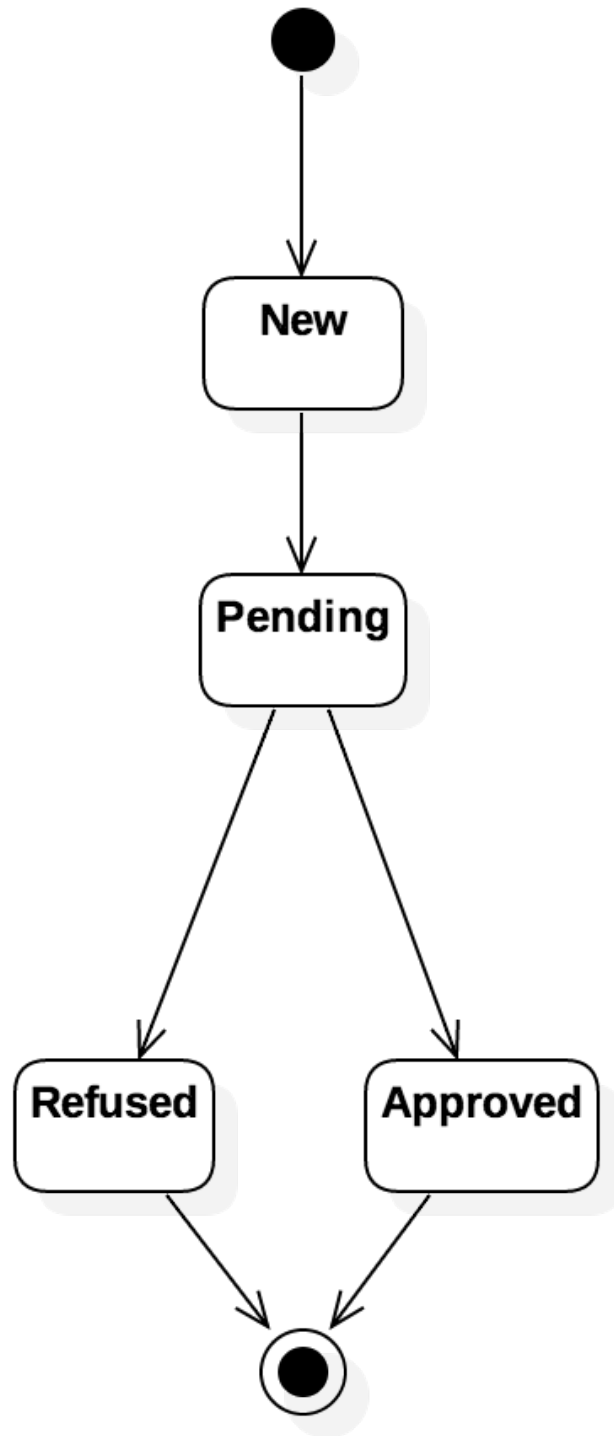


Figure 22: Request State Chart



## 3.5 Functional Requirements

### 3.5.1 Web Application

This subsection specifies in a more detailed level the requirements for the main functions specified in section 2.2.1 on page 10.

- [FR1] The application must allow every unregistered person above 13 years old to register providing email, ID number, password, name, surname and date of birth
- [FR2] The application must allow only registered user to access the service
- [FR3] The application must allow the user to choose his starting position
- [FR4] The application must allow the user to select the destination position
- [FR5] The application must allow the user to select the number of passengers
- [FR6] The application must check that only valid coordinates and addresses are selected by the user
- [FR7] The application must allow the user to reserve a taxi for a chosen date and position
- [FR8] The application must allow a taxi reservation only if booked 2 hours before the chosen date
- [FR9] The application must allow the user to activate the sharing option to allow other users to share the ride
- [FR10] The application must set a timeout for finding a shared ride.
- [FR11] The application must allow the user to find sharing rides available in his path.
- [FR12] The application must allow the user to modify his personal information (except for the ID number)

### 3.5.2 Mobile Application (*User*)

Requirements for the main functions specified in section 2.2.1 on page 10.

