

Politecnico di Milano

A.A. 2015-2016

Software Engineering 2: “myTaxiService”

Design Document

Version 1.0

Milica Jovanovic (mat. 835953), Pavle Vidanovic (mat. 854472)

02 December 2015

Contents

[1 Introduction 5](#_Toc436746359)

[1.1 Purpose 5](#_Toc436746360)

[1.2 Scope 5](#_Toc436746361)

[1.3 Glossary 6](#_Toc436746362)

[1.4 Reference Documents 6](#_Toc436746363)

[1.5 Document Overview 6](#_Toc436746364)

[2 Architectural design 7](#_Toc436746365)

[2.1 Overview 7](#_Toc436746366)

[1.1 High level components and their interaction 7](#_Toc436746367)

[2.2 Component view 8](#_Toc436746368)

[2.3 Deployment view 9](#_Toc436746369)

[2.4 Runtime view 10](#_Toc436746370)

[2.4.1 SignUp 10](#_Toc436746371)

[2.4.2 Request 11](#_Toc436746372)

[2.5 Component interfaces 11](#_Toc436746373)

[2.5.1 Guest Manager 11](#_Toc436746374)

[2.5.2 User Manager 12](#_Toc436746375)

[2.5.3 TaxiDriver Manager 12](#_Toc436746376)

[2.5.4 Admin Manager 13](#_Toc436746377)

[2.6 Selected architectural styles and patterns 14](#_Toc436746378)

[2.6.1 Architectural styles 14](#_Toc436746379)

[2.6.2 Patterns 17](#_Toc436746380)

[2.7 Other design decisions 19](#_Toc436746381)

[3 Algorithm design 20](#_Toc436746382)

[3.1.1 Request Algorithm 20](#_Toc436746383)

[3.1.2 Reserve Algorithm 21](#_Toc436746384)

[3.1.3 ConfirmDecline Algorithm 22](#_Toc436746385)

[3.1.4 SetAvailability Algorithm 23](#_Toc436746386)

[4 User interface design 23](#_Toc436746387)

[5 Requirement traceability 24](#_Toc436746388)

[6 References 24](#_Toc436746389)

# Introduction

## Purpose

The purpose of this document is to provide a comprehensive description of the structure of the myTaxiService system. It will state and analyze the design decisions made in order to satisfy all the requirements stated in the Requirements Analysis and Specification Document (RASD). This document is meant mainly as a guideline for developers of the software in question.

## Scope

The aim of this project is to develop and implement myTaxiService, an application similar to Uber, which makes the process of assigning an available taxi vehicle to possible passengers.

The developed system should allow new users to register. Users, once logged in, should be able to:

* request a taxi
* reserve a taxi
* cancel a ride
* check taxi availability around him
* receive a confirmation with information about the assigned vehicle and ETA once taxi is requested
* create/maintain user profile
* report a taxi driver

The developed system should allow new taxi drivers to register. Drivers, once logged in, should be able to:

* inform the system about their availability
* confirm/decline that they are going to take care of a certain call
* create/maintain taxi driver profile
* report a passenger

The system should keep information about new arrived requests, as well as the confirmed rides. A ride should have and id number, information about the passenger that requested the ride, as well as the code of the assigned vehicle and ETA. System should also keep information about taxi queues connected to particular zone of the city and ensure fair management of the queues. Developed system should keep information about the list of reservations made by passengers, such as id number of the reservation, information about the passenger that made the reservation and the time of reservation and time of the ride.

## Glossary

The following are the definitions of some commonly used phrases throughout the document:

|  |  |
| --- | --- |
| *CSS* | Cascading Style Sheets |
| *DD* | Design Document |
| *DB* | Database |
| *DBMS* | Database Management System |
| *ER* | Entity-Relationship |
| *HTML* | Hypertext Markup Language |
| *HTTP* | Hypertext Transfer Protocol |
| *HTTPS* | Hypertext Transfer Protocol Secured |
| *MVC* | Model View Controller |
| *RASD* | Requirement Analysis and Specification Document |
| *UML* | Unified Modeling Language |
| *ORM* | Object Relational Mapper |

## Reference Documents

* IEEE Design Document template
* Specification Document: myTaxiService Project AA 2015-2016.pdf
* RASD myTaxiSevice

## Document Overview

The document is essentially structured in six parts:

* Chapter 1: Introduction, gives description of document and some basic information about the software
* Chapter 2: Architectural Design, gives an overview of how  and why the system was decomposed, and  how the  individual parts work together
* Chapter 3: Algorithm Design, description of the most relevant algorithms of the software system
* Chapter 4: User interface Design, overview on how the user interfaces of your system will look
* Chapter 5: Requirements traceability, gives an overview of how the requirements defined in RASD map into the design elements defined in DD.
* Chapter 6: References

# Architectural design

## Overview

Identify modules in the software system and explain the relationships between the modules to achieve the complete functionality of the system. This is a high level overview of how responsibilities of the system were partitioned and then assigned to subsystems. Identify each high level subsystem and the roles or responsibilities assigned to it. Describe how these subsystems collaborate with each other in order to achieve the desired functionality.

