



Politecnico di Milano

A.A. 2015-2016  
Software Engineering 2: “myTaxiService”  
**R**equirements **A**nalysis and **S**pecification **D**ocument

Morenzetti Sergio 791558.  
Marcucci Antonio Pio 854473.

# **1 Introduction**

## **1.1 Purpose**

This document represents the Requirement Analysis and Specification Document (RASD), it explains both the application domain and the system to be developed. The main goal of this document is to describe the system in terms of functional and non-functional requirements, understand stakeholders' request in order to model the system, show the constraints and the limit of the software and simulate the typical use cases that will occur after the development. This document is intended to all developers and programmers who have to implement the requirements and to system analysts who want to integrate other systems with this one.

## **1.2 Actual System**

The government of a large city aims optimizing its taxi service through a new online taxi application. We suppose that until now nothing has been created and we have to start creating the entire application without using or modify a previous system.

## **1.3 Scope**

The aim of the project is to create a new online taxi service for helping people requesting taxi service.

Passengers can request a taxi either through a web application or a mobile app. Only registered users can request a taxi. It is also possible a special request in which user can specify the origin and the destination of the race. This type of reservation has to occur at least two hours before the ride. In this case, the system confirms the reservation to the user and allocates a taxi to the request 10 minutes before the meeting time with the user.

The system answers to the request by informing the passengers about the code of the incoming taxi and the waiting time, only if it's accepted by a taxi driver.

Otherwise the system notifies to the customer temporary unavailability of a taxi. This is possible thanks to the mobile app that can be also used by taxi drivers in order to inform the system about their situation.

The system guarantees a fair management of taxi queues and automatically computes the distribution of taxi in various zones based on the GPS information it receives from each taxi (each zone is extended for about  $2\text{km}^2$ ).

When a request arrives from a certain zone, the system forwards it to the first available taxi queuing in that zone.

## **1.4 Actors**

Visitor: all visitor users can only see the login page and complete the registration form to be able to access to all the functionality of the application as registered user.

Registered user: this type of user, after successful login, is the only that can use the taxi service interface.

Taxi driver: is a registered user who, by using the mobile app, offers taxi service declaring himself available and answering to requests.

Customer : is a registered user who, by using the mobile app or the web application, can request a taxi, make a reservation and edit it.

### **1.5 Goals**

List of the goals of myTaxiService application:

[G1] Every registered user is univocally identified.

[G2] Wants to simplify the access of registered passengers to the taxi service, allowing customers to make request via mobile app or web application.

[G3] Wants to guarantee a fair management of taxi queues.

[G4] Allow customers to edit a reservation of a taxi.

### **1.6 Definitions, Acronyms, Abbreviations**

#### **1.6.1 Definitions**

Taxi request : any registered user as customer demand of an available taxi in a specific zone.

Schedule of the ride: date & time of the programmed taxi reservation.

Taxi reservation : a special taxi request characterized by the specification of the origin , the destination and the schedule of the ride.

Taxi zones : is a city's zone which has size of, approximately,  $2\text{km}^2$ .

#### **1.6.2 Acronyms**

RASD: Requirements Analysis and Specification Document.

DB: DataBase.

DBMS: DataBase management system.

OS: Operating System.

JVM: Java Virtual Machine.

DMZ: Demilitarized zone.

#### **1.6.3 Abbreviations**

[Gn]: n-goal.

[Rn]: n-functional requirement.

### **1.7 Reference Documents**

Specification Document: Rasd meteocal example 2.pdf.

IEEE standard for requirement specification

## **1.8 Document overview**

This document is essentially structured in four part:

Section 1: Introduction, it gives a description of document and some basic information about software.

Section 2: Overall Description, gives general information about the software product with more focus about constraints and assumptions.

Section 3: Specific Requirements, such as functional and non functional, typical scenarios and use cases. To give an easy way to understand all functionality of this software, this section is filled with UML diagrams.

Section 4: Appendix, this part contains some information about the attached .als file.

## **2 Overall Description**

### **2.1 Product perspective**

The application we are planning is a web application and a mobile app which is not integrated with other existing system. It is only user based and does not need an administrator interface.

### **2.2 User characteristics**

The user that we expect to use our application is a person who want an easy and fast way to request a taxi. This user must be able to use a web browser or a Smartphone and have access to internet.

### **2.3 Constraints**

#### **2.3.1 Hardware limitations**

MyTaxiService doesn't have to meet any hardware limitations.

#### **2.3.2 Parallel operation**

MyTaxiService must support parallel operations from different users when working with database and with all operation done by the user after connection.

#### **2.3.3 Regulations**

MyTaxiService doesn't have to meet any regulation policies.

## **2.4 Assumptions and Dependencies**

### **2.4.1 Assumption**

- There isn't an administrator or privileged user.
- There isn't any dependence between users;
- Is not possible, for the same customer, to make a request before receiving the response of the previous one.
- There is no a fixed limit number of possible of request per day.
- If a taxi driver deletes his service, the waiting customer will be informed about it and reinserted in the list of request, with a certain priority.
- The taxi model is strictly connected to the taxi code and the taxi number plate; the application get this information from the official taxi company DB of the city.
- The Customer cannot decline the taxi service once requested.
- The list of taxi drivers in myTaxiService app is based on the official taxi company DB of the city.
- If the automatic localization of the customer's smartphone doesn't work efficiently , it is possible to provide a city address for the taxi request.
- If a taxi driver receives a phone call for a ride while he is in the available taxi queue, he can accept it, but he have to notify the system through the application that he is now busy, and that he can be removed from that list.
- myTaxiService is available only in the city zones.

### **2.5 Future possible implementation**

Future possible implementation can be additional services, like taxi sharing, payment using the app, etc. on top of the basic one. A user can enable the taxi sharing option. This means that he / she is ready to share a taxi with others if possible, thus sharing the cost of the ride. In this case the user is required to specify the destination of all ride which he/she wants to share with others. If other are willing to start a share ride from the same zone going in the same direction, then the system arranges the route for the taxi driver, defines the fee for all persons sharing the taxi and informs the passengers and taxi driver.

## **3 Specific Requirements**

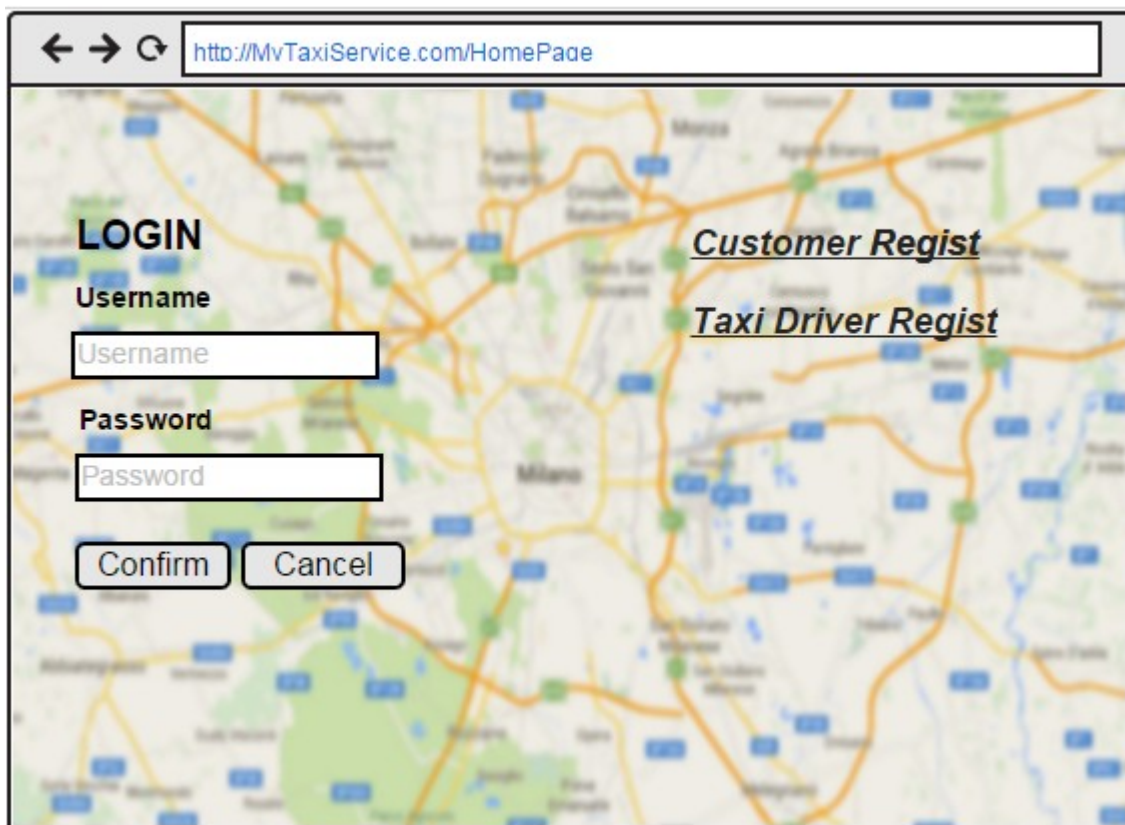
### **3.1 External Interface Requirements**

### **3.1.1 User Interfaces**

Here are presented some mockups that represent an idea of the structure of the application pages:

#### **3.1.1.1 Home Page**

The mockup above shows the home page of MyTaxiService. Here both customers and taxi drivers can log into the application and visitors can access to the registration form.



The mockup shows a web browser window with the address bar displaying <http://MvTaxiService.com/HomePage>. The main content area features a background map of Milan. Overlaid on the map is a login form on the left and two registration links on the right. The login form includes a 'LOGIN' heading, 'Username' and 'Password' labels, corresponding input fields, and 'Confirm' and 'Cancel' buttons. The registration links are 'Customer Regist' and 'Taxi Driver Regist', both underlined.

**LOGIN**

Username

Password

*Customer Regist*

*Taxi Driver Regist*

#### **3.1.1.2 Registration forms**

This mock shows the registration form page for customers.

