Politecnico di Milano

A.A 2016-2017

Design Document

Version 1.0

PowerEnjoy

Instructor : Prof. Di Nitto

Authors:
Amico Simone
Chianella Claudia Beatrice
Giovanakis Yannick

CONTENTS

1. INTRODUCTION

1.1 Purpose

The presented document is the Software Design document (SDD) for the PowerEnJoy platform project. The main purpose of this document is to be a guideline for the concrete implementation of the platform, provide developers with high level descriptions of the main algorithms, describe the architectural styles and patter, and generally establish the design standards for the development phase.
This document is intended for stakeholders, software engineers, and programmers and must be used as reference throughout the whole development of the system. The secondary audience for this document includes system maintainers and developers who wish to integrate the platforms services within their own software. The design choices listed here must be respected throughout the whole development of the platform and any other further expansion of the codebase.

1.2 Scope

The descriptions listed in this document define the design language that must be used by design stakeholders, and implement the Design leads to code philosophy. These design standards represent constraints to the development of the codebase, with all unspecified design decisions left to the developers. Key reasons for the design language described in this document are:

1. To facilitate the development, integration, expansion and maintenance of the platform.

2. To define a business identity (kept consistent in UX and marketing design).

3. To implement the requirements listed in the RASD of the project in a consistent way.

1.3 Definitions,Acronyms,Abbreviations

Throughout this document, the following definitions will be applied without further explanations:

- **Platform**: the set of software applications and hardware infrastructure that are part of the PowerEnjoy service. The platform includes:
  - Back-End Server application
  - Web Application
  - Mobile Application
  - On-board Display
  - MySQL Database

  Other third party software may be necessary to interface different components or support the listed applications.
- **System**: any individual component of the platform.
- **Back-End**:the software run on the back-end server of the platform which is used to handle the communication between the user applications. The term also addresses all the necessary software components that are needed to store data, perform calculations and manage the hardware (e.g. an operating system).
- **User Application**: set of applications that are used by a user which are the Mobile Application ,Web application and On-Board display application. [1]
- **User**: any person registered and authorized to use the above mentioned systems.

In addition this document will contain the following acronyms:

- **RASD**: Requirements and Analysis Specifications Document
- **DD**: Design Document
- **DBMS** : Data Base Management Systems

[1] For more informations check Section 2.4.2 on the *RASD*

- **UX**: User Experience
- **API**: Application Programming Interface
- E TANTI ALTRI

1.4 Reference Documents

- IEEE 1016-2009: "Software design description"
- Project description: Assignments AA 2016-2017.pdf
- UML Language Reference : `https://www.utdallas.edu/œchung/Fujitsu/UML_2.0/Rumbaugh--UML_2.0_Reference_CD.pdf`
- JAVA/GLASSFISH/RESTFUL/GOOGLEMAPS API..

1.5 Document Structure

The presented DD is divided in sections and structured as follows:

- Section 1 - Introduction: contains support information to better understand the presented document.
- Section 2 - Architectural design: contains a description of the architectural styles and patterns selected for the platform, which serve as an implementation guideline for developers.
- Section 3 - Algorithms design: contains a high-level description of the core algorithms of the back-end.
- Section 4 - User interface design: contains a description and a conceptual preview of the user interface and UX.
- Section 5 - Requirements traceability: links the decisions described in this document to the requirements specified in the RASD.
- Section 6 - Effort spent: contains a summary of the hours spent in producing the document.

2. Architectural Design

2.1 Overview

An essential part of the platform design is to satisfy not only functional requirements but non-functional requirements as well.This means that architectural choices are crucial at this point of the development for the sake of achieving the desired output in terms of requirements ,performance ,scalability and user experience.// An important focus will be set on how the different systems interact with each other and the back-end to obtain the events as required by sec. 3 of the RASD.Furthermore choices regarding upscaling and redundancy will be discussed.//

2.2 Component View

This section focuses on the component overview giving an insight on their core functionality and various interfaces.//

2.2.1 RESTful API

The RESTful API is a gateway communicator between clients and the back-end system: it's a stateless service which provides methods for data submission or requests returning the requested computations as a result.// RESTful API are an optimal solution for an application that must handle a vast number of users on a different number of platforms as it allows to guarantee the same user experience on all platforms. Furthermore the architectural properties positively affected by the constraints of the REST architectural include many important quality requirements such as performance, scalability and reliability.

2.2.2 User manager

This component handles user data, registration and authentication.The user manager has direct access to the DBMS and receives read and write requests from the RESTful API.

2.2.3 Notification manager

This component is responsible for implementing the push notification service towards the user-side applications. It is necessary to have this type of module running in the back-end because there are cases in which a communication must happen between the clients and the back-end, but no direct request is made to the RESTful API by the clients: for example when the reserved car is nearby, the system must send the user a 'ready to unlock' notification.

