Politecnico di Milano

A.A. 2015-2016
Software Engineering 2: “myTaxiService”
Design **D**ocument

Morenzetti Sergio 791558.
Marcucci Antonio Pio 854473.

1. Introduction

1.A Purpose

The purpose of this document is to provide documentation for the design and implementation of myTaxiService application.
In the Design Document (DD) we focus on an high level description concerning design and architecture aspects of our myTaxiService application.
We will identify the modules in which the system is decomposed, what are their interfaces, useful relations and the interaction among them.
We want to find out the main design principles and the criteria to organize the system, giving a documentation about this, and also to ensure continuity for future project development (high level description of logical components and how they interact).

1.B Scope

Once that the DD document is over, it will provide a sort of guide for the implementation phase.
In fact the architectural choices that will be described in this document will affect the request and reservation process as well as the queue management process.

1.C Definitions and Acronyms

Definitions:

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

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, 2km^2.

Acronyms :

RASD: Requirements Analysis and Specification Document.
DD:Design Document.
DB: DataBase.
DBMS: DataBase management system.

OS: Operating System.
JVM: Java Virtual Machine.
DMZ: Demilitarized zone.

1.D Reference Documents
Template for the Design Document (DD TOC.pdf)

1.E Document structure
This document is essentially structured in five parts:

Section 1: Introduction
It gives a description of document and some other general informations.

Section 2: Architectural design
It explain the main design choices for our application and give architectural description of the system in its parts.

Section 3: Algorithm design
It focuses on the definition of the most relevant algorithms of our project.

Section 4: User interface design
It provides an overview on how the user interface(s) of our system will look like.

Section 5 : Requirements traceability
It explains how our application requirements map into the design elements.

2 Architectural design

2.A Overview

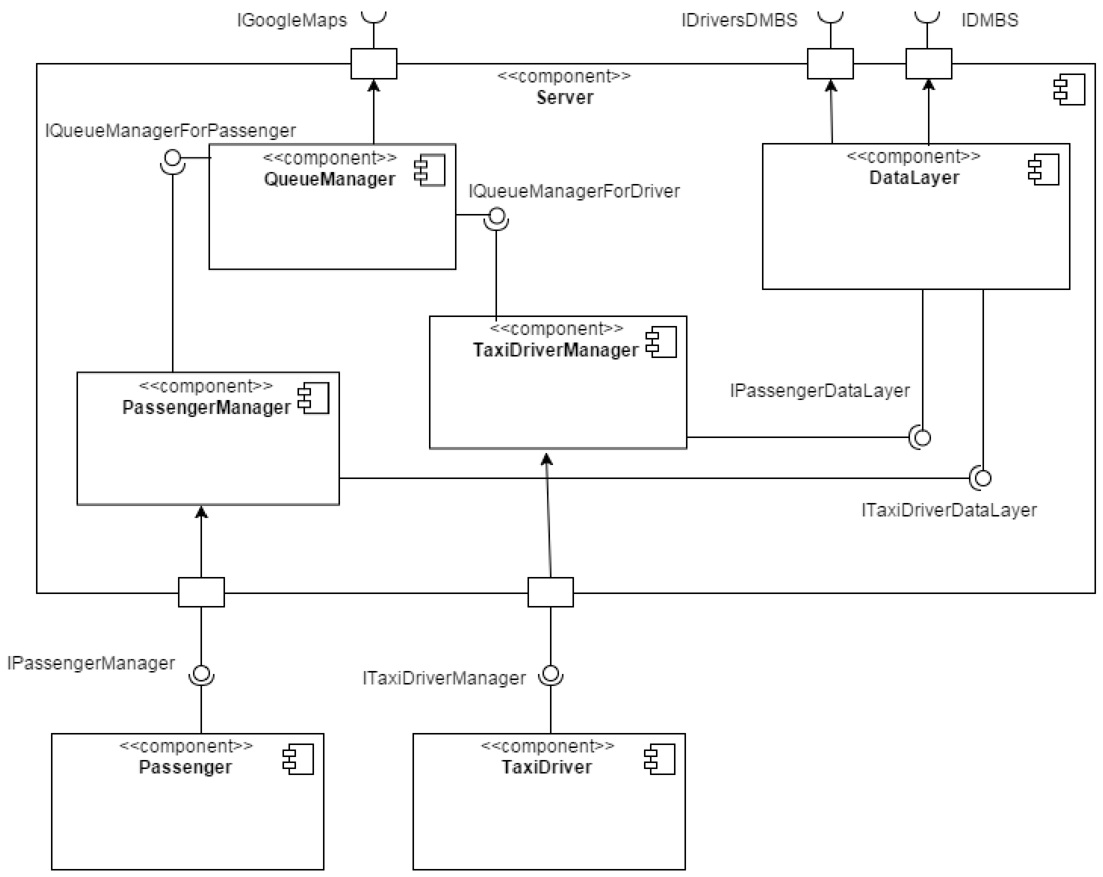
Since we identified as actors involved in myTaxiService application both taxi drivers and customers, these will be seen as clients from our application requesting services to the system (identified as the server), which will provide them a response.
For this reason we have chosen a client-server architecture, with thin client, divided in four tier :
- GUI : the clients are in charge of all presentation issues;

- Web Server: it handles https protocol; when it receives an https request it responds sending back an html page;
- Application Server: it implements the business logic of the application;
- Data Base : it stores all the information about customers, taxi drivers and reservation data.

We divided the server in two different tiers in order to distribute the workload , decrease the amount of the tasks that the components have to execute and thus improve the interaction between the GUI and the application tier.

2.B High level components and their interactions

In this section we will describe the main components of our system.
The Passenger and the Taxi Driver are the two possible client of our application, also identified as component of the system.
The Passenger is able to enjoy the services offered by the application, and this possibility is given by the Passenger Manager, which is a component of the server.
This component handles all the functionalities that myTaxiService offers to the customers (the same considerations can be done also for the Taxi Driver- Taxi Driver Manager components); it interacts with the Data Layer and the Queue Manager component.
The Queue Manager deals with the administration of the system queues;
for this reason it has to interact both with the Passenger and Taxi Driver Manager component.
The Data Layer is the one which is in charge of storing and retrieving data into DB (which is outside the server).
As already mentioned in the RASD, our system will be interfaced with Google Maps, which will provide to the application all the geographic data about our city; it will be interfaced also with an external DBMS containing an “official” list of the already existing taxi drivers of the city, and finally with the DBMS of our application in which we store users informations.
Since these interfaces are connected to component that we have considered as external to our application implementation, they will not be analyzed anymore.



