POLITECNICO
MILANO 1863

MyTaxiService

Design Document

Davide Citterio, Lorenzo Cunial, Massimo Beccari

December 4, 2015

Contents

1 Introduction

1.1 Purpose

The Design Document is a document to provide documentation which will be used to aid in software development by providing the details for how the software should be built. Within the Design Document there are narrative and graphical documentation of the software design for the project including supporting requirement information.

1.2 Scope

The architectural descriptions provided the functional view, module view, deployment view, business logic, user experience and entity-control-boundary models. Their will consider the functionalities specified in the RASD. These and many other informations lead to simplify the view of the system.

1.3 Definitions

Keyword	Definitions
Next stopping point	is the nearest destination/starting address following the route
Driver	is the taxi driver

1.4 Acronyms and Abbreviations

Acronym or abbreviation	Definition
MTS	My Taxi Service
JEE	Java Enterprise Edition
RASD	Relational Database Management System
XHTML	Extensible HyperText Markup Language
JSF	Java Server Faces
EIS	Executive Information System
SQL	Structured Query Language
API	Application Programming Interface
JDBC	Java DataBase Connectivity
ER	Entity Relationship
UX	User Experience
BCE	Boundary-Control-Entity
IDE	Integrated Development Environment
SMTP	Simple Mail Transfer Protocol

1.5 Reference Documents

1. IEE Standard for Information Technology - Systems Design - Software Design Descriptions: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5167255

2. Requirements Analysis and Specification Document: rasd12.pdf

1.6 Document Structure

The document is organized as follows:

- **Section 1,** Introduction, provides a synopsis of the architectural descriptions;
- **Section 2,** Architectural Design, specifies the general architecture, describes the basic structure and interactions of the main subsystems;
- **Section 3,** Algorithm Design, contains the definition of any important algorithm describes the system;
- **Section 4,** User Interface Design, provides an overview on how the actors (user, driver and administrator) interfaces of the system will look like;
- **Section 5,** Requirements Traceability, explains how the requirements defined in the RASD map into the design elements defined in this document;
- **Section 6,** References.

2 Architectural Design

2.1 Overview

Our system has a three-tiered architecture:

- The Client tier;
- The Application Server tier;
- The Database Server Tier.

The Application Server is split into two logical parts:

- The Web Tier;
- The Business Logic Tier.

The following figure shows the architecture overview:

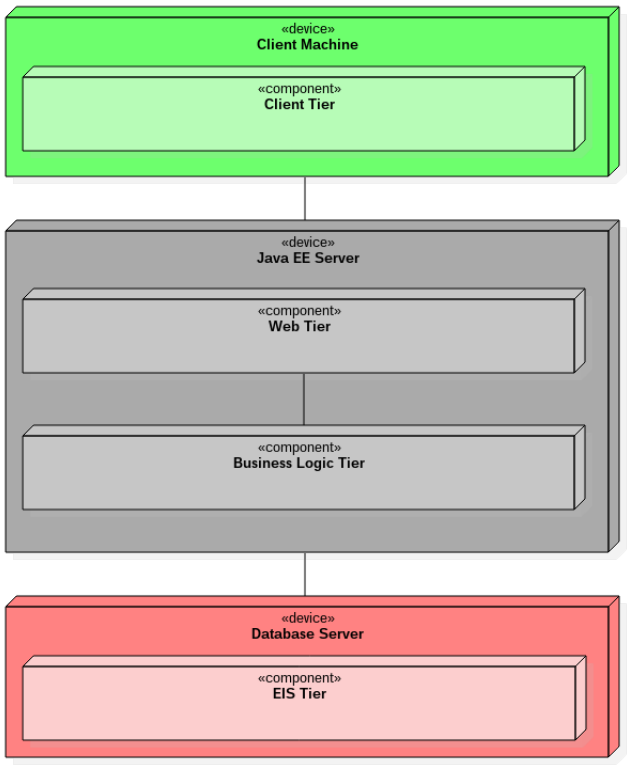


Figure 1: overview

2.2 High level components and their interactions

The main components of our system are:

- User Mobile App;
- Driver Mobile App;
- Web Pages and Managed Beans (Web Tier);
- Enterprise Beans and Entity Beans (Business Logic Tier);
- MySql Database.

The following figure shows those components and their interactions:

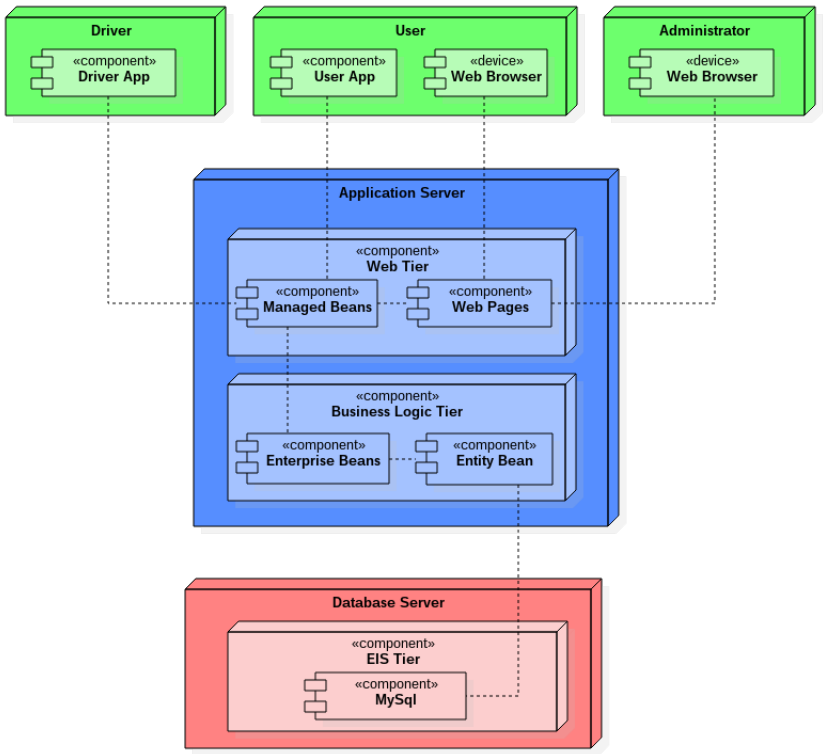


Figure 2: high level components and their interactions

2.2.1 The User

As it is shown in the figure 2, a user can interact with the system through a web browser or a mobile app.
In the first case, the user's web browser send requests for web pages; surfing these pages a user can make requests, reservations and all the operations he is allowed to do.
In the second case, the user can do all the allowed operations directly from the mobile app.
The information exchange between the user's browser, or mobile app, and the server is controlled by the JSF managed beans.
Once an operation request is received, the managed beans send the informations received from the user to the Business Logic's enterprise beans that performs the requested operation. If the operation needs some data, the enterprise beans access to the MySql database to retrieve those informations.

2.2.2 The Driver

The driver can interact with the system only using the mobile app. The interactions between the driver's mobile app and the system happen in the same way explained for the user's mobile app.

2.2.3 The Administrator

The administrator can interact with the system only using the web browser. The interactions between the administrator's web browser and the system happen in the same way explained for a user using the web browser.

2.3 Component view

Here we want to give more detailed informations about the high level components presented in the previous section, indentifying the sub-components of our system.

2.3.1 The Web tier

It is possible to identify three sub-components of the web tier:

- The user's pages and the related managed beans;
- The administrator's pages and the related managed beans;
- The managed beans for the communications with the driver's mobile app.

