Politecnico di Milano

Scuola di Ingegneria Industriale e dell'Informazione

M.Sc. in Computer Science and Engineering

myTaxiService

Software Design Document

Authors:

Angelo Gallarello

Edoardo Longo

Giacomo Locci

November 25, 2015

Contents

List of Figures

Abstract

This document provides a description of the conceptual framework of the system that will be implemented, MeteoCal, based on the specifications de- scribed in the document requirements analysis, the Requirement Analysis and Specification Document (RASD). It refers mainly to the working group that will be in charge of implementation and software maintenance. The first chapter provides an overview of the system and its architecture . In later chapters, it will be analysed in detail the design of the various ele- ments making up the application. First it will be presented the structure of the data, and then aspects related to the design of the functionality of the application, and finally the user interface design and logic connected. Note about the authors: as we tried to make the document independent of the implementation choices, we felt it necessary to refer these choices during the design phase . For this reason, where necessary, will be discussed implementation aspects relevant to the design phase .

The current document represents the *Software Design Document* (SDD) of *myTaxiService* system. This document is intended to be read and used by everyone involved with the development of the software (i.e. developers, testers, project managers) and the delegating institution (i.e. the government of the large city). Therefore its purpose is to provide the intended reader with a fulfilling guideline, that can be used as the main resource for the development of the system, and with a presentation of the challenges that will have to be addressed throughout the whole process.

An incredible amount of effort has been put into every section of this document; every aspect of the *system-to-be* has been thoroughly explored, meticulously analyzed and accurately studied so that every situation can be addressed in a unique way and can be interpreted without ambiguity.

In the following sections the system is broken down into its components, examining its functional and non-functional requirements. An in depth analysis of the relationship between the system and the external world is conducted throughout the document, stressing the assumptions that have been taken and highlighting the constraints that every requirement has to satisfy in order to have a functional and useful system.

The reader of this document will find UML class diagrams, sequence diagrams and use cases that provide all the needed information required to build the system. Furthermore, every use case is justified by the goals and assumptions that lead to the need of a specific use case.

Moreover, the delegating institution will find a reason for every

design choice taken and therefore is able understand the necessity of a specific feature.

This way, each and every reader will have a clear understanding of the *system-to-be*, by finding a justification for every decision taken concerning any aspect that is worth of considering for a successful development of the investigated system.

1 Overview

myTaxiService is a taxi service that will operate in a big city; the main purpose is to simplify the access of passengers to the service and to guarantee a fair management of the taxi queues.

The main stakeholders of the system are the *Users*, the *Taxi Drivers* and the *Operators* as highlighted in *section 1.3* of the *RASD*.

The system is composed of four main core applications :

- Mobile Application (User)
- Web Application
- Mobile Application (Taxi Driver)
- Back-End Application

as stated in *section 1.2.* of the *RASD* It's important to highlight that in this document the design of mobile application is based on the Android platform.

2 High Level Components

The system could be divide in three main high level components that do not necessarily correspond only to one real application:

Server

The Server component is the kernel of the service we want to provide, it incorporates most of the *business logic*, it stores most of the *data* and it provides programmatic interfaces to the clients.

User Client

The User Client components is an high level representation of the real clients available to the users of our service. It's modeled as a *thin client* and it relies on the *Server* to fulfill its tasks.

Taxi Driver Client

The Taxi Driver Client component is an high level representation of the real clients available to the taxi drivers registered to the service. It's modeled as a *thin client* and it relies on the *Server* to fulfill its tasks.

2.0.1 Components Interaction

From a high level perspective the system is design following the well known *client-server* paradigm.
The interaction between the components is handled by the Server that provides a programmatic interface that is able to receive remote call from the clients.
The clients never communicate directly with one another.

3 Component View

This section highlights the main features and roles of every component of the system. Moreover it describes the internal interfaces between different classes of every component.
External interfaces between components are described in *section 6*

3.1 Server

The Server is composed of:

Back-End Application

As stated in *section 1.2.2* of the *RASD*, the *Back-End Application* is the system component that handles most of the business logic.
The application is written in *Java EE* and to fulfill its tasks (see *section 3.5.3* of the *RASD*) it needs to interface with the Internet network using the *HTTPS protocol* and the *JAVA API for RESTful Web Service*[1], with a *MySQL database* and with external Google Maps API.