2.C Component view

In this section we describe the components identified in the previously, and we will find out also the modules that characterize them.

- Queue Manager:

  In this component we have identified two main modules, the first one related to the city zones, the second one refers to the association between these zones with the taxis queues and this means also the administration of the taxi drivers in the queues.
  The two modules are related each other because there is a queue for each city zone; for this reason we thought that an hashmap structure should be a good choice, using the zones as keys, and list of taxi drivers as values.
  Furthermore this component will retrieve geographic informations from GoogleMaps.

- Passenger:

  It's just a single module that describes all the attributes related to the customer of our application, like the name, surname etc..

- Taxi Driver:

  Like the passenger module, it is only one, and contains all the driver details.

- Passenger Manager:

  It has two main modules: one is about the first interaction that a client has to perform with the server in order to use application functionalities (login, or registration if it’s the first time) or the account administration;
  The other module is about the services offered by the application (like request or reserve a taxi, edit a reservation..)

- Taxi Driver Manager:

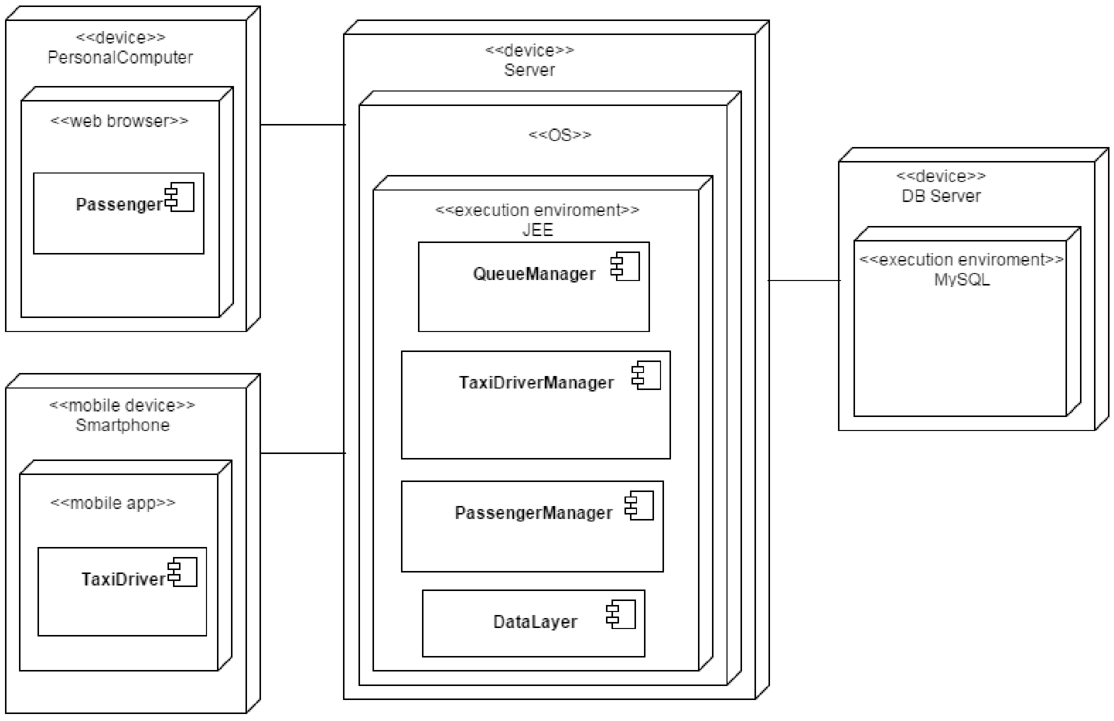
  Again the same situation explained for the Passenger Manager component, that is a module about first interactions, and another one about services exploitation.

- Data Layer:

  This component is characterized by just one module that interacts with the application DBMS in order to store all the user informations, and also with an external DB with the aim of checking whether the registering drivers are in a sense “official” .

2.D Deployment view

The deployment diagram shows how components identified in the previous section are mapped in the hardware structure of our system.
It’s characterized by cubes which represents nodes.
A node is either an hardware device or some software execution environment, and these nodes could be connected through communication paths to create an arbitrary networked system.



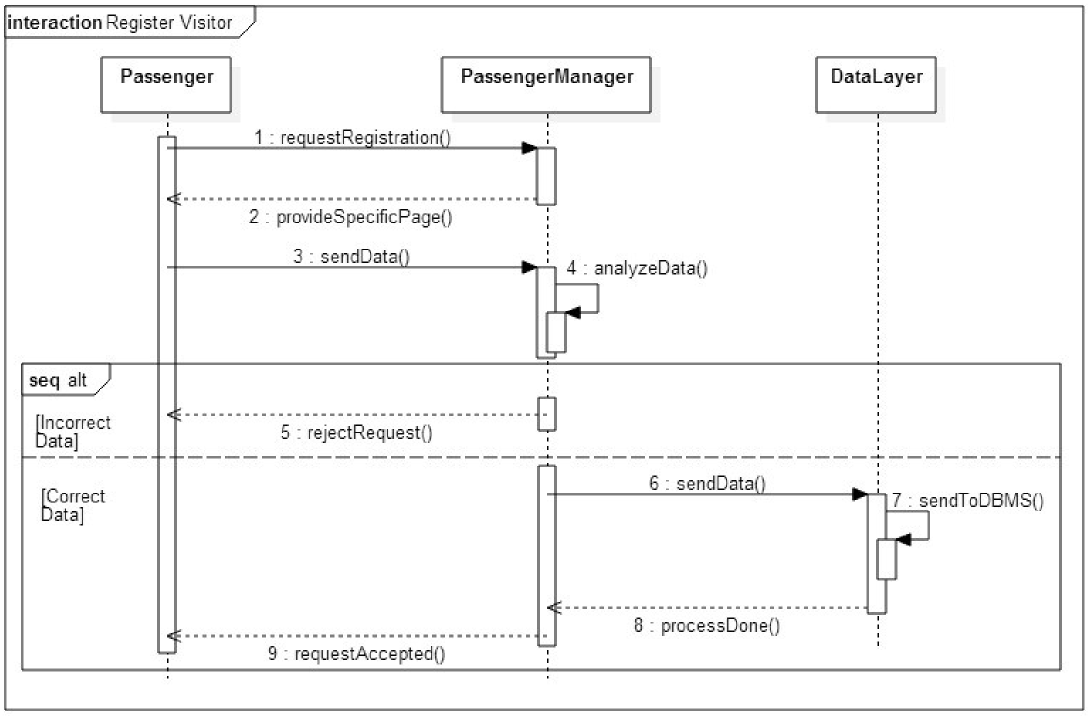
The passenger component should be mapped in the software environment of either a web browser (if the customer uses the myTaxiService from a personal computer device) or a mobile app environment (if he uses the application of his smartphone device) because this component concerns all the aspects referred to the customer side.
Since the taxi drivers can use only the application through their mobile phone, we just represented the mobile device and the mobile application environment.
All the components that represent all the logic aspects of the application, will be mapped in a Java Enterprise Edition environment, that should be implemented in every possible operative system of a dedicated server device.
We also identified the Database server with a MySQL execution environment.

