



Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 Project: myTaxiService  
Requirement Analysis and Specification Document

Damiano Binaghi, Giovanni Zuretti

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Purpose . . . . .	4
1.2	Legacy System . . . . .	4
1.3	Scope . . . . .	4
1.4	Definitions and Acronyms . . . . .	5
1.4.1	Definitions . . . . .	5
1.4.2	Acronyms . . . . .	5
1.5	Identifying Actors . . . . .	5
1.6	Goals . . . . .	6
1.7	Identifying Stakeholders . . . . .	7
1.8	References . . . . .	7
1.9	Overview . . . . .	7
<b>2</b>	<b>Overall description</b>	<b>8</b>
2.1	Product perspective . . . . .	8
2.2	User characteristics . . . . .	8
2.3	Constraints . . . . .	8
2.3.1	Regulatory Policies . . . . .	8
2.3.2	Hardware Limitations . . . . .	8
2.4	Assumption and dependencies . . . . .	8
2.5	Future possible implementation . . . . .	9
<b>3</b>	<b>Specific Requirements</b>	<b>10</b>
3.1	Functional Requirements . . . . .	10
3.1.1	Allow registration of a new user to the system . . . . .	10
3.1.2	Allow log-in to already existing users . . . . .	10
3.1.3	Allow users to modify his personal information . . . . .	10
3.1.4	Allow insertion of new taxis and taxi drivers by the Taxi-Data manager . . . . .	10
3.1.5	Allow Taxi-Data manager to delete taxis that no longer exist or taxi drivers who no longer provide the service . . . . .	10
3.1.6	Allow Application manager to manage all other users (insertion of new TaxiData manager, change privileges of some users, etc.) . . . . .	11
3.1.7	Allow customer to real-time book a taxi ride and notify the assigned taxi driver . . . . .	11
3.1.8	Allow customer to book in advance a taxi ride (advance-booking) and notify the assigned taxi driver . . . . .	11
3.1.9	Allow fair management of the queues . . . . .	11
3.1.10	Allow taxi drivers to either accept or refuse the assignation and notify customer (in case of acceptance) . . . . .	12
3.1.11	Allow taxi drivers to refuse an accepted assignation in case of accidents, problems with the vehicle or other sort of problems, and notify customer with the new assignation . . . . .	12

3.1.12	Allow customer to cancel a reservation . . . . .	12
3.1.13	Allow taxi drivers to report fake-users in case a user who has booked a taxi and did not canceled his request does not show up . . . . .	12
3.1.14	Do not allow customer to take the wrong taxi or taxi driver to pick up the wrong customer . . . . .	13
3.1.15	Allow the development of applications that need to inter- face with myTaxiService through APIs . . . . .	13
3.2	Non-functional requirements . . . . .	14
3.2.1	User Interface . . . . .	14
3.2.2	API Interfaces . . . . .	18
3.3	Scenarios . . . . .	18
3.3.1	First Scenario . . . . .	18
3.3.2	Second Scenario . . . . .	19
3.3.3	Third Scenario . . . . .	19
3.3.4	Fourth Scenario . . . . .	19
3.3.5	Fifth Scenario . . . . .	20
3.3.6	Sixth Scenario . . . . .	20
3.4	UML Models . . . . .	21
3.4.1	Use Case Diagram . . . . .	21
3.4.2	Class Diagram . . . . .	26
3.4.3	Sequence Diagrams . . . . .	27
3.4.4	State Charts . . . . .	32
<b>4</b>	<b>Alloy Modeling</b>	<b>33</b>
4.1	Alloy Code . . . . .	33
4.2	Generated Worlds . . . . .	38
4.2.1	General World . . . . .	38
4.2.2	Queue World . . . . .	39
<b>5</b>	<b>Used Tools</b>	<b>39</b>

# **1 Introduction**

## **1.1 Purpose**

This is the Requirement Analysis and Specification Document (RASD) for the myTaxiService project. The purpose of this document is to collect and describe both functional and non-functional requirements of the Taxi Service application which we were asked to design. We will analyze our customers and stakeholders needs in order to design and build the best system which suits those needs. To accomplish this, we will present several kinds of diagrams and models (such as UML class diagram, Use Case diagram, UML Sequence diagram etc.) in order to guide as much as possible developers and programmers, who eventually will implement the application, through the right steps which will allow to reach our stakeholders goals.

## **1.2 Legacy System**

In this particular case we have no legacy system or previous developed applications with which we need to interface our new system. The government of the city, for which we are going to produce this document, asked us to build the application from scratch, because no such thing already existed in the city. Thus, in our case no legacy system exists.

## **1.3 Scope**

The project aims to create a new application for the management of the taxi service for helping both customers, in order to make them book easily their rides, and taxi drivers, in order to let them manage their work in a more efficient and organized way. Our objective is, thus, to simplify the access to the taxi service for customers and to guarantee a fair management of the taxi queues. Taxis can be booked by any user who either downloaded the application on his/her mobile phone or via browser through the web application. Registration is required in both cases. Users will be notified with the code of the incoming taxi and the waiting time. To avoid waiting, users can book in advance their ride at least 2 hours earlier than the desired time. In such case, a taxi should be allocated to pick up the passenger right away at the desired time in the desired place. Drivers are stored in a queue and they need to be notified whenever the system assign them a new passenger. Drivers can either accept or refuse the assignation. Finally, the manager of the application (presumably the government of the city) should be able to modify the application any time, by adding new features and services through the use of programmatic interfaces.

## 1.4 Definitions and Acronyms

### 1.4.1 Definitions

In order to better understand who are the actors and which are the goals and the requirements of our application, we now define some useful terms we are going to use. Here is the list of terms:

- Taxi-Data manager: This term addresses the person (or the people) who will be in charge of simply add new taxis to the system or delete previous ones.
- Application manager: Super-user who manages all the other kind of users (customers, taxi drivers, taxi-data managers). For example, he can add new taxi-data manager and remove the old ones. We suppose the first of such user is created by programmers during development. Other application managers may be created by the application manager himself.
- Real-time-booking: This term is used for referring those times when a customer has immediate need of a taxi ride but he did not think of booking it in advance. Therefore, it identifies all those booking instances which need to be served immediately (or as soon as possible) by the system.
- Advance-booking: On the contrary, this term addresses the action of booking in advance a taxi ride.
- Fake-user: a user who booked a taxi and did not canceled his reservation but do not show up at the appointment.

### 1.4.2 Acronyms

- RASD: Requirement Analysis and Specification Document.
- DB: Data Base
- API:Application Programming Interface

## 1.5 Identifying Actors

We now proceed with the identification of the actors that we will encounter in modeling our system.

- Guest: A person who opens the home page of the web or mobile application. He can choose only to either sign up or log in, if he has already signed up.
- User: A guest who makes the log in becomes a user. A user can modify his personal profile or log out. This actor is a generalization of all the following actors.

- Customer User: It identifies the category of users which are going to exploit the taxi service. Such user can book a taxi ride (either in real-time or in advance).
- Taxi User: It identifies the category of taxi drivers. Such user can accept or refuse an assignation, report fake users and signal accidents.
- Taxi-Data Manager: see section 1.4 for definition.
- Application Manager: see section 1.4 for definition.

## 1.6 Goals

We are now going to list the goals of the myTaxiService application:

- Allow registration of a new user to the system.
- Allow log-in to an already registered user.
- Allow users to modify their personal data.
- Allow insertion of new taxis or new taxi driver by the Taxi-Data manager.
- Allow Taxi-Data manager to delete taxis that no longer exist or taxi drivers who no longer provide the service.
- Allow Application manager to manage all other users (insertion of new TaxiData manager, change privileges of some users, etc.)
- Allow customer to real-time book a taxi ride and notify the assigned taxi driver.
- Allow customer to book in advance a taxi ride (advance-booking) and notify the assigned taxi driver.
- Allow fair management of the queues.
- Allow taxi drivers to either accept or refuse the assignation and notify customer (in case of acceptance).
- Allow taxi drivers to refuse an accepted assignation in case of accidents, problems with the vehicle or other sort of problems, and notify customer with the new assignation.
- Allow customer to cancel a reservation.
- Allow taxi drivers to report fake-users in case a user who has booked a taxi and did not canceled his request does not show up.
- Do not allow customer to take the wrong taxi or taxi driver to pick up the wrong customer.
- Allow the development of applications that need to interface with my-TaxiService through APIs.