User's pages and managed beans

In the following figure are shown the user's pages and the related managed beans. These pages and beans are responsible for every feature related to users.
As said in the previous section, a user can interact with the system using a web browser (thin client) or his mobile app (fat client):
the web browser downloads the web pages related to users generated by the corresponding beans;
the mobile app has a built in graphic interface, so it only downloads the needed data (in xml format) directly from the managed beans.

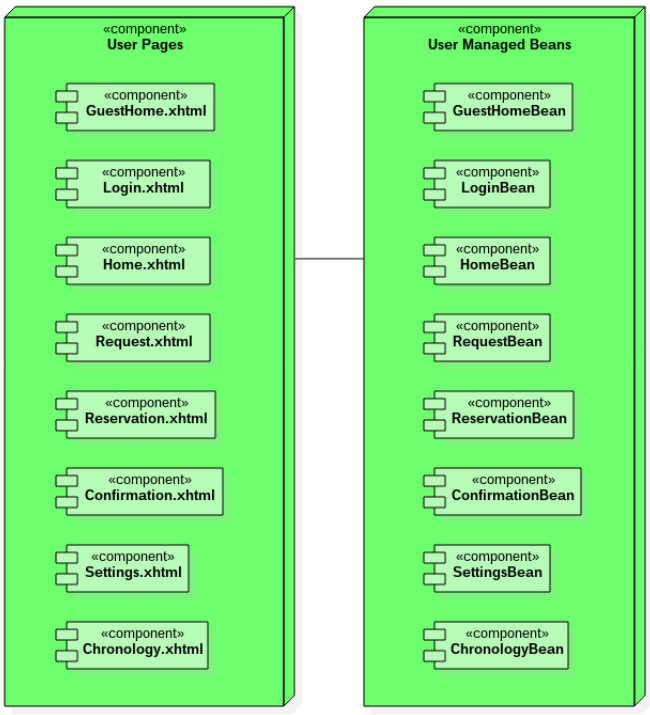


Figure 3: user's pages and related managed beans

Component	
Name	GuestHome.xhtml
Description	The home page for a guest user, with a registration form
Responsibilities	• Display the registration form
Related Bean	GuestHomeBean
Bean Responsibilities	• Load registration form • Check inserted data • Redirect to Home.xhtml or display error message

Component	
Name	Login.xhtml
Description	User interface for user's login
Responsibilities	• Display the login form
Related Bean	LoginBean
Bean Responsibilities	• Load login form • Check inserted data • Redirect to Home.xhtml or display error message

Component	
Name	Home.xhtml
Description	The home page for an authenticated user, with the links to all the allowed operation for a user
Responsibilities	• Display the buttons (links) to all the possible user operations • Display the settings and logout links
Related Bean	HomeBean
Bean Responsibilities	Case 1: the user clicks on an operation's button • Redirect to the selected operation page Case 2: the user clicks on the Settings link • Redirect to Settings.xhtml Case 3: the user clicks on the Logout link • Log out the user

Component	
Name	Request.xhtml
Description	The request page
Responsibilities	• Display the request form • Display the settings and logout links
Related Bean	RequestBean
Bean Responsibilities	In all cases: • Load request form Case 1: the user clicks on the Done button • Check inserted data • Redirect to Confirmation.xhtml or display error message Case 2: the user clicks on the Cancel button • Redirect to Home.xhtml Case 3: the user clicks on the Settings link • Redirect to Settings.xhtml Case 4: the user clicks on the Logout link • Log out the user

Component	
Name	Reservation.xhtml
Description	The reservation page
Responsibilities	• Display the reservation form • Display the settings and logout links
Related Bean	ReservationBean
Bean Responsibilities	In all cases: • Load reservation form Case 1: the user clicks on the Done button • Check inserted data • Redirect to Confirmation.xhtml or display error message Case 2: the user clicks on the Cancel button • Redirect to Home.xhtml Case 3: the user clicks on the Settings link • Redirect to Settings.xhtml Case 4: the user clicks on the Logout link • Log out the user

Component	
Name	Confirmation.xhtml
Description	The confirmation page
Responsibilities	• Display the confirmation page • Display the settings and logout links
Related Bean	ReservationBean
Bean Responsibilities	Case 1: the user clicks on the Got it button • Check inserted data • Redirect to Home.xhtml Case 2: the user clicks on the Settings link • Redirect to Settings.xhtml Case 3: the user clicks on the Logout link • Log out the user

Component	
Name	Settings.xhtml
Description	The user's settings page
Responsibilities	• Display the settings form with the current user's informations • Display the settings and logout links
Related Bean	SettingsBean
Bean Responsibilities	In all cases: • Load settings form with the current user's informations Case 1: the user clicks on the Done button • Check inserted data • Display success message and redirect to Home.xhtml, or display error message Case 2: the user clicks on the Cancel button • Redirect to Home.xhtml Case 3: the user clicks on the Settings link • Reload Settings.xhtml Case 4: the user clicks on the Logout link • Log out the user

Component	
Name	Chronology.xhtml
Description	The user's chronology page, in which he can modify the pending reservations and see all his past operations
Responsibilities	• Display the user's pending reservations • Display the user's past operations • Display the settings and logout links
Related Bean	ChronologyBean
Bean Responsibilities	In all cases: • Load the user's pending reservations, with their informations • Load the user's past operations, with their informations Case 1: the user clicks on a pending reservation: • Redirect to Reservation.xhtml Case 2: the user clicks on the Settings link • Reload Settings.xhtml Case 3: the user clicks on the Logout link • Log out the user

Administrator's pages and managed beans

In the following figure are shown the administrator's pages and the related managed beans. These pages and beans are responsible for every feature related to administrators.

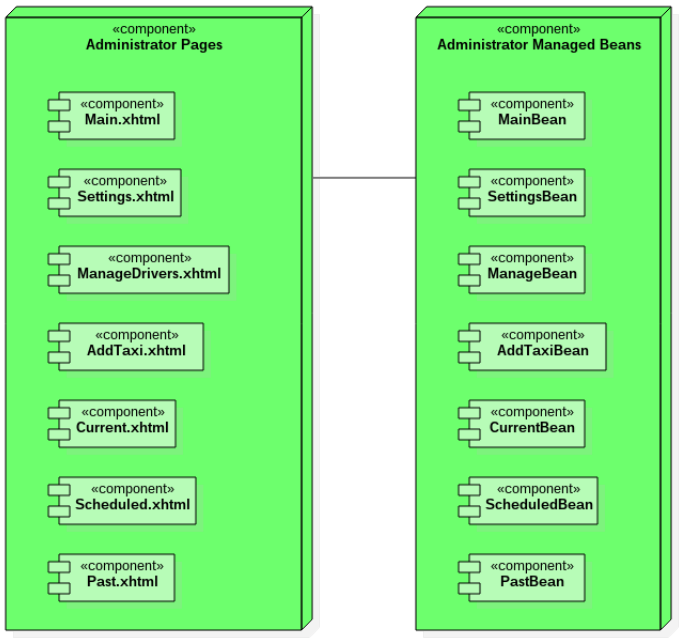


Figure 4: driver's pages and related managed beans

Component	
Name	Main.xhtml
Description	The main page for an administrator, with a map in which the position of all taxis is shown and a list that contains all the taxis and their status, ordered by zone
Responsibilities	• Display the map • Display the taxi list, with buttons (links) to the options page of a taxi • Display the Current Drives, Scheduled and Past links • Display the settings and logout links
Related Bean	MainBean
Bean Responsibilities	In all cases: • Load the map • Load the taxi list Case 1: the administrator clicks on the Options button of a driver • Redirect to ManageDrivers.xhtml Case 2: the administrator clicks on the Current Course link • Redirect to Current.xhtml Case 3: the administrator clicks on the Scheduled link • Redirect to Scheduled.xhtml Case 4: the administrator clicks on the Past link • Redirect to Past.xhtml Case 5: the administrator clicks on the Settings link • Redirect to Settings.xhtml Case 6: the administrator clicks on the Logout link • Log out the administrator