← → ↻ <http://MyTaxiService.com/RegistrationForm>

**Name**

**Telephone number**

**Surname**

**Username**

**E-mail**

**Password**

**Birth date**

**Repeat Password**

**Address**

This mock shows the registration form page for taxi drivers.

← → ↻ <http://MyTaxiService.com/RegistrationForm>

<b>Name</b>	<b>Telephone number</b>	<b>Taxi code</b>
<input type="text"/>	<input type="text"/>	<input type="text"/>
<b>Surname</b>	<b>Username</b>	<b>Taxi number plate</b>
<input type="text"/>	<input type="text"/>	<input type="text"/>
<b>E-mail</b>	<b>Password</b>	
<input type="text"/>	<input type="text"/>	
<b>Birth date</b>	<b>Repeat Password</b>	
<input type="text"/>	<input type="text"/>	

### **3.1.1.3 Personal page**



This mockup shows the personal page of an hypothetical customer. On the left side customer can request a simple taxi service, on the right side customer can request a reservation. In both case customer can choose taxi model. When the model is specified, the application knows that “normal” means a maximum of 5 people per car, in opposite to “large” that means a maximum of 7 people. For the simple request if the GPS doesn’t work the customer can select the starting position using a menu in which there are all the city’s streets. Moreover customer can click a specific button to go on the specific page to modify his account or his reservations.

← → ↻ <http://MyTaxiService.com/PersonalPage>

**Welcome "Username"**

[Modify Account](#) [Modify reservation](#)

*Indicate your position with GPS*

*Select your starting position*

Madison Street, 8 ▼

*Taxi model* Large ▼

[Request taxi](#)

*Select origin*

Madison Street, 8 ▼

*Select destination*

Madison Street, 8 ▼

*Hour* 12am ▼

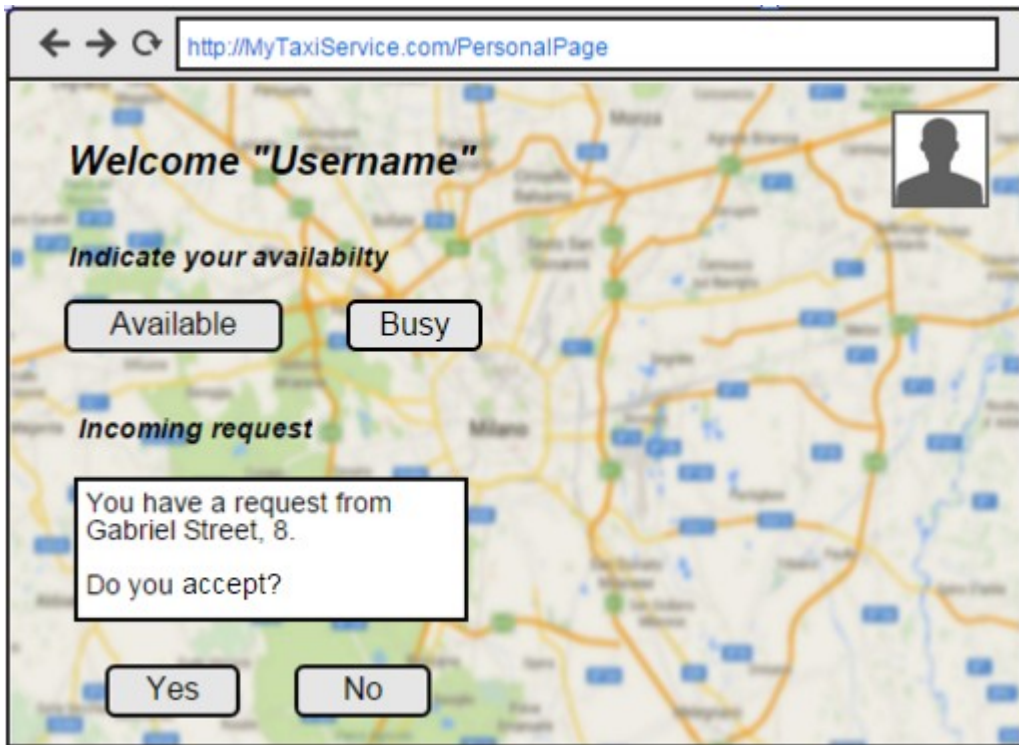
*Date* 4/22/2012 ▼

*Taxi model* Normal ▼

[Reservation](#)

[Home](#)

This mockup shows the home page of an hypothetical taxi driver. In this page the taxi driver can indicate his availability and he/she also can accept or not an incoming request.



#### **3.1.1.4 Edit account page**

This mockup shows the page in which a customer can modify his account details or delete it.

← → ↻ <http://MyTaxiService.com/EditAccount>

**Modify your account**

**Name**

**Surname**

**E-mail**

**Birth date**


**Address**

**Telephone number**

**Username**

**Password**

**Repeat password**



### **3.1.1.5 Edit reservation page**

This mockup shows the page in which a customer can modify his reservation details or delete it, if it is possible.

**Modify your reservation**

Select your reservation 22/10/2015 ▼

**Origin**  
Madison Street,8 ▼ Modify

**Taxi model**  
Large ▼ Modify

**Destination**  
Gabriel Street,8 ▼ Modify

**Hour**  
12px ▼ Modify

**Date**  
4/22/2012 ▼ Modify

Save changes

Delete reservation

### **3.1.3 Hardware Interfaces**

This project does not support any hardware interfaces.

### **3.1.4 Software Interfaces**

- Database Management System (DBMS): MySQL.
- Java Virtual Machine (JVM) : JEE.
- Application server: Glassfish (application server java EE open source).
- Operating System (OS) : Application must be able to run on any os which supports JVM and DBMS specified before.

### **3.1.5 Communication interface**

The internet protocol on which this application is based is TCP and exploit the application https.

### **3.1.6 Memory**

The minimum memory requirements are:

- Primary Memory: 4MB+.

## **3.2 Functional Requirements**

3.2.1 [G1] Every registered user is univocally identified.

- [R1] The access to myTaxiService shall only be granted after that a user types an authorized username and password.
- [R2] After a successful registration, using an unique username, a user receives a confirmation email which authorizes him the access to myTaxiService functionalities.

3.2.2 [G2] Wants to simplify the access of registered passengers to the taxi service, allowing customers to make request via mobile app or web application.

- [R1] A registered user can log into the application writing his username and password.
- [R2] Once registered, a user can access to myTaxiService functionalities just writing username and password.
- [R3] A customer is able to see the answer of the system to his taxi request, eventually with the expecting waiting time.

3.2.3 [G3] Wants to guarantee a fair management of taxi queues.

- [R1] Each city zone is associated to a different queue of taxis.
- [R2] The system automatically computes the distribution of taxis in the various zones based on the GPS information it receives from each taxi.
- [R3] A taxi driver can indicate his availability and his will to accept or not a request, whenever he receive one.
- [R4] A taxi driver can declare himself busy, in order to be deleted from the queue.

3.2.4 [G4] Allow customers to edit a reservation of a taxi.

- [R1] A customer have to specify the origin, the destination and the meeting time of the reservation.
- [R2] A customer is able to do a reservation at least 2 hours before the ride.
- [R3] Allow customer to modify or delete an existing reservations.

## **3.3 Domain Assumption**

-A taxi must belong to at most one queue in every moment;

-Given any pair of city areas, there is no intersection between the two.

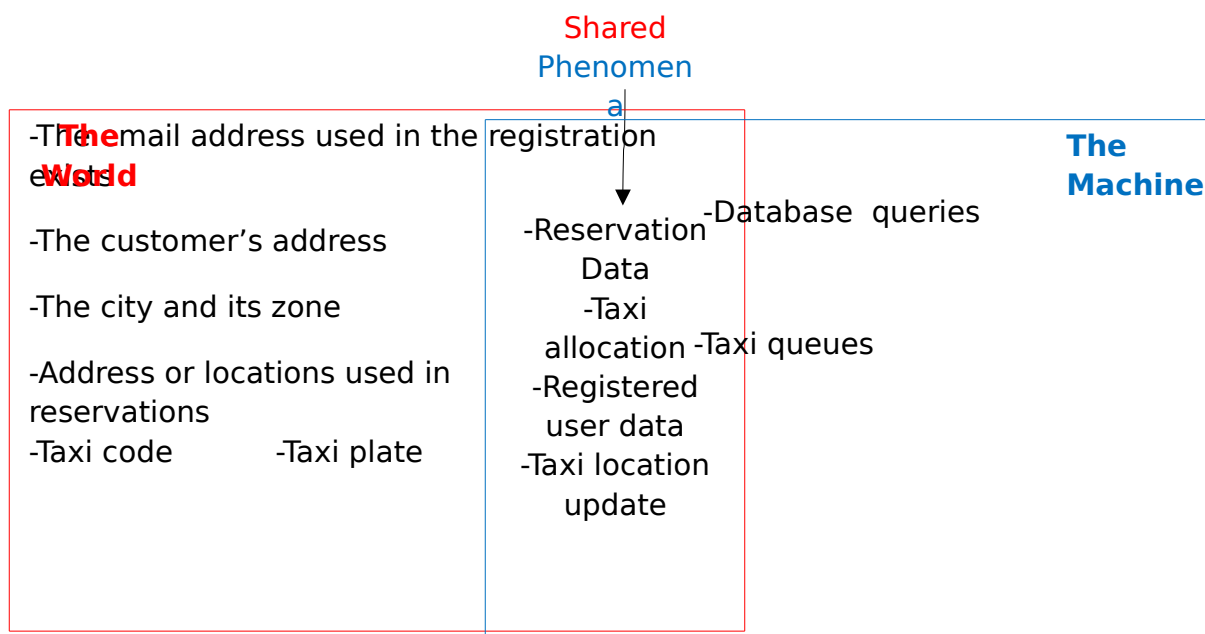


- A taxi can carry at most a number of passengers equal to those for which his car is homologated.
- Once that a taxi driver has accepted a request, and he is on his way to the meeting point, cannot pick up a different passenger on the road.
- Taxi driver must always try to go through the fastest path, to carry a passenger from origin to destination.
- In the reservation, the destination must be different from the starting position.

