Power EnJoy
Design Document

Niccolo' Raspa, Matteo Marinelli

December 6, 2016

Software Engineering 2 Course Project

Contents

1 Introduction

1.1 Purpose

This is the Design Document for the Power Enjoy Service. It's aim is to provide a functional description of the main architectural components, their interfaces and their interactions, together with the algorithms to implement and the User Interface Design. Using UML standards, this document will show the structure of the system and the relationships between the modules. This document is written for project managers, developers, testers and Quality Assurance. It can be used for a structural overview to help maintenance and further development.

1.2 Scope

PowerEnjoy is a digital management system for a car-sharing service that exclusively employs electric cars. It allows registered clients (Power Users) to use a vehicle paying only on the basis of the actual use during each individual rental. For a more detail description of the domain and the requirements please refer to the Requirement and Specification Document.

The software system is divided into four layers, which will be presented in the document. The architecture has to be easily extensible and maintainable in order to provide new functionalities. Every component must be conveniently thin and must encapsulate a single functionality (high cohesion). The dependency between components has to be unidirectional and coupling must be avoided in order to increase the reusability of the modules.

Futhermore, to increase cohesion and decoupling as much individual components must not include too many unrelated functionalities and reduce interdependencies.

1.3 Definitions, Acronyms, Abbreviation

RASD:	Requirements Analysis and Specification Document.
DD:	Design Document.
DBMS:	Relational Data Base Management System.
DB:	Database layer,
UI:	User Interface.
Backend:	Term used to identify the Application server.
Frontend:	The components which use the application server services (web frontend and the mobile applications).
SOA:	Service Oriented Architecture.
JDBC:	Java DataBase Connectivity.

JPA:	Java Persistence API.
EJB:	Enterprise JavaBean.
ACID:	Atomicity, Consistency, Integrity and Durability.

1.4 Reference Documents

This document refers to the following documents:

- Project rules of the Software Engineering 2 project
- Requirement Analysis and Specification Document (from the previous delivery)

1.5 Document Structure

This document is structured in five parts:

Chapter 1: Introduction. This section provides general information about the DD document and the system to be developed.

Chapter 2: Architectural Design. This section shows the main components of the systems with their subcomponents and their relationships, along with their static and dynamic design. This section will also focus on design choices, styles, patterns and paradigms.

Chapter 3: Algorithm Design. This section will present and discuss the main algorithms for the core functions of the system, independently from their concrete implementation.

Chapter 4: User Interface Design. This section shows how the user interface will look like and behave, by means of concept graphics and UX modeling.

Chapter 5: Requirements Traceability This section shows how the requirements in the RASD are satisfied by the design choices, and which compontents will implement them.

2 Architectural Design

2.1 Overview

This chapter provides a comprehensive view over the system components, both at a physical and at a logical level. This description will follow a top-down approach, starting with the description of the high-level components and their relations and interactions. We will then reason and describe the single components and the functionalities they must implement. We will especially focus on the components that implement the core logic of our application and using sequence diagrams we will describe the runtime behaviour of the system.

We will also include deployment diagrams to show the physical implementation of the system.

2.2 High Level Components

Before describing the actual system architecture we introduce the high level components of our application. Following the requirements and the specification listed on the RASD, we identify what components are needed in order to implement them and only after we have detected them, we will focus on the architecture and explain our architectural decision and the technlogies chosen. It's important to follow this process to generalize our design as much as possibile and abstract from implementation details, in this way we are able to describe only the essence of our system.

The main high level components of the system are the following:

Mobile Application: Power Enjoy is a car sharing service therefore it must be implemented with mobility in mind. Since the majority of the mobile devices have a GPS module and we need to have access to the user position for our application, it makes sense to require that the main user has our mobile application installed.

Application Server: This component contains all the logic for the system application. It will implement all the required functionalities and communicate both with the mobile application and the external services.

Web Server: This component does not contain any application logic, it's used to provide a web interface interface to the user. It helps to separate presentation from logic.

Web Browser: Using a web browser the user is able to communicate with the Web Server to obtain the required web pages.

Database: This components is responsible for data storage and retrieval which is crucial for our application. It will not implement any logic but it stores all the information needed for the correct functioning of our service. It must guarantee ACID properties and be accessible from the Application Server.

To give a complete overview of the system, we list also the external services which are not part of our system but with which the system depends to implement some functionalities (please refer to the RASD for a more detailed description).

High level components which are not part of our system:

Assistance Service: Power Enjoy is in charge of the management of the car-sharing system. All the secondary functionalities (recharging vehicles on-site, bringing cars back from unsafe areas and fixing malfunctions) are handled by an assistance service. This components provides an API to handle all the assistance request.