Component	
Name	Settings.xhtml
Description	The page in which an administrator can modify settings like fees and enable/disable additional services like the sharing mode
Responsibilities	• Display the settings form with the current global settings • Display the settings and logout links
Related Bean	SettingsBean
Bean Responsibilities	In all cases: • Load settings form with the current global settings Case 1: the administrator clicks on the Done button • Check inserted data • Display success message and redirect to Main.xhtml, or display error message Case 2: the administrator clicks on the Cancel button • Redirect to Main.xhtml Case 3: the administrator clicks on the Settings link • Reload Settings.xhtml Case 4: the administrator clicks on the Logout link • Log out the administrator

Component	
Name	ManageDrivers.xhtml
Description	The page in which an administrator can add or delete drivers
Responsibilities	• Display the list of all taxi drivers
Related Bean	ManageBean
Bean Responsibilities	In all cases: • Load the list of all the current taxi drivers Case 1: the administrator clicks on the Add New button • Redirect to AddDriver.xhtml Case 2: the administrator clicks on the Delete button • Display success message and redirect to Main.xhtml, or display error message Case 3: the administrator clicks on the Settings link • Redirect to Settings.xhtml Case 4: the administrator clicks on the Logout link • Log out the administrator

Component	
Name	AddTaxi.xhtml
Description	The adding driver page
Responsibilities	• Display the driver registration form
Related Bean	AddTaxiBean
Bean Responsibilities	In all cases: • Load the driver registration form Case 1: the administrator clicks on the Done button • Check inserted data • Display success message and redirect to ManageDrivers.xhtml, or display error message Case 2: the administrator clicks on the Cancel button • Redirect to Main.xhtml Case 3: the administrator clicks on the Settings link • Redirect to Settings.xhtml Case 4: the administrator clicks on the Logout link • Log out the administrator

Component	
Name	Current.xhtml
Description	The current drives page
Responsibilities	• Display the list of the current drives
Related Bean	CurrentBean
Bean Responsibilities	In all cases: • Load the list of all the current drives Case 1: the administrator clicks on the Delete button • Display success message or error message Case 2: the administrator clicks on the Settings link • Redirect to Settings.xhtml Case 3: the administrator clicks on the Logout link • Log out the administrator

Component	
Name	Scheduled.xhtml
Description	The scheduled drives page
Responsibilities	• Display the list of the scheduled drives
Related Bean	ScheduledBean
Bean Responsibilities	In all cases: • Load the list of all the scheduled drives Case 1: the administrator clicks on the Delete button • Display success message or error message Case 2: the administrator clicks on the Settings link • Redirect to Settings.xhtml Case 3: the administrator clicks on the Logout link • Log out the administrator

Component	
Name	Past.xhtml
Description	The past drives page
Responsibilities	• Display the list of the past drives
Related Bean	PastBean
Bean Responsibilities	In all cases: • Load the list of all the past drives Case 1: the administrator clicks on the Settings link • Redirect to Settings.xhtml Case 2: the administrator clicks on the Logout link • Log out the administrator

Driver's managed beans

In the following figure are shown the managed beans for the communications between the system and the driver's app. These beans are responsible for every feature related to administrators. As for the user's mobile app, the driver's mobile app has a built in graphic interface, so it only downloads the needed data from the corresponding managed beans.

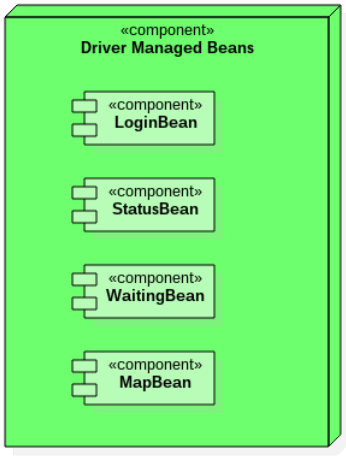


Figure 5: driver's managed beans

Component	
Name	LoginBean
Description	Managed bean for driver's login
Responsibilities	• Check inserted data • Communicate the driver the login outcome

Component	
Name	StatusBean
Description	Managed bean for driver's status
Responsibilities	• Communicate the driver's status If the driver changes status: • Receive the new status

Component	
Name	WaitingBean
Description	Managed bean for calls
Responsibilities	If the driver receives a call: • Communicate the driver the call • Receive the answer from the driver

Component	
Name	MapBean
Description	Managed bean for rides' informations like maps and routes
Responsibilities	Case 1: the driver reports a miss • Receive the missing report Case 2: the driver taps on the start button • Receive the information that the ride has started for a user Case 3: the driver taps on the finish button • Receive the information that the ride has finished for a user • Communicate the fee for the user who has finished the ride • If the user that finished the ride was the last user of that ride, the bean communicates the zone in which the driver has to go

2.3.2 The Business Logic tier

The sub components of the business logic tier are:

- the EJB Authentication Manager (AuthenticationBean);
- the EJB Taxi Queue Manager (TaxiQueueBean);
- the EJB User Operation Manager (UserOperationBean);
- the EJB Shared Rides Manager (SharedRideBean);
- the EJB Payment Manager (PaymentBean);
- the EJB Driver Allocation Manager (DriverAllocationBean);
- the EJB Driver Operation Manager (DriverOperationBean);
- the EJB Administrator Operation Manager (AdminOperationBean);
- the JPA Entity Manager (EntityManagerBean).

Here follows a figure representing the EJB and JPA beans and an explaination of their functionalities.

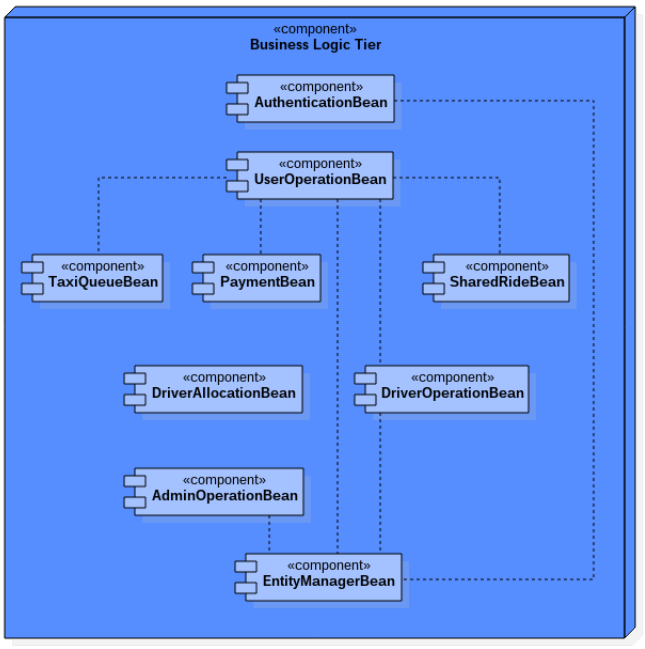


Figure 6: the business logic component beans

Component	
Name	AuthenticationBean
Description	Session bean for the management of users' registration and users', drivers' and administrators' login
Responsibilities	• Register a new user • Login a user/driver/administrator • Logout a user/driver/administrator

Component	
Name	TaxiQueueBean
Description	Session bean for the management of taxi queues in the various zones
Responsibilities	• Insert and extract drivers from the queue

Component	
Name	UserOperationBean
Description	Session bean for the management of user operations
Responsibilities	• Create a user's request • Create a user's reservation • Load the user's operations • Modify the user's pending reservations • Send the user a notification • Load the user's informations • Modify the user's informations