## High level components and their interaction

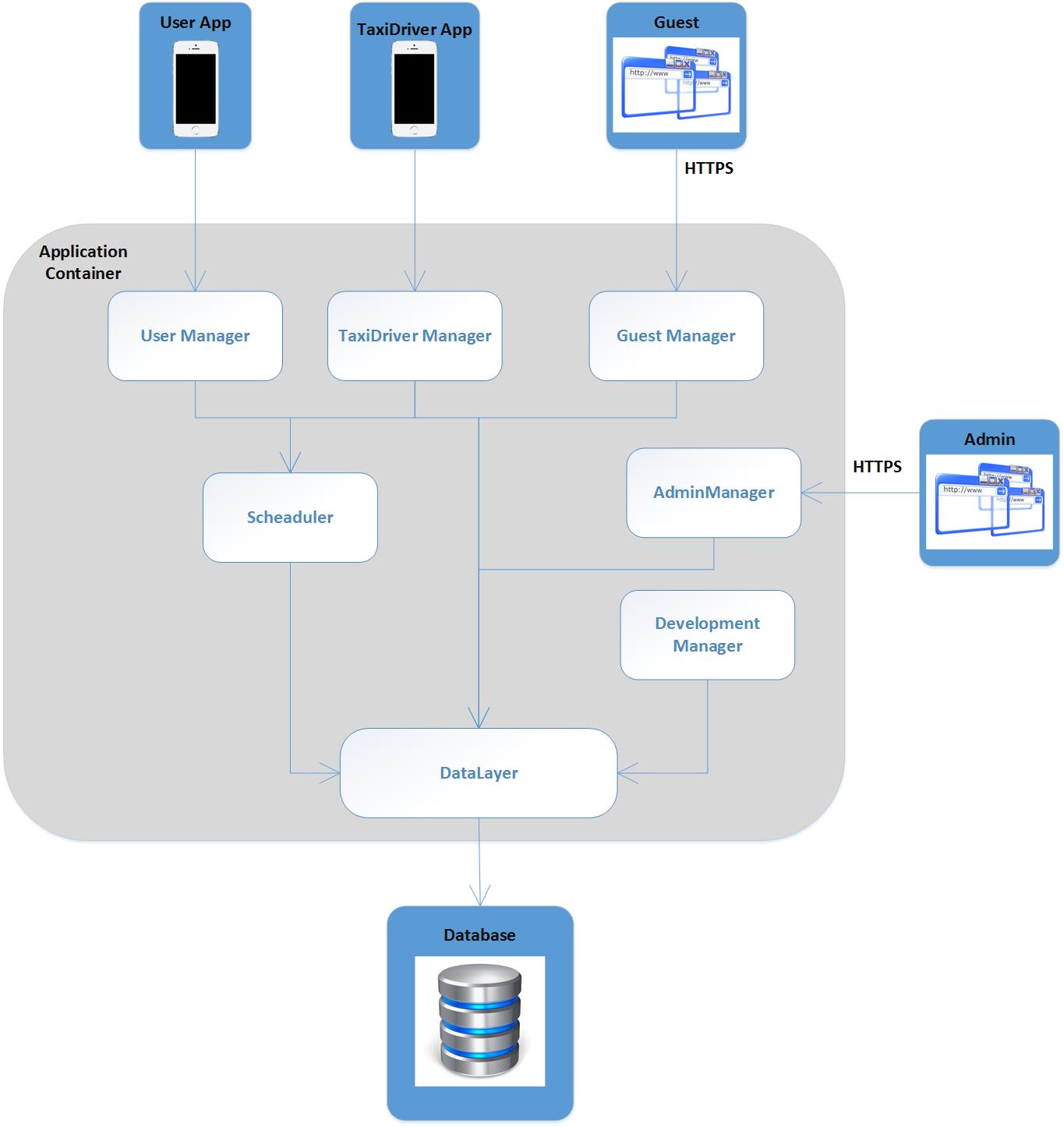


Figure 2.1

* *Guest Manager* component will provide interface for unregistered users or guest to create a profile.
* *User Manager* is component of the system that will provide interfaces for users (passengers) with an options of signing in, requesting, reserving taxi, managing profile, reporting driver, checking notifications and canceling a ride.
* *Taxi Driver* Manager is component of the system that will provide interfaces for taxi drivers with an options of signing in, confirming a ride, setting availability, reporting a user, checking notification and canceling a ride.
* *Admin Manager* is component of the system that will provide interfaces for taxi drivers with an options of signing in, viewing reports and banning a user.