2.E Runtime view

In this section we will to show how the components, identified in this project, interact to accomplish specific tasks related to our use cases.
The sequence diagram below, are referred to the main tasks of the

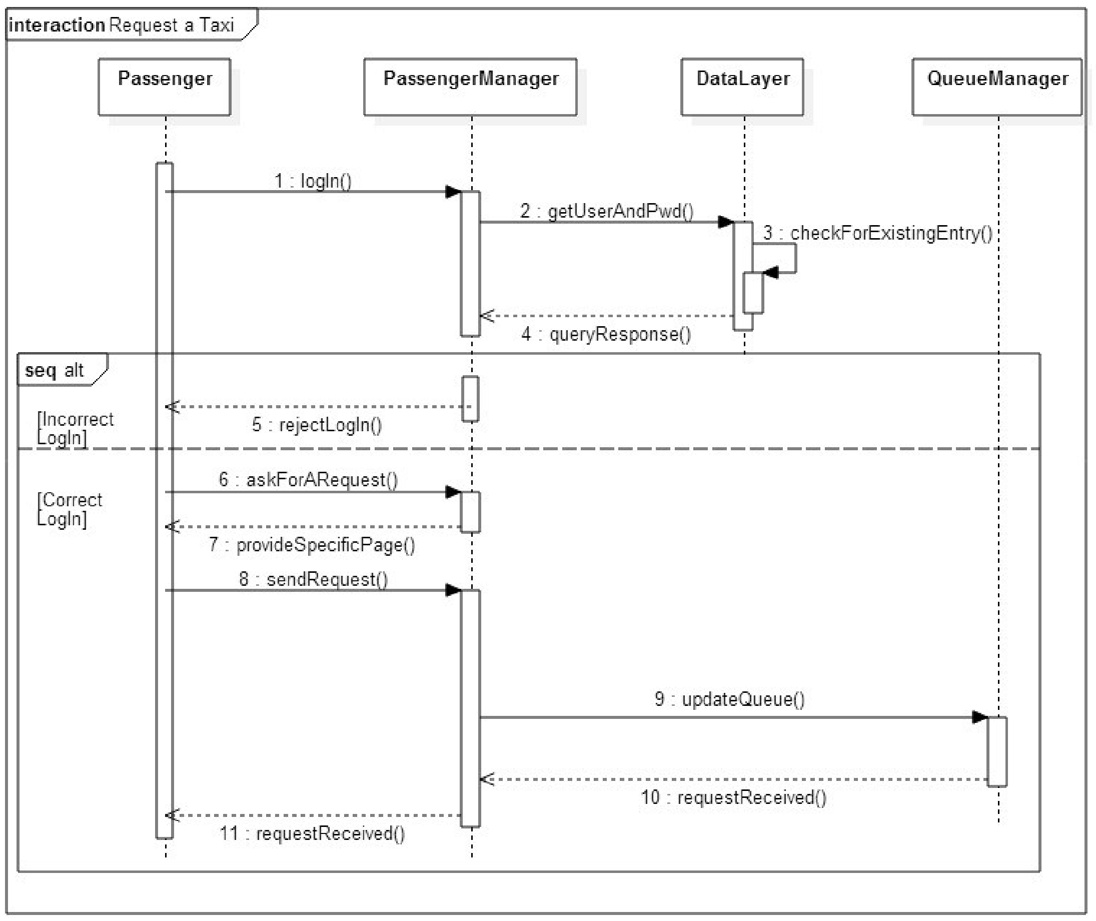
application.

- “Register Visitor” use case



The components involved to accomplish this task are:
- Passenger, which starts the registration process;
- Passenger Manager, that has the task to check incoming visitor data and answer to registration request through an email confirmation;
- DataLayer, that in this case has the functionality of communicating with DBMS sending data to store in the db related to a new passenger account.

