myTaxiService
-
Design Document

Davide Cremona (matr. 852365), Simone Deola (matr. 788181)

December 4, 2015

Contents

Chapter 1

Introduction

This chapter is intended to give an overall description of this document, it contains various sections like:

- **Purpose:** in this section is described the purpose of this document.
- **Scope:** in this section is described the scope of the myTaxiService system.
- **Definitions, Acronyms, Abbreviations:** here are listed all the definitions, all the acronyms and all the abbreviations that the reader will encounter in this document.
- **Reference Documents:** here are listed all the documents that the reader may need to read to understand better what is written in this document.
- **Document Structure:** here is explained the internal structure of this document, giving a brief description of each chapter.

1.1 Purpose

The MyTaxi Service Design Document is intended to provide an explanation of how the system has been designed. It's also destined to give to the software development team an overall guidance to the implementation of the architecture of the software project. These goals are achieved by describing:

- The system architecture;
- Architecture high-level components;
- Interaction between components;
- Patterns and Styles used to design these components.

1.2 Scope

The scope of myTaxiService is to simplify the interaction between Taxi Drivers and Customers. The main goal is to provide a system to request a taxi and ensure that this will arrive in a small amount of time. Also, customers can reserve taxis for a future ride. The system also ensures a fair management of taxi queues that are divided in city zones in order to reduce waiting times and to provide a fair division of work between Taxi Drivers.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Customer: a Customer is the end-user (a taxi customer) that makes requests or reservations to use taxis.
- Taxi Driver: a Taxi Driver is a driver of a taxi. He can receive requests and accept or decline them.

1.3.2 Acronyms

- RASD (or R.A.S.D.): is the Requirement Analysis and Specification Document.
- DD (or D.D.): is the Design Document (this document).
- MVC (or M.V.C.): is the Model-View-Controller software design pattern.
- DBMS (or D.B.M.S.): it's the Data Base Management System.
- UI (or U.I.): it's the acronym for User Interface.

1.4 Reference Documents

You can refer to the Requirement Analysis and Specification Document for a better understanding of what is described in this document (myTaxiService R.A.S.D. Link).

1.5 Document Structure

This document is divided in 6 main Chapters:

- Introduction: this chapter is to give a general overview of this document (purpose, references etc..).
- Architectural Design: in this chapter is described the general architecture of the system and his components. It's also described how the components interact with each other and the Design Patterns and Architectural Styles used to design them.

- Algorithm Design: here is described the most relevant algorithms used to create the system.
- User Interface Design: here is described the end-user interface of the mobile and web applications.
- Requirements Traceability: here is explained how the requirements described in the RASD document are implemented in the various components of the system.
- References: in this chapter, there are useful references to external resources that help the understanding of this document.

Chapter 2

Architectural Design

In this chapter there will be described the architecture of the system and the various components that will compose the system. Here it's also described how these components will interact to each other to perform the various tasks that the system has to do to provide to the users the functions described in the RASD document. Finally, there will be described which architectural styles and patterns have been used to design the entire system.

2.1 Overview

MyTaxiService system architecture has been designed as a 3-tier architecture (Data Tier, Logic Tier and Presentation Tier) to facilitate the development of the system using the MVC software design pattern. In fact the three tiers represent in some way the Model (Data Tier), the Controller (Logic Tier) and the View (Presentation Tier) of the software.
The architecture of the system is described by the following figure:

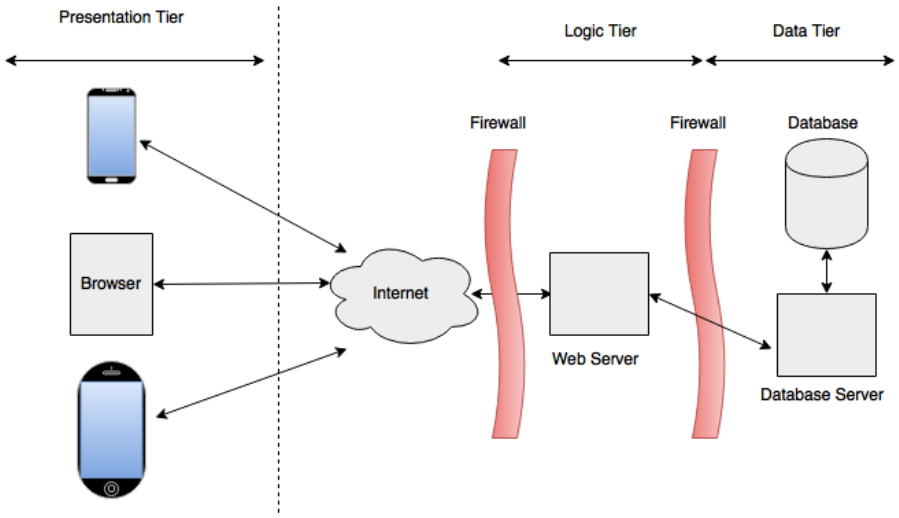

Figure 2.1: System Architecture

The components of the system are divided in the three tiers:

- In the Presentation Tier there are implemented classes and the objects that compose the view part (what the user see). From this tier users can performing actions to query the system and use the web and mobile applications.
- The Logic Tier is the place where are implemented classes and objects that compose the controller (actual logic of the system). This part is in charge of taking requests by the users applications and perform changes on the data. This part is also in charge to notify the applications (presentation tier) of the data changes.
- The Data Tier is the place where are implemented the classes and the objects that interact directly with the database. These objects represent the entities of the database and are able to perform creation, modification and deletion of database records.

2.2 High Level Components and Their Interaction

In this section each Tier is decomposed in his high level components. For each component is given a short description of his purpose and how the component interacts with the other components.

2.2.1 Data Tier

The Data Tier is the part of the system that includes the database and all the classes and the objects that will interact with it. It's composed by these components:

- **Database:** it's the data structure where all the system data are stored. The management of this component is delegated to the DBMS. The DBMS will provide a query language to extract data from the database.

- **Database Manager:** this component represents the Model of the MVC design pattern, it will include all the objects and the classes of the Data Model. This component also contains all the classes that are in charge to perform queries to the DBMS in order to make the data available to the Logic Tier.

2.2.2 Logic Tier

In this Tier are implemented all the classes and the objects that are used to logically organize the requests and the responses from/to the Presentation Tier and the Data Tier. This Tier represents the Controller of the MVC software design pattern and it's composed by these components:

- **IO Manager:** This component is in charge to collect all the requests from the myTaxiService clients and to forward them to the correct component of the system by calling other components functions. It's also in charge to collect messages (like notifications, error messages...) from the internal components and to forward them to the correct user.

- **Requests an Reservations Manager:** This component receives from the IO Manager component requests that concern requests for taxis and to reserve a future taxi ride. It's in charge of checking the validity of the data that receives and, if it's necessary, to call functions of the Database Manager to insert these requests into the database.
  Another task of this component is to continuously check if there are some requests or reservations that are unhandled in the database and to allocate a taxi that has to be selected using the Zones Manager. Once a taxi is allocated, a notification to the Taxi Driver is sent through the IO Manager.

- **Zones Manager:** This component is in charge to manage the zones of the cities. It receives GPS updates of Taxi Drivers from the IO Manager and compute them to update the zone queues of the relative city. This component also orders Taxi Drivers queues in function of their priority and provide some useful functions (for instance: get the first available Taxi Driver).

- **Profile Manager:** This component receives requests by the IO Manager that concerns the visualization or the modification of the users profiles. It checks if the modifications can take place and then calls Database Manager functions to modify the database entries. Once a modification is performed, then this components will send a notification to the user by calling the right IO Manager function.
  Another task of this component is to provide registration and login functions to the users interacting with the Database Manager.

2.2.3 Presentation Tier

The purpose of this tier is to make the data and their changes visible to the user. Presentation Tier is also in charge to make the system functions available to the users according to their permissions (see the RASD to read about functionality differences between Customers and Taxi Drivers). This Tier represents the View of the MVC design pattern and it's composed by these components:

- **Web Application UI:** this component contains all the objects that the users need to comunicate with the system, via web browser.
- **Mobile Application UI:** it contains all the necessary components to comunicate with the system via mobile devices.

2.2.4 Schema

To give a visual description of what is written so far, here is provided a schema of the system:

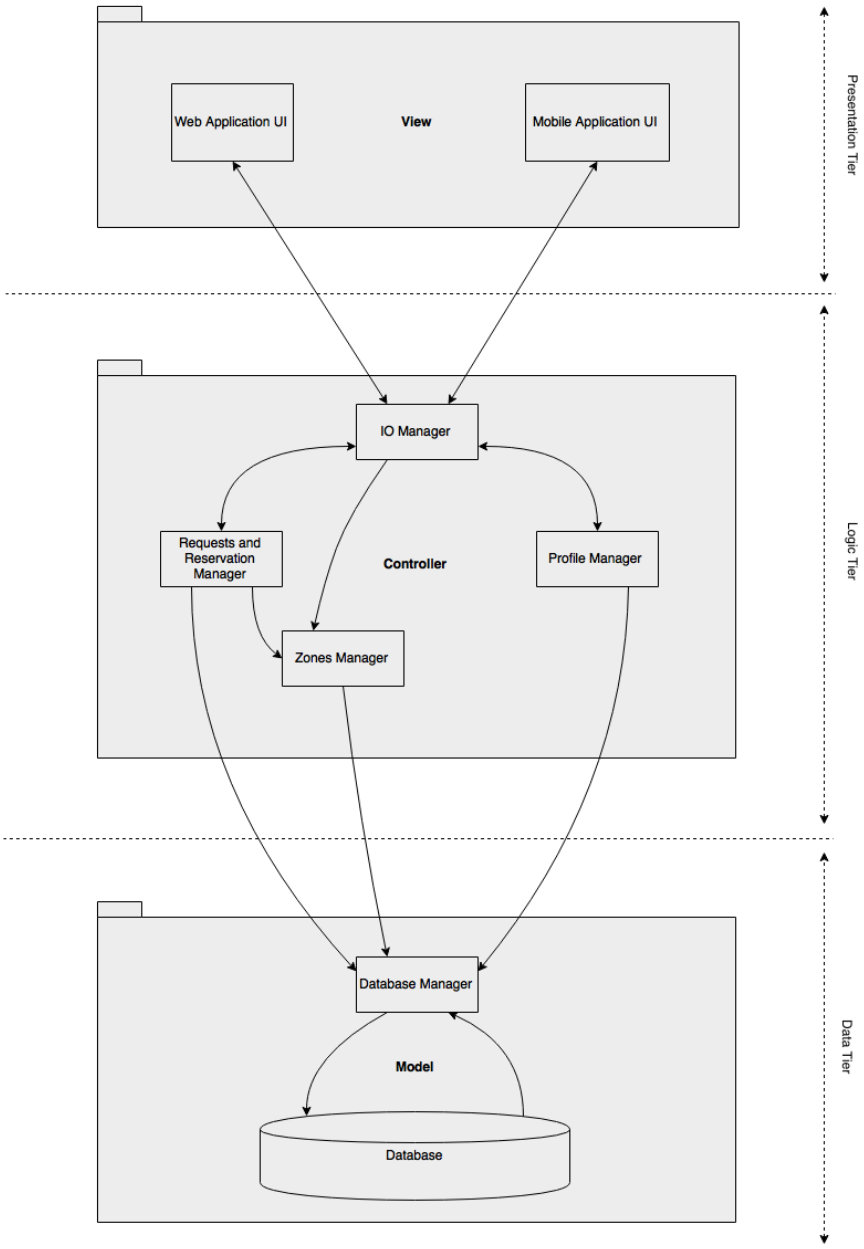

Figure 2.2: High Level Components and Interactions

2.3 Component View

In this section will be given a more detailed description of the component described in the previous section. For each component are described the main classes to give a general overview of how the system is going to be implemented.

2.3.1 Data Tier

This part represents the model of the system (using a MVC architecture), so the classes used in this section are the representation of the information that must be stored to use the system.

Database

The database is the data structure, so aren't specified the classes used by it, all the information stored are managed by the Database Manager.

Database Manager

The main classes used to represent the stored data of the application are:

- Registered User: this class contains the information about the registered user.
- Taxi Driver: this class extends the Registered User class and represents the information about the taxi drivers
- Customer: this class extends the Registered User class and represents the information about the customers
- Zone: this class represents the information about the zones. In this class is also stored the queue of the taxi drivers that are available in this specific zone.
- Ride: this class contains the general information that Ride and Request share. This abstract class follows the pattern State to represent the state of a ride. All Rides start in the state NOT_HANDLED, then they can pass to the state HANDLED, then from this state they can pass to the state NOT_HANDLED.
- Request: this class extends the Ride class and represents the information of the requests for a ride.
- Reservation: this class extends the Ride class by adding information about the starting and the ending position, and the time of the meeting. Adding this information can be represented the reservation in the system.

