Design Document

Alessandro Comodi, Davide Conficconi, Stefano Longari

December 3, 2015

Contents

1 Introduction **4**

1.1 Purpose 4

1.2 Scope 4

1.3 Glossary 5

1.4 Reference Documents 6

1.5 Document Structure 6

2 Architectural Design **7**

2.1 Overview 7

2.2 High Level Components and Their Interaction 7

2.2.1 Client tier 9

2.2.2 Web tier 9

2.2.3 Business tier 9

2.2.4 Data tier 9

2.3 Component View 10

2.3.1 Client tier 10

2.3.1.1 Mobile-Client 10

2.3.1.2 Web-Client 10

2.3.2 Web tier 11

2.3.2.1 Web-Server 11
2.3.3 Business tier 11
2.3.3.1 Connection-Handler 11
2.3.3.2 Application-Controller 11
2.3.3.3 Ride Manager 11
2.3.3.4 Queue Manager 12
2.3.3.5 Security Manager 12
2.3.3.6 Data Access Manager 12
2.3.4 Data tier 12
2.3.4.1 DBMS 12
2.3.4.2 E-R schema of the Database 13
2.4 Deployment View 14
2.5 Runtime View 15
2.5.1 Login 15
2.5.2 Shared request 16
2.5.3 Availability Management 18
2.5.4 Reservation 18
2.6 Component Interfaces 19
2.6.1 Business tier and Data tier 19
2.6.2 RideManager interfaces 20
2.6.3 QueueManager interfaces 21
2.6.4 SecurityManager interfaces 22
2.6.5 ConnectionHandler interfaces 22
2.7 Selected Architectural Styles, Patterns and Other Design Decisions 23
2.7.1 Client-Server 23
2.7.2 Multi-tier Architecture 24
2.7.3 Model-View-Controller 24

2.7.4 Observer pattern 24

3 Algorithm Design **25**

3.1 QueueManager Main Positioning Algorithm 25

3.1.1 How does it work 25

3.2 QueueManager Main Distribution-Prevision Algorithm 26

3.2.1 How does it work 26

3.3 RideManager Main Ride-creating Algorithm 26

3.3.1 How does it work 27

3.4 RideManager main Reservation-handling Algorithm 28

3.4.1 How does it work 28

4 User Interface Design **29**

4.1 Mockups 29

4.1.1 User side application 29

4.1.2 Driver side application 30

5 Requirement's Traceability **41**

5.1 Registration and Login 41

5.2 User side taxi reuqest, sharing, reservation 41

5.3 Taxi driver side and notifications 41

5.4 Queue management 42

6 Other Information **43**

6.1 Working hours 43

6.2 Used tools 44

Chapter 1

Introduction

1.1 Purpose

The Design Document has the objective of defining the detailed design choices for MTS, explaining with accuracy the components of the system. In particular it has the purpose of opening the black box system showed in the RASD and analyzing his components.

1.2 Scope

As described in the RASD, the system to be implemented serves to help users in requesting or reserving taxis and to manage fairly the queues of taxis around the various zones of the city.

Users who are enrolled with the system can make requests or reservations. To insert the starting and destination addresses there is the possibility to use a map, ore eventually write the addresses by hand. Users are also able to add the sharing option, thanks to which they can share the ride with other users and split the price.

Taxi drivers, on the other hand, have the possibility to confirm or decline rides. They can count on a fair management of the taxi queues which distributes cleverly the taxis so that it is impossible to have drivers to whom do not arrive any ride request. This is given by the fact that each taxi, when set to available, is added in a FIFO queue of a specific zone.

1.3 Glossary

In this section will be explained the unclear terms and achronyms in order to avoid ambiguities and help the reader in understeanding this document.

- S2B: it is the achronym for "System To Be".
- RASD: Requirement Analysis & Specification Document.
- DB: acronym for DataBase.
- DBMS: DataBase Management System.
- MVC: Model View Controller design pattern.
- FIFO: First-In First-Out policy.
- OS: Operating System.
- JavaEE: Java Enterprise Edition.
- XML: It is the eXtensible Markup Language.
- MTS: MyTaxiService, it is the name of the application.
- JDBC: Java DataBase Connectivity.

1.4 Reference Documents

The documents used as references in order to redact this document are:

- IEEE Std 1016 - 2009 for Software Design Descriptions.
- RASD for MTS, previously redacted.

1.5 Document Structure

This document is structured in Chapters, each describing a different design aspect of the S2B.

- Chapter 2 deals with the Architectural design of the system, describing a detailed view of the various components and their behavior among each other.
- Chapter 3 analyzes the main algorithms that are needed in order to let the application work as required. These algorithms are written in natural language.
- Chapter 4 is about the User Interfeces design and here are present the mockups thanks to which the developer can have an idea on how the front-end of the application must be implemented.
- Chapter 5 shows the link between the choices made in the design of the system and the requirements written in the RASD.
- Chapter 6 contains information relative to this document but not with the system itself.

Chapter 2

Architectural Design

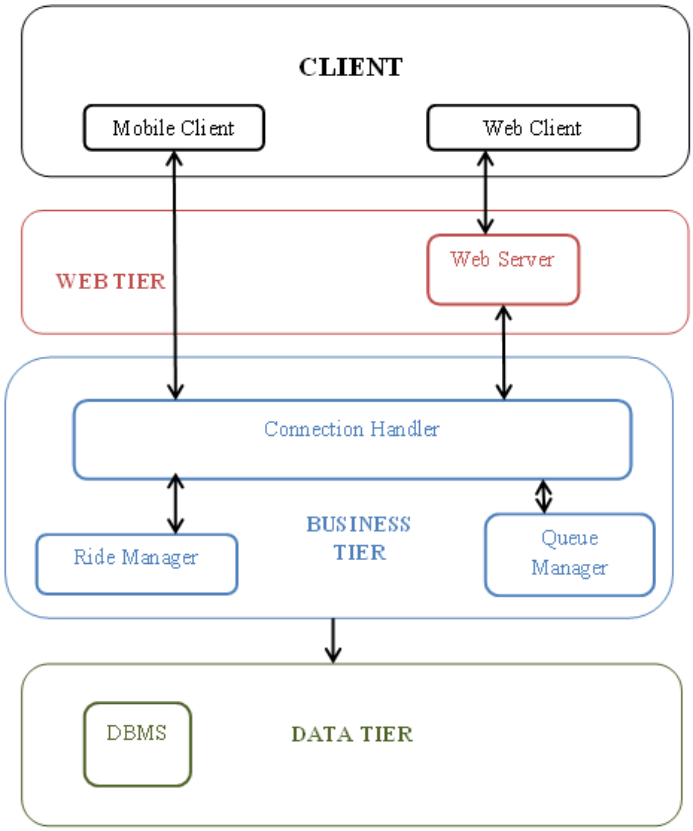
2.1 Overview

This section displays the main architectural project. Firstly it's presented a high level representation about our S2B and its components interactions. Afterwards the document shows in a deeper way different kinds of view about the system. Eventually it describes other various features about design and architecture decisions.

2.2 High Level Components and Their Interaction

This section aims to provide, through the diagram in Figure 2.1, a general overview about system's architecture and main interactions between each component.

Figure 2.1: High level components



The architectural style appointed is a client-server application with a multi-tier style: a client tier, a tier for the business logic, a web tier and a tier for the data storage. All main communications between each component and, or, tier happen in an asynchronous way.