### 3.4 The world and the machine

We have used “The World & Machine” model to make a first domain analysis of MyTaxiService application.

In this project phase we have identified phenomena occurring in the world, occurring inside the machine and the shared phenomena between the two, that are controlled by the world and observed by the machine or vice versa.



## 3.5 UML MODELS

### 3.5.1 Use Case

#### 3.5.1.1 Registration process of a visitor

Name of Use Case	RegisterVisitor
Actors	<ul style="list-style-type: none"> <li>• Visitor</li> </ul>
Entry condition	The visitor wishes to create an account to access all myTaxiService functionalities.
Flow of Events	<ul style="list-style-type: none"> <li>• The visitor clicks on the register button to start the registration process, in the home page of the application.</li> <li>• The visitor fills all the fields of the registration form.</li> <li>• The visitor clicks on the confirm button.</li> <li>• The application store all the date in the DB</li> <li>• The application sends to the visitor email address a confirmation email.</li> </ul>
Exit condition	<p>The registration process terminates after the confirmation email.</p> <p>Now the visitor is able to log into the application, use his/her credential and start exploit myTaxiService.</p>
Exceptions	<ul style="list-style-type: none"> <li>• If one or more account fields are not properly written or valid (like an existing username, invalid password or an already used taxi code..) the system will ask the visitor to reinsert them correctly.</li> <li>• If the confirmation email never arrives to the visitor, he has to redo the registration process from scratch.</li> </ul>

### ***3.5.1.2 A registered user log into myTaxiService.***

Name of Use Case	Login
Actors	<ul style="list-style-type: none"> <li>• Registered user</li> </ul>
Entry condition	The user wants to connect to myTaxiService application.
Flow of Events	<ul style="list-style-type: none"> <li>• The registered user write his username and password into the corresponding fields on the home page of the application.</li> <li>• The registered user clicks the confirm button.</li> </ul>
Exit condition	The registered user is successfully connected to the application and he is now able to fully use its functionalities
Exceptions	If the log-in response is negative, the system asks the registered user to enter the password again.

### ***3.5.1.3 Modification process of a user.***

Name of Use Case	ModifyAccount
------------------	---------------

Actors	<ul style="list-style-type: none"> <li>Registered User</li> </ul>
Entry condition	The user needs to modify his already existing account.
Flow of Events	<ul style="list-style-type: none"> <li>The Registered user logs into myTaxiService, with his username and password.</li> <li>The user clicks on the modify account button, on the personal page.</li> <li>The system queries the DB, and provides to the registered user all his account data.</li> <li>The user modify the account fields that he intended to change.</li> <li>The user clicks on the button "Save Changes".</li> <li>The system store the modified data in the DB.</li> <li>The application sends to the registered user a confirmation email.</li> </ul>
Exit condition	<p>The modify process terminates after the confirmation email.</p> <p>Now the registered user has successfully modified his data.</p>
Exceptions	<ul style="list-style-type: none"> <li>If one or more account fields are not properly written or valid the system will ask the visitor to reinsert them correctly.</li> <li>If the confirmation email never arrives to the registered user, he has to redo the modification process.</li> </ul>



#### **3.5.1.4 Account deletion of a user.**

Name of Use Case	DeleteAccount
Actors	<ul style="list-style-type: none"><li>Registered user</li></ul>
Entry condition	The registered user wants to delete his account.
Flow of Events	<ul style="list-style-type: none"><li>The registered user logs into myTaxiService, with his username and password.</li><li>The registered user clicks on the modify account button, on the personal page.</li><li>The system queries the DB, and provides to the registered user all his account data.</li><li>The user clicks on the delete button.</li><li>The system deletes user's data from the DB.</li><li>The application sends to the registered user a delete confirmation email.</li></ul>
Exit condition	The delete process terminates after the confirmation email. Now the user has successfully deleted his account.
Exceptions	<ul style="list-style-type: none"><li>If one or more account fields are not properly written or valid the system will ask the user to reinsert them correctly.</li><li>If the confirmation email never arrives to the user, he has to redo the delete process.</li></ul>

#### **3.5.1.5 Customer requests a taxi**

Name of Use Case	RequestTaxi
Actors	<ul style="list-style-type: none"><li>Customer</li></ul>
Entry condition	The customer needs a taxi
Flow of Events	<ul style="list-style-type: none"><li>The customer logs into his account via mobile app or web application</li><li>The customer, in the "request taxi" section, specifies his position via GPS or via scroll down menu, and the taxi model.</li><li>The customer clicks on the request button.</li><li>The system notifies that the request has been correctly received.</li></ul>
Exit condition	<ul style="list-style-type: none"><li>The request process terminates when customer has successfully send his request.</li></ul>
Exceptions	If the starting position indicate through GPS service is not served by myTaxiService the system notify him to insert a correct position.

### **3.5.1.6 Request forwarded to a taxi**

Name of Use Case	ForwardRequest
Actors	<ul style="list-style-type: none"><li>• Customer, Taxi driver</li></ul>
Entry condition	A customer has sent a request for a taxi
Flow of Events	<ul style="list-style-type: none"><li>• Once the system has received the passenger's request, it computes the distribution of taxis in the area. Once the first driver of the considerate queue is selected, the systems sends him a notification providing the meeting point address.</li><li>• The driver has to send an answer to the request:<ul style="list-style-type: none"><li>-if the driver accept the request he has to send the answer to the system, through the right button, which will inform the passenger; The system have to remove the taxi driver from the queue and compute the waiting time.</li><li>- if the driver decline the request he has to send the answer to the system, through the right button, which will send at the end of the queue the taxi driver and resend a request to the new first taxi driver.</li></ul></li><li>• The customer receives the answer to his request.</li></ul>
Exit condition	A driver has accepted the request.
Exceptions	<ul style="list-style-type: none"><li>• If there isn't any available taxi drivers for the zone of the incoming request the system has to provide one in an adjacent zone.</li><li>• If there isn't any available taxi drivers the system notify the passenger inviting him to try later or moving in another position and retry the request.</li></ul>

### **3.5.1.7 Customer reserves a taxi**

Name of Use Case	ReserveTaxi
Actors	<ul style="list-style-type: none"><li>• Customer</li></ul>
Entry condition	The customer needs a taxi reservation
Flow of Events	<ul style="list-style-type: none"><li>• The customer logs into his account via mobile app or web application</li><li>• The customer indicate the starting and the destination position via scroll down menu, the date, the meeting time and the taxi model.</li><li>• The customer clicks on the reservation button.</li><li>• The system notifies that the request has been correctly received.</li></ul>

Exit condition	<ul style="list-style-type: none"> <li>The reservation process terminates when customer has successfully send his request.</li> </ul>
Exceptions	<ul style="list-style-type: none"> <li>If for the indicate date or meeting time there are not available taxi the system notify the customer that his reservation will not be done.</li> <li>If meeting time is smaller than 2 hours + time in which request was sent, the reservation is not accepted and the passenger is required to correct the meeting time in order to finalize his reservation.</li> </ul>

### **3.5.1.8 Customer modifies a reservation**

Name of Use Case	ModifyReservation
Actors	<ul style="list-style-type: none"> <li>Customer</li> </ul>
Entry condition	The customer needs to modify a taxi reservation
Flow of Events	<ul style="list-style-type: none"> <li>The customer logs into his account via mobile app or web application</li> <li>The customer select one of the existing reservations from his account page.</li> <li>The customer modifies those fields that need to be modified</li> <li>The customer clicks on the “save changes” button.</li> <li>The system notifies that the reservation has been correctly modified.</li> </ul>
Exit condition	<ul style="list-style-type: none"> <li>Customer has successfully modified his reservation.</li> </ul>
Exceptions	If customer has modified some fields adding invalid informations, the system notify him to reinsert data correctly.

### **3.5.1.9 Customer deletes a reservation**

Name of Use Case	DeleteReservation
Actors	<ul style="list-style-type: none"><li>• Customer</li></ul>
Entry condition	The customer needs to delete a taxi reservation
Flow of Events	<ul style="list-style-type: none"><li>• The customer logs into his account via mobile app or web application</li><li>• The customer select one of the existing reservations from his account page.</li><li>• The customer clicks on the “delete reservation” button.</li><li>• The system notifies that the reservation has been correctly deleted.</li></ul>
Exit condition	<ul style="list-style-type: none"><li>• Customer has successfully deleted his reservation.</li></ul>
Exceptions	If customer try to delete a reservation that will occurs in 2 hour or less, the system notify him that is not possible.