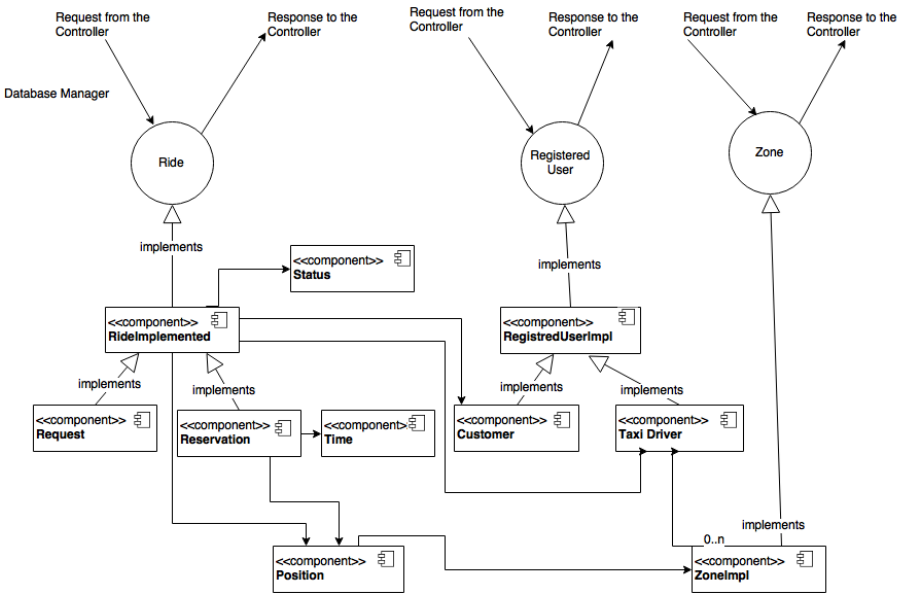

Figure 2.3: Data Manager Structure

2.3.2 Logic Tier

Here are described in short the various sub-components that are included in the Logic Tier (Model package).

IO Manager

This component will be implemented using the Command Pattern (see Command Pattern on Wikipedia). It will contain the following main classes, in general, for each functionality of the UI there is a Message class that, in order to be brief, are not all listed:

- **Message Type** (Interface) **:** this sub-component is the Interface which is extended by each Message class.
- **Message Types:** this sub-component is an enumeration in which each enumeration entry returns an implementation of the Message Type Interface.
- **Message Types Map:** this sub-component is an Hash Map that associates the name of the elements of the enumeration, the enumeration element is identified by that name.
- **Messages Handler:** this sub-component receives messages from the View classes. It selects the right message to send by processing the message name and selecting (through the Message Type Map) the right Message Class.

- **Request Message:** This sub-component is in charge to call Requests and Reservations functions to add a Taxi Request in the myTaxiService database.
- **Reservation Message:** This sub-component is in charge to call Requests and Reservations functions to add a Taxi Reservation in the myTaxiService database.
- **Customer Notification Message:** This sub-component is used to send notifications to a Customer. These notifications are used mainly to confirm to a Customer that his request has been handled.
- **TaxiDriver Notification Message:** This sub-component is used to send notifications to a Taxi Driver. These notifications are mainly used to advise a Taxi Driver that he's been selected for a certain ride request.
- **Login Message:** This sub-component is in charge to forward login requests from users to the Profile Manager.
- **Registration Message:** This sub-component is in charge to forward registration requests from users to the Profile Manager.
- **Ok Message:** This sub-component is used to send success notifications to a specific user.
- **Error Message:** This sub-component is used to send error notifications to a specific user.
- **End Ride Message:** This sub-component is in charge to call the functions of the Requests and Reservations Manager that can set as ended a ride. An End Ride Message is sent by the clients' applications to notify that a ride is ended.
- **Profile Modification Message:** This sub-component is in charge to call the functions of the Profile Manager that can modify a user profile.

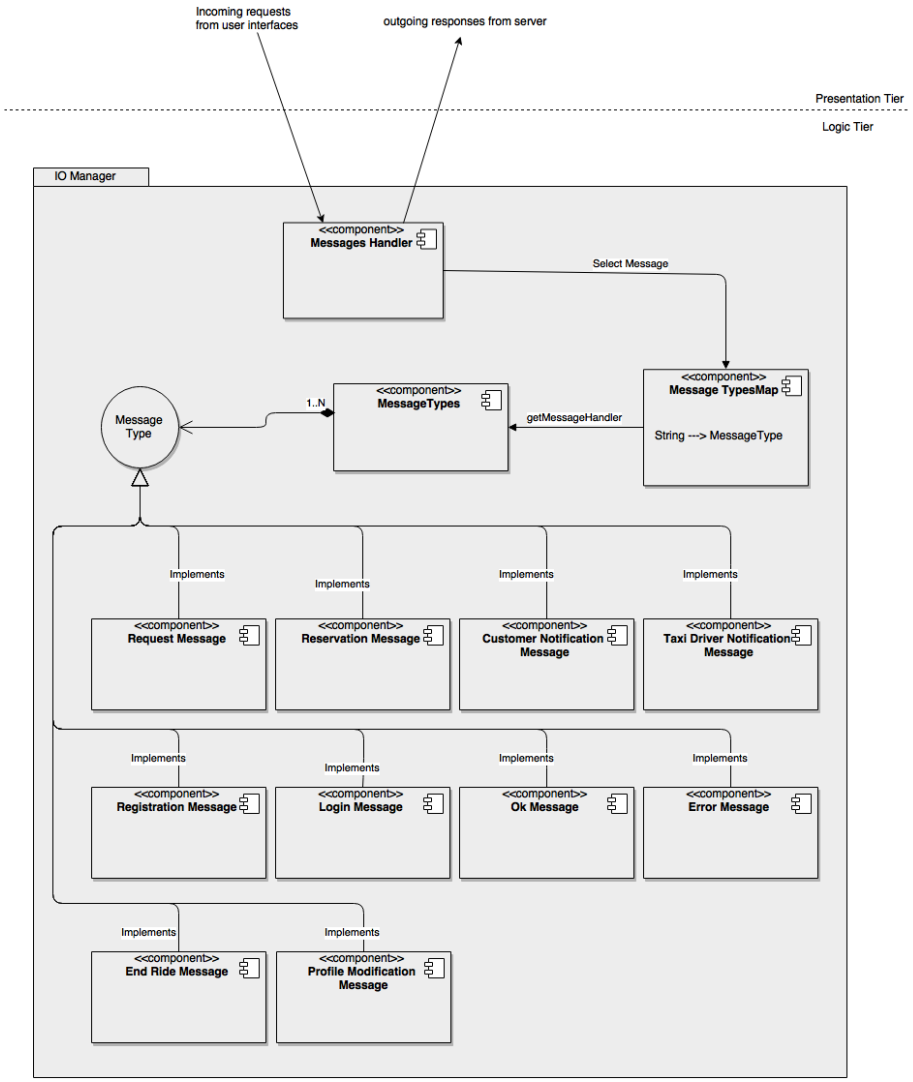

Figure 2.4: IO Manager Structure

Requests and Reservations Manager

This component contains all the classes that are in charge to add/remove ride requests and ride reservations. In this component are also contained classes that check the database to find not handled requests or reservations that have to be handled.
Here are contained these main classes:

- **Taxi Request Adder:** This sub-component is in charge to create the data for a new request entry in the database and send them to the Data Tier calling the Database Manager function that add a new Ride in the Database. These data are Customer ID, Customer Position, Customer Zone, Request Time and Date.

- **Taxi Reservation Manager:** This sub-component is in charge to create the data for a new taxi reservation entry in the database and send them to the Data Tier calling the Database Manager function, that add a new Reservation in the Database. These data are Customer ID, starting position, ending position, request time and date.
  This sub-component is also in charge to send data about the Reservations to the Customers and to delete one Reservation if it can be done.

- **Taxi Allocator:** This sub-component is in charge to check if there are reservations or ride requests that are not been handled yet and handle them. This means that Taxi Allocator has to communicate with Zones Manager (to get the first available Taxi Driver), with the IO Manager (to notify Customers and Taxi Drivers) and with the Database Manager (to get reservations and requests to allocate).

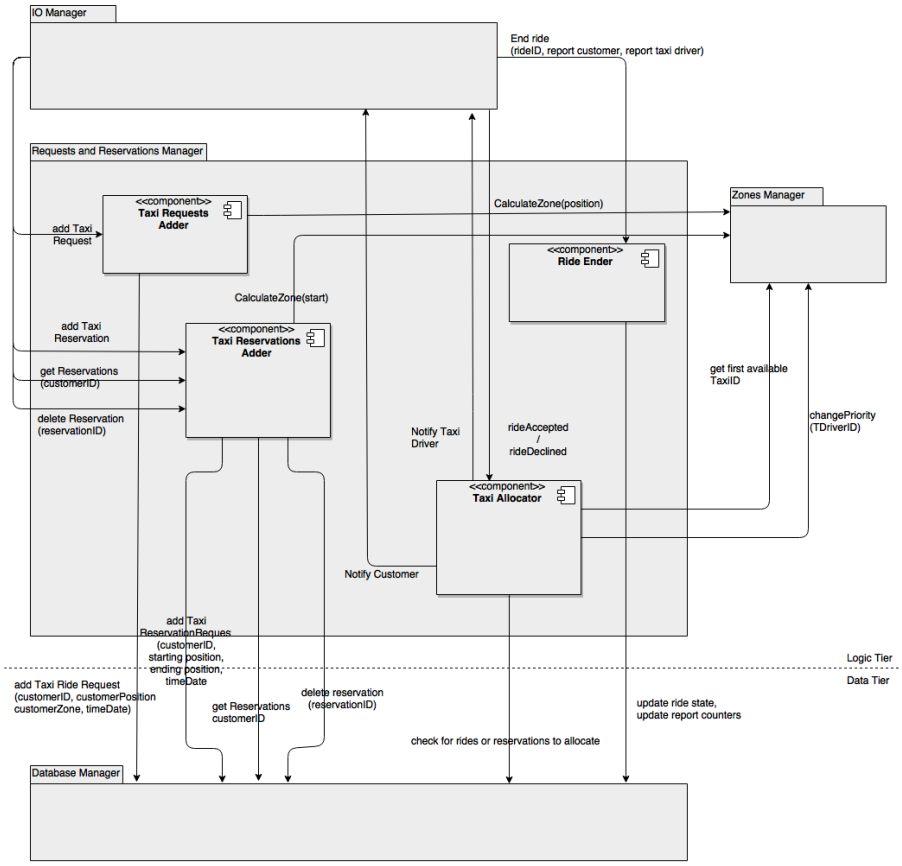

Figure 2.5: Requests And Reservations Manager Structure

Zones Manager

This component contains all the sub-components needed to manage zones and taxi queues. It provides some functions to get available Taxi Drivers and to notify that a Taxi Driver has been chosen for a ride.

- **Zone Calculator:** The only function of this sub-component is to return the zone associated to a certain position.
- **Queues Manager:** This sub-component is in charge to keep the queues updated and to provide functions to select Taxi Driver from them.
- **GPS Updater:** This sub-component is in charge to receive GPS updates from IO Manager and to ask to the Database Manager to update them.

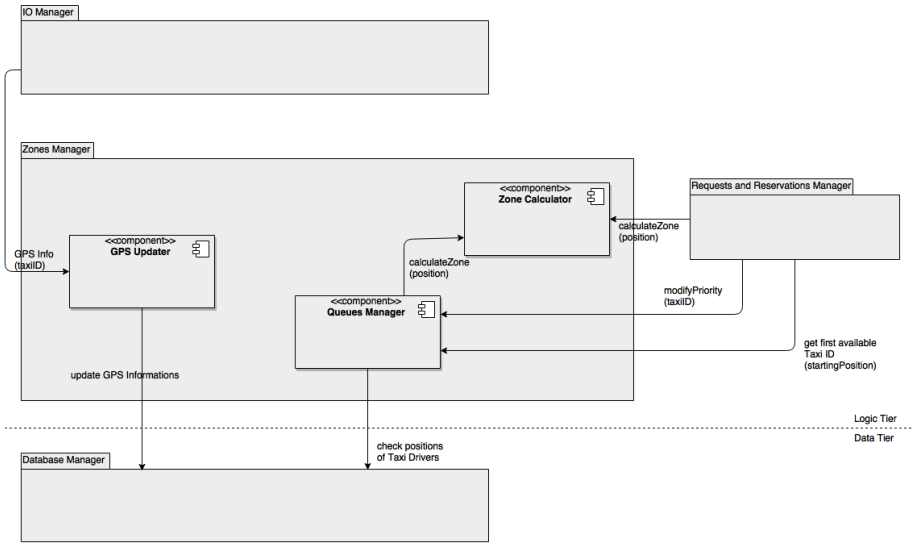

Figure 2.6: Zones Manager Structure

Profile Manager

This component has to check if a registration or a login request is legal or not and, if it is, then the relative function of the Database Manager has to be called.

- **Registration Manager:** This sub-component is in charge to accept registration requests from IO Manager, check the credentials using the Credential Checker and then, if necessary, register a new user using the functions provided by the Database Manager.
- **Login Manager:** This sub-component is in charge to accept login requests from IO Manager, check the credentials using the Credential Checker and then set a user as logged in using the functions provided by the Database Manager.
- **Profile Modifier:** This sub-component is in charge to accept profile modification requests from the IO Manager, check the new credentials and, if it is required, change the profile credentials using the functions provided by the Database Manager.
- **Profile Viewer:** This sub-component is in charge to provide to the IO Manager information about an user.
- **Credential Checker:** This sub-component is in charge to check the credentials provided by the others sub-components. Here are implemented all the rules about the acceptance of users credentials (like password composition policy etc...).

- **Profile Security:** This sub-component is in charge to retrieve a password if the customer asks for it. It checks if the email is correct using the Credential Checker and also checks if the email is registered in the system database. Then, this sub-component, sends an email to the user with his password.

