Power EnJoy

Design Document

Version 1.0.0

Redaelli Marco 877622 Zanoli Francesco 877471

01/12/2016

Contents

Figure Contents

Chapter 1

Introduction

This document contains the complete design description of *PowerEnJoy*. This includes the architectural features of the system down through details of what operations each code module will perform and the database layout. It also shows how the use cases detailed in the RASD will be implemented in the system using this design. The primary audiences of this document are the software developers but the level of the description is high enough for all the stakeholders to capture the information they need in order to decide whether the system meets their requirements or in order to begin the development work.

1.1 Scope

PowerEnJoy is a car sharing service developed to fit all the reality, as small or large city. The main goals of the system are:

- simplify the access of driver to the service
- guarantee a fair management of reservation and use of electric car

The system architecture will be a three-tier architecture: client, server application and server database. It will be created by using the MVC architectural pattern. The system will be divided into components with respect to the principles leading to good design:

- Each individual component will be smaller in order to be easier to understand
- Coupling will be reduce where possible
- Reusability and flexibility will be increase in order to make easier future implementation

The system will have efficient algorithm in order to increase its performance; in the document will be given special attention to the sharing algorithm.

1.2 Definitions, acronyms and abbreviations

In the document are often used some technical terms whose definitions are here reported:

- **Layer:** A software level in a software system.
- **Tier:** An hardware level in a software system.
- **Relational Database:** A digital database whose organisation is based on the relational model of data, as proposed by E.F. Codd in 1970.
- **Cocoa MVC:** A strict application of MVC principles.
- See the correspondent section in the **RASD** for more definitions.

For sake of brevity, some acronyms and abbreviations are used:

- **DD:** Design Document.
- **GPS:** Global Positioning System.
- **GUI:** Graphic User Interface.
- **API:** Application Programming Interface.
- **MVC:** Model View Controller.
- **ETA:** Estimated Time of Arrival.
- See the correspondent section in the **RASD** for more acronyms and abbreviations.

1.3 References

- Software Engineering 2 Project AA 2016/2017: Assignments AA 2016-2017
- ***PowerEnJoy* RASD v1.0:** Requirements Analysis and Specification Document for *PowerEnJoy*
- **IEEE Std 1016-2009:** IEEE Standard for Information Technology - Systems Design - Software Design Descriptions
- **ISO/IEC/IEEE 42010:** International Standard for Systems and software engineering - Architecture description

1.4 Document Structure

This document is essentially divided in seven main sections:

- **Introduction:** it gives a description of the document and some information about the system design and architecture.
- **Architectural Design:** This is the core of the document. It gives general information about the architectural design. It also describes how the system will be divided into components and how the components communicate. It also has a description of the design pattern and architectural styles that will be used.
- **Algorithm Design:** it gives a description of the main algorithm that will be implemented. Without using a specific language it describe the different steps the algorithm will do.
- **User Interface Design:** it gives a description of the user interfaces of the system and a detailed list of mockup
- **Requirements Traceability:** this section documents the life of a requirement and provides bi-directional traceability between various associated requirements.
- **Appendix:** it provides informations that are not considered part of the actual DD. It includes: software and tools used, project group organisation.

Chapter 2

Architectural Design

2.1 Overview

This chapter describes the software system, the relationships between software components and the interaction between the actors and the system. It also describes resources, that are all the elements used by the external component to design such a services. To do so this section is divided in three different parties:

- **Component View:** Describe the division of the system in layers and how the work flow is organized through them. It contains the UML component diagram and fast definition of all the parties
- **Deployment View:** Explains how the division of the system in Tiers is done.
- **Runtime View:** Describes with some UML Sequence diagram how each software component interacts with each other.