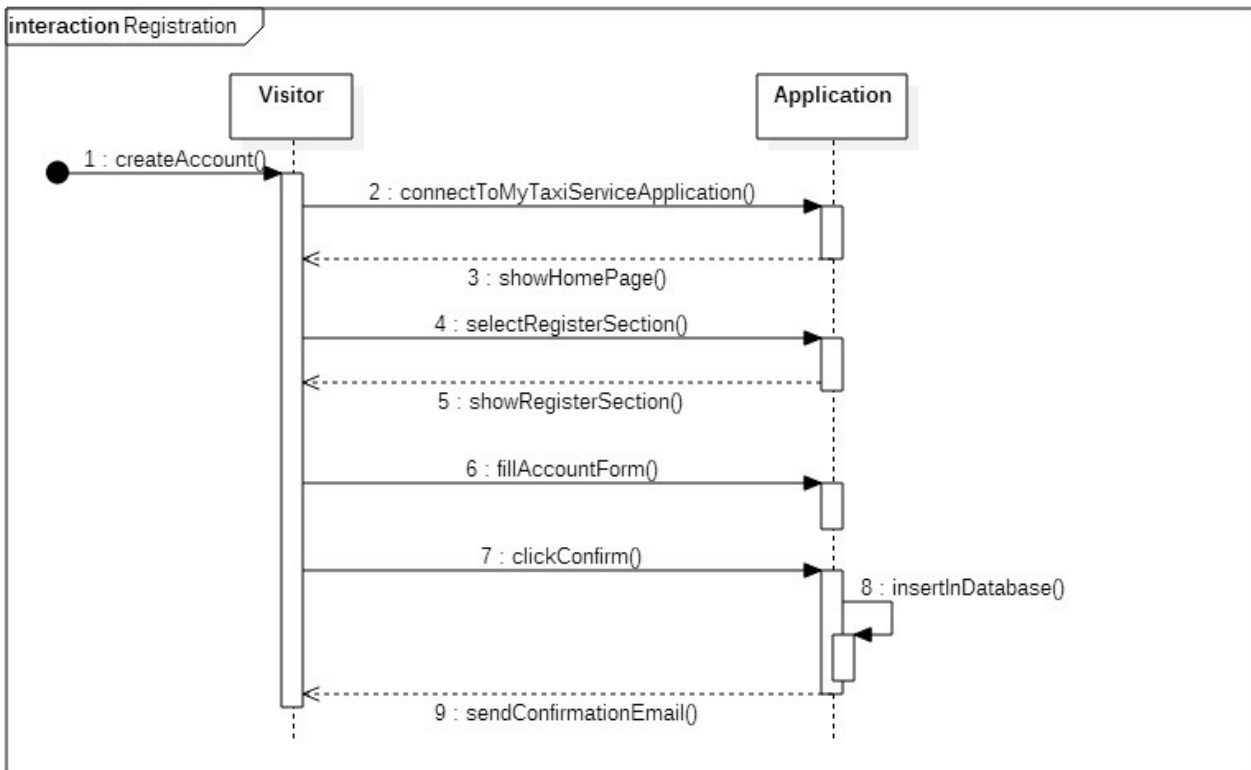
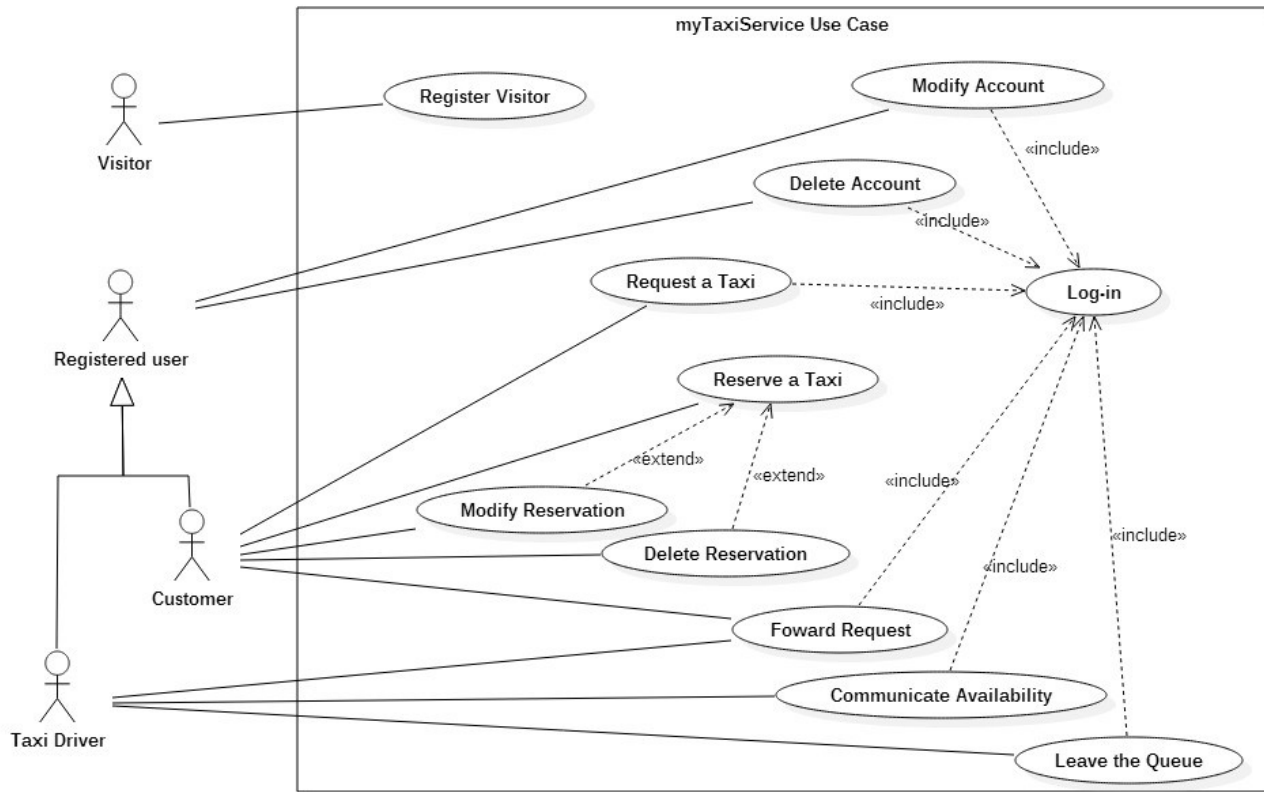
### **3.5.1.10 Availability process of a taxi driver.**

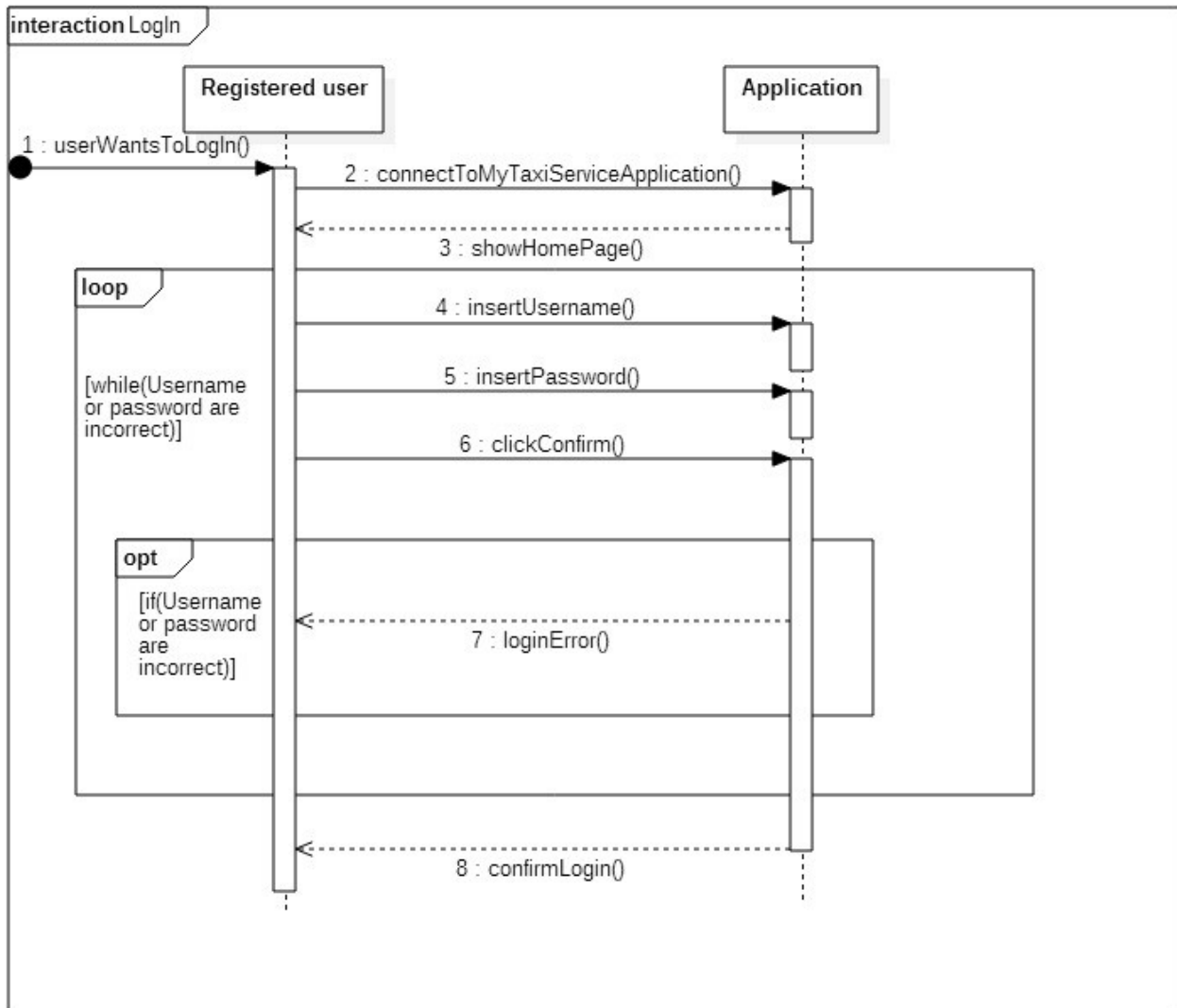
Name of Use Case	CommunicateAvailability
Actors	<ul style="list-style-type: none"><li>• Taxi driver</li></ul>
Entry condition	The taxi driver wants myTaxiService application knows that he is ready for a ride, and wishes to be moved in the queue of taxis.
Flow of Events	<ul style="list-style-type: none"><li>• The taxi driver logs into myTaxiService, with his username and password.</li><li>• The taxi driver clicks the “available” button.</li><li>• The system computes the association between the position received by the taxi driver’s GPS and the corresponding city zone, and inserts the taxi code in the appropriate queue.</li><li>• The application send a confirmation notification to the taxi driver.</li></ul>
Exit condition	The process ends when the confirmation is sent to the driver.
Exceptions	<ul style="list-style-type: none"><li>• If the taxi has already declared as available, and the driver click on available button, the system communicates him that he already is placed in a</li></ul>