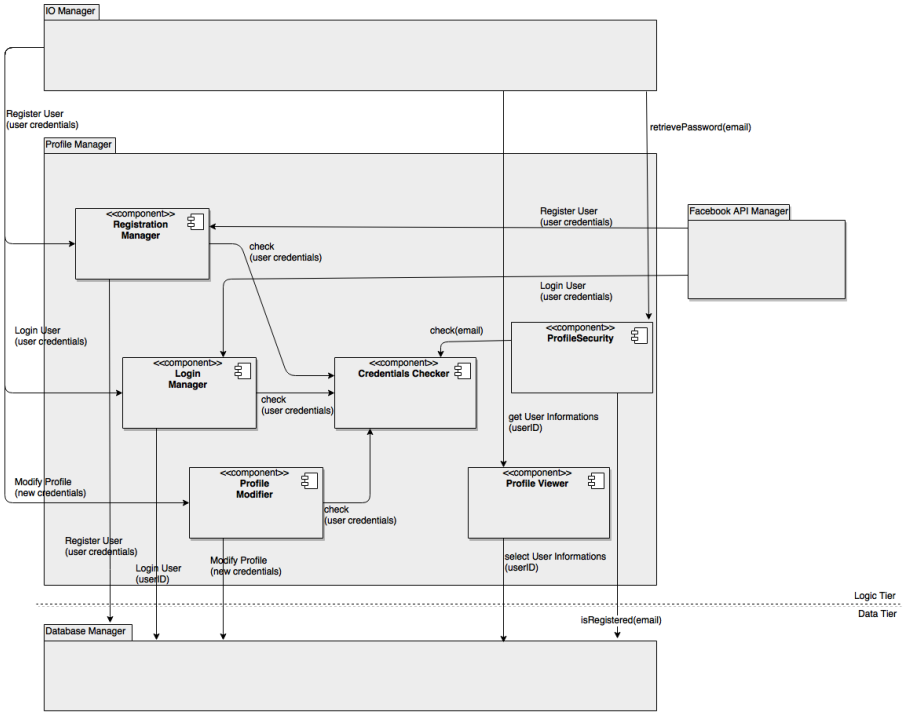

Figure 2.7: Profile Manager Structure

2.3.3 Presentation Tier

In this tier, the two main components that are found are the Web Browser User Interface and the Application User Interface. This two components are different in the programming language and they will run on different devices. This two components must comunicate with the controller via the same set of class that must send and read data in the same way. The idea was that the view of the UI can change to be optimized for the device, but the communication with the Logic tier must be the same. So each UI must have two kinds of classes: the first is composed by the classes of visualization, that must follow the organization shown on the mock object of the RASD. For example on the RASD we describe all the pages and all the input form of the two kinds of application. The second one must contain a class that can send messages to the Logic Tier and an another class that can receives its responses. The first class is called CommandSender and the second one ResponseReceiver.

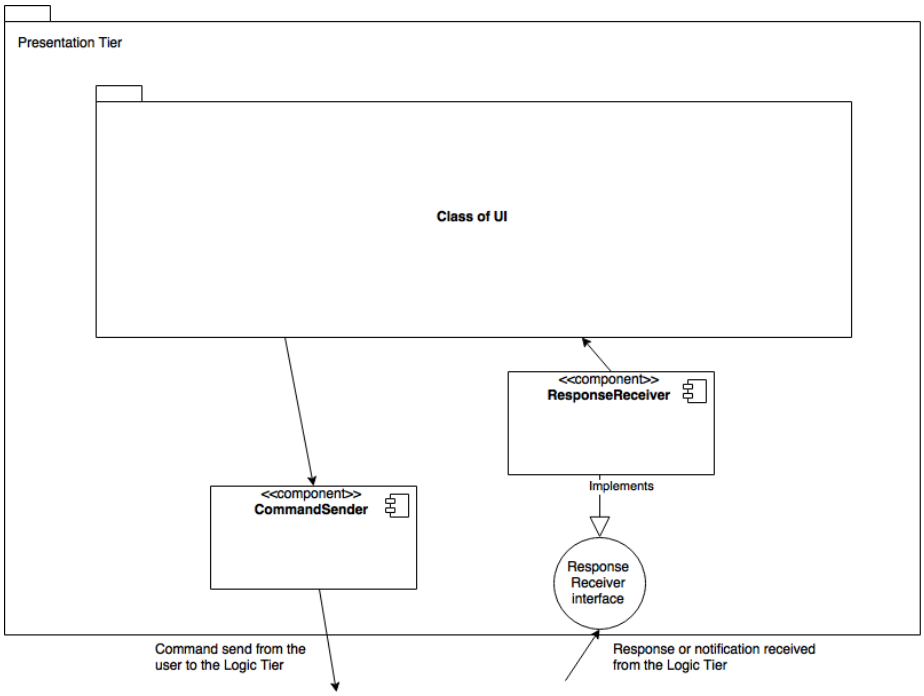

Figure 2.8: Presentation Tier Structure

2.4 Deployment View

The main hardware devices involved in this system are the following:

- Web Server: on this server there is the logic tier of the application, this tier contains the logical part of the system, so this devices must have the higher computation power between the devices. It offers via HTTP the connection from the user, and accesses to the database via XMPP (communication protocol based on XML typical used for inter-server communication).

- Data Base Server: on this server there is the data tier of the application. This device must store in its memory all the data of the system. So, to avoid problem, the memory of this device must be suitable to store all the information. This device must be connected to the web server with the purpose of making available to the logic tier the information that the data tier store. The communication between these two tiers takes place in XMPP.

- User Devices: this kind of devices is divided into Mobile Devices and Computers. In this kind of devices there is the presentation tier of the application. The kind of component used on this devices depends on the type of devices

and on the OS installed (as we have described on the chapter Software Limitation in the RASD) . We can't know the hardware specification of this devices, so the presentation tier must be as light as possible. The communication between these devices and the Web Server takes place in HTTP through the internet.

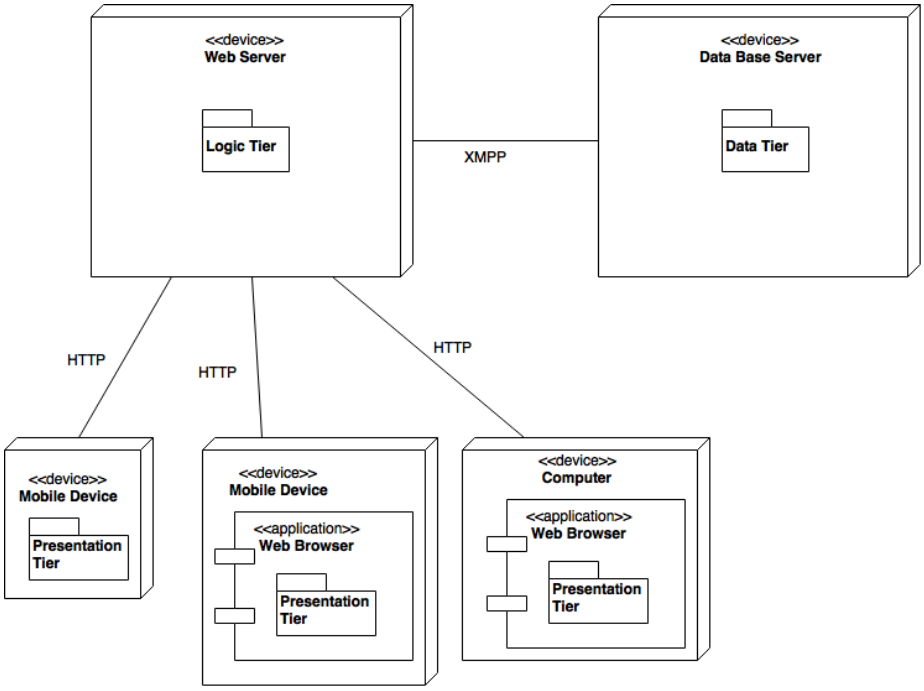

Figure 2.9: Deployment Diagram

2.5 Runtime View

In this section is described the role of the components for each scenario of the RASD (Section 3.3 of RASD)

2.5.1 Registration Through the Creation of a New myTaxiService Account

Runtime View relative to the Scenario S.1 of the RASD.

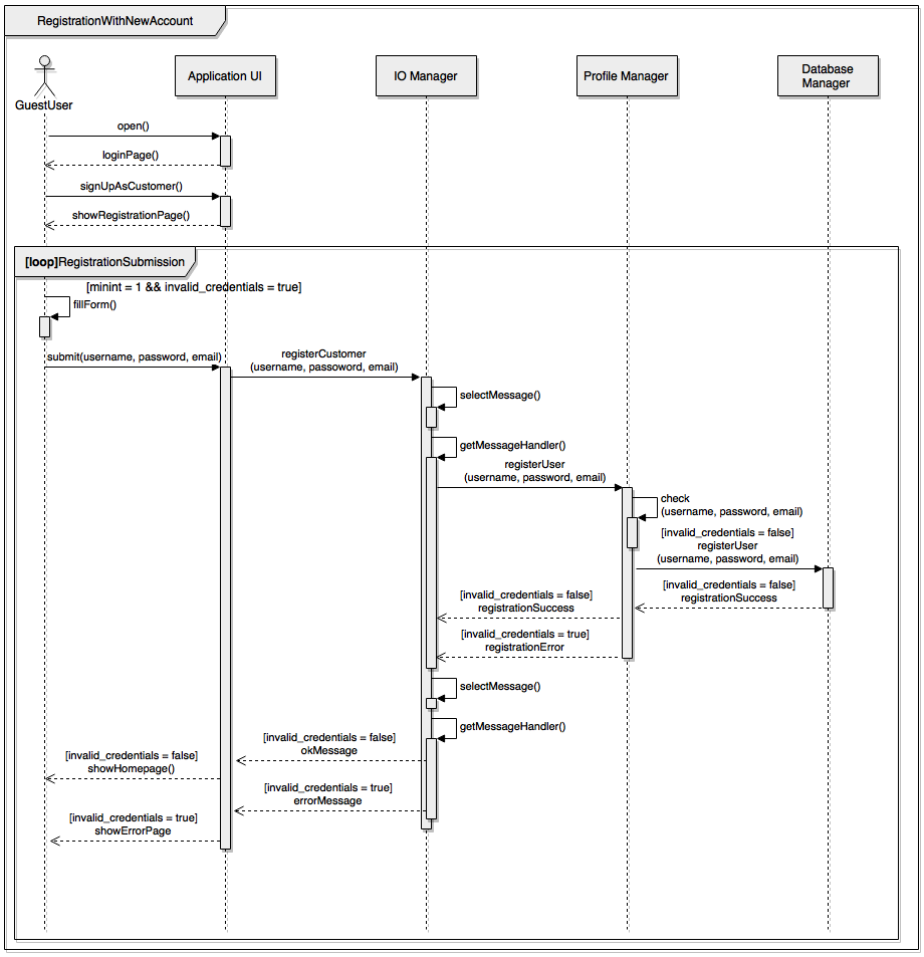

Figure 2.10: Registration with New Account Runtime View

2.5.2 Registration Using Facebook Account

Runtime View relative to the Scenario S.2 of the RASD.

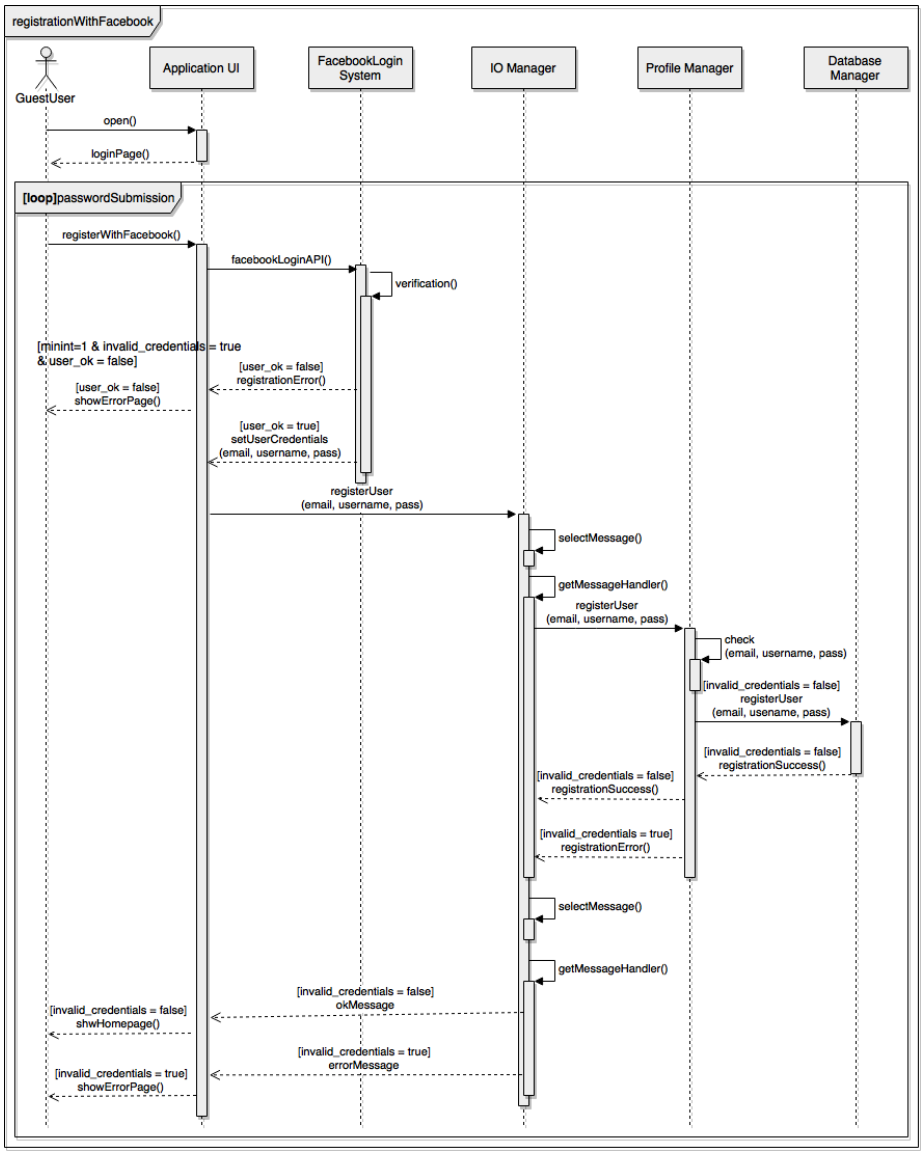

Figure 2.11: Registration with Facebook Account Runtime View

2.5.3 Taxi Driver Registration

Runtime View relative to the Scenario S.3 of the RASD.
The registration of a Taxi Driver is quite similar to the registration of a Customer. In order to be brief, this kind of registration is not modeled with a Sequence Diagram because it's enough to take the "Registration Through The Creation Of A New myTaxiService Account" and substitute the "registerUser(username, password,

email)" call with a "taxiDriverRegistration(username, password, email, license, taxiNumber)" call.

