



# **Requirements Analysis And Specification Document**

Authors:

Giovanni Bucci

Riccardo De Togni

# Summary

<b>1. Introduction</b>	<b>3</b>
1.1 Description of the Problem	3
1.2 Glossary	3
1.3 Goals	4
1.4 Domain Properties	5
1.5 Assumption	5
1.6 Proposed System	5
1.6.1 Product Perspective	5
1.6.2 Constraints	6
1.7 Possible Future Implementations	6
1.8 Stakeholders	7
<b>2. Specific Requirements</b>	<b>7</b>
2.1 Functional Requirements	7
2.2 Non Functional Requirements	10
2.2.1 Look Requirements	10
2.2.2 Usability	18
2.2.3 Privacy Requirements	18
<b>3. Software Design - UML</b>	<b>18</b>
3.1 Actors Identifying	18
3.2 Possible Scenarios	18
3.3 Actor: Guest	20
3.4 Actor: User	25
3.5 Actor: Taxi Driver	30
3.6 Actor: Administrator	35
3.7 Domain Model	41
3.7.1 Domain Class Diagram	41
3.7.2 Main dynamics of the system	42
3.7.3 State chart Diagrams	47
<b>4. Model Coherence Analysis</b>	<b>49</b>
4.1 Alloy Code	49
4.2 Generated World	52
<b>5. Tools</b>	<b>53</b>

## **1. Introduction**

### **1.1 Description of the Problem**

The aim of the project is to improve taxi services of large cities. The objective is on one hand to simplify the access by users, making bookings easier and faster, and on the other hand to grant fairness in queue assignment of taxis.

The system should be able to register two main consumer categories: User and Taxi Driver. A Taxi Driver has to be registered in order to access the service, after so it can communicate its availability and accept or deny a call.

An unregistered User could call a taxi just giving its identification data, without a regular access and even without a formal registration.

A Registered User could obviously call a taxi, once it accessed the system, and also book it in advance, providing starting and arriving point.

This is the peculiarity of this product, because it implements a feature that does not exist in the actual taxi service. In fact, it is impossible to reserve a taxi well in advance without using an application like the one that will be analysed here.

### **1.2 Glossary**

Before starting to describe in details the project, it is necessary to define some words that will assume a specific meaning during the documentation:

- **GUEST:** a person who has not signed up yet. Guests have no power until they sign up with one exception. If a Guest just want to call a taxi it could simply insert its identification data.
- **USER:** a person that has already signed up as a customer. It could call a taxi, as guest does, but it also could reserve it in advance, compiling a specific form.
- **TAXI DRIVER:** a person who has signed up as a taxi driver. In order to complete its registration it has to provide its identification data and its driving license too.
- **SYSTEM:** the environment formed by the application itself and its features.
- **CITY ZONE:** each city is divided in zones. Every zone has approximatively the same territorial extension, so a city zone is one of the portions of the metropolitan area.
- **QUEUE:** an ordered list of taxi drivers that have previously provided their availability.
- **CALL A TAXI:** the action which can be performed both by guests and user, that consists in asking for a single taxi ride without any advance.

- RESERVE A TAXI: the action that could be performed only by Users. A user can forward the request for a taxi from a specified place to another in advance.
- SERVICE: the service that is provided by the application.
- DENY/DENIAL: when a request is not satisfied. It produce the shifting of the considered taxi driver to the bottom of the queue.
- ACCEPT: when a request, both coming from a reservation or a taxi call, is taken by a taxi driver who assumes the charge to bring passengers to the destination.

### 1.3 Goals

The system will provide the following features, grouped by user category:

- Guest
  - Sign Up into the system
  - Call a taxi
- User
  - Sign Up into the system
  - Log into the system
  - Book a Taxi in advance (Reserve a Taxi)
  - Call a Taxi
- Taxi Driver
  - Sign Up into the system
  - Log into the system
  - Give/Remove availability (Take place into a queue)
  - Respond to a request (Accept or Deny)
- The system should track out the position of each Taxi and User in different moment for each one:
  - When a Taxi driver gives its availability
  - When User request for a Taxi

## 1.4 Domain Properties

Here are some considerations about the system that are essential to ensure consistency:

- A person could have both taxi and user account but it can never be logged in at the same time with different types of account.
- The entire city territory is divided in zone of equal extension
  - Each zone has a related queue of taxis

## 1.5 Assumption

It is necessary to make some assumption in order to build an unambiguous system. These decisions will remain unchanged for the whole documentation:

- Each queue has a First In – First Out (FIFO) policy based on chronological order: when a taxi driver give its availability it will be placed in the last position of its zone queue.
- There is no way to change the position of a particular taxi driver in a queue: at every request the queue will shift of one position.
- After a predetermined time, if the person who requested the taxi fails to appear to the rendezvous point it will be considered as a denial.
- After a predetermined time, if the taxi driver is not responding to a request (is on top of the queue but it does not accept or deny) it will be considered as a denial.
- Taxis after giving its availability will not exit its zone.
- Users can have just one reservation at a time.
- Taxi Driver cannot register twice with the same Taxi License.

## 1.6 Proposed System

### 1.6.1 Product Perspective

The service has to be consumed both through a web application and a mobile app. For this reason it will be implemented a web application, reachable simply by any kind of browser, and a dedicated app which will adapt the user interface of the web application to the mobile devices, but it will not provide any further service.

The system will also provide all needed APIs to grant the possibility for future implementation. Some of them will be listed in this chapter.

### 1.6.2 Constraints

- **Regulatory Policy:**
  - Privacy of all registered people must be granted; it means that the system will save all the data in order to work properly but nothing of these data will be visible or sold to third parts.
  - A taxi driver must have valid driving and taxi license.
- **Hardware Limitation**
  - The application could run both on personal computers and mobile devices, so it is necessary for the customer to have one of those.
  - No particular hardware features are needed: you only need a browser in order to run a web application. The mobile app will be implemented to ensure the broadest compatibility for operating system.
- **Interface with other application**
  - This application has an interface with GPS related Apps, in general with an app that is provided with some localization protocol. This is in order to calculate Users' position when they request a taxi (Guest/User) or they give availability (Taxi Driver).

### 1.7 Possible Future Implementations

This application allows many potential further implementations. Giving APIs to the programmers it permit to enlarge the service.

These are examples of possible future developments of the application:

- Create a social network-like environment where Users can communicate and for instance share a taxi.
- Create a profiling instrument that aims to evaluate both taxi drivers and users reliability. It will depends on customer feedback: for instance if a taxi driver miss an appointment it will receive a negative feedback.
- Create a function that allow large groups of people to easily benefit of Taxi service. In particular it will be useful to develop an algorithm that can manage multiple taxi call.
- Implement a Cost Evaluation instrument that allows to foresee the cost of a trip. It can do this collaborating with a Map App that support traffic evaluation.

## 1.8 Stakeholders

As Taxi Service is a public service the main external stakeholder will be the local administration. Its aim is, as said, to improve the approachability for customers and fairness for taxi drivers. So the administration expects an instrument able to build a meeting point for these two categories. This project may have other side stakeholders, for instance the society that will provide the localization system, concerned to be the only partner of the project relatively to this compound. Although the idea is to maintain this application ad-free, it will keep open the possibility to insert some advertisements in order to raise money, so another stakeholder may be an advertising company.

## 2. Specific Requirements

In this chapter the requirements of the application will be analysed in detail. They are divided in functional requirements and non-functional requirements. The formers define the functions of the application i.e. the specific behaviour of every single part of the system that occur in each situation. The latter denote criteria that can be used to judge the operation of a system, or a feature that the system should have in order to accomplish predetermined goals.

### 2.1 Functional Requirements

Starting from the domain properties (proposed in paragraph 1.4) it is possible to find the functional requirements of the application:

#### **[G1] Registration of people to the system**

[R1]The system should provide a form to fill with personal data in order to forward the registration request.

[R1.1]To avoid bot attacks that can crash the system a person have to solve a captcha™.

[R1.2]There will be two different forms to register to the system, one for each type of registration such as "Taxi Driver" or "User".

[R2]In case of Taxi driver registration an Administrator must check the validity of its license before it can start to use the service.

[R2.1]After the validation the system automatically send a mail with a summary of the personal data.

[R1.3]In case of User registration a confirmation mail will be delivered containing a link to activate the account.

## **[G2]Building a fair Taxi Queue**

[R1]The system should provide the functionality to build fairly a queue of taxis

[R1.1]A taxi driver could provide its readiness that can be translated in the desire to be queued.

[R1.2]When a taxi give its availability the system view its position on the map and choose the right queue to post it.

[R1.3]The taxi is added to the queue of its zone in the last position; the order of taxis is based on the entry time and date.

[R1.4]The fairness is granted by the impossibility to move up or down in the queue the taxis once they are included.

## **[G3]Calling a taxi**

[R1]The system should provide a functionality that allow customers to forward a taxi call.

[R1.1]Taxi call can be forwarded both by Guests and Users.

[R1.2]Guest has to provide each time its personal data in order to be recognized and it could call a taxi only from the web application (the mobile app needs a registration).

[R1.3]User has to log into the system to access the feature.

[R1.4]Both have to specify the number of participants in order to advise the system properly about how many taxi are needed.

[R1.5]It is required to have a location function to provide the precise position to the system, but it will be also possible to specify it manually.

[R1.6]If the call is forwarded from the web application it is possible to specify manually the position given the inaccuracy of the locator based on Wi-Fi

[R2]The system should also provide a functionality that allow taxi drivers to accept or deny customers' requests

[R2.1]When a request is forwarded the system advise the first taxi driver in the queue



[R2.2]The taxi drivers has to accept or deny the request

[R2.2.1]If it accepts the system will provide the identification code of the taxi to the customer

[R2.2.2]If it denies the system put it in the last position of the queue and it advise the next taxi driver

#### **[G4]Book a taxi in advance**

[R1]The system will provide a feature only for registered users that allows to make a reservation for a taxi

[R1.1]A user that wants to reserve a taxi have to make the booking at least two hours in advance

[R1.2]It has to provide the starting and the ending point and the desired time

[R1.3]The system will look for the right queue and ten minutes before the reservation time will forward the request to the first taxi driver of the queue

[R1.3.1]To resolve contention due to simultaneous request the system must have a function that is able to build a sort of request queue that will be forwarded one by one waiting for the taxi driver response

[R1.3.2]The request queue are composed by reservation and normal request

Now given these requirements, it is possible to define more specifically the functionality of each category of customer:

- **Guest:** it can access only basic functionalities:
  - Sign Up into the system
  - Make a taxi call
- **User:** it is a registered guest and it has the access on guest's features and farther functionalities:
  - Log into the system
    - ◆ To do so it has to be activated in advance and fill the login form with correct credentials
    - ◆ Once logged in if it use a mobile app it will never have to fill the login form again, if it does not logout manually
  - Make a taxi call

- ◆ It can call a taxi just giving (automatically or manually) its position since it is already recognized
- Book a taxi in advance
- Modify its personal data
- Retrieve the password
- **Taxi Driver:** it is the mainstay of the system, it provides the taxi service and here there are its functionalities:
  - Log into the system as a taxi driver
  - Provide its availability
  - Accept/Deny taxi request
  - Modify its personal data
  - Retrieve its password