2.2.1 Client tier

This layer represents the different methods of access to the application: via one of the main common browsers (e.g.: Firefox, Chrome, etc. . .), or through the mobile application, available for the main mobile OSs. This logic level has the duty of representing information to the users and interacting with them (this can be seen as the View part of the MVC pattern). The Web Client communicates with the business logic only through a Web tier.

2.2.2 Web tier

This layer contains a Web Server that functions as middleware between the Web Client and the Business Logic Application.

2.2.3 Business tier

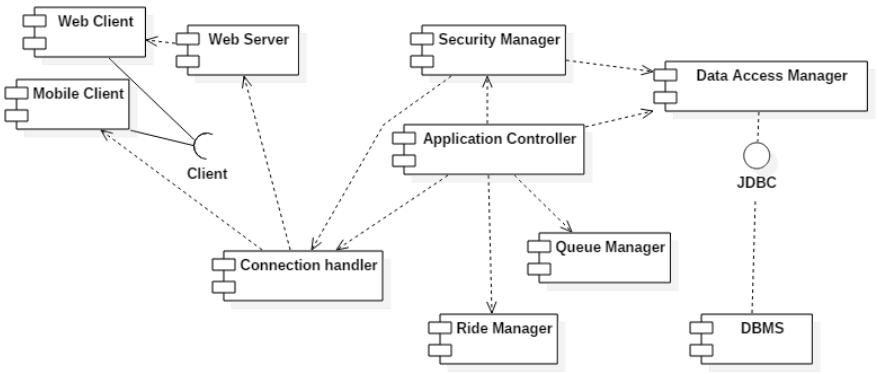
This layer represents the system's core in fact all the application logic is inside this tier. It has three main components: one represents the Manager of all the rides, another one the Queue Manager and in the end a component which handles all the connections with the front end side, called Connection Handler. The Business tier can also access to the data in the Data tier.

2.2.4 Data tier

This layer contains a DBMS which takes care about the data, their storage and their access. Only the Business tier can access to this tier.

2.3 Component View

Figure 2.2: Component view



2.3.1 Client tier

2.3.1.1 Mobile-Client

The mobile client is the application that has to be downloaded and installed in the mobile devices from the application stores of the various OSs. It communicates directly with the connection handler through which it sends and receives all the messages necessary to the application in order to fulfill its functionalities.

2.3.1.2 Web-Client

The web client, differently from the mobile client, has no need to be installed in the PC device of the users, but is sufficient to use a web browser (e.g. Chrome, Safari, Firefox, ...). All the messages that the web client sends and receives from the Connection handler pass through an external Web server.

2.3.2 Web tier

2.3.2.1 Web-Server

This component, which is not integrated within MTS, is an external service offered by GlassFish and is written entirely using Java EE. It puts in communication the web client with the connection handler. This component is needed in order to let the information coming from the connection handler to be readable by the web browser, so it handles only the presentation layer of the application. All the application logic is handled by the Application controller in the Business tier.

2.3.3 Business tier

2.3.3.1 Connection-Handler

The connection handler is the component that handles all the communication between the client tier and the application itself. It is connected with the Controller application and the Security Manager. It sends all the information parsed in XML transparently with respect to the client components.

2.3.3.2 Application-Controller

This component is the core of the application, it communicates with all the other components in the business tier. It handles the logic of the application.

2.3.3.3 Ride Manager

This component is necessary to handle successfully all the matchings between a user and a driver in order to have all the right information of a ride and to send all the notifications to the right users and drivers. It contains some of the main algorithms described in Chapter 3.

2.3.3.4 Queue Manager

This component is another important part of the application since it handles all the placements of the various taxis around the zones. It contains the main algorithms for the queue management described in Chapter 3.

2.3.3.5 Security Manager

The security manager is the component that has to grant the security of the application. It prevents guests who are not enrolled to the system to access to it. It also ensures that all the private information about the users are kept safe.

2.3.3.6 Data Access Manager

This component has the task to communicate with the Data tier and more precisely with the DBMS. It receives all the information to store or to retrieve to and from the DBMS. It uses JDBC to communicate with the DBMS.

2.3.4 Data tier

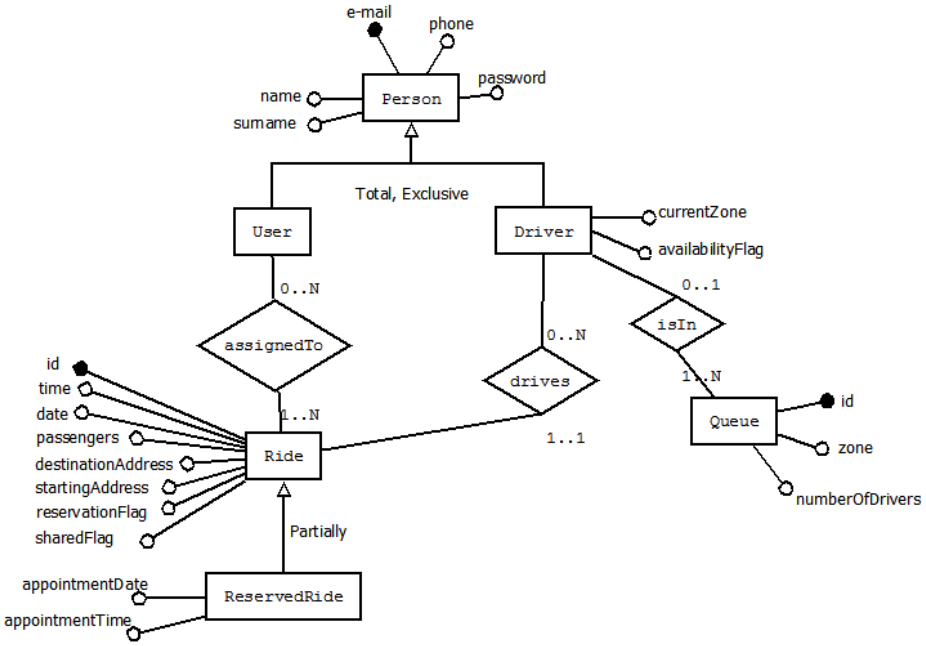
2.3.4.1 DBMS

This is the Data Base Management System, which has the task to store correctly all the data received from the business tier and to retrieve all the information that the Data access manager asks to it. The DBMS must ensure the durability and consistency of the stored data.

