

Design Document

myTaxiService

AA 2015/2016



POLITECNICO
MILANO 1863

Carlo Broggi

Nicola Ghio

Summary

1 Introduction	4
1.1 Purpose	4
1.2 Scope.....	4
1.3 Definitions, Acronyms, Abbreviations	4
1.4 Reference Documents	4
1.5 Document Structure	5
2 Architectural Design.....	6
2.1 Overview	6
2.2 High level components and their interaction.....	6
2.3 Component View.....	7
2.4 Deployment view	9
2.5 Runtime view	10
2.6 Component interfaces.....	11
2.7 Selected architectural styles and patterns	12
3 Algorithm Design	13
3.1 Request a Taxi.....	14
3.2 Reserve a Taxi	16
3.3 Queuing.....	17
3.3.1 Accept	19
3.3.2 Refuse	21
3.4 Retrieving a driver outside the queue	22
4. User Interface Design.....	23
4.1 Customer GUI.....	23
4.1.1 Login	24
4.1.2 Register	25
4.1.3 Main page.....	26
4.1.4 Ask For a Taxi	27
4.1.5 Notification	28
4.1.6 Profile.....	29
4.2 Driver GUI	30
4.2.1 Login	30
4.2.2 Main page.....	31
4.2.3 Notification	32

4.3 Administrator GUI	33
4.3.1 Login	33
4.3.2 View Driver.....	34
4.3.3 Search Driver.....	35
4.3.4 Add Driver	36
5 Requirements Traceability	37
5.1 Functional Requirements	37
5.2 Non functional requirements	37
6 References.....	39

1 Introduction

1.1 Purpose

This document a second step into the developing of myTaxiService system, it directly follows the RASD document, which has been previously provided and approved.

The design document is intended to be a guide for developers during the implementation of myTaxiService system. We are going to show the architecture of this system by providing a high/medium level view of the system-to-be and we will to show how it is meant to be built.

Since this document it's not meant to show the effective implementation of the software, we are not going to give specific details about the protocols with which each component interact with the system neither examples of possible implementation of the code.

1.2 Scope

As already explained in the RASD, we want to build a system able to provide both a fair distribution of the work among the taxi driver located in the city both a quick and reliable service for all the customers. We want provide an intuitive way of interaction for the users and also to give all the information they need as soon as they need it. We will also show how the communication between customer and driver will happen along with how (and when) the information will be stored into the database.

1.3 Definitions, Acronyms, Abbreviations

Here is a list of all the terms we will use into this document

- *RASD: Requirement Analysis Specification Document*
- *All the terms already included in the RASD*
- *Tier logical level of the architecture.*

1.4 Reference Documents

Within this document we will refer many times to the RASD so, since the Design Document is a further step into the developing of a system, it is suggested to read the RASD in order to completely understand this document Document.

We also referred to the IEEE standard for Design Document for the structure of this document.

1.5 Document Structure

This design document is structured as follows:

- *An architectural part in which we will show and explain our main architectural decisions.*
- *A list of the critical algorithms that will take place in our system*
- *Mockups of how we mean the User Interface to be*
- *An explanation of how our architectural choices respect the requirements described in the RASD documentation.*

2 Architectural Design

2.1 Overview

In this chapter, as previously mentioned, we are going to show how the architecture of our system will be built. We will start from a high level view and we will go deep into details in order to show the logic components that characterize our system, where they should be placed in the hardware and how they will interact with each other at runtime.

More precisely, the following topics will be discussed.

- High level architectural view
- Component view
- Deployment view
- Runtime view
- Component interfaces
- Architectural styles and patterns

2.2 High level components and their interaction

A high level view of the architecture of our system is shown in fig 2.1

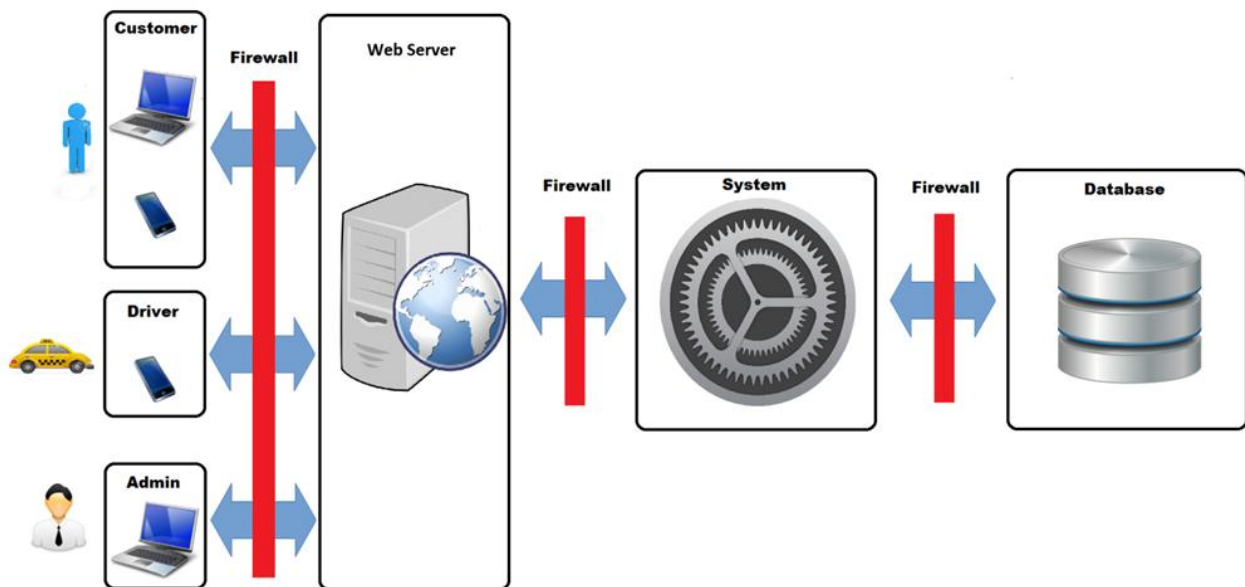


Fig 2.1

Our architecture is composed by the User Side and by a 3 Tier configuration:

User Side: the leftmost one, it refers to part of the software in direct contact with the user. Since Each device (mobile or PC) interacts with the system through the internet, this Tier is composed by two layers

- The Client layer i.e. the client application (for mobile) and the web browser (for the access through web page)
- The Web layer that is made by all the software components that will receive requests from the User, will send them to the system and vice versa.

1. **Web Tier:** it is made by a web server, it receives messages from all the users, forward them to the system and vice versa.
2. **Application Tier:** this is the real core of the system. It is made of all the components that manage the request made by the users and allow a correct communication between them.
3. **Data Tier:** The hardware where all the information are permanently stored. It is separated by the system and it is connected with him via LAN (for example with a TCP/IP communication protocol)

2.3 Component View

We decided to divide our system in various components in order for it to be as much cohesive as possible. Each is independent, has been created for a specific task so it has a unique set of functionalities and is able to communicate only with a few other specific ones.

A component diagram is shown in Fig 2.2

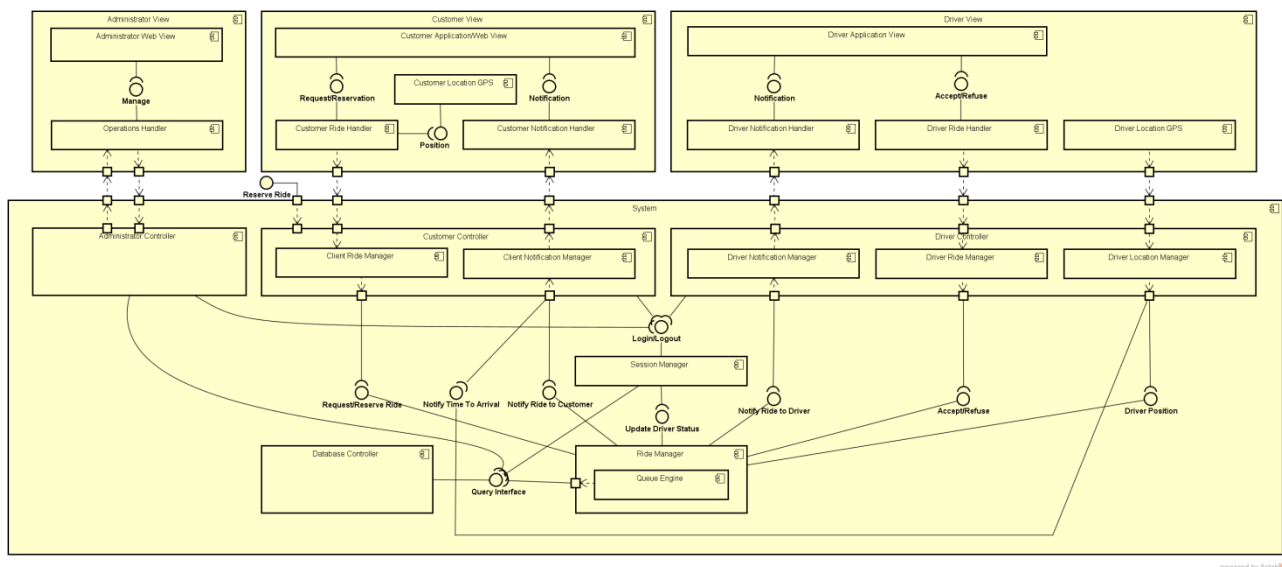


Fig 2.2

The component diagram shows that the software is divided in components and sub components, for every one of them we are going to give a brief explanation.

1. **Customer View:** this macro component contains all the sub-components that interact directly with the Customer such as:
 - **Customer Application View:** it is the part of the system that generate the graphic part and manages the events generated by the Customer.
 - **Customer Ride Handler:** the party of the system that generate the effective request/reservation of a taxi, it is in charge of sending all the necessary information to the Customer Controller.