2.5.4 Registered User Login

Runtime View relative to the Scenario S.4 of the RASD.

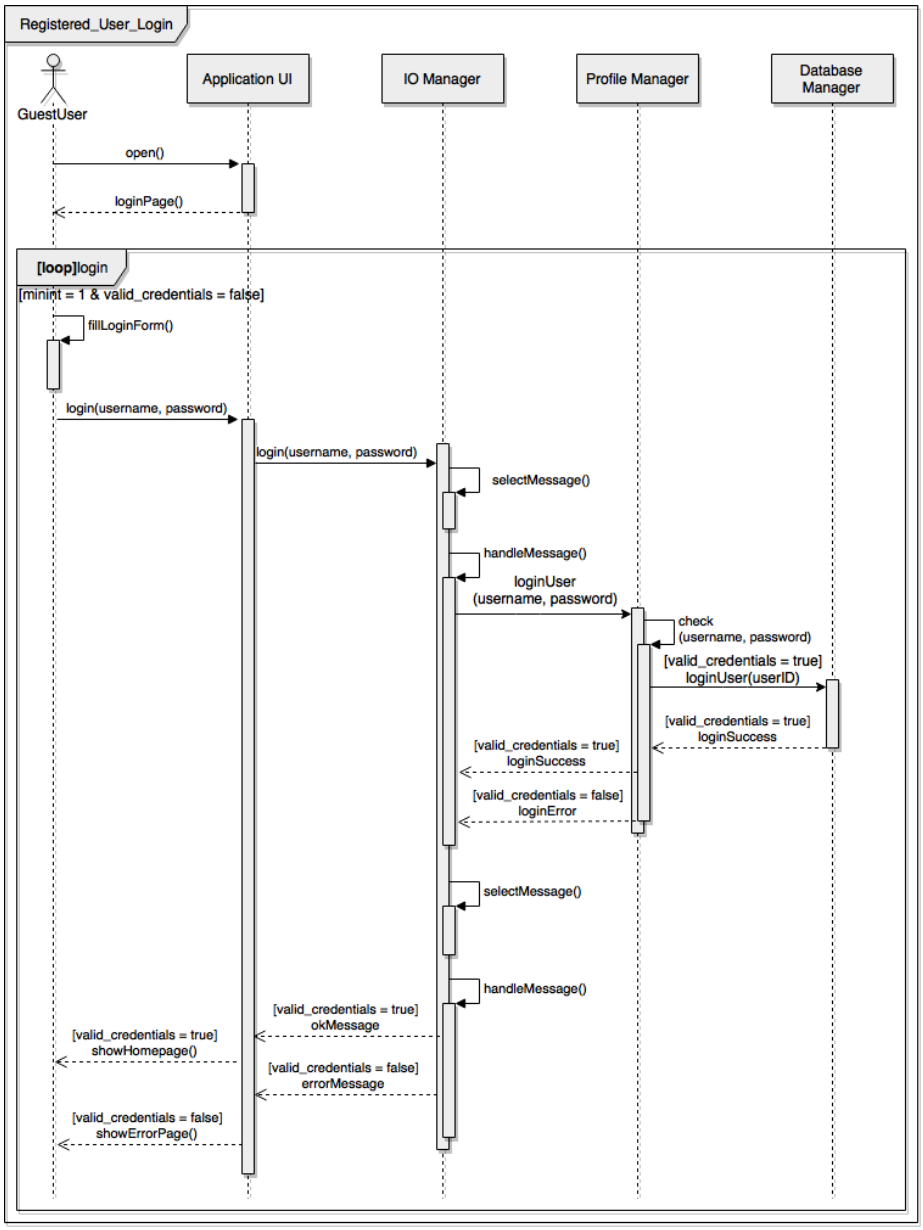

Figure 2.12: Registered User Login Runtime View

2.5.5 Customer Facebook Login

Runtime View relative to the Scenario S.5 of the RASD.

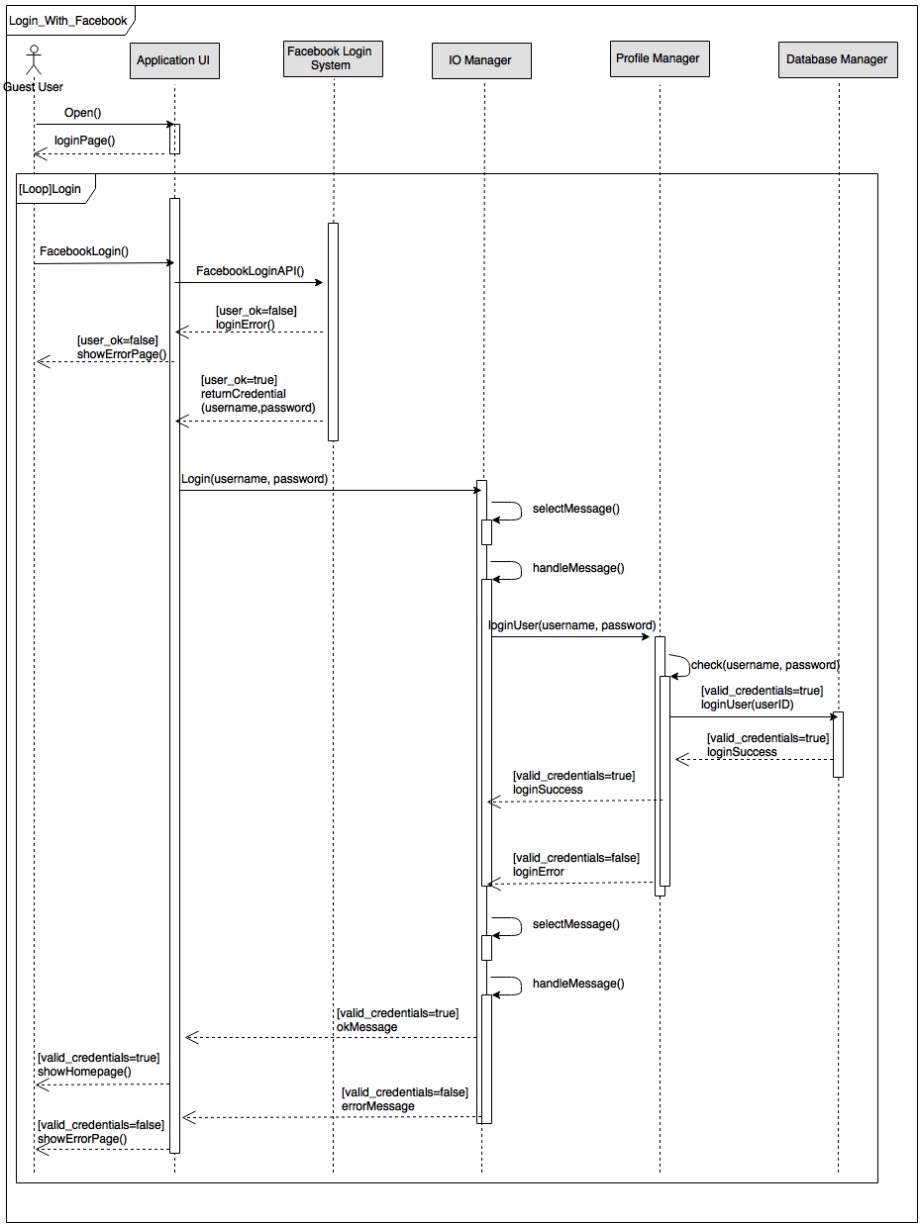

Figure 2.13: Registered User Facebook Login Runtime View

2.5.6 Profile Modification

Runtime View relative to the Scenario S.6 of the RASD.

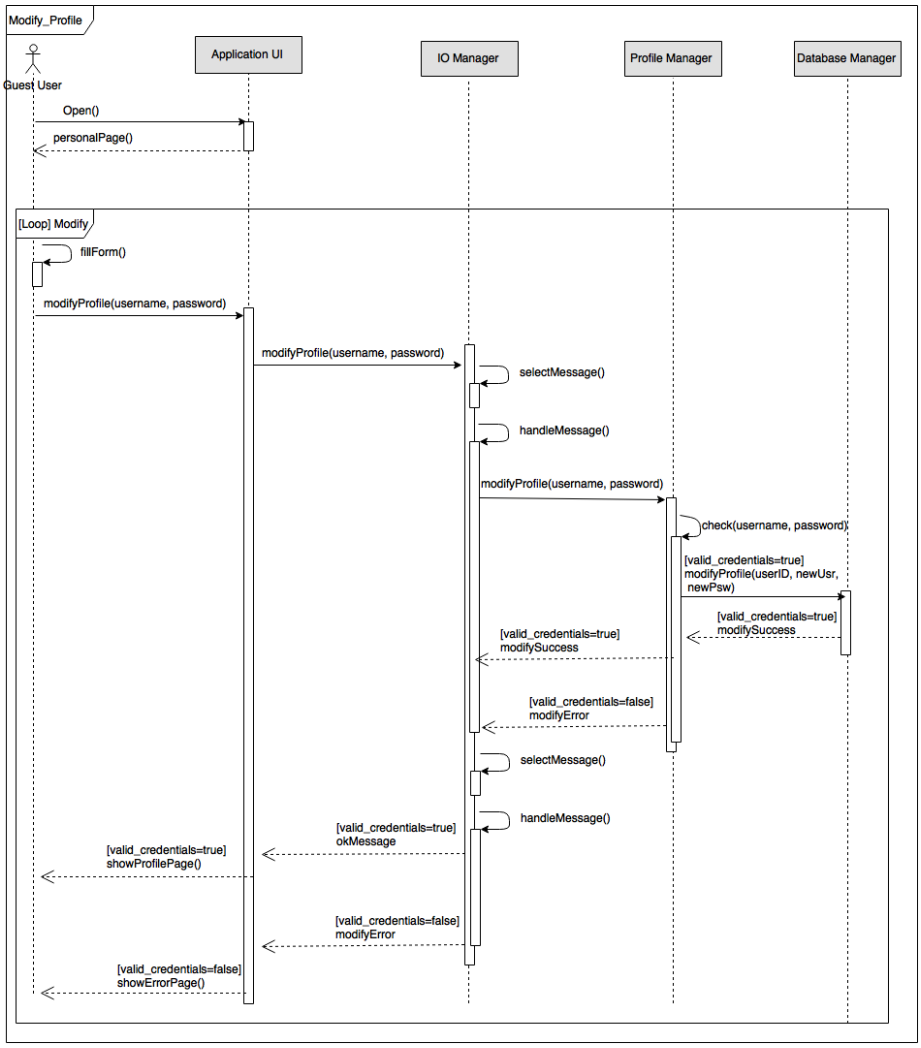

Figure 2.14: Modify Personal Profile Runtime View

2.5.7 Password Retreival

Runtime View relative to the Scenario S.7 of the RASD.