- “Request a Taxi” use case

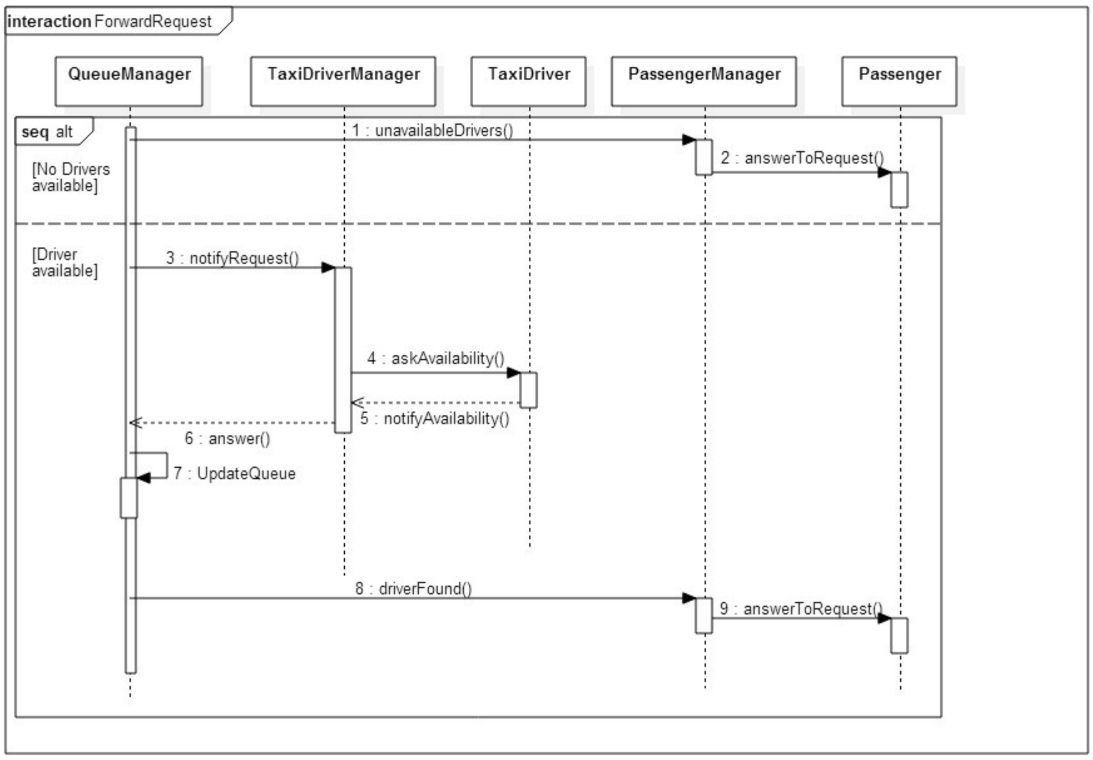


To accomplish this functionality, the components involved are:

- Passenger, which starts the process performing firstly a login action and then, in case of correct log in, asking for a request.
- Passenger Manager, that has the task to check log in data and answer in both positive and negative cases the answer to his request; furthermore, in case of correct log in, it has to provide a specific page to permit to passenger component

possibility to perform a request; after receiving the request, it has also to forward these data to the QueueManager component;
- DataLayer, that in this case has the task of sending a query to dbms to check if the instance of the incoming user is already existing into the db;
- QueueManager, that receives the data of the incoming request.

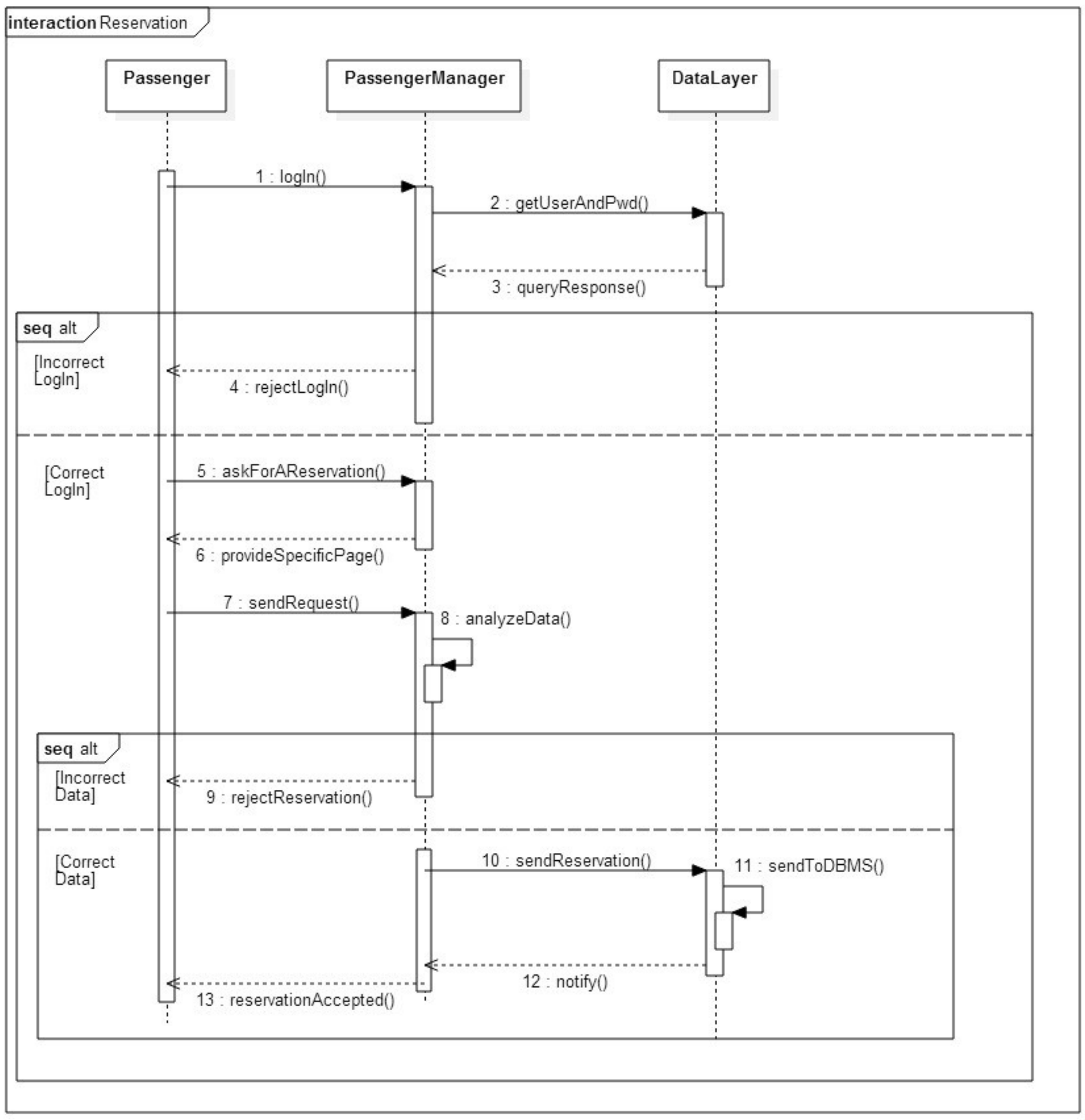
- "Forward Request" use case



This functionality starts after the queue manager has analyzed the queue availability, indeed we analyze the case in which the involved queue is empty and also there are no drivers available for the "nearest queue" and the case in which the involved queue is not empty.
So components involved in this task are:

- QueueManager, that after having analyzed the queue availability has the task of notifying the PassengerManager about the drivers availability in both positive and negative cases; but only in the positive one QueueManager has also to update the involved queue;
- TaxiDriverManager, which is the component with the task of asking for drivers availability, once analyzed the request; moreover, after receiving a response,it has to notify the QueueManager about the answer in order to permit the update of the queue;
- TaxiDriver, which has to provide a response to the TaxiDriverManager;
- PassengerManager, that has to send the answer of the request, to the Passenger component;
- Passenger, that in this case receives the response of his request.

