

Version: 1.0.0

Release date: 25-11-2016

Riccardo Cattaneo 873647

Fabio Chiusano 874294

PowerEnJoy, Design Document

Table of Contents

1 Introduction 3

1.1 Purpose 3

1.2 Scope 3

1.3 Definitions, Acronyms, Abbreviations 4

1.4 Reference Documents 5

1.5 Document Structure 5

2 Architectural Design 6

2.1 Overview: High level components and their interaction 6

2.2 Component view 6

2.3 Deployment view 6

2.4 Runtime view 6

2.5 Component interfaces 6

2.6 Selected architectural styles and patterns 6

2.7 Other design decisions 6

3 Algorithm Design 7

4 User Interface Design 8

5 Requirements Traceability 9

6 Effort spent 10

7 References 11

# Introduction

## Purpose

This document describes the hardware and software architecture of the PowerEnjoy System. Therefore, it outlines hardware tiers and all parts of the software and how they will work and cooperate together.

In particular, this document contains information about:

* Architecture Design with related pattern used;
* Main components and software interface design;
* Runtime behaviour of the system
* User Interface Design

## Scope

PowerEnjoy is a car-sharing service based on a mobile web application.

It allows the user to see, thanks to the help of an external search-on-a-map handler, where the electric cars are, only if they are close to an address provided by either the user or his/her GPS Location.

Hence, it allows users to reserve an electric car and to get on board when he/she is close to it.

The car software takes into account the minutes of usage of the car, the number of passengers, the battery level and the location of release.

The system then calculates the charges the user for the ride.

The main purpose of the system is to create a new and smart car-sharing service, that incentivize virtuous and green behaviours.

## Definitions, Acronyms, Abbreviations

* DD: Design document.
* RASD: Requirements Analysis and Specification Document.
* JSE: Java Serial Edition.
* JEE: Java Enterprise Edition.

## Reference Documents

## Document Structure

This document specifies the architecture of PowerEnJoy spreading from the general into the specific. Also it describes the architectural decisions and tradeoffs and justifies them. The design was guided by a top-down process approach and the document structure reflects this tactic.

The document is organized as follows:

1. Introduction, provides a synopsis of the architectural descriptions.
2. Architectural design, provides a general description of PowerEnJoy including its  functionality and matters related to the overall system and its design.
3. Algorithmic design.
4. User Interface design.
5. Requirements traceability.
6. Effort spent.

# Architectural Design

## Overview

We propose to make a web app that will give users a comfortable way to use our service. The reasons according to which we have chosen a web app instead of the web site are explained in the “Proposed system” chapter of the RASD document.

The architecture of PowerEnjoy Service has three tier:

* The presentation will be provided to the user thanks to the cooperation between the client and the Web Tier of the Server;
* The Logic will be provided by the Main Server;
* The Persistence funzionalities will be provided by a Database Server, which communicate directly to the Main Server.