## Component view

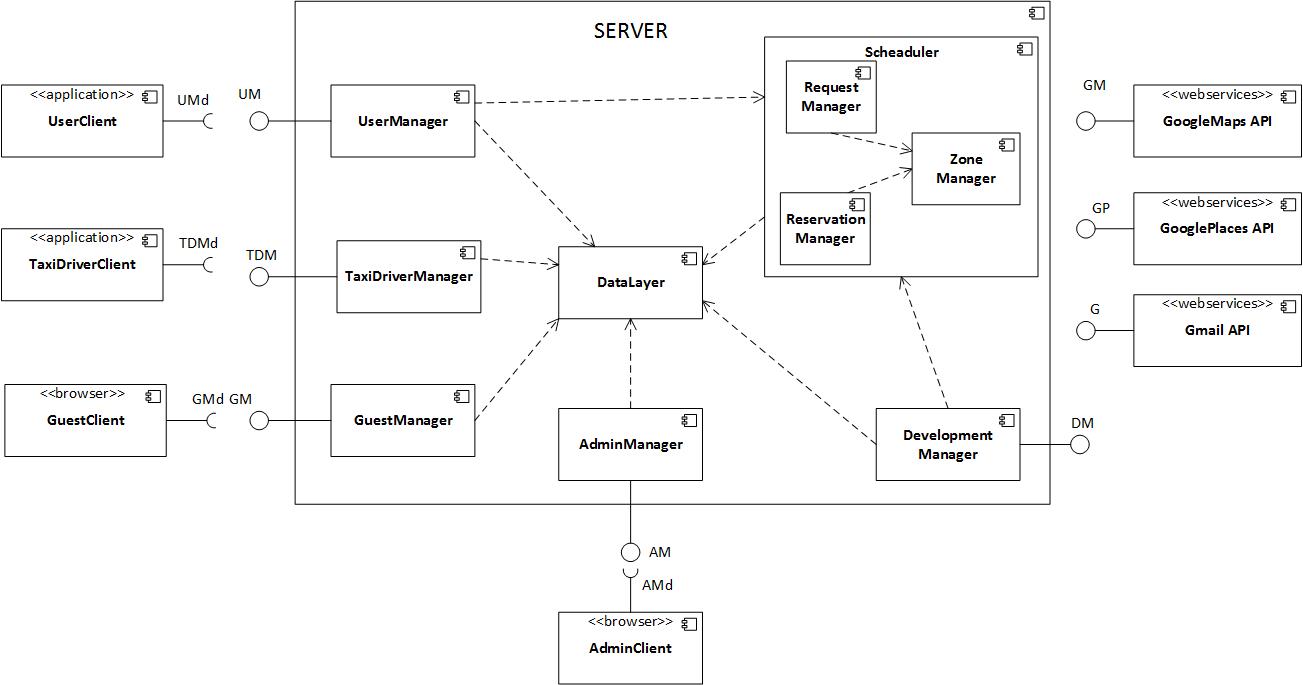


Figure 2.2

## Deployment view

## Deployment2

Figure 2.3

## Runtime view

### SignUp

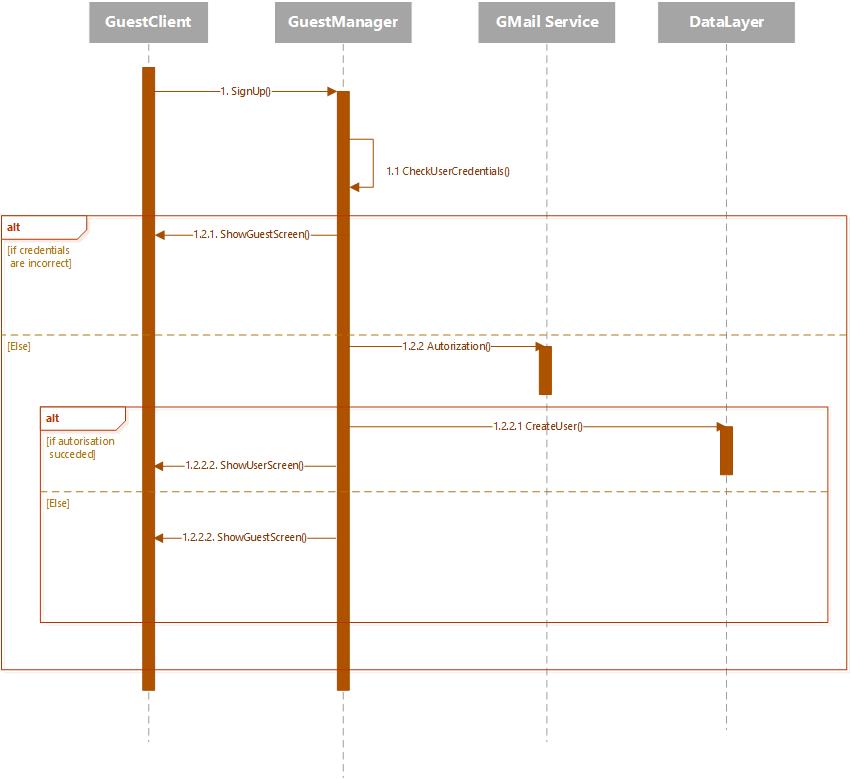


Figure 2.4

### Request

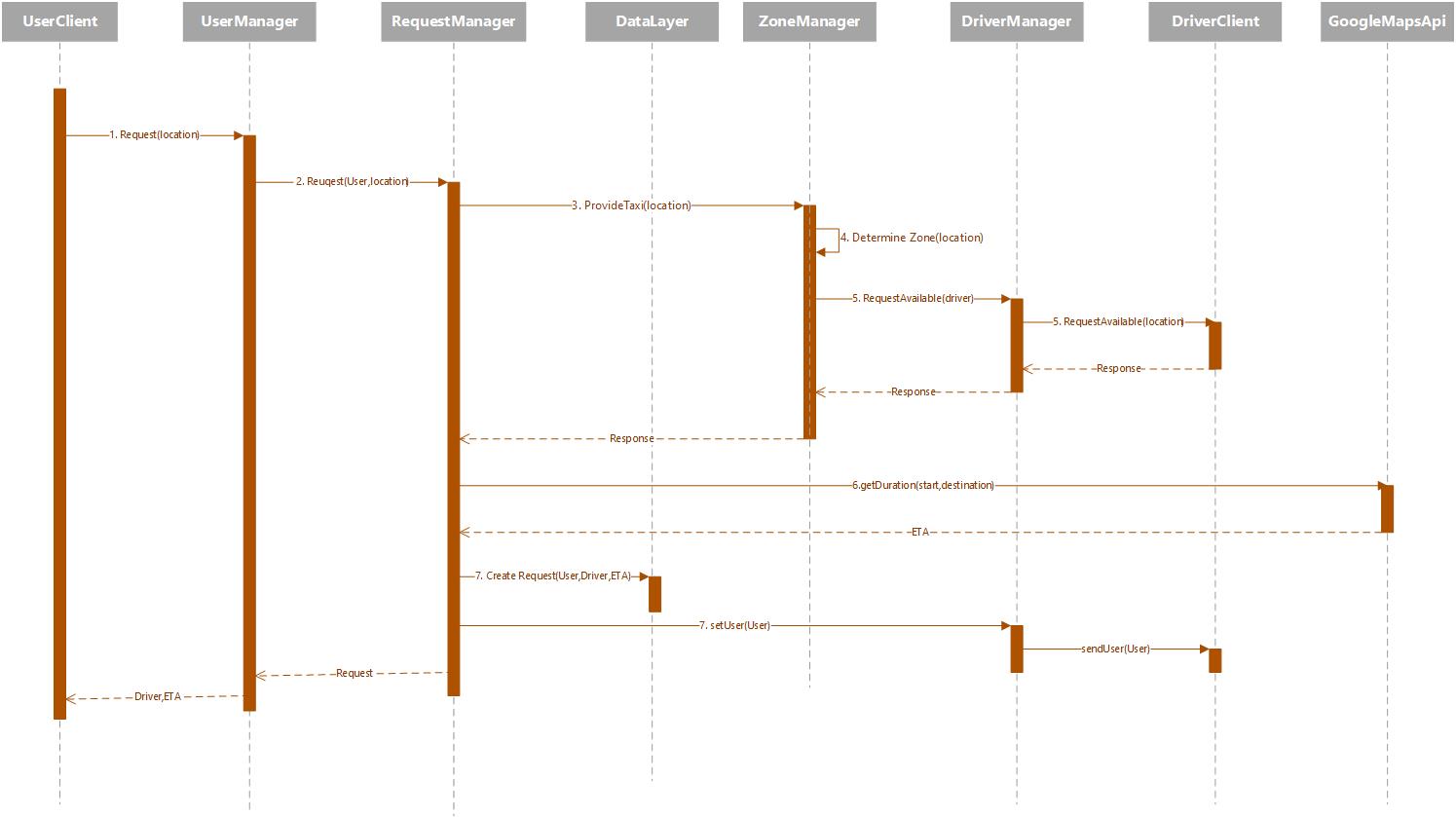


Figure 2.5

## Component interfaces

### Guest Manager

#### User signUp(name,surname,email,password,phone,img)

signUp function after submitting the inputted data sends it to the web server which checks the credentials if they already exist, uses Gmail API for email authentication. If received data is valid a new tuple is inserted in DB and sent back to the client app in form of JSON. After client app receives the string in encodes it to User object.