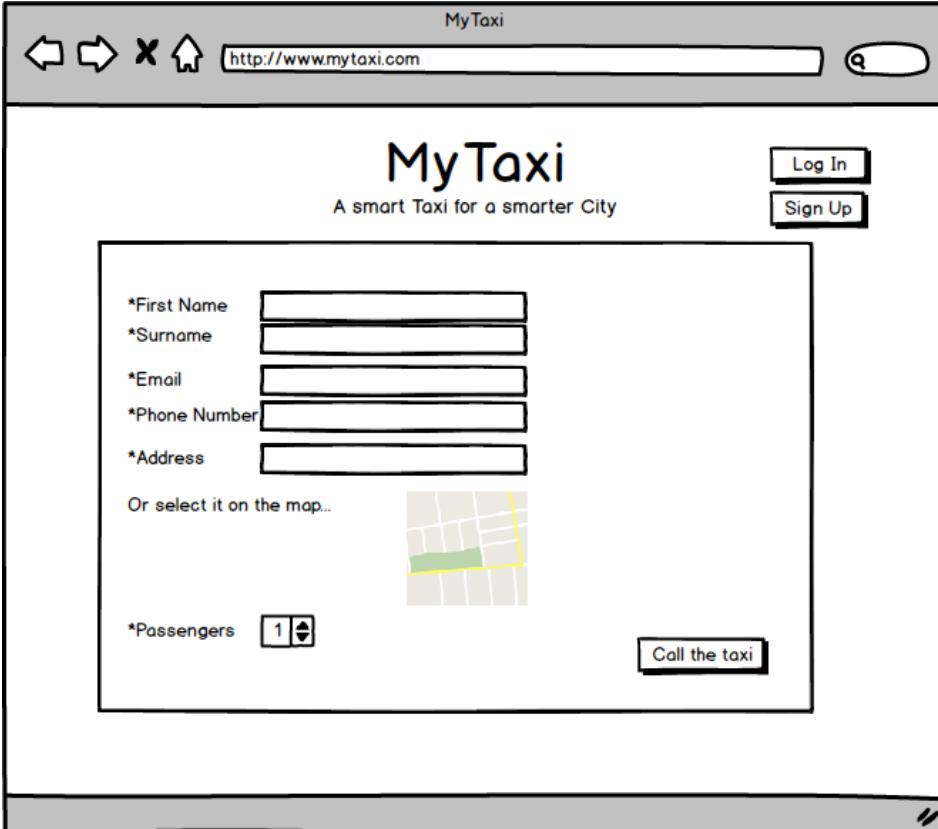
## 2.2 Non Functional Requirements

### 2.2.1 Look Requirements

The system will be implemented as a web service, which can be also used through a mobile app. Both these environments have to be attractive in order to impress the random guest or the first time user. The underlying theme of the entire graphic design will be interface minimalism and ease of use.

The home page will present the application Logo and motto, two buttons to get to the Login or the Sign Up page, and the form that have to be completed in order to call a taxi.

Home Page, Web Application version



The image shows a web browser window with the title "MyTaxi". The address bar displays "http://www.mytaxi.com". The page content includes the "MyTaxi" logo and the tagline "A smart Taxi for a smarter City". In the top right corner, there are "Log In" and "Sign Up" buttons. The main form area contains input fields for "\*First Name", "\*Surname", "\*Email", "\*Phone Number", and "\*Address". Below these fields is a map with a green highlighted area and the text "Or select it on the map...". At the bottom left of the form is a "\*Passengers" field with a spinner set to "1". A "Call the taxi" button is located at the bottom right of the form.

MyTaxi  
A smart Taxi for a smarter City

Log In  
Sign Up

\*First Name  
\*Surname  
\*Email  
\*Phone Number  
\*Address

Or select it on the map...

\*Passengers 1

Call the taxi

Home Page, Mobile Version



The image shows a mobile phone screen displaying the "MyTaxi" app. The status bar at the top shows "ABC" and "09:30 PM". The app header includes the "MyTaxi" logo and a hamburger menu icon. The form layout is similar to the web version but adapted for a smaller screen. It includes input fields for "\*First Name", "\*Surname", "\*Email", "\*Phone Number", and "\*Address". Below these is a map with a green highlighted area and the text "Or select it from the map...". At the bottom left is a "\*Passengers" field with a spinner set to "1". A "Call the Taxi" button is at the bottom. A note "Marked elements have to be filled" is positioned above the button.

MyTaxi

\*First Name  
\*Surname  
\*Email  
\*Phone Number  
\*Address

Or select it from the map...

\*Passengers 1

Marked elements have to be filled

Call the Taxi

In the sign up page the guest is asked to enter its personal data and to choose a password for its account. In order to sign up every user has to accept the Terms of Use and Privacy Policy of the application.

A taxi driver could send its registration clicking on the specific link in the homepage. Its account needs a verification check by an administrator before it can be activated.

### *Taxi Registration, Web Application Version*

The image shows a web browser window with the title "MyTaxi". The address bar displays "http://www.mytaxi.com". The page content includes the "MyTaxi" logo and the tagline "A smart Taxi for a smarter City". A "Home" button is located in the top right corner. The main section is titled "SIGN UP" and contains a form with the following fields: \*First Name, \*Surname, \*Username, \*Email, \*Phone Number, \*Password, \*Confirm password, \*Driving License, and \*Taxi License. Each field is represented by a text input box. At the bottom left of the form, a note states "Marked elements have to be filled...". A "Register" button is positioned at the bottom right of the form. The browser window has standard navigation buttons (back, forward, stop, home) and a search icon in the top left corner.

MyTaxi

http://www.mytaxi.com

Home

## MyTaxi

A smart Taxi for a smarter City

### SIGN UP

\*First Name

\*Surname

\*Username

\*Email

\*Phone Number

\*Password

\*Confirm password

\*Driving License

\*Taxi License

Marked elements have to be filled...

Register

*Taxi Registration, Mobile Version*



A mobile application interface for taxi driver registration. The screen shows a status bar at the top with signal strength, 'ABC', time '09:31 PM', and battery level. Below is a header with a home icon, the title 'MyTaxi', and a menu icon. The main heading is 'Registration for Taxi Drivers'. The form contains nine fields, each with an asterisk indicating it is required: First Name, Surname, Username, Email, Phone Number, Password, Confirm Password, Driving License, and Taxi License. A 'Register' button is located at the bottom of the form.

Registration for Taxi Drivers

\*First Name

\*Surname

\*Username

\*Email

\*Phone Number

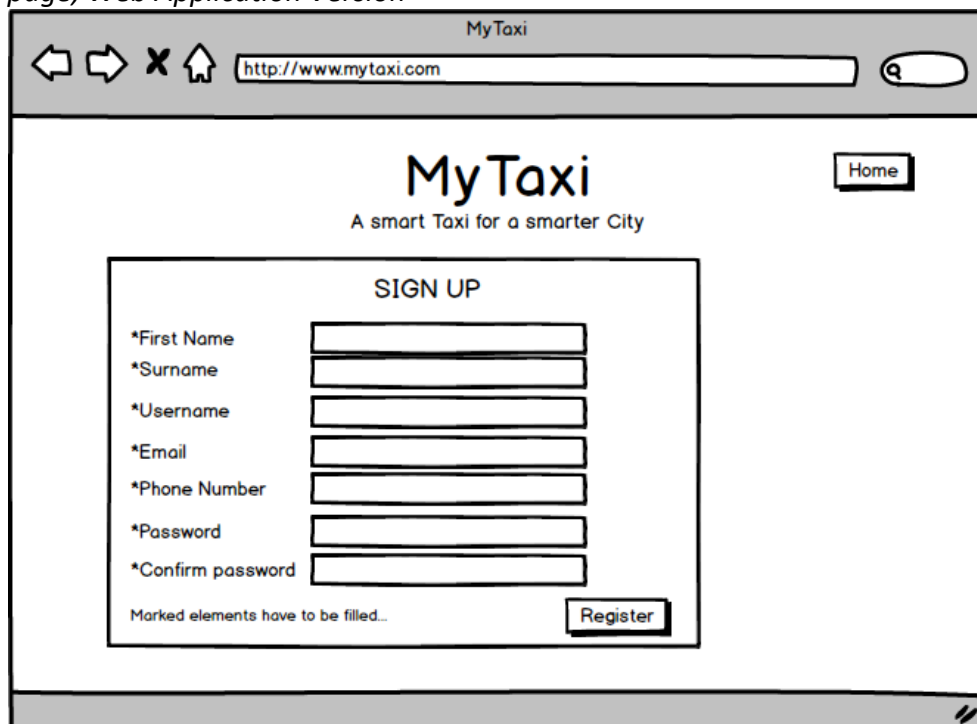
\*Password

\*Confirm Password

\*Driving License

\*Taxi License

*Sign Up page, Web Application Version*



A web application interface for taxi driver registration. The browser window shows the URL 'http://www.mytaxi.com'. The page has a header with the 'MyTaxi' logo, the tagline 'A smart Taxi for a smarter City', and a 'Home' button. The main content area is titled 'SIGN UP' and contains a form with eight fields, each with an asterisk indicating it is required: First Name, Surname, Username, Email, Phone Number, Password, Confirm password, and a 'Register' button. A note at the bottom of the form states 'Marked elements have to be filled...'. The browser window also shows navigation icons and a search bar.

MyTaxi

A smart Taxi for a smarter City

SIGN UP

\*First Name

\*Surname

\*Username

\*Email

\*Phone Number

\*Password

\*Confirm password

Marked elements have to be filled...

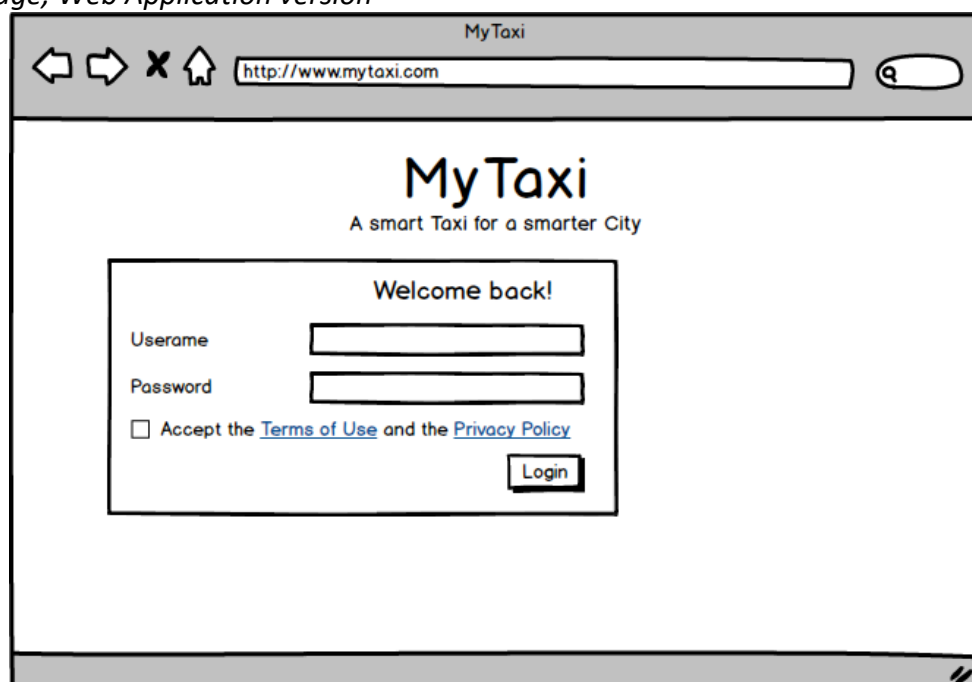
*Sign Up page, mobile version*



A mobile phone screen displaying the 'MyTaxi Sign Up' page. The status bar at the top shows 'ABC' and '09:31 PM'. The page has a home icon on the left and a menu icon on the right. The title 'MyTaxi' is centered at the top, followed by 'Sign Up'. Below this are eight input fields, each with an asterisk and a label: '\*First Name', '\*Surname', '\*Username', '\*Email', '\*Phone Number', '\*Address', '\*Password', and '\*Confirm Password'. A 'Register' button is located at the bottom of the form area.

