SE2 Design Document

Edoardo Giacomello Mattia Fontana

November 30, 2015

Contents

1 Introduction **2**
- 1.1 Purpose . . . 2
- 1.2 Scope . . . 2
- 1.3 Definitions, Acronyms, Abbreviations . . . 2
- 1.4 Reference Documents . . . 2
- 1.5 Document Structure . . . 2

2 Architectural Design **2**
- 2.1 Overview . . . 2
- 2.2 High level components and their interaction . . . 4
- 2.3 Component View . . . 7
  - 2.3.1 Mobile Application . . . 7
  - 2.3.2 Web Application . . . 8
  - 2.3.3 Web Server . . . 8
  - 2.3.4 API . . . 8
  - 2.3.5 Authentication Manager . . . 9
  - 2.3.6 Request Manager . . . 9
  - 2.3.7 Zone Manger . . . 10
  - 2.3.8 Taxi Manager . . . 10
  - 2.3.9 Account Manager . . . 10
  - 2.3.10 Administration Manager . . . 10
  - 2.3.11 Notification Manager . . . 10
  - 2.3.12 Data Model . . . 10
  - 2.3.13 Database Manager . . . 10
- 2.4 Deployment View . . . 10
- 2.5 Runtime View . . . 10
- 2.6 Component Interfaces . . . 10
- 2.7 Architectural Styles and Patterns . . . 10
- 2.8 Other Design Decisions . . . 10

3 Algorithm Design **10**

4 User Interfce Design **10**

5 Requirements Traceability **10**

6 References **10**

7 Tools and Document Information **10**

1 Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms, Abbreviations

GUI : Graphical User Interface. It is the set of graphical elements such text, buttons and images which the user can interact with.

Thin Client : It is a design style for the client in which the application running on the client contains the least business logic possible. In our case the client application will only serve as a presentation interface.

1.4 Reference Documents

- Structure of the Design Document

1.5 Document Structure

2 Architectural Design

2.1 Overview

The MyTaxiService application has been designed to run on a client-server architecture.
In particular, the general system has been divided in some sub-systems and each of those has been designed to run on a different phisical machine. Since the modularity of the system has been taken into account, eventual further design decisions that could increase or decrease the tier size of the architecture will be supported.
The sub-systems that have been identified are the following:

Client Application (1..N) : This is the user frontend of the system. It cames in form of a web-interface or a mobile web application and it has two possible realizations which are the Passenger Interface or the Taxi Driver interface. There's included also a realization that is currently designed only in a web interface form, which is the system administrator interface.

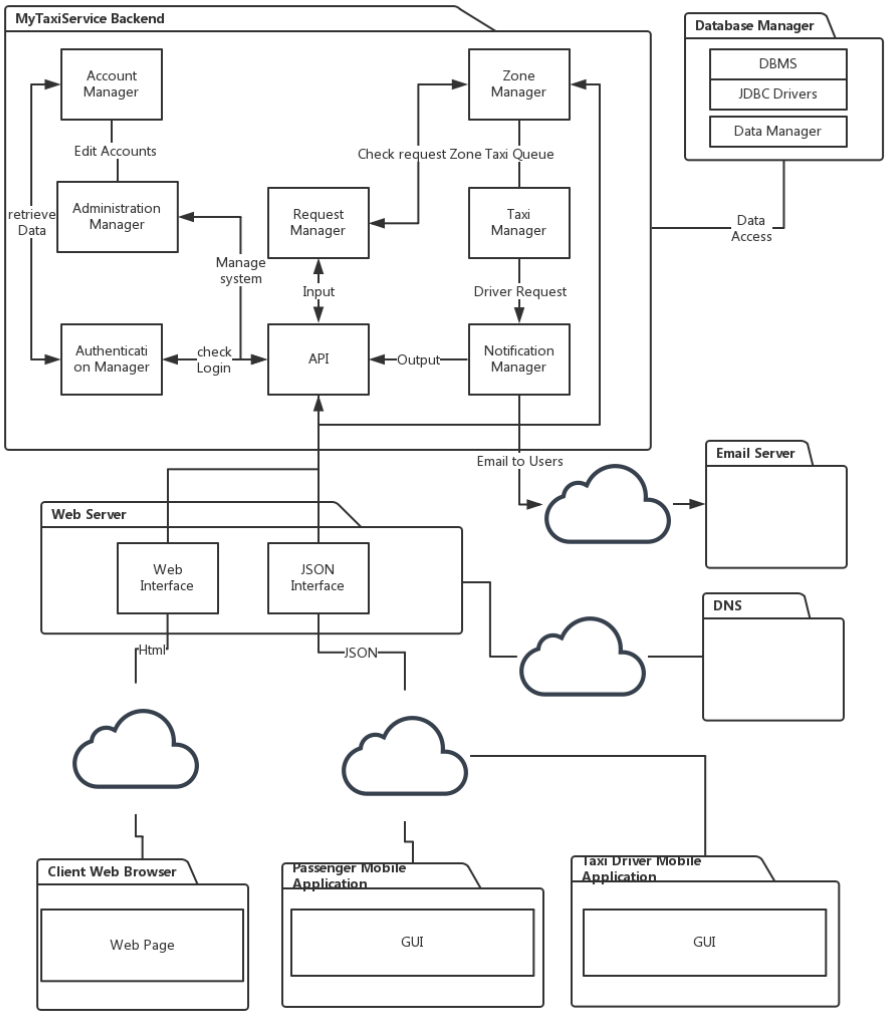
Web Server (1..N) : This is the main interface between the Backend and the Client application. The web-server could be distributed on several machines that are managed by a load-balancer, and should provide a firewall as well. The main purpose of this sub-system is that to generate the web-pages for the web-interface and forwarding/translating requests from the client and the backend.