2.3.4.2 E-R schema of the Database

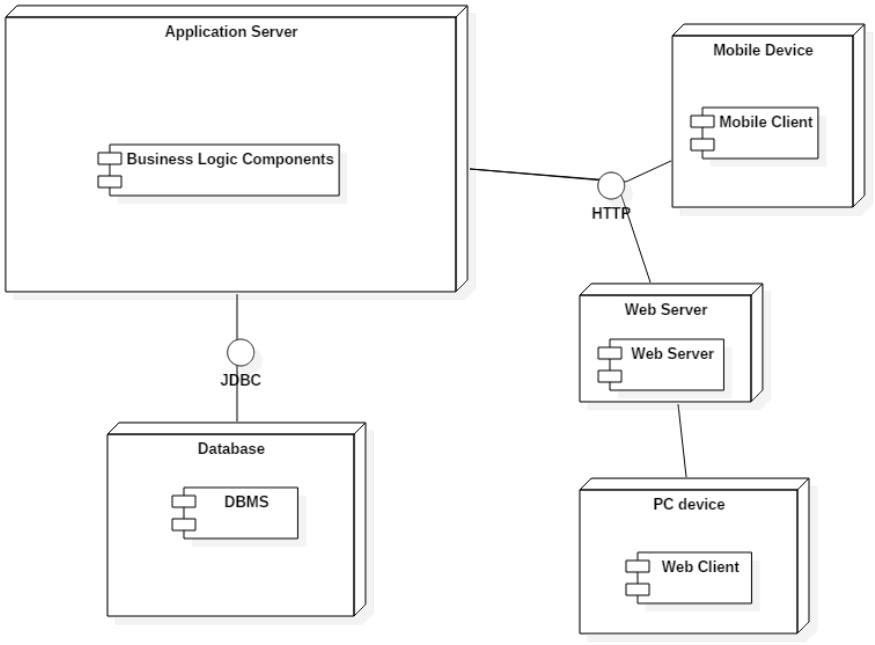
In this subsection there is an Entity-Relation schema representing an overview of the DB entities.

Figure 2.3: E-R schema



2.4 Deployment View

Figure 2.4: Deployment view



The S2B was thought to mirror the architectural design, therefore each logic tier is mapped into a physical layer. The client side of the application is sited on a device for each user: the mobile client is deployed on a mobile device, while, on the web side, all users can reach the application through one of the main common browser. The web server is placed on a separate machine because it is going to be deployed on an external service like Glassfish. All the Business logic components are deployed together on a server farm, since the application is meant to be used by a large number of users. The DBMS stands apart from all other components for security purposes.

2.5 Runtime View

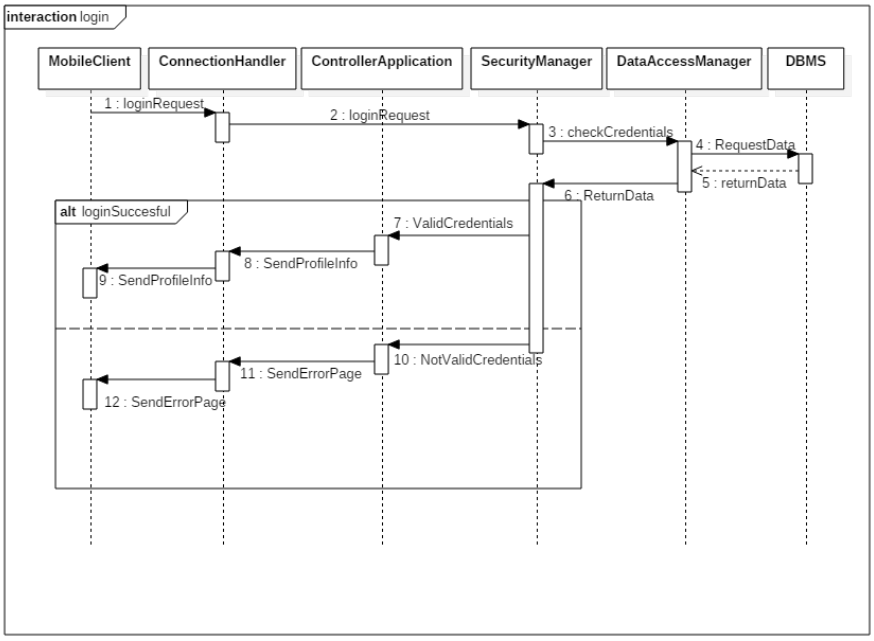
In this section the reader can find the modality in which the various components interact with each other, and this is explained through the use of some UML sequence diagrams, with their relative description.

2.5.1 Login

The Login sequence diagram is exposed in Figure 2.5.

When an application requests to login it sends its mail and password to the system. The connectionHandler gets its request and translates it for the security manager. This asks the dataAccessManager to query the DBMS for the username and his password. When the query answer returns to the securityManager if they match the username and password given by the application the profile of the user that corresponds to those credentials is returned to the device through the controllerApplication.

Figure 2.5: Login sequence diagram

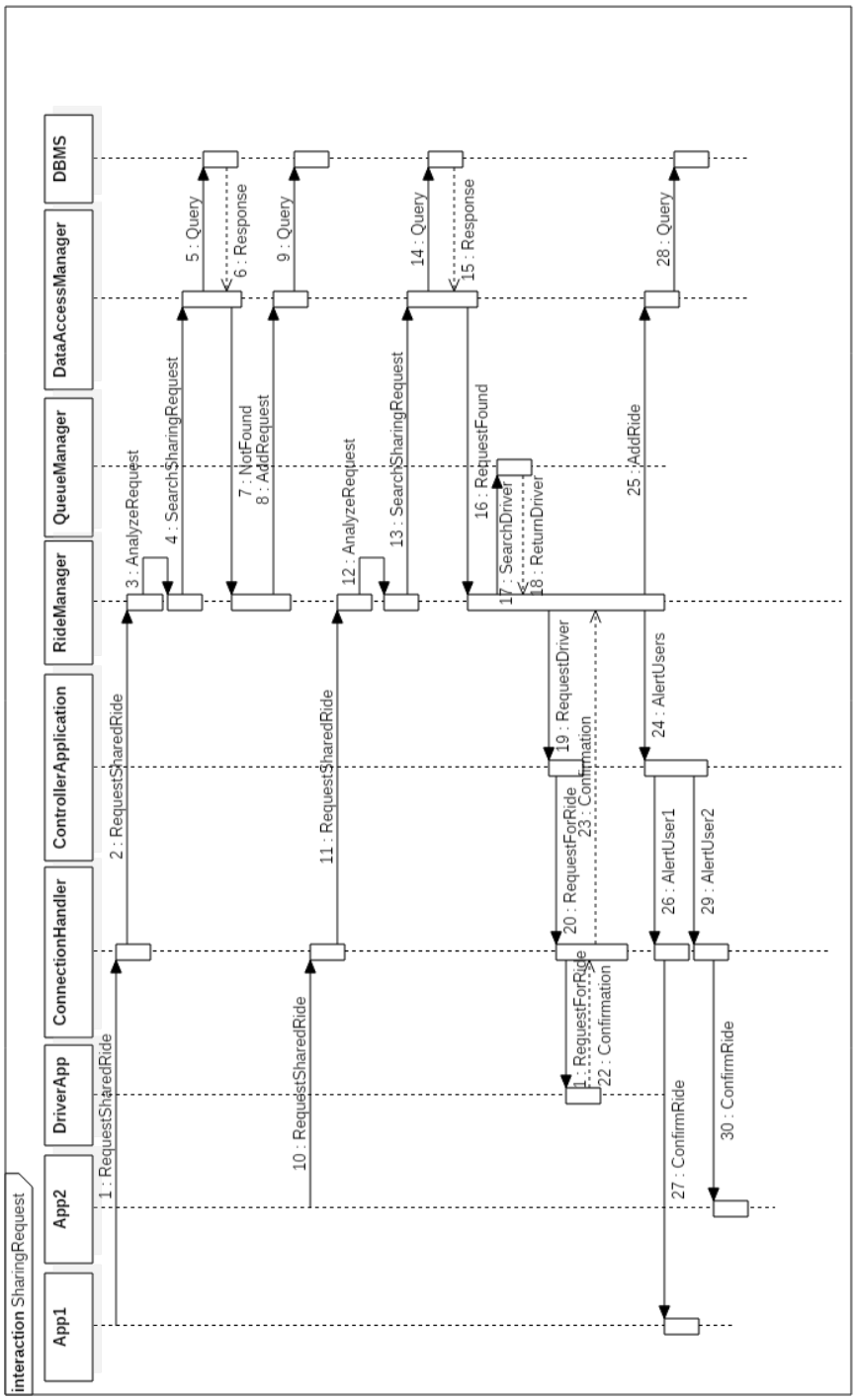


2.5.2 Shared request

The sared request sequence diagram is exposed in Figure 2.6.