- [FR13] The application must allow every unregistered person above 13 years old to register providing email, ID number, password, name, surname and date of birth

- [FR14] The application must allow only registered user to access the service
- [FR15] The application must allow the user to choose his starting position
- [FR16] The application must allow the user to use the GPS interface to select the starting position
- [FR17] The application must allow the user to select the destination position
- [FR18] The application must allow the user to select the number of passengers
- [FR19] The application must check that only valid coordinates and addresses are selected by the user
- [FR20] The application must allow the user to reserve a taxi for a chosen date and position
- [FR21] The application must allow a taxi reservation only if booked 2 hours before the chosen date
- [FR22] The application must allow the user to active the sharing option to allow other users to share the ride
- [FR23] The application must allow the user to find sharing rides available in his path.
- [FR24] The application must allow the user to modify his personal information (except for the ID number)

### 3.5.3 Mobile Application (*Taxi Driver*)

Requirements for the main functions specified in section 2.2.2 on page 11.

- [FR25] The application must allow only registered drivers <sup>9</sup> to access the service
- [FR26] The application must send every 2 minutes a position update to the Back-End Application.
- [FR27] The application must allow the driver to accept a ride request in just one tap
- [FR28] The application must allow the driver to refuse a ride request in just one tap

---

<sup>9</sup>See section 2.5.4 on page 14

- [FR29] The application must allow the driver to report most common issues using a drop-down
- [FR30] The application must show a list of the last 30 rides, providing information about time, passenger and cost.
- [FR31] The application must allow the driver to report an absent user selecting the appropriate ride request.
- [FR32] The application must notify the driver when the sharing option is active
- [FR33] The application must show the driver all the starting and arrival position when the sharing option is active
- [FR34] The application must dim the screen light during late hours and when a gallery is detected by the GPS

#### **3.5.4 Back-End Application**

Requirements for the main functions specified in section 2.2.3 on page 11.

- [FR35] The application must accept registration request from the Mobile Application (*User*) and the Web Application and create a new user
- [FR36] The application must allow a taxi reservation only if booked 2 hours before the chosen date
- [FR37] The application must provide a control panel to allow the operator to register a new taxi driver
- [FR38] The control panel must be accessible only with a secret password
- [FR39] The application must generate automatically login credentials for new taxi driver
- [FR40] The application must record every ride and store all the data in the database
- [FR41] The application must handle one queue for every city zone
- [FR42] The application must assign a driver to a queue if and only if the driver is in the zone of the corresponding queue
- [FR43] The application must sort the queue in a descending order using the driver idle time as ordering parameter



- [FR44] The application must calculate an estimated time and cost for every ride request using Google's Maps API
- [FR45] The application must allow two users to share a taxi ride only if their path are admissible<sup>10</sup>
- [FR46] The application must update the database after a user request some data modification
- [FR47] The application must send a password reset email when the user signals that he has lost his password
- [FR48] The application must ban a user if he cancel a request that has been already processed
- [FR49] The application must ban a user if the taxi driver report the user as absent
- [FR50] The application must accept a ride request from a user only if the user is not banned
- [FR51] The application must decline an Immediate Request from a user if it finds another pending Immediate Request from the same user or if the user is involved in an Active Ride.
- [FR52] The application must reply to reservation immediately but must look for an available taxi only 10 minutes before the chosen date
- [FR53] The application must be developed in order to be extensible <sup>11</sup>

---

<sup>10</sup>See section 1.3 on page 6

<sup>11</sup>The application developer should follows these guidelines <https://docs.oracle.com/javase/tutorial/ext/basics/spi.html>

### 3.6 Alloy

In this section we provide an Alloy model for the *system-to-be*.

The Alloy model includes all major requirements and constrains and checks the consistency of the most important components of the application.

We explicitly chose not to implement the possibility of one *User* to request a taxi for multiple people. This choice is due to the fact that, having relatively large taxis with a relatively large number of available seats, we could have incurred in overflow problems when calculating the sum of the alloy “Int” primitive. We believe that this choice is not limiting the model, that would need just one more property to a *signature* and one more *fact* in order to implement that.