Backend (1) : This is the sub-system in which the business logic resides.

Database (1..N) This is the sub-system in which the business data resides. It can be replicated or distributed for avoiding data corruption and undesired behaviours.

Third Party Services (N) Those sub-systems are auxiliary third-party services that the system can avail itself of for providing core services that will not be directly implemented on the system, such that email notification, DNS lookups, Map Services, etc.

2.2 High level components and their interaction



Here follows the high-level description of each component or module of the application, which is devided in a particular subsystem.

1. **Client Application**
   (a) **Website** This is the web interface the passengers and administrators can access from a web browser. It will make use of AJAX for fetching data from the server.
   (b) **GUI** These are the user interfaces for passenger or taxi drivers.
   (c) **Mobile Client** This is a client for making requests to the Server and retreiving results
2. **Web Server**
   (a) **Browsing Request Interface** This component hosts the web interface and generates web pages for displaying results to the user
   (b) **Resource Request Interface** This component collects all the requests from user web and mobile interfaces and redirect them to the server APIs.
3. **DataBase Manager**
   (a) **DBMS** This component is the actual DBMS which manages the business data.
   (b) **JDBC Drivers** Drivers that provide an interface between the DBMS and the application.
   (c) **Data Manager** This component will host all the functions useful for managing data on the server and it will offer an interface of the database for the backend. In particular, the SQL and DDL queries are stored here.
4. **Backend**
   (a) **API** This is the main interface for the backend and will provide all the system functionalities that can be accessed by other subsystems. It will parse all incoming JSON requests and activate the other components accordingly. It will also support an access for system developers, that will be regulated by an authorization token mechanism.
   (b) **Authentication Manager** This component will be in charge of managing all the authentications for the system. In particular it will manage the login functionality and the authorization token for accessing the APIs.

(c) **Account Manager** This component will manage the user accounts and the password reset service.

(d) **Request Manager** This component will host the core algorithm which manages the taxi queues and will process the requests and reservations.

(e) **Taxi Manager** This component will manage the distribution of the active taxis and the notification of incoming requests by interacting with the Notification Manager.

(f) **Notification Manager** This component will send notifications to the users, such as email for the passengers and notifications for the taxi drivers.

(g) **Zone Manager** This component will manage the zones in which the area is divided. It will also make the look-up for a location into a zone and it will offer map utility functions.

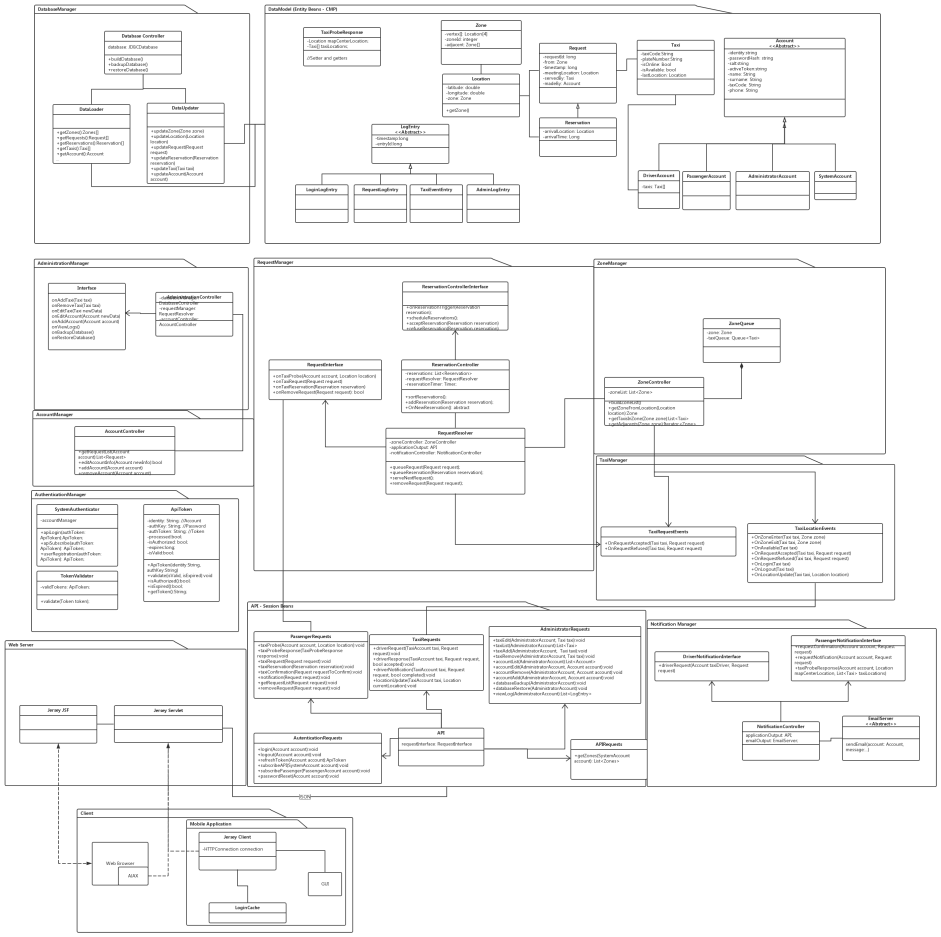
(h) **Log Manager** This component will manage all the system logs, both debugging and business inspection.