	queue.
--	--------

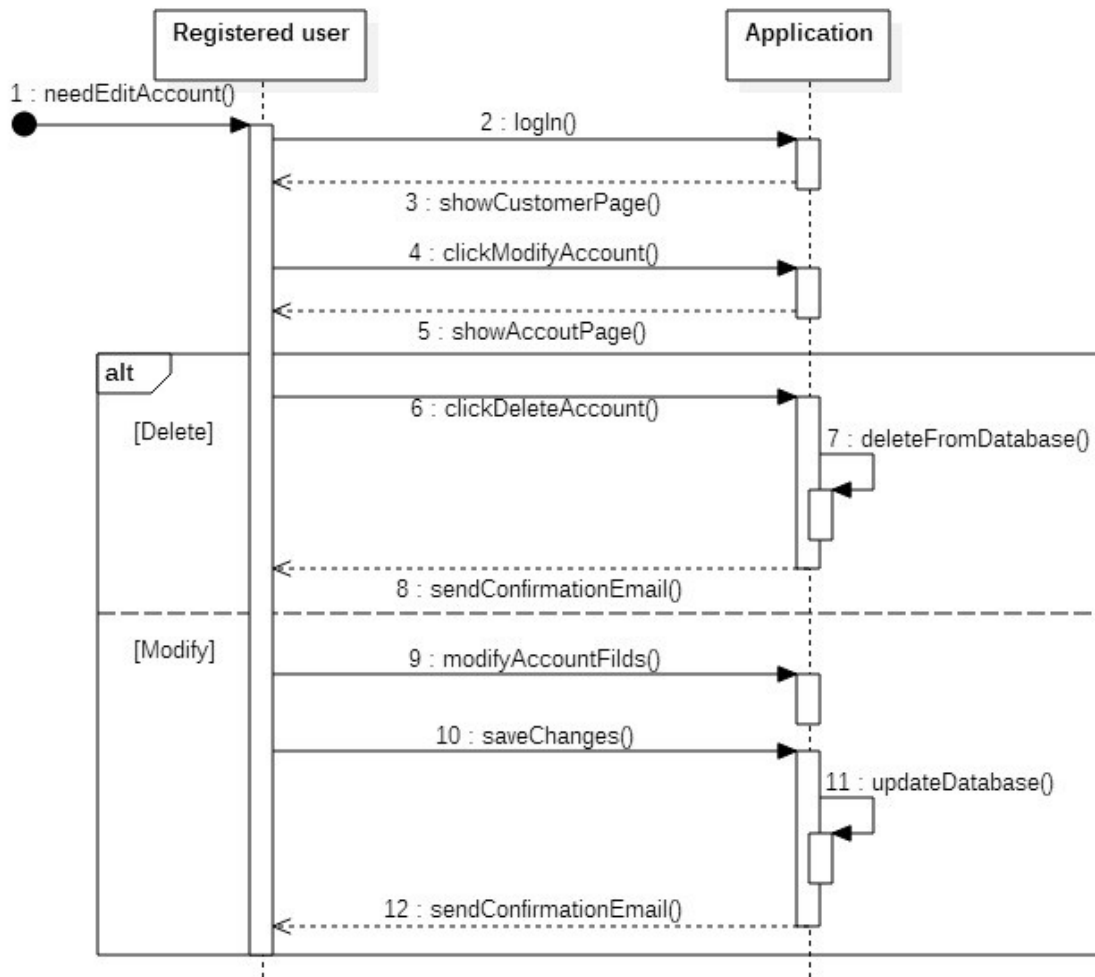
#### **3.5.1.11 Exiting process from the queue of a taxi driver.**

Name of Use Case	LeaveTheQueue
Actors	<ul style="list-style-type: none"> <li>• Taxi driver</li> </ul>
Entry condition	The driver has picked up a passenger along the way, or has to stop working or for other reasons: he is now busy and cannot provide taxi service through the application anymore. He needs to be removed from his queue of taxis.
Flow of Events	<ul style="list-style-type: none"> <li>• The taxi driver logs into myTaxiService, with his username and password.</li> <li>• The taxi driver chooses the application button "busy".</li> <li>• The system remove the taxi code from its queue.</li> <li>• The application send an acknowledgement to the driver</li> </ul>
Exit condition	<p>The process ends when the acknowledgement is sent to the driver.</p> <p>The driver is not on the available taxis queue anymore.</p>
Exceptions	<ul style="list-style-type: none"> <li>• If the taxi is already busy, do not belong to any queue or has already accepted a request the busy option is not available.</li> </ul>



