MyTaxiService

# Design Document

Dushica Stojkoska
Mite Ristovski

Version 1.1
December 2015

# Contents

# 1 Introduction

## 1.1 Purpose

Main goal of this document is to provide the overall system architecture of MyTaxiService application, which is a part of the project of Software Engineering 2 course at Politecnico di Milano. This document will describe technical decisions of the design process and its justifications.

It will be shown how the structure of the software system will satisfy the requirements previously defined in Requirements Analysis and Specification Document, also serving as an input for the next phase of the development process of the system.

The intended audience of this document is all the people actively participating in the software engineering course, including professors and tutors.

## 1.2 Scope

MyTaxiService is intended to be an application which will ease access of passengers to the taxi service in the city. MyTaxiService will allow users, or the passengers, to make requests for taxi services, as well as reservations of taxi for specific time periods. The system will provide fair distribution and management of taxis for the passengers, through automatical computation of distribution of taxis. This distribution of taxis will be based on GPS information each taxi sends to the system. Taxi drivers will be able to confirm or decline passenger's request. After confirmation of drivers for a request, the system will send a notification to the passenger, containing coresponding taxi code and the waiting time. Moreover, the system will offer programmatic interfaces - API, to enable development of additional services to the basic ones available in the application.

This document presents high level description of the components involved and how they interact.

## 1.3 Definitions, Acronyms, Abbreviations

| Keyword | Definition |
|---|---|
| ***User*** | All registered users in the system, more specifically: administrators, passengers and taxi drivers. |
| ***Administrator*** | Administrator has a role to manage the system and register taxi drivers to the system. |
| ***Passenger*** | Users that request the service of the system. |
| ***Taxi driver*** | Users that enable the services, they accept or decline requests from the passengers. |
| ***Request*** | Asking for available taxi. |
| ***Reservation*** | Asking for available taxi for a specific date and time period. |

Table 1: Definitions

| Acronym or abreviation | Definitions |
|---|---|
| ***RASD*** | Requirements Analysis and Specification Document |
| ***FR*** | Functional Requirement |
| ***NFR*** | Non-functional Requirement |
| ***DBMS*** | Database Management System |
| ***AS*** | Application Server |
| ***JEE*** | Java Enterprise Edition |

Table 2: Acronyms and abreviations

## 1.4 Reference Documents

- MyTaxiService Requirement Analysis and Specification Document (RASD)
- ISO/IEC/IEEE 42010:2011, Systems and Software Engineering Architecture Description

## 1.5 Document Structure

The document is organized as follows:

**Introduction**
This section describes the purpose of the project and the general functional requirements. It introduces the reader definitions, acronyms and abbreviations used in this document.

**Architectural Design**
This section describes the architectural design part, and provides information for the components.

**Algorithm Design**
A general description of main algorithms that should be used in development phase is provided in this section.

**User Interface Design**
In this section is represented how user interfaces of the software product will look like.

**Requirements Traceability**
In this part is explained how requirements and specifications previously defined in the RASD document are implemented into the design elements in this document.

**Appendix**

This part contains information about the tools used and the number of hours spent for making this document.

# 2 Architectural Design

## 2.1 Overview

This section provides detailed view over the system components.

MyTaxiService software product is intended to ease access of the users of taxi services. In order to fullfill its efficiency, MyTaxiService can be accessed from different platfroms. Users should be able to make requests from different locations, as well as reservations for different date and time periods. The system needs to support multiple users at a time, process all kinds of communication between users, elaborate answer in short time, and notify the users with the coresponding notifications for each type of user. Such notifications, for the most of the time, will be triggered by single event - the system will notify passengers and taxi drivers. In the notifications, accessing of the data will occur, like taxi identifier, etc.

First, high-level component view is presented in *High level components and their interaction*, which will be discussed in more details in *Component view* part. The deployment view of the physical tiers of the system will be discussed in *Deployment view* part. Through several sequence diagrams, in *Runtime view* is explained the dynamic behavoir of the software. *Component interfaces* part presents interfaces between the system components. In *Selected architectural styles and patterns* are presented architectural design styles that will be used in development of MyTaxiService, as well as advantages and reasons for choosing these architectural design styles. Design decisions not mentioned in previous subsections will be stated in *Other design decisions* part.