5. **Third-Party Services**

(a) **DNS** This service will offer Domain Name Lookup for accessing the various subsystems.

(b) **Email Server** This service will allow the unidirectional communication between MyTaxiService and the users.

A more in-depth view of the system is presented in this diagram:



2.3 Component View

In this section a more detailed description of each component will be presented

2.3.1 Mobile Application

This is the Mobile application for the passenger or the taxi manager. It consists on the following main components:

GUI It is the graphical user interface as described in the "User Interface Design" section.

RequestBuilder : It is the class that takes inputs from the GUI and builds requests for the server or, vice-versa, receives the messages from the server and display them on the GUI

LoginCache : This class will temporary store the user login information and the authorization token, for avoiding to request login data to the user as much as possible. For the implementation it will possible to rely on specific OS functions (as those offered by Android).

2.3.2 Web Application

This component consists in the MyTaxiService website.

2.3.3 Web Server

This component hosts the website along with some mechanisms for managing sessions and secure communications to the server APIs. Other than hosting the website, the purpose of this component is to avoid unauthorized requests to the server, and requesting the users a new login in the case their auth token is invalid or expired.
The authorization token is a key that is generated by the application server after a successful login and it has an expiration date, after which the client will have to login again in order to make new requests. The client must provide an active auth token along with every request it makes in order to be correctly identified by the system. This will apply both for users and systems that accesses the APIs, for further information please check the sequence diagrams section. The web browser will make use of Jersey, which is a reference implementation of JAX-RS for RESTful systems. The communications will occour using JSON, for which a jersey plugin is available.

Jersey JSF This component generates the web pages and provides a web interface for the users.

Jersey Servlet This component manages all the incoming and outcoming requests from and to the application APIs. It both generate contents for the AJAX calls from the web browser and the mobile application.

2.3.4 API

This component is the main input/output interface of the application server and its main purpose is that to route request and responses to the right component of the application. Another important aspect that the implementation must cover is the masking of responses, that is the removal of sensible business data from the responses: i.e. when a user requests for the available taxi, he is supposed to recevive only the taxi code and the taxi location and not all the data which belongs to the Taxi class. Another example can be the masking of authentication data for a system administrator who requests the list of the accounts. /newline This class will also offer a programmatic access for system developers.

API This is the main API class and will implement serveral interfaces explained below. In the case a request is incoming it will translate it into Java messages and call a proper function of the application server. Otherwise, if a request is outcoming of the application server it will build the message for the client.

AuthenticationRequests This interface contains all the methods useful for authenticating a system or a user.

PassengerRequests This interface contains all the methods useful for managing the requests from/to the passengers.

TaxiRequests This interface contains all the methods useful for communicating with the taxi driver application

AdministrationRequests This interface contains all the methods useful for system administrators

APIRequests This interface will contain all the methods useful specifically for the system developers.

2.3.5 Authentication Manager

This component will check if a system or user is making a legitimate login request, and eventually generate and assign a new AuthToken.

SystemAuthenticator This class offers the main methods for logging in or registering a new user or a new dependant system.

ApiToken This class represents ad authorization token which is generated or fetched from the database. It will also offer methods for checking the token validity and permissions to a user.

TokenValidator This class offers functionalities for checking if a request is legitimate given an Auth token.

2.3.6 Request Manager

This component hosts the core of the application, which is the management of taxi, zones and requests.

RequestResolver This class implements the core algorithm for managing

ReservationController

RequestInterface

ReservationControllerInterface

2.3.7 Zone Manger

2.3.8 Taxi Manager

2.3.9 Account Manager

2.3.10 Administration Manager

2.3.11 Notification Manager

2.3.12 Data Model

2.3.13 Database Manager

2.4 Deployment View

2.5 Runtime View

2.6 Component Interfaces

2.7 Architectural Styles and Patterns

The following design patterns have driven the design process of this project:

- MVC: Model-View-Controller design pattern. This pattern separates the business data, the user interface (or the interface between systems) and the core modules that runs the business logic.
- Thin-Client

2.8 Other Design Decisions

3 Algorithm Design

4 User Interfce Design

5 Requirements Traceability

6 References

- MyTaxiService Requirement Analysis and Specification Document

7 Tools and Document Information