Payment Service: Every Power Enjoy user, in order to use our service, is required to have an account registered to a third party payment service, with a valid payment method. This components provides an API to handle all the payments functionalities.

Car On Board System: Every Power Enjoy vehicle comes with a pre-installed on board system which registers and notifies all the car activity. This components provides an API to monitor and remotly control every vehicle.

In the figure below we rapresent all the components listed above in a layered fashion, highlighting the relations among the different parts:

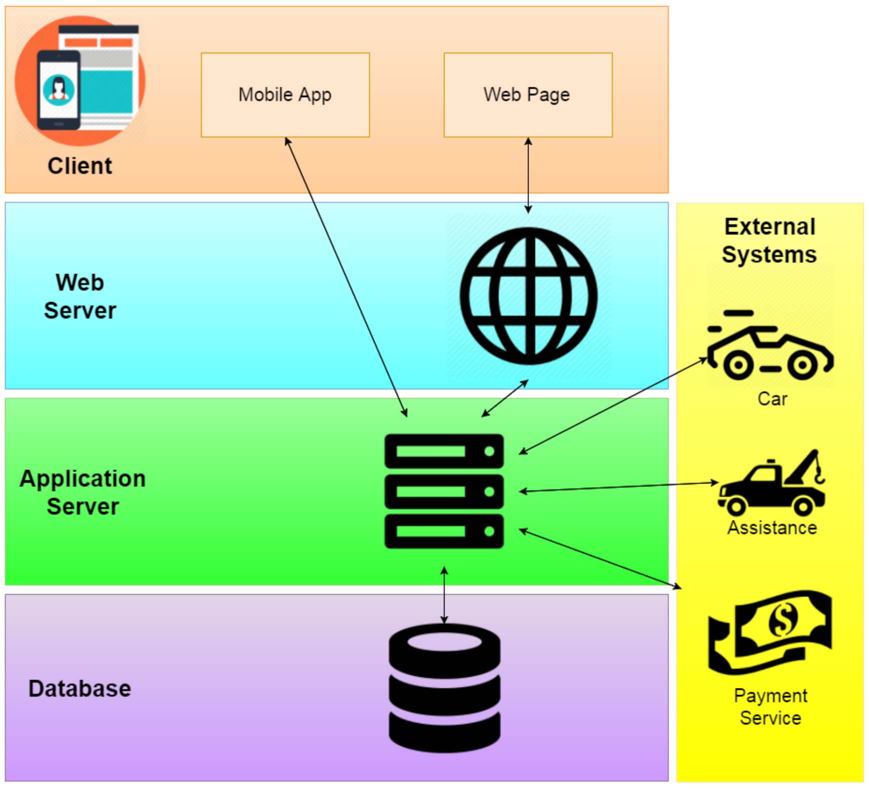


Figure 1: High Level Components

From this diagrams it's important to point out:

- This layered rapresentation suggests a four tier architecture
- Separating Application Server and the Web Server improves scalability. We expect Power Enjoy usage to grow in differents cities and in this way we're able to seperate the tasks and optimize each layer individually to support increasing loads.
- The Application Server is the bottleneck of our system. Every other components is in relation with it, therefore it's performance is stricly related to the performance of our system. But since every components expects different functionalities from the Application Server we can parallelize using threads and split the work load among different modules.

2.3 Component View

In this subsection we will look inside every single component and describe all the internal subcomponents. It's important to identify the relevant modules without increasing granularity to much. This will allow to have an efficient load balance in the present and it will be easy to integrate new functionalities in the future.

In a divide-and-conquer fashion for every component we will specify the implementation chosen and at the end we will how to connect the single components.

2.3.1 Application Server

This components implements the logic of the Power Enjoy Application, it's the core of our business and in this part of the document we'll explore the subcomponents inside. To provide a natural continuation from the RASD, we will start from the Class Diagram, and we will focus on the control object. We will look at the singles control functions, logically group them in cohesive groups and map them in modules of our system.