## 2.2 High Level Components and Their Interaction

The main high level components of MyTaxiService software product are structured like this:

- **Client**: this layer consists of mobile client and web browser. The mobile client communicates directly with the application server, while the web browser have access to the web server.
- **Web server**: this layer is in charge of providing web interface, and it does not contain any application logic.
- **Application server**: this layer is in charge of the application logic of the system - all algorithms, computations and policies are performed here. It offers a service-oriented interface.

- **Database**: this layer is in charge of data storage and retrieval and it does not contain any application logic.

The interractions between these components are shown in this figure.

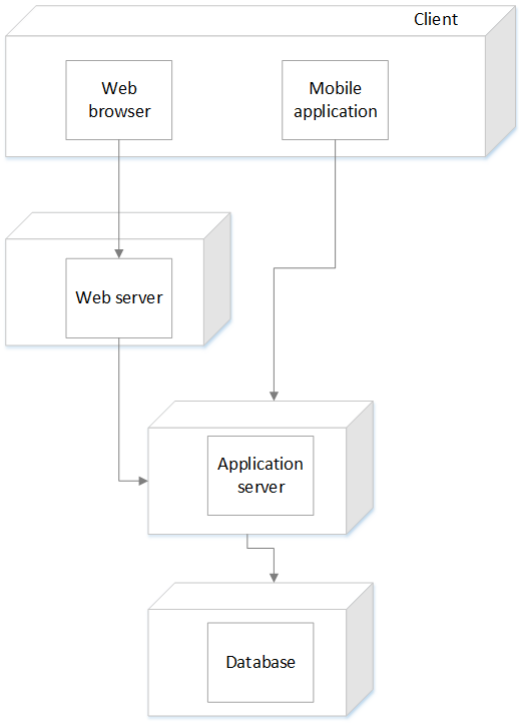



Figure 1: High-level view

## 2.3 Component View

More detailed description of the tiers is presented in the following figure:

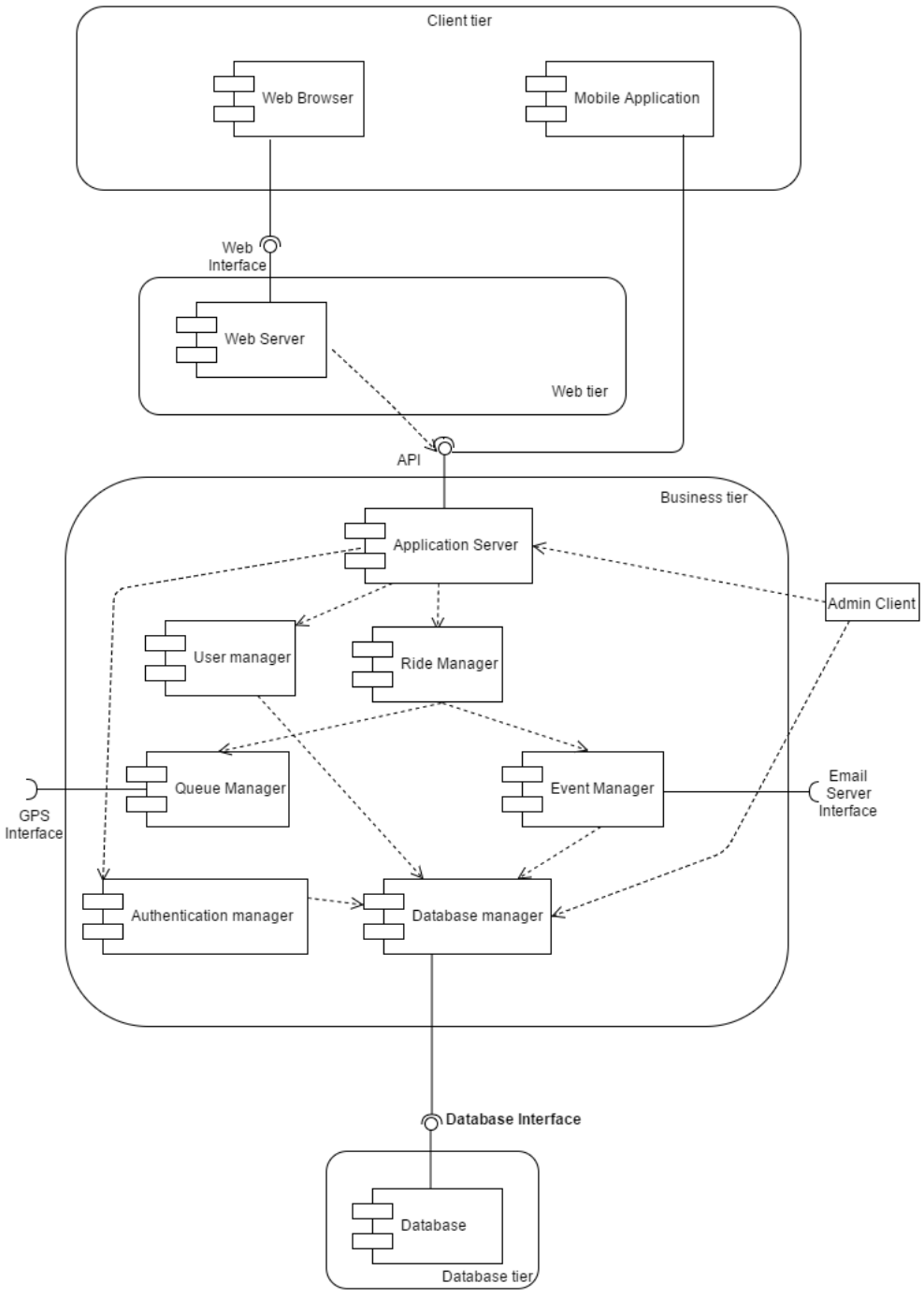



Figure 2: Component view

**Client:**

Web browsers tend to talk to the 'web components' on the web tier by inserting and submitting data in the input forms. All major web browsers are considered.

On the other hand, mobile client implementation depends on the specific platform. Inputs from the user interface are translated into functions to the application tier's APIs.

**Web server:**

The web tier consists of components that handle the interaction between clients and the application tier. As mentioned previously, this tier only implements the presentation layer, while the logic is processed by the application server tier.

**Application server:**

This server is implemented in the business tier. The business tier consists of components that provide the business logic for an application. The core functionalities of our application will come from this tier.

This tier analyses the data coming from the web tier and, according to the request, it modifies or asks for the required information stored in the database, then it is able to send the result to the web tier.

**User manager:**

Manages users, interacts with Database manager.

**Authentication manager:**

Ensures that user's credentials are correct, handles situations with invalid data. Interacts with Database manager.

**Ride manager:**

Handles requests and reservations, or everything connected with rides. It is a central part of the application, interacts with Queue and Event manager.

**Queue manager:**

Manages the taxi zones and queues logic, and it is connected with the GPS interface. It memorize, but does not store in the DB, the position of every active taxis received by the GPS Interface.

**Event manager:**

Receives messages from Ride manager, interacts with Database manager and Email server.

**Database manager:**

Handles the access point of the middle tier to the database tier.

**Database:**

All application data is stored here. It can be accessed by the business tier, as well as by the administrators, who can directly add driver account.

DBMS has to ensure ACID properties. Access to the data must be granted only to authorized users possessing the right credentials.

## 2.4 Deployment View

This section presents a view of how the software components are going to be placed on the hardware.