### 3.6.1 Alloy code

```
module myTaxiService

/***** Classes *****/

sig Person {}

/** User */
sig User {
  status: one UserStatus,
  requests : set Request
}

abstract sig UserStatus {}

lone sig ActiveUserStatus extends UserStatus {}
lone sig InactiveUserStatus extends UserStatus {}

/** Request */
abstract sig RequestStatus{
}

lone sig ApprovedRequestStatus extends RequestStatus {}
lone sig RefuseRequestStatus extends RequestStatus {}

abstract sig Request{
  status: one RequestStatus
}

sig ImmediateRequest extends Request{
}

sig ReservationRequest extends Request{
}

/** Ride */

abstract sig Ride {
  driver: one TaxiDriver,
  user: some User
}

sig SingleRide extends Ride {} {#user = 1}
sig SharedRide extends Ride {} {#user > 1}

sig TaxiDriver {
  car: one Taxi,
  status: one TaxiStatus
}

sig TaxiZone {
  queue: one Queue
}

sig Taxi {
  seats: one Int
}

sig Queue {
  driver: set TaxiDriver
}
```

```

}

one sig QueueManager {
  queues: set Queue
}

/****** RideManager *****/

sig RideReqMap{
  ride : lone Ride,
  request : some Request
}

one sig RidesManager {
  qm: one QueueManager,
  ridesmap : set RideReqMap
}

abstract sig TaxiStatus {}

lone sig TaxiBusy extends TaxiStatus {}
lone sig TaxiAvailable extends TaxiStatus {}

/****** Facts *****/

//there are enough seats to accommodate all the users
fact allusersFit {
  all r: Ride | #r.user ≤ r.driver.car.seats
}

//each taxi has a maximum number of seats
fact maxTaxiSeats {
  all t: Taxi | quantityIsInBounds[t.seats]
}

//there exists at least one user per ride
fact rideHasReasonToExist {
  no r: SingleRide | #r.user < 1
}

//there exists at least one driver per taxi
fact taxiCarHasReasonToExist {
  all t: Taxi | one d: TaxiDriver | t in d.car
}

//two different rides can't have the same user
fact userHasOneRide {
  all u: User | lone r: Ride | u in r.user
}

//two different queues can't have the same taxi driver
fact uniqueQueueForDriver {
  all d: TaxiDriver | lone q: Queue | d in q.driver
}

//two different taxi drivers can't have the same car
fact uniqueTaxi{
  all t: Taxi | lone d: TaxiDriver | t in d.car
}

//two rides can't have the same driver
fact uniqueDrive {
  all d: TaxiDriver | lone r: Ride | d in r.driver
}

```

```

}

//if the driver is available he is not in a ride
fact availability {
    all d: TaxiDriver | isAvailable[d] implies !( isInRide[d] )
}

//if the user is active he is in a ride
fact activity {
    all u: User | isActive[u] iff ( isUserInRide[u] )
}

//if the user is active he is in a ride
fact inactivity {
    all u: User | isInactive[u] iff !( isUserInRide[u] )
}

//if the driver is available he is in a queue
fact availabilityInQueue {
    all d: TaxiDriver | isAvailable[d] iff !( isNotInQueue[d] )
}

//if the driver is in a ride then the status is busy
//this is simply one sided because it may happen that a driver is simply
    ↪ busy but not working
fact unavailability {
    all d: TaxiDriver | isInRide[d] implies isBusy[d]
}

//if the driver is busy, he is not in a queue
fact noBusyDriverBelongsToQueue {
    all d: TaxiDriver | isBusy[d] implies isNotInQueue[d]
}

//all queues belong to the QueueManager
fact allQueuesBelongToQueueManager {
    all q: Queue | one qm: QueueManager | q in qm.queues
}

//two zones cannot have the same queue
fact oneZoneOneQueue {
    all q: Queue | one z: TaxiZone | q in z.queue
}

//two different maps cant have the same ride
fact uniqueRideForMap{
    all r: Ride | one m: RideReqMap | r in m.ride
}

// for every request there is one user
fact uniqueRequestForMap{
    all rq:Request | one u: User |rq in u.requests
}

// for every request there is at most one map
fact uniqueRequestForMap{
    all rq:Request | lone m: RideReqMap |rq in m.request
}

// if the req is approved there is exactly one map
fact mapForApproved{
    all rq:Request | isApprovedRequest[rq] implies one m: RideReqMap | rq