Component	
Name	SharedRideBean
Description	Session bean for the management of shared ride reservations
Responsibilities	• Check the sharing compatibility between the current pending reservations that have the shared option activated

Component	
Name	PaymentBean
Description	Session bean for the calculation of the fee/fees of requests or reservations
Responsibilities	• Calculate the fee for requests and non-shared reservations • Calculate the fee for each passenger of a shared reservation

Component	
Name	DriverAllocationBean
Description	Session bean for the allocation of the drivers in the city zones
Responsibilities	• Calculate the new zone in which a driver must be allocated

Component	
Name	DriverOperationBean
Description	Session bean for the driver operations management
Responsibilities	• Change the driver status • Send the driver a call • Receive an administrator's message

Component	
Name	AdminOperationBean
Description	Session bean for the administrator operations management
Responsibilities	• Load the informations about taxi driver's • Add/delete a taxi driver • Load the informations about rides • Delete pending rides and reservations • Load the settings informations • Modify the settings informations • Enable/disable additional services • Send a message to a driver

Component	
Name	EntityManagerBean
Description	Entity Manager bean for the management of persistent data
Responsibilities	• Create/remove entities • Find/run queries on entities

2.4 Deployment view

The following figure shows the deployment view, with the artifacts that need to be developed.

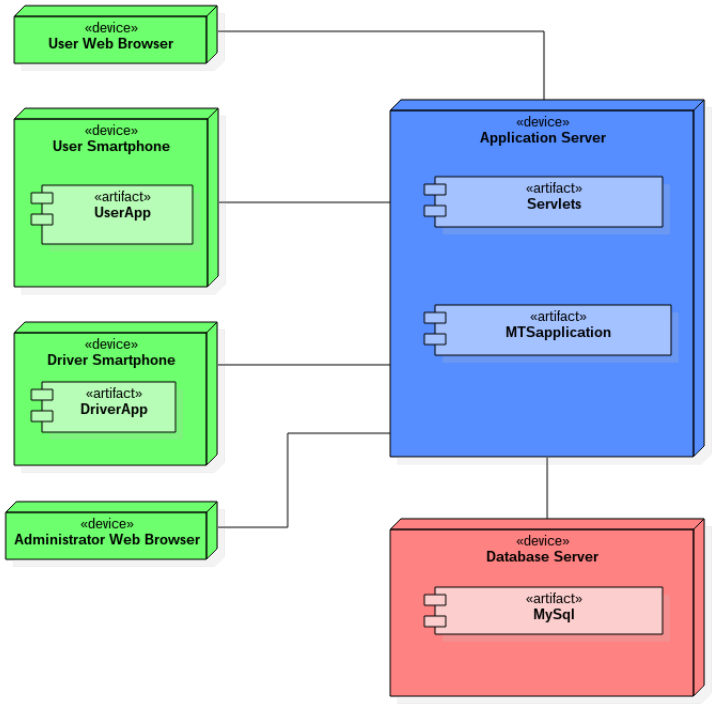


Figure 7: deployment view

2.5 Runtime view

The following figure shows a possible runtime view of the system. It includes:

- a user client making a request;
- a driver client receiving a call;
- an administrator client managing a taxi driver and changing the system setting daily fee.

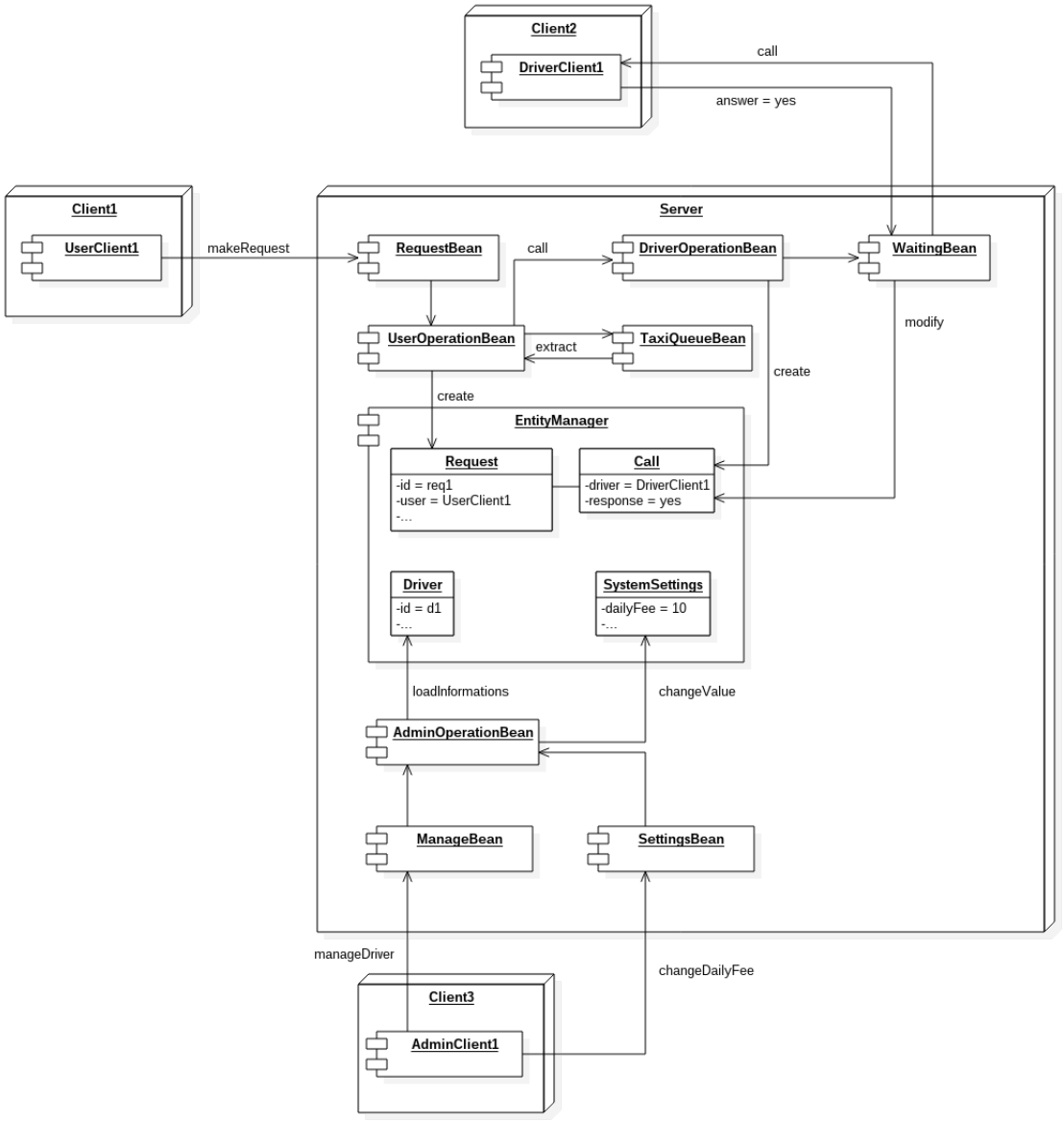


Figure 8: a possible runtime view of the system

For the sequence diagrams, see the related Requirements Analysis and Specification Document, from page 37.

2.6 Component interfaces

We provide 3 different user experience diagram, one for each actor of MTS: user, driver and the administrator. This UX diagrams, according to all the mockups provided in the RASD, define the navigation and the action that each actor can do in MTS System. The blue boxes define the Screens, the yellow ones define input form and the white ones represent data derived from database, according to Class diagram and ER Schema. For user and his user experience, there is no difference between navigate via browser or mobile application.

