# Power EnJoy

**D**esign **D**ocument

Software Engineering 2

A.A. 2016/2017

Version 1.0

Emanuele Ghelfi (mat. 875550)
Emiliano Gagliardi (mat. 878539)

December 3, 2016

# Contents

# 1 Introduction

## 1.1 Purpose

This software design document describes the architectures and the system design of Power EnJoy. It's intended mainly for developers but it has a hierarchical structure. It completes the RASD and defines the component that led to the satisfaction of the goals previous defined. It starts from an high level description of the architecture and then it goes into detail.
This document has to identify:

- Architecture of the system
- Interactions between components
- Main algorithms of the system

The main purpose is to gain a general understanding of how and why the system is decomposed and how individual components work together.

## 1.2 Scope

PowerEnJoy is a software that manages a car sharing service for electric cars. The aim of this software is to make simple and quick the reservation of cars. So the system should provide users with real time information about availability of cars, their status and their positions. Users, after the reservation, can directly get their car in pre-defined parking areas. The service will be accessible only to registered users, giving some personal information and data needed to the payment. The price of the ride is computed with a fixed amount of money per minute, displayed by the car, and finally charged. To avoid useless reservation, where a user doesn't pick up the car, the reservation expires after a fixed time, the car returns available, and the user is charged with a fee. Cars must be locked in the safe areas, and only users that have made the reservation can unlock them. The software has to provide also management functionality for administrators and operators in order to ensure a simple managing of the system.

## 1.3 Definitions, Acronyms, Abbreviation

- RASD: Requirements and Specifications Document
- DD: Design Document (this document)
- JSON
- REST
- RESTful
- HTTP

- JDBC
- API

## 1.4 Reference Documents

- RASD released before this document.
- Assignments AA 2016-2017.
- DD from previous years.

## 1.5 Document Structure

**Introduction**: this is a general overview of the document.
The *Purpose* part describe the audience and the main goals of this document.
The *Scope* part has to provide a description and scope of the software and explain the goals, objectives and benefits of the project.
*Reference Documents* are previous documents of this project and documents used as examples and reference.
**Architectural Design**: this section explain the relationship between the modules to achieve the complete functionality of the system (requirements defined in the RASD).
It contains an high level overview of how responsibility of the system were partitioned and then assigned to subsystem (components).
In this part of the document are identified each high level subsystem and the roles or responsibility assigned to it in order to achieve a more detailed comprehension of the software to be. It's also described how these components collaborate with each other in order to achieve desired functionality. There is a focus on the interface provided by individual components in *Component Interfaces* section.
*Deployment View* gives a description of how the software to be it's intended to be deployed.
*Runtime View* gives a description of the interaction between components in the most important use case of the system.
In the section of *Selected Architectural Styles and Patterns* are described which styles and patterns have been followed in the realization of the system. There is a focus on the rationale of these decisions.
**Algorithm Design**: this section explains the most important algorithm of the software to be. Pseudo–code has been used in order to avoid unnecessary implementation details.
**User Interface Design**: this section refers to the same section in RASD and provides some extensions.
**Requirements Traceability**: this section describe how requirements defined in RASD have been mapped to system components defined in section 2.