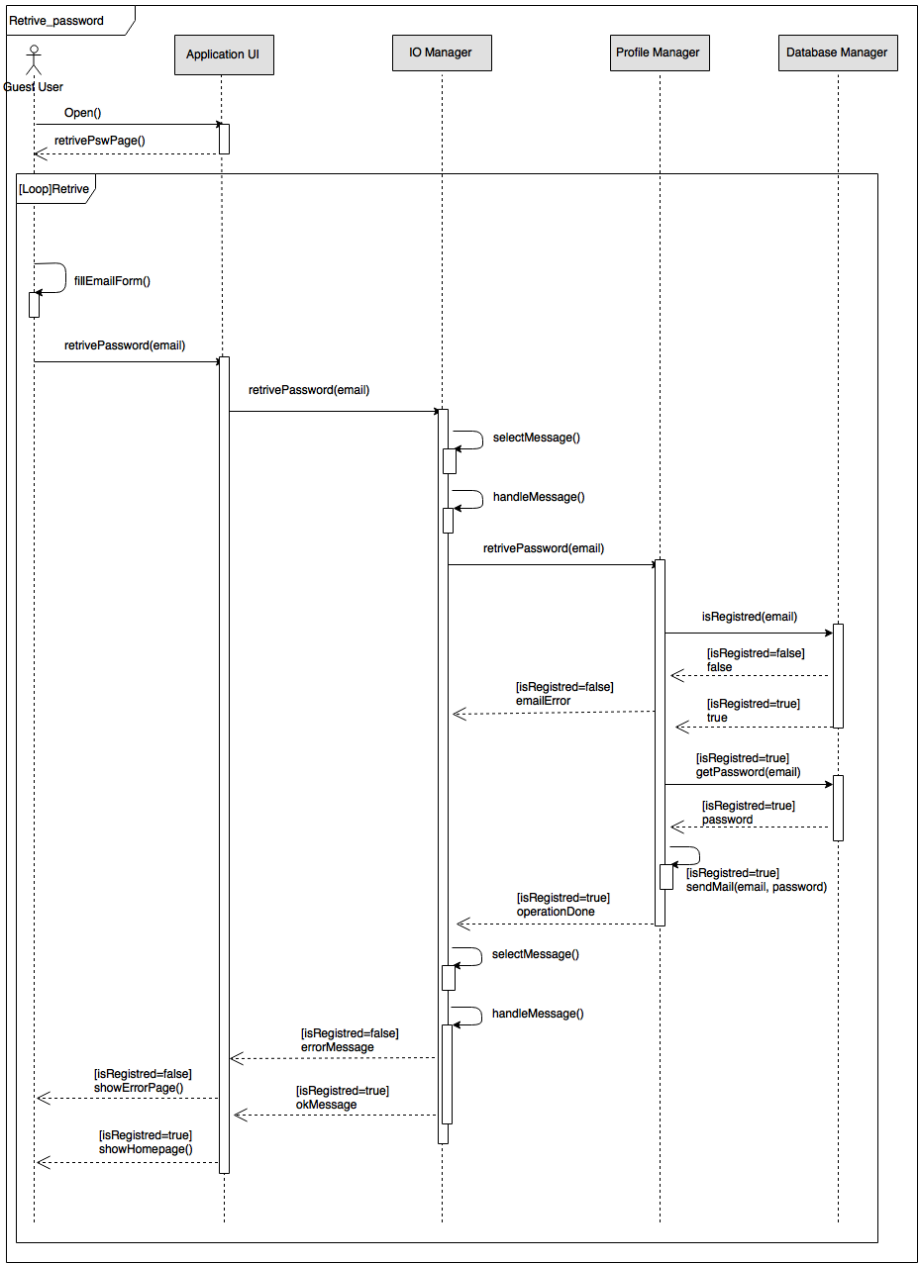

Figure 2.15: Password Retriving Runtime View

2.5.8 Taxi Driver Reporting

Runtime View relative to the Scenario S.8 of the RASD.

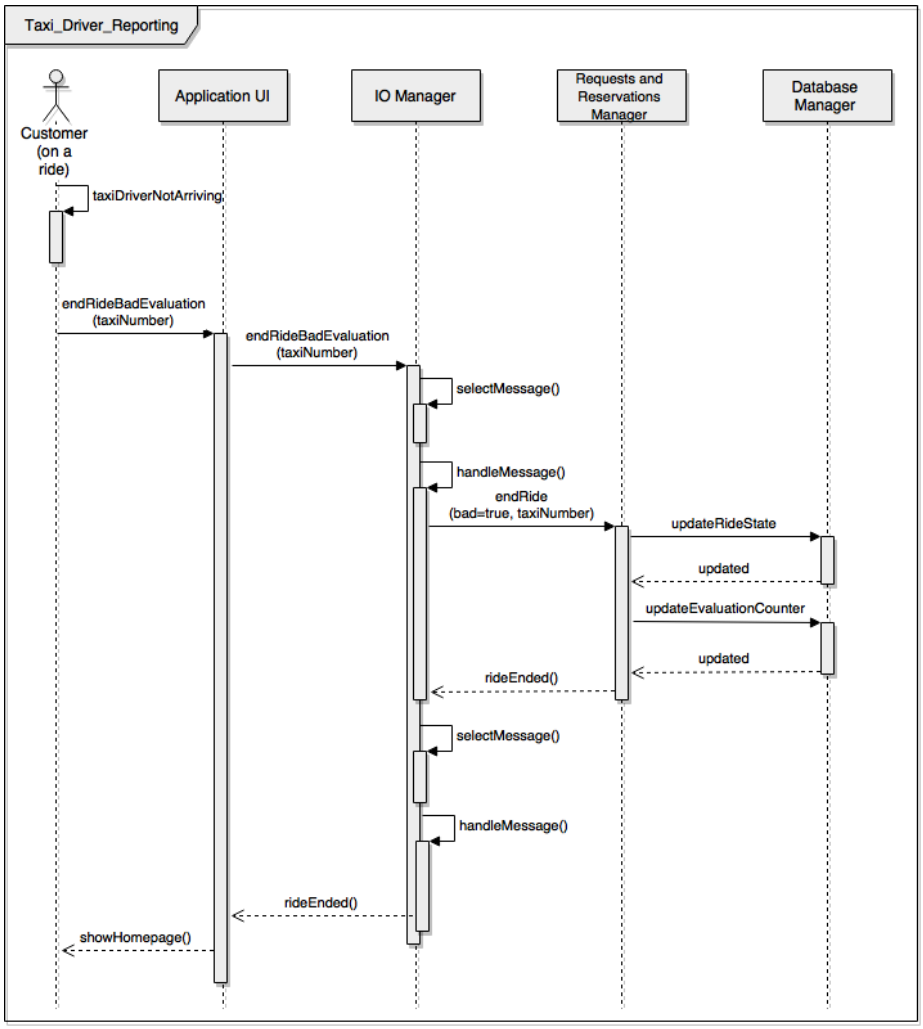

Figure 2.16: Taxi Driver Reporting Runtime View

2.5.9 Customer Reporting

Runtime View relative to the Scenario S.8 of the RASD.

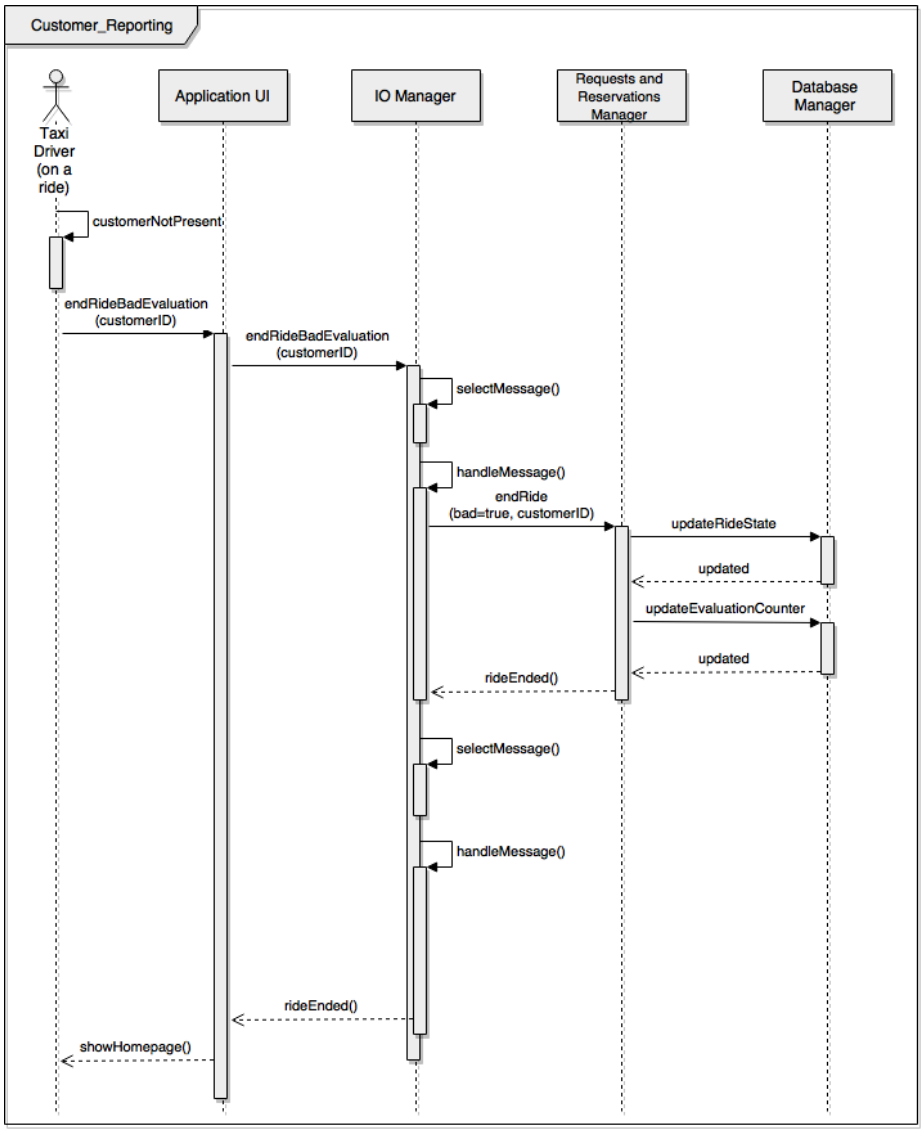

Figure 2.17: Customer Reporting Runtime View

2.5.10 Request A Taxi

Runtime View relative to the Scenario S.9 of the RASD.

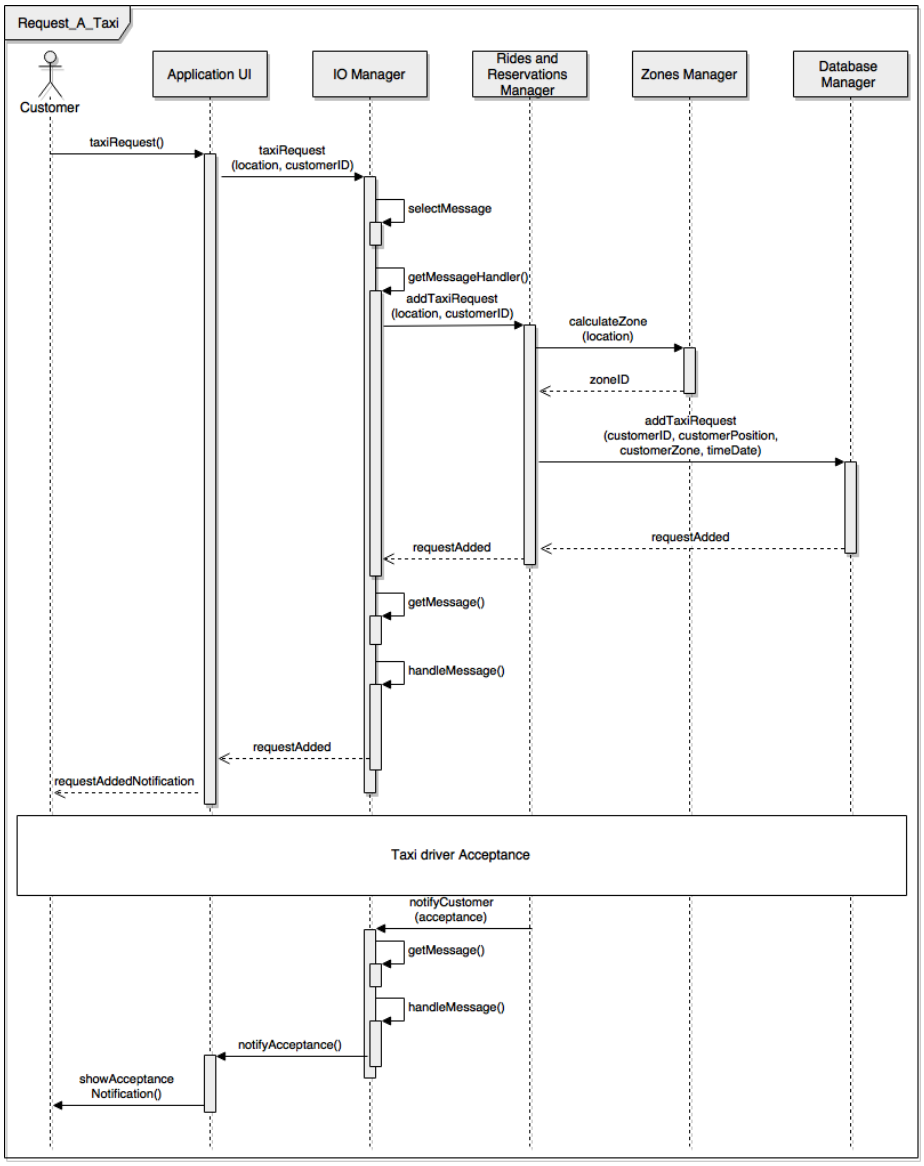

Figure 2.18: Request a Taxi Runtime View

2.5.11 Reserve A Taxi

Runtime View relative to the Scenario S.10 of the RASD.