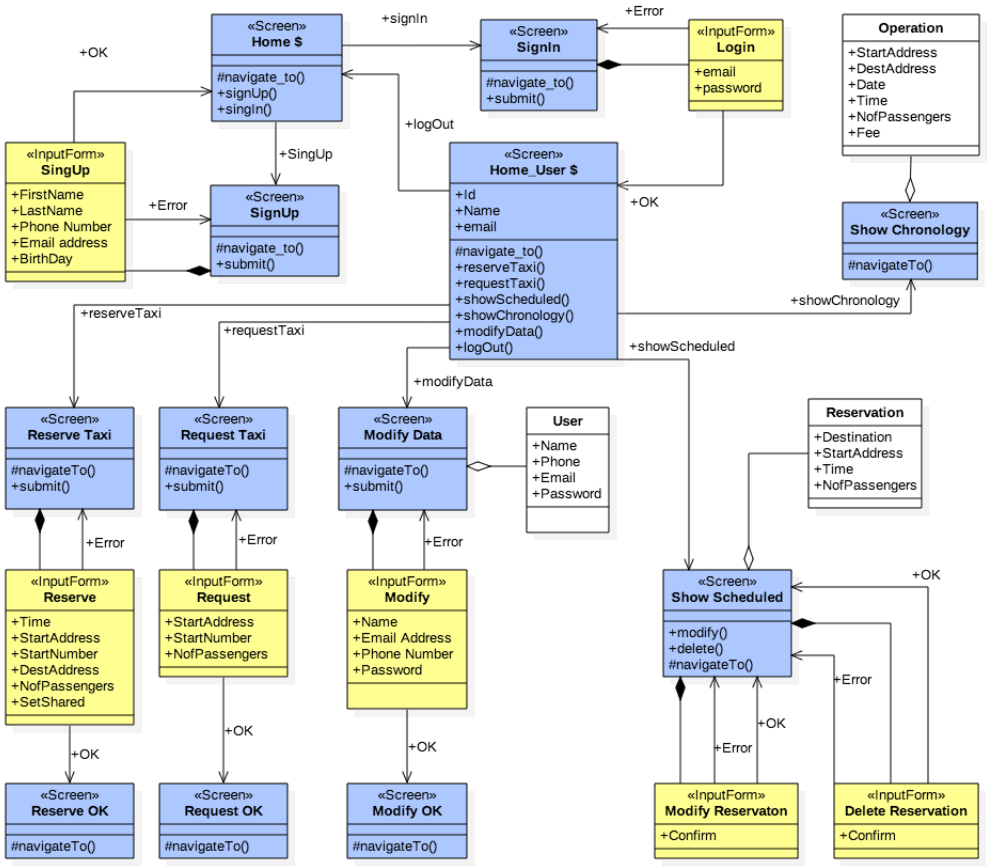


Figure 9: User UX Diagram

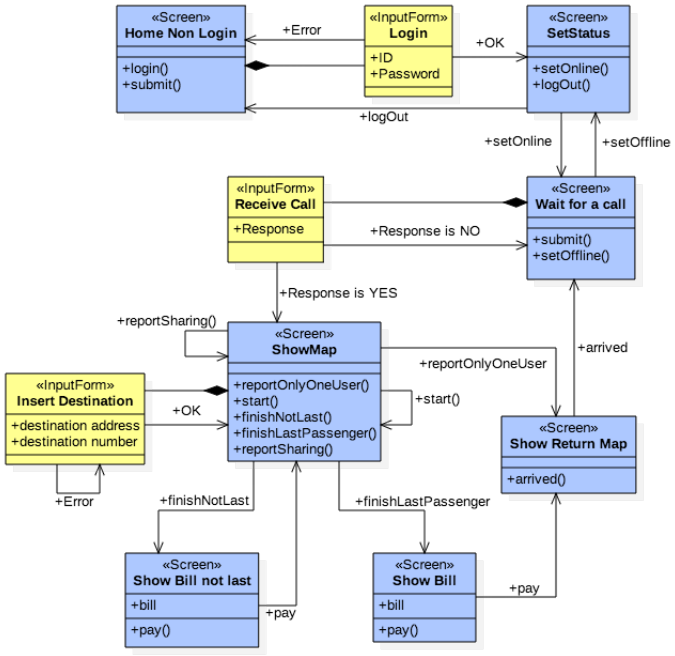


Figure 10: Driver UX Diagram

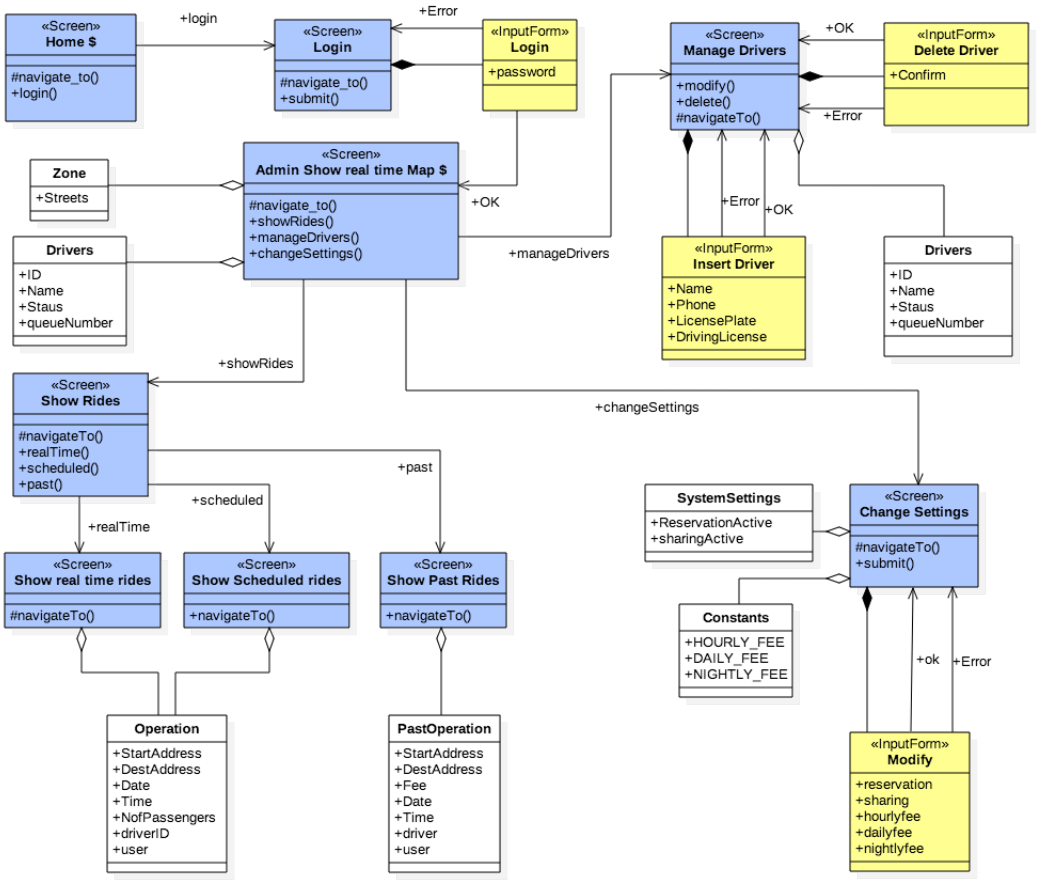


Figure 11: Administrator UX Diagram

2.7 Selected architectural styles and patterns

2.7.1 Architectural styles

MTS System will be based on client-server architecture, this allow the division of roles between clients that represent the users, drivers and administrator interfaces and the server that represents the system. They make request to the server, the engine of the system, that computes every actions of the human actors and response to them. The style choose is tier based, usual for the enterprise application. There are 4 different logic tiers organized in 3 physical tier, that maximize the modularity of the entire system: the client tier, the web and business tier and the data tier. The client tier is implemented on the device of the users/drivers, the web tier and the business tier are on the Application Server, the data tier is implemented by a relational database management system. By using this architectures MTS has the possibility to centralize all data in a physical different server to increase security.