The request for a ride by a logged user is received from the rideManager that analyzes it. If it is a request to share a ride the rideManager asks the dataAccessManager to search in the DBMS for other sharing requests. If the dataAccessManager doesn't find any sharing requests near the area of this one, the rideManager leaves the request pending in the DBMS for some minutes before canceling the sharing option and sending a taxi to the user. If in these minutes another sharing request arrives from the same area of the first one it checks if the two requests are compatible and in case they are it provides a shared taxi for both the users.

Figure 2.6: Shared request sequence diagram

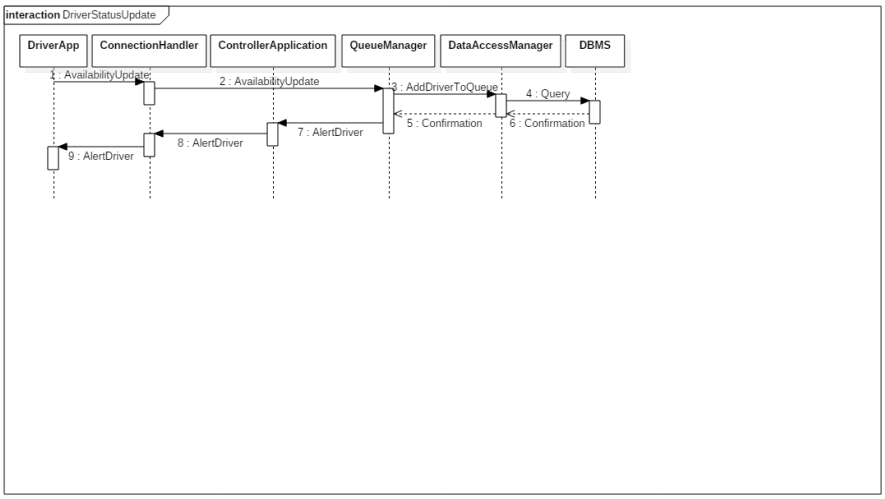


2.5.3 Availability Management

The availability management sequence diagram is exposed in Figure 2.7.

When a driver changes his availability his application requests the QueueManager to set him available/busy. The QueueManager then alerts the dataAccessManager to set the driver available/busy in the DBMS, in addition to put him in a queue if the status change is to available.

Figure 2.7: Availability management sequence diagram

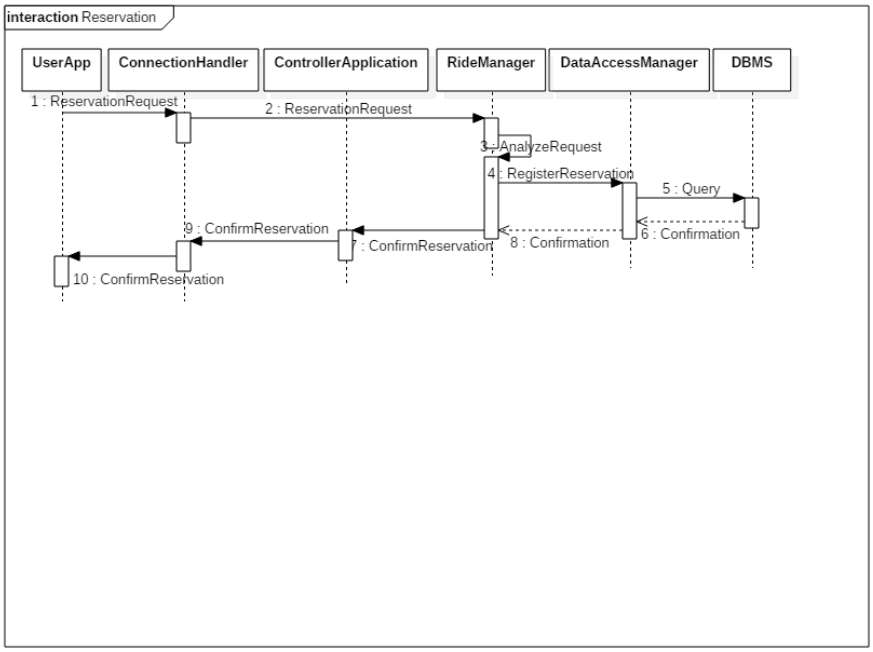


2.5.4 Reservation

The reservation sequence diagram is exposed in Figure 2.8.

When a reservation for a taxi is made, the rideManager registers the request in the DBMS to remember it. Every five minutes a program runs to check all the reservations and to alert the rideManager if there are some that are in need to be set up as rides.

Figure 2.8: Reservation sequence diagram



2.6 Component Interfaces

In this section the reader can find how the various components interact among each other, i.e. the methods offered and visible to the external world.

2.6.1 Business tier and Data tier

Business and data tier communicate through the Data access manager and the DBMS. The two components can interface one with the other thanks to JDBC, which takes care of all the interaction between the Data access manager and the Database. Thanks to JDBC the developers have no constraints on what kind of DBMS they are going to adopt.

2.6.2 RideManager interfaces

This component offers several methods regarding the rides and the most important are listed below:

- *Ride createRide(User user, String startingAddress, String destinationAddress, boolean sharingOption);*
  - This method is callable from other components in order to create a new Ride. It will gather the information present in the parameters which are the user, the starting and destination addresses and the sharing option and it will return the created Ride. The ride will contain all the information relative to it, which include the starting and destination addresses, the user, the driver assigned, the sharing option. The created ride will be only a request because, in order to create a reservation ride, a different method must be called.
- *Ride createRide(User user, String startingAddress, String destinationAddress, boolean sharingOption, Date date, Time time);*
  - This method is similar to the previous one, but it includes also the date and time as parameters, which indicate that the ride is a reservation and not a simple request. It returns the created Ride as before, which will contain also the time and date of the appointment of the reservation.
- *Ride reservationInfo(User user);*
  - This method will return all the rides that correspond to a reservation given in input a specific user.
- *Ride requestInfo(User user);*

  - This method will return the pending request of the user if there is one, otherwise it will return a *null object.*

- *boolean deleteRide(Ride ride);*

  - This method deletes the ride that is given as parameter. The ride is deletable only if it is a reservation and, if so, it returns "true", otherwise it returns "false".

2.6.3 QueueManager interfaces

The queue manager offers all the interfaces regarding the allocation of the taxi drivers and their queuing.

- *Driver getDriver(String startingAddress);*

  - This interface retrieves the first available taxi driver relative to the zone of the input address and returns a driver, which will be associated to a ride.

- *void dequeueDriver(Driver driver);*

  - This method is used to dequeue a taxi driver from its relative zone when this one is set to "busy". There is no return value.

- *Zone enqueueDriver(Driver driver, Zone currentZone);*

  - This method offers the possibility to enqueue a taxi driver whenever he becomes "available". The queue manager analyzes the distribution of taxis following the relative algorithm in Chapter 3 and assignes a zone to the driver. The assigned zone is returned from this method.