#### User signIn(email,password)

signIn function checks sent data in the server side. If the inputted credentials are valid i.e. tuple exists in DB an instance of User object is returned back to client app in the same way as in signUp function.

### User Manager

#### Request makeRequest(source\_address,user)

makeRequest function creates Request object, with user and source address initialized. Then it uses findZone(source\_address) function of scheduler that returns zone of the user. Next, the Zone object contains queue of available taxi drivers, and the request is sent to first driver in the queue, and if his response is negative request is passed to next driver in queue. Then request instance is created and it is sent both to user and driver as confirmation.

#### Reservation makeReservation(source\_address,time\_of\_ride,destination\_address,user)

makeReservation function creates Reservation object, with user, source and destination address and time of the ride initialized. Then it uses findZone(source\_address) function of Scheduler that returns zone of the user. Next, the confirmation of reservation is sent back to user. Moreover, the Zone object contains queue of available taxi drivers, and 30 minutes before the ride, the reservation is sent to first driver in the queue, and if his response is negative reservation is passed to next driver in queue. When the driver is found the reservation instance is created and stored in DB and then 10 minutes before the ride the confirmation message is sent to user with drivers details.

#### String report(description,id\_user,id\_banned)

report function creates Report object with inputted data and stores it to DB.

#### String manageProfile(user)

manageProfile function receives modified User object and updates appropriate user tuple in DB.

#### TaxiDriver[] checkTaxisAvailable(source\_address)

checkTaxisAvailable function contacts ZoneManager which returns a Zone object with the respect to inputted source address. Zone object contains queue of available taxi that is send back to client.

#### String cancelRide(request)

cancelRide function deletes particular request tuple of the database. After that notification message is sent both to user and driver related to this request.

### TaxiDriver Manager

#### String confirmDeclineRide(request)

confirmDeclineRide is function of TaxiDriverClient interface. Driver will accept or decline a ride and his response is positive, the request attribute driver is instantiated and Google Maps API is invoked and provides a route. Otherwise, appropriate driver is moved to the end of a queue of appropriate zone (setAvailability(true, id\_driver)).

#### String setAvailable(available\_bool,id\_driver)

setAvailable is function of Driver Manager interface. First, we contact the DB to get appropriate driver. Then, ZoneManager is contacted in order to provide drivers zone based on his current location. Based on available\_bool value the driver is added or removed from the zone queue.

### Admin Manager

#### String banUser(id\_report)

banUser is function of AdminManager component that uses id\_report parameter to find particular user\_id. After that, it deletes user tuple from DB. The confirmation message is sent back to admin.

#### Report[] viewReports()

viewReports is function of AdminManager component which returns all reports from DB to admin.