Once a User is registered it can access to the reservation function i.e. the possibility to book a taxi in advance just giving information about the desired place and time. Clicking on the “Easy Call” button it could also call a taxi without any advance like an unregistered user does. Before getting access to its category functionalities it has to Login.

*Login Page, Web Application version*

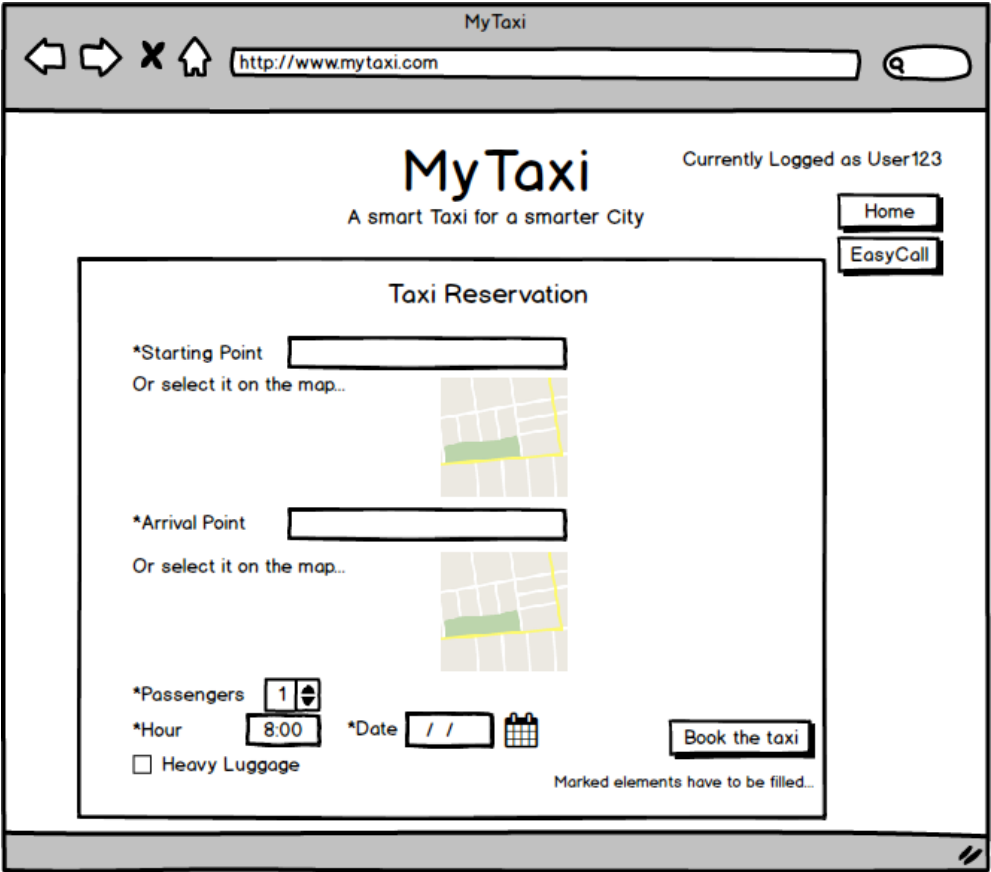


A web browser window displaying the 'MyTaxi' login page. The browser's address bar shows 'http://www.mytaxi.com'. The page title is 'MyTaxi' with the tagline 'A smart Taxi for a smarter City'. The main content area is titled 'Welcome back!' and contains a login form. The form has two input fields: 'Username' and 'Password'. Below these fields is a checkbox labeled 'Accept the [Terms of Use](#) and the [Privacy Policy](#)'. A 'Login' button is positioned at the bottom right of the form.

Login Page, Mobile Version



Taxi Reservation Page, Web Application version



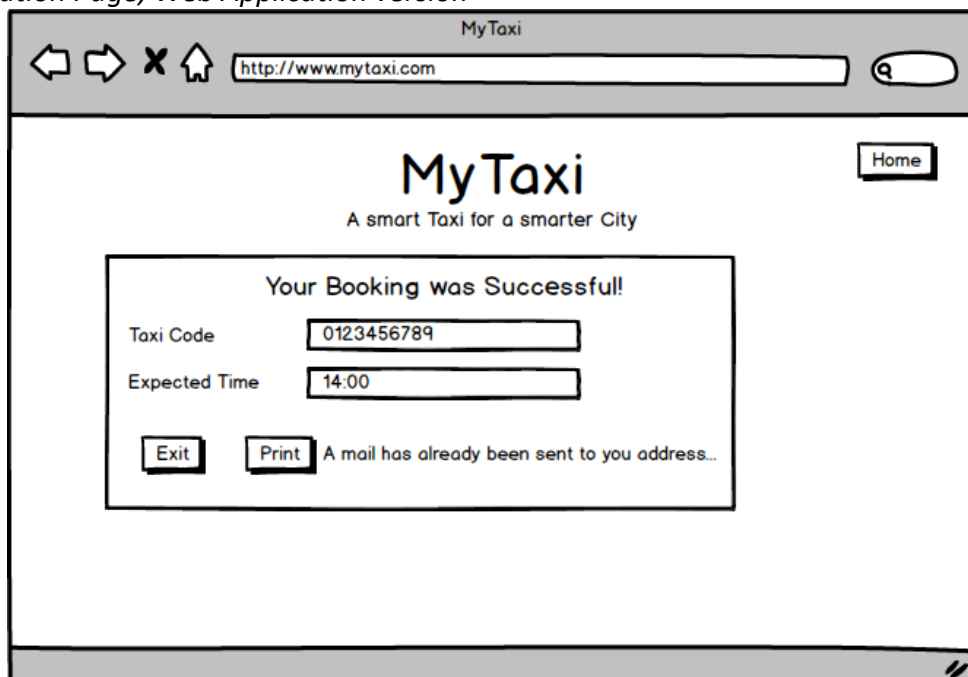
### Taxi Reservation, Mobile Version



The image shows a mobile application interface for 'MyTaxi' on a smartphone. The screen displays a 'Taxi Reservation' form. At the top, there's a status bar with 'ABC' and '09:31 PM'. The app header includes a home icon, the title 'MyTaxi', and a menu icon. The main form has two map sections for selecting 'Starting Point' and 'Arrival Point', each with a text input field and a map preview. Below the maps, there are fields for '\*Passengers' (set to 1), a date picker (06/11/2015), a checkbox for 'Heavy Luggage', and an '\*Hour' field (set to 15:00). A 'Book the taxi' button is at the bottom.

If a Call was Successful the guest or the user that have made it will be redirect to a Confirmation Page designed like this one.

### Confirmation Page, Web Application version



The image shows a web application interface for 'MyTaxi' displaying a confirmation page. The browser address bar shows 'http://www.mytaxi.com'. The page header includes the 'MyTaxi' logo, the tagline 'A smart Taxi for a smarter City', and a 'Home' button. The main content area features a box with the message 'Your Booking was Successful!'. Inside this box, there are two text input fields: 'Taxi Code' with the value '0123456789' and 'Expected Time' with the value '14:00'. Below these fields are 'Exit' and 'Print' buttons. A message at the bottom of the box states 'A mail has already been sent to you address...'.



Confirmation Page, Mobile Version



### **2.2.2 Usability**

The social background of the users of the system will be wide varied so the application must be user-friendly and easy to use as much as possible. Minimalism of the design will help users focus on main functionalities rather than get lost over a multiplicity of links.

### **2.2.3 Privacy Requirements**

- The system will protect users' personal data. In particular Login Credentials will be stored after being subjected to a hash function. In this way the system is protected against database hack and even an administrator can steal users' credential.
- The system will ensure that it will not share the journeys made by its users.

## **3. Software Design - UML**

### **3.1 Actors Identifying**

The actors of this informative system are mainly four:

- Guest
- User
- Taxi Driver
- Administrator

The first three have been already described in details in the glossary, they are the main consumer of the service. The administrator, as its name suggests, has administration powers such as check the drivers' license or eliminate bad users.

### **3.2 Possible Scenarios**

#### **Scenario: Make a Taxi Call**

It's about midnight and Cinderella has to come back home after the party organized by Christopher, but she notices that her sisters already took her car, and she can't reach the

house on foot. With relief she remembers of an advertising seen the day before, so she picks out her phone, and opens MyTaxi, an app for “Smart Taxi for a Smarter City” as the advertisement says.

She has to hurry, but she is not worried: she just needs to insert the number of passengers, her phone number and her email, and in a few minutes she receives a message with her taxi code and the estimated waiting time. Thanks to the new service, she could come back home.

### **Scenario: Taxi Reservation**

Mr. White is working in his office, but he is anxious about an important meeting with Alice in the afternoon. How can he be sure not to be late? He then turns on the computer, and in the news he read about the improve in the taxi service, so he looks for MyTaxi website, he sign up, and he will be able to book a taxi just providing starting and arriving addresses, and for what time to book. The system saves his request, and at the indicated time a taxi will wait for him on time.

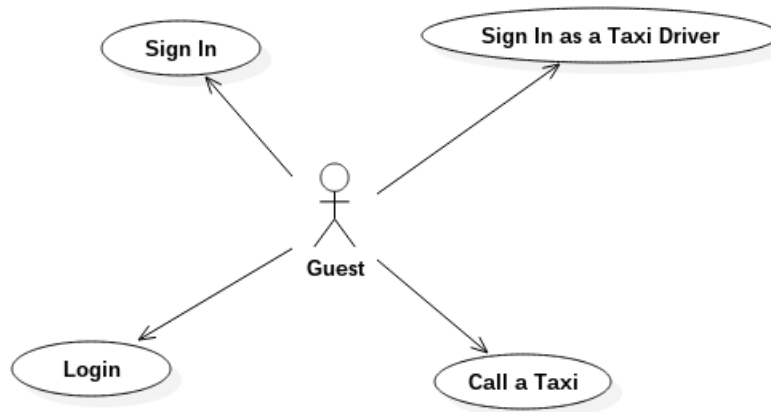
### **Scenario: Taxi Registration and first day of work**

Mark has just obtained his brand new taxi license, when a leaflet informs him of the new project about improving the taxi service. He then find the MyTaxi website, he starts the procedure to contribute as a taxi driver, inserts his data, and just waits for the confirmation from the site.

Once accepted, he just needs to give his availability using the app, and the system will calculate his zone and insert him in the queue, sending him the requests. When he receives a notification, he accepts and goes to take the client at the specified address.

At the end of the day, he just removes his availability to be free to go home.

### 3.3 Actor: Guest



#### Use Case: Sign Up

##### Description:

A guest, who doesn't have a profile yet, can submit his data to the system in order to become a registered User.

##### Actors:

Guest

##### Input Condition:

Null

##### Output Condition:

Guest successfully ends the registration process, and becomes a User. From now on he/she can log in to the application using his/her credential and have access to peculiar functionalities.

##### Events flow:

- Guest on the home page clicks on the “register” button to start the registration process;
- Guest fills in at least all the mandatory fields;
- Guest clicks on “confirm” button;

- The application will save the data in the database.

**Exceptions:**

- The guest has already registered;
- The chosen user name is already used;
- The email inserted is already associated to another user;
- One or more mandatory fields are empty or not valid;

All the exceptions cause the application to notify the error to the user with an alert window. The application then come back to the registration form, and the Event Flow will restart from the filling step.

**Use Case: Sign Up as Taxi Driver****Description:**

A guest, who doesn't have a profile yet, can submit his data to the system in order to become a Taxi Driver.

**Actors:**

Guest

**Input Condition:**

The Guest must have valid driving and taxi license.

**Output Condition:**

Guest successfully ends the registration process, and becomes a Taxi Driver. From now on he/she can log in to the application using his/her credential and have access to peculiar functionalities.