- **Customer notification Handler:** this part of the system is in charge of receiving all the information from the Customer Controller, transform them into a notification and to send them to the Customer Application View so that the customer will be able to see them.
 - **Customer Location GPS:** this part of the system is activated when a customer request a taxi through mobile application and does not insert the adress of the meeting point, it will provide the actual position of the customer to the Ride Handler that will communicate it to the Customer Controller.
2. **Driver View:** this macro component contains all the sub-components that interact directly with the Driver such as:
- **Driver Application View:** it is the part of the system that generate the graphic part and manages the events generated by the Driver.
 - **Driver Notification Handler:** this part of the system is in charge of receiving all the information from the DriverController, transform them into a notification and to send them to the Customer Application View so that the driver will be able to see them
 - **Driver Ride Handler:** this is the part of the system that allows the driver to queue in a certain zone.
 - **Driver Location GPS:** this part of the system communicates the actual position of the taxi that will be automatically
3. **Administrator View:** this macro component contains all the sub-components that interact directly with the Administrator such as:
- **Administrator Web View:** it is the part of the system that generate the graphic part and manages the events generated by the Administrator.
 - **Operations Handler:** it is the part of the system that allows the Administrator to interrogate the database in order to perform read/write operations on it (the ones he is allowed to)
4. **System:** this is the core of myTaxiService, it manages all the Users and the interaction between them while storing and retrieving data from the Database. It is made by this macro-components.
- **Administrator Controller:** this part of the system connects the Administrator with the database. It is in charge of controlling the credentials used to log in by querying the Session Manager and to send the query made by the Administrator to the Database Controller.
 - **Customer Controller:** this part of the system is in charge of elaborating all the messages coming from all the Customers and sending them to the inner parts of the system and viceversa. It is composed by:
 - **Client Ride Manager:** it is in charge of receiving all the request/reservation from the customers and sends the information to the Ride Manager.
 - **Client Notification Manager:** it is in charge of receiving messages from the Ride Manager that contain all the information that has to be provided to the customer and to send it to him.
 - **Driver Controller:** this part of the system is in charge of elaborating all the messages coming from all the Drivers and sending them to the inner parts of the system and vice versa. It is composed by:
 - **Driver Ride Manager:** it is in charge of receiving from all drivers the queue requests, their decisions about accepting/refusing a ride and sends them to the Ride manager.
 - **Driver Notification Manager:** it is in charge of receiving messages from the Ride Manager that contains all the request notifications and dispatches them to the single Drivers.
 - **Driver Location Manager:** it is in charge of periodically querying each Driver Location GPS in order to check whether a taxi is still in a zone or not, if not it the Ride Manager will be notified.
 - **Session Manager:** it is in charge of keeping track of the connected Users and their status. If a user accidentally disconnects from the system and reconnects after a while the Session Manager will recover the previous information.

- **Ride Manager:** this part of the system is in charge of assigning a driver to a pending request. and to keep.
 - **Queue Engine:** this part of the system is in charge of keeping track of the taxi queues in the various zones on the city.
- **Database Controller:** this part of the system is the one that directly communicates with the Database. It is the one that can load/store data on the Database.

2.4 Deployment view

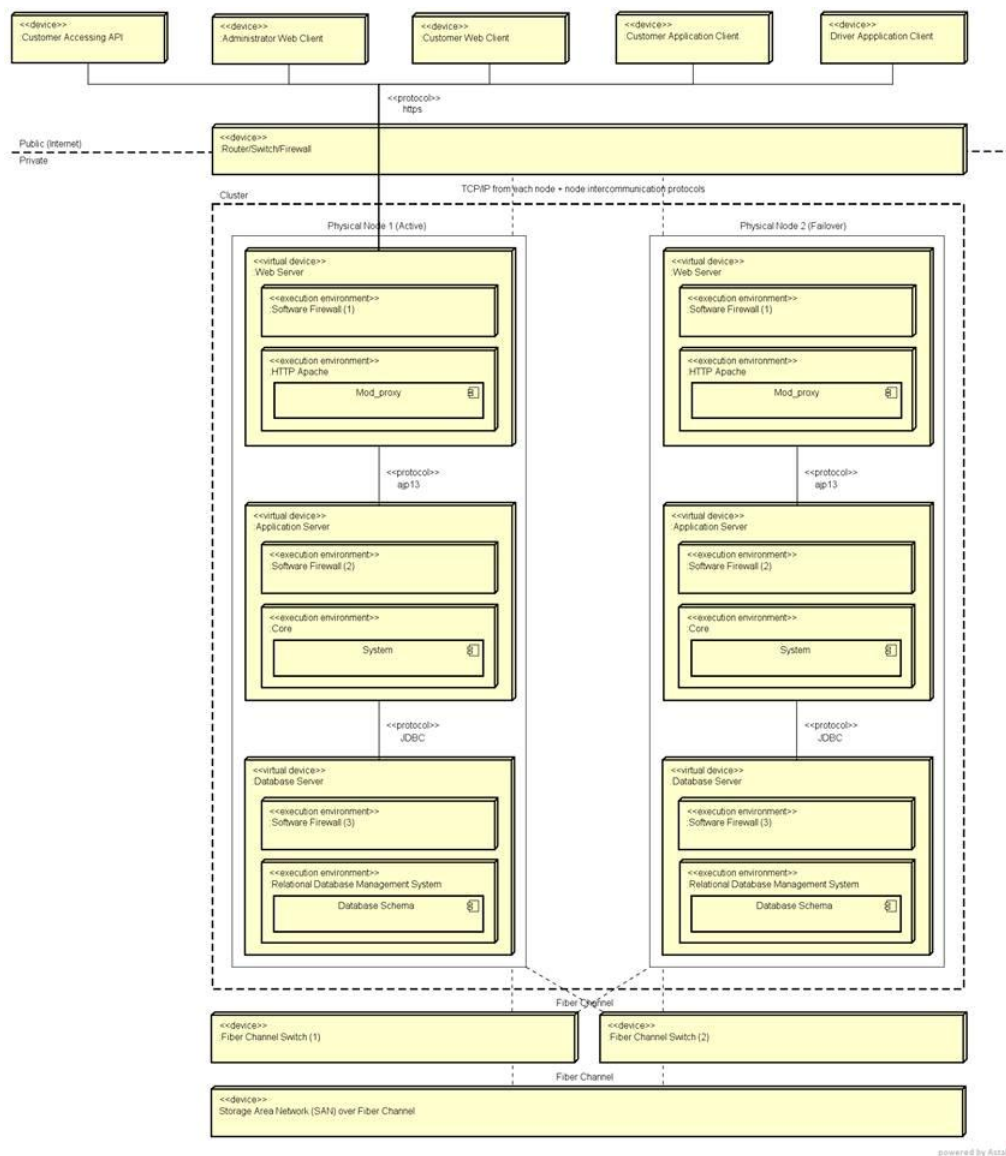


Fig 2.3

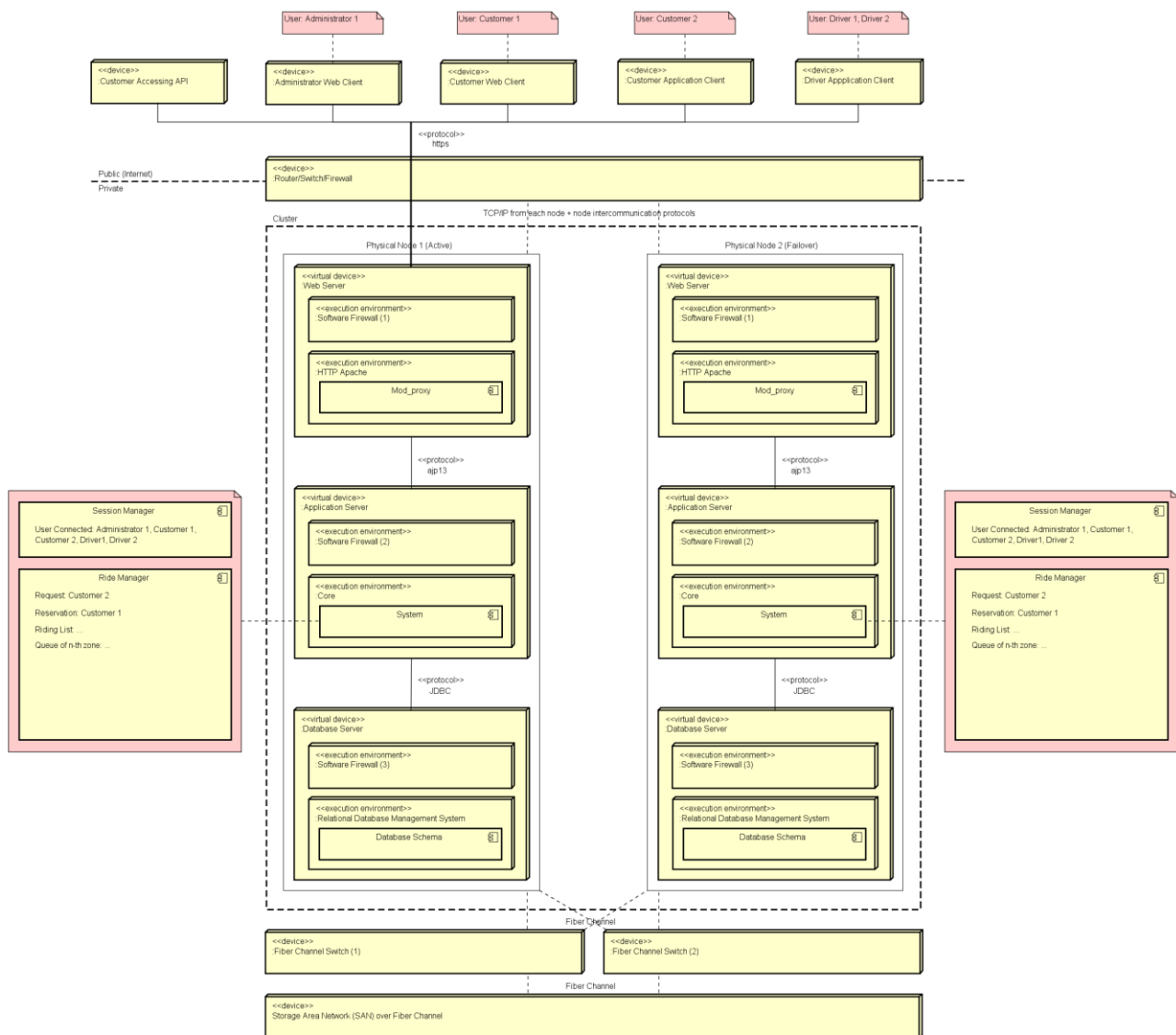
In this component diagram we can see that two physical node that are clustered. Every node runs a bare-metal hypervisor that let us run different virtual machines on each node.

The nodes are used in this way: the first is the active node and the second the failover one.

If the active node fails, the failover node continues working without downtime, because in this node there is a shadow copy of the virtual machines running in the active node. We can also add other failover nodes in order to increase redundancy in our system.