[1]See `https://jax-rs-spec.java.net/`

Back-End Internal Interfaces
The *Back-End Application* is built to be very modular and to grant *interchangeability* between components.
There are four main classes that constitute the kernel of the application :

- **QueueManager** Handles queue policies.
- **RideManager** Creates and manage rides. Is connected to the *RequestManager* via the *RideManagerInterface* and directly depends on *QueueManager*
- **ActorManager** Create and update data about users an taxi drivers. Is connected to the *RequestManager* via the *ActorManagerInterface*
- **PositionManager** Update taxi drivers position. Is connected to the *RequestManager* via the *PositionManagerInterface* and directly depends on *QueueManager*
- **RequestManager** Get and build *Request* object from the requests received via *HTTP*

MySQL Database

The MySQL database fulfill the task off storing and granting access to all the data generated and used by the service.
A *database dump* is performed daily during the period of minor activity of the service [2].
The connection between the *Java EE* application and the databased is supported by the *JDBC connector*[3]

3.2 User Client

Different real clients are available to the end users of the system.
As stated in *section 1.2.2* of the *RASD* a native mobile application is developed for Android, iOS, Blackberry and WP.
Moreover a Web Application is also available.
To fulfill the requirements expressed in *section 3.5.1* and *section 3.5.2* of the *RASD*, all the clients need to communicate with the Server making calls to the REST API using platform specific API for REST HTTP calls.

[2] At first, when no activity data is available, the dump will be performed at 04:00 A.M
[3] See `http://dev.mysql.com/downloads/connector/j/`

3.3 Taxi Driver Client

Different real clients are available to the taxi driver registered to *myTaxiService*.
As stated in *section 1.2.2* of the *RASD* a native mobile application is developed for Android, iOS, Blackberry and WP.
To fulfill the requirements expressed in *section 3.5.1* and *section 3.5.2* of the *RASD*, all the clients need to communicate with the Server making calls to the REST API using platform specific API for REST HTTP calls.

3.4 Clients Internal Interfaces

Mobile clients are composed mainly by subclass of platform specific components.
Interfaces between components are therefore specified in the SDK of each platform. However it's important to highlight that every mobile application has to interface with a *Network Component* that handles *HTTP* requests.