## Selected architectural styles and patterns

### Architectural styles

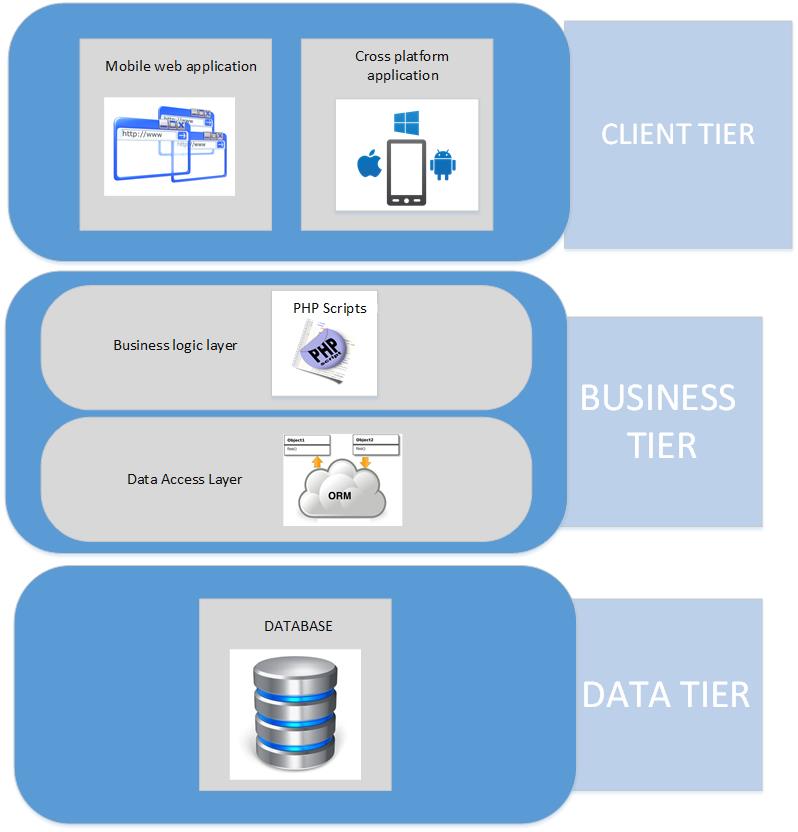


Figure 2.6

myTaxiService architecture is a 3-tier architecture, which consists of Client tier, Business tier and Data tier.

* **Client Tier**: it contains Cross platform application and Mobile Web application and it is the layer that interacts directly with the actors. As our project will be a web application the client will use a web browser to access pages
* **Business Tier:** Is consisted of two layers, Business logic layer and Data access layer. Business logic layer contains php scripts that are representing web services with application logic. Data access layer accepts the data from client and packs it in a form acceptable by database server, as well as extracting data from database and forms an object from them.
* **Data Tier**: Establishes a connection with MySql database and enables storing and retrieving all relevant data.

Client-Server

The client–server model of computing is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Clients and servers exchange messages in a request–response messaging pattern: The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication.

Three-tier architecture is a client–server software architecture style in which the user interface (presentation), functional process logic ("business rules"), computer data storage and data access are developed and maintained as independent modules, most often on separate platforms.