## 1.7 Identifying Stakeholders

Our main stakeholder, which is the financial stakeholder, is the government of the city who has contacted us for the development of myTaxiService. The principal needs of this stakeholder are to improve the quality taxi service of the city and to have both users and taxi drivers (which are potential voters in future elections) satisfied with how the application works. However, the government is not the only stakeholder we have. Indeed, customers and taxi drivers are also very important stakeholders of our application. Their needs are low switching costs, i.e. the cost (in terms of money, time, required-learning-time, etc.) of passing from how customers booked their taxis and how drivers managed their work before to the new way introduced by our application should be low, and, in order to be satisfied, they should perceive a sensible improvement in the service.

## 1.8 References

- Specification Document: Assignment 1 and 2.pdf.
- IEEE Standard for requirement and specifications (std830-1998)

## 1.9 Overview

This document is structured in four parts.

- Section 1: Introduction, which gives a very high level description of what are the goals and objectives of the application.
- Section 2: Overall description, which focuses on the constraints and the assumptions we had to do in collecting the requirements of our application.
- Section 3: Specific Requirements, which contains the list of requirements, scenarios and the UML diagrams (Use Case, Class diagram, Sequence diagrams and state charts). The aim of this section is to understand the actual functionalities that must be implemented in the application.
- Section 4: Alloy Modeling, which contains the alloy model which allows us to verify consistency of the proposed solution model.

## **2 Overall description**

### **2.1 Product perspective**

The application will be released both as a web application and a mobile application. It will not have to interface with any legacy system or previous developed application, but it will provide APIs for future application development. For every type of user, the application will provide a different interface in order to best answer the particular needs of the different type of users.

### **2.2 User characteristics**

We expect the user to use our application in order to book in a easy way a taxi ride (in case the user is a customer) or to better manage his daily work (in case the user is a taxi driver). The user will need access to internet both for the web application and the mobile application.

### **2.3 Constraints**

#### **2.3.1 Regulatory Policies**

The application will be under two regulatory policies. The former is the privacy policy which will require the customers to accept during registration the personal data treatment document which regulates the privacy in the country in which our city is located, while it will require the taxi drivers to sign a paper module for the same purpose. The latter is the cookie policy which will require users who access the web application to accept the usage of cookies by the system, again according to the policy of the country in which our city is located.

#### **2.3.2 Hardware Limitations**

Internet connection is required for the application to work. Devices which are not able to connect to the internet will not be able to run the application.

### **2.4 Assumption and dependencies**

- One user with privileges of application manager is inserted during development of application and cannot be deleted.
- Number of taxis is sufficient to satisfy the needs of the customer in an efficient way.
- Number of drivers who refuse an assignation is relatively small within respect to the total number of taxis available (otherwise the system could collapse).
- In case of advance-booking we assume that 10 minutes are enough for the system in order to find a taxi in the queue which will answer the request of the customer and will arrive on time at the pickup place.



- Notifications are correctly sent and delivered by the system (no notification-loss).
- Taxi-Data manager inserts only users who are actually taxi drivers and the data are correct.
- Taxi-Data manager will not delete any taxi driver who is still in activity.
- Taxi drivers refuse an already accepted request only when they actually have problems with the vehicle, they make accidents or they encounter other sorts of problem which will not allow them to complete the assigned job. In other words, drivers do not fake accidents.
- Taxi drivers can not report as fake-users users that actually show up at the appointment.
- There will be a dynamic timeout (based on the position of the taxi) after which, if the taxi does not arrive at the pick-up place and the driver did not signal any kind of problem or accident, the system automatically re-assigns the customer to a new taxi. We assume such timeout is reasonably long and well dimensioned.
- For security purposes, we assume that reservation code is correctly delivered to the user without leaking any information.

## 2.5 Future possible implementation

We will give access to APIs in order to allow possible future implementation of applications which will interface with myTaxiService to deliver other type of services. Examples could be a service of Taxi Sharing, in which two or more people exploit the same taxi ride to reach their destination, or a service of private transportation (similar to already existing services such as BlaBlaCar or Uber), in which there are no longer taxi drivers, but any private citizen can be a taxi driver himself. Other kind of services, which we did not think about, could be implemented. What we will actually do is, as already said, to provide sufficient APIs to allow other people to build any kind of application which could interface with myTaxiService.

## 3 Specific Requirements

### 3.1 Functional Requirements

#### 3.1.1 Allow registration of a new user to the system

- The system must provide the sign up functionality.
- The system must send an e-mail to the address specified by the registering user for confirmation.
- When confirmation is received the system must add the new user to the user table in the DB.

#### 3.1.2 Allow log-in to already existing users

- The system must provide an input form in which the user can insert his login data.
- The system must check the correctness of the inserted data. (Right email matched with the right password)
- If log-in data are correct, the system must redirect the user to his personal page.

#### 3.1.3 Allow users to modify his personal information

- The system must provide an interface for the modification of personal data by the users.
- The system must update the modified information in the DB.

#### 3.1.4 Allow insertion of new taxis and taxi drivers by the Taxi-Data manager

- The system must provide an input form for the Taxi-Data manager to insert new taxis and taxi drivers.
- In case a new taxi driver is added, the system must generate a unique password and send it to the taxi driver.
- The system must add the new items to the respective tables in the DB.

#### 3.1.5 Allow Taxi-Data manager to delete taxis that no longer exist or taxi drivers who no longer provide the service

- The system must provide an interface which allow the deletion of taxis and drivers.
- The system must go in the DB and actually delete the item selected by the Taxi-Data manager.

**3.1.6 Allow Application manager to manage all other users (insertion of new TaxiData manager, change privileges of some users, etc.)**

- The system must provide an interface which allows the Application Manager to modify the data he wants and he is allowed to modify.
- The system must perform the changes on the DB.

**3.1.7 Allow customer to real-time book a taxi ride and notify the assigned taxi driver**

- The system must provide an input form which allows the customer to insert the information about his position.
- The system must assign from the taxi queue of the zone in which the customer is (or, if no taxis in that zone are free, from the other closest zone) a taxi cab to the request.
- The system must send a notification to the driver of the assigned taxi cab.

**3.1.8 Allow customer to book in advance a taxi ride (advance-booking) and notify the assigned taxi driver**

- The system must provide an input form which allows the customer to insert information about the starting point, the destination point and the time at which he will need the taxi.
- The system must save this reservation into the advance-booking table of the DB.
- 10 minutes before the time specified in the reservation, the system must allocate a taxi to answer the request.
- The system must send a notification to the driver of the allocated taxi cab.

**3.1.9 Allow fair management of the queues**

- Whenever a driver ends a ride the system must insert him as last in the queue of the zone where he is.
- The system must place as first in the queue of each zone the driver that in that zone has the highest absolute waiting time (i.e. we see waiting time as a score independent from the zone in which the taxis are. Higher the time a driver has been waiting for a new request to arrive, higher his score is, and thus higher the probability for him to be the first of the queue independently from the zone in which he is).

- When a free taxi changes zone the system must automatically find the gps position of the taxi and insert it in the queue of the respective new zone in the right position.

#### **3.1.10 Allow taxi drivers to either accept or refuse the assignation and notify customer (in case of acceptance)**

- The system must provide an interface to allow the taxi driver to either accept or refuse an assignation.
- The system must send a notification to the customer related to the request when a taxi driver accepts the assignation.
- The system must reinsert the driver as last in the queue if the request is rejected.

#### **3.1.11 Allow taxi drivers to refuse an accepted assignation in case of accidents, problems with the vehicle or other sort of problems, and notify customer with the new assignation**

- The system must provide an interface for the taxi drivers to cancel an assignation in case of accidents or other problems.
- The system must automatically search for a new taxi to answer the request of the customer.
- The system must send a notification to the customer with the code of the new assigned taxi.

#### **3.1.12 Allow customer to cancel a reservation**

- The system must enable customers to access their previous reservation.
- The system must allow to cancel previous reservation when no longer needed.

#### **3.1.13 Allow taxi drivers to report fake-users in case a user who has booked a taxi and did not canceled his request does not show up**

- The system must provide an interface which allows the drivers to report customers who did not cancel their reservation but they did not show up at the appointment place.
- The system will need a function that keeps counts of how many times a user is reported as fake.
- After a certain fixed threshold the system must notify the application manager who will take the appropriate countermeasures.

- The system will automatically reinsert the driver into the queue as if the assignment was never created.

**3.1.14 Do not allow customer to take the wrong taxi or taxi driver to pick up the wrong customer**

- With the notification of assignation the system must also send to the customer a unique code associated to the taxi which is going to provide the ride.
- The system must allow taxi driver to verify the correctness of the code.
- The system must check that the code inserted by the taxi driver matches the code generated by the system itself for that particular ride.

**3.1.15 Allow the development of applications that need to interface with myTaxiService through APIs**

- The system must provide programmatic interfaces to allow other application to access user data (only read, no modification) and user's functionalities (e.g booking taxis).
- The system must provide programmatic interfaces to allow users to use the same log-in credentials in other systems which interface with our system.
- The system must provide programmatic interfaces to allow other applications to access the list of reservation (only read, no modification)

## 3.2 Non-functional requirements

### 3.2.1 User Interface

We present now some mocks to give an idea of how the web application and the mobile application will be structured.