Every node shares the secondary storage linked to a SAN (Storage area network) thanks to a Fiber Channel link (we use 2 switches in order to make redundant the path between the SAN and the nodes). The SAN uses also redundant Fiber Channel interfaces and a RAID system to keep the data safe. The two nodes then can communicate between them and also with the internet by attaching them to a router/switch. Every node has the same virtual machine running (synced by the ethernet link through the router/switch). On each node there are three virtual machines: one with the DMBS, one with the Application Core and the other with the web server that allows to communicate outside permitting the remote app to run correctly and the correct working of the website.

2.5 Runtime view



powered by Astah

2.6 Component interfaces

Here you define the interfaces of your components, that is, which operations they offer to the external world, their meaning, any input and output parameter (name, possible set of values/type)

In this section we try to explained how each component communicate with the others by describing the interfaces they offer and their meaning:

- **Administrator View:** it communicates with **Administrator controller** with `sendRequest` that will contain an HTTP message while he will accept responses with `getNotification`.
- **Customer View:** it communicates with **Customer Controller** controller with `sendRequest` that will contain an HTTP message while he will accept responses with `getNotification`, as the Administrator.
- **Driver View:** it communicates with **Driver Controller** with `sendRequest` that will contain an HTTP message while he will accept responses with `getNotification` but will also be able to communicate his GPS location through `sendGPSLocation` that will also contain an HTTP object. Moreover, through `getInformation` it could request information about the queue so that the driver will be able to see his position in the queue and the queues around the city.
- **Administrator Controller:** it will communicate with the **Session manager** with `LoginUser` and `LogoutUser`, that will accept as input the credentials of a User and `getUserState` that will return the information about the Session of the administrator if the connection fails. It will also communicate with the **Database Controller** with a query in order to read/write data on the DB.
- **Customer Controller:** it will communicate with the **Session manager** with `LoginUser` and `LogoutUser`, that will accept as input the credentials of a User and `getUserState` that will return the information about the Session of the driver if the connection fails. It will communicate with **Ride Manager** sending a ride request with `sendTaxiRequest` that will contain the user information and will return a boolean to see if the request has been received correctly by the system, this functioning is similar to another function such as `sendTaxiReservation` that will also contain the GPS coordinates and the time of the meeting. It will communicate with the **Customer View** with `sendNotification` that will contain all the information that needs to be provided to the Customer.
- **Driver Controller:** it will communicate with the **Session manager** with `LoginUser` and `LogoutUser`, that will accept as input the credentials of a User and `getUserState` that will return the information about the Session of the driver if the connection fails. It will communicate with **Ride Manager** sending a queue request with `sendQueueRequest` that will contain the driver information (ID, current zone and so on) and will return a boolean to see if the request has been received correctly by the system. It will also communicate driver's decision through `sendDriverDecision` such as if he refused or not a ride. It will communicate with the **Driver View** with `sendNotification` that will contain all the information that needs to be provided to the driver while periodically sending a `getGPSPosition` request that will return the actual position of the driver to check if he got out of the zone or not.
- **Session Manager:** it communicates with the **DB Manager** with a query by `getUserInformation` in order to retrieve User Credentials.
- **Ride Manager:** communicates through `updateUserStatus` with the **Session Manager** in order to update drivers and customers status (such as if they are in a ride or not and so on...). It communicates with **Customer Controller** with `sendRideInformation` that will contain the estimated time of arrival and the Taxi ID once his request has been accepted. It will communicate with **Driver Controller** through `sendRideNotification` that will notify the first driver in the queue, retrieved by requesting it to the **Queue Manager** through `getFirstDriver` that accepts as parameter a zone, that there is a new

pending request. It communicates also with **Database Controller** with a `storeInformation` and `getInformation` which explanation is trivial.

- **Database Controller:** it offers the previously described interfaces.

2.7 Selected architectural styles and patterns

1. **Three Tier Architecture:** as easily visible from Fig 2.1 our main architectural choice is a 3 tier architecture which components have been already described in 2.2. This brings to us the following advantages:
 - **Locations:** We divided our system in three different devices that can be placed in different locations if needed.
 - **Scalability:** Since each tier can scale horizontally we can augment the number of machines in order to sustain an incrementing amount of load if needed.
 - **Availability:** If one of the device goes down it can be replaced with another one so the system will not crash.
 - **Security:** as shown in fig 2.1 our system is protected from malicious attacks thanks to the utilization of firewalls that are placed between each tier. This mean that sensible data will be difficult to access from unauthorized users.
2. **Client-Server:** The other main architectural choice is to implement a Client-Server system. This brings us the following advantages:
 - **Centralisation:** in this architecture there is a centralized control. Servers help in administering the whole set-up. Access rights and resource allocation is not in charge of the Client.
 - **Proper Management:** All the files are stored at the same place. In this way, management of files and their recovery becomes easier.
 - **Back-up:** as all the data is stored on server its easy to make a back-up of it. Also, in case of some break-down if data is lost, it can be recovered easily and efficiently
 - **Accessibility:** from various platforms in the network, server can be accessed remotely.
 - **Memory:** as new information is uploaded in database, each device need not have its own storage capacities increased. All the changes are made only in central computer on which server database exists.
3. **SOA (Software Oriented Architecture):** This architectural styles brings us the following advantages:
 - **Indepedency:** each component is independent and they communicate by offering interfaces one to the other. This will allow to change the internal behavior of a component without compromising the entire system.
 - **API:** the API that are exposed by the system can be seen as a separated set of services offered to the external. Also, this style gives us also the possibility to increment the number of the offered services by developing other functionalities.
 - **Information flow:** this style grants an organized flow between the component of our system, diminishing the number of messages sent and saving cost.
 - **Mediator:** this is the pattern that we used to plan all our system. Basically there is a central component, indicated as System in the component diagram, that puts in communication the components of the Application Tier. In this way we can increase the cohesion of our system lowering the number of messages sent.

3 Algorithm Design

In this chapter we are going to show the fundamental algorithms that will take place during the execution of our system. We decided to represent the principal logical components of our system and to show how every algorithm will involve each one of them.

The following algorithms will be explained:

- *Customer request a taxi*
- *Driver's Queue*
 - *Drivers accepts ride*
 - *Customer is not present at a ride*
 - *Drivers refuses ride*
- *Customer reserves a taxi*
- *No driver is present in the zone when a request arrives*

3.1 Request a Taxi

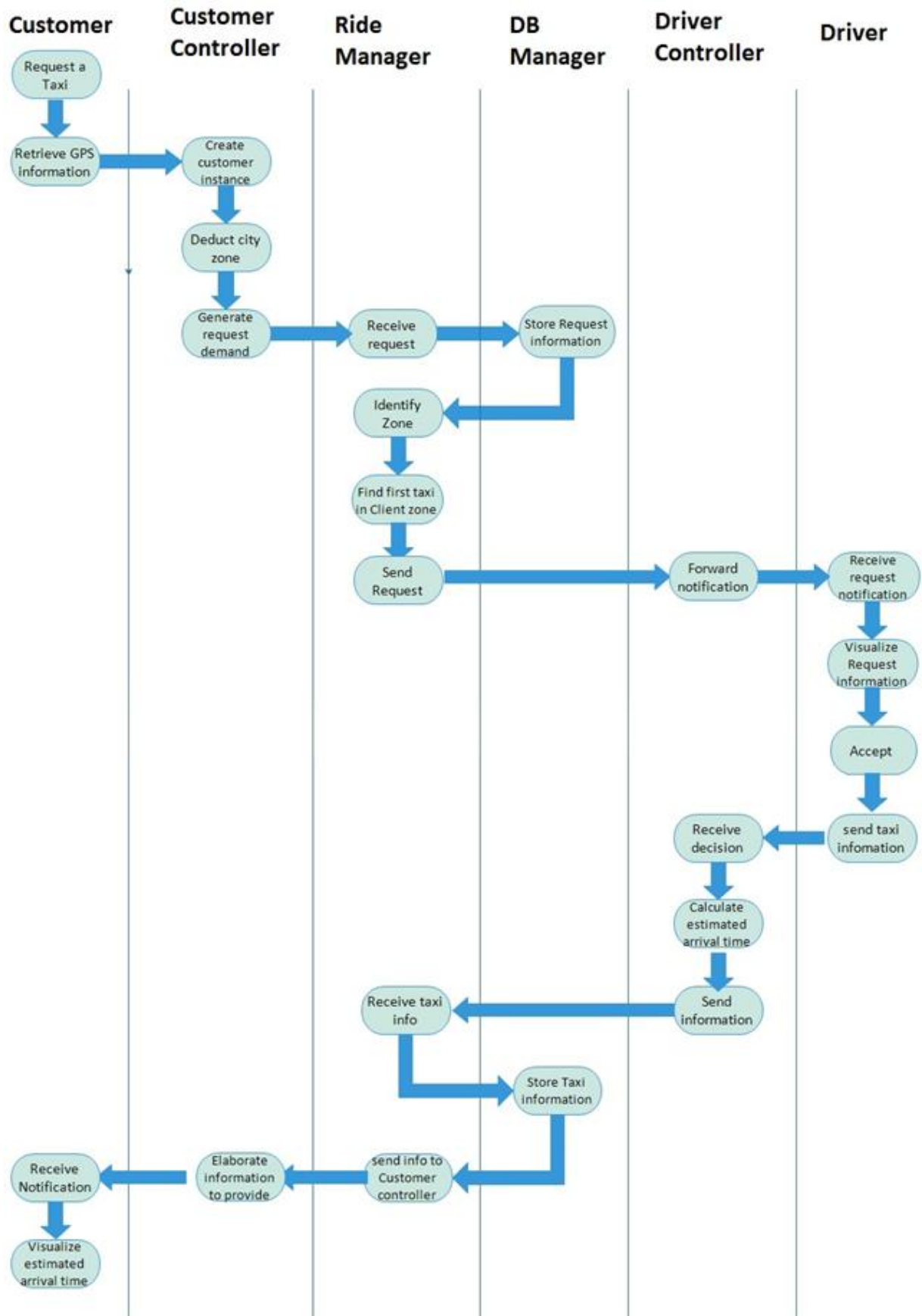


Fig 5.1

Fig 5.1 shows the flow of events that occurs when a customer requests a taxi. For complexity reasons and in order to underline the request we supposed that the customer was already logged in and that the first driver accepts the request.