• "Reserve a Taxi" use case



In order to accomplish this functionality, the components involved are :

- Passenger, that starts the process performing firstly login action and then, in case of correct log in, asking for a reservation;
- PassengerManager, has to check incoming log in data, interacting with data layer component, and give an answer to passenger component in both positive and negative cases; , it has to provide a specific page to permit to passenger component possibility to perform a reservation; if data are incorrect it has to notify this to passenger component;
- DataLayer, has two different task in this case, firstly it has to query dbms to check if the incoming log data are already in the db; secondly it has to send reservation data, that will be

stored into the db .

2.F Component interfaces

The interfaces that we identified in the component diagram will be the connectors between the different components of our system; it has been represented like a combination of “0)” where the “0” means that the component provides a service, the “)” instead stays for a required service.

- IPassengerManager: this interface represents the main services

  provided by myTaxiService application (from Passenger Manager component) to the customers (passenger component); in fact is through this a user is able to make request, reservations, registration and so on.
  It will contain a list of methods that will connect the passenger credentials of to his virtual representation in our system.
  In other words the interface is the node through the passenger flow of information passes by.

- ITaxiDriverManager: the same considerations that we made for

  the previous interfaces should be done for this one, just substituting the request, reservations services with those that are related to the taxi driver (like accepting/refusing a request..etc).

- IQueueManagerForPassenger: this interface provides to the

  passenger manager component the possibility of associating a taxi driver, taken from a queue, to a passenger, where is it possible .
  This is reason why the Queue Manager is the component which is providing a service, and not vice versa.

- IQueueManagerForTaxiDriver: thanks interaction with the Queue

  Manager component, the interface gives to the taxi driver manager the possibility of adding/deleting a particular taxi from its associated queue; it depends on the driver actions.

- IPassengerDataLayer: it connects DataLayer component to

  Passenger Manager, and through this interface all the passenger account information along with his reservations will pass by;

- ITaxiDriverDataLayer: this interface will serve as a bridge for all the account informations of the taxi drivers, from the Manager to the DataLayer component.

2.G Selected architectural styles and patterns

As already mentioned, we adopted a client-server architecture, characterized by a four tier structure.
The reason of this choice is that, apart from the client and the database, we divided the server in two components: the Web Server and the Application Server.
This separation has the scope of the workload distribution .

About the patterns we focused on the MVC (Model-View-Controller) architectural schema.
MVC separates the application’s data model, user interface, and control logic into three distinct components so that the modifications to one component can be made with minimal impact to the others.
The Model contains the logic of the application, its core, that can be only modified by the view through the supervision of the controller; when something changes in the application, the model has the possibility of notifying the view.
The Controller responds to events, typically user actions, and invokes these “changes” on the Model (but not the view).
The View renders the model into a form suitable for interaction, and typically it is a user interface element. In this way the user are able to see changes in the Model and to provide changes, if it is the case.
We chose MVC because the separation that it provides seems to be very efficient for this kind of application.

3 Algorithm design

In this section we will describe few algorithms that characterize our application.
This algorithm uses GoogleMaps data and the city zone division.
It should return an ordered list of all the zones apart from the

incoming request zone.

```
computeCloseZones (Zone z){
    for(i = 0; i < number of zones; i++){
        if z is not equal to the i-zone
            so add the i-zone into a list;
    }
    order the list from the closest to the farest zone from z;
    return the list;
}
```

Request accomplishment:
These algorithms have to interact both with the data of the request and the queues of taxis.

```
requestManagement (Request r)
{
    if it is not true that at least a taxi driver is present in one among
    all the queues of the system
        so return “request unsatisfiable”;
    else
        z = zone associated to the GPS information of r;
        q = queue associated to the zone z;
        if q is empty
            so list = call the algorithm computeCloseZones( z );
            for ( i=0; I < size of the list ; i++ ) {
                if i – zone has a not empty queue associated
                    so z = i – zone;
                    q = queue associated to zone i;
                    break;
        }

        send a message to the first driver of q;
}
```

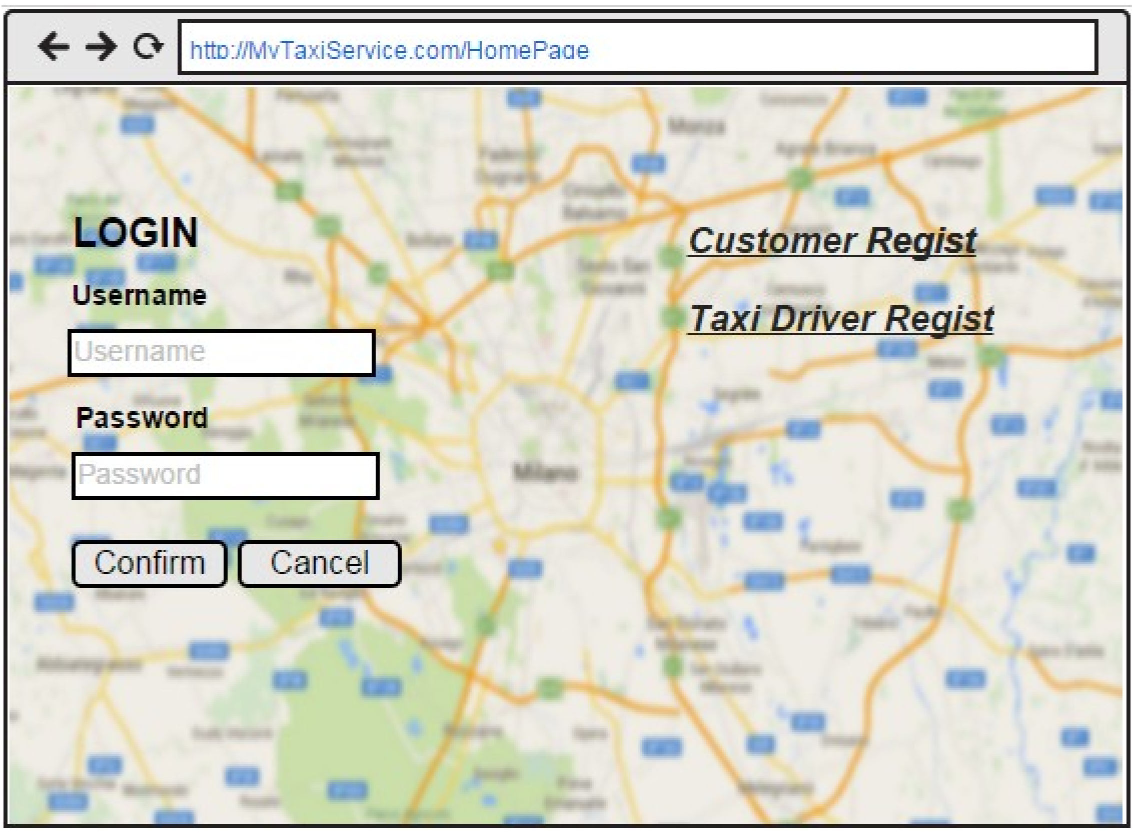
```
requestResponse (DriverAnswer a, Request r)
{
    if a is yes
        so delete the drivers from the associated queue;
    else
        put the driver at the and of the associated queue;
        call algorithm requestManagment(r);
}
```