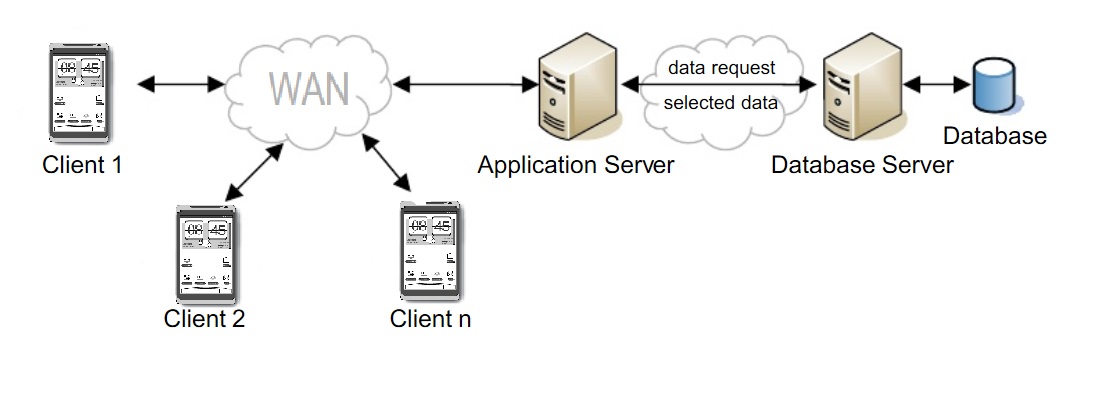


Figure 2.7

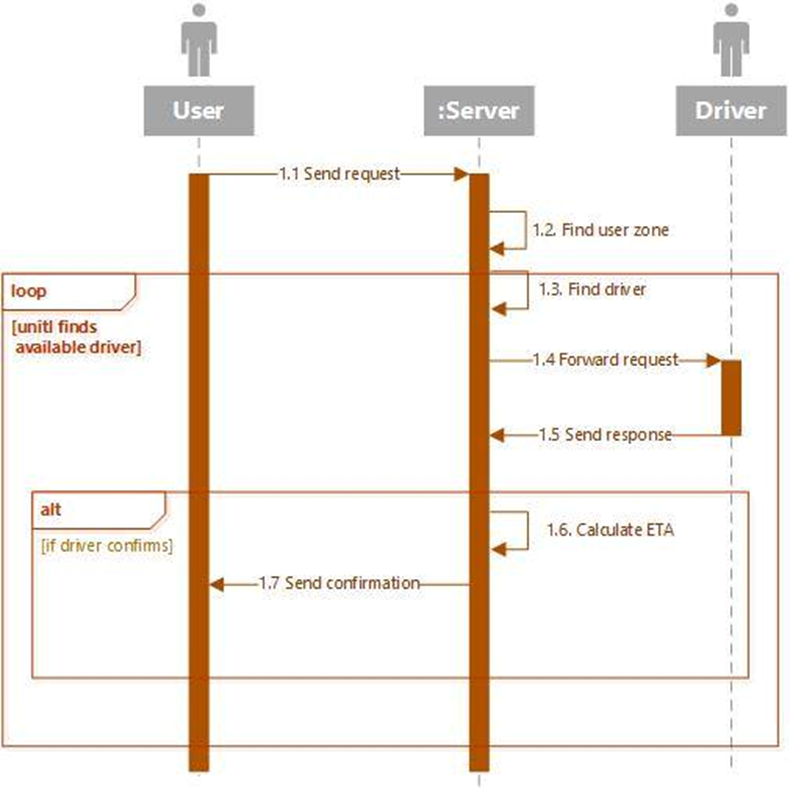


Figure 2.8

myTaxiService system will be built on three tier client-server architectural style. Communication between the users and the taxi drivers goes through the server as a mediator. Purpose of the server in myTaxiSystem is to handle both users and drivers requests. Server processes the requests and queries the database. In this scenario, the database represents the third tier and is responsible for managing the stored data. After processing the request server sends a response back to the users or drivers. For example:

* user requests a taxi
* request is sent to the server
* server finds the zone and takes first available taxi from the queue
* server forwards the request to the taxi driver
* driver responds positive/negative
* response is sent to the server
* server calculate ETA and send response to user or he finds next taxi in the queue and send him the request

#### SOA

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. The purpose of SOA is to allow users to combine fairly large chunks of functionality to form applications built from existing software services. If a service presents a simple interface that abstracts away its underlying complexity, then users can access independent services without knowledge of the service's platform implementation.

* + When user fills the request form, system will contact the Google Places service for suggesting and autocompleting the input address. The suggested address is sent back to the user.
  + When navigation page is requested by the taxi driver, Google Map service is contacted. Service provides the driver with map page with appropriate route.
  + Google Maps Geocoding service provides coordinates for inputted address.
  + Gmail API as email service of user’s authorization.

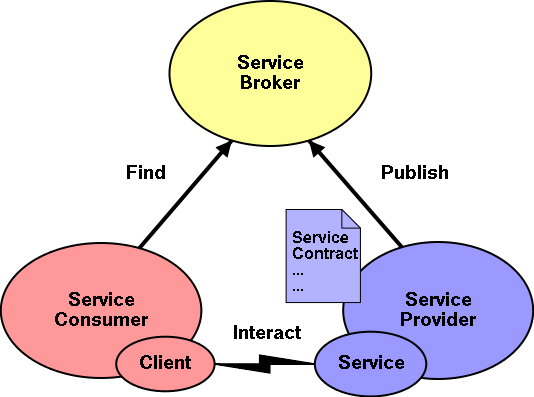


Figure 2.9

### Patterns

#### Singleton

Singleton pattern is a design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system.

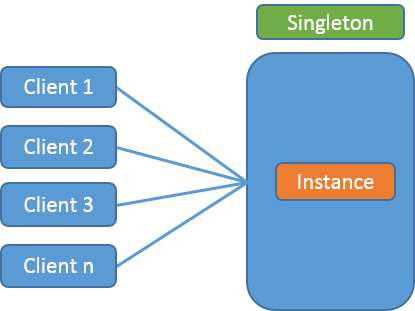


Figure 2.10

myTaxiService system will have one instance of Database. Every time access to the database is needed, system through this instance will open connection to the database. After the transaction the connection will be closed.

#### MVC

The model-view-controller pattern proposes three main components or objects to be used in software development:

* A Model, which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.
* A View, which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)
* A Controller, which contains the logic of system and is a mediator between the view i.e. user interface and his interaction with it and model which contains the data stored for our system.



Figure 2.11

In our system the use of MVC design pattern in three tier architecture is presented in next picture:

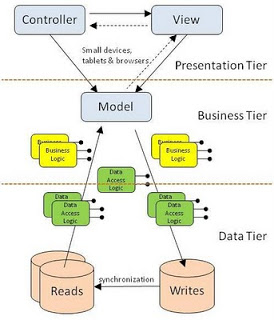
[](http://1.bp.blogspot.com/-YbwFniURtpE/Tna9TcC35TI/AAAAAAAAAKY/WQZcUQ1rL0w/s1600/mvcNtier.jpg)

Figure 2.12

When the MVC and three-tier approaches are brought together the View and Controller are considered the presentation tier, the Model exists in the business tier (and has access to many business and data tier modules). To a certain extent the Model could span both the business and data tiers. It is this authors view that the Model would not exist in the layer of the data tier that is specifically Data Access Layer (DAL) code including SQL or code optimized for DAL within a cached environment.  
Some important aspects of this diagram include is the separation of data reads from writes. Writes take considerably more data storage resources than do reads, so separating these into different physical data stores can have huge benefit. It should also identified that the modules for the business and data logic can be numerous. And these exist for maintainability and scalability purposes. It is also reasonable that the Model may access the Data Tier directly without going through a Business Tier module.