**3.2.1.1 Web application Homepage** In the home page users can either sign up or log in.

A browser window mockup showing the homepage of 'MyTaxyService'. The address bar displays 'http://www.mytaxyservice.com'. The page has a yellow header bar. Below the header, the text 'Welcome to MyTaxyService page' is followed by 'The application to better manage your taxi ride into town'. There are two main sections: 'Login' and 'Register', separated by an 'or'.

**Login**

Email Address

Password

**Login**

**or**

**Register**

Surname

Name

Email Address

Password

Confirm password

**Register**

**3.2.1.2 Web application Customer Homepage** This mock shows the page the system loads for the customer after the log in.

A browser window mockup showing the customer homepage of 'MyTaxyService'. The address bar displays 'http://www.mytaxyservice.com/myPage/'. The page has a yellow header bar with 'My booking', 'Account setting', and 'Logout' links. The main content area is titled 'First step' and contains a 'Set my start position' text input field. Below the input is a map with a location pin. At the bottom, there are two yellow buttons: 'Istant booking' and 'Advance booking'.

My booking Account setting Logout

**First step**

Set my start position

**Istant booking** **Advance booking**

**3.2.1.3 Web application Advance Booking specifications** In this mock we show how the navigation proceeds in case of Advance-booking.

A browser window showing the 'Second step' of the advance booking process. The URL is 'http://www.mytaxyservice.com/myPage/'. The navigation bar includes 'My booking', 'Account setting', and 'Logout'. The main content area is titled 'Second step' and contains a text input field labeled 'Set my arrival position'. Below this is a map with a location pin. Under the map, there is a 'Time of booking' section with a date picker set to '4/22/2012' and a time picker set to '8:40'. A yellow 'Book' button is at the bottom.

**3.2.1.4 Web application Waiting Page after Request Acceptance** This mock shows the page the customer faces when his request is accepted.

A browser window showing the 'Your request has been accepted' page. The URL is 'http://www.mytaxyservice.com/myPage/'. The navigation bar includes 'My booking', 'Account setting', and 'Logout'. The main content area is titled 'Your request has been accepted' and 'Stay around this position'. Below this is a map with a location pin. Under the map, there is a 'Waiting time: 3 minutes', 'Taxi plate: WY256', and 'Request Code: HM-247'. A red 'Cancel booking' button is at the bottom, followed by the text 'Many cancellation lead signal'.

**3.2.1.5 Taxi-Data manager Web Application Homepage** In this mock we show how it is supposed to appear the homepage of a Taxi-Data manager user.

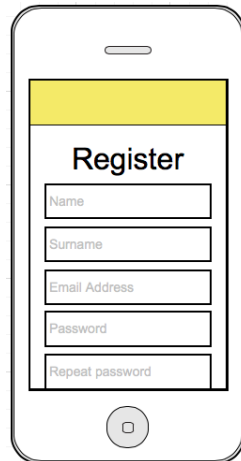
A browser window mockup showing a registration form. The browser's address bar contains the URL `http://www.mytaxyservice.com/myPage/`. The page has a yellow header bar with the text "Account setting" on the left and "Logout" on the right. The main content area is titled "Registration taxi account" and contains the following form fields: "First Name", "Second name", "Driving license code", "Taxi plate", "email address", and "Date of birth". The "Date of birth" field is a date picker showing "4/22/2012". Below the fields is a yellow "Register" button.

**3.2.1.6 Login in the Mobile Application** The following mock represents the Login page, which is also the home page of the mobile application.

A mobile application screen mockup showing a login page. The screen has a yellow header bar. Below it, the title "Login" is centered. There are two input fields: "Email Address" and "Password". Below these fields is a yellow "Login" button. At the bottom of the form area is a blue "Register" link. The screen is framed by a grey border representing the mobile device, with a home button at the bottom.

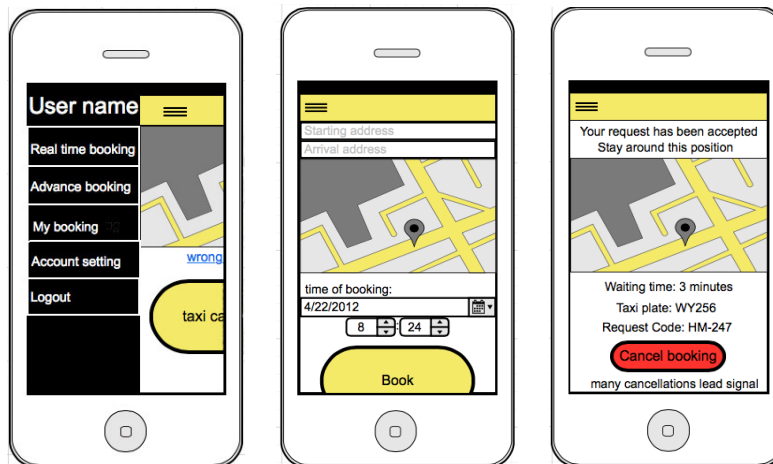


**3.2.1.7 Registration in the Mobile Application** This mock shows the registration form for the Mobile Application.



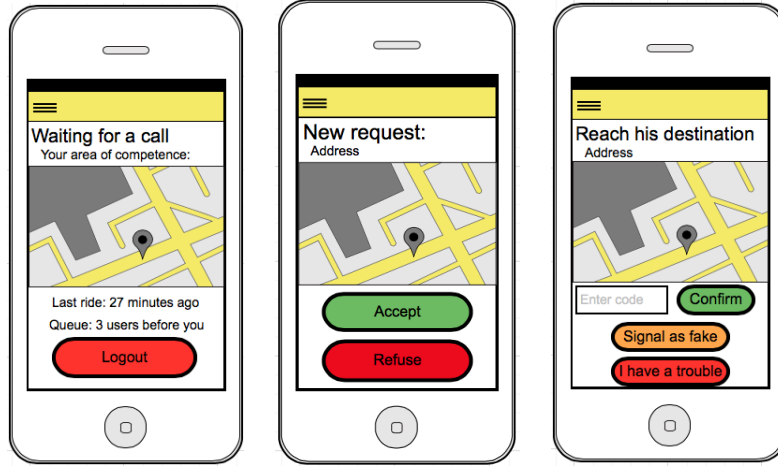
A mockup of a mobile application registration form. The form is titled "Register" and is displayed on a smartphone screen. It features a yellow header bar. Below the header, there are five input fields: "Name", "Surname", "Email Address", "Password", and "Repeat password". The form is centered on the screen, and the smartphone has a home button at the bottom.

**3.2.1.8 Advance-Booking in the Mobile Application (Customer side)**  
With the next three mocks we will show what a customer should see when he books in advance.



Three mockups of a mobile application showing the advance booking process. The first mockup shows a menu with options: "User name", "Real time booking", "Advance booking", "My booking", "Account setting", and "Logout". The second mockup shows a map with a starting address, arrival address, and a "Book" button. The third mockup shows a confirmation screen with the text: "Your request has been accepted Stay around this position", "Waiting time: 3 minutes", "Taxi plate: WY256", "Request Code: HM-247", and a "Cancel booking" button.

**3.2.1.9 Taxi driver management in Mobile Application** We finally show some mocks that will help to understand how drivers can manage their work on their mobile phone thanks to myTaxiService.



### 3.2.2 API Interfaces

For retrieving the position of taxis, the time needed to reach a customer, the position of a customer etc. through GPS, the application will need to interface with the Google Maps APIs. These are open APIs and the most common used for providing services of position retrieval and for previsions on the time duration of a given itinerary. Furthermore, the system will need to interface with a Data Base Management System which is now not specified, because it will be chosen at development time.

## 3.3 Scenarios

We are now going to list some scenarios which could help in getting a better idea of how the application is going to work.

### 3.3.1 First Scenario

It is Friday (16th of October), and on Sunday (18th of October) Ms. White needs to go to the airport. She does not have a car and no one of her friends can help her to get to the airport. However she heard about the new application for taxi service the city government has launch in the previous week, and, though she is not an abitual taxi customer, she is willing to try this new service. Therefore she downloads the application on her mobile phone, and, after the installation, she enters the home page. Here she can access the registratin form, in which she can fill in her data. After submission and confirmation she is ready to log in and start to use the application. After the log in procedure, she can access to advance reservation thanks to the curtain menu on the left. Therefore, she

access the curtain menu, she touches “Advance booking”, she activates the GPS of her smartphone, and she fills in the form for booking. She sets as starting point her home (which is her current position), as destination the airport, the date at the 18th of October and the time at 8.00 am. She touches the button “Book” and then she can log out from the application. On the 18th of October, Damian, a taxi driver (taxi plate: DB 711 BP), is notified on his phone at 7.50 am that Ms. White has booked a taxi for 8.00 am and he accepts the request. Ms. White is immediately notified that the taxi with plate DB 711 BP is coming to pick her up and that the code of her request is L0V3. When the taxi arrives she gives her code to Damian who enters the code in the confirmation form. Then they leave for the airport. Once arrived at the airport, Damian pushes the “end ride” button and gets his money.