**Events Flow:**

- Guest on the home page clicks on the “sign up as a taxi driver” button to start the registration process;

- Guest fills in at least all the mandatory fields, and upload data related to driving license and taxi license;
- Guest clicks on “confirm” button;
- The application will save the data in the database.
- After a check by administrators, the Guest will be authorized to login as a Taxi Driver;

**Exceptions:**

- The guest has already registered;
- The chosen user name is already used;
- The email inserted is already associated to another user;
- One or more mandatory fields are empty or not valid;
- Driving license or taxi license are not valid;
- Driving license or taxi license are already associated to another user.

All the exceptions cause the application to notify the error to the user with an alert window. The application then come back to the registration form, to restart the Events Flow from the filling step.

## **Use Case: Login**

**Description:**

A guest already registered can provide his credential to log in to the application, becoming a User and thus gaining User or Taxi Driver privileges.

**Actors:**

Guest

**Input Conditions:**

The Guest must be already registered to the system.

**Output Conditions:**

The Guest is promoted to User or Taxi Driver.

**Events Flow:**

- Guest fills in the log in form already present in the home page.
- The application verifies the inserted credentials, and if they are correct, promotes the guest and shows additional features.

**Exceptions:**

If the credentials are not correct, so user name and password don't match, an alert window will be shown and the access denied, remaining on the home page, and giving the guest the possibility to try again.

**Use Case: Call a taxi**

**Description:**

The guest has the opportunity to call for a taxi ride without being registered to the site.

**Actors:**

Guest, Taxi Driver

**Input Condition:**

Null

**Output Conditions:**

The first free taxi will answer the call and the guest will be served as requested.

**Events Flow:**

- Guest fills in the form in the home page, providing a name, a surname, the number of passengers and a phone number;
- Guest submits the request.
- His/her data, and his/her position, detected by using GPS, will be processed by the

system, that starts sending the request to the first taxi in the zone queue;

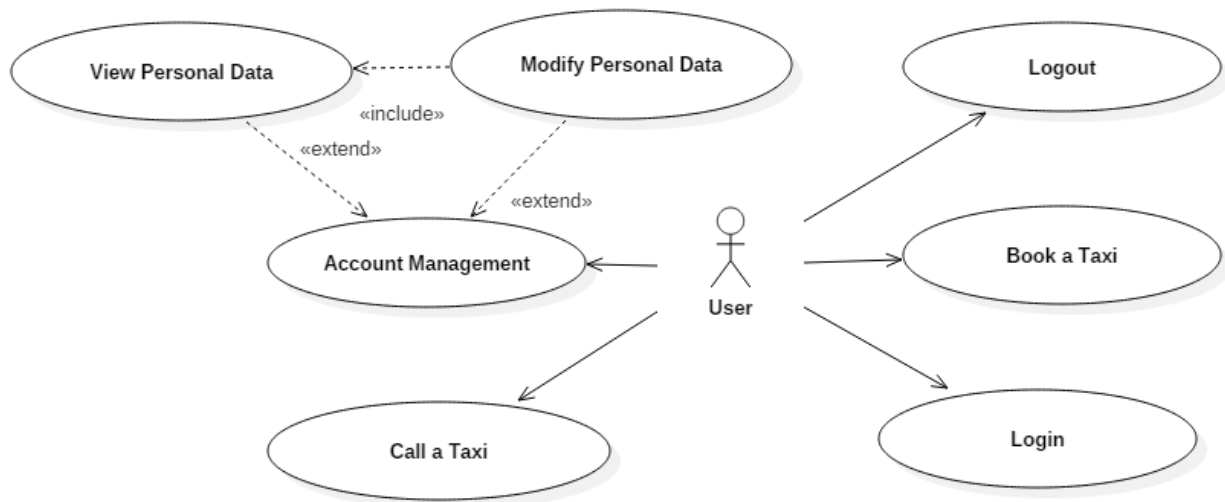
- The taxi accept the request, and goes to take the guest in the place reported by the system.

**Exceptions:**

- If the guest doesn't fill in all the fields, or the values inserted are not valid, the request cannot be sent, and the user will be warned;
- If the taxi declines the request, the system will move that driver to the bottom of the queue, and proceeds asking to the next taxi in the queue, following the order.
- If problems arise with positioning or times, taxi drivers can use the customer's phone number to call them and make verbal arrangements.



### 3.4 Actor: User



#### Use Case: Login

##### Description:

A User can provide his credential to log in to the application, being recognized by the system as a User and thus gaining in fact User privileges.

##### Actors:

User

##### Input Conditions:

The User must be already registered to the system.

##### Output Conditions:

The User is recognized and authorized to operate as a User.

##### Events Flow:

- User fills in the log in form already present in the home page.
- The application verifies the inserted credentials, and if they are correct, recognizes the User and shows additional features.

##### Exceptions:

If the credentials are not correct, so user name and password don't match, an alert window will be shown and the access denied, remaining on the home page, and giving the User the possibility to try again.

## **Use Case: Logout**

### **Description:**

This functionality allows the User to close the current session on the site and brings him/her back to Guest level.

### **Actors:**

User

### **Input Conditions:**

The User must be logged in.

### **Output Conditions:**

The User closes the current session, loses his/her privileges and becomes a Guest.

### **Events Flow:**

- User clicks on the “Logout” button in the page.
- The system close the session, and brings the User back to home page, as a Guest.

### **Exceptions:**

There are no possible exceptions.

## **Use Case: Call a taxi**

### **Description:**

This functionality is similar to the one presented for the Guest use case, but in this case most of the data are already present in the database, so the procedure is faster and easier.

### **Actors:**

User, Taxi Driver

### **Input Conditions:**

The User must be logged in.

### **Output conditions:**

The first free taxi will answer the call and the User will be served as requested.

### **Events Flow:**

- User inserts the number of passengers for the requested ride;
- User submits the request;
- The system collects the number inserted, together with User data and his/her position tracked with GPS, and sends a request to the first free taxi in the zone queue;
- The taxi accept the request, and goes to take the User in the place reported by the system.

### **Exceptions:**

- If the User doesn't provide the number of passengers, or the value inserted is not valid, the request cannot be sent, and the user will be warned;
- If the taxi declines the request, the system will move that driver to the bottom of the queue, and proceeds asking to the next taxi in the queue, following the order.

- If problems arise with positioning or times, taxi drivers can use the customer's phone number to call them and make verbal arrangements.

## **Use Case: Book a taxi**

### **Description:**

The User can book a taxi in advance, to be sure to have a ride on time and without problems.

### **Actors:**

User, Taxi Driver

### **Input Conditions:**

The User must be logged in.

### **Output Conditions:**

The User will have a booked ride for the time and the ride requested.

### **Events Flow:**

- The User fills in the form for the booking procedure, deciding the time of the ride, that has to be at least two hours after the time of the reservation procedure, the starting address, the destination, and the number of passengers;
- The system checks and saves the values inserted sending a confirmation mail to the user;
- Ten minutes before the arranged time, the system generates an automatic request to ask for the ride requested by the User;
- The automatic call works as a normal call for generic guest or users (see use case Call a Taxi for specific information).

**Exceptions:**

If the User tries to insert a time that is not at least two hours later that moment, the system will reject the request, and will give the User the opportunity to try again with other times.

**Use Case: Account Management****Description:**

A User can see his data saved in the website and modify them, if necessary.

**Actors:**

User

**Input Conditions:**

The User must be logged in.

**Output Conditions:**

The User can see the data inserted in the database, if he/she modified them, change will be saved.

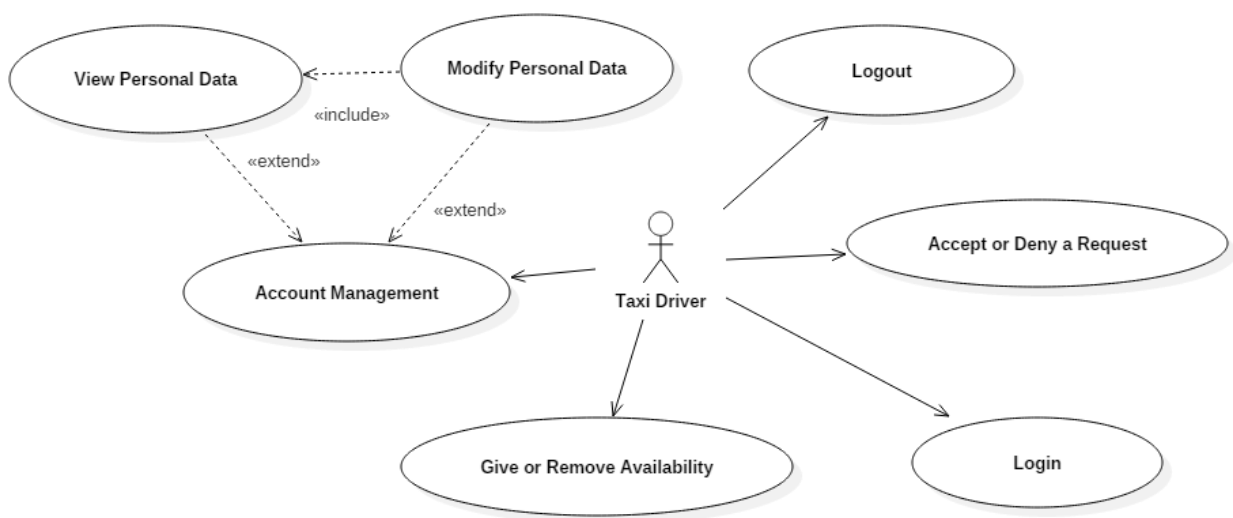
**Events Flow:**

- User clicks on the “account management” button in the home page.
- In the new page the User can see a list of text field containing his data.
- By writing in them, the User can modify them.
- At the end of the process, he/she can click on the “save” button to confirm the changes made.

**Exceptions:**

- If new data inserted are incorrect or there are empty fields, and the User tries to save the changes, an error message will be shown, and changes won’t be saved. The User can retry the process.
- If the User exits the page without saving, changes won’t be saved.

### 3.5 Actor: Taxi Driver



#### Use Case: Login

##### Description:

A Taxi Driver can provide his credential to log in to the application, being recognized by the system as a User and thus gaining in fact Taxi Driver privileges.

##### Actors:

Taxi Driver

##### Input Conditions:

The Taxi Driver must be already registered to the system.

**Output Conditions:**

The Taxi Driver is recognized and authorized to operate as a Taxi Driver.

**Events Flow:**

- Taxi Driver fills in the log in form already present in the home page.
- The application verifies the inserted credentials, and if they are correct, recognizes the Taxi Driver and shows additional features.

**Exceptions:**

If the credentials are not correct, so user name and password don't match, an alert window will be shown and the access denied, remaining on the home page, and giving the Taxi Driver the possibility to try again.

**Use Case: Logout****Description:**

This functionality allows the Taxi Driver to close the current session on the site and brings him/her back to Guest level.

**Actors:**

Taxi Driver

**Input Conditions:**

The Taxi Driver must be logged in.

**Output Conditions:**

The Taxi Driver closes the current session, loses his/her privileges and becomes a Guest.

**Event Flow:**

- User clicks on the “Logout” button in the page.
- The system close the session, and brings the Taxi Driver back to home page, as a Guest.

**Exceptions:**

There are no possible exceptions.

**Use Case: Give/Remove Availability****Description:**

The Taxi Driver let the system knows if he/she is available or not to accept requests.

**Actors:**

Taxi Driver

**Input Conditions:**

The Taxi Driver has to be logged in.

**Output Conditions:**

The Taxi Driver is inserted in a queue, or he/she is removed from it.

**Event Flow:**

- When ready to accept requests, a Taxi Driver gives the availability, and he/she will be inserted by the system in the zone queue based on his position tracked by GPS.
- When he/she removes the availability, he/she will be removed from the queue where he/she was lastly placed.

**Exceptions:**

There are no possible exceptions.