2.6.4 SecurityManager interfaces

The security manager has to offer the possibility to grant the authenticity and safeness of data and information.

- *boolean validID(String IDdocument);*
  - This method checks whether the ID document inserted by a guest in order to register is valid or not. The input parameter is a string containing the ID code and the return value is a boolean: true if the ID is valid, false otherwise.
- *boolean validCredentials(String user, String password);*
  - This method verifies if the user e-mail and password match or if they are present in the system. As input parameters there are two strings representing respectively the user e-mail and the associated password. The return value is a boolean which is: true if the user has inserted correct credentials or if he is actually registered, false otherwise.

2.6.5 ConnectionHandler interfaces

This component has the important task to put in communication the client tier with the business tier. It offers all the procedures that consent the information to travel from one side to the other and viceversa. The interfaces offered are many and here the reader can find some of the most important ones.

- *Object fromClientToServer(Object message);*
  - This method, in which there are generic objects as parameter and return types, permits the communication from client to server. The

generic “message” can be complex, in fact it contains all the information about the message as the user who sent it and the data stored within it.

- *Object fromServerToClient(Object message);*
  - As in the previous method, this one grants the communication from the server to the client tier. In this case the message contains the information about who is the user that has to receive the message and what is the data contained within it.

2.7 Selected Architectural Styles, Patterns and Other Design Decisions

In this section all the main architectural Styles, Patterns and other design decisions are presented and the reasons that brought to select them are explained.

2.7.1 Client-Server

This architectural styles is adopted due to the fact that the application was thought as follows: a user, who registers himself, accesses to the client application side, via mobile applcation or via web browser, and asks for a service. The kind of service depends on the type of user: if it is a customer user, the service required is to provide to him a taxi; if it is a driver user, the service required is to provide him, or her, a fair management of his/her future ride and assign him to a zone. Therefore the Server has to provide all this kind of services in the best possile way.

2.7.2 Multi-tier Architecture

The architecture of the S2B is a multi-tier architecture: Data tier, Business tier, a Presentation tier interposed,only for the web side, and a final Presentation tier. This choice allows to minimize as possible the feasible coupling between components and an implicit security feature. Thanks to the multi-tier architecture the system will be easly maintainable and the scalability it's improvable.

2.7.3 Model-View-Controller

The MVC pattern is adopted for the primary front-end purpose of the application: the Model is represented by the principal business components, which achieve all the required functionalities, and by the DB; the View is the client, both mobile and web, with the addition of the web server which also fulfils the same objective. The Controller of the application is the "Application Controller" component which has to catch the changes in the Model. The same pattern is selected to give to the S2B a Programmatic Interface to the server side of the application.

2.7.4 Observer pattern

When an user receives the confirmation about his ride he can check the status of the taxi, or when he does a reservation he can check the status of his application. All this kind of events are solved with a Observer pattern: when the position of the taxi, or the reservation, (the subject), changes its status, the user's client (observer) is notified about these variations.

Chapter 3

Algorithm Design

3.1 QueueManager Main Positioning Algorithm

Its objective is to distribute taxis in order to avoid overpopulation of a zone and lack of drivers in another.

3.1.1 How does it work

When a taxi driver sets himself as available the system is alerted and checks his position through GPS. It then searches for all the zones with a low amount of taxis compared to the number of requests per hour in the zone. If the taxi driver is already in one of those zones he is directly put in the queue of the zone. If is not, he is given a series of zones in which he can go to be put in a queue. The system then continues to check the position of the driver until he reaches one of those zones. As he reaches one he is put in that queue and alerted.

3.2 QueueManager Main Distribution-Prevision Algorithm

Its objective is to register every ride and use those data to check taxis distribution.

3.2.1 How does it work

Each time a ride is set the system registers that information with the place and time (considering days of the week). Each time a taxi reaches the position to take his clients the system registers the amount of time needed. Each time a taxi ends a ride the system registers the amount of time used by the taxi. Each time a request cannot be fulfilled because there is no taxi available the system registers it. Using those data, knowing the amount of taxi in each zone and the general distribution the algorithm then can proceed to calculate which zones have already the number of taxi needed to accept all the incoming requests in a reasonable time and which ones instead are too slow or will be. If there are zones of the second kind, when the Main Positioning Algorithm receives a request for positioning someone it is given one of those. If the city zones are instead already all of the first kind the system will send taxis to different zones, starting from the ones with more requests per hour, to try to statistically allow every driver to work the same as others.

3.3 RideManager Main Ride-creating Algorithm

Its objective is to create rides from requests and reservations considering the sharing option.

3.3.1 How does it work

Every time a request is created the system starts this algorithm to try to match the request with an actual ride: It checks for the starting position of the request to know in which zone it is in, and asks the QueueManager for informations about the drivers in that zone, like the maximum number of seats available. If the request has the sharing option enabled the system searches for other pending requests with sharing enabled. If there aren't the request is left on waiting status for 5 minutes, to wait for other requests with sharing enabled. If one is found the system in the next step handles the two requests as one with the amount of passengers that is the sum of those of the requests. If another is not found the request is handled as one with no sharing option enabled. The system now asks the QueueManager for a driver in that zone that has the number of seats necessary to take the request. (it is the QueueManager that has to care about finding the first one in queue, the RideManager has no informations about that) If there is one, the user(s) is alerted about the incoming taxi and a ride is created matching user(s) and driver. If there is no taxi available with that amount of passengers (or no taxi in queue at all) the system checks the nearest zone to the starting position of the request. If in the queue of that zone there is a taxi available with that amount of passengers the ride is created with that taxi and the user(s) alerted it might take some time before the taxi arrives. If there is no taxi available within all the zones adjoining the one from which the request comes from, the user is alerted to try again in some minutes and the request dismissed. If the request was the union of two sharing requests those are divided and the system searches for them as they were normal requests.

3.4 RideManager main Reservation-handling Algorithm

Its objective is to assure the reservations are fulfilled and the user doesn't have to wait for the taxi at the moment indicated.

3.4.1 How does it work

One hour before the reservation starting time the system checks the position of the reservation and through that the zone it is in: The algorithm at this point postpones itself for an amount of time relative to the amount of taxis in the area (for example, if the area is higly overpopulated by taxis the algorithm postpones itself of 45 minutes for the research of the taxi, while if the area is completely empty it might start the research immediately). When the time arrives, the system searches for a driver in the zone that is available. As reservations have the priority respect to normal requests, it continues to search for the first driver that sets itself as available if there is no one in the queue and matches him with the reservation, telling him the position he has to be in and how much time does he have left.

Chapter 4

User Interface Design

4.1 Mockups

In this section will be presented the mockups of the application. They will include both the views for the Mobile and Web application.

4.1.1 User side application

Here are presented the mokups relative to the user side mobile and web application. Each mockup shows the possible actions that the user can choose.

- In Figures 4.1 and 4.3 there are the reprasantation of what a guest can do when he opens the application respectively in the mobile and web application. He has two choices: login, if he has already made an account, otherwise he can click on the register button and go to the registration page.
- In Figures 4.2 and 4.4 there are the reprasantation of the registration page regarding respectively the mobile and web application. A guest has to insert on all the form inputs and press the “register” button.