### **3.3.2 Second Scenario**

Joe Villain is a taxi driver who wants to access the privileges of the new myTaxiService application in order to better organize his work. He has to compile a paper form and submit it to the municipal office which manages the taxi drivers in the city. Here one of the office workers, John Beary, who can access the system as a Taxi-Data manager, receives the paper document with all the information correctly compiled. He then accesses the system with his credential, fills in the form to add a new taxi driver, submits such form and logs out. A confirmation mail with the password to access the system is sent to Joe who can now start to use myTaxiService to manage his work.

### **3.3.3 Third Scenario**

Mr. Little Jack spent the night at a music concert, and now he needs to go back home. However, his physical conditions after such event do not allow him to take the car to drive back safely to home. Luckily, just that afternoon, he downloaded myTaxiService, thus he does not need to drive himself home. He just needs to book a taxi right away and he will be safely home. Thus he accesses the application from his smartphone, he touches on the “instant taxi call” button, and he waits until a taxi confirms his request and comes to pick him up to bring him home.

### **3.3.4 Fourth Scenario**

Matt Knoll is a young taxi driver who likes to drive fast in order to pick as many clients as possible. He is waiting for a new request to come, when the system contacts him and asks him if he could pick up the customer Ann Calves in a street which is not that far from the place where he is. He accepts of course, but he is too confident, and by passing over a red light, he ends up hitting in the back another car, so that he is forced to cancel the previous confirmation. Thus, he accesses the application on his mobile, and the window for the pick-up confirmation is shown to him. He presses the button “I have troubles”, and the request of Ann is automatically reassigned to another taxi driver.

### **3.3.5 Fifth Scenario**

Lorence Trill is a taxi driver who gets contacted by myTaxiService to pick up Mr. Jam Pi at the train station of the city and bring him to the theater. Lorence arrives at the station and start looking for the man he needs pick up. There are many taxis and many cars so it might be possible that the man who sent the request could have trouble in finding him. However after 30 minutes no one has shown up yet, and an angry Lorence is obliged to observe that maybe someone put up joke and Mr. Jam Pi probably does not even exist. He access the application on his mobile, and, in the pick-up confirmation window, he touches the button “signal as fake”. He then re-enters the queue as if Mr. Jam Pi never reserved for a ride.

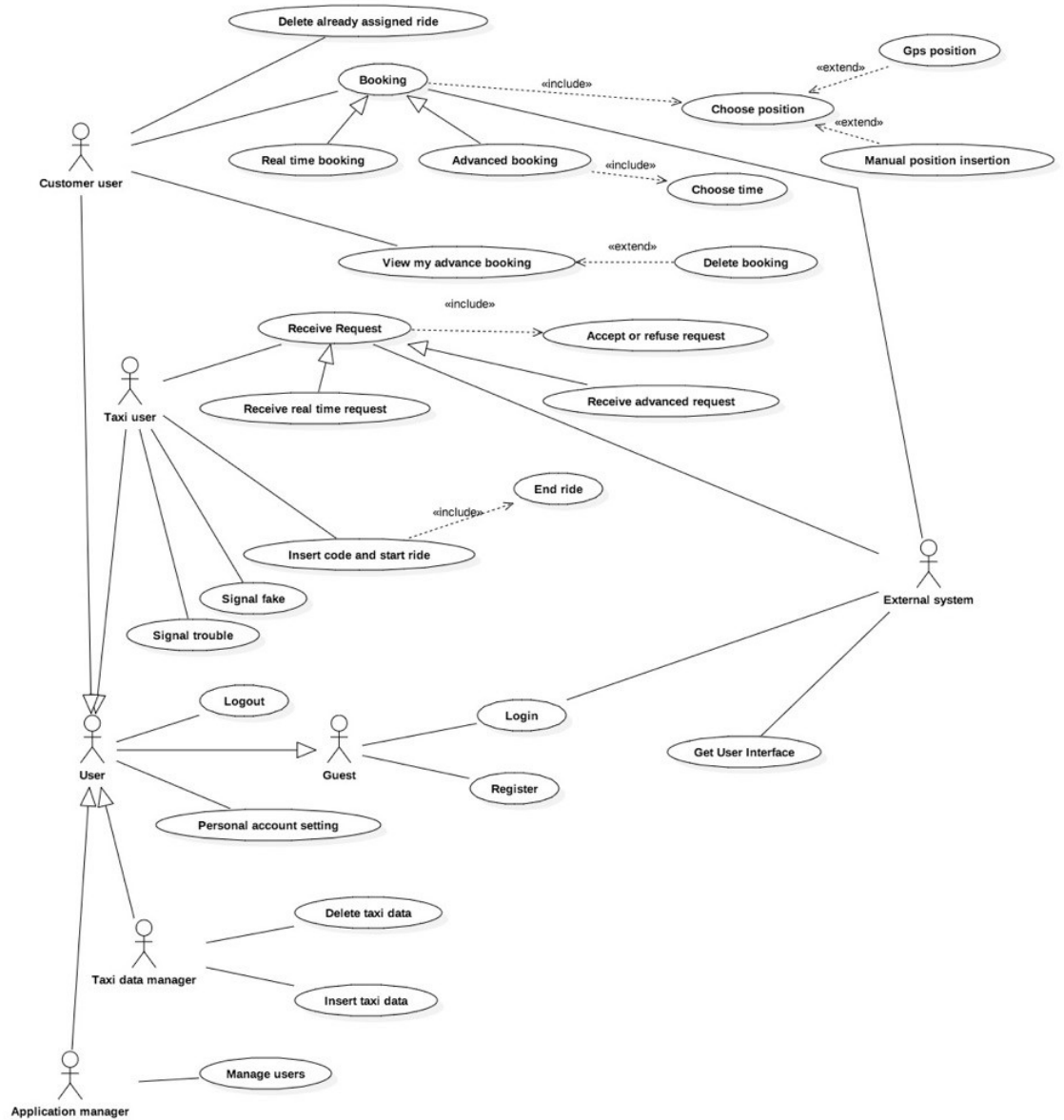
### **3.3.6 Sixth Scenario**

Mr. Andrew Josfate is a very smart and inventive man. When he found out about myTaxiService, he had the idea of implementing a new application for Taxi Sharing, which will allow more customers who are willing to go to the same place to share the same ride and divide the costs. However he does not want to build a new application from scratch and he would like to interface with myTaxiService to use some of its features. Luckily, myTaxiService provides APIs to let othe applications to interface with itself. Therefore, Mr. Josfate accesses the documentation about myTaxyService and finds out that it provides access to user data and functionalities, it allows to reuse the login procedure and to access the table of the reservations. Mr. Josfate is very happy because he will just need to implement the system of communication between users and his application will be ready to run and make him rich.

## 3.4 UML Models

### 3.4.1 Use Case Diagram

We present now the Use Case Diagram for the whole application derived from previous stated requirements and from the described scenarios. In the next sections we are going to explain more in details the Use Cases and the flow of events associated to them.



We are now going to provide a more detailed description of the most significant use cases. As a disclaimer, whenever we will use words such as “button” or “input form” we are not making any design decision. We will use them in order to give a more clear vision of how the system could work.

#### **3.4.1.1 Register Use Case**

- Name: Register
- Actors: Guest
- Entry Condition: The guest is not already registered to the application.
- Flow of Events:
  - The guest accesses the Homepage.
  - The guest clicks the “Registration” button.
  - The guest fills in a form in which he has to insert: his last name, his first name, his email address and a password.
  - The guest clicks on “Register”.
  - The system shows to the guest the “Waiting for Confirmation Page”.
  - When the guest confirms registration the system redirects him to his personal homepage.
- Exit Condition: Registration is successful.
- Exceptions:

#### **3.4.1.2 Log in Use Case**

- Name: Log in
- Actors: Guest
- Entry Condition: The guest is already registered to the application.
- Flow of Events:
  - The guest accesses the Homepage.
  - The guest fill in the login form with his email and password.
  - The guest clicks the “Log in” button.
  - The system checks if the data matches an already registered account information.
  - The system let the guest to enter in his profile (Guest becomes User).
- Exit Condition: The system shows to the User his personal Homepage.
- Exceptions: The Guest fills in the form with wrong information, therefore the system shows an error message.