4 User interface design

This section will provide an overview on how the user interfaces of our application will look like.

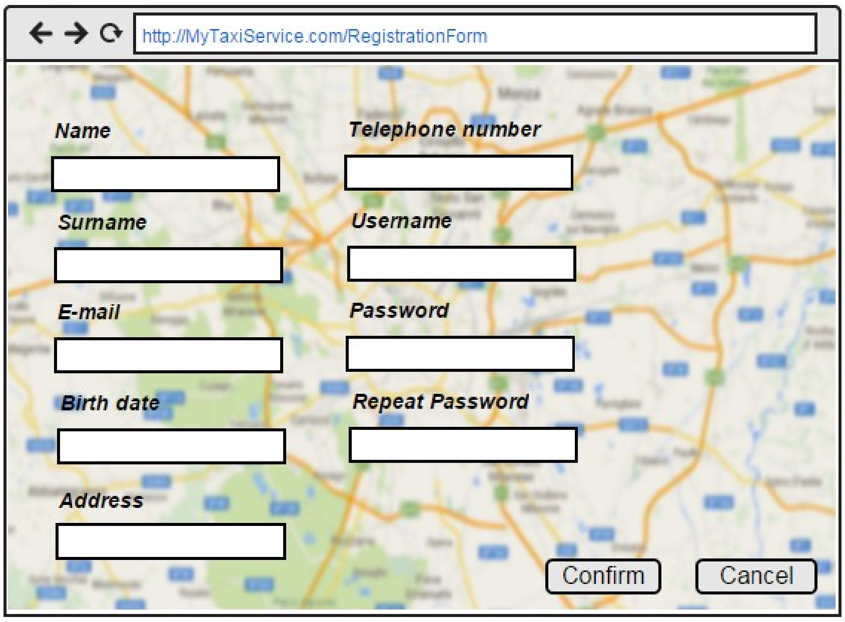
Home Page

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.

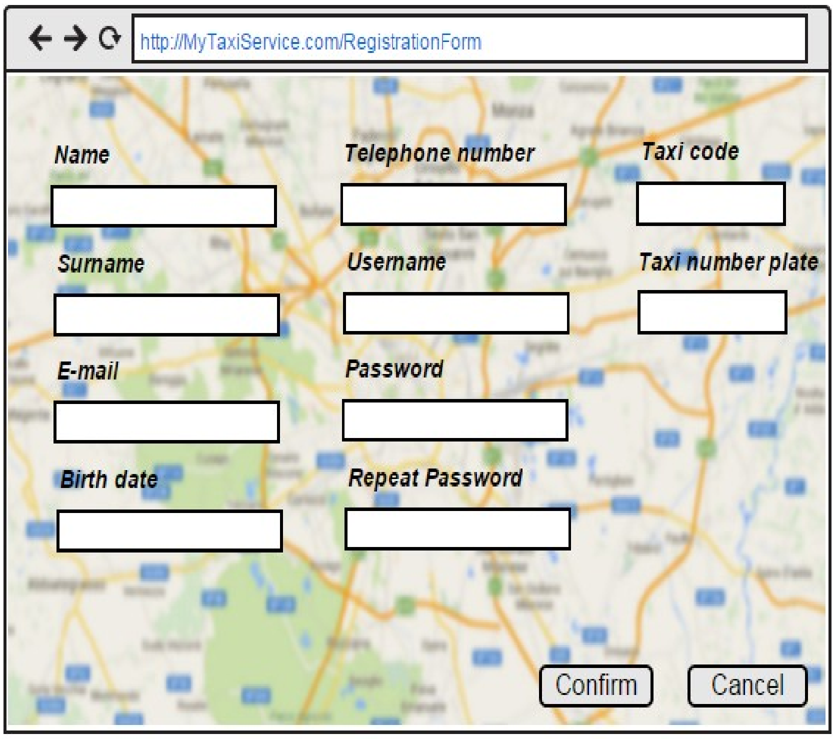


Registration forms

This mock shows the registration form page for customers.



This mock shows the registration form page for taxi drivers.