For completeness, we are now going to explain briefly what happens after the customer gets on the taxi and they arrive at the designated location:

- 4. As soon as the customer pays, the driver confirms that the ride ended via mobile app.*
- 5. Ride details (Customer, driver, route, cost) are saved into the database.*
- 6. Driver Controller allows the driver to queue again.*

As shown in the picture, data are saved in the Database only when there is a confirmation by one of the User (Customer and Driver) that they will participate to the request and when the ride is concluded. The former happens because of security reasons (for example, if a person disappears it could be useful to know what his last location was) while the latter is done mainly because of business motivations.

3.2 Reserve a Taxi

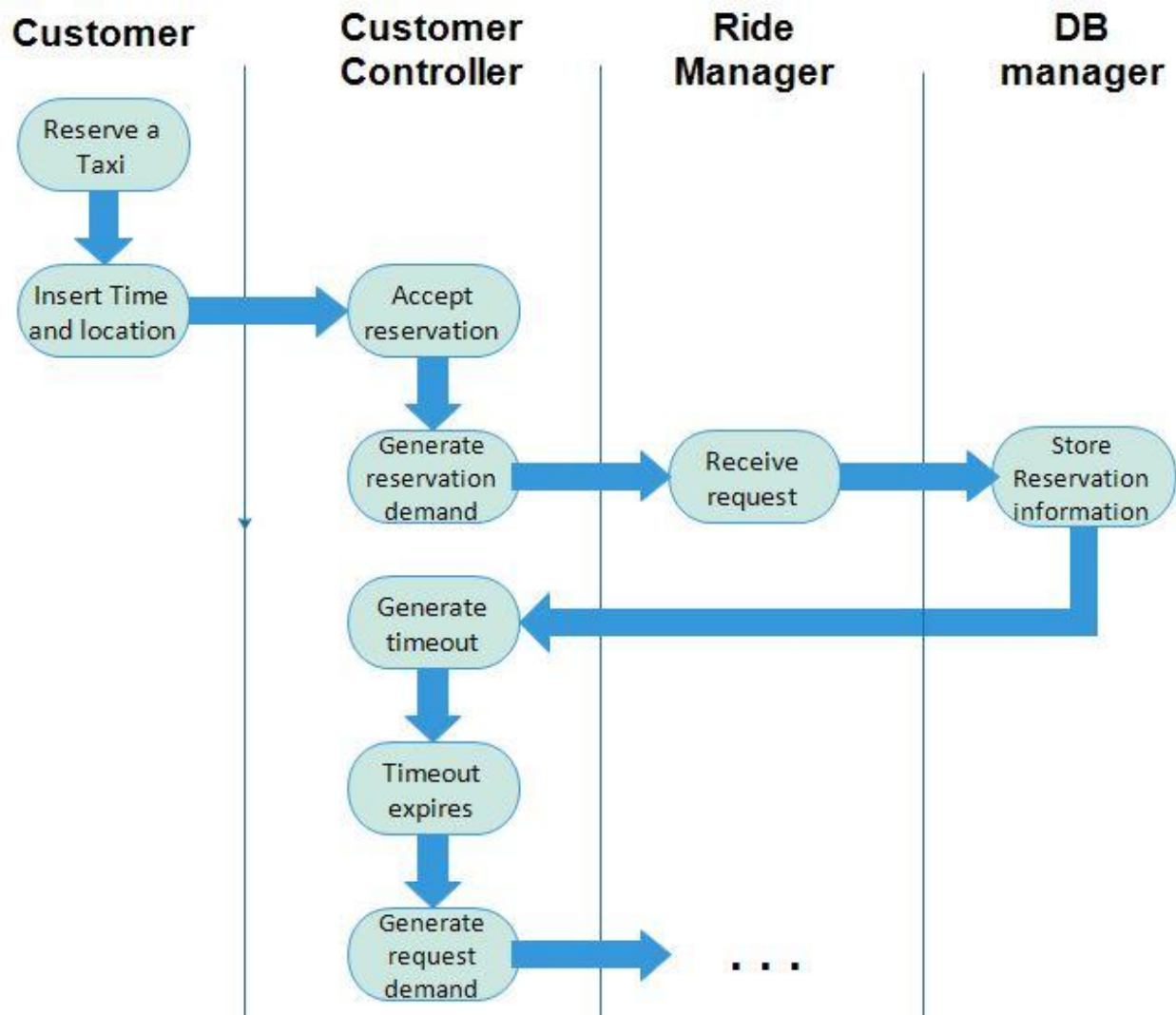


Fig 5.2

Fig 3.1 shows the flow of events that occurs when a customer reserves a taxi.

Basically, the reservation is treated just like a request but it is generated at a specific time before the meeting time. In this way, since we supposed that there are enough taxis to cover all requests, there is no pre-assigned driver. When the Customer controller generates the request, it will communicate to Ride Manager that this request corresponds to the previously made reservations and so the driver who will accept the Ride will be notified that he will have to wait for the customer for 10 minutes.

Please note that the reservation request is stored in the Database.

3.3 Queuing

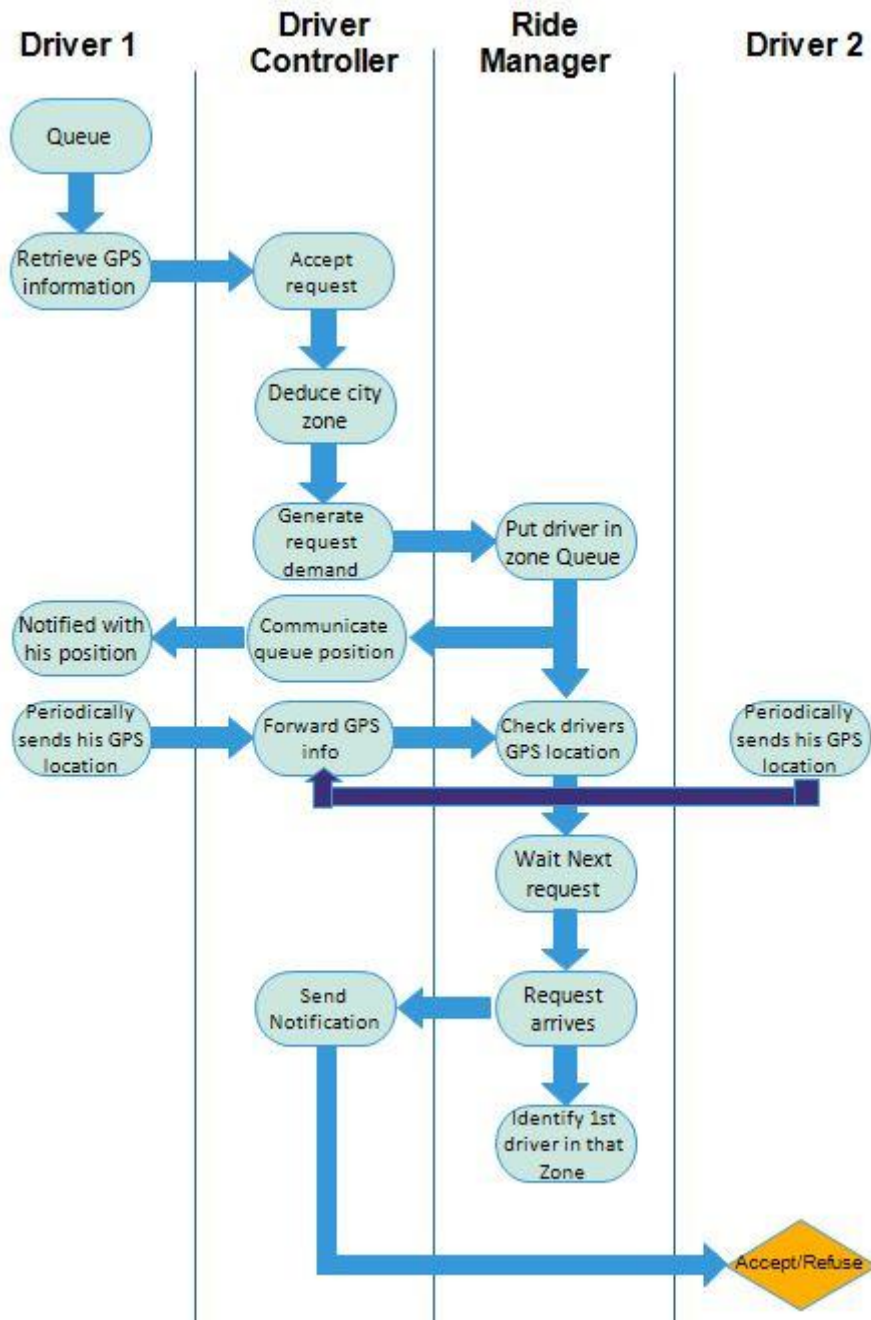


Fig 3.2

Figure 3.2 shows what happens when a driver decides to queue in a certain zone.

In this situation we supposed to have an additional driver, named Driver 2, that is the first driver of that queue. As soon as a request arrives to the Ride manager, the first driver is notified and he can decide to Accept or to Refuse, the explanation of what happens after depends on what Driver 2 decides.

Please note that, since only taxis that are in a certain zone can queue for that specific zone, the position of each driver is updated automatically and periodically. If the Ride Manager finds out that a driver has exited the zone he was queuing in, it will remove it from the queue.