However the implementation of this solution will have high costs in term of building infrastructure for the servers and for the maintaining. So it is possible to adopt the cloud solution, by using the same client-server and four-tier architecture but using a virtualized service provide by some datacentre. This is the best solution in terms of:

- cost because there is only a rent of virtual server;
- security because the datacentres have high policy of security and prevent loss of information;
- availability because datacentres guarantee the presence of current and redundancy of hardware;
- scalability because if the MTS increase registered users or traffic there is the possibility to rent higher spaces without change anything.

The cloud solution is at Iaas level (Infrastructure as service), so we get from the cloud some machines where we install and run the MTS software system. Iaas provide also some load balancing and autoscaling mechanism that creates new instances of the components to enable the scalability if the system required it. Cloud solution allow us to develop software using Java Enterprise Edition and its paradigms. Users can access directly via web browser or via mobile application, without install any plugin or dedicated software (except the install of the mobile application). Due to the technology choise of using Java Enterprise edition, we have to consider some constraint:

- the system is based on JEE 7 platform;
- the system uses Enterprise Java Beans (EJB);
- the system uses Java Persistance API (for the logic unit);
- the system uses JavaServer Faces (JSF) and Servlets for the implementation of web presentation;
- JavaMail for sending mail to user;
- JavaSMS, a library that allow the system to communicate with external SMS gateway server.

On the server side, GlassFish Server is the one selected to provide the service, MySql is the relational database management system, Eclipse plus JEE plugin is the IDE that will use for the developing. Due to this choice the paradigm that will be used to develop MTS is the MVC pattern, that separate Model (so the data), View (the presentation level) and control (the engine, the logic unit). Now we described for each tier some details.

Client tier: it's the level that allow users to connect to the system with their devices. This tier is hosted or on the user's browsers or on the user's device (like smartphones or tablets), on the drives devices and on the device of the administrator. This tier use the HTTP protocol to communicate with the server: if the connection is via web browser, the server response to request with XHTML pages and the client could run scripts in JavaScript language for example to improve the user experience or to made asynchronous call. If the connection is via mobile application (made by the users or by the drivers) the response is sent in XML format that will be parse by the device.

Web Tier: it receives request from client tier. It manage web pages (format .XTML and XML response) and manage beans. All the responses are generated after a specific request made by a client and after the interaction with the business logic. It implements the Java Server Faces technology to manage beans and Java Servlets. This tier has also to control the status of the users and their authorization, by using the cookie technologies and the Java Authentication and Authorization Service (JAAS).

Business tier: it contains the application logic, it communicates directly with the database tier. Like the web tier it is developed with JEE technology and it will run on the compatible server. All the application data are represented by Entity Beans object. The application logic and the communication with the database are made by Enterprise Java Beans components. It has to communicate with an email server that implement the SMTP protocol and with a SMS Gateway server. These servers are external with the respect to MTS system, and has to be provided by the cloud service provider or another provider. It also implements Session Beans to control the authentication of the users.

Data Tier: it contains the relational database management system, MySQL server as said above and it aim is to maintain data persistent. The communication with the logic tier is made with SQL query, using the JDBC technology. Is is a physical tier, so it could be on the same machine that host the logic unit but also be on a different machine. This choice depends by the provider and it has to be indifferent for the developing and practical. The database at the first release of MTS has not to be distributed. But there is nothing that prevent the implementation of this choice. The decision of using distributed database has not influence the rest of the System according to modularity of tier-based architecture.

2.7.2 Patterns

In order to develop a better software we defined some design pattern that are used in MTS system.

- **Observer pattern** will be used to update the status of each taxi that are online: the subject of this design solution is the object represent the taxi that update the observer when it changes it status. The observer is the application server that computes how the taxi has to do. The status should be offline, free, busy and return to zone.
- **Proxy pattern** will be used to update the position of each taxi. This feature is used by the administrator in his homepage, where a map shows him the position of each taxi, as reported in the mockup on the RASD. To prevent a multiple request of the taxis position (for example if the admin makes multiple refresh of the page), proxy server allow us to request periodically the position of each taxi that will be save in the server and show to Administrator when he requires it.
- **Singleton pattern** will be used to create the Administrator, because it has to be only one.

2.8 Other design decisions

2.8.1 About Mobile Applications

We have to make some clarifications about the mobile application. There will be some different releases of users mobile application, one for each most important apps market. So the application will be developed using Java for Android devices, using Swift for apple-based devices and specific Microsoft tools for WindowsPhone devices. About the drivers, in according to RASD, can access to the system with and android device, so there will be a unique release that will not be published on app market but installed in each drivers devices. So, in this design document we don't focus the attention on developing this apps, we define only how they can communicate with the system, so via HTTP request and parsing XML response of the server. Each application will be coded separately, using its standard rules.

2.8.2 More about Data Level

We provide for a better management of data level and to improve the practical implementation of this tier, an Entity Relationship Diagram and it Logic rappresentation. It include what kind of data and how they are connected in the database of system to be. It is built according to Class Diagram described in the RASD.