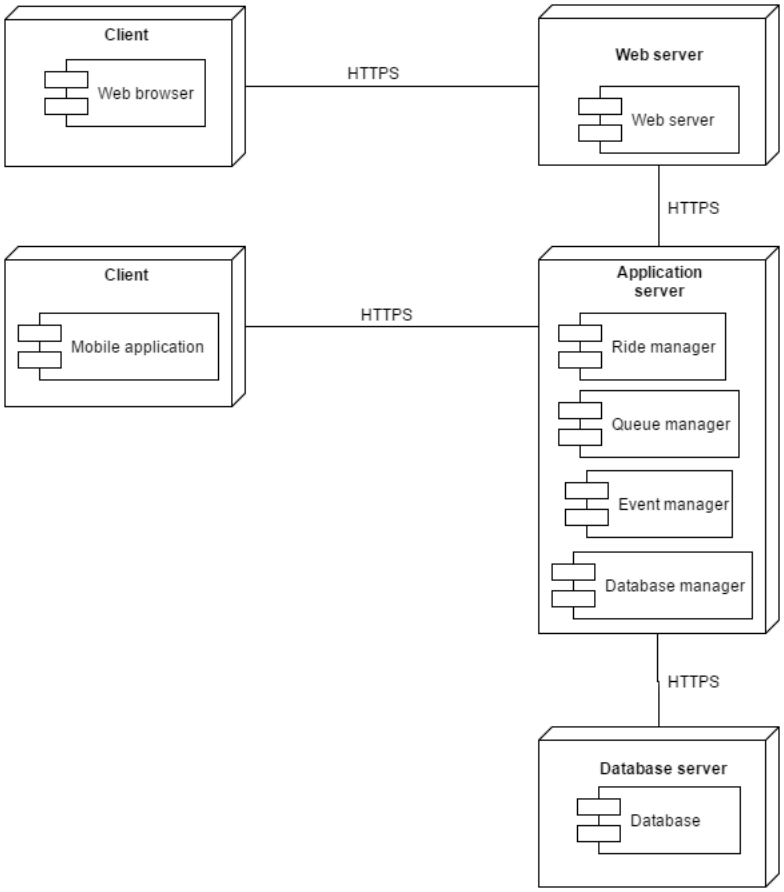
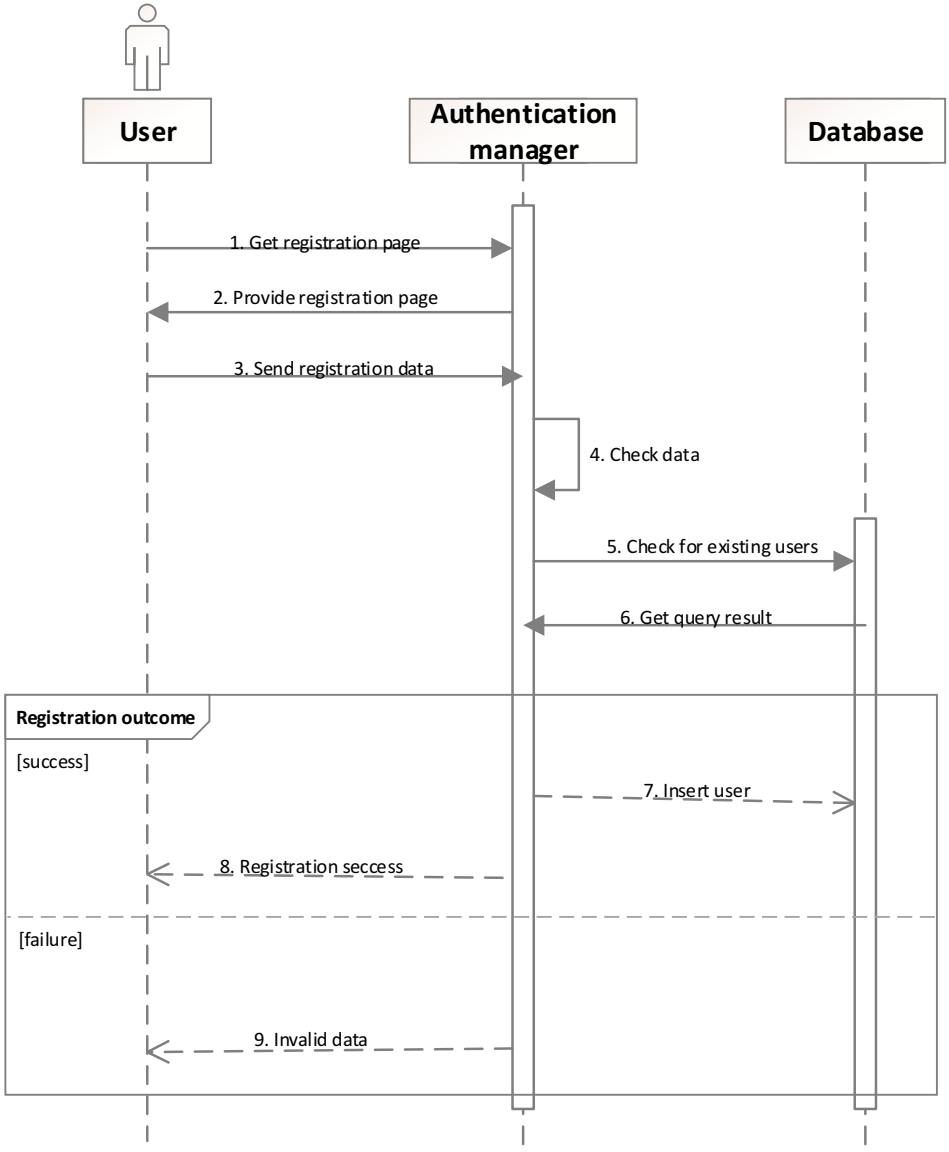