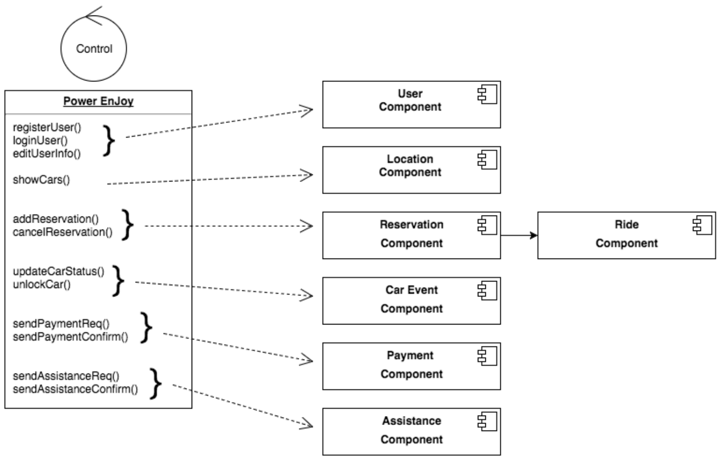


Figure 2: Mapping of control function to subcomponents of our system

A brief desription of the functionalities expected from each module:

User Component This module provides the logic for a User and a Power User regarding all the user management features, namely: user login, user registration, user deletion, user profile editing. It will perform all the validation of the credentials received before inserting them into the system.

Location Component This module handles the position of each car and user of the system. It's used to show cars on the map and to perform proximity checks to unlock vehicles.

Reservation Component This module is responsible to manage all the current reservation and to accept new reservations from Power Users. It will be responsible for timing features (expiration of the reservation) and for avoiding undesiderable behaviours (reservation of an unavailable car, double-booking and multiple reservations). It's strongly connected to the Car Event Component.

Ride Component This module creates and manages new rides from active reservations. It will manages all the relevant informations (unsafe park timing, battery level, sharedRideLenght) that receives from the Car Event Component. When a Ride ends it will communicate it to the Reservation Component.

Car Event Component This modules interfaces with the API of the car on board system. It's a "low level" component that collects all the car data for other components to use and can be used to remotly control the vehicle. It will signal to the interested component all the events of the car (car locking/unlocking, motor ignition, malfunctioning).

Payment Component This modules interfaces with the API of the Payment Service to request payments and receive confirmations. It will not perform the fee calculation but it will receive the final price and check for price variations via other components It will also flag/unflag users as banned in case of Pending Payment.

Assistance Component This modules interfaces with the API of the Assistance Service to request assistance (recharge on site, fix malfunctions, bring car back from unsafe to safe areas) and receive confirmations once the assistance is provided. Once a car is fixed it will update the car informations (e.g. new position, new battery level).

Implementation Choice

This component will be implemented using Jave Enterprise Edition 7 (JEE7) using Enterprise JavaBeans (EJB) to implement the modules described above. The platform incorporates a design based largely on modular components running on an application server (GlassFish Server in this case) which is a natural consequence from the description above. It also provides support for large-scale, multi-tiered, scalable, reliable, and secure network applications. This modularity helps to handle such complex system and it make easy to insert the functionalities (as new beans) in the future.

The next figure will show the Application Component implemented as session beans logically grouped in EJBContainers.