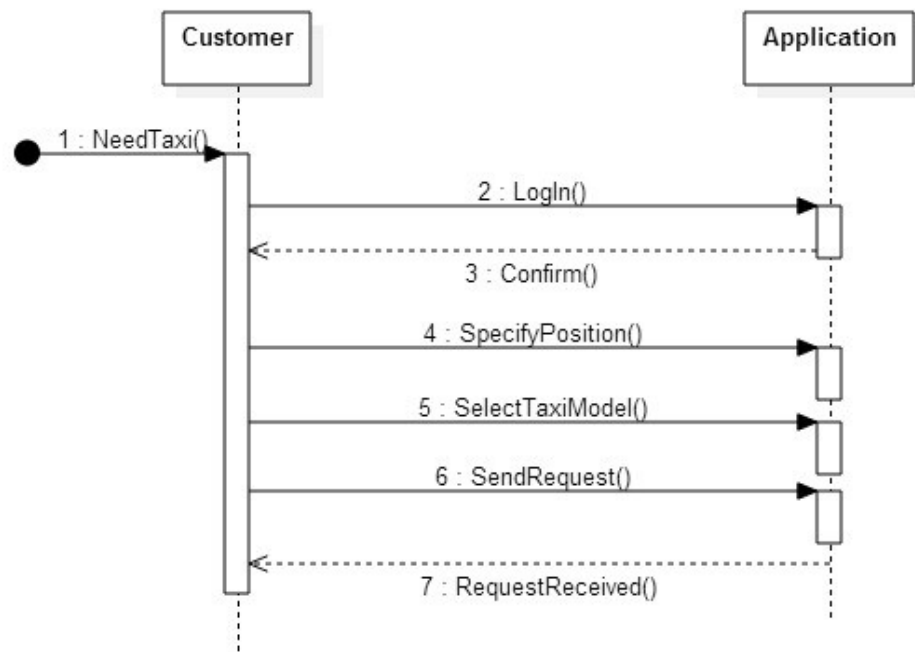


interaction Modify/Delete Account

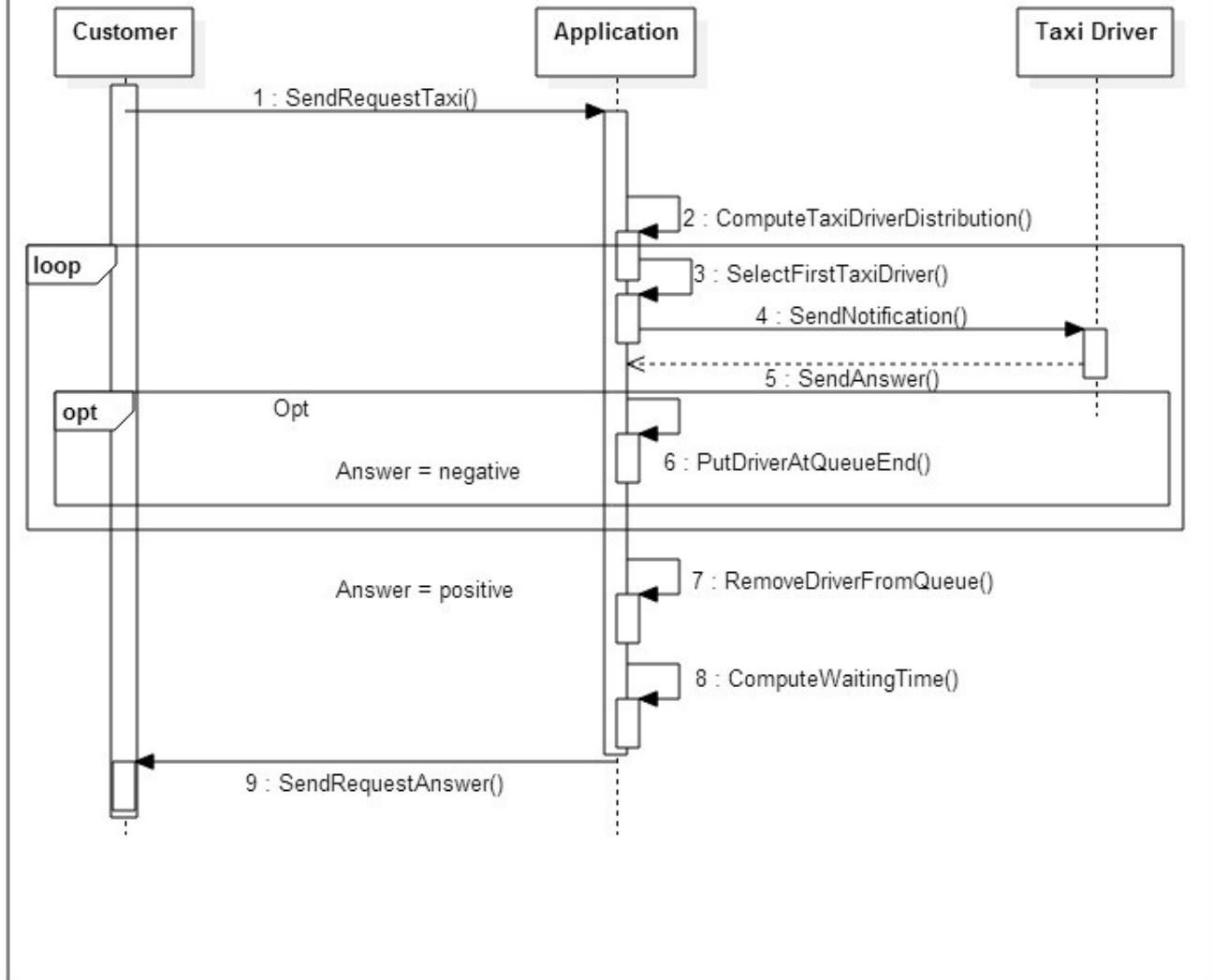




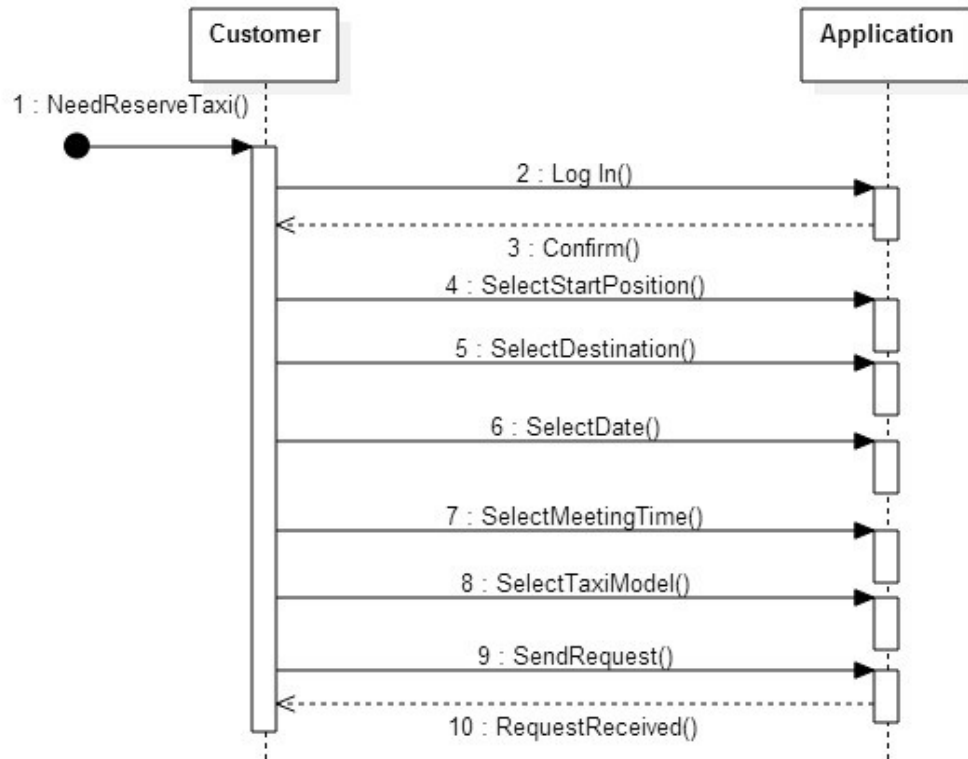
interaction Request Taxi



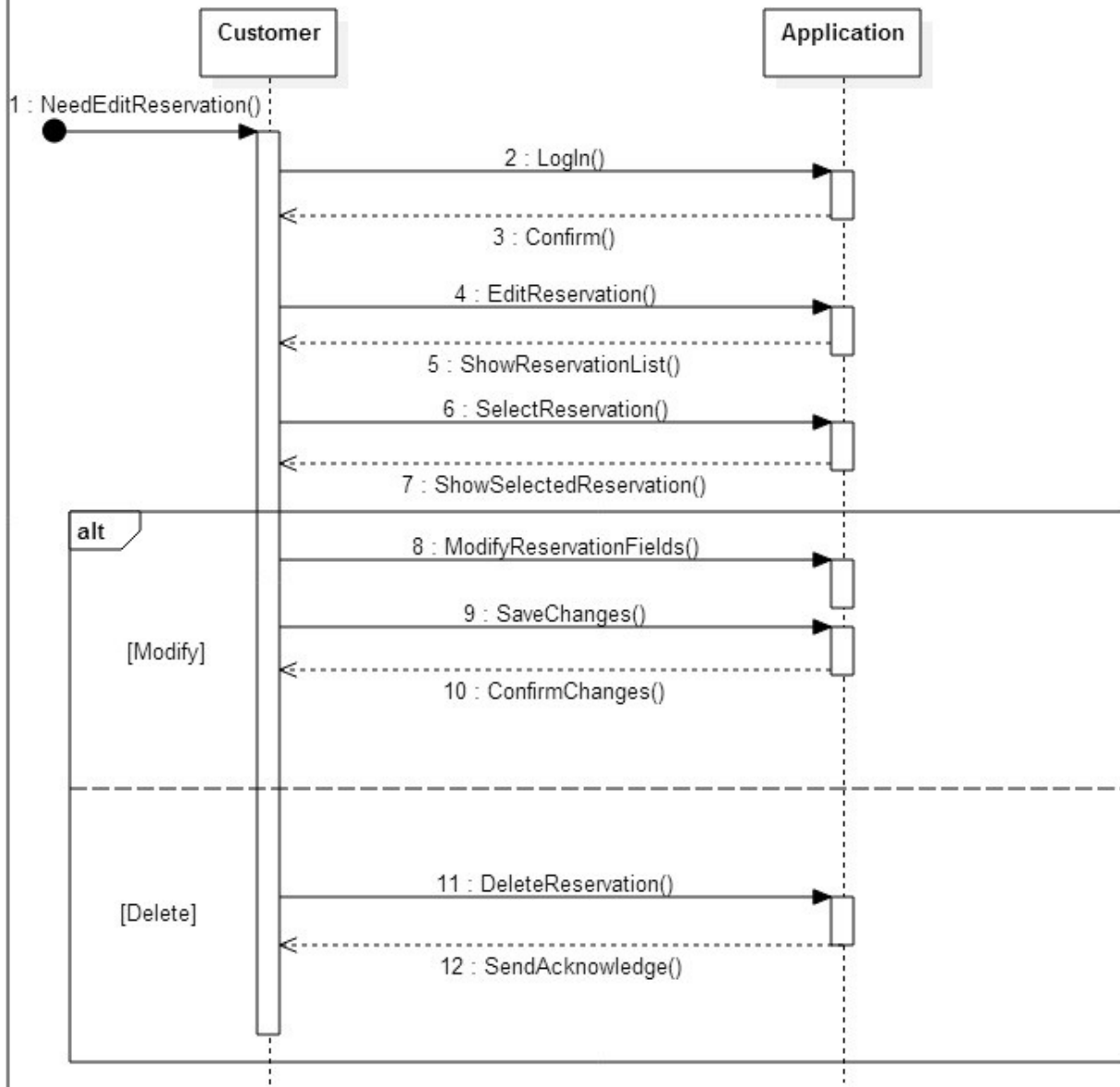
interaction Request Forwarded



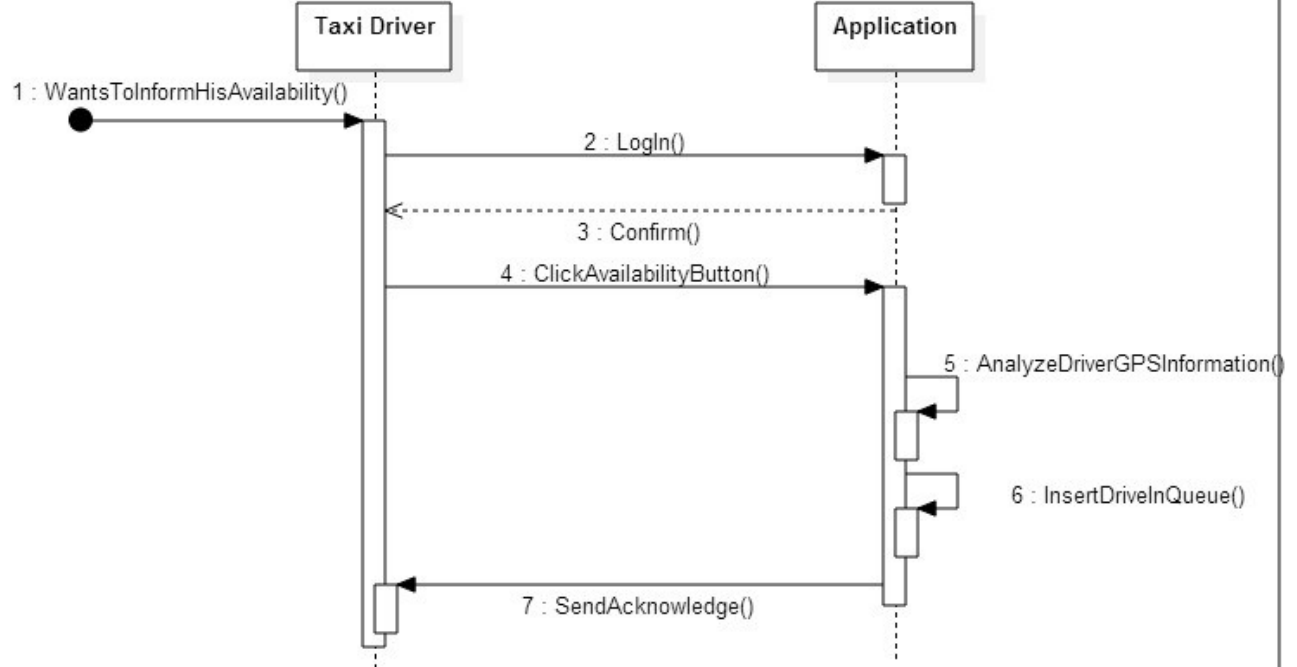
# interaction Reserve Taxi

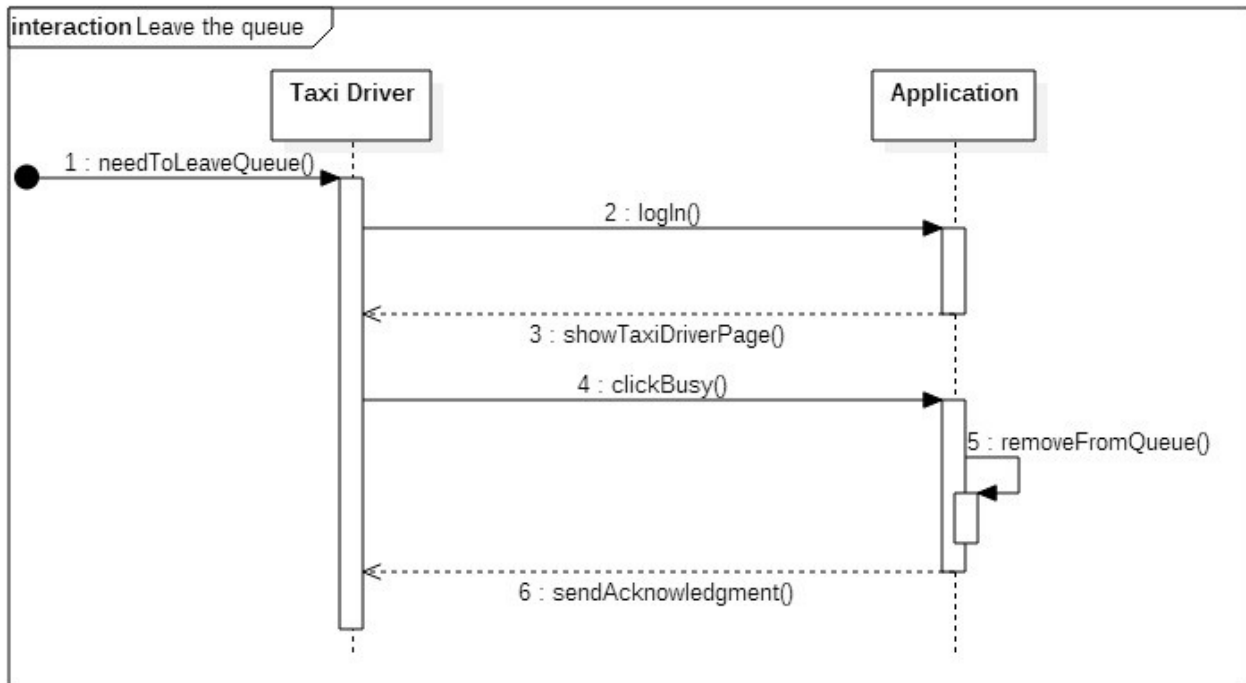


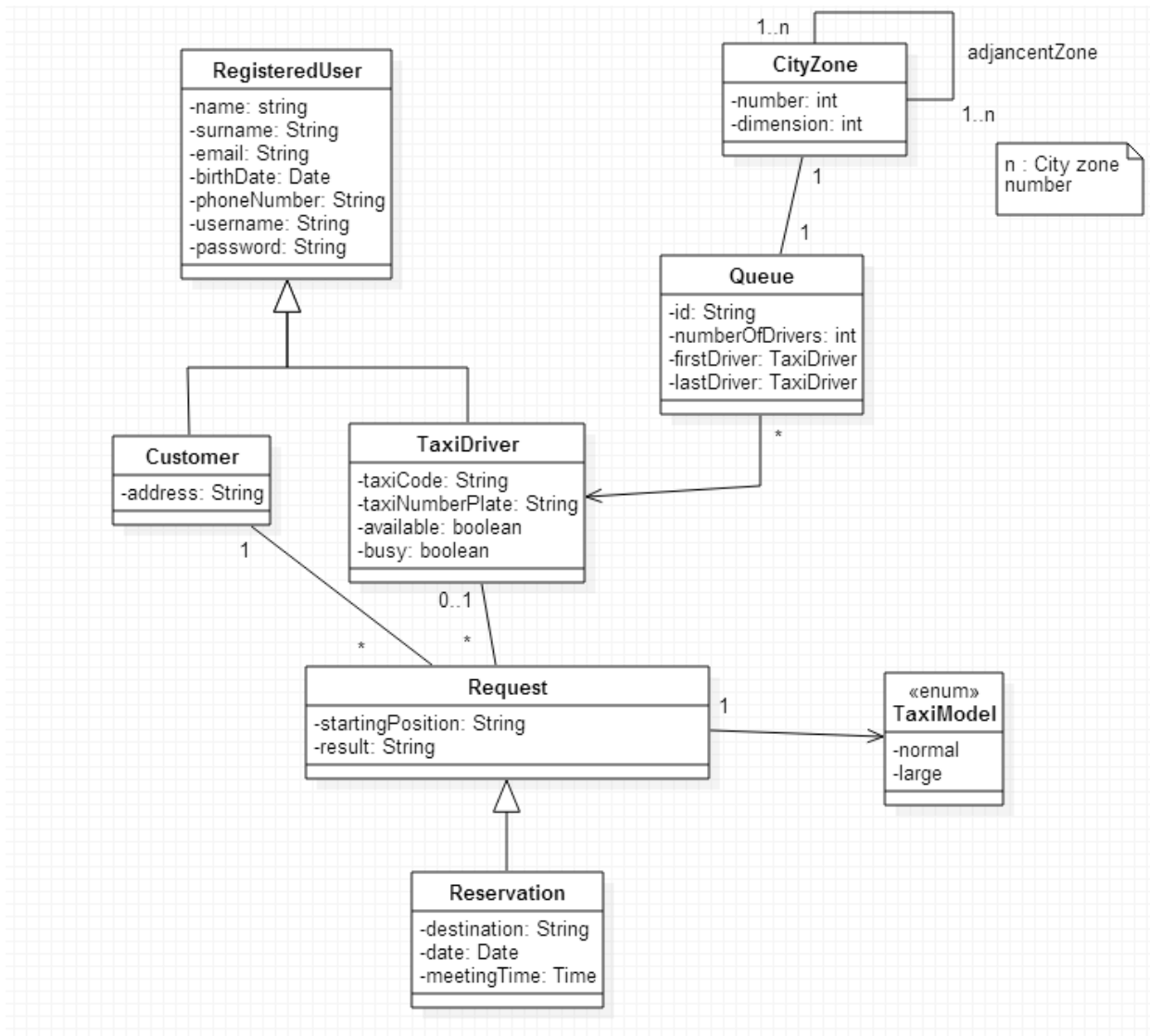
interaction Modify/Delete Reservation



**interaction** Communicate Availability







## 3.6 Scenarios

### 3.6.1 Scenario 1

Marcus has just left a Disco club which is not close his place, so he decides to take a taxi instead of walking because it would take too much time, and it's already too late.

Calling several numbers of taxi services of the city, he is not able to talk with an operator, because there are too much requests at the same time, from the same zone.

Fortunately Marcus has just registered in MyTaxiService, and with his Smartphone, he can easily log in, and then requests a taxi writing the club's address.

After a few minutes he will receive a confirmation answer by the system to his request, containing also the informations of the code of the incoming taxi and the waiting time.

### **3.6.2 Scenario 2**

Martina has a very important work meeting in London in 2 days, and she has the plane very early in the morning the same day.

She wants to be sure about being at the airport in advance, so she doesn't feel like take risks by calling a taxi the same morning of the departure.

So Martina decides to reserve a taxi for that morning by using MyTaxiService, specifying his home address as starting point, the airport as destination, and the time that she prefers to leave. In this way his "taxi problem" has been solved.

### **3.6.3 Scenario 3**

Romolo and his 5 friends want to meet Numa Pompilio at the Colosseo, but they have to cross all Rome city in order to reach it.

They want to take the same taxi but they know that usually taxi cars have only 4 places for passengers.

So Romolo decides to use MyTaxiService app which allows to choose between the normal or the large taxi car.

After a few minutes the Big taxi car arrived and the 6 guys were able to go to the Colosseo together, also saving moneys.

### **3.6.4 Scenario 4**

MyTaxiService has been suggested to Ulisse as a valid application for requesting taxis in a very simply way. Ulisse wants to give a try to this new application and decides to start the registration process.

He chooses "Argo" as username and then fills all the other mandatory fields.

When he clicks the "confirm" button the registration process fails and the application alerts him that "Argo" is already used as username by another user, and that for this reason he has to choose another username.

So Ulisse restarts the registration process and this time writes "Nessuno" in username field, then he clicks the "confirm" button.

This time the registration process ends correctly and the new profile is created.