2.2 High level components and their interaction

This section contains all the architectural software specification. In particular it explains all the different part displayed in the UML Component View (2.1) and link them with their dependencies. Before starting analyse all the component of the structure is important to know that the most part of the communications are made with a Client-Server style and only some is Point-to-Point, in plus the system is composed by different tiers:

- **Client:** This is the mobile application used by the users. All the communication done by the client to the server are using the pattern Client-Server
- **Application Server:** This is the core of the system.It is responsible to manage all the user's interaction with the car and the customer service. It is composed by different par as explain bellow

- **Database Server:** This is the memory of the system, all the interactions with this tier are made by the application server with a Point-to-Point communication
- **External Services:** To simplify the complexity of the system without deleting some functionality external services are connected with our Application Server

In particular the application server is composed by different components that are explained in this section.

2.3 Tiers

2.3.1 PowerEnJoy

Type System

Node PowerEnJoy

Description This is the core of the system and contains all the functionality provided to the user, in plus it provided API for the maintenance and the interoperability with other system and it is the main entry for every request coming from the mobile application

Dependencies Google APIs, PayPal API, Maintenance API, PowerEnJoy Database

Resources ??

Operations

- Account API (Sign-up, Sign-in, Login, Personal Information editing, Personal Information handling)
- Reservation API (Reservation Management, Reservation Time)
- Assistance API (Incident management from the system, Error and problem from the system)
- Rent API (Start a rent, End a rent, Bill management)
- Query Problem API (Request and Information to the Maintenance service)

2.3.2 PowerEnJoy Database

Type Database

Node PowerEnJoy Database

Description This is the part of the system devoted to store the data

Resources ??

Operations

- Database API
- Data storare management
- Data presentation
- Security management
- Multiuser access control
- Backup and recovery management ? Data integrity management
- Data transaction management

2.3.3 Mobile App

Type Mobile App

Node Client

Description This is the interface that allow the users to interact with the system, rent car, look for near cars

Dependencies Account API,Reservation API,Rent API, Assistence API

Resources Supported devices (see RASD for further informations)

Operations

- Sign-in, Sign up, login, personal informations editing
- Reserve a car, Rent a car, Looking for a car
- Notification from service
- Assistance request

2.4 System Part

2.4.1 Ride Manager

Type Subsystem of PowerEnJoy

Node PowerEnJoy Server

Description This component provide all the data to calculate the total bill and all the information about the rent as the path followed, the time spent on the car and the number of passenger

Dependencies Account Manager,Data Manager, Car Manager API

Operations

- Rent API
- Reservation and Rent
- Data of the rent to calculate the bill
- Assistance request

2.4.2 Account Manager

Type Subsystem of PowerEnJoy

Node PowerEnJoy Server

Description This component manages users connected to the service. It manages sign-up, sign-in and login functionalities, profile editing. All data are stored on the database through Data API.

Dependencies Data Manager

Operations

- Account API
- Sign-up, Sign-in
- Login
- Personal Information editing

2.4.3 Car Manager

Type Subsystem of PowerEnJoy

Node PowerEnJoy Server

Description This component manages all the information and the interaction with the cars, and communicate with the other service letting they storing all the information they need. Especially with the Ride Manager who is in charge of obtaining all the information about the number of passenger, the state of the battery and the position of the car.

Dependencies Map Service

Operations

- Reservation API
- Data of the car
- Status of the car

2.4.4 Bill Manager

Type Subsystem of PowerEnJoy

Node PowerEnJoy Server

Description It's in charge of calculate and ask for the rent payment, asking to the Ride Manager it can have all the information it needs to calculate the total cost and pretending to the external system the amount requested

Dependencies PayPal API, Ride Manager.

Operations
- Calculate total cost, apply discount and overtaxes
- Request the amount of the rent to the external payment method

2.4.5 Problem Manager

Type Subsystem of PowerEnJoy

Node PowerEnJoy Server

Description This is the part of the system dedicated to the customer service, able to request assistance for some accident and to request maintenance on a particular car.

Dependencies Query Problemes API

Operations
- Assistance API
- Ask for maintenance on a car
- Helping Customer with problems of the system

2.4.6 Data Manager

Type Subsystem of PowerEnJoy

Node PowerEnJoy Server

Description This component provides access to all of the data contained in the database. It provides various functions that allow entry, storage and retrieval of large quantities of information and provides ways to manage how that information is organized.