## **Use Case: Accept/Deny a Request**

### **Description:**

The Taxi Driver has the possibility to accept or decline a request from a customer.

### **Actors:**

Taxi Driver

### **Input Conditions:**

The Taxi Driver has to be logged in and be available.

### **Output Conditions:**

If the Taxi Driver accepts the request, he/she will go to the place of the meeting with the customer; otherwise he/she will wait for another request.

### **Event Flow:**

- When the Taxi Driver is the first of his zone queue, he/she will receive the next available request;
- Taxi Driver can decide to accept or reject the request;
- If he/she accepts, he/she can go to the place pointed by the system, to take on the customer;
- If he/she rejects, he/she will be put at the bottom of the queue.

### **Exceptions:**

If the Taxi Driver doesn't answer the request, after a minute time the request will be considered as rejected by the Taxi Driver, that will be moved to the bottom of the queue.

## **Use Case: Account Management**

### **Description:**

A Taxi Driver can see his data saved in the website and modify them, if necessary.

### **Actors:**

Taxi Driver

### **Input Conditions:**

The Taxi Driver must be logged in.

### **Output Conditions:**

The Taxi Driver can see the data inserted in the database, if he/she modified them, change will be saved.

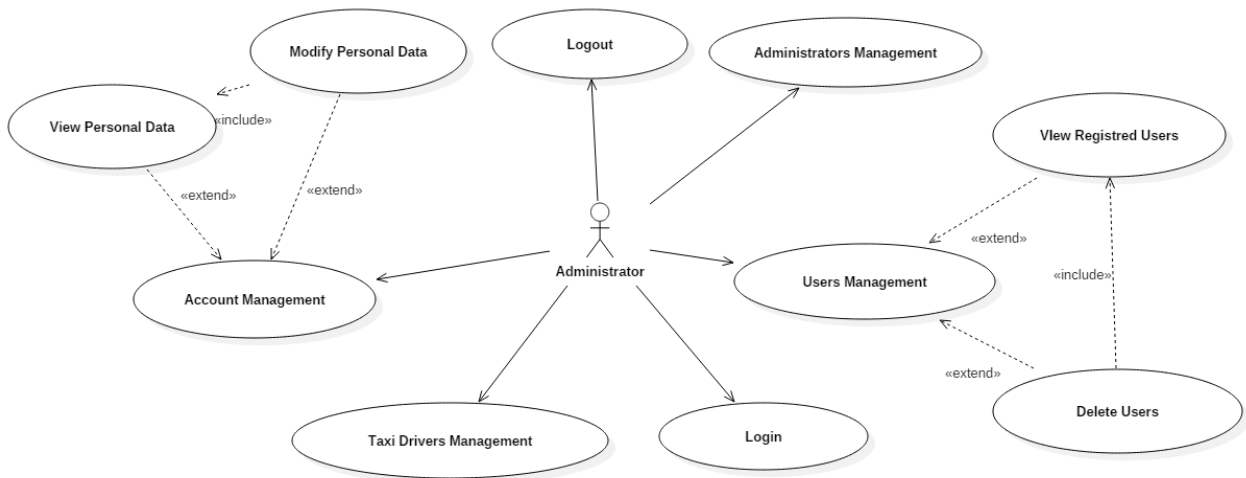
### **Events Flow:**

- Taxi Driver clicks on the “account management” button in the home page.
- In the new page the Taxi Driver can see a list of text field containing his data.
- By writing in them, the Taxi Driver can modify them; he/she can’t modify licenses codes.
- At the end of the process, he/she can click on the “save” button to confirm the changes made.

### **Exceptions:**

- If new data inserted are incorrect or there are empty fields, and the Taxi Driver tries to save the changes, an error message will be shown, and changes won’t be saved. The Taxi Driver can retry the process.
- If the Taxi Driver exits the page without saving, changes won’t be saved.

### 3.6 Actor: Administrator



#### Use Case: Login

##### Description:

Administrator can provide his credential to log in to the application, being recognized by the system as an Administrator and thus gaining in fact Administrator privileges.

##### Actors:

Administrator

##### Input Conditions:

The Administrator must be already registered to the system.

##### Output Conditions:

The Administrator is recognized and authorized to operate as an Administrator

##### Events Flow:

- Administrator fills in the log in form already present in the home page.
- The application verifies the inserted credentials, and if they are correct, recognizes the Administrator and shows additional features.

**Exceptions:**

If the credentials are not correct, so user name and password don't match, an alert window will be shown and the access denied, remaining on the home page, and giving the Administrator the possibility to try again.

**Use Case: Logout****Description:**

This functionality allows the Administrator to close the current session on the site and brings him/her back to Guest level.

**Actors:**

Administrator

**Input Conditions:**

The Administrator must be logged in.

**Output Conditions:**

The Administrator closes the current session, loses his/her privileges and becomes a Guest.

**Event Flow:**

- Administrator clicks on the "Logout" button in the page.
- The system close the session, and brings the Administrator back to home page, as a Guest.

**Exceptions:**

There are no possible exceptions.

## **Use Case: Remove Users**

### **Description:**

The Administrator has the power to see the list of registered users, and remove any of them if necessary.

### **Actors:**

Administrator, User (passive presence).

### **Input Conditions:**

Administrator has to be logged in, User has to be registered.

### **Output Conditions:**

The selected User will be removed from the system.

His data, however, will remain saved in the database.

### **Events Flow:**

- Administrator clicks on the “users management” button to access the dedicated page;
- In the management page he/she can see the list of registered users;
- Administrator select one of the users by clicking on his/her user name;
- By clicking the “delete” button, the Administrator removes the user from the system.

### **Exceptions:**

There are no possible exceptions.

## **Use Case: Taxi Drivers Management**

### **Description:**

The Administrator has the job to check every request from new taxi driver, in order to verify if data related to licenses are correct. If so, the Administrator gives the Taxi Driver the permission to work.

### **Actors:**

Administrator, Taxi Driver (passive presence).

### **Input Conditions:**

Administrator has to be logged in, Taxi Driver has to be registered.

### **Output Condition:**

The Taxi Driver considered will be authorized to work, or rejected.

### **Events Flow:**

- Administrator clicks on the “Taxi drivers management” button to access the dedicated page;
- In the management page he/she can see the list of pending requests;
- Administrator select one of the requests by clicking on the user name;
- A list of related data will be shown in the page, letting the Administrator check the information inserted by the user;
- By clicking on the “Accept” button the Taxi Driver will be accepted, otherwise by clicking on the “Deny” button the Taxi Driver will be rejected.

### **Exceptions:**

There are no possible exceptions.

## **Use Case: Account Management**

### **Description:**

An Administrator can see his data saved in the website and modify them, if necessary.

**Actors:**

Administrator

**Input Conditions:**

The Administrator must be logged in.

**Output Conditions:**

The Administrator can see the data inserted in the database, if he/she modified them, change will be saved.

**Events Flow:**

- Administrator clicks on the “account management” button in the home page.
- In the new page the Administrator can see a list of text field containing his data.
- By writing in them, the Administrator can modify them.
- At the end of the process, he/she can click on the “save” button to confirm the changes made.

**Exceptions:**

- If new data inserted are incorrect or there are empty fields, and the Administrator tries to save the changes, an error message will be shown, and changes won't be saved. The Administrator can retry the process.
- If the Administrator exits the page without saving, changes won't be saved.

## **Use Case: Administrators Management**

### **Description:**

An administrator has the power to create other Administrator profiles, submitting their data to the system.

### **Actors:**

Administrator

### **Input Condition:**

Administrator must be logged in

### **Output Condition:**

A new Administrator is created, and his data will be saved in the database.

### **Events flow:**

- Administrator on the home page clicks on the “Administrators Management” button to start the registration process;
- Administrator fills in at least all the mandatory fields;
- Administrator clicks on “confirm” button;
- The application will save the data related to the new administrator in the database. The Administrator has to give the new credential to the designed person. He/she can log in as an Administrator with these credential.

### **Exceptions:**

- The new Administrator is already in the database;
- The chosen username is already used;
- The email inserted is already associated to another profile;
- One or more mandatory fields are empty or not valid;

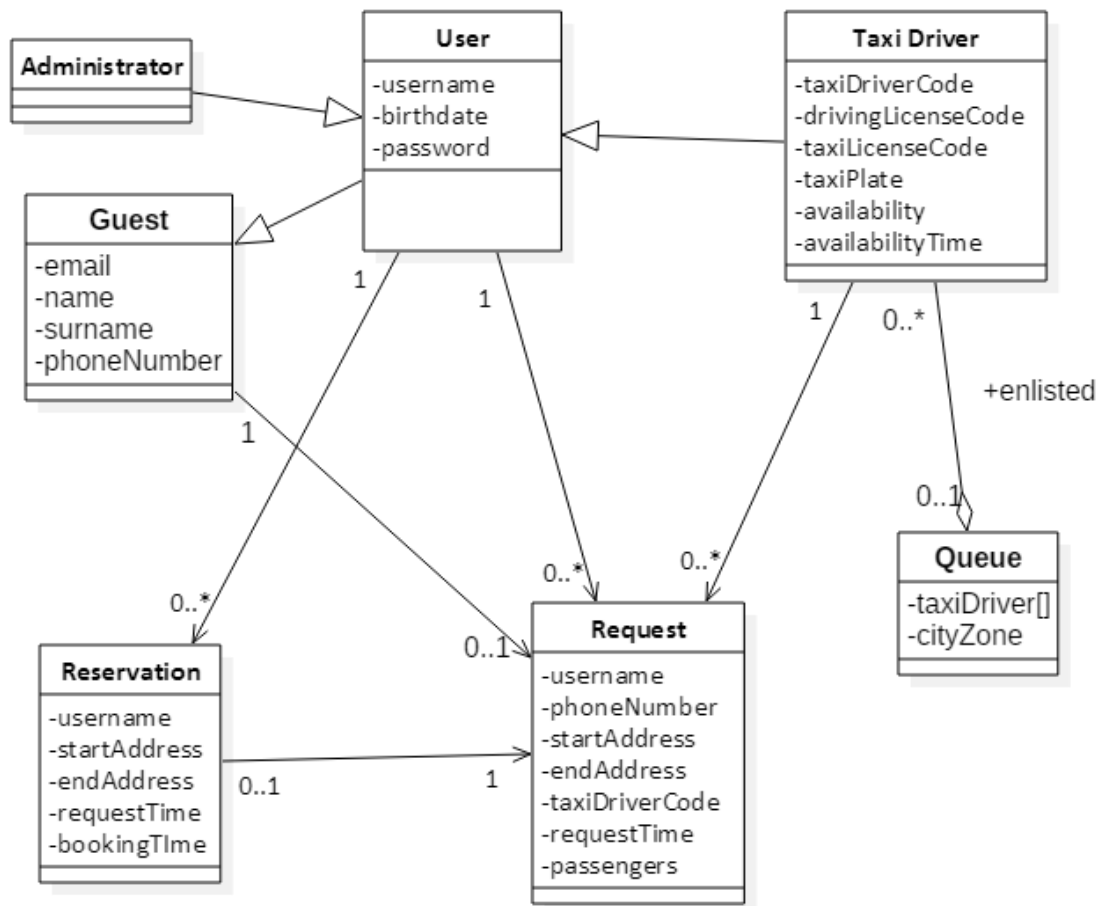
All the exceptions cause the application to notify the error to the Administrator with an alert window. The application then come back to the registration form, and the Event Flow will restart from the filling step.



## 3.7 Domain Model

### 3.7.1 Domain Class Diagram

In this paragraph it will be presented the Domain Class Diagram, a conceptual model of the system that contains the abstract entities described in this document. It also emphasize their mutual relations that could be for instance “generalization” or “composition”.



User, Taxi Driver and Administrator are generalizations of the Guest entity, so they inherit all its attributes which is personal data that allow to identify a person.