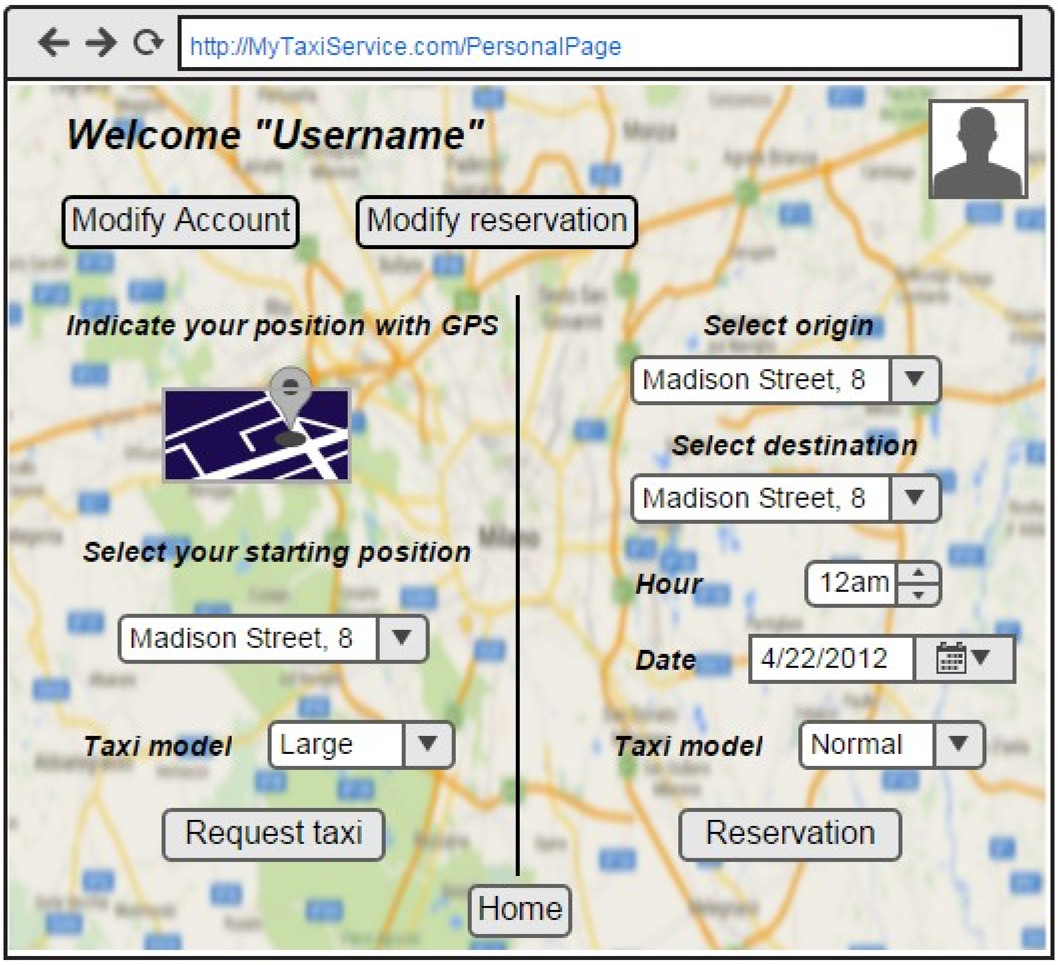


Name	Telephone number	Taxi code
Surname	Username	Taxi number plate
E-mail	Password	
Birth date	Repeat Password	

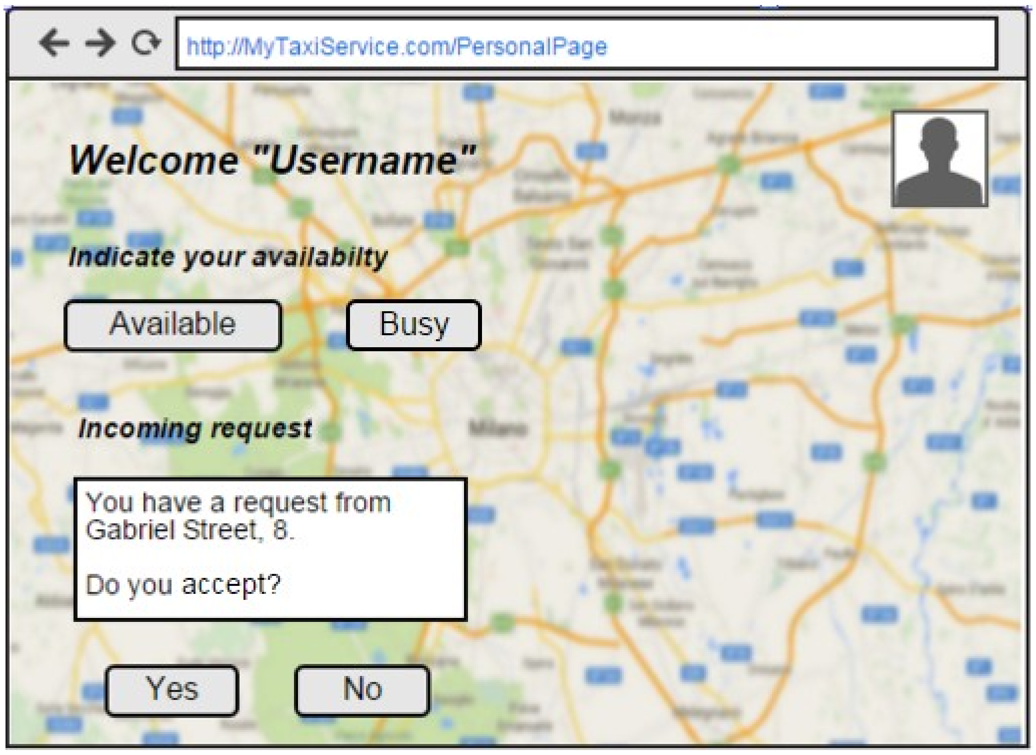
Personal page

This mockup shows the personal page of an hypothetic 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.