Figure 3: Deployment view

## 2.5 Runtime View

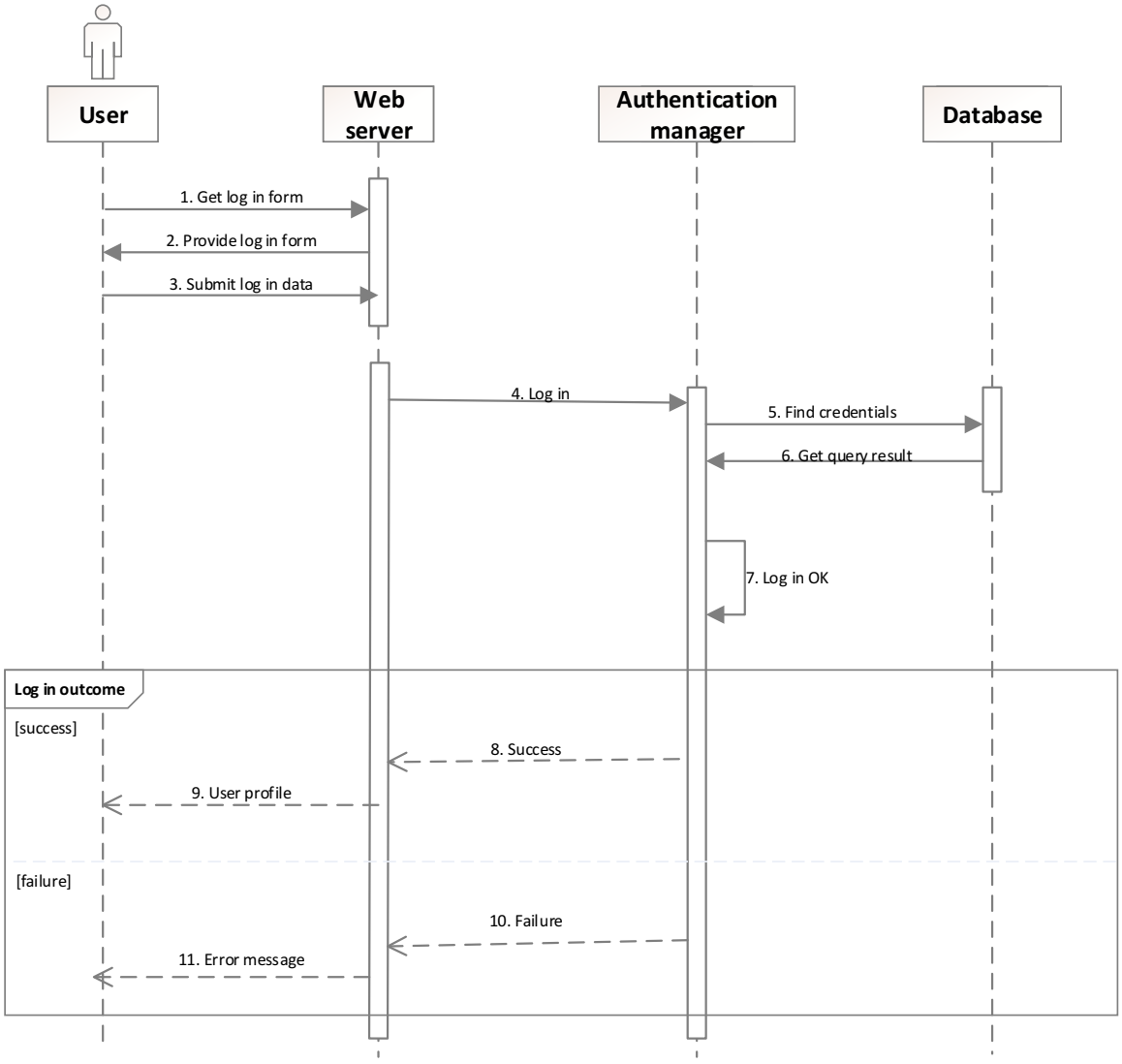
In this section is presented the dynamic behaviour of the system. Through sequence diagrams is shown how the software modules previously described in the component view of the system interact with each other. Sequence diagrams are made only for the more meaningfull functionalities of the system.