Both User and Guest entities are related with the Request entity, that is given by their ability to call a taxi. The cardinality of this relation “1” for the request, a request must have an owner, and “0..1” for Guest and User, a Guest or a User may not have a pending request. A request has the attributes that ensure a univocally association between User and Taxi. So it has the user’s identification data and the taxi code. It has the attribute that

contain the number of passengers.

User is related also with Reservation entity. This relation has the same cardinality than the relation between user and request. A Reservation generates a request ten minutes before the time specified by its attribute "requestTime". This involves creating the relationship between Reservation and Request. A Reservation must have a request but a request may not have been generated by a reservation.

As it has been said before each request must have a taxi driver. It justifies the relationship between the two entities.

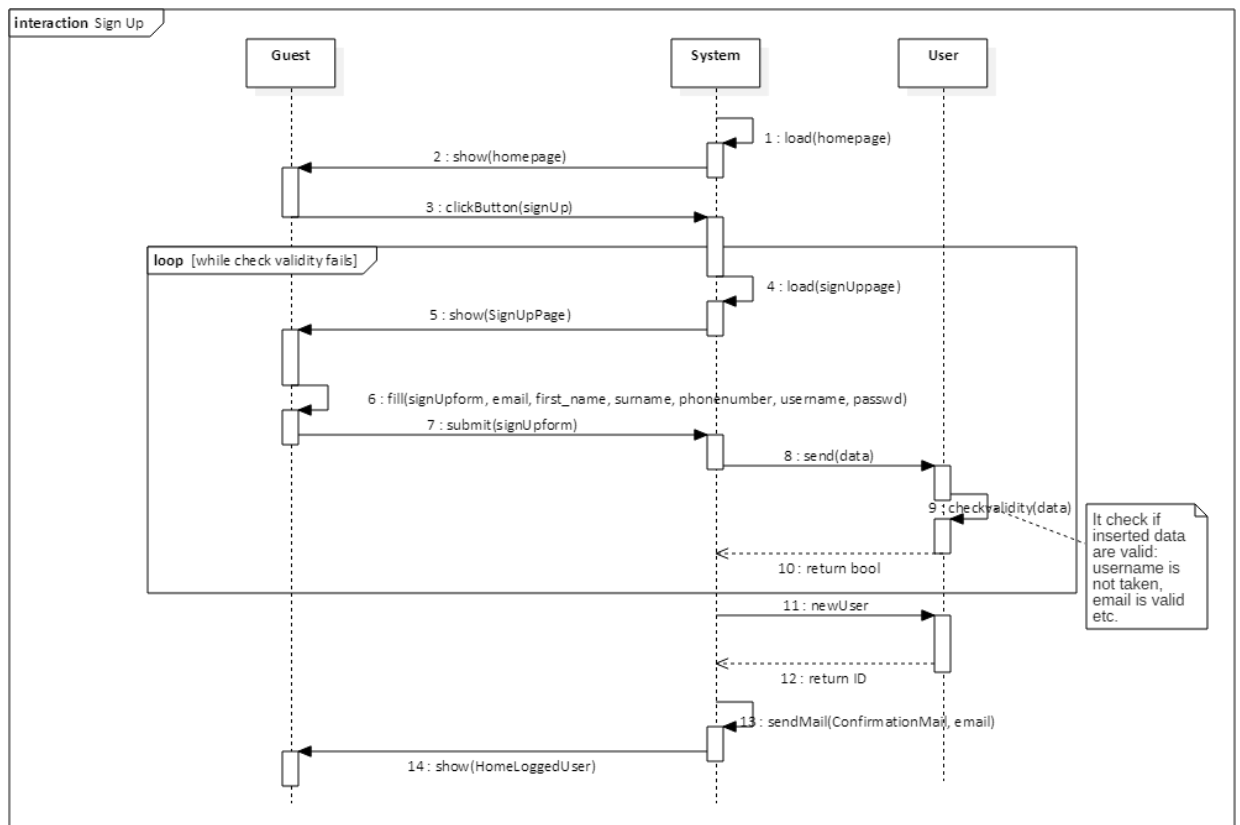
The Queue entity has an attribute that define which area is associated with, and a list of taxi. The composition relationship describes the fact that the queue is formed by objects of type taxi.

### **3.7.2 Main dynamics of the system**

In this paragraph it will be analysed in detail the most important behaviours of the system. UML language provides Sequence Diagram as a tool to model the dynamic behaviour analysis. It will be presented the Taxi Call situation, the Reservation of a taxi and basic dynamics like Sign Up and Login.

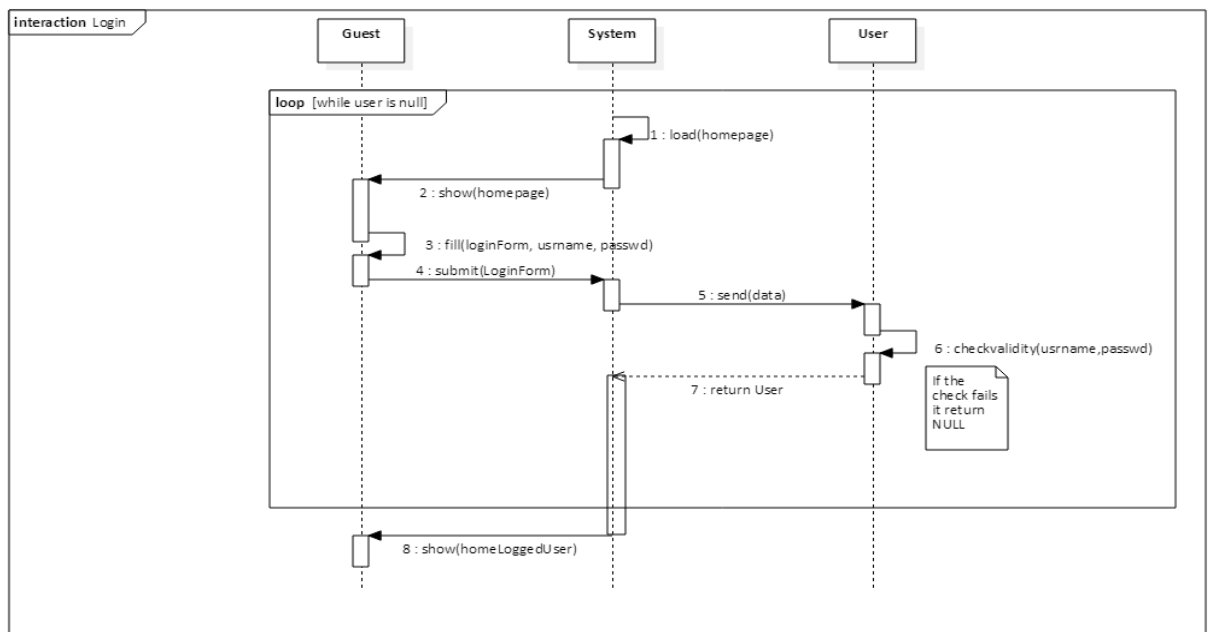
The first Sequence Diagram represent the Sign Up functionality: as we can see there is a loop that holds until guest fill properly the Sign Up form.

### Sign Up Sequence Diagram:



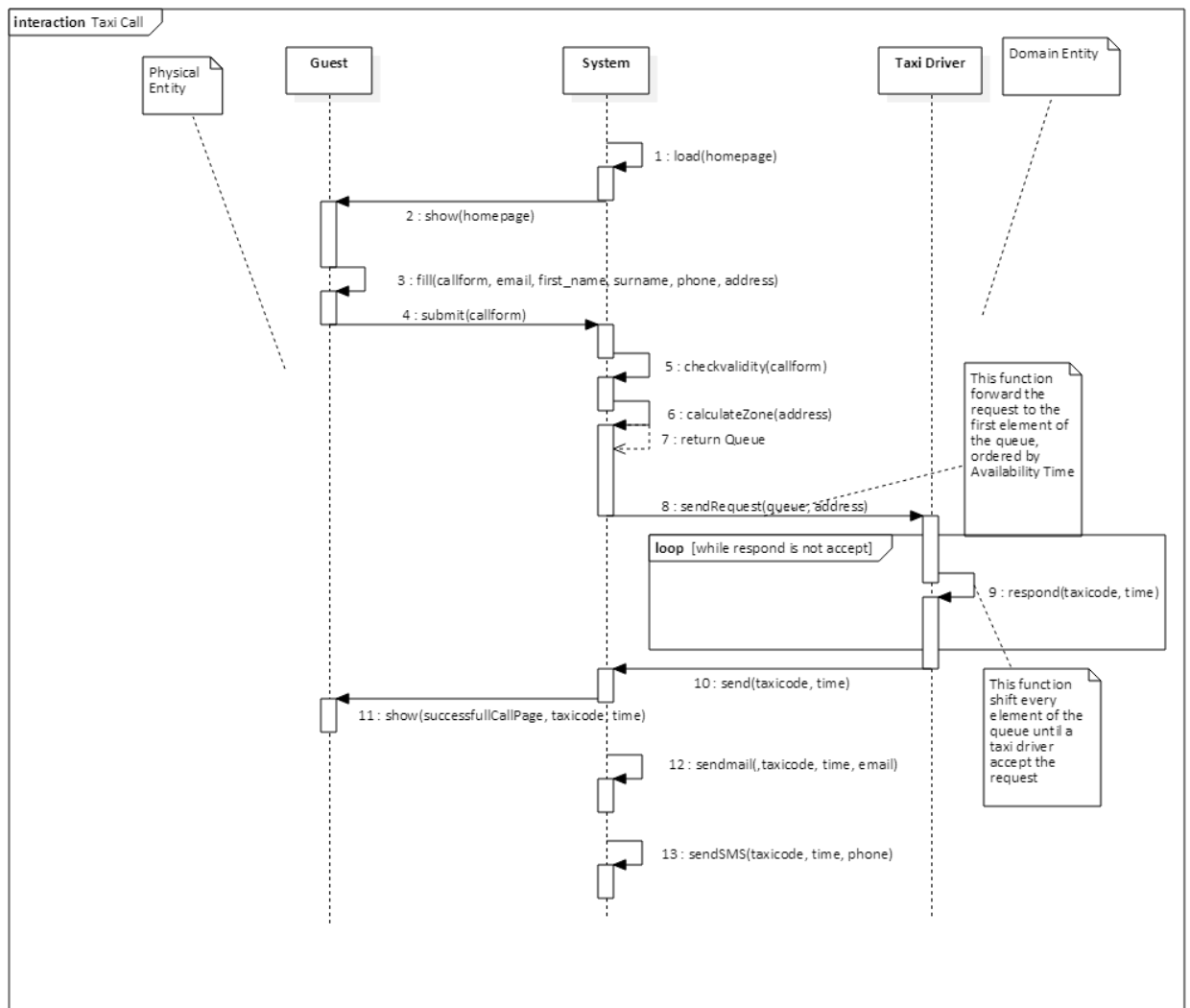
After the Sign Up a User get access to Login functionality. As it has been said in the User Interface paragraph, a User has to click on Login button in the homepage in order to reach the Login Page.