2.2.4 Vehicle manager

The car manager component's job is to manage all the vehicles throughout the city.It can access the DBMS to query vehicle information but stores essential vehicle information locally such as location ,battery level and number of seats. When the search manager gets triggered it forwards requests to the vehicle manager to get all available car in compliance to the user input.

2.2.5 Search manager

The search manager component handles all incoming search requests forwarded by the user applications and interfaced through the RESTful API. The main functionality is to handle the user input and query the vehicle manager accordingly, returning the desired output to the user.
In the event of a booking request the search manager forwards the demand to the Ride Manager.

2.2.6 Ride Manager

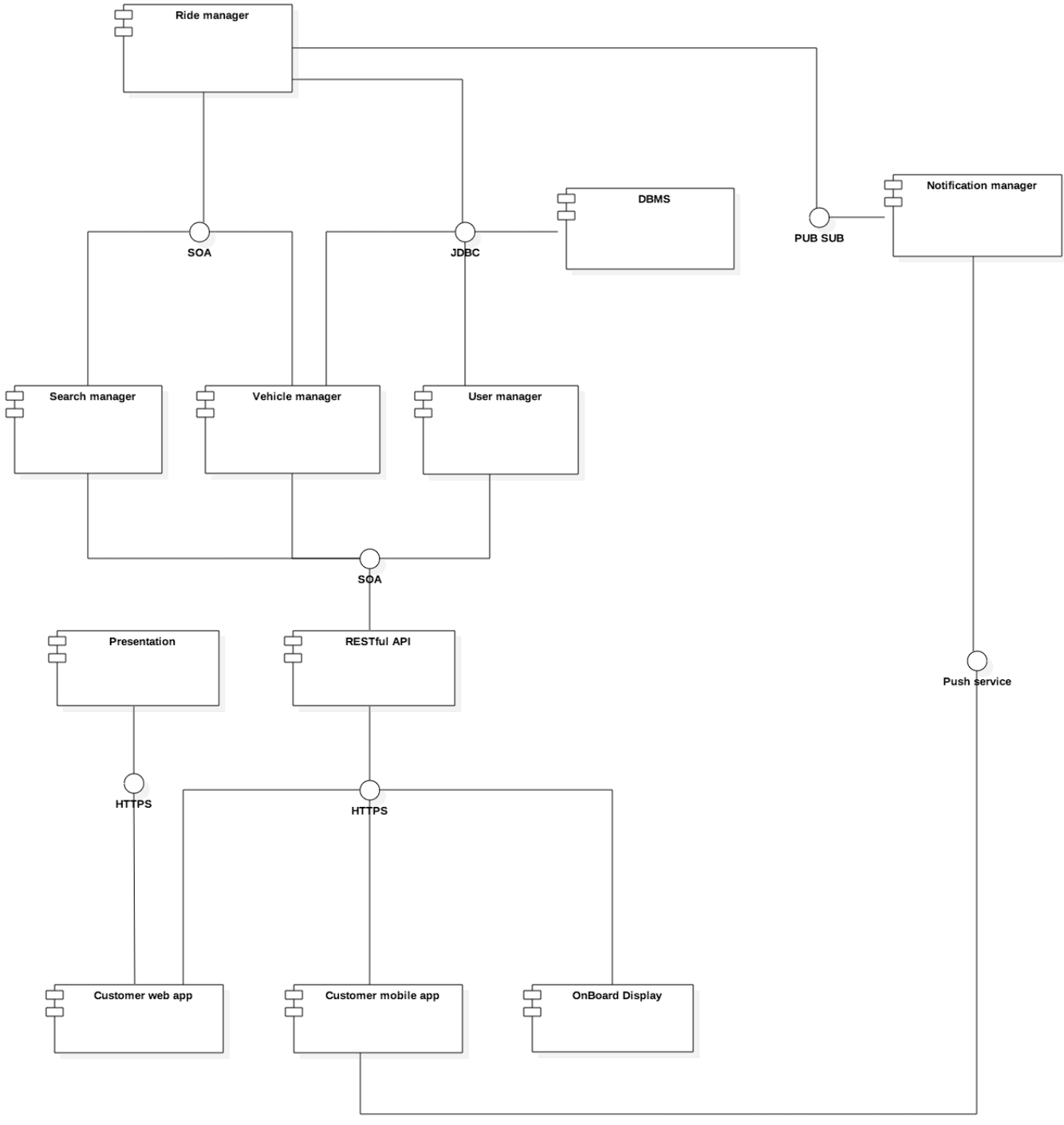
The Ride manager component examines all incoming reservation request and creates *Ride objects* according to the user selection and user data.The ride manager is connected to the following components:

- *Vehicle manager*: need to communicate and receive communications about car status changes in terms of availability, battery level and location.
- *Notification manager*: need to inform the user about remaining time to unlock the car or send the user 'ready to unlock ' notifications.

