myTaxiService
-
Design Document

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

November 24, 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 better understand what is written in this document.
- **Document Structure:** here is explained the internal structure of this document, giving a fast 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 chieved by describing:

- The system architecture;
- Architecture hig-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 orded 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 request 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.

1.3.3 Abbreviations

Definitions Acronyms Abbreviations section

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 used 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 interacts with each other and the Design Patterns and Architectural Styles used to design them.
- Algorithm Design: here is described the most relevants 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 described 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 helps to understand 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 have to do to provide to the users the functions described in the RASD document. Finally, there will be described what 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 tier represents 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:

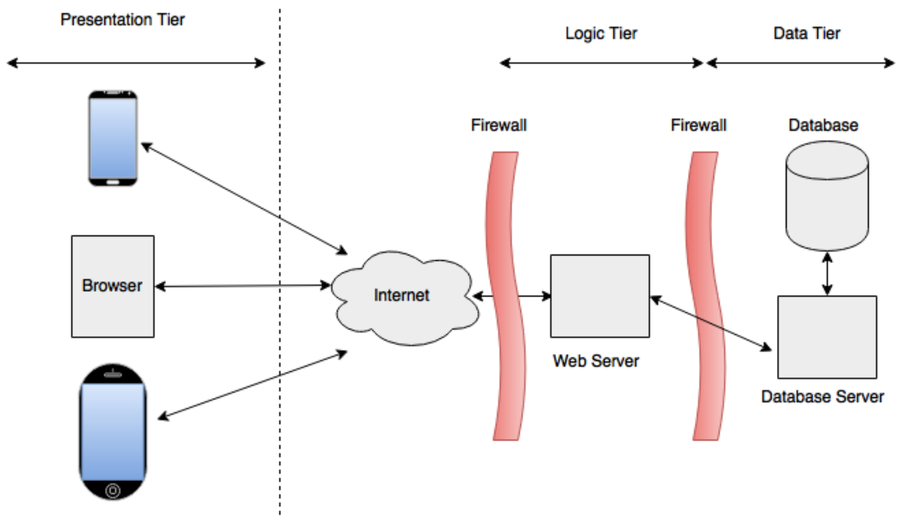


Figure 2.1: System Architecture

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

- In the Presentation Tier there are imoplemented classes and the objects that compose the view part (what the user see). From this tier users can perform 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 applciations 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 object that interacts directly with the database. These objects represents 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 others components.

2.2.1 Data Tier

Ther Data Tier is the part of the system that include 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 there 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 concerns requests for taxis and for reserve a future taxi ride. It's in charge of check the validity of the data that receives and, if it's the case, 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, then 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.

- **Facebook API Manager:** The Facebook API Manager is in charge to communicate with the Facebook system to perform Facebook Login and Registration. Once the transactions with the Facebook system is ended, then all the data are communicated to the Profile Manager as for a normal Login or Registration.

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 contain 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 it's provided a schema of the system:

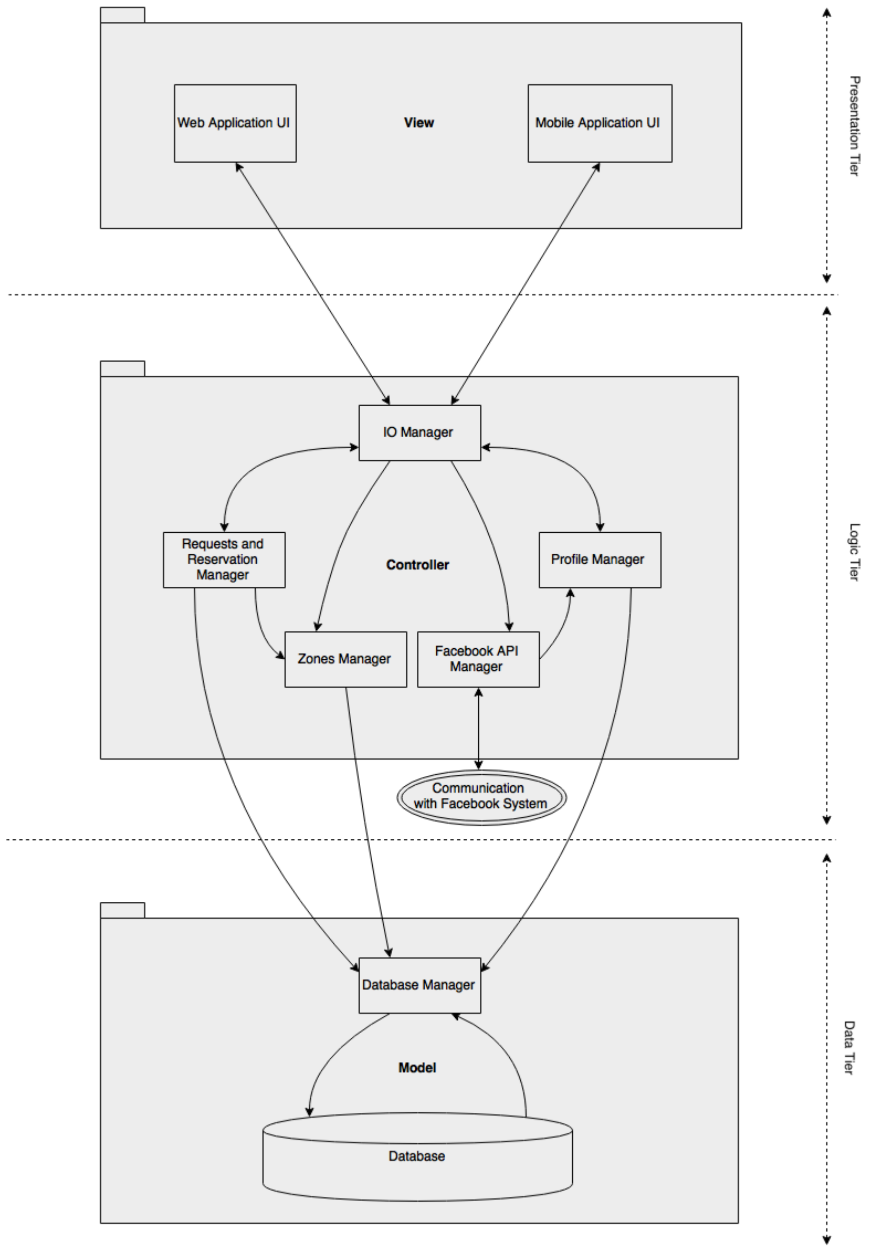


Figure 2.2: High Level Components and Interactions

2.3 Component View

"Per ogni componente individuato nella sezione precedente, definiamo le classi principali e le descriviamo."

On this section we take the component described in the previous section and give a more detailed description. For each component we describe the main classes to give a general overview of how the system is going to be implemented.

2.3.1 Data Tier

This part represent the model of the system (using a MVC architecture), so the class we use in this section was the representation of the information that must be stored to use the system.

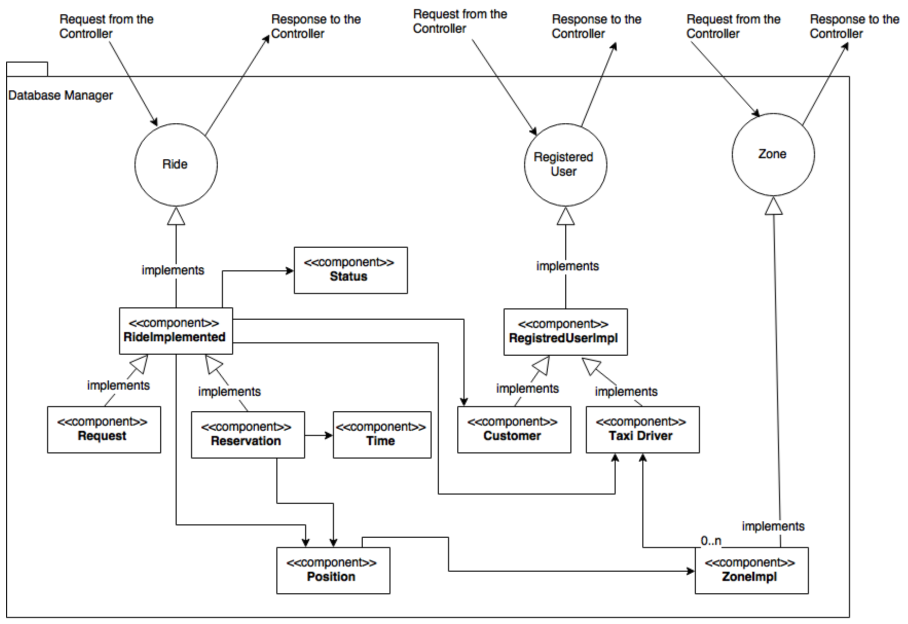
Database

the database is the data structure we don't specify the classes used by this structure, all the information we store are managed by the Database Manager.