### Login Sequence Diagram:



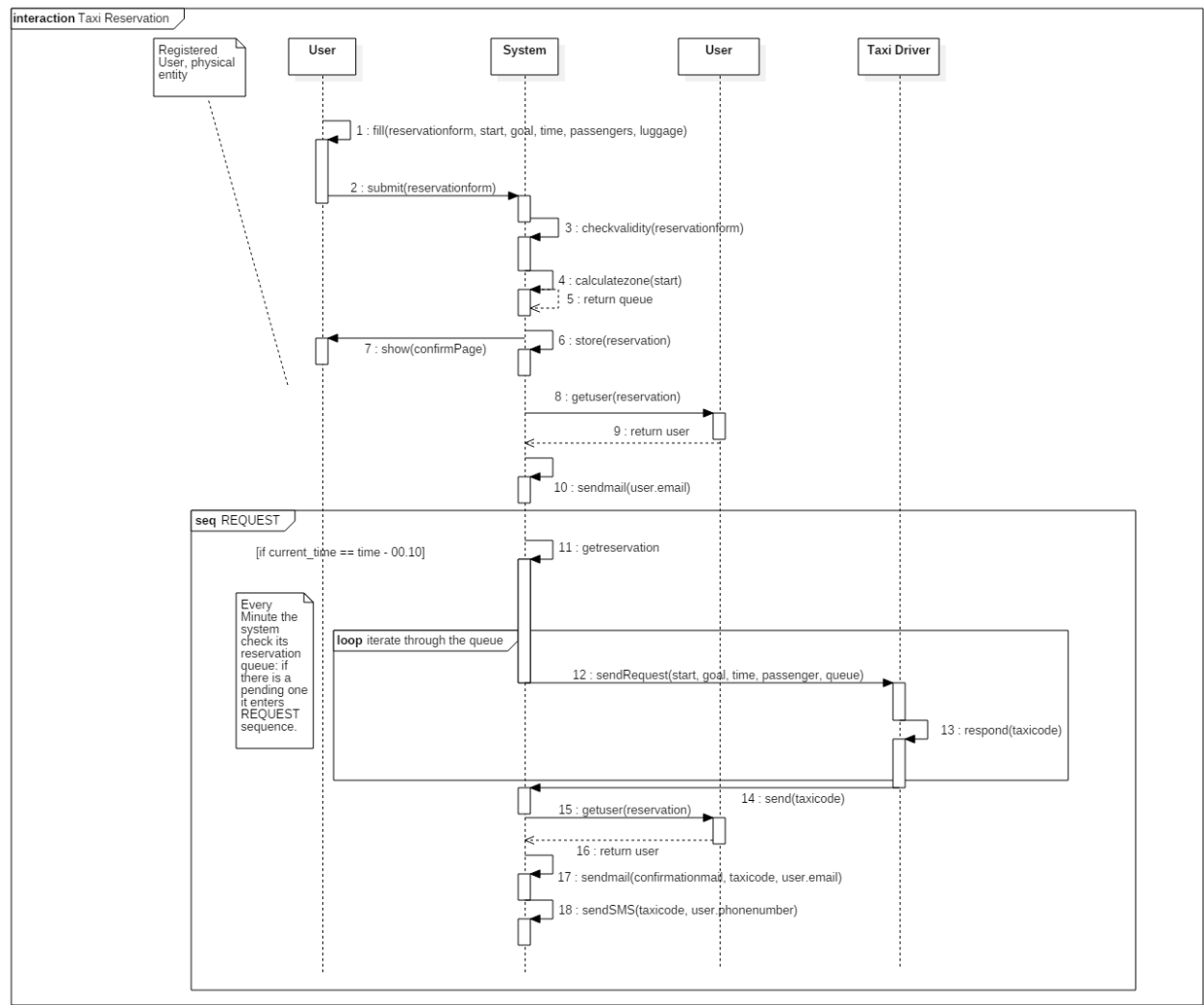
Unregistered people, called Guests in this system, can call a taxi, right from the homepage. This sequence diagram explain how the system work in that situation:

*Taxi Call Sequence Diagram:*



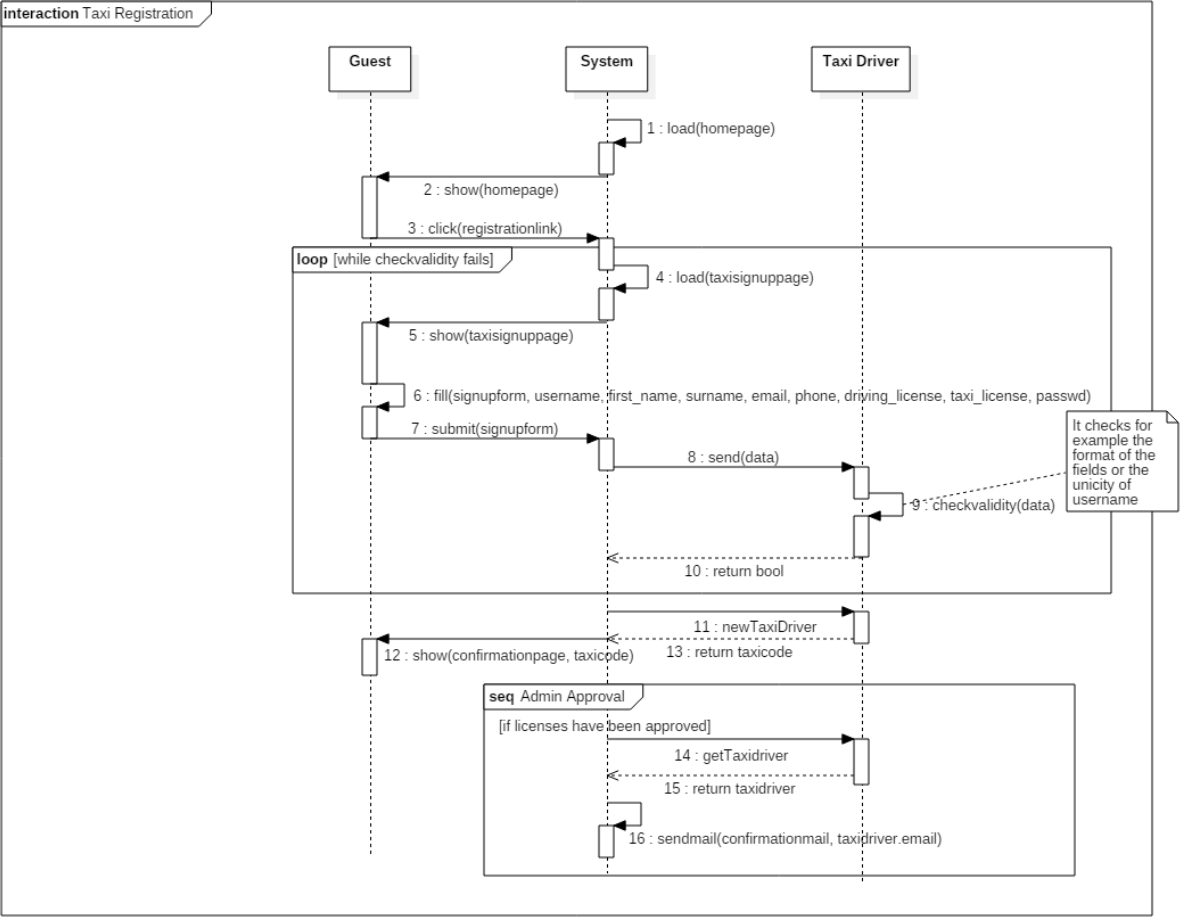
Users can also Reserve a taxi in advance, as previously mentioned. The system ensures that the reservation becomes a Request 10 minutes before the scheduled time, checking every minute the reservation list. When a taxi driver accepts the request the system sends an email and a SMS with the correspondent taxi code.

Taxi Reservation Sequence Diagram:



In order to become an effective member of the project a taxi driver has to be approved from an Administrator.

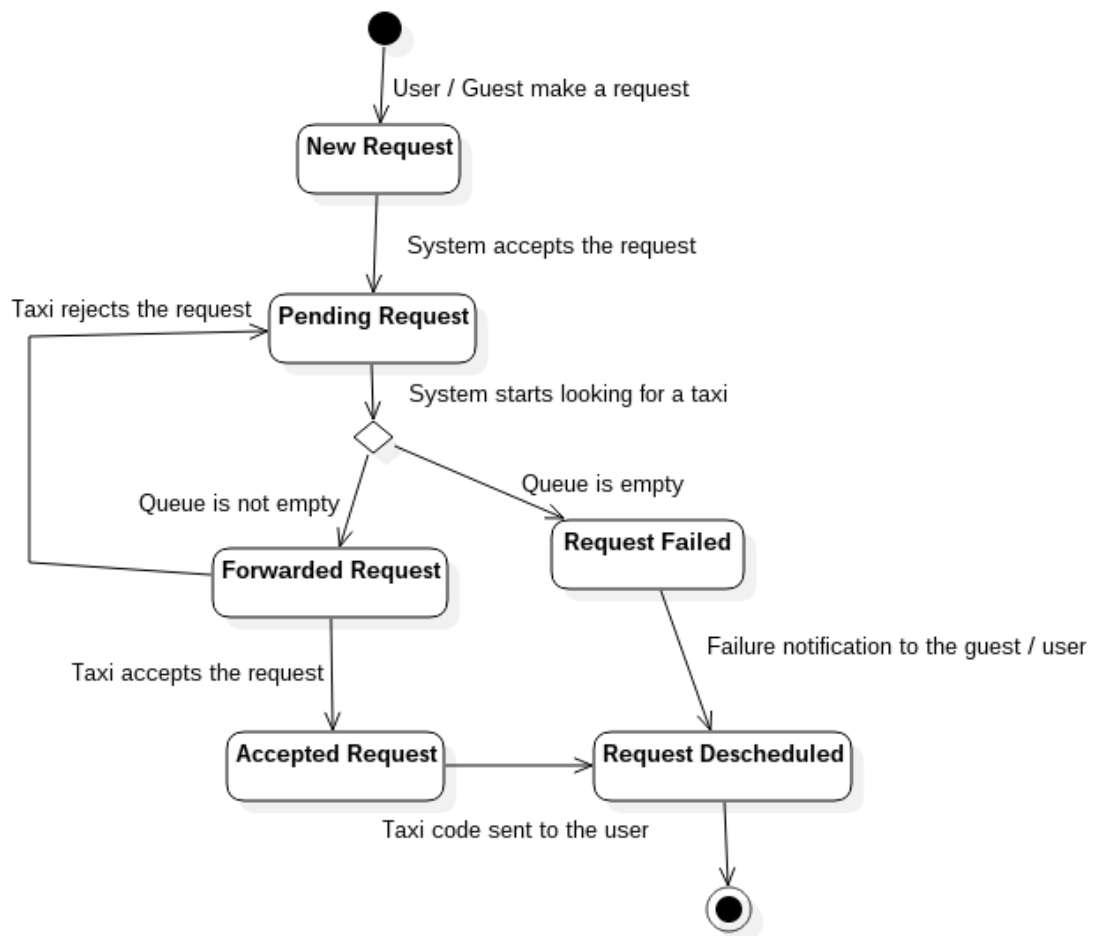
Taxi Driver Registration:



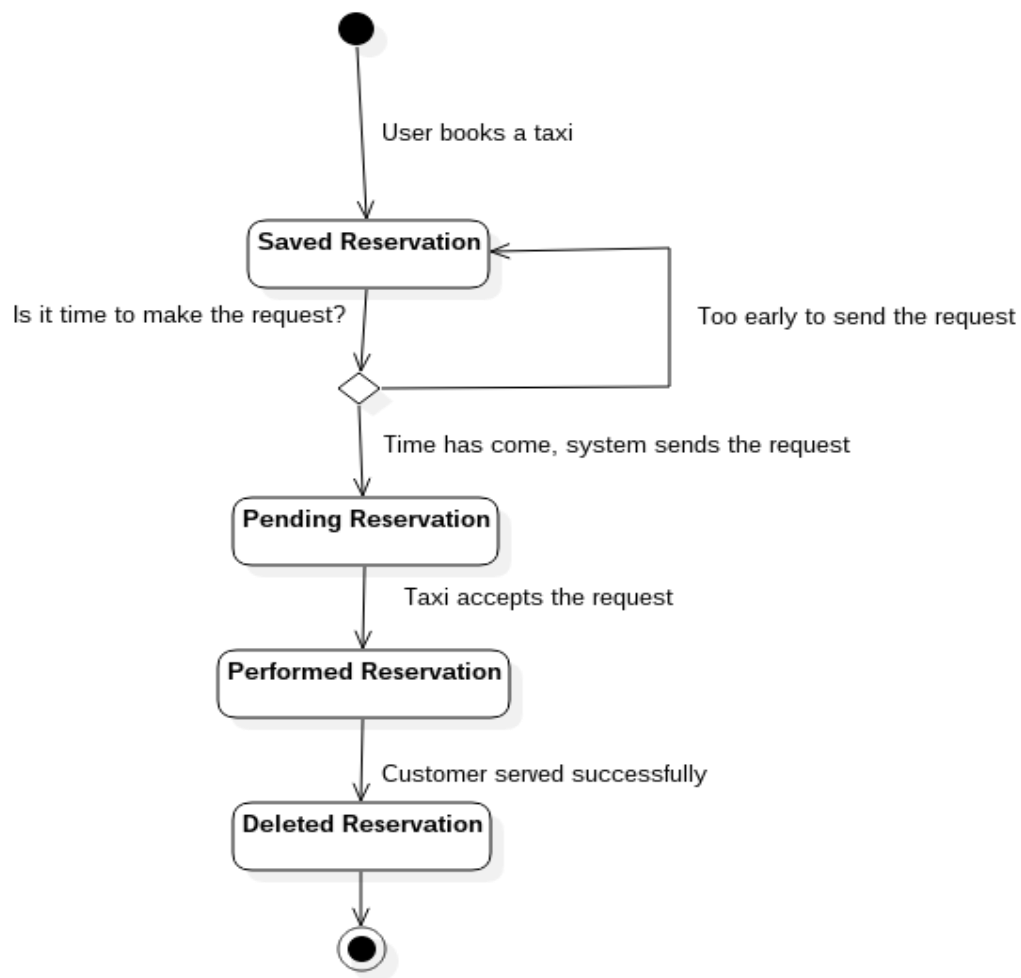
### 3.7.3 State chart Diagrams

The previous dynamics are here analysed state by state through the State chart diagrams. In particular it will be presented the Taxi Call State Diagram, the Taxi Reservation State Diagram and the Taxi Registration State Diagram.

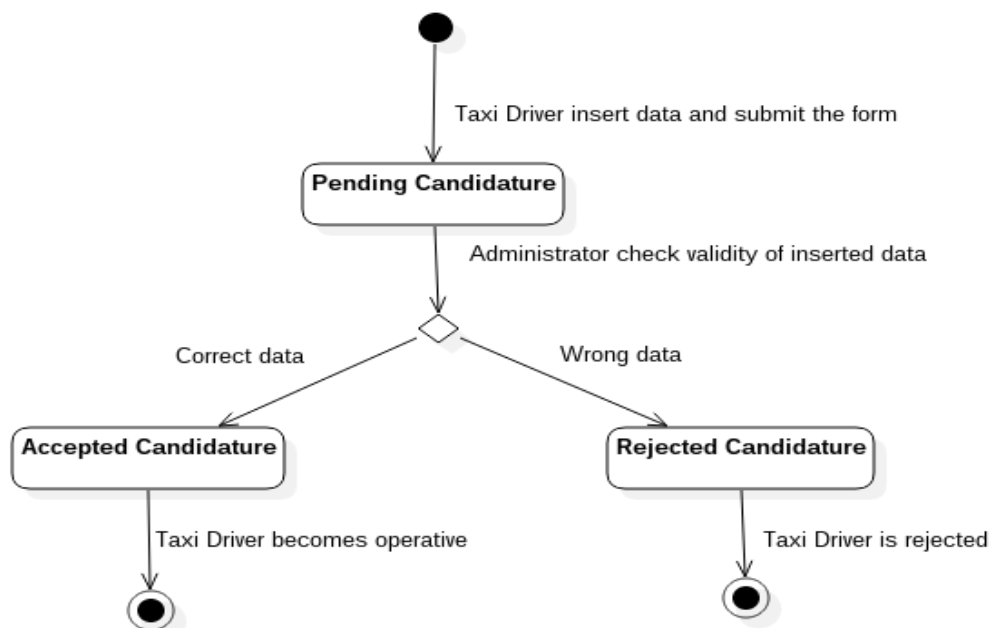
#### *Taxi Call*



## *Taxi Reservation*



## *Taxi Registration:*





## 4. Model Coherence Analysis

### 4.1 Alloy Code

In this paragraph it will be presented the alloy code that model this system. The focus is on “facts” that define the rules and the constraints that must holds in order to maintain consistency with respect to what has been described in this Document:

```
module MyTaxi

//SIGNATURES

sig Request{
    client: one Guest,
    taxi: one TaxiDriver,
    zoneQueue: one Queue
}

sig Reservation{
    client: one User,
    generatedRequest: one Request
}

sig Guest{
    request: lone Request
}

sig User extends Guest{
    reservation: lone Reservation
}

sig TaxiDriver{
    actualRequest: lone Request,
    actualQueue: lone Queue
}

sig Queue{
    queuedTaxis: set TaxiDriver
```

```
}
```

```
//FACTS
```

```
//request facts
```

```
fact{
```

```
    client = ~request
```

```
    actualRequest = ~taxi
```

```
//every taxi associated to a request must be in the queue selected  
by the request
```

```
    all q: Queue, t: TaxiDriver, r: Request | t in q.queuedTaxis  
and r.taxi = t
```

```
    implies r.zoneQueue=q
```

```
//every user who made a reservation must be also the client of the  
generated request
```

```
    all r: Request, u: User, s: Reservation | s.client = u and  
s.generatedRequest = r
```

```
    implies r.client=u
```

```
}
```

```
//queue facts
```

```
fact{
```

```
    actualQueue = ~queuedTaxis
```

```
//the taxi driver that accept a request have to be listed in the  
actual queue identified by the request itself
```

```
    all q:Queue, t:TaxiDriver, r:Request | r.taxi=t and  
r.zoneQueue=q
```

```
    implies t in q.queuedTaxis
```

```
}
```

```
//reservation facts
```

```
fact{
```

```
    client = ~reservation
```

```
//only at most one reservation for every request
    all b,b': Reservation | b.generatedRequest =
b'.generatedRequest
        implies b = b'
}

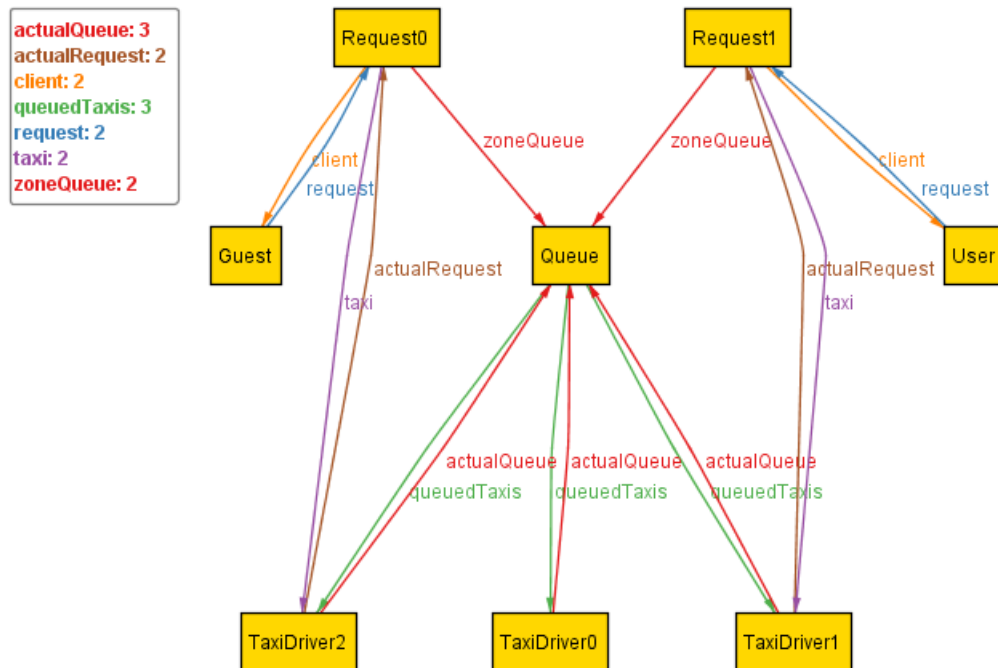
pred show1{
    #Queue=1
    #Reservation=0
    #Guest > 0
    #User > 0
}

pred show2{
    #Queue=2
    #Reservation=1
    #Guest > 0
    #User > 0
}

run show1 for 3
run show2 for 3
```

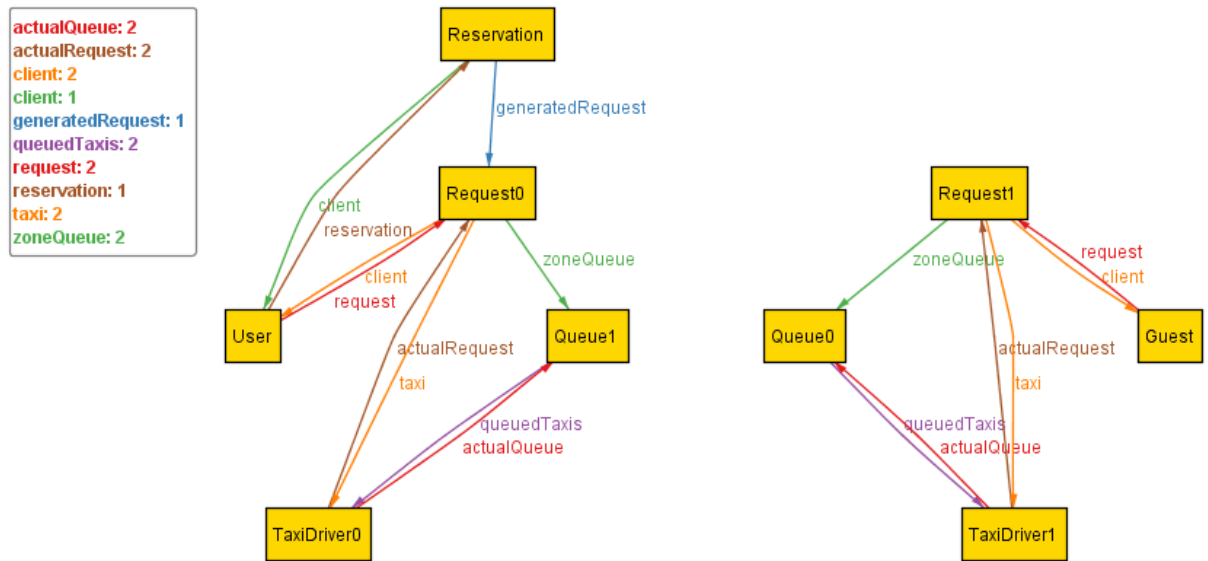
## 4.2 Generated World

### World 1



In this world it is shown a classic situation in which guests and users make requests, and they are paired with the taxi that accepts the call. It is clear that obviously every request is generated by only one client, and only one taxi will answer the call. Every queue contains the requests made in that zone, and there can be taxis that are available in the queue but not busy yet.

## World 2



In this particular world it is shown a reservation case. A registered user can make a reservation that will generate a request. This request will be processed as a normal request made by the user, and so associated to a queue and accepted by a taxi. It is also present a second queue just to show that every queue works independently.

## 5. Tools

- Star UML: Sequence Diagram, Use Case Diagram, State chart Diagram, Domain Class Diagram;
- Balsamiq Mockup: User Interfaces, both browser and mobile version;
- Microsoft Word to build RASD document;
- Alloy 4.2: creation of the alloy model;