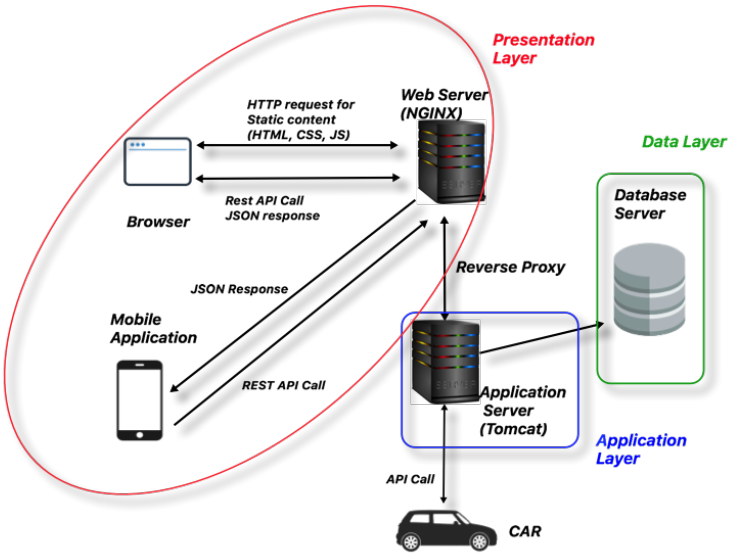


Figure 1: High level architecture

# 2 Architectural design

## 2.1 Overview

## 2.2 High level component and their interaction

## 2.3 Component view

### 2.3.1 Car proxy component

Car proxy is the abstraction of cars in the server. It must be invoked by other internal subsystems that need information about cars physical status (real time informations), like position, battery life etc. This component absolutely doesn't care about information related to reservations or rides.
The main purpose of this component is to request cars using their API, but its implementations should also guarantee that the systems working on cars don't have to support high amount of parallel request. In order to do that, car proxy will use the database, storing a physical characteristic related to the timestamp of the API call that has provided that information. When a request arrives, car proxy decides which data should be provided: the one got through the API call to the car, or the one stored in the database. If the stored data are sufficiently recent then they can be provided, otherwise it is needed to make the request.
As it can be seen in the diagram, this component has a considerable fan-in, it could be necessary in future to make it scalable.

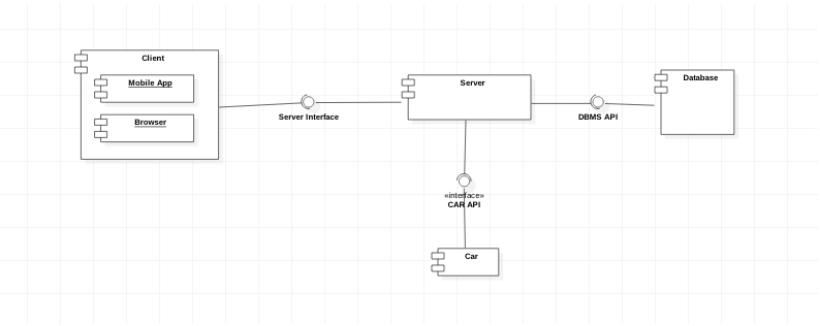


Figure 2: High level component diagram

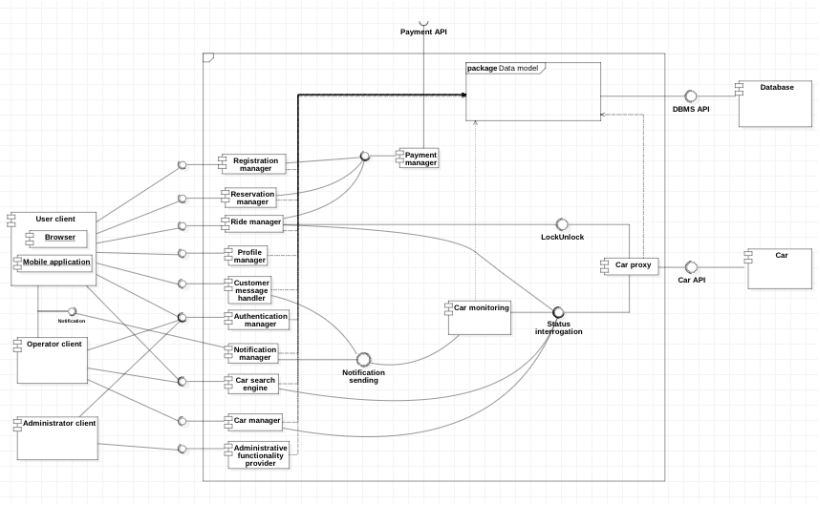


Figure 3: More detailed component diagram

### 2.3.2 Car monitoring component

This component is substantially a daemon. It periodically asks to the car proxy component the informations that operators need to do car maintenance. If a damage has been detected, the notification component should be called, in order to communicate it to an operator.

### 2.3.3 Ride manager

Ride manager is the component that takes care abut the data in the server corresponding to rides in the real world. Furthermore, when a ride is set to pause, it use the car proxy component to lock or unlock the car.

### 2.3.4 Reservation manager

This component manages users reservation request, setting the status of a car from available to reserved and vice versa. It should also take care about reservation expiration, resetting the car state from reserved to available, and taxing the user through the component payment manager.

### 2.3.5 Registration manager

This component accepts request from guest of joining the Power Enjoy service. It checks if the guest has a valid drive license and adds it to the users. It also verifies the validity of the payment informations received by the guest using the services provided by the payment manager component.

### 2.3.6 Profile manager

This component reply to users that want to see their past utilization of the power enjoy services.

### 2.3.7 Customer messages handler

This component receives messages about users regarding malfunctions of cars, and use the notification manager to notify the operator that should take care about that.

### 2.3.8 Authentication manager

The authentication manager the component that manages the login of user, operator, and administrator, and administrates session.

### 2.3.9 Car search engine

This component interrogates the car proxy component and get cars position, performing the two type of car research.