Moreover it sends messages to the on-board display to inform the user about the response to a *end ride request*.

- *DBMS*: need to update the user reservation and payment history once a ride has been marked as completed.

2.2.7 Component diagram


Ride manager
DBMS
Notification manager
SOA
JDBC
PUB SUB
Search manager
Vehicle manager
User manager
SOA
Presentation
RESTful API
Push service
HTTPS
HTTPS
Customer web app
Customer mobile app
OnBoard Display

2.3 Deployment View

2.4 Runtime View

2.5 Component Interfaces

2.6 Selected architectural styles and patterns

2.7 Other design decisions

3. Algorithm Design

3.1 Search and Reserve Car Manager

The Search and Reserve Car Manager provides all the methods to manage and dispatch the incoming search and reserve requests.

- **showAvailableCars(Address):** searches into the database for the available cars within 5km from the Address and show them to the user on the map.
- **reserveCar(Car):** marks the car as reserved, starts the timer and shows it to the user in the application with the shortest path to reach the car.
- **unlockCar(Car, Position):** if the user is near the car, unlocks it and stops the timer.

3.2 Ride and Park Car Manager

The Ride and Park Manager provides all the methods to manage ride requests.

- **startRide(PinCode, MoneySavingOption, Destination):** if the PinCode correspond to the one assigned to the user, starts the timer of the ride, shows it on the on-Board application screen along with the other informations such as charge level, gps, etc. If the flag MoneySavingOption is true, then searches and shows to the user the path to the best place where to park in order to save money.
- **parkCar(Car):** If the car position is not in the parking area shows an error message and doesn't let the user to end the ride. If the car position is in the parking area, retrieves and saves informations about the presence of other people in the car from the sensors and tells the user that he/she can park there.

- **lockCar(Ride):**Waits until all the doors are closed. Stops the timer. If there were at least two passengers, adds the relative discount. If the battery level is over a certain level adds the relative discount. If the battery level is under a certain level adds the relative fee. If the position of the Car is too far away from the nearest power grid adds the relative fee. If the car is plugged in a power grid adds the relative discount. Calculates the final fee that the user has to pay and charges it to him/her. Finally marks the car as available.

3.3 Low Level Algorithm Description

3.3.1 Search and Reserve Car Manager

- **showAvailableCars(Position address)**{

```
List<Car> nearCars = new ArrayList<Car>;
for(Car c in CarFactory.availableCars)
     if(distance(c.position, address)<=5)
          nearCars.add(c);
Console.showList(nearCars);
}
```

- **reserveCar(Car car. User user)**{

```
car.status = reserved;
Timer t = new Timer();
t.start();
Path path = GoogleMaps.getPath(user.position,car.position);
Console.showInfo(path,timer);
}
```

- **unlockCar(Car car, User user)**{

```
if (distance(user.position,car.position)<0.002); //2 meters
     car.unLock();
else
Console.showMessage("You are not near the car!");
}
```

3.3.2 Ride and Park Car Manager

- **startRide(Car car, User user, int pinCode, MoneySavingOption mso, Position destination)**{

```
if (user.pinCode == pinCode){
     Timer t = new Timer();
     t.start();
     if (mso.status == True){
          Path path = GoogleMaps.getPath(car.position,destination);
          Ride ride = new Ride(car,user,path,timer);
          }
     else
          Ride ride = new Ride(car,user,timer);
     }
else{
     Console.showMessage("Error, PinCode not valid!");
     return null;
     }
return ride;
}
```

- **parkCar(Car car, Ride ride)**{

```
while (car.position is not in parkingArea )
     Console.showMessage("You are not in a Parking Area.");
if (car.position is in parkingArea){
     ride.numberPassenger = car.passenger;
     car.status = lock;
     }
}
```

- **lockCar(Car car,Ride ride)**{

```
if (car.doors is not close){
     ride.status = notFinished
     return false;
     }
ride.status = finished;
ride.timer.stop();
car.lock();
```

```
float fee = ride.calculateFee();
if (ride.numberPassenger >=2)
     totalFee = ride.fee - 0,1*ride.fee;
if (car.batteryLevel> 0,5)
     totalFee = ride.fee - 0,2*ride.fee;
if (car.isInCharge == true)
     totalFee = ride.fee - 0,3*ride.fee;
if (car.position is not in powerGrid or car.batteryLevel>0,2)
     totalFee = ride.fee + 0,3*ride.fee;
car.status = available;
ride.fee = totalFee;
return true;
}
```

4. User Interface Design

5. REQUIEREMENTS TRACEABILITY

6. Effort Spent

7. REFERENCES