#### 3.4.1.3 Advance Booking Use Case

- Name: Advance Booking
- Actors: Customer user
- Entry Condition: Customer user has successfully logged in.
- Flow of Events:
  - The Customer user clicks on “Advance Booking” in his personal Homepage.
  - The system shows to the user a form for the reservation.
  - The Customer user fills in the form with information about the place, the date and the time on which he will need a taxi.
  - The Customers click on the “Confirmation” button.
  - The systems shows to the user that the operation was successful.
- Exit Conditions:
  - The system redirect the Customer user to his personal Homepage.
  - 10 minutes before the reservation time the system finds a taxi and send a notification to the Customer user.
- Exceptions: The user does not fill in all the mandatory in fields; in such case the system shows an error and does not complete the operation.
- Notes: The Real-time Booking case is just the same, the only differences is that date and time are not specified by the user and that as exit condition we should add the fact that the system starts searching for a taxi to answer the request.

#### 3.4.1.4 Receive Request Use Case

- Name: Receive Request
- Actors: Taxi user
- Entry Conditions:
  - A customer user has booked a taxi and the system has assigned him one.
  - The Taxi user is logged in.
- Flow of Events:
  - The Taxi user is notified by the system that he was assigned to a new customer.

- The system propose to the Taxi user the choice of either accepting or refusing the job.
- The Taxi user accepts or refuse the assignation.
- In case of acceptance, the system will show to the Taxi user the page in which he will be able to insert the verification code sent to the Customer User.
- In case of refuse, the system will redirect the Taxi user to his personal Homepage.
- Exit Conditions:
  - In case of acceptance the system sends a notification with the taxi plate and the verification code to the user.
  - In case of refuse the system will put the Taxi user as last in the queue of the zone in which the taxi user is.

#### **3.4.1.5 Insert Code and Start Ride Use Case**

- Name: Insert Code and Start Ride
- Actors: Taxi user
- Entry Conditions:
  - The Taxi user has accepted a request of a Customer user.
  - The Customer user has provided the verification code.
- Flow of Events:
  - The Taxi user insert the verification code.
  - The Taxi user clicks the “Verify” button.
  - The system verifies that the code matching is correct.
  - The system sends confirmation to the Taxi user.
- Exit Condition: The Taxi user starts the ride.
- Exceptions: Verification code does not match, therefore the Taxi user must look for the right customer.

#### **3.4.1.6 Insert Taxi Data Use Case**

- Name: Insert Taxi Data
- Actors: Taxi data manager
- Entry Condition: The Taxi data manager has successfully logged in.

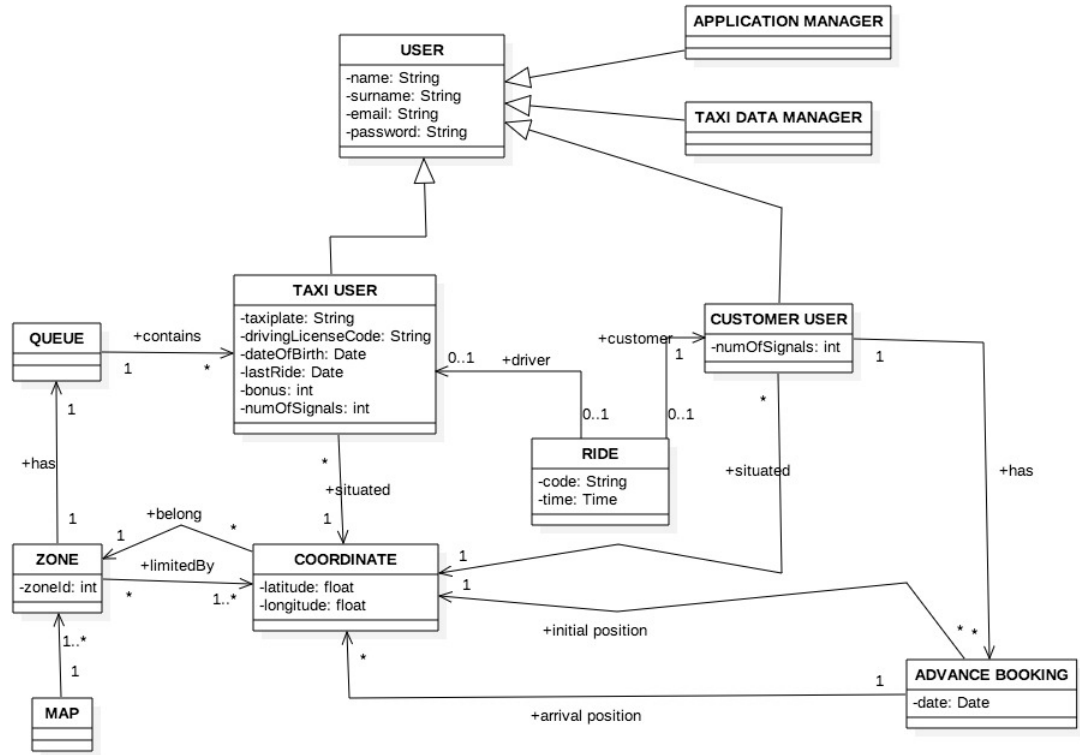


- Flow of Events:
  - The Taxi data manager fill in the form for adding a new taxi to the system.
  - The Taxi data manager clicks on the “Register” button.
  - The new taxi driver is inserted in the system which shows confirmation of success to the Taxi data manager.
- Exit Condition: The system redirects the Taxi data manager to his personal Homepage.
- Exceptions: The Taxi data manager does not fill all the mandatory fields. In such case, the system will report the error.

#### **3.4..1.7 Get User Interface Use Case**

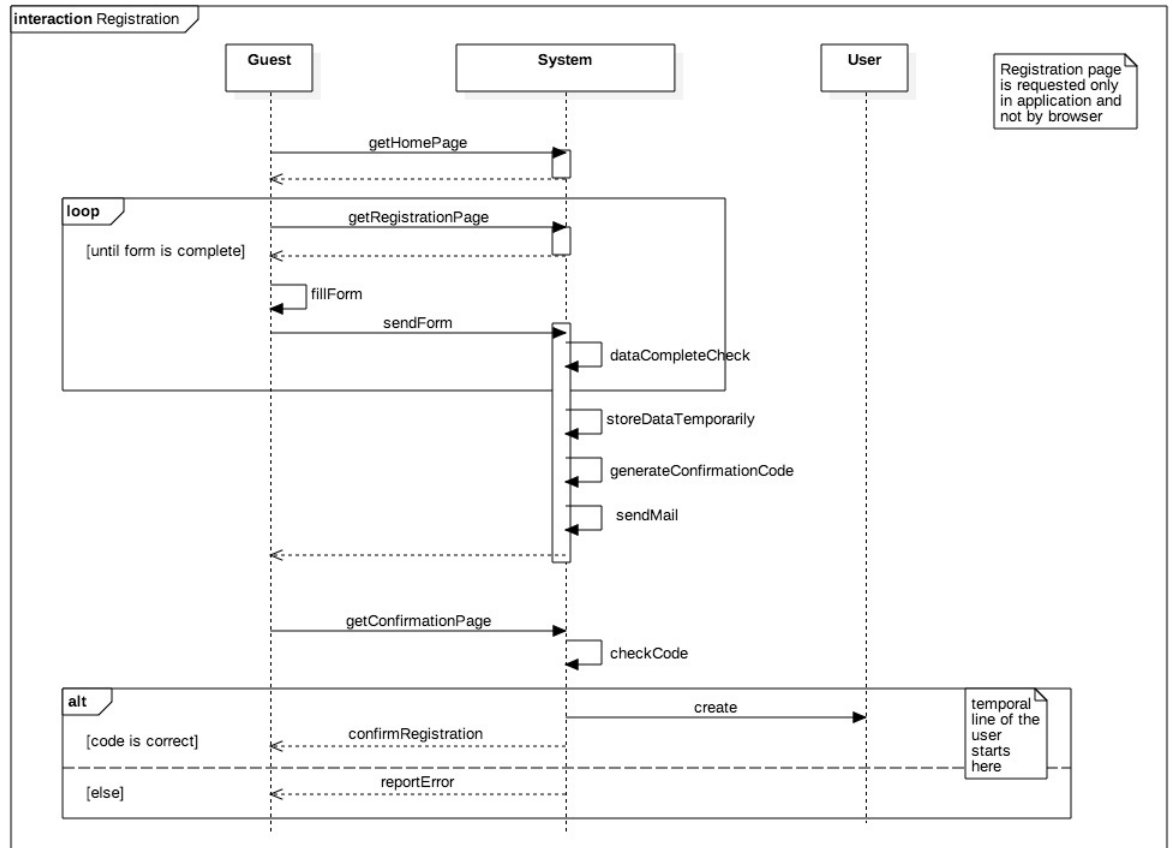
- Name: Get User Interface
- Actors: External System
- Entry Conditions: The external system wants to use one or more functionalities of a user and asks the access to our system.
- Flow of Events:
  - Our system must require the log in of the user of which the external system wants to access the functionalities.
  - The external system must successfully log in to our system.
  - Our system must provide the right interfaces to guarantee access to user’s functionalities.
- Exit Condition: The external system is able to use the requested functionalities.
- Exceptions: Log in fails, in such case our system must not allow access to user’s functionalities.