## Other design decisions

# Algorithm design

### Request Algorithm

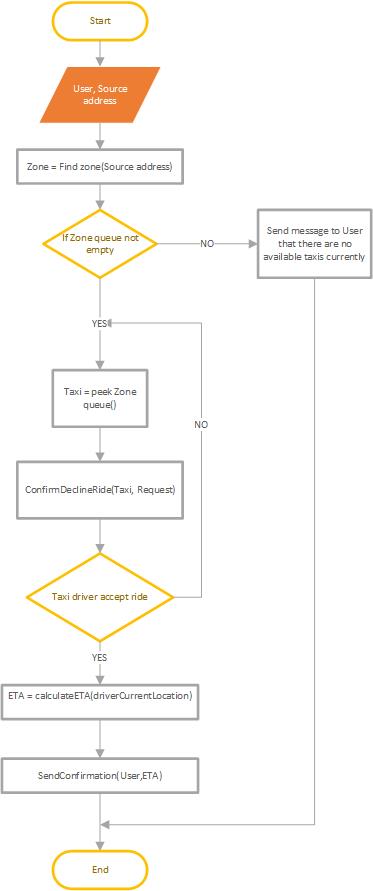


Figure 3.1

### Reserve Algorithm

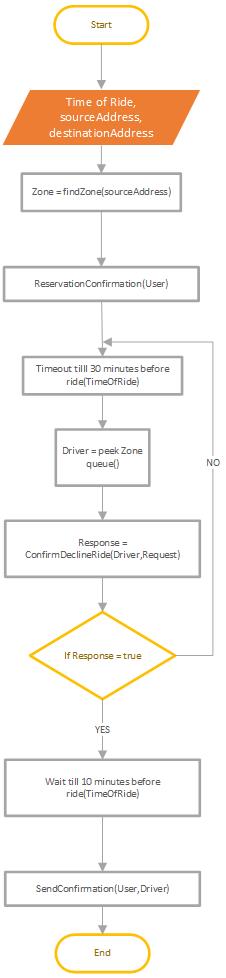


Figure 3.2

### ConfirmDecline Algorithm

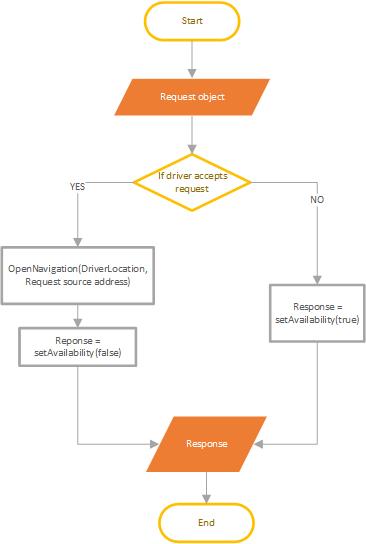


Figure 3.3

### SetAvailability Algorithm

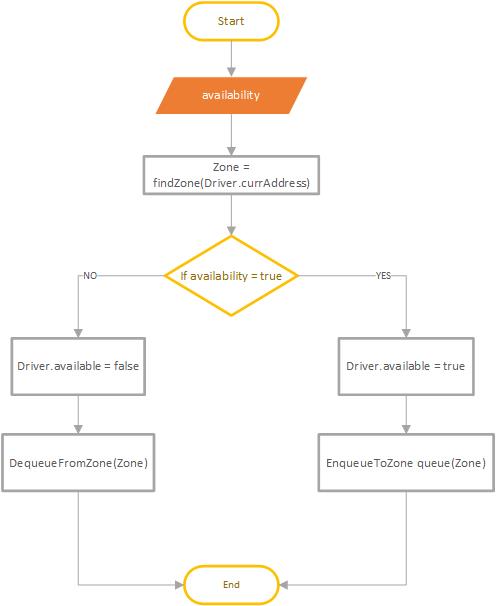


Figure 3.4

# User interface design

Refer to the chapter 3.1 of RASD documents. In chapter 3.1 are shown user interfaces of myTaxiService system.

# Requirement traceability

# References

* IEEE Design Document template
* MVC in a three-tier architecture, Peter Rawsthorne
* https://msdn.microsoft.com/en-us/library/ee658117.aspx
* Singleton Design Pattern- Java Tech Guy must read!, Rudra Narayan Garnaik
* Software Design and Software Architecture, Politecnico di Milano
* Advanced Software Design and architectures: Architecture styles, Damian A. Tamburri