Database Manager

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

- Registered User: this class contains the information about the registered user.
- Taxi Driver: this class extends the Registered User class and represent the information about the taxi drivers
- Customer: this class extends the Registered User class and represent the information about the customers
- Zone: this class represent the information about the zones. On this class we store also the queue of the taxi driver that are available on this specific zone.
- Ride: this class contain the general information that Ride and Request share. This abstract class follows the pattern State to represent the state of a ride. All Ride start in the state NOT_HANDLED, then can pass in state HANDLED, then from this state can pass in the state NOT_HANDLED.
- Request: this class extends Ride and represent the information of the requests for a ride.
- Reservation: this class extends Ride by adding information about the starting and the end position, and the time of the meeting. Adding this information we can represent the reservation in our system.



Request from the Controller
Response to the Controller
Request from the Controller
Response to the Controller
Request from the Controller
Response to the Controller
Database Manager
Ride
Registered User
Zone
implements
<<component>> Status
implements
<<component>> RideImplemented
<<component>> RegistredUserImpl
implements
implements
implements
implements
<<component>> Request
<<component>> Reservation
<<component>> Time
<<component>> Customer
<<component>> Taxi Driver
implements
0..n
<<component>> Position
<<component>> ZoneImpl

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, for the sake of brevity, 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 enum entry returns an implementation of the Message Type Interface.
- **Message Types Map:** this sub-component is an hashmap that associates the name of the elements of the enumeration, the enumeration element 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:**
- **Reservation Message:**
- **Customer Notification Message:**
- **TaxiDriver Notification Message:**
- **Login Message:**
- **Registration Message:**
- **Ok Message:**
- **Error Message:**
- **End Ride Message:**
- **Profile Modification Message:**