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

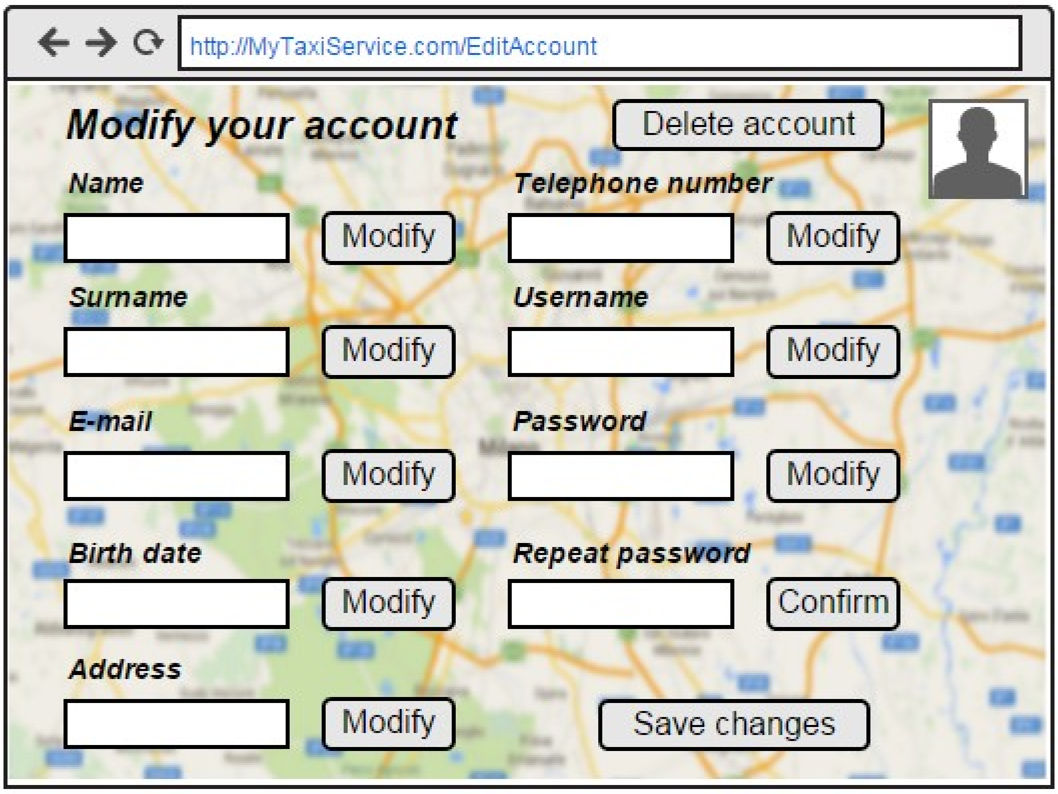

http://MyTaxiService.com/PersonalPage
Welcome "Username"
Indicate your availabilty
Available
Busy
Incoming request
Yes
No

You have a request from Gabriel Street, 8.

Do you accept?

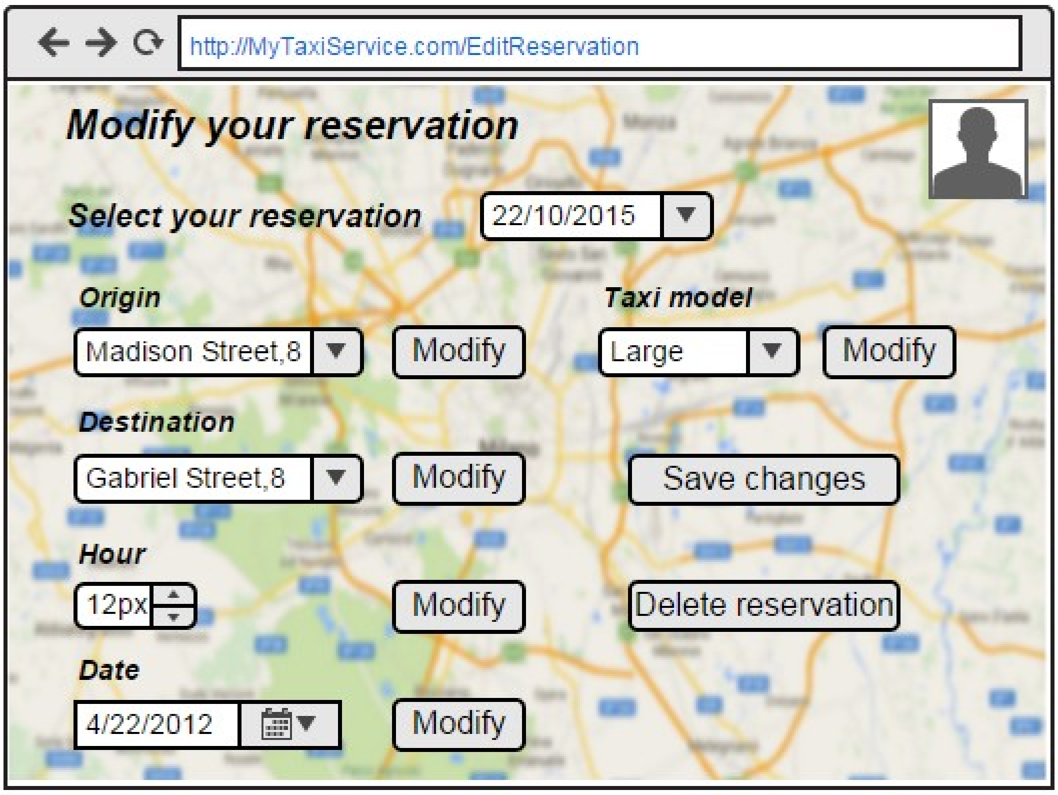
Edit account page

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



Edit reservation page

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



5 Requirements traceability

In this section we will explain how the functional requirements defined in the RASD document map into the components defined in this document;

[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.

FUNCT. REQUIREMENT	COMPONENT	COMMENT
R1	Passenger, PassengerManager,	Passenger : get username and

	DataLayer	password Passenger Manager : check if user and password are correct DataLayer : provide data account previously stored
R2	Passenger, PassengerManager	Passenger : receives email PassengerManager : sends email

[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.

FUNCT. REQUIREMENT	COMPONENT	COMMENT
R1	Passenger, PassengerManager	Passenger : provides log in data PassengerManager : receives log in data
R2	Passenger, PassengerManager	Passenger : provides log in data PassengerManager : checks log in data and provides myTaxiService functionalities
R3	Passenger, PassengerManager	Passenger : receives request answer PassengerManager : sends request answer

[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.

FUNCT. REQUIREMENT	COMPONENT	COMMENT
R1	QueueManager	QueueManager : manages association between city zones and taxi queues
R2	QueueManager, TaxiDriverManager	QueueManager : computes the distribution TaxiDriverManager : provides taxi driver information
R3	TaxiDriver, TaxiDriverManager	TaxiDriver : sends info about availability TaxiDriverManager: receives info about availability
R4	TaxiDriver, TaxiDriverManager	TaxiDriver : specifies the busyness TaxiDriverManager: receives busyness info

[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.

FUNCT. REQUIREMENT	COMPONENT	COMMENT
R1	Passenger	Passenger : provides reservation data
R2	Passenger, PassengerManager	Passenger : provides reservation data Passenger Manager : receives and checks reservation data
R3	Passenger, PassengerManager, DataLayer	Passenger : modifies or deletes an existing reservation PassengerManager : provides the existing reservation and sends modifications or deletion info DataLayer : updates stored reservation data