**Registration (mobile application)**

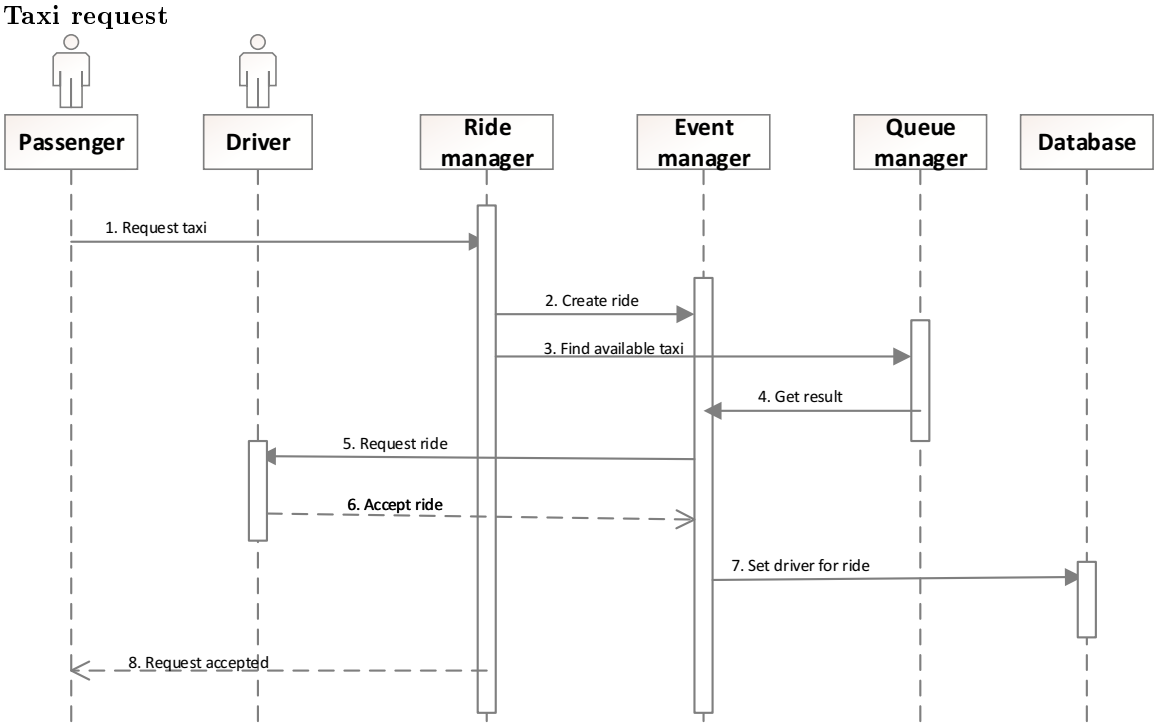



Sequence diagram 1: Registration

**Log in (web browser)**




Sequence diagram 2: Log in




Sequence diagram 3: Request

## 2.6 Component Interfaces

**User manager**

- createNewUser: this interface allows the user to access an interface that will allow him to register.
- updateUser: this interface allows the user to access an interface that will allow him to make updates on his profile.

**Authentication manager**

- verifyUser: this interface allows connection to the database in order to retrieve the correct query for verifying the user.
- logIn: this interface allows an access to the functionality of registretion and authentication of the user.

**Ride manager**

- rideRequest: this interface forwards user request to other components to handle the request for taxi. This is a bridge between user and ride component.

- rideReservation: this interface has the same role as the previously explained one.

**Event manager**

- manageNotifications: this interface deals with all types of notifications that need to be forwarded from the system to the user interface.

**Queue manager**

- receiveLocationInfo: this interface exposes taxi driver position.
- maintainQueues: this interface deals with maintaining the correct positions of the taxis in a zone.

**Database manager**

- manageRequests: this interface forwards queries for requests and reservations.
- manageUsersData: this interface allows for the application server and the users to be notified of the correctness of user's data.

## 2.7 Selected Architectural Styles and Patterns

### 2.7.1 Client-Server Style

Client-server style is used in MyTaxiService software product when:

user's browser, which is a client in this case, communicates with the web server;

web server, also a client in this case, communicates with the application server;

appication server, as a client, communicates with the database, which is as a server.

The choice for using this style comes because client-server style is the most common used style for managing this type of functionalities. Because it is well known and already implemented style, it can be easily reused for managing these functionalities.