```

```

    ↪ in m.request
}

// if the req is not approved there is exactly zero map
fact mapForRefused{
    all rq:Request | !isApprovedRequest[rq] implies no m: RideReqMap | rq
    ↪ in m.request
}

//two different maps cant have the same req
fact uniqueRequestForMap{
    all rq: Request | one m: RideReqMap | isApprovedRequest[rq] iff rq in m
    ↪ .request
}

//all maps belong to ride manager
fact allMapToRideManager{
    all m : RideReqMap | one mn: RidesManager | m in mn.ridesmap
}

// if the map has a single ride the map has only one request
fact oneRequestInMapForSingleRide{
    all m: RideReqMap | m.ride in SingleRide implies #m.request = 1
}

// if the map has a shared ride the map has the same much users as the ones
    ↪ that requested it
fact correctRequestInMapForSharedRide{
    all m: RideReqMap | m.ride in SharedRide implies #m.request = #m.ride.
    ↪ user
}

fact userNumberCorrect {
    all m: RideReqMap | m.ride.driver.car.seats ≥ #m.request
}

// a request and a ride are connected iff they have the same user
fact userInRideAndInRequest {
    all m: RideReqMap | userOfMap[m] = ridersOfMap[m]
}

/***** Functions *****/

fun numberOfSeats [r: Ride]: Int {
    r.driver.car.seats
}

fun userOfRequest[rq:Request]: User{
    { u: User | rq in u.requests }
}

fun ridersOfMap[m: RideReqMap]: User {
    {u: User | some rd: Ride | rd = m.ride ∧ u in rd.user}
}

fun userOfMap[m: RideReqMap]: User {
    {u: User | some r: Request | r in u.requests ∧ r in m.request}
}

fun mapOfRideRequest[rd:Ride, rq: Request]: RideReqMap {

