

**D**esign **D**ocument **(DD)**

Computer Science and Engineering (CSE)

Software Engineering 2 Project

Year 2015/16

*Date: 04/12/2015 – Version: 1.0*



**STUDENTS:**

*Martino Andrea (****788701****)*

*Marchesani Francesco (****852444****)*

**PROFESSOR:**

*Mirandola Raffaela*

# Table of Contents

1. Table of Contents 2

2. Introduction 3

2.1. Purpose 3

2.2. Scope 4

2.3. Definitions, Acronyms, Abbreviations 4

2.4. Reference Documents 4

2.5. Document Structure 6

3. Architectural Design 7

3.1. Overview 7

3.2. Component View 9

3.3. Deployment View 10

3.4. Runtime View 10

3.5. Component interfaces 10

3.6. Selected architectural styles and patterns 11

3.7. Other design decisions 13

4. Algorithm design 13

5. User Interface Design 13

6. Requirements Traceability 13

7. References 15

7.1. References list 15

# Introduction

## Purpose

This **Design Document** contains information about the **architecture** and other features of *myTaxiService design* such as **high-level** **algorithms** and **user interface** (with recalls from the previous **RASD**). In order to maintain a **modular** and **scalable** system for the future, we will enter in details only if strictly necessary (otherwise, we will keep a higher level of abstraction).

This **document** is coherent with *the official template* of the project on the *Beep platform*.

As we said for the **RASD**, it is important to underline that some parts of this document may evolve in the future (this may occurs for several causes).

Anyway, we will try to maintain coherence as much as possible.

Here is a resume of the steps of the project, with the related deadlines:

## Scope

The main scope of this **DD** (*Design Document*) is to give an overall guidance to the **architecture** of the **project**, which is *myTaxiDriver* (**Software Engineering 2 project** of year 2015/16 - **Politecnico di Milano**).

We described the main **goals** and **objectives** of the project in the previous *Requirements Analysis and Specification Document*.

## Definitions, Acronyms, Abbreviations

* **DD:** *Design Document*
* **RASD**: *Requirements Analysis and Specification Document*
* **mTS**: *myTaxiService*
* **JEE**: *Java Enterprise Edition*
* **UML**: *Unified Modelling Language*
* **Servlet**: A servlet is a program that extends the capabilities of a server.
* **Layer**: logical level of the architecture.
* **Tier**: physical level of the architecture.

**Note:** *for the full Glossary may be helpful to see also the paragraph 1.5 of the RASD.*

## Reference Documents

The main reference document is the **Requirements Analysis and Specification Document (RASD)** of *myTaxiService*.



*Preview of the Requirements Analysis and Specification Document (RASD), our reference document.*

## Document Structure

The **Document Structure** completely follows the template hosted on the *Beep Platform* (PDF file *DD TOC – Design Document Template*).

# Architectural Design

## Overview

We start from a big consideration: *myTaxiService* is a big project. As we have seen before, in the RASD, we have a lot of potential users and a big amount of requirements.

Thus, we considered to develop our application using **Java Enterprise Edition** (**JEE**).

This will also be useful to satisfy important *Non-Functional Requirements* such as scalability, portability, availability, reliability and so on.

The applications of *myTaxiService* (the web application and the mobile one) will be *large-scale*, *multi-tiered, scalable, reliable* and the network will be *secure*.

The application developing takes as reference the standard of **Java Enterprise Edition 7** (**JEE7**), the last release available now.

We will use a *three-tier physical architecture* mapped on *four logical layers*, as the standard of JEE.

Here is the general schema of the **architecture**:



Now we will see in a deeper level of detail the meaning of each layer:

* **Client Layer**: it contains Application Clients and Web Browsers and interacts directly with the actors (Customers and Taxi Drivers). In our application, the Client can access via browser (*web application*) or via smartphone (*mobile application*).
* **Web Layer**: it contains the *Servlets* and *Dynamic Web Pages* that needs the elaboration. This tier receives the requests from the *Client* *layer* and forwards the pieces of data collected to the *Business Layer*.
* **Business Layer**: it contains the application logic (with the *Java Beans* and the *Java Persistence Entities)*. This will permit the communication between the System of mTS and the target users (Costumers and Taxi Drivers).
* **EIS (Data Layer)**: it will contain all the *data* concerning Costumers, Taxi Drivers, Calls, Reservations and other useful information. It is crucial to manage data according to strict policies about security and privacy.

It is important to underline that *myTaxiService* will also use external pre-built software products from the point of view of the business logic, such as **Google Maps** for the maps and the **online payment services API** (*e.g. PayPal*).

Here is a schema of the **three physical levels** (*tiers*) of the architecture:



**Note:** *It is important to underline that both the Web Layer and the Business Layer are mapped on the same physical machine in our architecture (the Java EE Server).*

## Component View

According to JEE Standard, in order to develop a modular system that can evolve in future without big problems we decided to make use of a *Component-based approach*.