### 2.7.2 Model-View-Controller (MVC)

Model-View-Controller is a design pattern chosen for MyTaxiService software product, and both clients - mobile application and web application are build using MVC.

The choice for this design patter comes because it is the most common and most efficient pattern for this type of applications.

### 2.7.3 Multi-Tier Pattern

Multi-tier pattern is used in MyTaxiService software product in order to divide parts of the software that perform different task.

The choice for choosing this pattern comes from the various benefits:

Maintainability: Because each tier is independent of the other tiers, updates or changes can be carried out without affecting the application as a whole.

Scalability: Scaling out the application comes easily by using this pattern.

Flexibility: Flexibility comes from possibility to manage each tier independently.

## 2.8 Other Design Decisions

Our intention is to use Java Enterprise Edition as a programming language, GlassFish Server as an application server, and MySQL Server for providing database management.

This choice comes because JEE allows and supports the development of multi-tier architecture, which is architecture used in MyTaxiService.

# 3 Algorithm Design

**Taxi Managment 1** Distribute unqueued taxis

**Ensure:** Taxi GPS information available
  tInfo = getAvailableTaxis()  ▷ receive taxis not in Queue and not checked
  **for** each $t$ in tInfo **do**
    **if** $t$.status = Available **then**
      Q = findClosestQueue($t$.location)
      AddToQueue($t$, Q)
    **else**
      CheckTimer($t$)
      RemindDriver($t$)  ▷ Remind driver to update its status
    **end if**
  **end for**

**Rides Handler 2** Manage rides as being requested

**Ensure:** Taxi GPS information available

```
loop                          ▷ This can be threaded, assuming atomicity for each event
    s = WaitForJob()
    if s.type = Ride then
        taxi = ∅
        repeat
            Q = findClosestQueue(s.location)
            taxi = Q.dequeue()
            c = WaitForConfirmation(taxi)
        until c ≠ confirmed
        SetTimer(taxi)
        call Taxi Manager                          ▷ Reorder queues
    else
        MakeReservationEvent(s)          ▷ Taxi will be assigned 2h before
    end if
end loop
```

# 4 User Interface Design

Please refer to the section *2.1.2 User Interfaces* in the RASD document.

# 5 Requirements Tracebility

In this section is presented the association between requirements defined in the RASD document and the design components described in this document. The assocciation is presented in the followin table, where in the left column are listed the requirements, while on the ride column are listed the main components that allow particular system functionallity.

| Requirements | Design components |
|---|---|
| **FR1**: Create passenger | User manager<br>Authentication manager<br>Database manager |
| **FR2**: Create taxi driver | User manager<br>Authentication manager<br>Database manager |
| **FR3:** Consult user profile | User manager<br>Database manager |
| **FR4**: Update user profile | User manager<br>Database manager |
| **FR5**: Register to the system | User manager<br>Authentication manager<br>Database manager |
| **FR6**: Log in | User manager<br>Authentication manager<br>Database manager |
| **FR7:** Log out | User manager<br>Authentication manager<br>Database manager |
| **FR8:** Create request | Ride manager<br>Event manager |
| **FR9:** Consult request | Ride manager<br>Event manager |
| **FR10**: Create reservation | Ride manager<br>Event manager<br>Database manager |
| **FR11**: Consult reservation | Ride manager<br>Event manager<br>Database manager |
| **FR12:** Taxi driver can accept request | Ride manager<br>Event manager<br>Database manager |
| **FR13:** Taxi driver can decline request | Ride manager<br>Event manager<br>Database manager |
| **FR14:** Taxi driver can accept reservation | Ride manager<br>Event manager<br>Database manager |
| **FR15:** Taxi driver can decline reservation | Ride manager<br>Event manager<br>Database manager |
| **FR16**: System sends code of the taxi and waiting time | Ride manager<br>Event manager<br>Database manager |

Table 3: Requirements tracebility

The component design has partially satisfied some of the non-functional requirements as well. One example is *NFR1:The software product can be used only by registered users* and *NFR2:User password must be saved securely,* where these functionallities can be achieved with Authentication manager.

# 6 Appendix

## 6.1 Hours of Work

Mite Ristovski $\sim$ 25 hours
Dushica Stojkoska $\sim$ 28 hours

## 6.2 Software and Tools Used

- LyX (http://www.lyx.org/) to format this document
- https://www.draw.io/ to create UML diagrams - component and deployment diagram
- Microsoft Visio to create the sequence diagrams