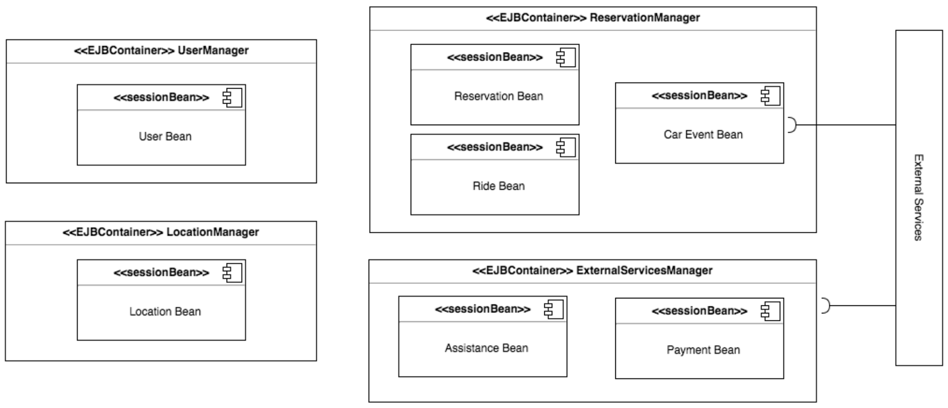


Figure 3: Application as Java Entity Beans

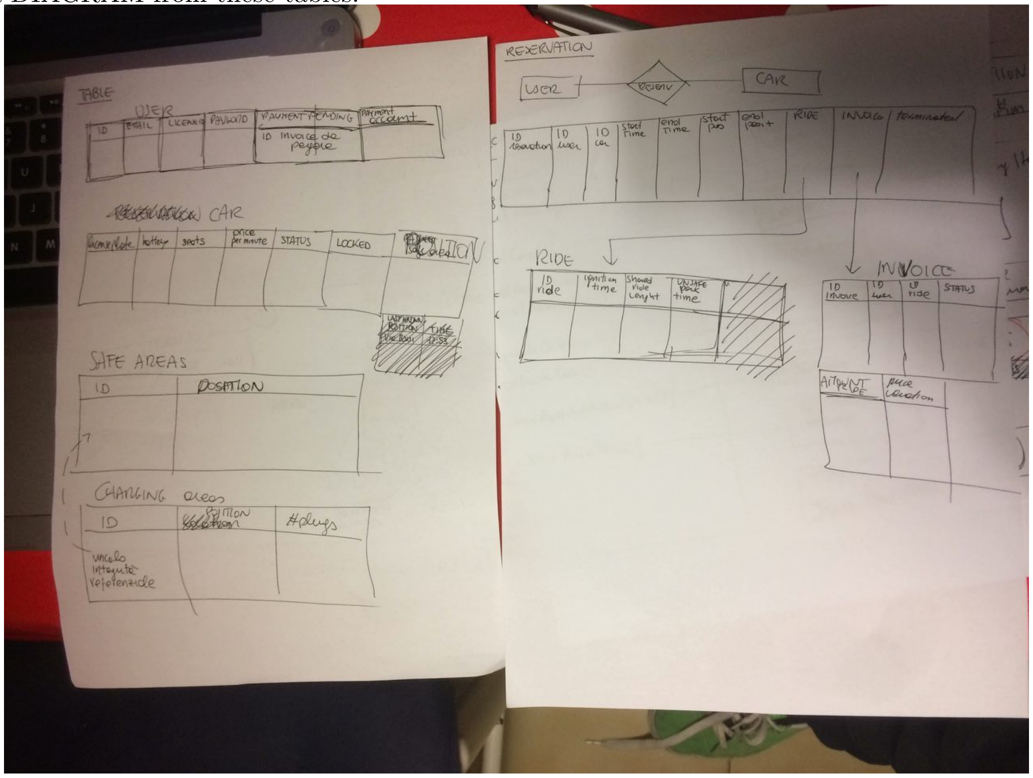
For the sake of clarity some information have been hidden from the diagram below but can be found in the next sections.

In Section 2.5 we will provide more details on the interfaces of each module.

In Section 2.6 using UML Sequence Diagrams we highlight the relationship between each component during a runtime analysis of our system.

2.3.2 Database

ER DIAGRAM from these tables:



Discorsi Vari e poi spiegazione della scelta di MySQL con Java Persistence API e diagramma con Entity Beans (Parte alta del class diagram)

- JDBC/JPA JDBC (Java Database Connectivity) was chosen as connector between Database and the application server. Java database con- nectivity interface (JDBC), incfact, is a software component that allows Java applications to interact with databases. To enhance the connection, JDBC re- quires drivers for each database. These drivers connect to the database and implement the protocol to transfer query and respective results between the client and database. JDBC was chosen because it is available for any DBMS also thanks to the ODBC bridge. ODBC infact allows programmers to make SQL requests that will access data from DB distinct without having to know the proprietary interfaces of each DB. In this way it is easier to change the database without altering the application layer. The Java Persistence API (JPA) is a Java application programming interface specification that describes the management of relational data in applications using Java Platform, Standard Edition and Java Platform, Enterprise Edition.

- The access to the DBMS is not implemented with direct SQL queries:

instead, it is completely wrapped by the Java Persistence API (JPA). The object-relation mapping is done by entity beans. The Entity Beans representing the database entities (Figure 2.5) are strictly related to the entities of the ER diagram (Figure 2.4).

2.3.3 External Services

Qualche parola sugli external services

2.3.4 Client

- Spiegazione del perche' mobile-first.
- Spiegazione del perche' aggiungere un webserver
- Diagramma della mobile app (con gps ecc...)
- JAX-RS to implement proper RESTful APIs to interface with clients and the Web Server;
- • To interface with external systems, existing RESTful APIs defined by the partner (payment handlers, maintenance system) will be used.

2.4 Deployment View

In this subsection we'll move on the physical side of our application, describing his deployment with the support of UML diagrams.

2.5 Component Interfaces

Interfacce dei beans della sezione precedente

2.6 Runtime View

Sequence diagrams to descrive the way componenets interact to accomplish specific tasks typically related to your use cases.

2.7 Selected Architectural Styles and Pattern

Explain styles/patterns used and why/how

3 Algorithm Design

Focus of the most relevant algorithmic part

4 UI Design

Overview of how the user interace of your system will look like
Solo mockups niente UX Diagram

5 Requirements Traceability

Explain how requirements defined in the RASD map to the design elements that you have definited in this document.

6 Effort Spent

Niccolo' 10 Ore