Dependencies PowerEnJoy Database

Operations
- Data API
- Data access
- Data presentation
- Data organization

2.5 Component View

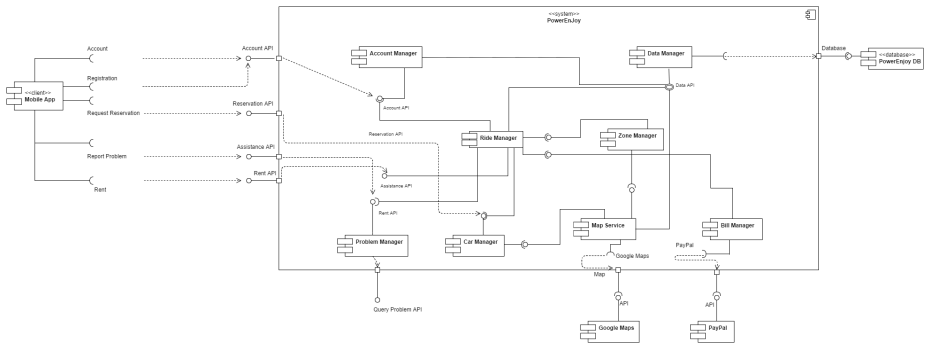

Figure 2.1: Component diagram of the entire system

2.6 Deployment view

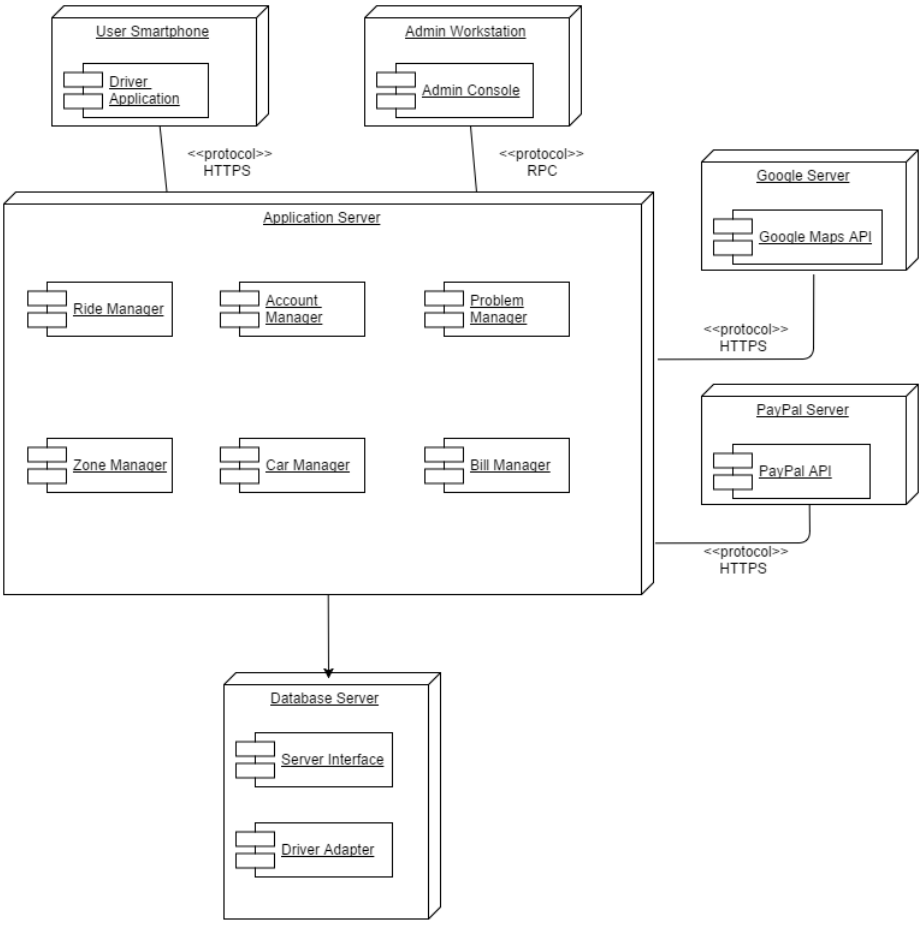

Figure 2.2: deployment diagram for PowerEnJoy

2.7 Runtime view

2.8 Component interfaces

2.9 Selected architectural styles and patterns

Various architectural and logical choices have been justified as following:

- A **3-tier architecture** has been used: client, server application and server database.