3.3.1 Accept

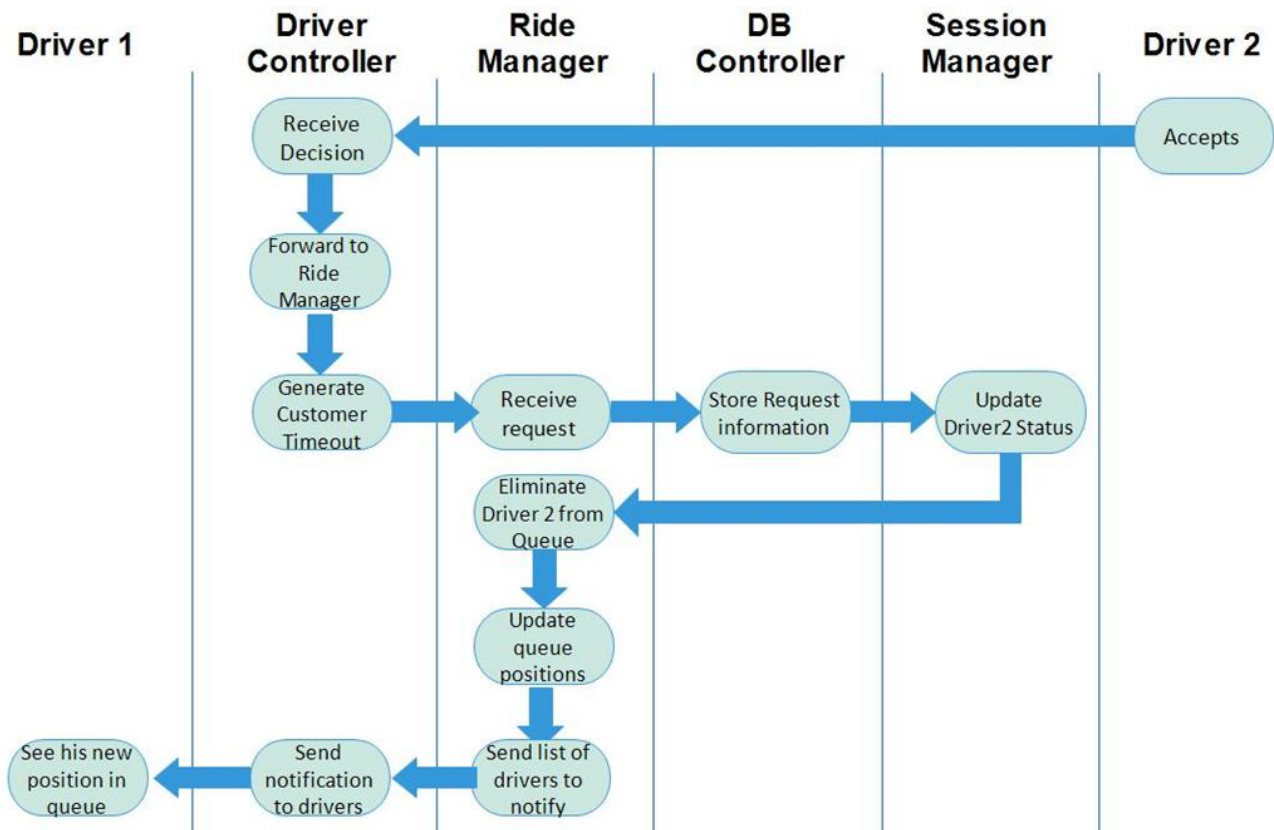


Fig 5.3

Fig 3.3 shows the flow of events that takes place if Driver 2 accepts.

As you can see, as soon as the driver accepts, three main things happen:

- A timeout is started so that if a customer does not arrive to the location point the driver can fee him
- his status is updated so that he can not queue for a ride until he finishes the one he is assigned to.
- Queue's positions are updated and every driver in that queue, for example Driver 1, receives a notification that communicate his new position in the queue.

3.3.1.1 Customer does not arrive in time

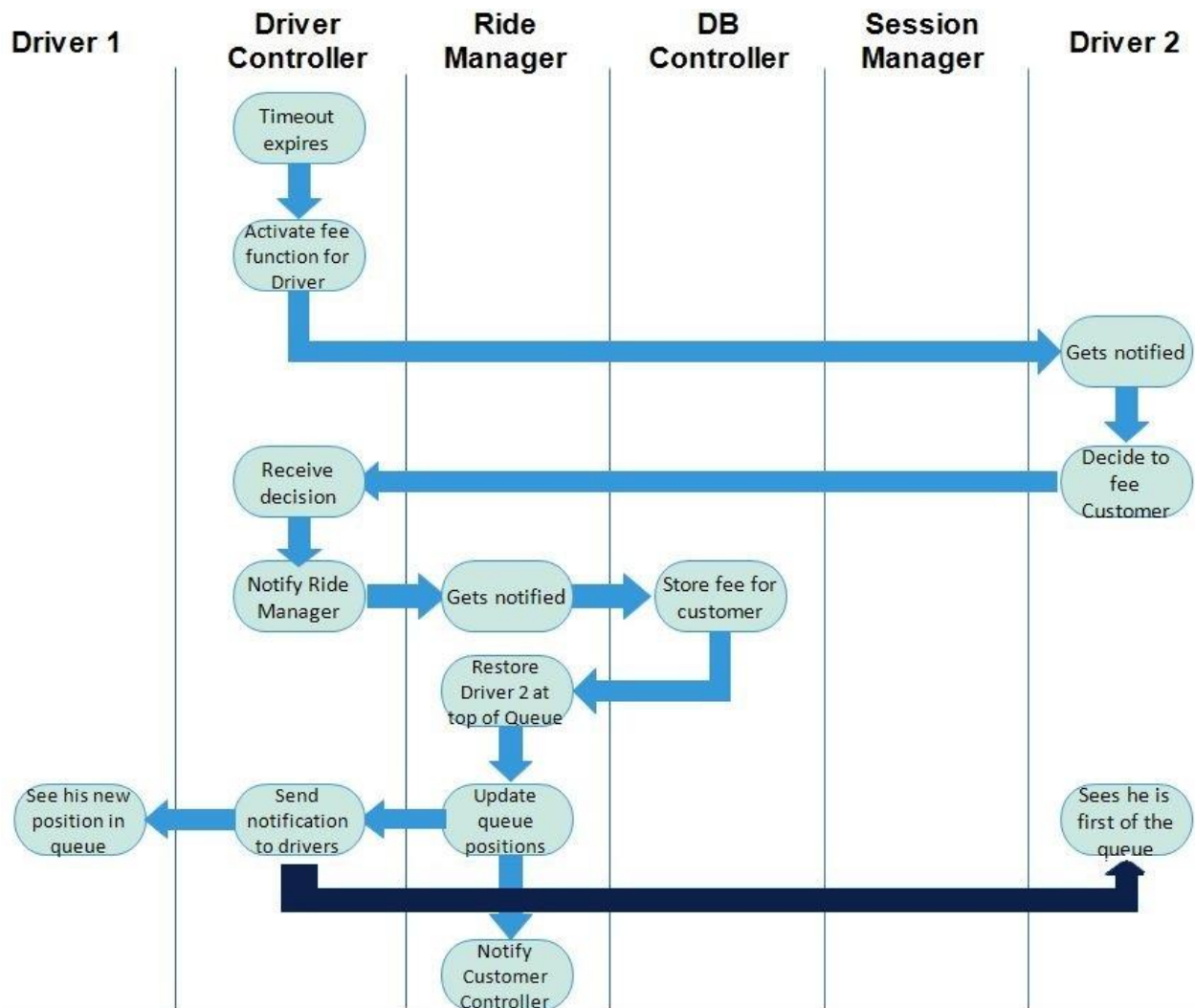


Fig 5.4

Fig 5.4 shows the flow of events that takes place if Driver 2 accepts but the customer does not arrive.

As you can see, as soon as the timeout expires, three main things happen:

- Driver is notified that he can now apply a fee to the customer
- If he does it he come back in the queue again at the first positions
- The position of the other drivers is shifted back by 1

3.3.2 Refuse

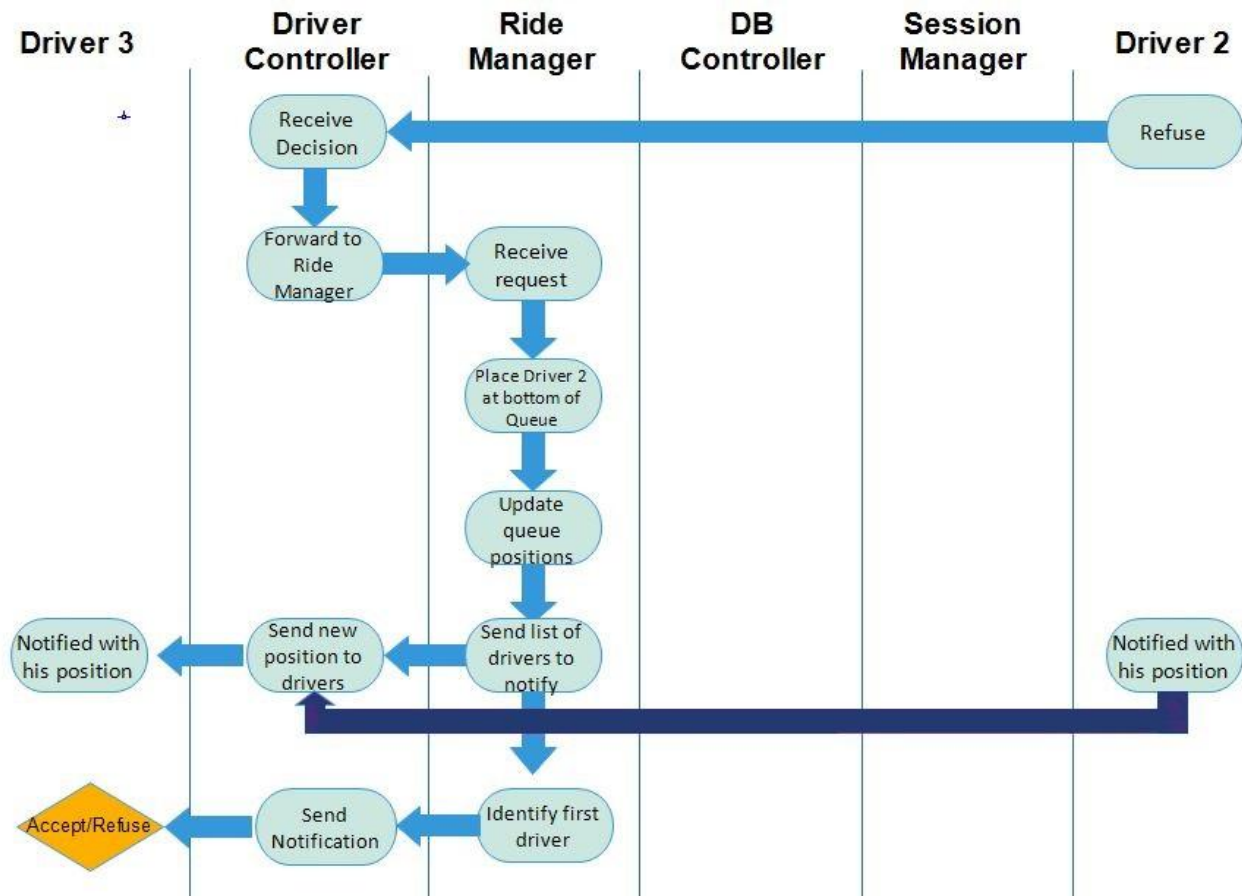


Fig 5.5

Fig 5.5 shows the flow of events that takes place if Driver 2 refuses.