Client Tier

Presentation Tier

Mobile Web-App

Web Tier

Presentation Tier



Business Tier

Business Logic Tier

Main Server



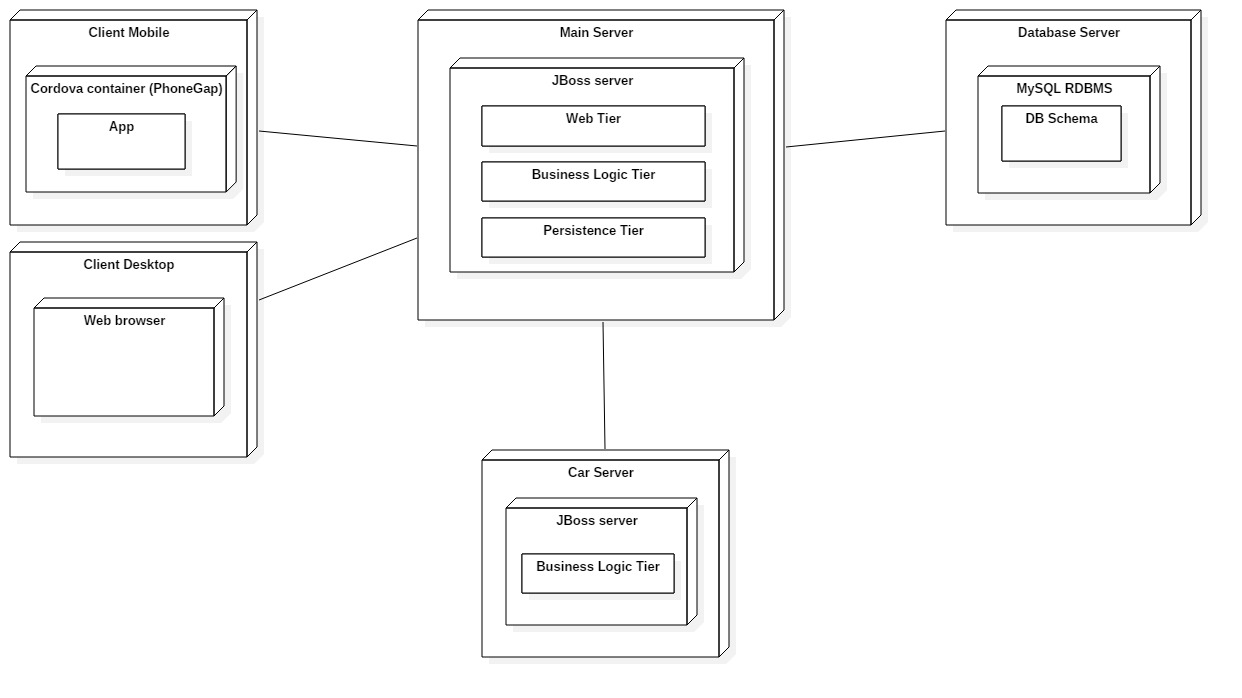
EIS Tier

Database Tier

Database Server

## High Level Components and their interactions

### Overview



* Client Tier, represented by the web app, that will be available to all the major mobile operating systems and will be developed in a way to communicate with the PowerEnjoy servers through RESTful API and HTTP requests, consequently the front-end will lie on the server;
* On the server side, the Main Server contains two software tiers:
  + Web Tier, that manages the communication with the web app.

It receives requests from the app and provides it and updated front-end, thanks to JavaServerPages (JSP);

* + Business Logic Tier, that elaborates requests from client thanks to the embedded logic and manages interactions with Database, thanks to the Java Percistence API (JPA);
* Database Tier, that contains and manages persistent data in an efficient way.

Of course, the electric cars must be able to communicate with the server, so they must be provided with an Internet connection and an on-board computer that must be able to run Java software. Cars will have small logic, mainly dedicated to communicate sensors data with Logic Tier of the Main Server.

The Main Server will also take advantage of external handler, such as Payment Handler and Search-on-a-map Handler, in order to manage successfully functionalities provided to users.

### Technologies used

* The web app will be implemented using PhoneGap framework:
  + Interactions with the Server will be done via HTTP calls, using RESTful services offered;
* The Main Server will be implemented with Java Enterprise Edition (JEE):
  + Communication with the web app will be possible thanks to Java API for RESTful Web Services (JAX-RS);
  + The front-end functionalities, that clients need, will be provided with Java Server Pages (JSP);
  + Business Logic will be implemented using Session Java Beans, and communication with WebTier will be managed with the EJB Container;
  + Data management and interaction with Database Server will be implemented thanks to Java Persistence API (JPA), and will be composed of Entity Java Beans;
* For the Database Server we will use MySQL.

## Component view

## Deployment view

## Runtime view

Looooots of sequence diagrams.

## Component interfaces

Components class diagrams, very detailed. Text explanations of all classes: what the class does, what its methods do, design constraints, permormance issues…

## Selected architectural styles and patterns

Three tiers, event-based, MVC…

## Other design decisions

Maybe data model (class diagrams, Object Relationship diagrams) and data storage (ER diagrams, SQL stamements for tables creation).

# Algorithm Design

- GPS usage (not sure)

- Search for near cars:

1. App asks server for near cars and sends it its location;
2. Server retrieves cars locations quickly thanks to some spatial data structures (https://en.wikipedia.org/wiki/Spatial\_database);
3. Server sends the locations to the app;
4. App is happy.

- Search for near parkings: similar to search for near cars.

# User Interface Design

All the screens!

How to go from a screen to another.

Description of each screen.

Some scenario examples.

# Requirements Traceability

Table with the following columns:

- Requirement

- Description

- Design reference (all the references in this document that together satisfy the requirement).

# Effort spent

# References