This is due to the fact that we're developing an overall light application that doesn't need a lot of computing power, especially on the client side. Therefore, it is possible to structure the system in a logically simple and easily understandable way, without having to lose much in terms of optimization. Besides, this allows to obtain a good compromise between thin client and database tiers, and a clear correspondence between tiers and layers (i.e. no layer is distributed among multiple tiers).

- For similar reasons, a **SOA (Service Oriented Architecture)** has been chosen for the communication of the application server with the front ends. This improves flexibility, through modularity and a clear documentation, and simplicity, through an higher abstraction of the components.
- The entire system is designed within the principles of **Modular programming**, focusing on assigning each functionality to a different module. This greatly improves extensibility and flexibility, and allows for an easy implementation of the APIs.
- The **Client&Server** logic is the most common, simple way to manage the communication both between client and application server, and between application server and database.
- The **MVC (Model-View-Controller)** pattern, besides being a common choice in object-oriented languages like Java, allows for a clear logical division of the various elements of the program.
- The **Adapter** pattern is largely used for portability and flexibility of the various modules.

2.10 Other Design Decision

The system uses **Google Maps** to perform all the operations related to maps, i.e. map and position visualization, geolocalization (either through GPS or user input) as well as distance, route and ETA calculations. This is an easy, fast to develop solution that relies on a worldwide, well-known and well-established software.

Chapter 3

Algorithm Design

Chapter 4

User Interface Design

Chapter 5

Requirements Traceability

5.1 Requirements Traceability

In this section are mapped all the functional requirements identified in the **RASD**, grouped by the **Use Case** they refer to.

5.1.1 Registration

Functional Requirements	***Design Elements***
FR1	**Account manager** does not consider unfulfilled registrations. No data is stored.
FR2	The **Account manager** consider a valid registered user (allowed to rent a car) after the first login, if the status of the user stays "inactive" for more than one day after the registration date the **Account manager** delete the data about this user.
FR3	**Account manager** does not consider unfulfilled registrations.
FR4	**Account manager** ensures the uniqueness of users.
FR5	Validation of all the form is done by the **Mobile Application** even the photos taken from the camera and not from the gallery

5.1.2 Login

Functional Requirements	***Design Elements***
FR6, FR7	**Account Manager** check the existence of the user's email and the correspondent password before allowing the user to log-into the system
FR8	Blank fields let the **Client Application** to ignore the login.
FR9	The **Client Application** provide this functionality and the **Account Manager** takes care of the procedure to follow to allow the user have another name
FR10,FR11,FR12	The **Account Manager** take only exiting user on the database and handle all the procedure to restore a password, from the generation of the new one until the sent of the mail with the new access code.
FR13	The **Account Manager** is allowed to generate only one password a day for one user.
FR14	The **Client Application** remeber email and password after a login until the user do a logout. It's also provide the email and password every time the system requires its.

5.1.3 Logout

Functional Requirements	***Design Elements***
FR11	The Passenger can perform the Logout procedure by interacting with the **PS Web View** or **PS Application View**.
FR12	All the logic about the navigation inside the application and in general about the building of the View is contained into the **PS Web View** or **PS Application View**.
FR13	The **PS Request Creator**, **TD Request Creator** and the **RESTful Service** are capable of handling connection errors without making the relevant systems crash.

5.1.4 View Request and Reservations

Functional Requirements	***Design Elements***
FR14	The data is accessible through the **Query Manager**, that calls the **Model Query Service** to reach the **Passenger DB Adapter**.
FR15	**PS Application View** and **PS Web View** handle this interface logic.
FR16	**PS Application View** and **PS Web View** handle this interface logic.
FR17	The **PS Request Creator** and the **RESTful Service** are capable of handling connection errors without making the relevant systems crash.