Figure 4: Class diagram of the data model

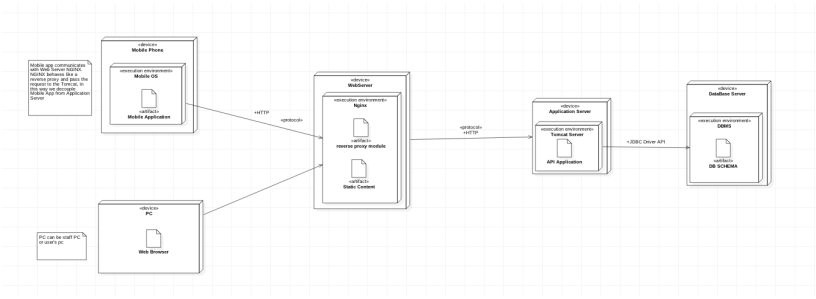


Figure 5: Deployment view

#### 2.3.10 Car manager

The car manager component can be used by operators to change car status in the server. For instance, if an operator is going to work on a car, it should use this component to switch the car status to "under maintenance".

#### 2.3.11 Administrator functionality provider

This component provides the functionalities accessible only by the administrators that are listed in the RASD document.

#### 2.3.12 Payment manager

This component uses external API of the accepted payment service. It should be used not only for carry out the payments, but also to verify during the registration of a user that the provided payment informations are correct.

#### 2.3.13 Notification manager

Notification manager is used to send notifications to users (when their reserved car changes status), and to operator (when the system detects that an intervention on a car is needed, or a user make a communication).

## 2.4 Data model

## 2.5 Deployment view

## 2.6 Runtime view

## 2.7 Component interface

## 2.8 Selected architectural styles and patterns

# 3 User Interface Design

This section refers to the same section on RASD document. Here there are only some extensions in order to go more into details.

# 4 Requirements Traceability

In this section is described how requirements defined in the RASD are mapped to components described in this section. The set of components have to fulfill all the requirements defined in the RASD.
It is understood that Mobile Application and CarOS are used in interactions with users and cars so here aren't listed in order to avoid useless repetition. Also the Persistency Manager is used in order to keep coherency with data.
Here there is a list of all goals of out system (component here are associated to goals but indirectly to all requirements that ensure the fulfillment of the goal).

- [**G1**.**1**]Only user should be able to reserve a car.
  - **Authentication Manager**: manages user logins and sessions.
  - **Reservation Manager**: manages reservation and allows only registered user to make a reservation.
  - **Registration Manager**: allows the registration of guests and manages all information related to registration.
- [**G1**.**2**]User should be able to unlock reserved car when they are close to it.
  - **Ride Manager**: allows user to unlock a car.
  - **Reservation Manager**: allows user to reserve a car.
  - **Car Proxy**: allows communication with car.
- [**G1**.**3**]User should be aware of how much they are going to pay during the ride.
  - **Ride Manager**: communicates to the car the current ride cost.
  - **Car Proxy**: same as before.
- [**G1**.**4**]If reserved car pass under maintenance, the user that made the reservation must be notified.
  - **Car Monitoring**: monitors periodically the status of the cars and calls notification manager.
  - **Notification Manager**: notifies user and operators.
  - **Car Proxy**
- [**G1**.**5**]User should be able to set pause status during a ride.
  - **Ride Manager**: manages all stuff related to the ride.
- [**G1**.**6**]User should be able to restart a paused ride.
  - **Ride Manager**

- [**G2**]The reservation of two (or more) cars at a time must be forbidden.
  - **Reservation Manager**
- [**G3**]Users and Guests must be able to search cars.
  - **Car Search Engine**: searches car with respect to user's preferences.
  - **Car Proxy**
- [**G4**]Induce users to keep a virtuous behavior.
  - **Ride Manager**: manages the calculation of discounts and fees related to a ride.
  - **Car Proxy**
  - **Payment Manager**: manages the payments.
  - **Reservation Manager**: if a reservation expires it manages the fee.
- [**G5**]Users have to pay an amount of money based on the ride's duration.
  - **Ride Manager**
  - **Payment Manager**
- [**G6**]Guarantee a ready maintenance of cars.
  - **Car Monitoring**
  - **Notification Manager**:notifies operators when a car needs maintenance.
  - **Car Manager**: allows operators to know car status and car positions. Allows also operators to set car as available after maintenance.
  - **Customer Message Handler**: receives messages from users about malfunctions and notifies operators through notification manager.
  - **Car Proxy**
- [**G7**]Admins must be able to manage the system.
  - **Administrative Functionality Provider**: allows administrators to manage safe areas and operators