```

```

    {m: RideReqMap | m.ride = rd ∧ rq in m.request}
}

/****** Predicates *****/

pred isAvailable [ d: TaxiDriver ] {
    some s: TaxiAvailable | d.status in s
}

pred isActive [ u: User ] {
    some s: ActiveUserStatus | u.status in s
}

pred isBusy [ d: TaxiDriver ] {
    some s: TaxiBusy | d.status in s
}

pred isInactive [ u: User ] {
    some s: InactiveUserStatus | u.status in s
}

pred isInRide [ d: TaxiDriver ] {
    some r: Ride | d in r.driver
}

pred isUserInRide [ u: User ] {
    some r: Ride | u in r.user
}

pred isNotInQueue [ d: TaxiDriver ] {
    no q: Queue | d in q.driver
}

pred quantityIsInBounds [n: Int] {
    n > 1 ∧ n < 6
}

pred isApprovedRequest[r: Request]{
    one s: ApprovedRequestStatus | r.status in s
}

pred areConnected[rd: Ride, rq: Request]{
    one m : RideReqMap | rd = m.ride ∧ rq = m.request
}

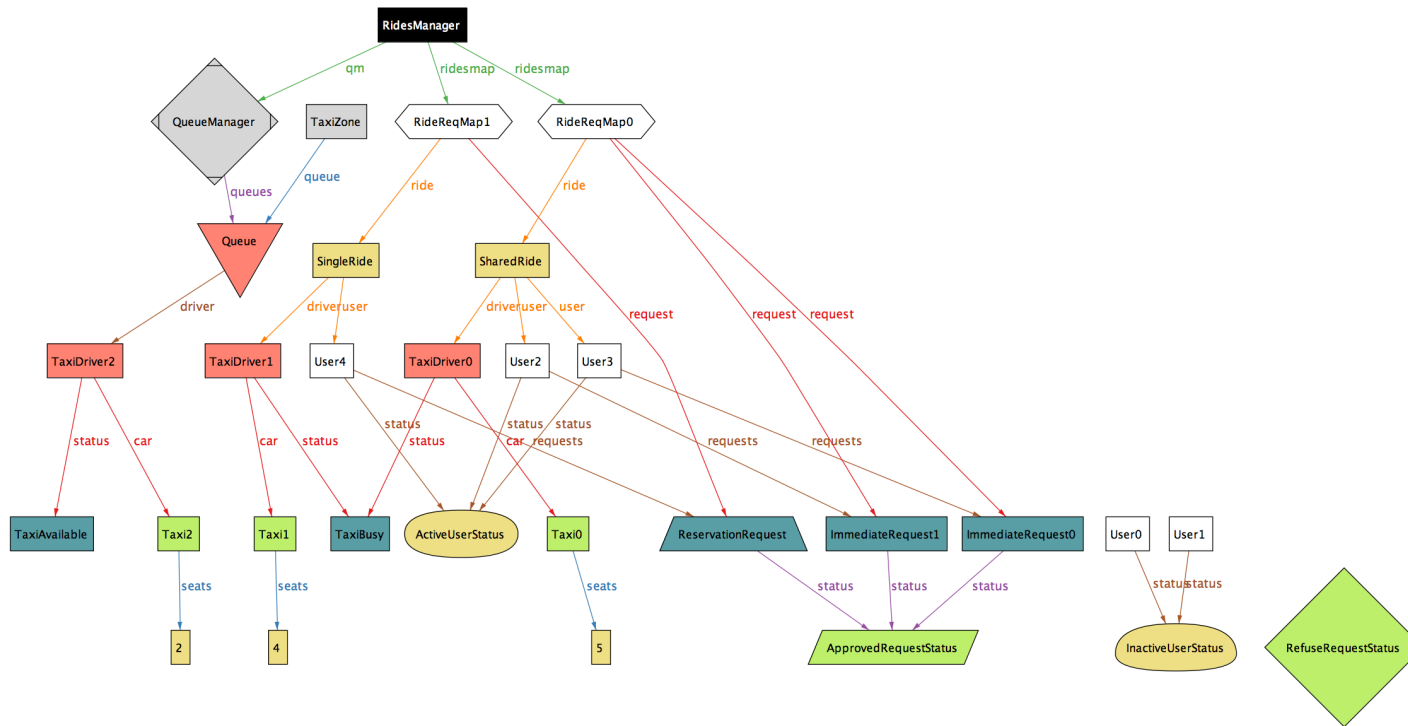
pred showGeneral {
    #SingleRide = 1
    #SharedRide = 1
    #User = 5
    #TaxiDriver > #Ride
    #Queue = 1
}

run showGeneral for 5

```

### 3.6.2 Alloy world considerations

Figure 23: The world generated by the predicate *showGeneral*.





We chose some minimum numbers for the cardinality of the key objects, in order to showcase some of the core relationships that make up the foundations of the modeled world.

We decided to generate two instances of *Ride*, one for each kind, that is one *Single Ride* and one *Shared Ride*. The former involves only one *User*, while the latter two *Users*.

The number of *Users* is enough to have some involved in a ride and others that are inactive.

We also have one more taxi driver than the ones needed for the rides, in order to showcase the queue implementation and the taxi zones.

# Appendices

## A Tools

- *Sublime Text 2* as editor
- *LatexTools* for *Sublime Text 2* + *MacTex* to build
- *Alloy 4.2* for Alloy modeling
- *Trello* for team coordination
- *Git* + *Git Flow* for version control

## B Hours of work

- Angelo Gallarello : 30 hours
- Edoardo Longo : 30 hours
- Giacomo Locci : 30 hours