5.1.5 Handle Personal Profile

Functional Requirements	***Design Elements***
FR18	This validation is done by the **Profile Manager**.
FR19	The **Profile Manager** uses the interfaces exposed by the **Query Manager** to guarantee so.
FR20	This information is sent as a response to a profile modification request by the **RESTful Service**
FR21	The **PS Request Creator** and the **RESTful Service** are capable of handling connection errors without making the relevant systems crash.

5.1.6 Taxi Reservation

Functional Requirements	***Design Elements***
FR22	This validation is done by the **Taxi Ride Manager**.
FR23	The **Taxi Ride Manager** uses the interfaces exposed by the **Query Manager** to guarantee so.
FR24	This information is sent as a response to a Taxi Reservation request by the **RESTful Service**
FR25	The **PS Request Creator** and the **RESTful Service** are capable of handling connection errors without making the relevant systems crash.

5.1.7 Taxi Request

Functional Requirements	*Design Elements*
FR26	Data validation is done by the **Taxi Ride Manager**. If the data don't respect the constraints, then the **Taxi Ride Manger** informs the related Passenger of the issue.
FR27	When a Passenger sends a Taxi Ride request to , the **RESTful Service** handles it. Then the **Taxi Ride Manager** completes the **Taxi Ride** request by adding the Zone of the Starting Address. In addition, this component keeps the model updated by adding the ride to the persistent data layer. Later on the **Queue Manager** picks the ride from the Model and associates it to a Taxi Driver. In conclusion this component uses the **Dispatcher** to notify the Passenger with notification message that contains the Taxi Driver ID and Travel Time.
FR28	Through the **Dispatcher**, users can be reached in every moment by notification messages.
FR29	The **PS Request Creator**, the **PS Receiver**, the **RESTful Service** and the **Dispatcher** are capable of handling connection errors without making the relevant systems crash.

5.1.8 Notify Problem

Functional Requirements	*Design Elements*
FR30	The **TD Application View** is provided with a specific screen that allows the Taxi Driver to report a problem. In addition, Taxi Driver has a chance to specify whether or not he can solve the given problem These data are then sent to the **RESTful Service** in order to be processed by the **Taxi Driver Manager**.
FR31	This task is done by the **Taxi Driver Manager**.
FR32	This is sent a response to a Notify Problem request by the **RESTful Service**.
FR33	This is done by the **Taxi Driver Manager** via the **Dispatcher**.
FR34	The **TD Request Creator**, the **TD Receiver**, the **RESTful Service** and the **Dispatcher** are capable of handling connection errors without making the relevant systems crash.

5.1.9 End of Ride

Functional Requirements	***Design Elements***
FR35	The Taxi Driver pushes a button on the **TD Application View** that calls the **TD Request Creator**, which sends a request to the **RESTful Service**. The request is then forwarded to the **Taxi Ride Manager** by the **RESTful Service**. The **Taxi Ride Manager** handles the operation.
FR36	When a Taxi Driver has notified the end of his current Taxi Ride, he is associated to a Available status. Then the **Queue Manager** is waiting for Taxi Drivers to become Available to enqueue them into their current Zone.
FR37	The response to the request made to achieve FR35 contains the result of the **Taxi Driver** action.
FR38	The **TD Request Creator** and the **RESTful Service** are capable of handling connection error problem without making the relevant systems crash.

Appendix A

Appendix

A.1 Tools

- **TeXstudio:** LaTeX editor used to write the document.
- **Alloy Analyzer 4.2:** Used to build an Alloy Model and to check its consistency.
- **StarUML:** Used to build UML Class Diagram and UML Use Case Diagrams.
- **draw.io WebSite:** Used to design the UML Sequence Diagram and the Mockup (wep portal and smartphone app).

A.2 Hours of work

In the following are listed the hours of work that each member of the group did:

1. Marco Redaelli: 41 *hours*
2. Francesco Zanoli: 41 *hours*

A.3 Version History

In the following are listed the differences between versions:

1. First version