### 3.4.2 Class Diagram

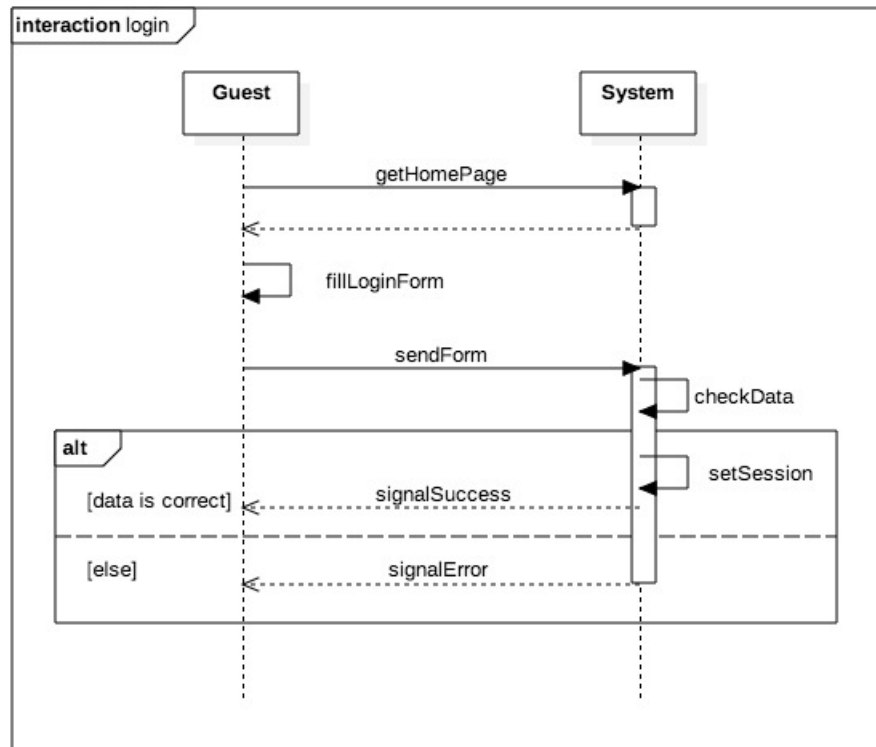


### 3.4.3 Sequence Diagrams

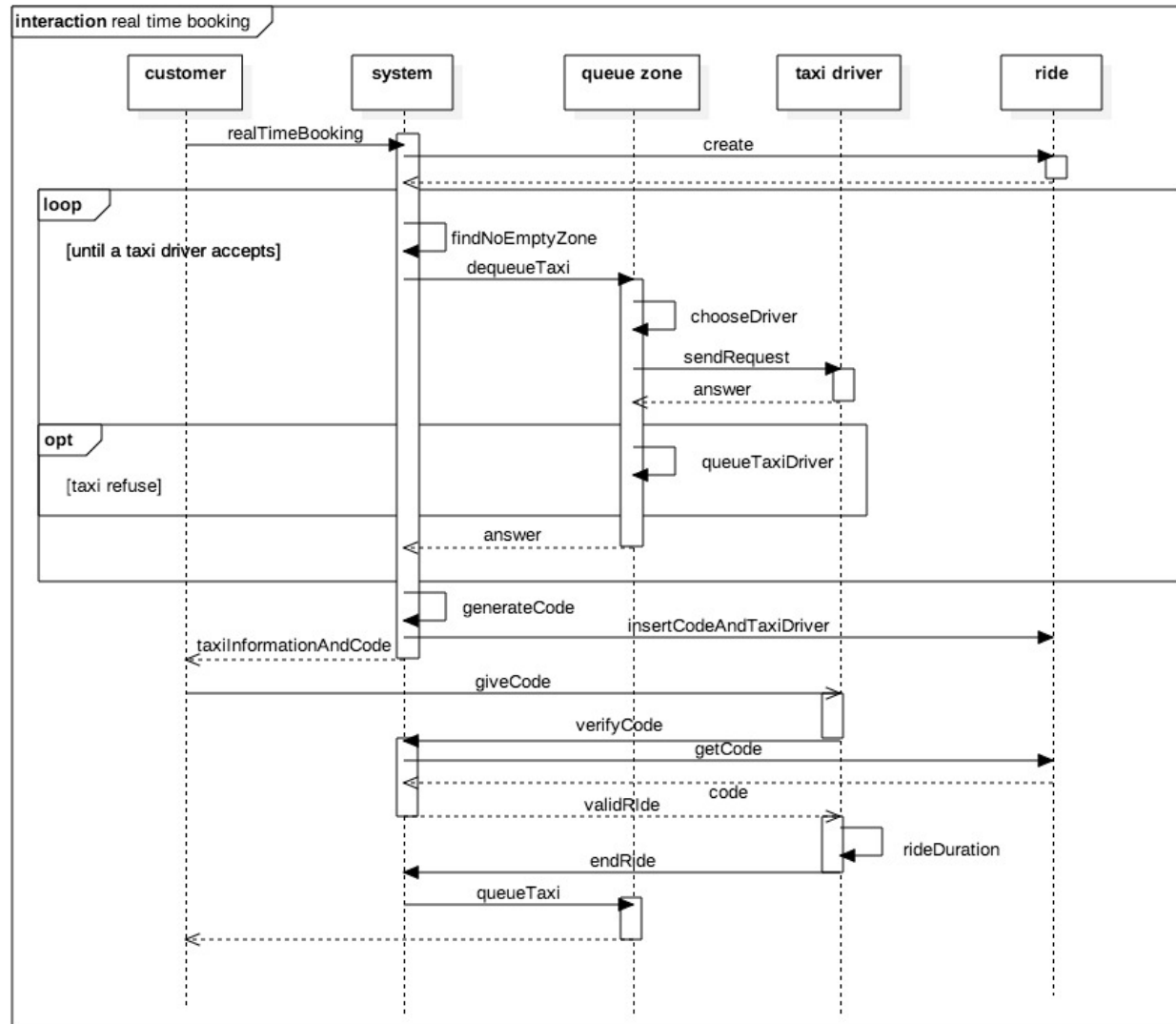
#### 3.4.3.1 Registration



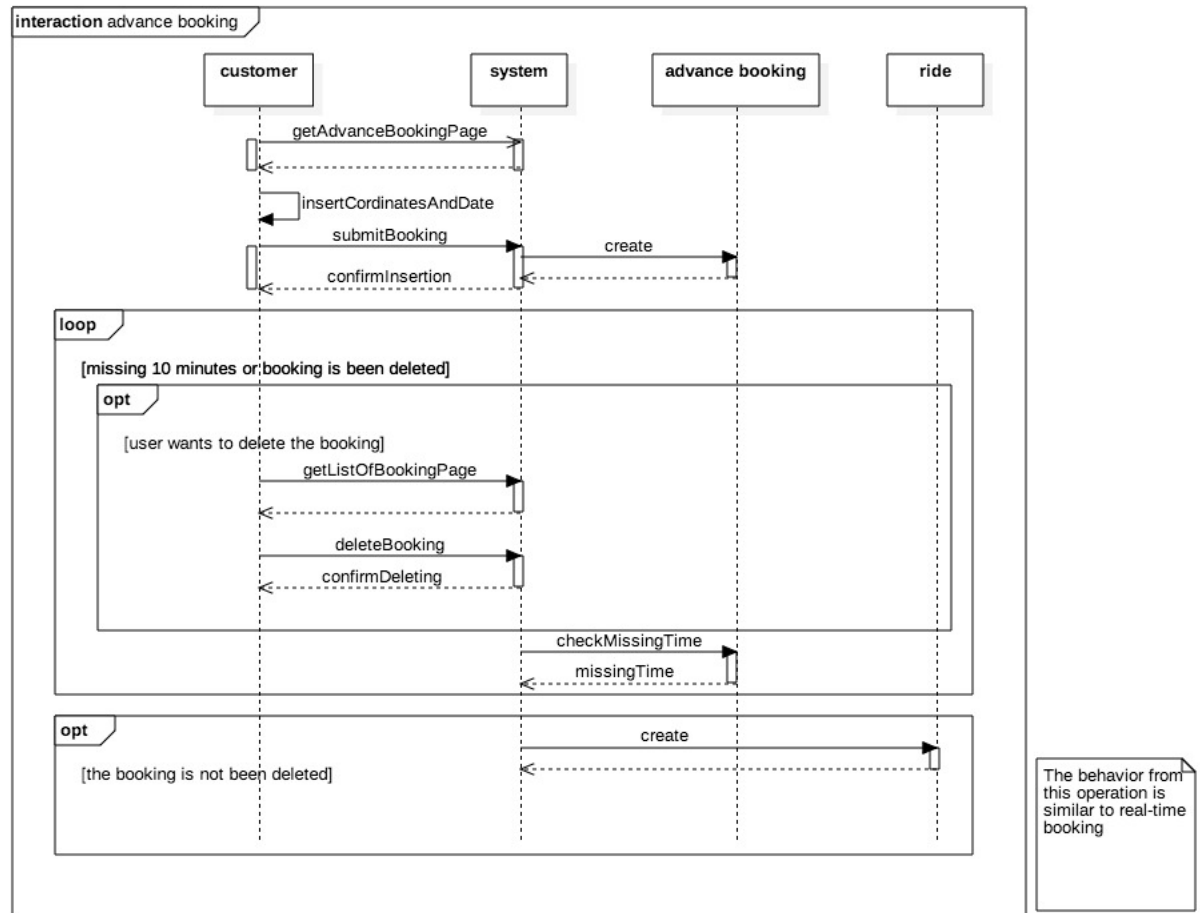
### 3.4.3.2 Log in



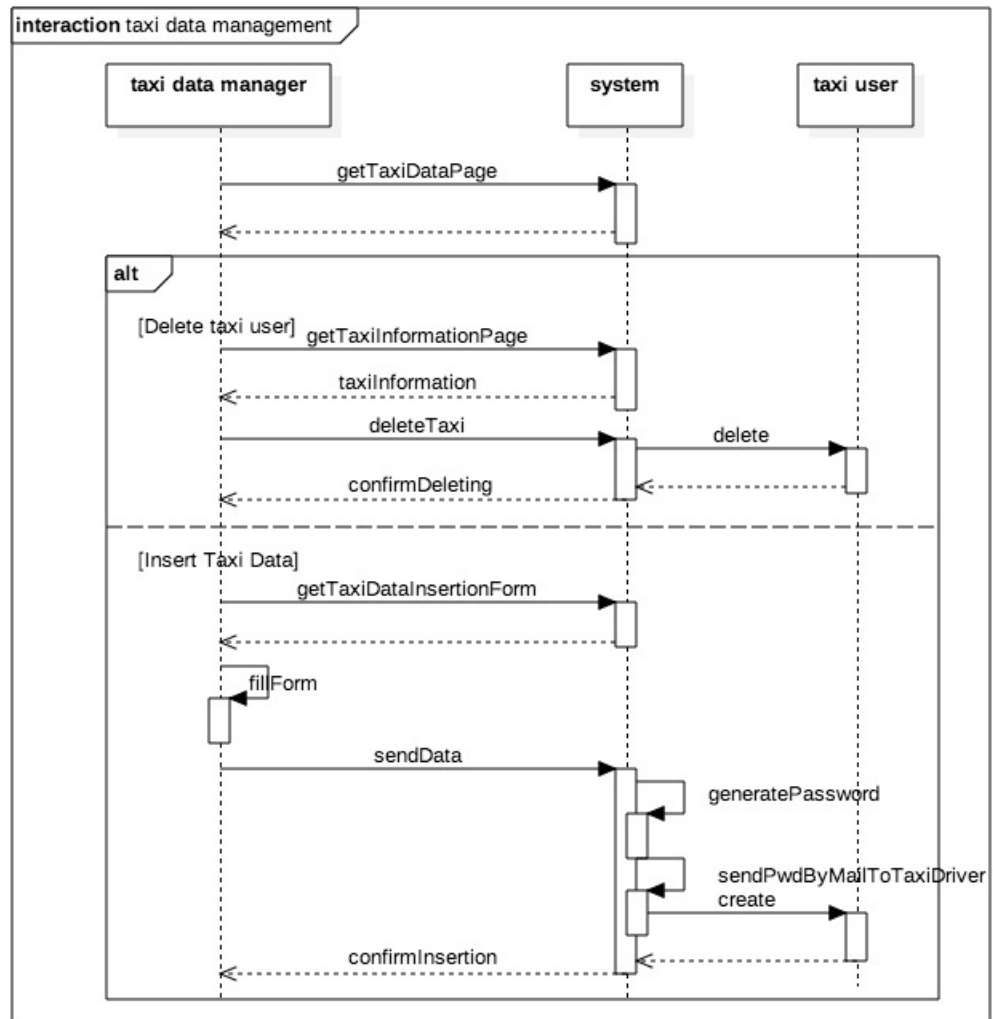
### 3.4.3.3 Real-time Booking



### 3.4.3.4 Advance Booking

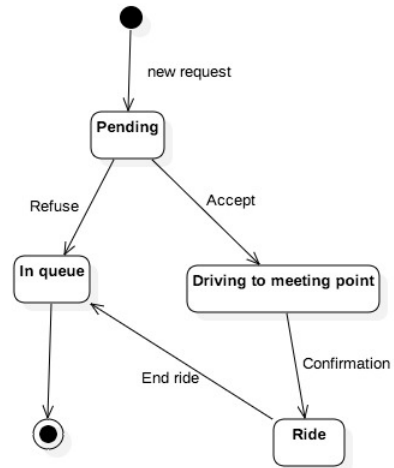


### 3.4.3.5 Taxi Data Management

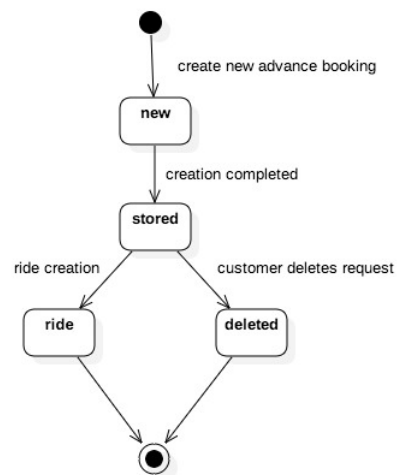


### 3.4.4 State Charts

#### 3.4.4.1 Taxi Driver Class



#### 3.4.4.2 Advance Booking Class





## 4 Alloy Modeling

We have used Alloy Analyzer to specify the properties of our application starting from the Class Diagram. We are now going to present the alloy code and some of the instances obtained through the show predicates.

### 4.1 Alloy Code

```
sig Integer{}

sig Float{}

sig Strings{}

abstract sig Bool{}

one sig True extends Bool{}

one sig False extends Bool{}

abstract sig User{
  name : one Strings,
  surname : one Strings,
  email : one Strings,
  pwd : one Strings,