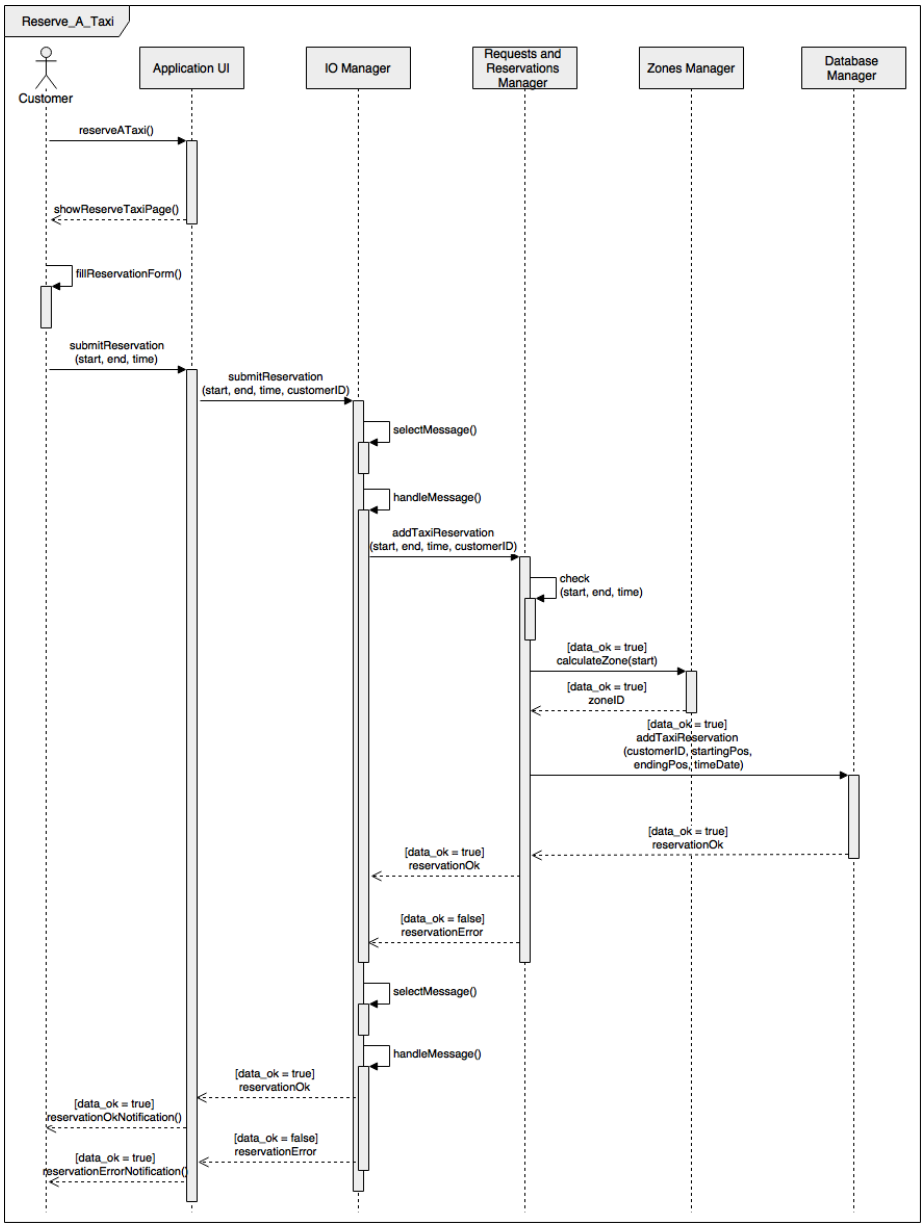

Figure 2.19: Reserve a Taxi Runtime View

2.5.12 Delete A Reservation

Runtime View relative to the Scenario S.11 of the RASD.

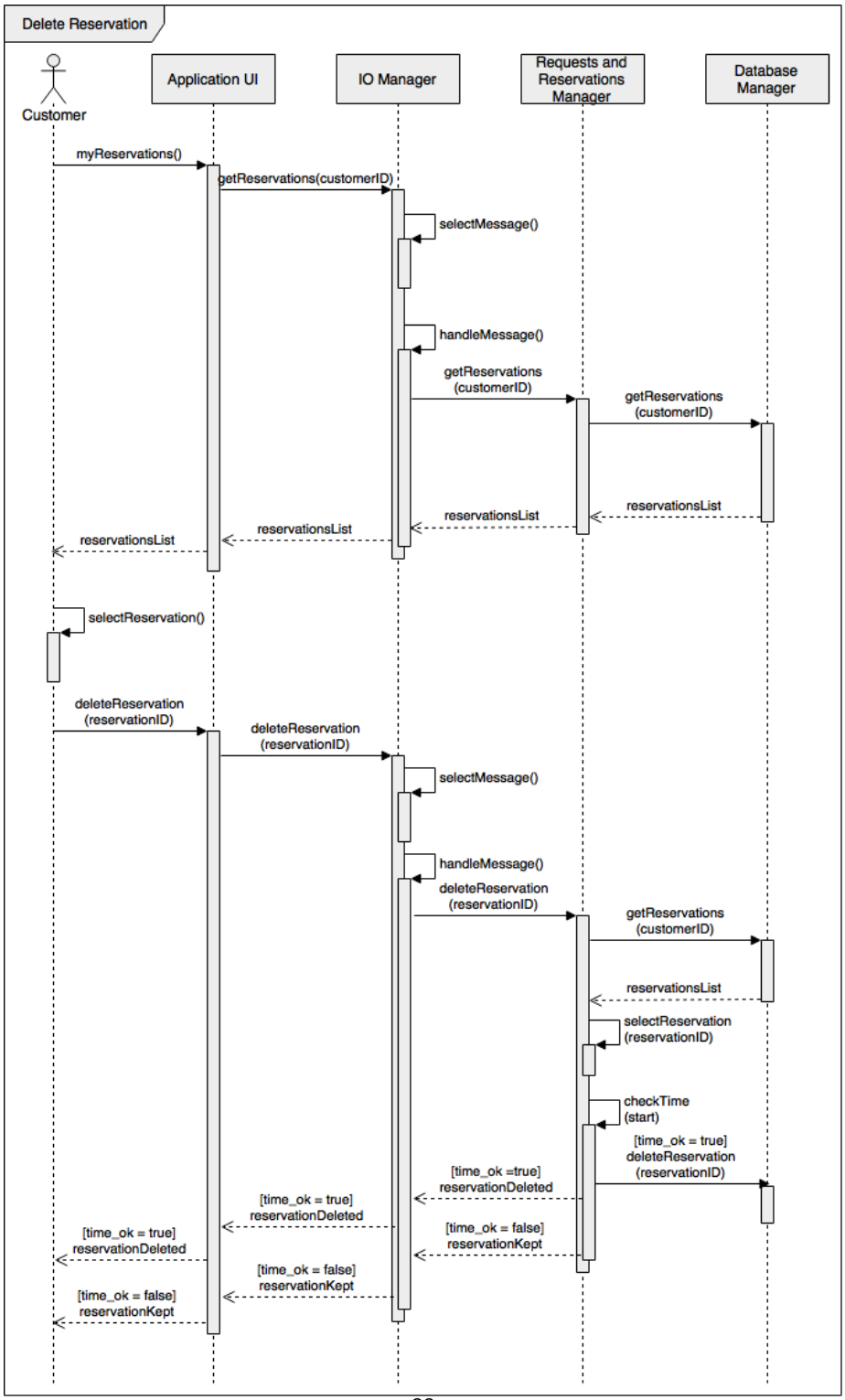

Figure 2.20: Delete one Reservation Runtime View

2.5.13 Accept Or Decline a Ride Request

Runtime View relative to the Scenario S.12 of the RASD.

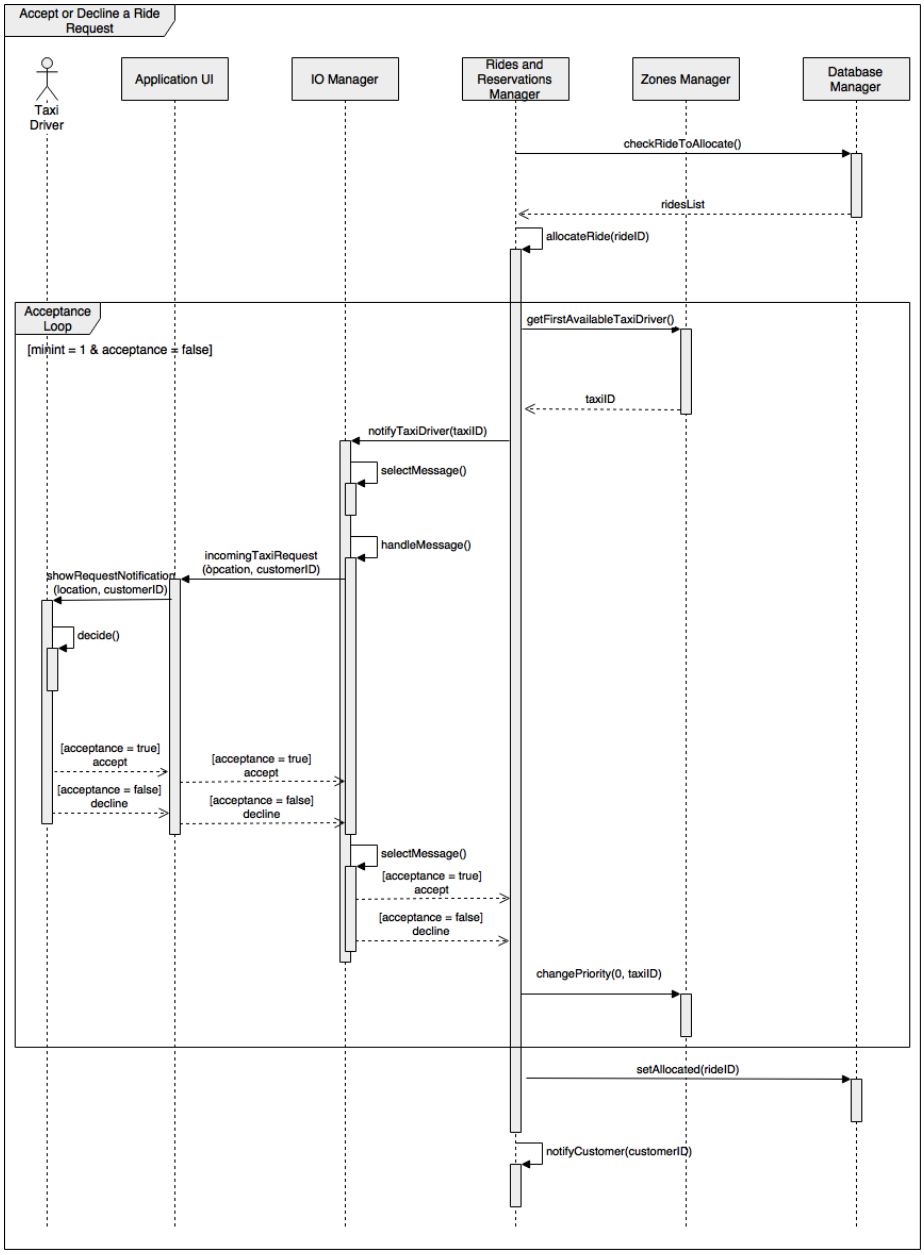

Figure 2.21: Acceptance or Declination of a Ride Request Runtime View

2.5.14 Taxi Driver Availability Notification

Runtime View relative to the Scenario S.13 of the RASD.

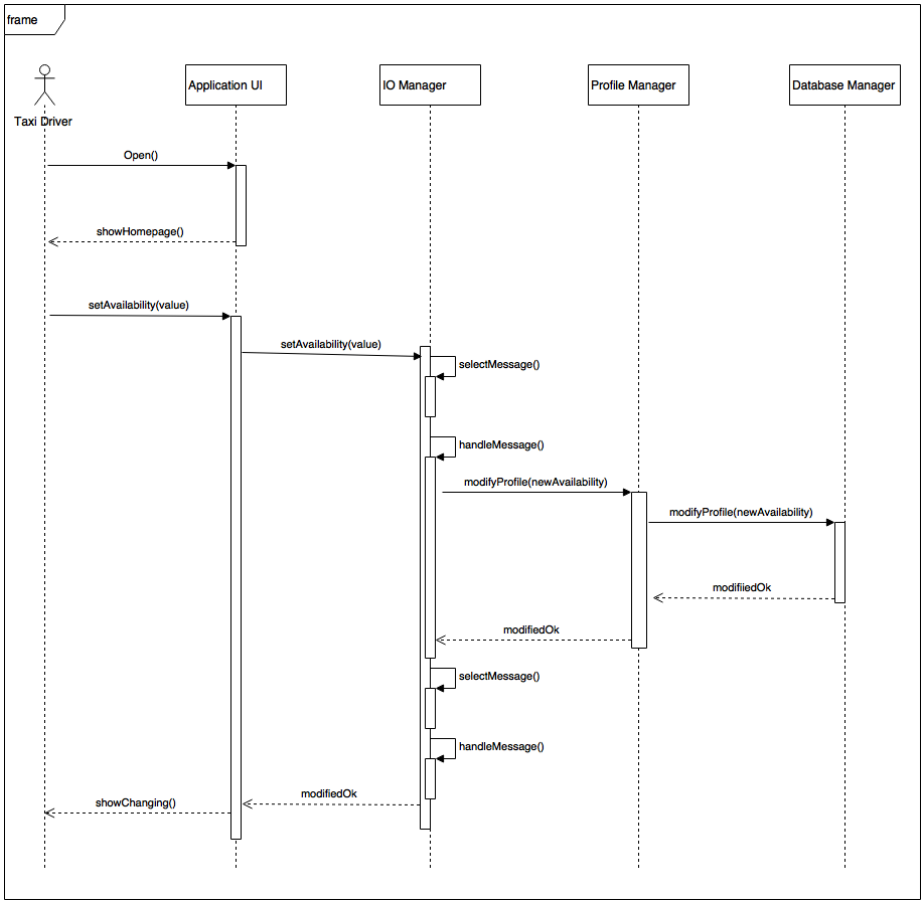

Figure 2.22: Availability Modification Runtime View

2.6 Component Interfaces

In this section are described the interfaces of all the components of the system. The interfaces describe all the public methods that can be called to access to the components functionality.