The application notifies Ulisse that he successfully registered to MyTaxiService and starting from now he can enjoy all its features.



### **3.6.5 Scenario 5**

Paul has just been noticed that the appointment with his boss this evening has been delayed.

Now he has to modify the taxi reservation that he made the day before using MyTaxiService mobile app: so he connects to MyTaxiService through the web browser of his computer, does the login with his own username and checks for his reservation; but he discovers that he cannot modify it because there is no time left. Indeed a reservation can be edit only 2 hours before the ride. So Paul will take the taxi and will arrive in advance at the appointment.

### **3.6.6 Scenario 6**

John was in doubt if taking a taxi to reach the stadium or not, but in the end he made the request by using MyTaxiService with his mobile phone. Few minutes later, while the taxi was arriving, a car stopped just close to John: it was his friend Michael, who was going to see the football game as well, and offered John a ride. So John accepted, and get in the car.

When the taxi arrived at the destination, Philip, the taxi driver, could not be able to find his customer; he tried to call John, thanks to the account informations, asking about his position.

John apologizes for not showing up, and Philip turns available.

### **3.6.7 Scenario 7**

Robert is a taxi driver, and recently decided to register in MyTaxiService with his taxi. He has just finished bringing a customer to his destination, after a very long and tiring ride of one hour, which led him in zone 2.

Now his taxi is available for the application, but Robert does not feel ready for another passenger right now, he need to take a break. Indeed he refuse with the mobile app of his smartphone the next customer, so he is moved in the last position in the queue of available taxi.

## **3.7 Non Functional Requirements**

### **3.7.1 Performance Requirements**

There are two types of performance requirements to analyze: static and dynamic numerical requirements.

About the static one we can say that the system can handle a large number, in respect to the city population, of simultaneous users and information.

About dynamic numerical requirements we can say that all transactions should be processed in measurable time: the response time of the application must be less than 1 second.

### **3.7.2 Software system attributes**

#### **3.7.2.1 Reliability**

The application server is partially free from technical errors.

#### **3.7.2.2 Availability**

The server must be available 24 hours per day in order to ensure the service accessible online anytime. To achieve this goal could be necessary to use an at least partially replicated system. This solution give more scalability to performance required by the system and could maintain an high level of performance especially in case of full load with a lot of connected users.