The formal definition of component is the following one:

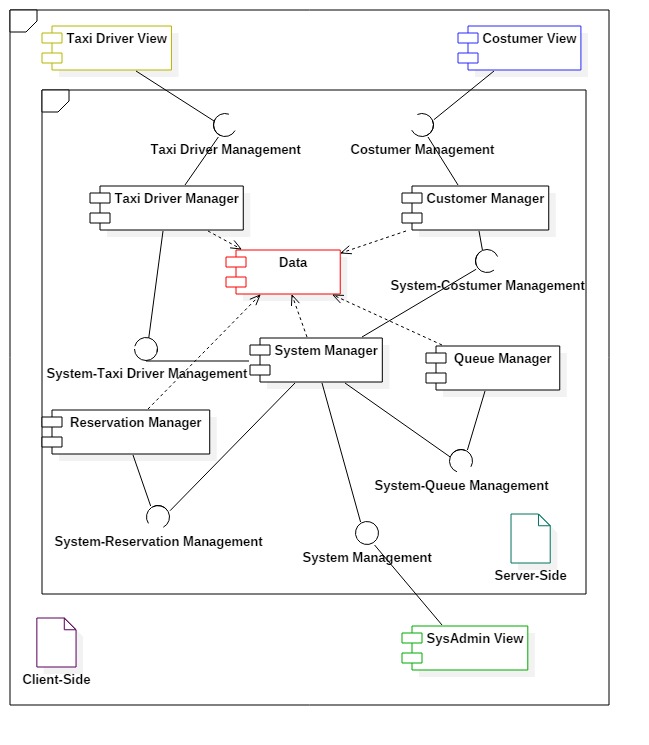
*A* ***Component*** *"represents a modular part of a system, that encapsulates its content and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces"*

We selected the main components of *myTaxiService* project with respect to the architecture of *Java Enterprise Edition*. From the point of view of the *logical architecture*, we will have:

* **Client Components**
* **Web Components**
* **Business Components**
* **Enterprise Information System (EIS) Components**

**Note:** as said before, the *Web Components* and *the Business Components* will be embedded in one physical machine (*JEE Server*).

This is the **UML Component** **Diagram** related to *mTS* project:



## Deployment View

## Runtime View

## Sequence Diagrams Here

## Component interfaces

## Selected architectural styles and patterns

### MVC (Model-View-Controller)

**Model-View-Controller** (also known as **MVC**) is one of the most diffused *structural pattern* in the applications with interfaces. It separates three logical parts: the **Model**, the **View** and the **Controller**. These parts interact each other according the following general schema:



*Model-View-Controller general schema*

In the specific case of *myTaxiService* these three logical parts will have the following *roles* and *interactions*:

* The **Controller** sends commands to the Model to update the Model's state (e.g., editing tuple in mTS Database with a new reservation). It must also send commands to its associated view (the Taxi Driver or the Web/Mobile Costumer one) to change the View's presentation of the Model (e.g., by scrolling through a reservation form).
* The **Model** stores data that is retrieved according to commands from the Controller and displayed in the View. It contains all the data about reservations, Costumers, Taxi Driver and son on.
* The **View** generates an output presentation to the user based on changes in the Model. There will be three different types of View: *Mobile Costumer View, Web Application Costumer View and Taxi Driver View*.

### State pattern

The **State pattern** is a behavioral software design pattern, also known as the **objects for states pattern**. This pattern is used in computer programming to encapsulate varying behavior for the same object based on its internal state. Here is a general schema in *UML*:



*State pattern general schema*

In the case of *myTaxiService* this pattern can be very helpful with the management of the **Taxi Drivers**. In fact, a Taxi Driver can be in different states from the point of view of the System.

Let us focus on the possibilities. The possible **states** of a Taxi Diver are:

* *At work – Available at the moment*
* *At work – Not available at the moment*
* *Not at work*

This will take trace of the status of any Taxi Driver during the whole day and can be used to compute the queue on the base of the available Taxi Drivers.

## Other design decisions

# Algorithm design

## Queue Management: M/M/s model

In order to explain the management of the queues it is important to recall the basics of the **M/M/s** model of **Queue Theory**.

We will not enter in details about the **implementation** of the model in our system with the source code of the **algorithm**, because it will only be a constraint for the successive phases of developing.

In fact, the **Design Document** must have a *higher level of abstraction*.

Anyway, it is good to give an idea of the Queue Management, with the right terminology and the basilar math notation. Here is a high-level schema of *myTaxiService* Queue Model:



**Model Assumption**: *interarrival times* and *service times* are identically distributed according to an *exponential distribution.*

Now let us define the probabilistic variables and constants of this model:

* **State of system** = *number of customers in queueing system.*
* **Queue length** = *number of customers waiting for service to begin.*
* **N(t)** = *number of customers in queueing system at time t (t ≥ 0).*
* **Pn(t)** = *probability of exactly n customers in queueing system at time t, given number at time t0 = 0.*
* **s** =*number of servers (parallel service channels) in queueing system.*
* **λn** = *mean arrival rate (expected number of arrivals per unit time) of new customers when n customers are in system.*
* **µn** = *mean service rate for overall system (expected number of customers completing service per unit time) when n customers are in system. Note:* ***µn*** *represents combined rate at which all busy servers (those serving customers) achieve service completions.*

# User Interface Design

# Requirements Traceability

|  |  |  |
| --- | --- | --- |
| Requirement | Component | Description |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# References

## References list

Here is a short list of the **references** for this **Design Document**:

* *Slides of the Software Engineering 2 course (from the* ***Beep Platform****)*
* *Design Document Template* *(by Prof. Raffaela Mirandola)*
* *Software Engineering: Principles and Practice (Hans Van Vliet)*