- In Figures 4.5 and 4.7 there are the actions that the user can perform in his home page both for the mobile and web application.
- In Figure 4.6 is presented the mockup for the management of the user profile.
- In Figures 4.8 and 4.10 there are the mobile and web mockups of one of the main features of the application, the taxi request. The user has to insert starting and destination addresses, the number of passengers and if he wants, he can check the sharing option.
- In Figures 4.9 and 4.11 there are the mobile and web mockups taxi reservation. The user has to insert starting and destination addresses, the number of passengers and if he wants, he can check the sharing option as in the request, but moreover he has to insert date and time of the ride.
- In Figure 4.12 there is the mobile mockup in which is presented the page where the user can check the position of the taxi and its excpected arrival time.
- In Figures 4.13 and 4.14 are presented the mobile and web mockups for the reservations page, where the user can check his reservations and, in case, delete them.

4.1.2 Driver side application

In this section are presented the mockups relative to the application available for the drivers.

- In Figure 4.15 there is the home page of the application with all the respective actions available for the driver.

- In Figure 4.16 there is the representation of the management of the drivers availability.
- In Figure 4.17 is represented the mockup relative to the confirmation of the rides.

Figure 4.1: Guest home mobile

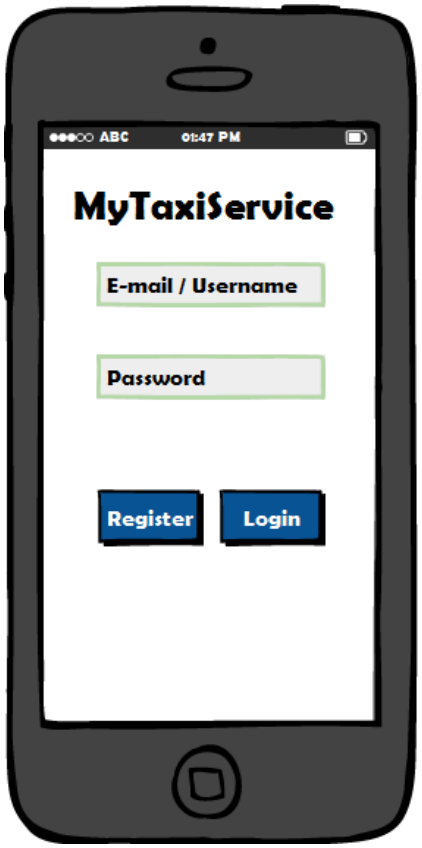


Figure 4.2: Guest registration mobile

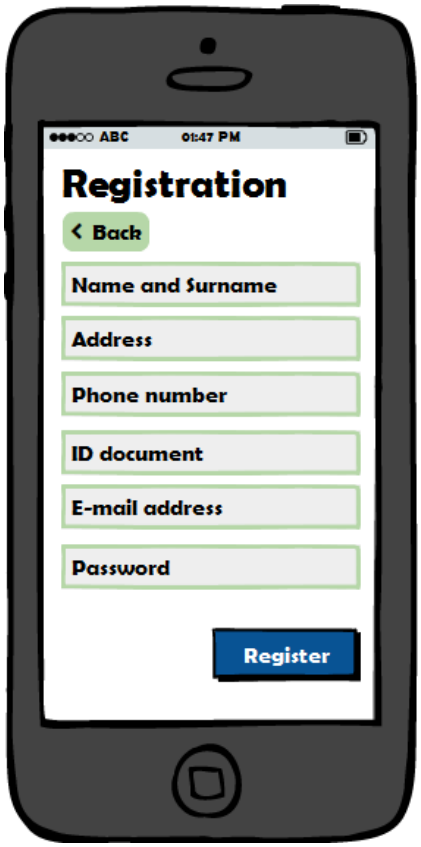


Figure 4.3: Guest home web

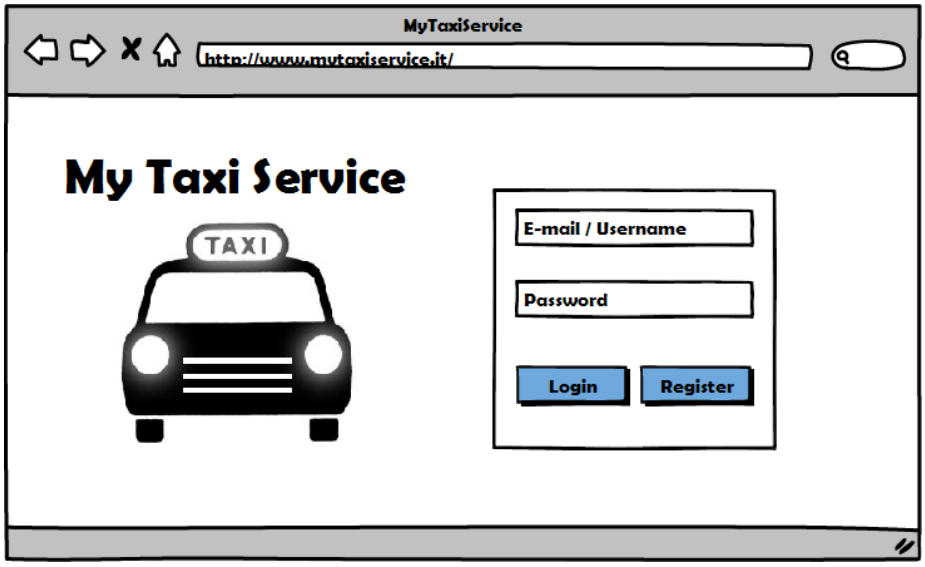


Figure 4.4: Guest registration web

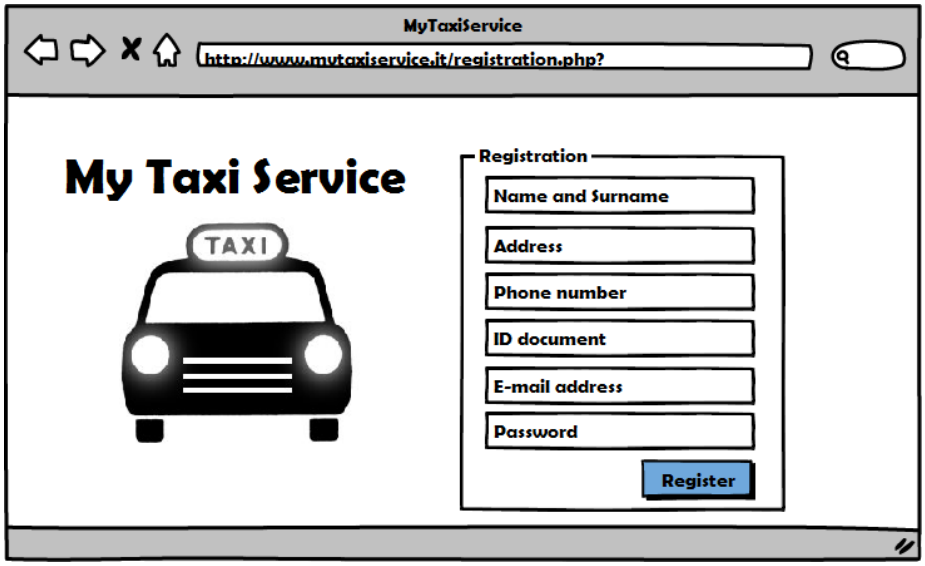


Figure 4.5: User home page mobile

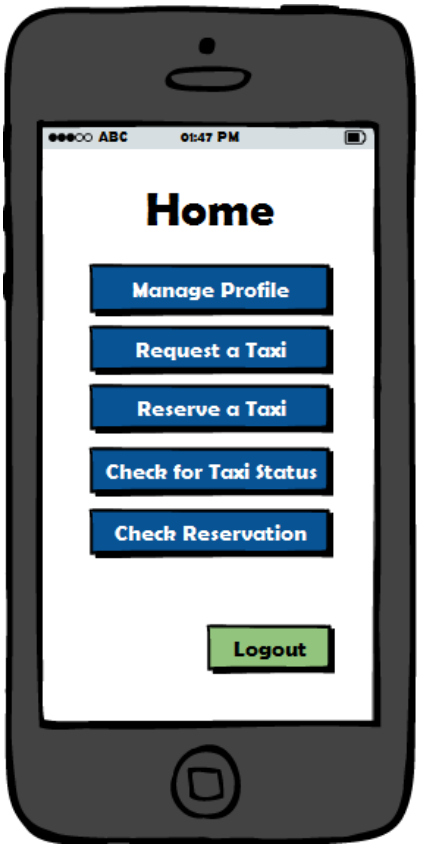


Figure 4.6: Manage profile

Figure 4.7: User home page web

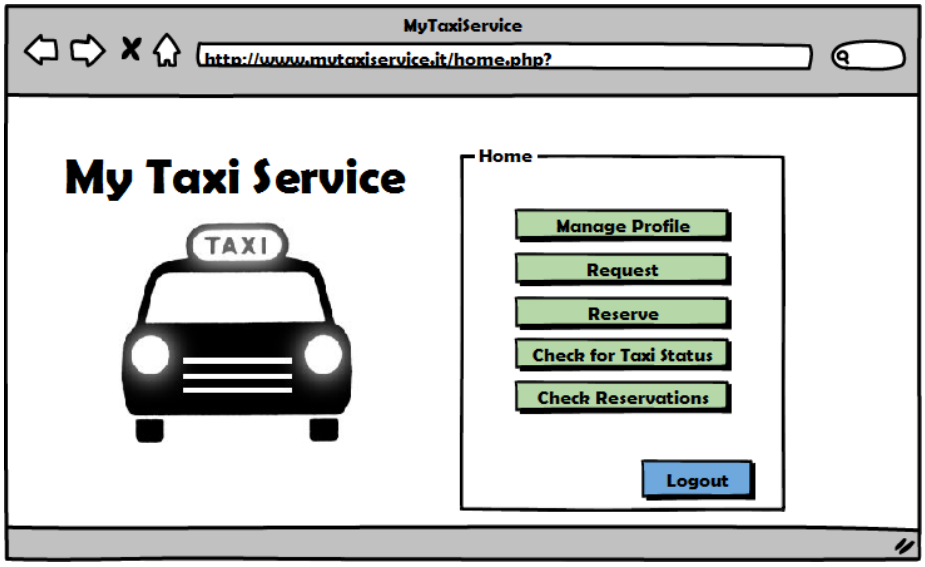


Figure 4.8: Taxi request mobile

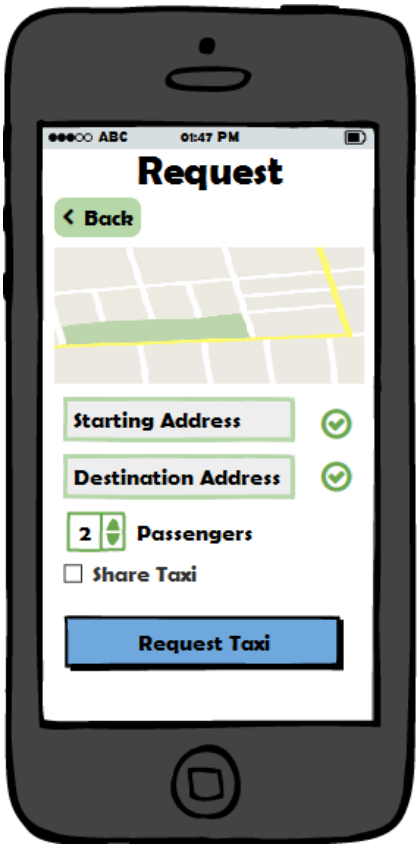


Figure 4.9: Taxi reservation mobile

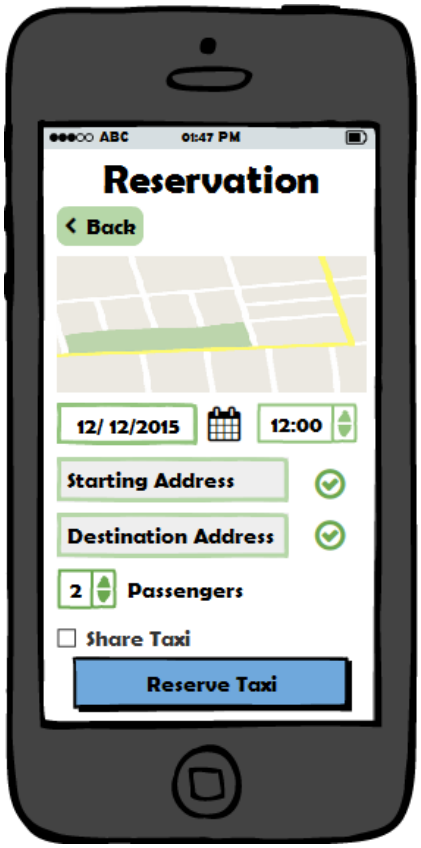


Figure 4.10: Taxi request web

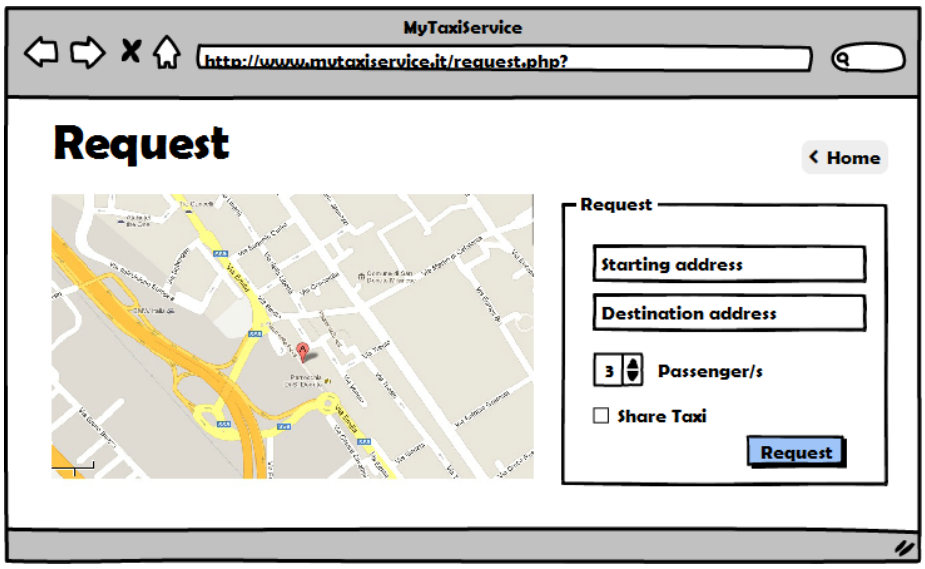


Figure 4.11: Taxi reservation web

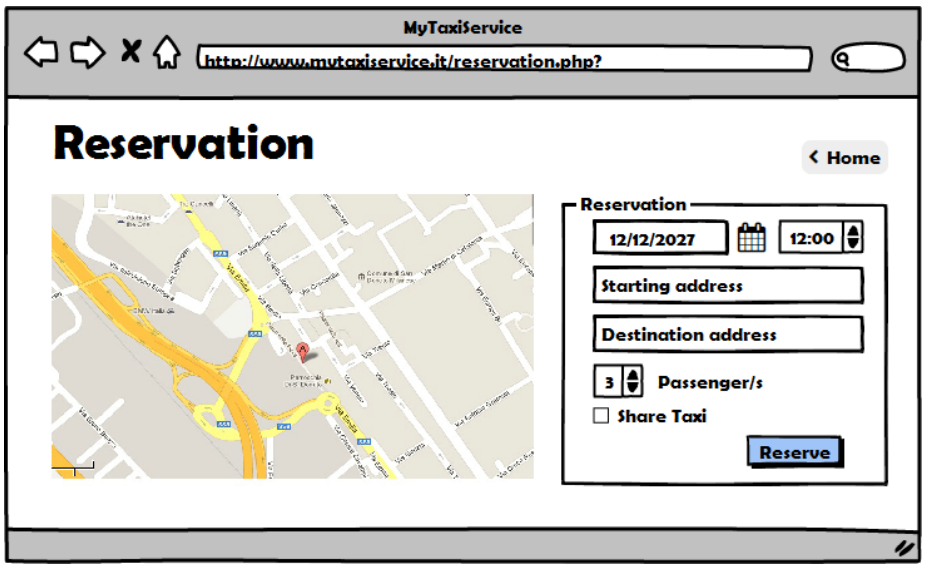


Figure 4.12: Check taxi status mobile

Figure 4.13: Check reservations mobile

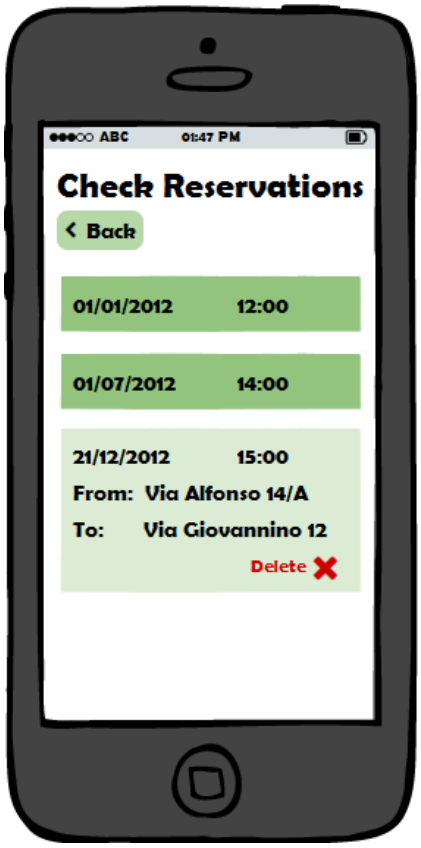


Figure 4.14: Check reservations web

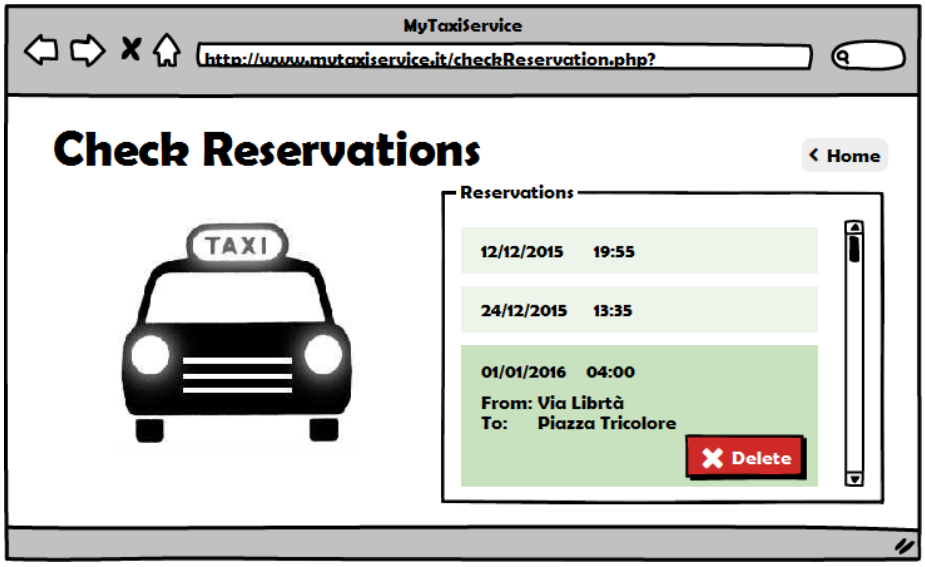
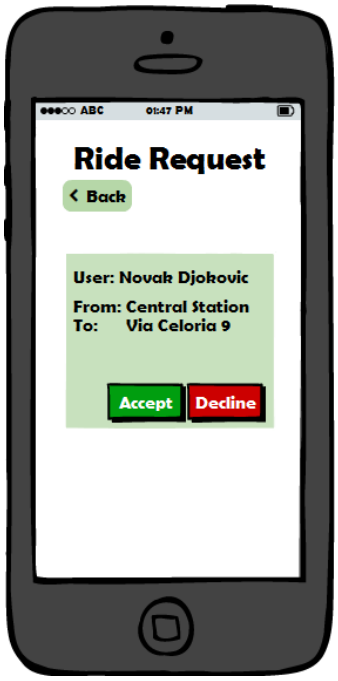


Figure 4.15: Driver home page

Figure 4.16: Availability management

Figure 4.17: Ride confirmation



Chapter 5

Requirement's Traceability

5.1 Registration and Login