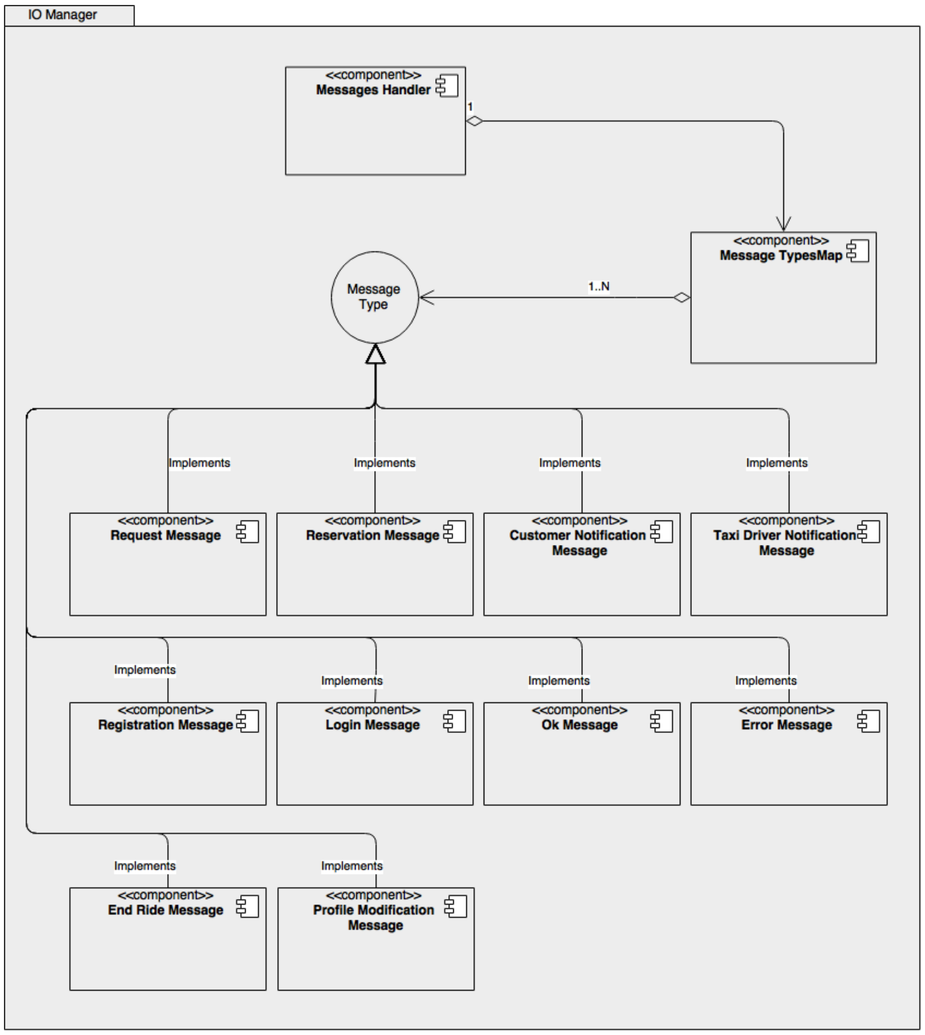

IO Manager
<<component>> Messages Handler
1
<<component>> Message TypesMap
1..N
Message Type
Implements
<<component>> Request Message
<<component>> Reservation Message
<<component>> Customer Notification Message
<<component>> Taxi Driver Notification Message
<<component>> Registration Message
<<component>> Login Message
<<component>> Ok Message
<<component>> Error Message
<<component>> End Ride Message
<<component>> Profile Modification Message

Figure 2.4: IO Manager Structure

Requests and Reservations Manager

Zones Manager

Profile Manager

Facebook API Manager

2.3.3 Presentation Tier

On this tier the two main components that we find are the Web Browser User Interface and the Application User Interface. This two components are different for programming language and it will run on different devices. But, this two component, 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 have two kind of classes, the first kind of classes are 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 kind of application. The second class of classes must contain a class that can sent message to the Logic Tier and one that can receive its responses. We call the first class CommandSender and the second one ResponseReciver.

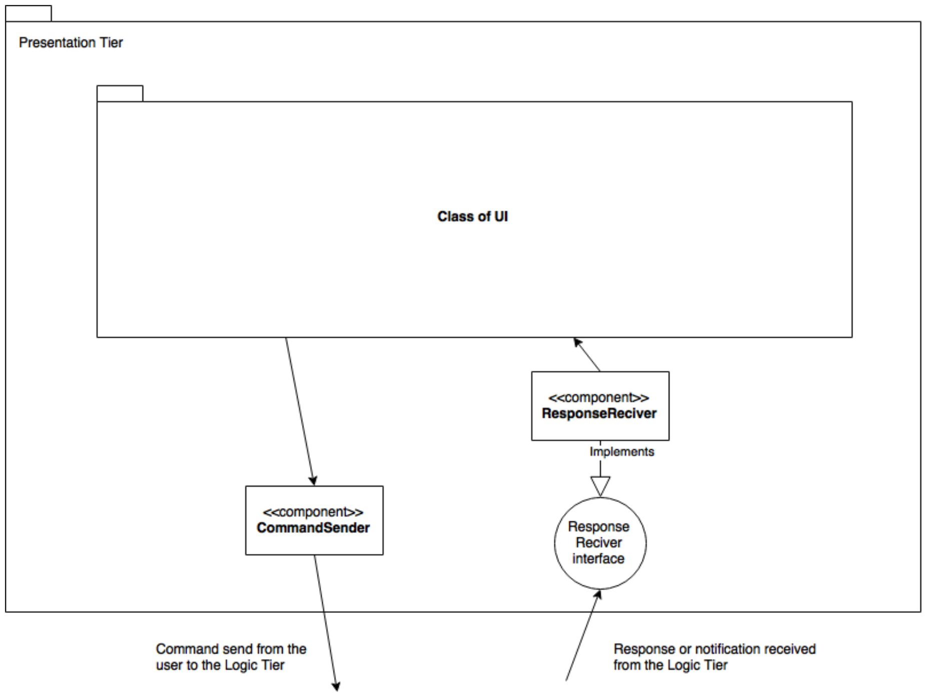

Presentation Tier
Class of UI
<<component>>
ResponseReciver
Implements
<<component>>
CommandSender
Response Reciver interface
Command send from the user to the Logic Tier
Response or notification received from the Logic Tier

Figure 2.5: Presentation Tier Structure

2.4 Deployment View

descrizione

2.5 Runtime View

Per ogni interazione tra i Package descritti nella High Level Components And Their Interaction descrizione

2.6 Component Interfaces

descrizione

2.7 Selected Architectural Styles and Patterns

Quali pattern abbiamo utilizzato? quali style architetturali? perchè?

2.8 Other Design Decisions

Pattern usati per database / hardware del sistema (3-tier)

Chapter 3

Algorithm Design

Algoritmi speciali usati nel nostro sistema, uno per uno descritti con pseudocodice o flow chart

Chapter 4

User Interface Design

Heyheyhey

Chapter 5

Requirements Traceability

I requisiti funzionali (nel RASD) come si tracciano all'interno del nostro sistema? I requisiti non funzionali come?

Chapter 6

References

Referenziamo ogni articolo, sito, libro eccc. esterno usato per scrivere il documento