  logged : one Bool
}

sig Date{
  day : one Integer,
  month : one Integer,
  year : one Integer
}

sig Time extends Date{
  hour : one Integer,
  minute : one Integer
}

sig Coordinate{
  latitude : one Float,
  longitude : one Float
}
```

```

sig Zone{
  coordinates : set Coordinate,
  queue : one Queue
}

sig AdvanceBooking{
  time : one Time,
  startPosition: one Coordinate,
  arrivalPosition: one Coordinate,
  customer: one CustomerUser
}

//FACTS

fact eachZoneHasOnlyOneQueue{
  //Each zone has only one queue
  all z1 : Zone, z2: Zone | z1.queue = z2.queue implies z1 = z2
}

fact eachQueueBelongToOneZone{
  //Each queue belong to only one zone
  all q1: Queue | #q1.^queue=1
}

fact activeTaxiNoQueue{
  //Riding taxis do not belong to any queue
  all q1: Queue, t1:TaxiDriver, r1:Ride | r1.taxiDriver = t1 implies t1.zoneQueue != q1
}

fact oneTaxiRideOnly{
  //A taxi driver can only have one active drive per time
  all t1:TaxiDriver, r1:Ride, r2:Ride | (r1.taxiDriver = t1 && r1!=r2) implies r2.taxiDriver != t1
}