As you can see, as soon as Driver 2 refuses, three things happen:

- Driver 2 is sent to the bottom of the queue
- Queue's positions are updated and drivers notified
- Driver 3, who now is the first one in the queue receives the notification for the same request Driver 2 refused.

3.4 Retrieving a driver outside the queue

When a request rises into a zone with no driver the following thing happen:

1. Ride Manager identifies the first driver in one of the surroundings zones, this can be done for example giving a number as a name for the zones and by sorting them in ascending order.
2. First Driver is notified with this request, he is also notified that this request does not come from his area.
3. He can Accept or refuse, if he refuses there will be no penalty for him.
4. If Driver refuses the systems asks for the first driver of another area and so on
5. If no one of there drivers accepts the search expands to the driver in the second position of the queue of the surrounding areas.
6. This system keeps going until a driver accepts or until a driver queue for the zone in which the request started.

4. User Interface Design

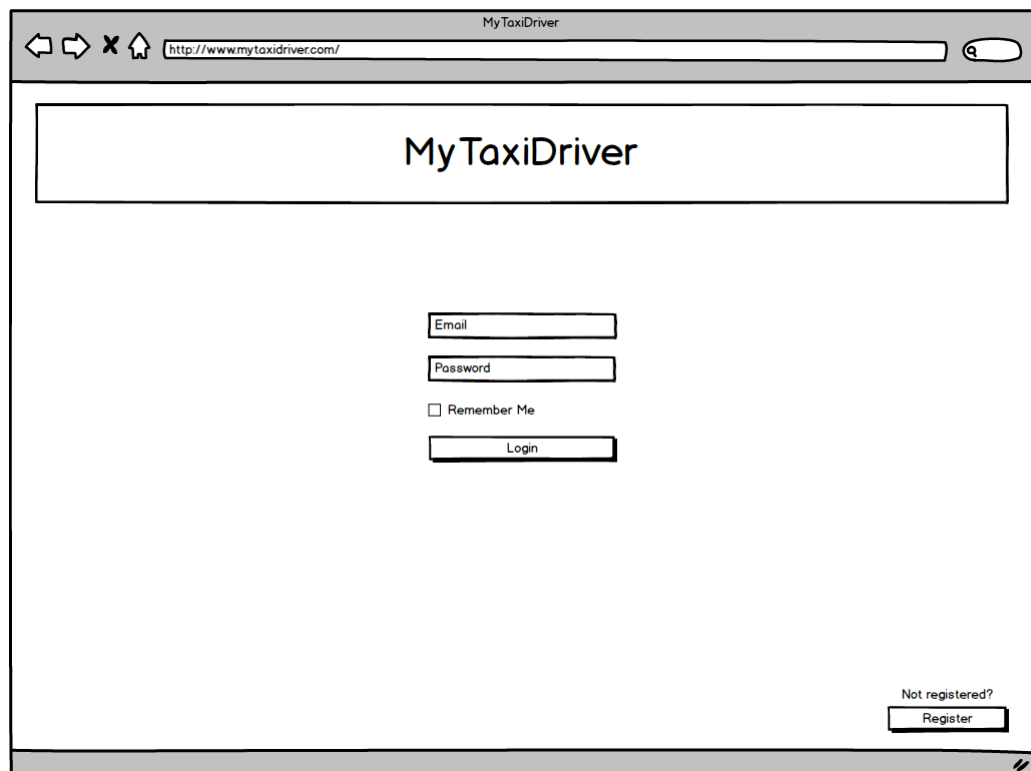
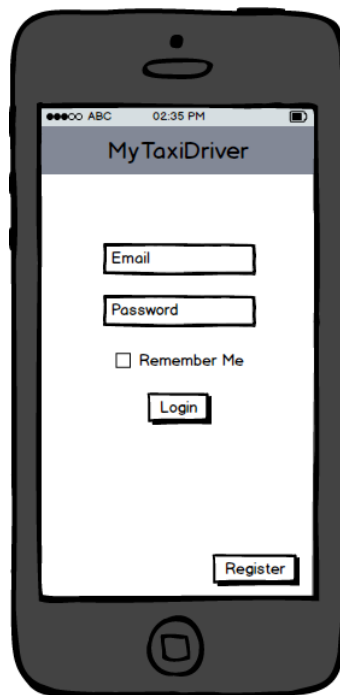
In this chapter we are going to show some mockups of the user interfaces. These will not look exactly like the final GUI but they can be useful to give an idea on how our system will look like.

4.1 Customer GUI

In this section we will give a preview of the customer interface:

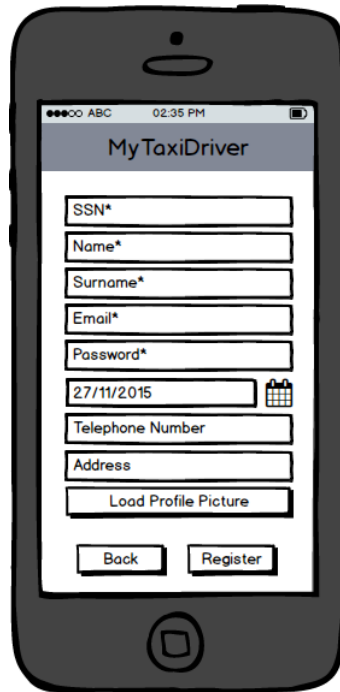
4.1.1 Login

This is how the login page will look like, both on mobile app and web browser:

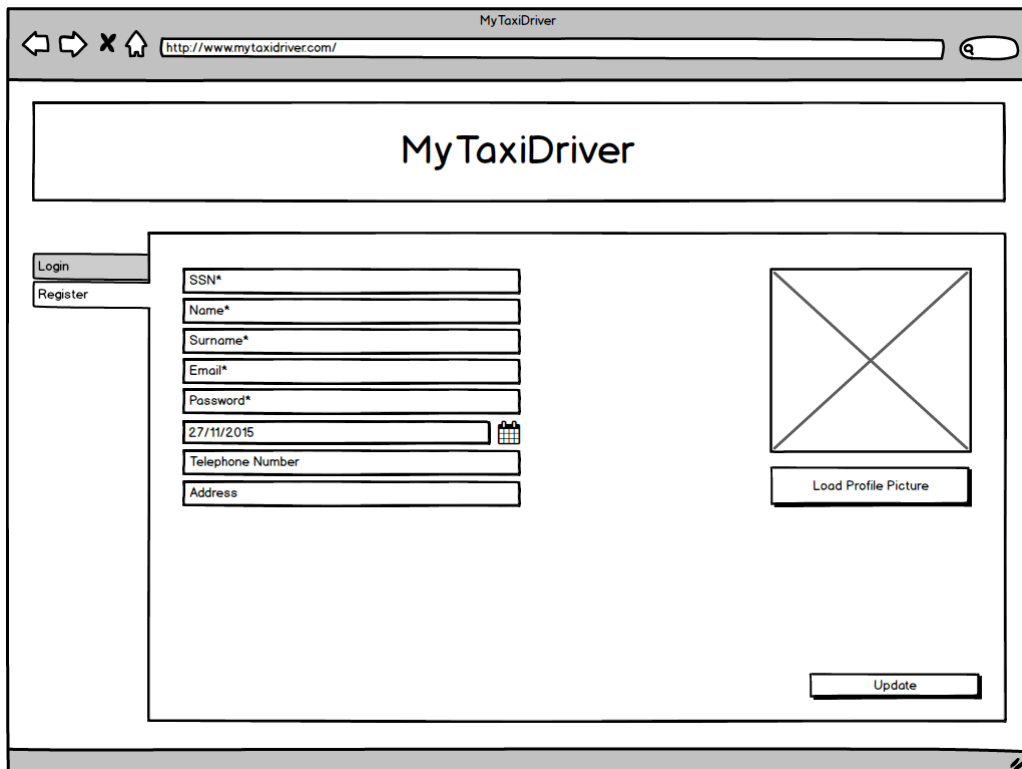


4.1.2 Register

This is how the register page will look like, both on mobile app and web browser:



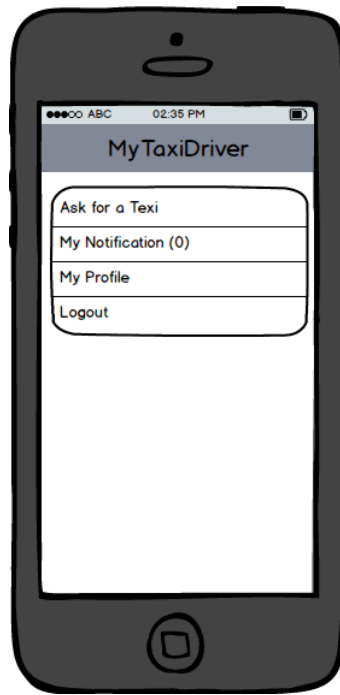
A mobile app interface for registration. The screen displays a form with the following fields: SSN*, Name*, Surname*, Email*, Password*, a date field (27/11/2015) with a calendar icon, Telephone Number, and Address. Below the form is a 'Load Profile Picture' button. At the bottom are 'Back' and 'Register' buttons. The status bar at the top shows 'ABC' and '02:35 PM'.



A web browser interface for registration. The browser address bar shows 'http://www.mytaxidriver.com/'. The page has a header 'MyTaxiDriver'. On the left, there are 'Login' and 'Register' tabs, with 'Register' selected. The main form area contains the same fields as the mobile app: SSN*, Name*, Surname*, Email*, Password*, a date field (27/11/2015) with a calendar icon, Telephone Number, and Address. To the right of the form is a large square placeholder for a profile picture with an 'X' inside, and a 'Load Profile Picture' button below it. At the bottom right of the form area is an 'Update' button.