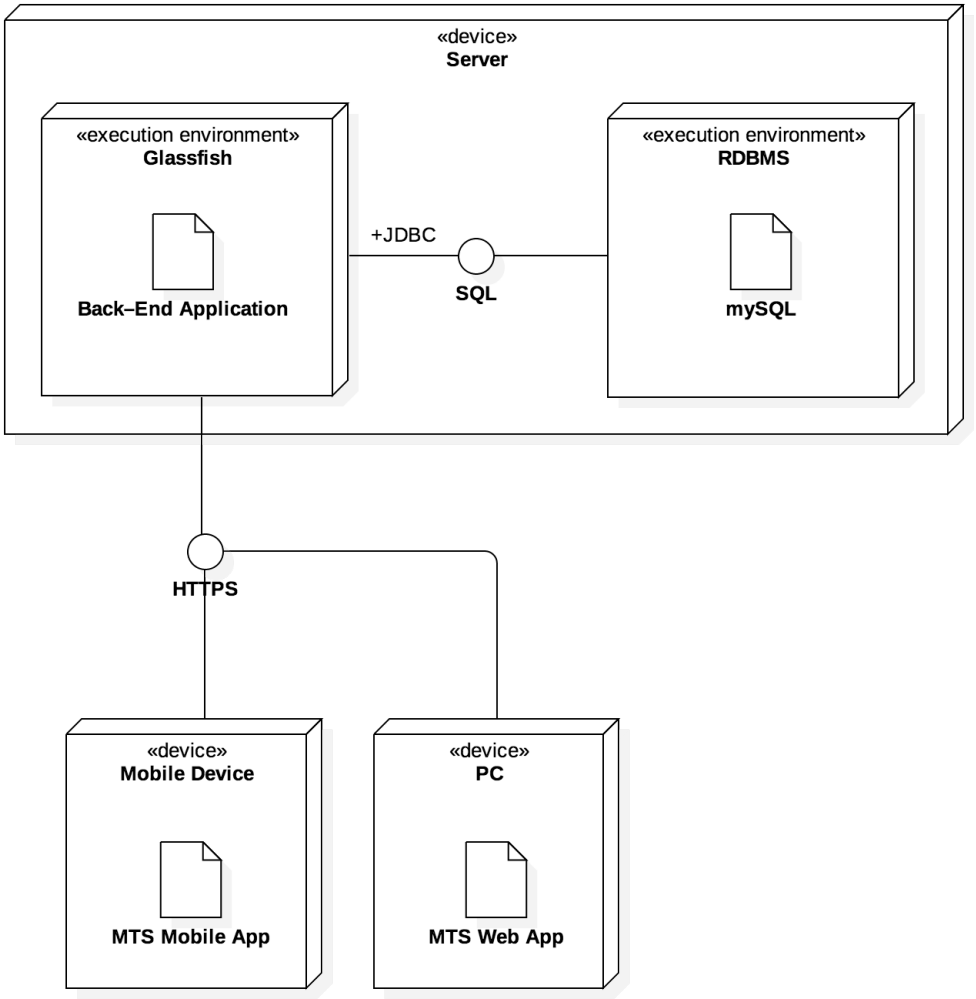
4 Deployment View

In this section we analyze the *Deployment View*, meaning that a presentation of the deployment point of view is provided.

4.0.1 Diagram

In order to have a successful deployment an in depth analysis of the main components that will have to be deployed is required. For this reason here is a diagram showing the *Deployment View*:

Figure 1: Deployment Diagram



4.0.2 Diagram Analysis

This diagram shows a two-tier architecture.

A main server is deployed: in this node the *mySQL DataBase* is executed. This is also where the *Back-End Application* will be deployed.

Both mobile applications, the *User* one and the *Taxi Driver* one, interface with the *Back-End Application* through *HTTP* protocol. Also the PC application, namely the *Web Application*, connects to *Back-End Application*.

5 Runtime View

5.1 Server View

In this section we focus on the *Runtime View* of the system.
While the system is up and running, the *Server* receives many *HTTP* requests from different clients that are handled by a *Load Balancing* component that distributes the calls uniformly to every real machine.
Every request from the users are registered by the server and saved in the *MySQL* database.
From the *internal* point of view of the *Back-End* application, request are at first parsed by the *Request Manager* and the dispatched to Java object that is in charge of computing the result of that request. Three diagrams are provided to better exemplify the flow of events.

5.2 Client View

From the client point of view, when a *User* or a *Taxi Driver* open his app, the client starts a first *"handshake"* to check for basic authentication data and if it's successful, the client can proceed with requests.
The flow of a request start from a *UI* component (like a *button*) and is finally handled by the class that implements the *HTTP* interface.

6 Components Interfaces

This section provides a description of the interfaces between the main components of the system.
Internal interfaces between different objects of every component are described in *section 3*

6.1 Back-End Application - Database

The *Back-End Application* uses *SQL* language to query the *Database*. Queries from *Java* are supported by the *Java Database Connectivity (JDBC) API* which is the *API is the industry standard for database-independent connectivity between the Java programming language and a wide range of SQL databases*[4].
This choice is made in order to exploit the *"Write Once, Run Anywhere"* feature of the *JDBC*.

6.2 Back-End Application - Client User

The connection between the *Back-End Application* and the *Client User*is provided by the Internet network and based on the *HTTP* protocol and supported by a *RESTful API* service.
The two components use different internal interfaces to connect to the network[5]

6.3 Back-End Application - Client Taxi Driver

The connection between the *Back-End Application* and the *Client Taxi Driver*is provided by the Internet network and based on the *HTTP* protocol and supported by a *RESTful API* service.
The two components use different internal interfaces to connect to the network.

[4]See `http://www.oracle.com/technetwork/java/javase/jdbc/index.html`
[5]See *section 3*