```

```

fact oneCustomerRideOnly{
  //A ride can only have one customer associated to it
  all c1:CustomerUser, r1:Ride, r2:Ride | (r1.customer = c1 && r1!=r2) implies r2.customer != c1
}

fact TaxiInRideOrInQueueIsActive{
  //All Taxi drivers in the queue are logged in
  all t1: TaxiDriver, q1:Queue | t1.zoneQueue = q1 implies t1.logged in True
  //All riding taxi drivers are logged in
  all r1:Ride | r1.taxiDriver.logged in True
}

fact activeTaxiInRideOrInQueueOnly{
  //All logged in taxi driver are either in a queue or driving a customer somewhere
  all t1: TaxiDriver | t1.logged in True implies #t1.zoneQueue >0 || (some r1:Ride | r1.taxiDriver = t1)
}

fact zoneHasBorder{
  //each zone has a border composed at least by three coordinates
  all z1:Zone | #z1.coordinates>2
}

fact advanceBookingCorrect{
  //An advance booking belongs to the set of advance bookings of the customer associated to the booking itself
  all ab:AdvanceBooking | ab in ab.customer.advanceBooking
  //An advance booking can belong to only one Customer
  all ab:AdvanceBooking | #ab.~advanceBooking = 1
}

fact oneAdvanceBookingMaxOneRide{
  //Each advance booking is associate maximum at one ride
  all ab:AdvanceBooking | #ab.~advance<2
}

```

```

fact rideCustomerEqualToBookingCustomer{
  //Each ride is associated only to an advance booking
  all r:Ride | r.customer = r.advance.customer
}

fact uniquePredecessor{
  //Each node has at most one predecessor node
  all n:Node | #n.~next <= 1
  //Each node can be head of one queue at most
  all n:Node | #n.~head <= 1
  //A node does not have a predecessor if and only if it is the head of one queue
  all n:Node | #n.~next = 0 iff #n.~head = 1
}

fact taxiDriverOnlyInANode{
  //Each Taxi Driver belongs at most to one queue
  all t:TaxiDriver | #t.~taxi <= 1
  //All the nodes element associated to taxi drivers start from a queue
  all t:TaxiDriver | #t.~taxi.~head = 0 implies t.~taxi.^(~next).~head = t.zoneQueue
  all t:TaxiDriver | #t.~taxi.~head = 1 implies t.~taxi.~head = t.zoneQueue
}

fact eachQueueHasAHead{
  //Each queue has a head
  all n:Node | #n.^(~next).~head = 1 or #n.~head=1
}

fact noSelfNode{
  //Every node does not have itself as successor
  all n:Node | n.next!=n
}

```

```

//ASSERTIONS

//Verifies that there exists no advance booking which is not associated to a customer
assert noReservationWithoutCustomer{
  no a:AdvanceBooking | (no c:CustomerUser | a.customer=c)
}

check noReservationWithoutCustomer

//Verifies that adding a new Advance Booking in the set of Advance Bookings of a customer is done correctly
assert addAdvanceBookingCorrect{
  all ab:AdvanceBooking, c1, c2:CustomerUser| addAdvanceBooking[ab,c1,c2] implies c1=c2
}

check addAdvanceBookingCorrect

// PREDICATES

pred show{
}

//Adds a new Advance Booking to the set of Advance Bookings of a customer
pred addAdvanceBooking(ab:AdvanceBooking,c1,c2:CustomerUser){
  #c1.advanceBooking=0 implies (c1=c2 and c2.advanceBooking = c1.advanceBooking + ab)
  else c2.advanceBooking = c1.advanceBooking + ab
}

//Creates a world in which exists some advance booking associated to some ride
pred showRideCreation{
  some ab:AdvanceBooking, r:Ride | r.advance=ab
}

//Creation of a world with at least one taxi in queue for each zone
pred showQueue{
  all z: Zone| some t:TaxiDriver | t=z.queue.head.taxi
}

run showQueue for 3 but exactly 3 Zone
run show for 3 but 2 Ride, 10 User, 1 ApplicationManager, 1 TaxiDataManager, 6 TaxiDriver, 6 Node

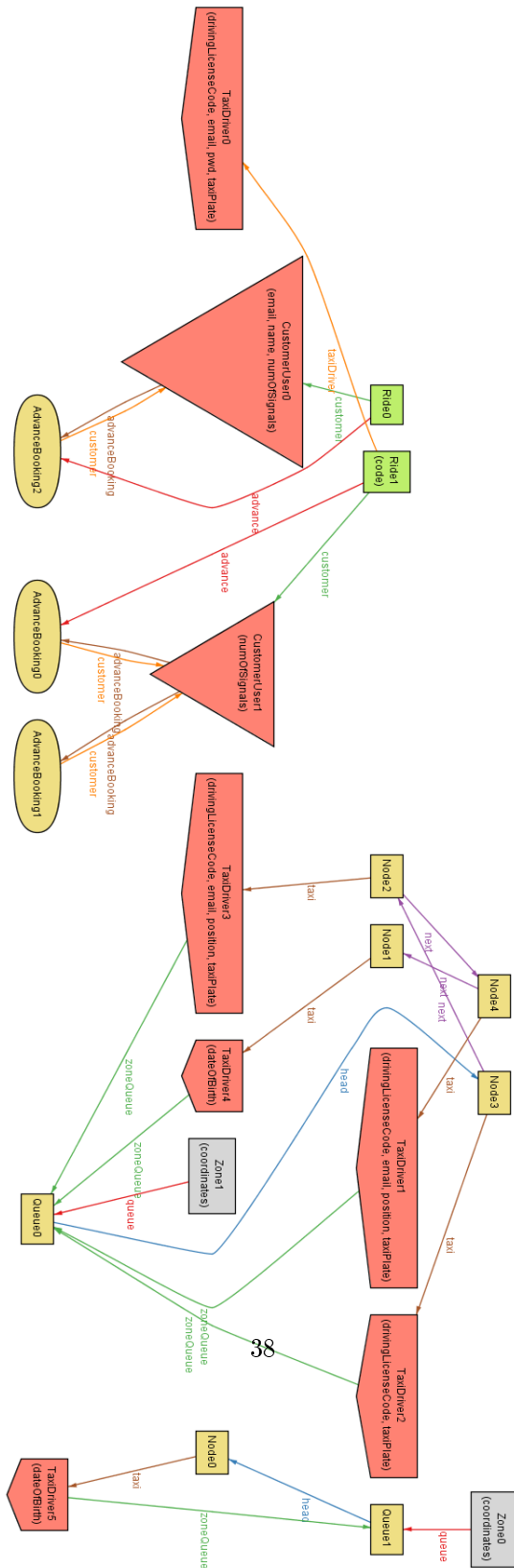
```

Here we present the results of the analysis

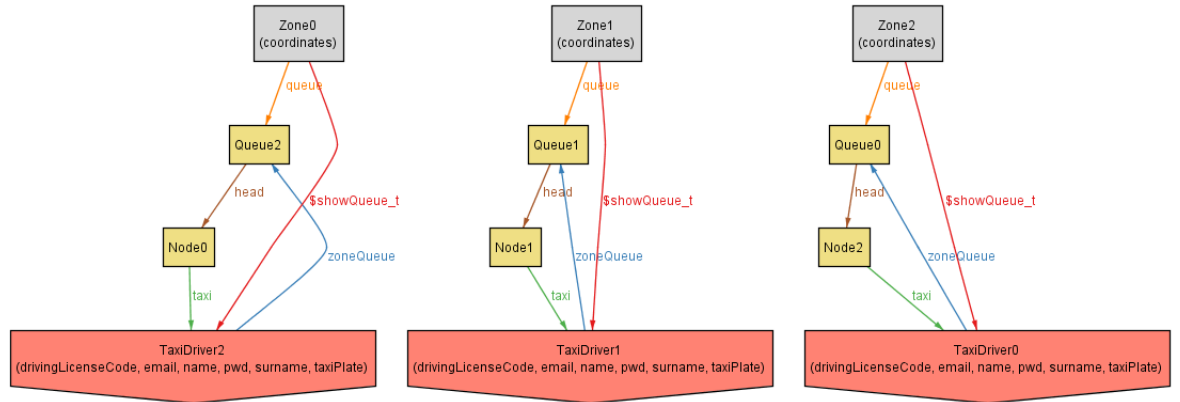
**4 commands were executed. The results are:**

- #1: No counterexample found. noReservationWithoutCustomer may be valid.
- #2: No counterexample found. addAdvanceBookingCorrect may be valid.
- #3: **Instance found.** showQueue is consistent.
- #4: **Instance found.** show is consistent.

### 4.2.1 General World



### 4.2.2 Queue World



## 5 Used Tools

- LyX (<http://www.lyx.org/>): to redirect and format this document.
- Star UML ([staruml.io](http://staruml.io)) : to draw Use Case Diagram, Class Diagrams, Sequence Diagrams, State Charts.
- Alloy analyzer (<http://alloy.mit.edu/alloy/>): to prove consistency of our model.

Hours spent for redacting the document:

- Damiano Binaghi: 35h
- Giovanni Zuretti: 35h