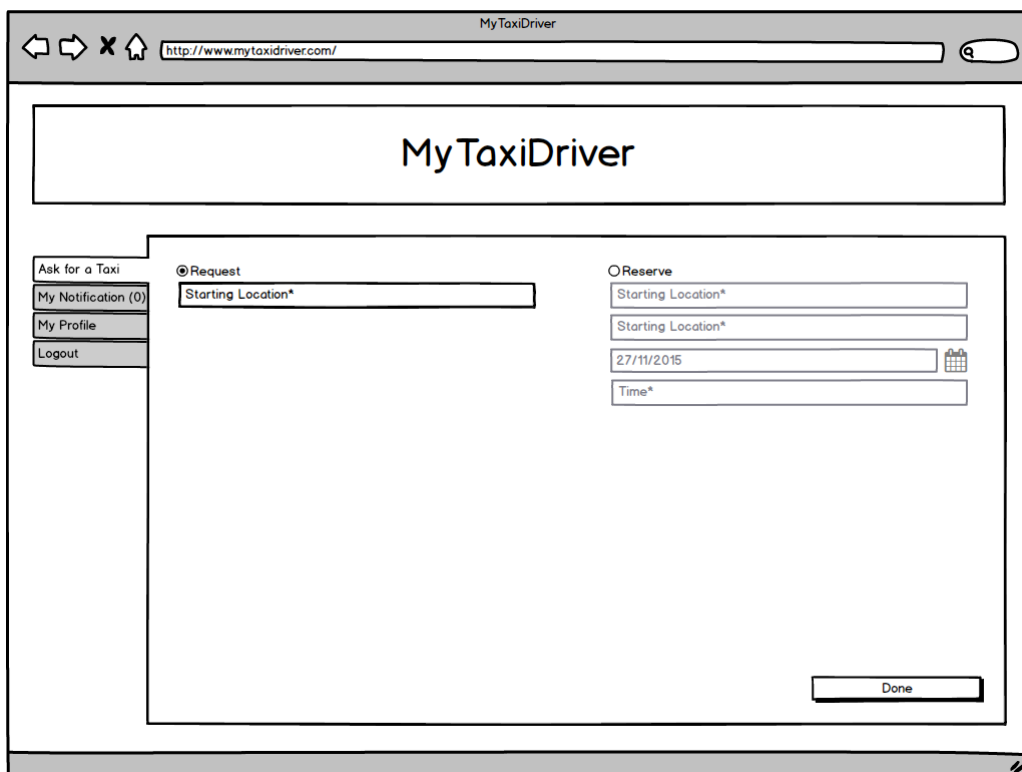
4.1.3 Main page

This is how the main page will look like on mobile app :



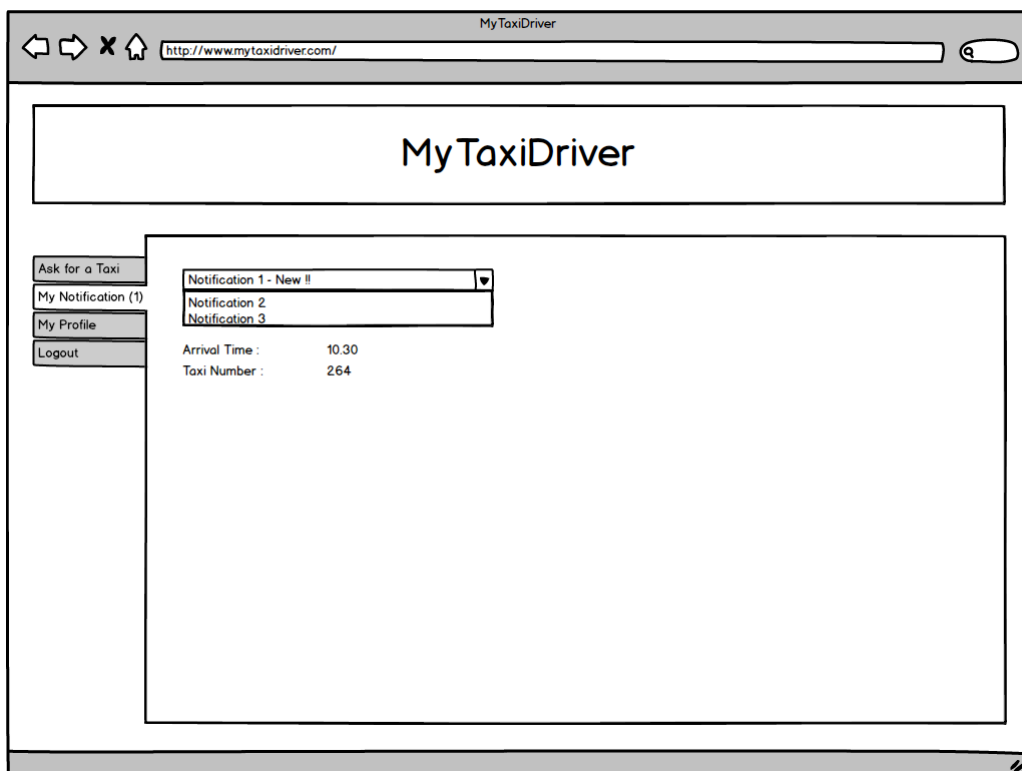
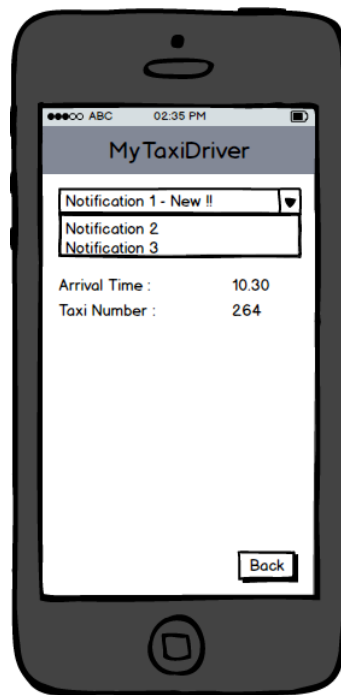
4.1.4 Ask For a Taxi

This is how the page for requesting/reserving a taxi will look like, and mobile app and on web browser:



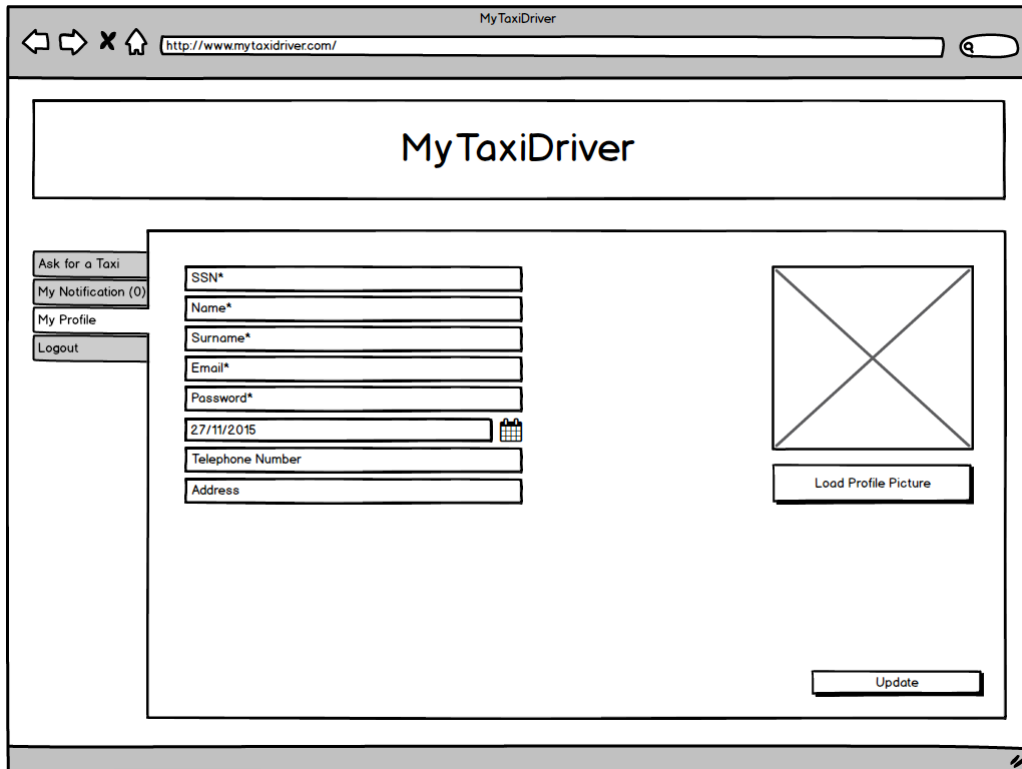
4.1.5 Notification

This is how a notification page will look like, both on mobile app both on web browser:

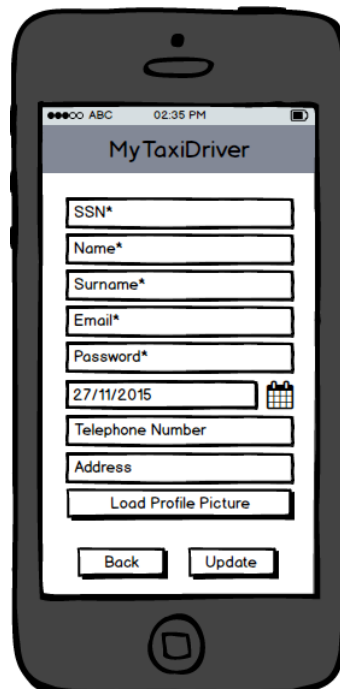


4.1.6 Profile

This is how the profile page will look like, both on mobile app both on web browser:



The web browser view shows the MyTaxiDriver profile page. The browser's address bar displays 'http://www.mytaxidriver.com/'. The page has a header with the 'MyTaxiDriver' logo. On the left, a sidebar contains links: 'Ask for a Taxi', 'My Notification (0)', 'My Profile', and 'Logout'. The main content area features a form with the following fields: 'SSN*', 'Name*', 'Surname*', 'Email*', 'Password*', a date field with '27/11/2015' and a calendar icon, 'Telephone Number', and 'Address'. To the right of the form is a placeholder for a profile picture, represented by a square with an 'X' and a 'Load Profile Picture' button below it. An 'Update' button is located at the bottom right of the form area.



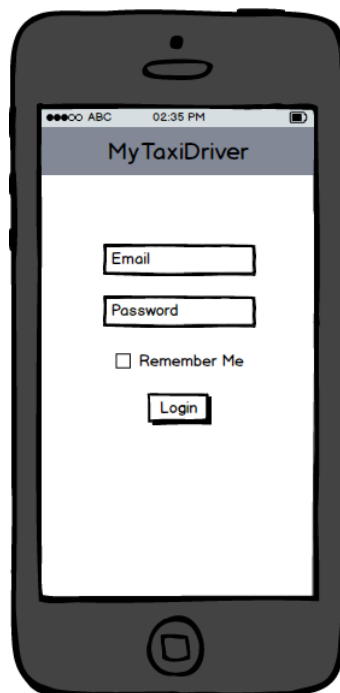
The mobile app view shows the MyTaxiDriver profile page on a smartphone. The status bar at the top shows 'ABC' and '02:35 PM'. The app's header displays 'MyTaxiDriver'. The form fields are identical to the web version: 'SSN*', 'Name*', 'Surname*', 'Email*', 'Password*', a date field with '27/11/2015' and a calendar icon, 'Telephone Number', and 'Address'. Below the form is a 'Load Profile Picture' button. At the very bottom, there are 'Back' and 'Update' buttons.