2.6.1 Database Manager Component

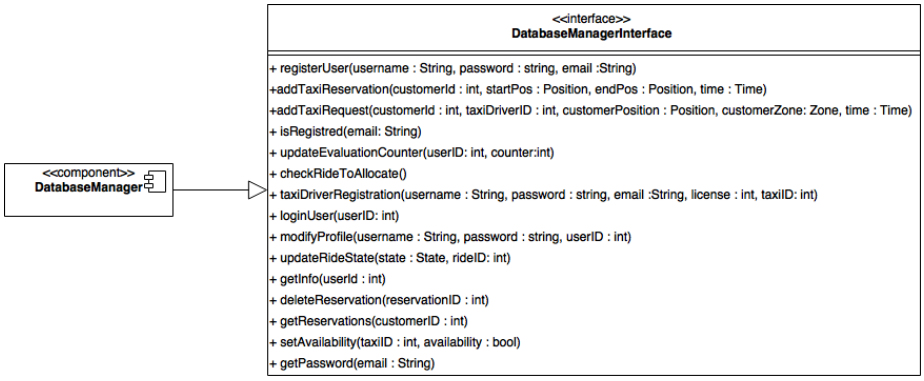

Figure 2.23: Database Manager Component

2.6.2 Profile Manager Component

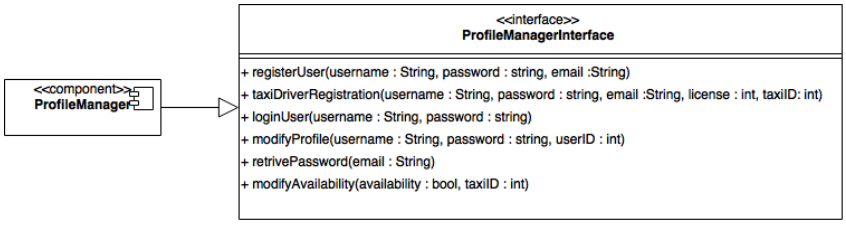

Figure 2.24: Profile manager component

2.6.3 IO Manager Component

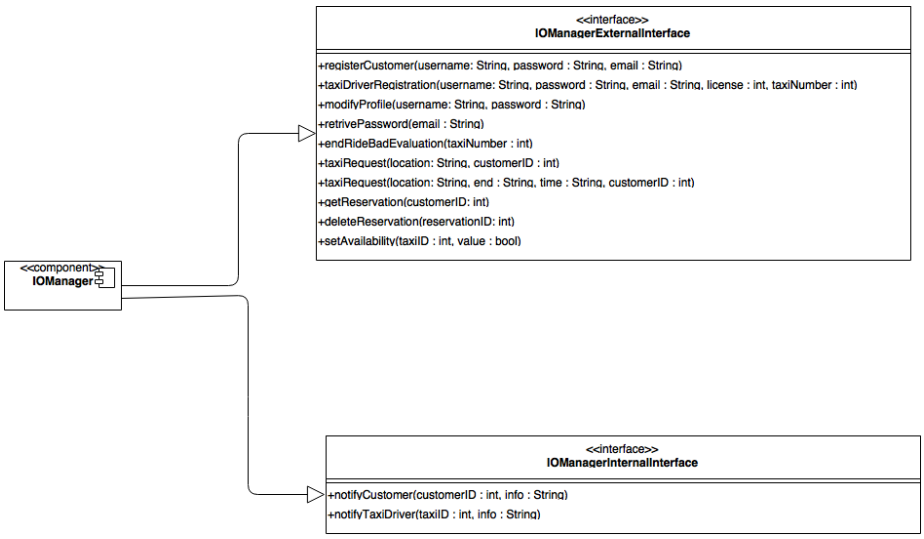

Figure 2.25: IO Manager Component

Note: In this case there are two interfaces because this component is a borderline one and needs two ways to be accessed. The Internal interface contains the public method that can be called by the component of the Logic tier. The external Interface can be called by all the external tiers.

2.6.4 Request And Reservation Manager Component

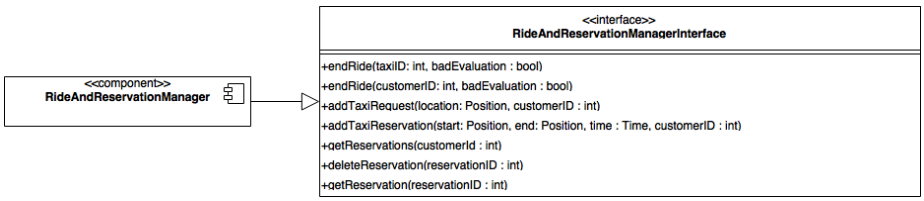

Figure 2.26: Reservation and Request Manager Component

2.6.5 Zone Manager Component

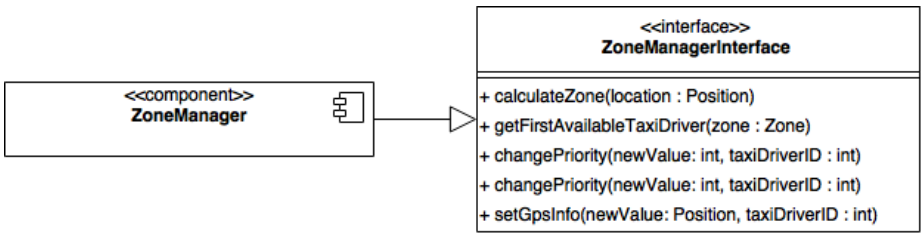

Figure 2.27: Zone Manager Component

2.6.6 Presentation Tier Component

All the components that are presented in the Presentation tier have no specific interfaces, the interfaces change for the type of visualization. In this document, we refer to their methods in an intuitive way. For example, if we want to open a page, we call the Open() methods, that will response with a Show() method followed by the page that is required.

2.7 Selected Architectural Styles and Patterns

Here are described all the main decisions about Architectural Styles and Patterns.

2.7.1 Architectural Styles

- **Distribution System:** For the distribution of the system has been selected a Client-Server (3 tier) Architectural Style. All the information about this architectural style are in the "Multitier Architecture" page of Wikipedia (link).
- **Structure:** The selected structure for myTaxiService System is a Layered Structure (Presentation-Logic-Data) and each Layer has a Component-Based Architecture, in fact, each Layer is divided in Components and Sub-Components to ensure the system modularity and scalability.
- **Messaging:** To manage messages between Clients and Server is used a Message-Oriented Middleware that is implemented in the IO Manager in the Logic Tier. This architectural style ensures that the messages are handled in the right way and that the logic is insulated from the client application.

2.7.2 Patterns

- The main pattern used is the **MVC pattern**. We have chosen this pattern because of all the advantages it involves. For example the separation between

the three main layer makes the system compatible with possible future changing. The fact that the View is separated from the rest of the system gives the possibility to implement different kind of UI, that is required for this system. Moreover, the separation of the three part of the system makes possible a Parallel development that speeds-up the creation of the system, and so on.

- The previous pattern is implemented on a **three-tier architecture**. This architecture splits the three parts of MVC on three different tiers. The Model part is implemented in the Data Tier, the Control part is implemented in the Logic Tier and the view part is implemented in the Presentation Tier. This pattern is the best choice if you choose to implement a MVC pattern.

- The IO Manager, implements a **Command Pattern**. We have chosen this pattern to handle the command sent to the server because it makes the communication simpler and makes the system scalable. For example if we want to implement a new command that the user can do, we just add a new command class and make it available for the Message handler class. At this point, the previous system is still working and, if the user wants to use the new functionality, he should only update his front end application.

- The Ride class implements the **State Pattern**. We have chosen this pattern because of the nature of the class that changes different states in his lifecycle.

2.8 Other Design Decisions

All the main Design Decisions are well explained in the previous sections.

Chapter 3

Algorithm Design

This section is intended to explain in a better way some functions and algorithms that can be ambiguous in the previous presentation of the system.

3.1 IO Manager Algorithms

Here are described some functions and algorithms relative to the IO Manager component.

3.1.1 Select Message

The select message procedure has to select the right message handler for an incoming/outgoing message. It's described by the following pseudocode:

Algorithm 1 Select Message Procedure

1: **procedure** SELECTMESSAGE($rawMessage$) ▷ rawMessage is an xml (or other markup language like json) message
2: $messageHandler \leftarrow messagesTypeMap.getMessageHandler($
3: $rawMessage.type)$
4: $returnmessageHandler.handleMessage(rawMessage)$
5: **end procedure**

Note on handleMessage(): every single message class, that is presented on the IO Manager component, can handle incoming messages or outgoing messages and the implementation of handleMessage() for each of this sub-components is different depending on the message that has to be handled.

3.2 Requests and Reservations Manager Algorithms

Here are described some functions and algorithms relative to the Requests and Reservations Manager component.

3.2.1 Check Rides To Allocate

The checkRidesToAllocate procedure is invoked by a scheduler every 2 minutes. This procedure has to check if there are rides or reservations that have to be handled and to put them into the ride-handling stack.

Algorithm 2 Check Rides to Allocate

```
1: procedure CHECKRIDESTOALLOCATE
2:     tempList ← DatabaseManager.getReady()  ▷ get all the rides that need to be handled
3:     for all element in tempList do
4:         allocateRide(element)                ▷ handle the single ride
5:     end for
6: end procedure
```

3.2.2 Allocate Ride

Allocate Ride starts a new Thread to perform the handling in parallel with the others operations.

Algorithm 3 Allocate Ride

```
1: procedure ALLOCATERIDE(ride)
2:     taxiID ← 0                                ▷ initialization
3:     repeat
4:         taxiID ← ZonesManager.getFirstAvailableTaxiDriver()
5:         msgTD ← createMessage("TaxiDriverNotif", taxiID)
6:         response ← IOManager.selectMessage(msgTD)
7:         ZonesManager.changePriority(0, taxiID)
8:     until response == decline
9:     DatabaseManager.setAvailability(taxiID, false)
10:    msgC ← createMessage("CustomerNotif", ride.CustomerID)
11:    IOManager.selectMessage(msgC)
12: end procedure
```

3.3 Zones Manager Algorithms

Here are described some functions and algorithms relative to the Zones Manager component.

3.3.1 Check Positions

This procedure is invoked by a scheduler every 2 minutes and updates the zones of every Taxi Driver.

Algorithm 4 Check Positions

```
1: procedure checkPositions
2:     availableDrivers ← DatabaseManager.getAvailableTDs()
3:     for all dirver in availableDrivers do
4:         driverZone ← ZoneCalculator.calculateZone(
5:         driver.position)
6:         driver.setZone(driverZone)
7:     end for
8:     orderQueues(availableDrivers)
9: end procedure
```

3.3.2 Order Queues

This procedure has to order the queues in function of the priority of the Taxi Drivers.

Algorithm 5 Order Queues

```
1: procedure orderQueues(drivers)
2:     QueuesManager.resetQueues()
3:     setCurrentDrivers(drivers)
4:     for all driver in drivers do
5:         zoneQueue ← QueuesManager.getQueue(driver.zone)
6:         zoneQueue.orderedInsert(driver)  ▷ insert the driver ordering by the
   priority
7:     end for
8: end procedure
```

3.3.3 Modify Priority

This procedure has to modify the priorities of the current queued Taxi Drivers and reinsert the driver in his queue tidily.

Algorithm 6 Modify Priority

```
1: procedure modifyPriority(new, taxiID)
2:     driver ← getFromCurrentDrivers(taxiID)
3:     driverQueue ← QueueManager.getQueue(driver.zone)
4:     driverQueue.remove(taxiID)            ▷ remove from the current zone
5:     driver.setPriority(new)
6:     driverQueue.orderedInsert(driver)                ▷ reinsert in order
7: end procedure
```

3.4 Profile Manager Algorithms

Here are described some functions and algorithms relative to the Profile Manager component.

3.4.1 Check Email

This procedure is in charge to tell if the given email is a valid one.

Algorithm 7 Check Email

```
1: procedure checkEmail(emaik)
2:     regex ← "[a − zA − Z1 − 9] + @[a − zA − Z]?.[a − zA − Z]2, 3"
3:     if regex.match(email) then
4:         return true
5:     end if
6:     return false
7: end procedure
```

3.4.2 Check Password

This procedure is in charge to tell if the given password is a valid one.

Algorithm 8 Check Password

```
1: procedure checkPassword(password)
2:     length ← password.length()
3:     numbers ← containsDigits(password)
4:     letters ← containsCharacters(password)
5:     return ((length > 8) and numbers and letters)
6: end procedure
```

3.4.3 Check Username

This procedure is in charge to tell if the given username is a valid one.