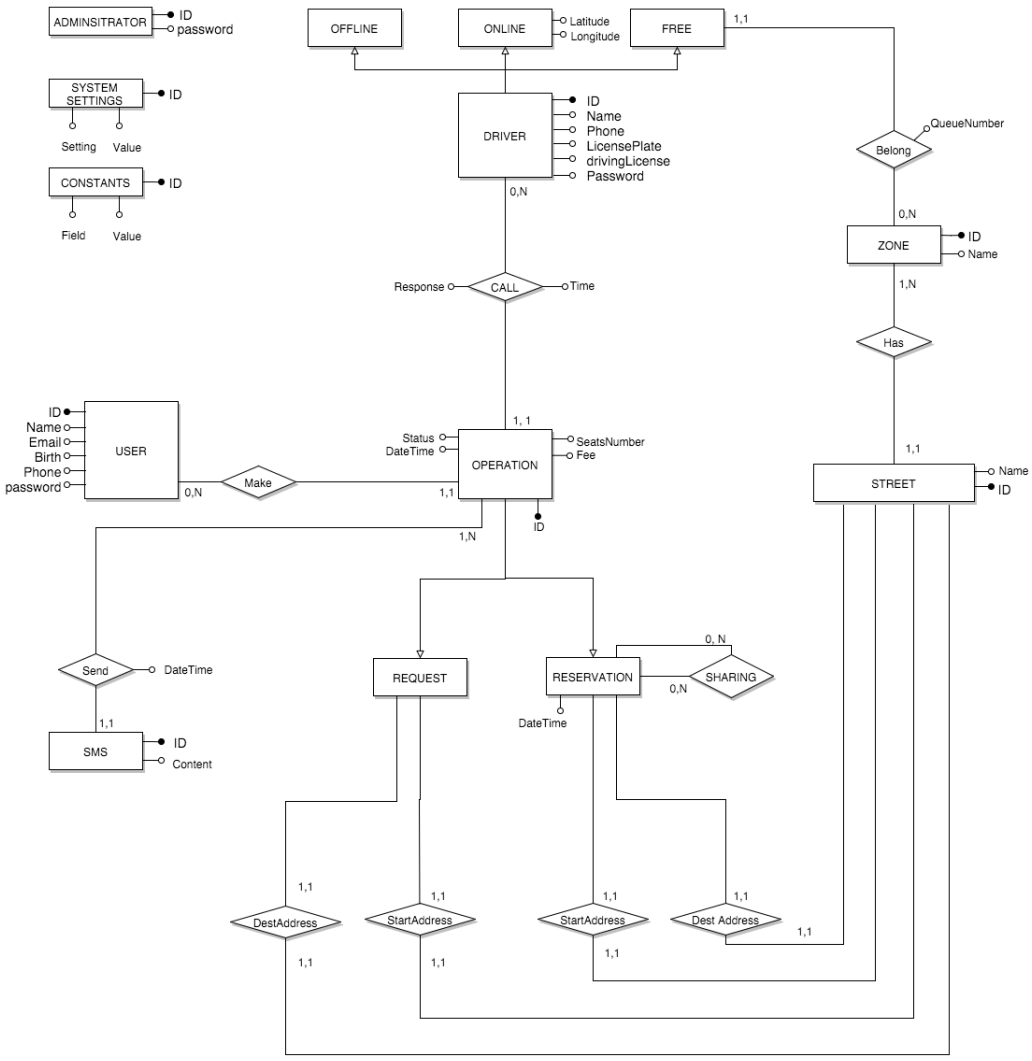


Figure 12: ER diagram

LOGIC DIAGRAM

ADMINISTRATOR (ID, Password)
SYSTEM_SETTINGS (ID, Setting, Value)
CONSTANTS (ID, Field, Value)
USER (ID, Name, Email, Birth, Phone, Password)
USER_MAKE (IDUser, IDOperation)
SMS (ID, IDOperation, Content, Date)
REQUEST (ID, Status, DateTime, SeatsNumber, Fee, StartAddress, DestAddress)
RESERVATION (ID, Status, DateTime, SeatsNumber, Fee, DateTimeStart, StartAddress, StopAddress)
SHARING (IDRes1, IDRes2)
STREET (ID, Name, ZoneID)
ZONE (ID, Name)
DRIVER (ID, Name, Phone, LicensePlate, DrivingLicense, Password, Status, Latitude, Longitude, ZoneID, QueueNumber)
CALL (IDOperation, IDDriver, Time, Response)

3 Algorithm Design

In this part we are focusing in which way are implemented, more or less, the crucial functionalities of the system. In particular:

1. management of taxi queuing;
2. management of the shared-payment;
3. composition of the shared-route;
4. management of the driver allocation.

3.1 Management of taxi queueing

After the driver set is status to online, him will be allocated into the queue of the taxi corresponding to the zone is situated.Moreover, it is allocated in the taxi queue every time he finish a ride or he don't found an user (as tell in the rasd)). In order to grant this , inside the system we have the follow implementation:

```
class TaxiQueues{
  List queues;
  boolean insertDriver (Driver driver) {
    foreach (zone belongs to CityMap) {
      if(zone.contain(driver.getLongitude(),driver.getLatitude())) {
            queues.getQueue(zone).enQueue(driver);
            set Driver queue number;

            return true;

      }
    }

    return false;

  }

  Driver extractDriver (Zone userZone) {
    if(userZone contains at least a driver)

          return queues.getQueue(userZone).deQueue();

    foreach( i as {1,2,3,...})
      foreach(zone distant i from userZone)
        if(zone contains at least a driver)

          return queues.getQueue(zone).deQueue();

    return "error";

  }
  ...
  ...
}
```

We suppose that the methods upon are inside an object "TaxiQueues" consisting in a list of queues associated to each zone. When an user request a taxi, or reserve it, a driver will be extracted from the queue. If the driver accepts will be called extractDriver(), otherwise will be called extractDriver() and then insertDriver() in order to put the driver at the bottom of queue. The method extractDriver() analyzes if in the zone where the driver is situated there will be a taxi waiting for a call, otherwise the system tries to find the nearest zone with at least one taxi.

3.2 Management of the shared-payment

In order to reward all the people for using the sharing, and in order to ease our system, we pre-calculate the price of sharing before the allocation of the taxi. Even so the taxi take 30 minutes from A to B and the optimal time is 15 minutes, the users will pay the price for 15 minutes. The computation happens after the checking of a compatibility. We can summarize the policy of shared-payment by defining the algorithm of the function cost():

```
int cost(Address origin,Address destination,...) {
  cost:=fixed-fee;
  prec:= origin;
  foreach(address as {all crossed addresses from next(origin) to destination
    following the best route in terms of time})
    cost+=(variable-fee * requested_time(prec,address))/(number of people from
    prec to address);

  return cost;

}
```

The method next() calculates the successive address.

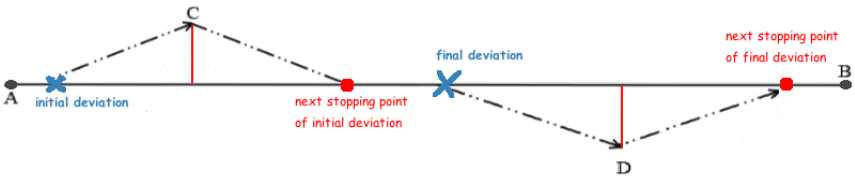
3.3 Composition of the shared-route

Before the allocation of a taxi we have the execution of the method checkCompatibility(). In this method we analyzed the shared-reservation submitted by an user and if there is affinity with another shared-reservation we will add into a container the info of the last one. Otherwise the method return false and the system provide to create a new container consists of the analyzed element. Obviously, the sharing will succeed only if the capacity of the shared-reservation is less than the residual capacity of the taxi. The container is a structure consisting on all reservations that are related to each other following the sharing-relation, it is a way to simplify our algorithm. The elements inside the container are ordered by estimated time arrival(from the top of the queue):

```
boolean checkCompatibilty(Reservation reservation) {
  foreach(container as Container)
    if(residual capacity of the container >= reservation capacity)
      if(there is affinity between container and reservation) {
       insert reservation into container in a position that mantains the
       ascending order of estimated time arrival;
       return true;
          }

  return false;

}
```

The ordered-insert is implemented by using any kind of algorithm because we have at most 4 elements as we known. Inside the method we have the parameter named “value” . In a nutshell, it is a time (in seconds) that represent the maximum distance from each of the the addresses contained in the shared-route. The affinity can be implemented as follow:

```
boolean affinty(Container container, Reservation reservation,Integer value,...){
  Address A := start address of the first reservation into container;
  Address B := destination address of the last reservation into container;
  Address C := start address of the reservation;
  Address D := end address of the reservation;
  boolean flag := true;
  foreach(initialDeviation as {all addresses from A to B})
    if(requested_time(initialDeviation,B)<value) {
      foreach(finalDeviation as {all addresses from C to B})
        if(requested_time(finalDeviation,D)<value)
          if(finalDeviation is collocated between C and nextStoppingPoint(initialDeviation))
            if((cost  from initialDeviation to C+cost from C to finalDeviation with one person
              more + cost from finalDeviation to D with one persone more+cost from D to
              nextStoppingPoint(initialDeviation))< cost from initialDeviation to
              nextStoppingPoint(initialDeviation))

              return true;

            else{
              if(there is at least a next stopping point from initialDeviation
                to finalDeviation){
                flag = true;
                foreach(stop_point in {all stopping points from initialDeviation
                  to finalDeviation})
                  foreach(user that have stop_point as destination address or starting address){
                    if(the route of user is not included between intial deviation
                      and final deviation)
                      if(stop_point is a destination address)
                        if((cost from the starting address of user to initialDeviation +
                          cost  from initialDeviation to C+ cost from C to
                          nextStoppingPoint(intialDeviation) with one person more + cost from
                          it to stop_point) >= cost from the starting address of
                                                user to stop_point){

                          flag := false;

                        }
                      else
                        if((cost from starting address of user to finalDeviation with one person
                          more + cost from finalDeviation to D with one person more +
                          cost from D to nextStoppingPoint(finalDeviation) + cost from it to
                          stop_point)>= cost from stop_point to the destination
                          address of the user){

                          flag := false;

                        }
                   }
                   if(flag)
                     return true;
              }
              else
                if((cost  from initialDeviation to C + cost from C to D with one person more+
                  cost from D to nextStoppingPoint(finalDeviation)) <
                  cost from initialDeviation to nextDestinationPoint(finalDeviation)){

                  return true;

                }
            }

  return false;

}
```