#### **3.7.2.3 Maintainability**

The application does not provide any specific API, but the whole application code will be documented to well inform future developers of how application works and how it has been developed.

#### **3.7.2.4 Portability**

The application could be used on any OS which supports JVM and DBMS.

#### **3.7.2.5 Security**

##### **3.7.2.5.1 External Interface**

Side MyTaxiService application implements a login authentication to protect the information of users. Password of user is saved using hashing mechanism but could not be enough. This system requires an 8 character password with number, letter and special character.

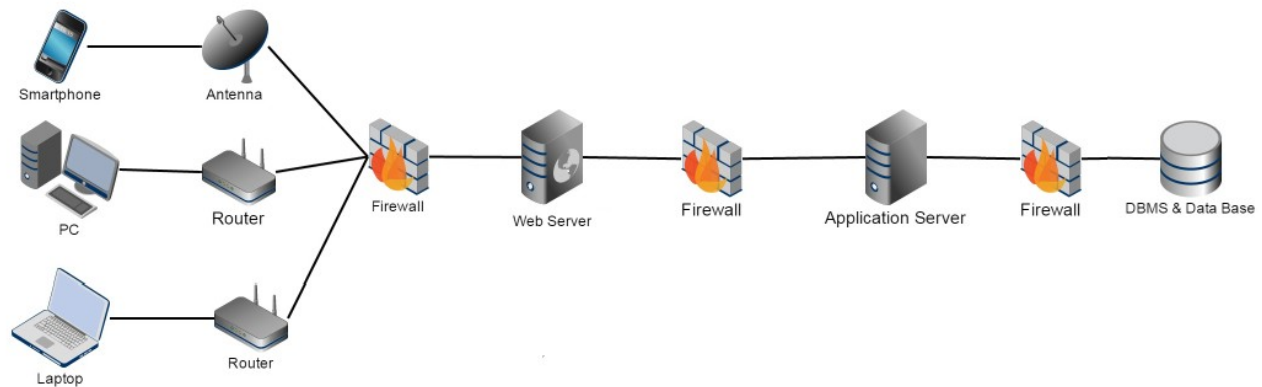
##### **3.7.2.5.2 Application Side**

The application side implements a filtering system to avoid external attack from malicious users that would kept inaccessible information. So https protocol and SSL protocol must be implemented to guarantee intact and secure communications.

##### **3.7.2.5.3 Server Side**

The server side architecture could be implemented dividing strongly the data from application. An idea of possible secure infrastructure

is well represented by the following picture. Application Server is separated from database and from the web server. All zone are divided by firewall creating two different demilitarized zone (DMZ), so the access to this zone is restricted and forbidden to not authorized user.



## 4 Appendix

### 4.1 Alloy

The following alloy model presented is created using the class diagram. We pasted below alloy code edited with Alloy tool.

```

/*ENTITY*/
open util/boolean
/*Declaration of Date*/
sig Date{
    day : one Int,
    month : one Int,
    year : one Int
}

/*Declaration of Time*/
sig Time{
    minute : one Int,
    hour : one Int
}
  
```

```

sig Name{}
sig Surname{}
sig Email{}
sig Phonenumbr{}
sig Username{}
sig Password{}

/* Declaration of RegisteredUser entity*/
sig RegisteredUser{
    name : one Name,
    surname : one Surname,
    email : one Email,
    birthDate : one Date,
    phoneNumber : one Phonenumbr,
    username : one Username,
    password : one Password
}
fact uniqueRegisteredUser{
    no disj r1,r2 : RegisteredUser | r1.username = r2.username
    no disj r1,r2 : RegisteredUser | r1.email = r2.email
    no disj r1,r2 : RegisteredUser | r1.phoneNumber = r2.phoneNumber
}

/*Declaration of Customer entity*/
sig Address{}
sig Customer extends RegisteredUser{
    address : one Address
}

/*Declaration of TaxiDriver entity*/
sig TaxiCode{}
sig TaxiNumberPlate{}
sig TaxiDriver extends RegisteredUser{
    taxiCode : one TaxiCode,
    taxiNumberPlate : one TaxiNumberPlate,
    available : Bool,
    busy : Bool
}
{

```

```
    (available = True implies busy = False) and (available = False implies busy
= True)
}
```

```
fact uniqueTaxiDriver{
    no disj d1,d2 : TaxiDriver | d1.taxiCode = d2.taxiCode
    no disj d1,d2 : TaxiDriver | d1.taxiNumberPlate = d2.taxiNumberPlate
}
```

```
/*declaration of enum TaxiModel*/
abstract sig TaxiModel{}
one sig normal extends TaxiModel{}
one sig large extends TaxiModel{}
```

```
/*Declaration of Request entity*/
sig StartingPosition{}
sig Result{}
sig Request{
    startingPosition : one StartingPosition,
    taxiModel : one TaxiModel,
    result : one Result,
    refersTo : one Customer,
    satisfiedFrom : lone TaxiDriver
}
```

```
/*Declaration of Reservation entity*/
sig Destination{}
sig Reservation extends Request{
    destination : one Destination,
    date : one Date,
    meetingTime : one Time
}
```

```
/*Declaration of Queue entity*/
sig Id{}
sig Queue{
    id: one Id,
    numberOfDrivers: one Int,
    firstDriver: one TaxiDriver,
    lastDriver: one TaxiDriver,
    hasDrivers : some TaxiDriver,
```

```

    hasZone : one CityZone
}

{
    hasDrivers.available = True
    (numberOfDrivers = 1) implies (firstDriver = lastDriver)
    (numberOfDrivers > 1) implies (firstDriver != lastDriver)
}
fact queueConstraint{
    no t : TaxiDriver | all disj q1 , q2 : Queue | t in q1.hasDrivers and t in
q2.hasDrivers
}
fact uniqueCityZone{
    no q1 : Queue | all disj z1,z2 : CityZone | q1 in z1.hasQueue and q1 in
z2.hasQueue
}

/*Declaration of CityZone entity*/
sig CityZone{
    number : one Int,
    dimension : one Int,
    hasQueue : one Queue,
    hasAdjancences : some CityZone
}
fact uniqueCityZone{
    no disj z1,z2 : CityZone | z1.number = z2.number
}
fact uniqueQueue{
    no z : CityZone | all disj q1,q2 : Queue | z in q1.hasZone and z in
q2.hasZone
}

/*Predicate and assertions*/
pred checkCorrectRequest{
    all c : Customer | no r : Request | !(c in Customer) and c in r.refersTo
}
run checkCorrectRequest

pred checkCorrectReservation{
    all c : Customer | no r : Reservation | !(c in Customer) and c in r.refersTo
}

```

```
}  
run checkCorrectReservation
```

```
pred addTaxiDriver[q1,q2 : Queue, t : TaxiDriver]{  
    q2.hasDrivers = q1.hasDrivers + t  
}  
pred addTaxiDriverInQueue () {  
    all q1,q2: Queue, t:TaxiDriver| t not in q1.hasDrivers and addTaxiDriver  
[q1,q2,t] implies t in q2.hasDrivers  
    and q2.numberOfDrivers = q1.numberOfDrivers + 1  
}  
run addTaxiDriverInQueue for 3
```

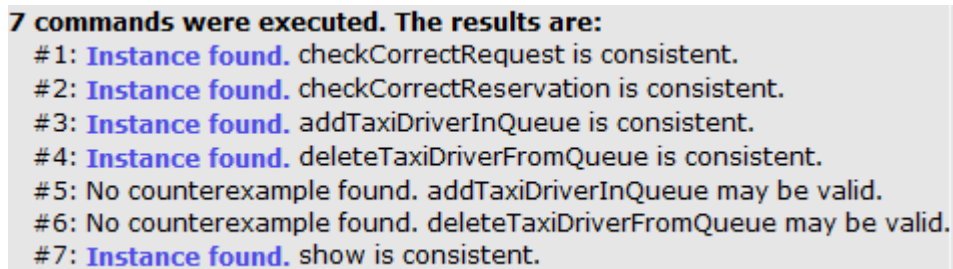
```
pred deleteTaxiDriver[q1,q2 : Queue, t : TaxiDriver]{  
    q2.hasDrivers = q1.hasDrivers - t  
}  
pred deleteTaxiDriverFromQueue () {  
    all q1,q2: Queue, t:TaxiDriver| t in q1.hasDrivers and deleteTaxiDriver  
[q1,q2,t] implies t not in q2.hasDrivers  
    and q2.numberOfDrivers = q1.numberOfDrivers - 1  
}  
run deleteTaxiDriverFromQueue for 4
```

```
assert addTaxiDriverInQueue{  
all t: TaxiDriver, q1, q2: Queue | (t not in q1.hasDrivers) and addTaxiDriver[q1,q2,  
t] implies (t in q2.hasDrivers)  
}  
check addTaxiDriverInQueue
```

```
assert deleteTaxiDriverFromQueue{  
all t: TaxiDriver, q1, q2: Queue | (t in q1.hasDrivers) and deleteTaxiDriver[q1,q2,t]  
implies (t not in q2.hasDrivers)  
}  
check deleteTaxiDriverFromQueue
```

```
pred show {  
    #RegisteredUser>2  
    #Customer > 1  
    #TaxiDriver > 1  
    #Request > 1  
    #Reservation > 1  
}  
run show for 4
```

This screenshot of the Alloy Analyzer software shows the consistency of the model.



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

- #1: **Instance found.** checkCorrectRequest is consistent.
- #2: **Instance found.** checkCorrectReservation is consistent.
- #3: **Instance found.** addTaxiDriverInQueue is consistent.
- #4: **Instance found.** deleteTaxiDriverFromQueue is consistent.
- #5: No counterexample found. addTaxiDriverInQueue may be valid.
- #6: No counterexample found. deleteTaxiDriverFromQueue may be valid.
- #7: **Instance found.** show is consistent.

## 5 *Revision*

This is 2.0 version of RASD that contains update of the document. In this version we have just deleted the goal number 5 and its relative functional requirement.