Algorithm 9 Check Username

```
1: procedure checkUsername(username)
2:     return (username.length() > 8)
3: end procedure
```

Chapter 4

User Interface Design

We have already described all the main interfaces that the application can show on the chapter 3.1.1 of the RASD. In that chapter we have described also the input and the action that each user can do, and shown a possible graphic layout of the pages using some mock object. In this chapter we will focus on the navigation between this pages and we will describe in a more specific way the input form. To do that we use a UX diagram UML, with which we give a global overview of all the possible pages with the relative forms.

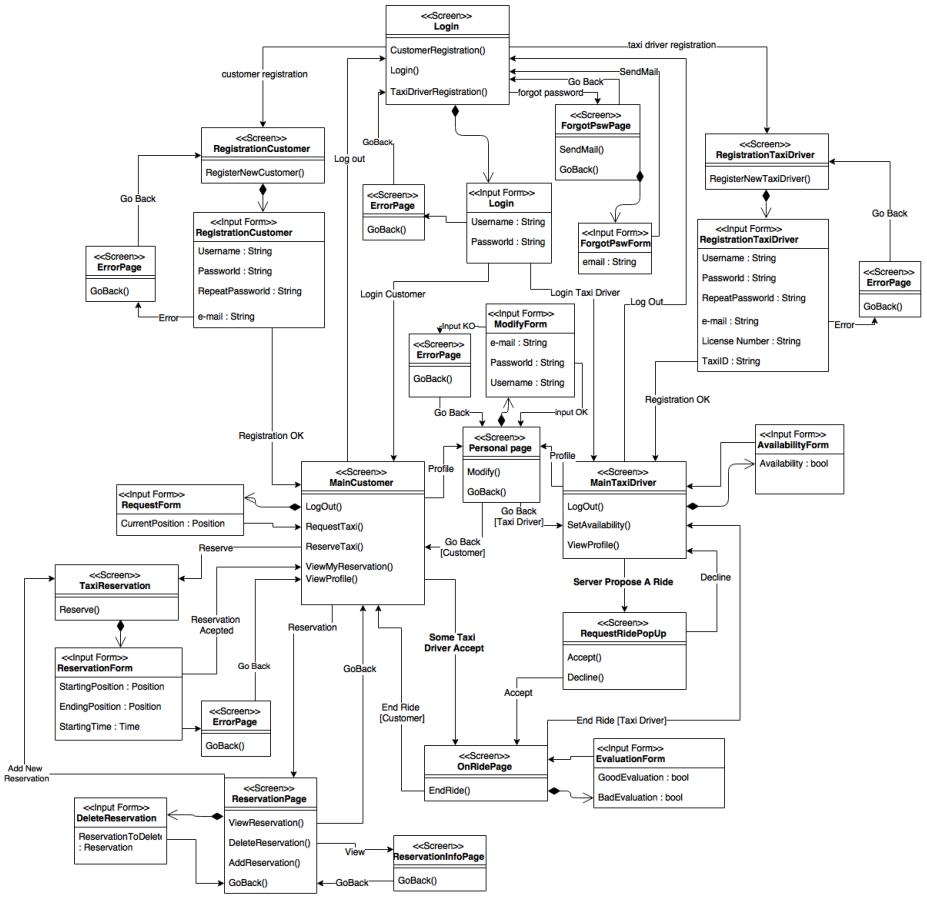

Figure 4.1: UX Diagram UML

Note: We use [Customer] or [TaxiDriver] on the transition to identify the return page if you are respectively logged as a customer or a taxi driver. We do this because the incoming pages have no differences if you are logged as a customer or as a taxi drivers. Note: The transitions with the bold text are transitions that do not depend on the input of the user. So, this kind of translations are, for example, pop up that are showed when on the system something has happened. E. g. a taxi driver that is on his main page can pass on the RequestRidePopUp screen without make any move, simply the pop up is shown when the system wants to ask to the taxi driver if he wants to take that ride.

Chapter 5

Requirements Traceability

In this section, we analyse every single Functional Requirement (ordered by Goals described in the RASD) and it's indicated in which components these requirements are implemented.

All the Goals (G.x) are well described in the RASD at section 3.2

5.1 G.1

With the Goal G.1, we want to allow Guest Users to become Customers by creating a new myTaxiService account. Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Provide a Registration Page with: 1) A text box where the user can insert his username. 2)Two text boxes where the user can insert his password (the second one is for security check). 3) A text box where the user can insert his email. 4) A button to submit informations to the system.	-	-	Application UI
[Req.2] Registration Page only accessible by Guest Users	-	-	Application UI
[Req.3] All the Informations submitted by the user must be validated by the system	-	Profile Manager	-

Table 5.1: Goal 1 Requirements Traceability

5.2 G.2

With the Goal G.2 we want to allow a Guest User to become a Customer by using his facebook Account. Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Provide a Registration Page containing: 1) A button that calls Facebook Login API	-	-	Application UI
[Req.2] The Registration Page shall be the only page accessible by a Guest User.	-	-	Application UI
[Req.3] All the informations must be evaluated by the Facebook System.	-	Facebook Login System	Application UI

Table 5.2: Goal 2 Requirements Traceability

5.3 G.3

With the Goal G.3 we want to allow a Guest User to become Taxi Drivers. Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Provide a Registration Page containing: 1) A text box where the user can insert his username. 2) Two text boxes where the user can insert his password (the second one is for security checks). 3) A text box where the user can insert his email. 4) A text box where the user can insert a valid taxi license. 5) A text box where the user can insert a valid taxi number. 6) A button to submit informations to the system.	-	-	Application UI
[Req.2] The Registration Page shall be the only page accessible by a Guest User.	-	-	Application UI
[Req.3] All the informations submitted by the Guest User must be validated by the system	-	Profile Manager	-

Table 5.3: Goal 3 Requirements Traceability

5.4 G.4

With the Goal G.4 we want to allow a Guest User to login with a myTaxiService Account. Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Provide a Login Page containing: 1) A text box where the user can insert his username 2) A text box where the user can insert his password 3) A button to submit informations to the system.	-	-	Application UI

Table 5.4: Goal 4 Requirements Traceability

5.5 G.5

With the Goal G.5 we want to allow a Guest User to login his Facebook Account. Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Provide a button that calls the Facebook Login API.	-	-	Application UI
[Req.2] Delegate to the Facebook System all the checks about the existence of the user.	-	Facebook Login System	-

Table 5.5: Goal 5 Requirements Traceability

5.6 G.6

With the Goal G.6 we want to allow a Guest User to view or modify his username and email. Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Provide an Homepage that contains: 1) A button that, if clicked, shows the page that contains the Registered User's informations.	-	-	Application UI
[Req.2] Provide a page used to show Registered User's informations that contains: 1) A clickable label showing the username of the Registered User. 2) A clickable label showing the email of the Registered User. 3) A box with a label inside that shows how many times the Registered User has been reported as a bad user.	-	-	Application UI
[Req.3] The personal Registered User's page shall be only accessible after the login.	-	Profile Manager	Application UI

Table 5.6: Goal 6 Requirements Traceability

5.7 G.7

With the Goal G.7 we want to allow a Registered User to retrieve his password if he doesn't remember it. Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Provide a link, on the login page, that will show to the Registered User a field where he can write his email to allow the system to send him an email containing the password.	-	Profile Manager	Application UI

Table 5.7: Goal 7 Requirements Traceability

5.8 G.8

With the Goal G.7 we want to allow a Registered User to signal another one if he has made a bad use of the system. Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Check if the registered user is correctly logged in.	Database Manager	-	-
[Req.2] For Taxi Drivers, provide an end-ride page that contains: 1) A label showing the username of the Customer that he has just served. 2) A label showing the starting location of the current ride. 3) A label that asks to the Taxi Driver if he wants to report the Customer. 4) A button that ends the current ride without reporting the Customer. 5) A button that ends the current ride incrementing the Customer's Bad Evaluation Counter.	Database Manager	Requests and Reservations Manager	Application UI
[Req.3] When one of the Taxi Driver end-ride page button is pressed, the system must ensure that the page is not reachable anymore and the Taxi Driver must only access to his Homepage.	-	-	Application UI
[Req.4] For Customers, provide an end-ride page that contains: 1) A label showing the username of the Taxi Driver that has just served him. 2) A label showing the starting location of the current ride. 3) A label that asks to the Customer if he wants to report the Taxi Driver. 4) A button that ends the current ride without reporting the Taxi Driver. 5) A button that ends the current ride and increments the Taxi Driver's Bad Evaluation Counter.	Database Manager	Requests and Reservations Manager	Application UI
[Req.5] When one of the Customer's end-ride page button is pressed, the system must ensure that the page is not reachable anymore and the Customer must access only to his Homepage.	-	-	Application UI
[Req.6] The end-ride page shall be only accessible after the Registered User logs in.	-	Profile Manager	Application UI

Table 5.8: Goal 8 Requirements Traceability

5.9 G.9

With the Goal G.9 we want to allow customers to require a taxi. Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Provide an Homepage that contains: 1) A map showing the current location of the Customer. 2) A button that **calls the function of the system to require a Taxi** (called "require button")	Database Manager	Zones Manager; Requests and Reservations Manager	Application UI
[Req.2] The require-taxi page shall be only accessible by Customers and after they have performed the login as Customers.	-	Profile Manager	Application UI

Table 5.9: Goal 9 Requirements Traceability

5.10 G.10

With the Goal G.10 we want to allow customers to reserve a ride. Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Provide an Homepage that contains: 1) A button that, if clicked, shows the page used to reserve a ride.	-	-	Application UI
[Req.2] Provide a page used to reserve a ride that contains: 1) A map where the customer can select the starting location for his ride. 2) A map where the customer can select the ending location for his ride. 3) A field where the customer can insert the starting date for the ride. 4) A field where the customer can insert the starting time for the ride. 5) A button that calls the function of the system to reserve a ride (called "reserve button")	Database Manager	Zones Manager; Requests and Reservations Manager	Application UI
[Req.3] The page used to reserve a ride shall be only accessible after the Customer logs in.	-	Profile Manager	Application UI
[Req.4] Reserve button must be clickable only if the reserve date and time are at least two hours after the current time.	-	-	Application UI
[Req.5] The system must notify the Customer that a Taxi Driver has accepted his reservation at least 10 minutes before the ride.	Database Manager	Requests and Reservations Manager; IO Manager;	-

Table 5.10: Goal 10 Requirements Traceability

5.11 G.11

With the Goal G.11 we want to allow customers to delete a previous reservations.
Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Provide an Homepage that contains: 1) A button that, if clicked, shows the page that contains all the reservations made by the customer.	-	-	Application UI
[Req.2] Provide a page that contains all the reservations made by the customer with these rules: 1) All reservations that have a difference between the start time and the current time of less than two hours are not erasable. 2) Others reservations are erasable. 3) The page also contains a button to add a new reservation. If clicked, that button leads to the page used to reserve a ride (described in the Functional Requirement Req.3 of G.4)	Database Manager	Requests and Reservations Manager;	Application UI
[Req.3] The page that contains all the reservations shall be only accessible after the Customer login.	-	Profile Manager	Application UI

Table 5.11: Goal 11 Requirements Traceability

5.12 G.12

With the Goal G.12 we want to allow Taxi Drivers to accept or decline a ride request.
Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Notify the Taxi Driver (if he is available) that is at the top of the zone-queue that a new taxi-request is made (Req.4 of G.9).	Database Manager	Requests and Reservations Manager; IO Manager; Zones Manager	Application UI
[Req.2] On the notification screen, show two buttons: one for the acceptance and one for the declination of the taxi-request.	-	-	Application UI

Table 5.12: Goal 12 Requirements Traceability

5.13 G.13

With the Goal G.13 we want to allow Taxi Drivers to notify their availability. Here are the requirements for this goal:

Requirement	Components		
	Data Tier	Logic Tier	Presentation Tier
[Req.1] Provide an home page that contains: 1) An ON/OFF button that signals to the system that the Taxi Driver is Available/Not Available. 2) A map showing his current zone and position.	Database Manager	Profile Manager	Application UI
[Req.2] The home page should be accessible only if the Taxi Driver is logged in.	-	Profile Manager	Application UI

Table 5.13: Goal 13 Requirements Traceability

Chapter 6

References

6.1 Used Software

To create this document we have used some common softwares:

- For the latex we have used two different softwares:
  - Simone Deola: TexShop, provided with the MacTex package (link)
  - Davide Cremona: Sublime Text editor (link) with LaTeXTools (link) and the Basic Package of MacTex (link)
- For all the schema we have used the online graph maker draw.io (link)

6.2 External Document

- Description of MVC pattern: `https://en.wikipedia.org/wiki/Model?view?controller`
- Description of 3-Tier architecture: `https://en.wikipedia.org/wiki/Multitier_architecture`
- Description of Command Pattern: `http://www.oodesign.com/command-pattern.html`
- Description of State Pattern: `https://en.wikipedia.org/wiki/State_pattern`
- XMPP info : `https://en.wikipedia.org/wiki/XMPP`
- XML info: `https://en.wikipedia.org/wiki/XML`
- HTTP info: `https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol`
- Description on UML Sequence Diagram (Runtime View of this document): `http://www.uml-diagrams.org/sequence-diagrams.html`

- Description of UX Diagram (User Interface Design of this document): `http://flylib.com/books/en/2.141.1.79/1/`

6.3 Hours of work

- Cremona Davide (852365): 35 Hours
- Deola Simone (788181): 35 Hours