These requirements can be retreived in the subsection 2.5.1 of the Runtime view, that explain how the login works, whereas the correctness of the data for registration and login are checked by the Security Manager component (susbection 2.3.3.5 of the Component view).

5.2 User side taxi reuqest, sharing, reservation

These other requirements can be retreived in subsection 2.5.2 and 2.5.4of the Runtime view. Furthermore it can be seen in section 3.4 and 3.4 how does it work in a deeper way for the user side.

5.3 Taxi driver side and notifications

These requirements can be retreived as in the previous section but in addition see the subsection 2.5.3.

5.4 Queue management

These requirements can be retreived in section 3.2 and 3.1, that show how the S2B guarantees the fair management of the queue required.

Chapter 6

Other Information

In this chapter are present all the information relative to the redaction of this document.

6.1 Working hours

Day	Alessandro	Davide	Stefano
19/11/2015	3	3	3
20/11/2015	5	5	3
23/11/2015	2.5	2.5	3.5
25/11/2015	-	3	2
26/11/2015	2.5	-	4
27/11/2015	4	3	-
01/12/2015	4	3	5
02/12/2015	3	4	3
03/12/2015	3	3	3
RASD hours	24	26.5	23
Total hours	51	53	49.5

6.2 Used tools

Following are listed all the main tools used in order to redact this document:

- LaTeX and LyX editor;
- Balsamiq Mockups 3.0;
- starUML: for the sequence diagrams;
- Google Drive;
- Dia: for the E-R schema;
- Microsoft Word;