The function nextStoppingPoint() returns the next stopping point from a particular address. The function cost() takes care with the number of the people. It calculates the price from a place to another in which we consider a particular number of people and the route with the best time from the origin to the destination (thanks to googleApi). This function is described more deeply in the next section. In our system works a policy that rewards the " lower price". In fact the system will add a new reservation to the container even if the starting point or the destination point are not contained in the original route. If we had chosen a policy that rewards the "time", we would have added only the reservations contained in the original route.

3.4 Management of the driver allocation

If a driver set is status to ready either he will be driven by the system in a place where he has to wait for a new call or he will wait where is located. This choice is made by the system according to the following policy: the driver remain in his zone if and only if the average of drivers in each zone is greater than the number of drivers located in the same zone of him. In the event that the condition isn't satisfied the driver will be driven in the nearest zone where it's valid. We can sum up the previous statement with the following algorithm:

```
driver_zone:= the zone of a ready driver;
count:= 0;
avg := 0;
i := 0;
foreach(zone as Zone){
  count++;
  avg=avg+size of the queue in zone;
}
avg := avg/count;
if(the size of the queue in driver_zone < avg)
  insert driver in the queue;
else
  foreach( i as {1,2,3,...})
    foreach(zone distant i from driver_zone )
      if(the size of the queue in zone < avg)
        drive the taxi in zone and then insert driver in that queue;
```

4 User Interface Design

We have alredy provided a detailed set of mockup describing the user interface of our system in the related RASD; so here we only provide a mockup of setting screen for the user that we do not represent in the RASD document.

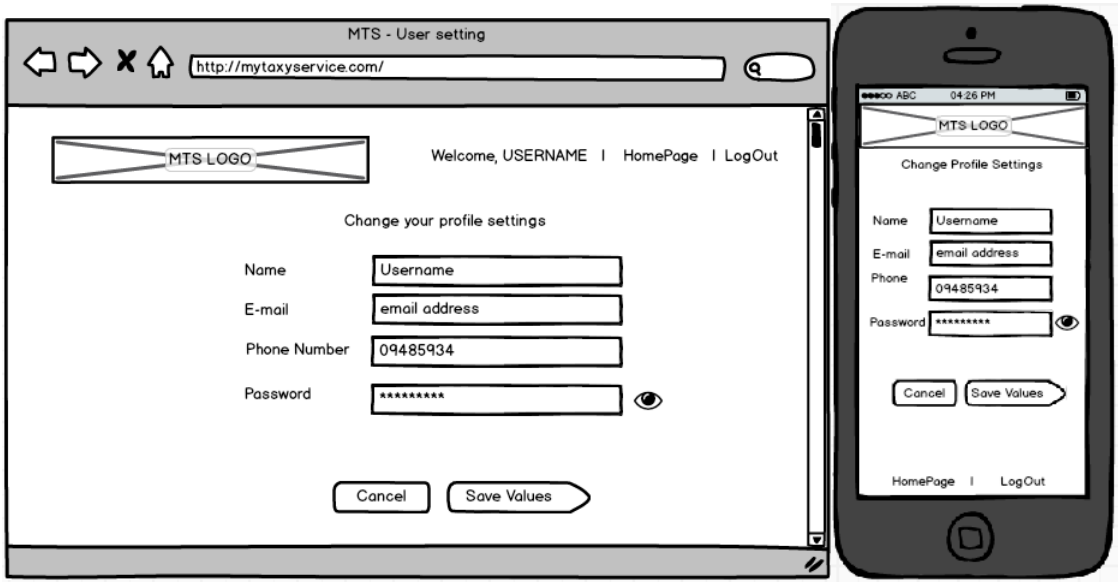


Figure 13: user settings page and mobile app interface

5 Requirements Traceability

In the RASD we identified the following functional requirements:
Users' functionalities:

- Sign up [**FR0**];
- Log in [**FR1**];
- Log out [**FR2**];
- Recover his password [**FR3**];
- Make a taxi request [**FR4**];
- Make a taxi reservation [**FR5**];
- Receive a confirmation sms [**FR6**];
- Modify a taxi reservation [**FR7**];
- Delete a taxi reservation [**FR8**];
- See all his reservations [**FR9**];
- Filter his reservation by date and price [**FR10**];
- Change his informations[**FR11**].

Drivers' functionalities:

- Log out/in [**FR12**];
- Set his status [**FR13**];
- Accept or deny a request [**FR14**];
- See the route toward a destination [**FR15**];
- See the fee for a ride [**FR16**];
- See the route toward a zone [**FR17**];
- Being notified when a user wants to join a shared ride [**FR18**];
- Receive an administrator's message [**FR19**].

Administrators' functionalities:

- See the current position of every taxi [**FR20**];
- See the current status of every on-line taxi (driving, waiting) [**FR21**];
- See the information about the current/past/scheduled rides [**FR22**];
- Add and remove taxi drivers from the system [**FR23**];
- Set the hourly fee [**FR24**];
- Set the fixed daily fee [**FR25**];
- Set the fixed nightly fee [**FR26**];
- Cancel a driver [**FR27**];
- Enable/disable additional services [**FR28**];
- Add a driver[**FR29**].

Some of the functionalites are partially included in the following UI-oriented components (which are responsabile of encapsulating both actor actions and system responses):

User Manged Beans:

- GuestHomeBean[**FR0**][**FR3**];
- LoginBean[**FR1**];
- HomeBean[**FR2**];
- RequestBean[**FR2**][**FR4**];
- ReservationBean[**FR2**][**FR5**];
- ConfirmationBean[**FR2**];
- SettingsBean[**FR2**][**FR11**];
- ChronologyBean[**FR7**][**FR9**][**FR10**].

Admin Managed Beans:

- MainBean[**FR20**][**FR21**];
- SettingBean[**FR24**][**FR25**][**FR26**][**FR28**];
- ManageBean[**FR27**];
- AddTaxiBean[**FR29**];
- CurrentBean[**FR22**];
- ScheduledBean[**FR22**];
- PastBean[**FR22**].

Driver Managed Beans:

- LoginBean[**FR12**];
- StatusBean[**FR13**];
- Waiting Bean[**FR14**];
- MapBean[**FR15**][**FR16**][**FR17**].

Others are partially included in the following business-oriented components:

- AuthenticationBean[**FR0**][**FR1**][**FR2**][**FR12**];
- UserOperationBean[**FR3**]-[**FR11**];
- TaxiQueueBean[**FR17**];
- PaymentBean[**FR16**];
- SharedRideBean[**FR5**];
- DriverAllocationBean[**FR17**]-[**FR19**];
- DriverOperationBean[**FR13**];
- AdminOperationBean[**FR20**]-[**FR29**];
- EntityManagerBean[**FR0**]-[**FR29**].

6 Work Time

To produce this document, each of us has worked around 24 hours.