4.2 Driver GUI

In this section we will give a preview of the driver interface, notice that they can only access through mobile application:

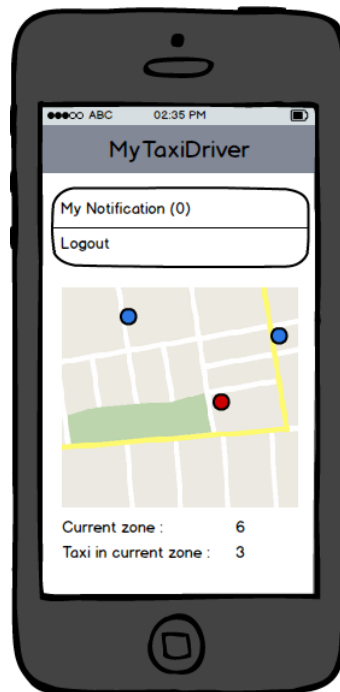
4.2.1 Login

This is how the login page will look like:



4.2.2 Main page

This is how the main page will look like:



4.2.3 Notification

This is how the a notification will look like:

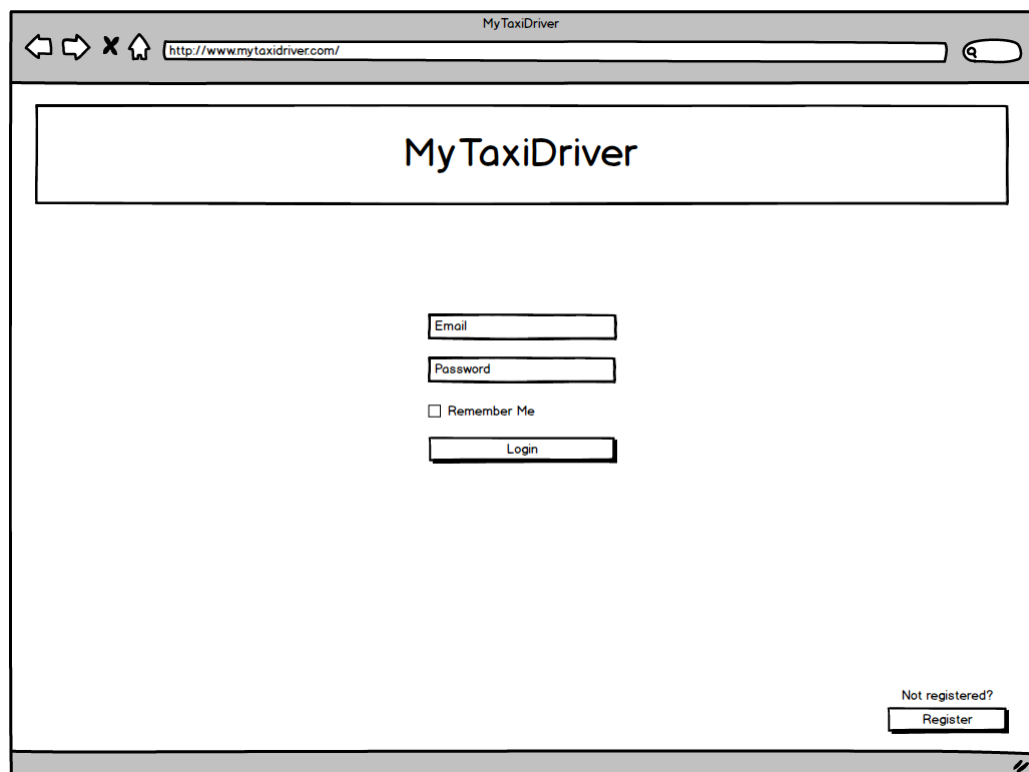


4.3 Administrator GUI

In this section we will give a preview of the driver interface, notice that he can only access through web browser:

4.3.1 Login

This is what the administrator will see when he tries to log in, notice that is the same page of the customer.



The image shows a web browser window with the title "MyTaxiDriver". The address bar displays "http://www.mytaxidriver.com/". The main content area features a large header box with the text "MyTaxiDriver". Below this, there is a login form with the following elements:

- An "Email" input field.
- A "Password" input field.
- A checkbox labeled "Remember Me".
- A "Login" button.

In the bottom right corner of the page, there is a link "Not registered?" and a "Register" button.

4.3.2 View Driver

This is what the administrator will see when looking for a driver:

The screenshot shows a web browser window titled "MyTaxiDriver" with the URL "http://www.mytaxidriver.com/". The page has a header with the "MyTaxiDriver" logo. On the left, there is a sidebar menu with the following options: "Manage Drivers", "Search Driver", "Add Driver", and "Logout". The main content area is titled "View Driver Profile" and contains a form with the following fields: "SSN*", "Name*", "Surname*", "Email*", "Password*", "Taxi Number*", a date field with the value "27/11/2015" and a calendar icon, "Telephone Number", and "Address". To the right of the form is a placeholder for a profile picture, represented by a square with an 'X' inside, and a button labeled "Load Profile Picture". At the bottom right of the form area, there are two buttons: "Delete" and "Update".

4.3.3 Search Driver

This is what the administrator will see when looking for a driver:

The screenshot shows a web browser window titled "MyTaxiDriver" with the URL "http://www.mytaxidriver.com/". The page has a header with the text "MyTaxiDriver". On the left side, there is a sidebar with four buttons: "Manage Drivers", "Search Driver", "Add Driver", and "Logout". The "Search Driver" button is highlighted. The main content area contains a form with a label "Driver*" and a text input field. At the bottom right of the form, there is an "Execute" button.

The screenshot shows the same web browser window as above, but now displaying search results. The sidebar is the same, but the "Search Driver" button is no longer highlighted. The main content area contains a table with two columns: "Driver" and "Taxi Number". The table has two rows of data: "Luca Rossi" with "20" and "Mario Rossi" with "37". The "Mario Rossi" row is highlighted in blue.

Driver	Taxi Number
Luca Rossi	20
Mario Rossi	37

4.3.4 Add Driver

This is what the administrator will see when adding a driver:

The screenshot shows a web browser window titled "MyTaxiDriver" with the URL "http://www.mytaxidriver.com/". The page features a header with the "MyTaxiDriver" logo. On the left, a sidebar contains navigation links: "Manage Drivers", "Search Driver", "Add Driver", and "Logout". The main content area is titled "Add Driver" and contains a form with the following fields: "SSN*", "Name*", "Surname*", "Email*", "Password*", "Taxi Number*", a date field with the value "27/11/2015" and a calendar icon, "Telephone Number", and "Address". To the right of the form is a placeholder for a profile picture, represented by a square with an 'X' and a "Load Profile Picture" button below it. An "Add" button is located at the bottom right of the form area.

5 Requirements Traceability

In this last session we will describe how the requirements we have expressed in the RASD are traced and respected in the design element we previously identified.

In order to do so, we are going to provide the list of the main requirement, both functional and non functional, and we will explain for each of them how they are satisfied.

5.1 Functional Requirements

- **Login and Sign up:** for both driver and customer we required to have a login (also for Administrator) and a Sign up Function . This has been provided, as the GUI mockups show, and the component who is in charge of this is the **Session Manager**. It must assure that the rules we have described for these functions are respected (i.e if credentials are correct and so on...) by directly interfacing with the **DB Manager** who retrieves data from the Database.
- **Customer functionalities:** such as what we defined in the RASD as “Personal Page”, “Request a Taxi”, “Reserve a Taxi” and the notifications. Those functions are provided by the **Web Layer** that allows the communication between the customer and the system. It creates a message that is sent to the **Customer Controller**: each customer request and every message that the system wants to deliver to the customer pass through it, which is connected to all the logged Customer and that checks and manage those messages.
- **Driver functionalities:** such as what we defined in the RASD as “Queue”, “Accept/Refuse”, “Check Queues”, “Fee Customer” and the notifications. Everything goes exactly as for the Customer functionalities except for the fact that every logged driver is now connected to the **Driver Controller** that has the same responsibilities of Customer Controller but regarding the drivers.
- **Queues:** in the RASD we specifically said that only the first driver of a zone can receive a request and that drivers can only queue in a zone they are in. This is managed by the **Queue Engine** that is in charge of keeping track of the various queue of the drivers. This component is a sub-component of the **Ride Manager** that, together with the queue engine is able to assign the right driver to the right request. **The Driver Location GPS** is the one that assure that all the rules related to Queue and location (i.e “A driver can queue only in the zone he is in” and “if a driver leaves a zone it will be removed from the queue”) by communicating periodically driver's position to the **Customer Controller** that will pack all these data coming from all the taxi and will forward them to the **Queue Manager** for a check.

5.2 Non functional requirements

- **Availability:** The parallel structure of our system will assure that it will be available for a satisfactory amount of time. Putting our machines in a **cluster**, in fact, will allow to lower the mean time between failure as much as needed.
Each node can be downtime for 1 hour/month maximum (12 hour/year each).

$$A \text{ of each node} = \frac{8760 - 12}{8760} = \frac{8748}{8760} = 0.99863$$

$$A \text{ total (parallel)} = 1 - (1 - 0.99863)(1 - 0.99863) = 0.999998$$

Only 5 minutes/year of downtime considering only hardware failure!!!

- **Security:** *In the triple tier architecture of our structure each section is separated from the others through **firewalls**, this will help decreasing the risk of malicious attempt of accessing sensible data.*
- **Robustness:** *We want our system not to crush if one of the component fails. This is achieved through the use of a cluster. Every component of a physical device is replicated into others identical devices and all of them are connector to the same, shared, replicated storage. This way, if a device fails the others will immediately take care of its load so that the system can keep functioning.*
- **Accesses and User Interfaces:** *according to the RASD, every user can access to the system using a specific page according to what kind of user they are. The login system, including the checking of the credentials, is managed by the **Session Manager** which receives. User interfaces are provided by the **User Side** previously descrided.*
- **API:** *as already mentioned before, thanks to the SOA style the system will be able to offer APIs to external clients.*

6 References

Within this document we will refer many times to the RASD so, since the Design Document is a further step into the developing of a system, it is suggested to read the RASD in order to completely understand this document Document.

We also referred to the IEEE standard for Design Document for the structure of this document.

For redacting and writing this document we spent more or less 35 hours per person