



POLITECNICO
MILANO 1863

Politecnico di Milano

COMPUTER SCIENCE AND ENGINEERING

SOFTWARE ENGINEERING 2

Design document

PowerEnJoy

Authors:

Francesco Fabiani
Jagadesh Manivannan
Niccolò Pozzolini

Professors:

Elisabetta Di Nitto
Luca Mottola

Contents

1	Introduction	2
1.1	Purpose	2
1.2	Scope	2
1.3	Definitions, acronyms, abbreviations	3
1.4	Reference documents	4
1.5	Document structure	4
2	Architectural design	5
2.1	Overview	5
2.2	Component view	5
2.3	Deployment view	5
2.4	Runtime view	6
2.5	Component interfaces	9
2.6	Selected architectural styles and patterns	9
2.7	Other design decisions	9
3	Requirements traceability	10
4	Hours of work	13
4.1	Francesco Fabiani	13
4.2	Jagadesh Manivannan	13
4.3	Niccolò Pozzolini	14

Chapter 1

Introduction

1.1 Purpose

The Design Document aims to describe more accurately all the functionalities provided by PowerEnJoy focusing on the software design of the system, such as the architectural components and the interfaces that allow them to interact.

The main objective of this DD is to illustrate to a team of developers how to scrupulously implement our service, with the help of UML diagrams explaining the purpose of every element.

1.2 Scope

The product described is PowerEnJoy, a car-sharing service which offers to its users exclusively electric cars. It includes the common functionalities of its category: permitting to registered users to obtain the position of all the available cars, reserving one within a certain amount of time and continuously displaying the up-to-the-minute cost of the ride are just few of them. Moreover, PowerEnJoy stimulates users to behave virtuously towards the ecosystem by applying various types of discounts under specific conditions.

There are four software components that constitute the PowerEnJoy project. First, a back-end server provides APIs in order to simplify the communication related to the interactions of a user with the cars. Then two applications are available for a user to allow him/her the employment of every functionality: a web-based application (intended for visualization from desktop) and a mobile one. Lastly, every vehicle will be equipped with an on-board computer, used by the driver to manage the ride with the available options and see real-time information related to it, such as the time spent,

the distance traveled and the total amount.

1.3 Definitions, acronyms, abbreviations

- *API*: Application Programming Interface
- *Car*: electric vehicle provided by the service
- *DB*: Database
- *DBMS*: Database Management System
- *DD*: Design Document
- *ER*: Entity-Relationship
- *GPS*: Global Positioning System
- *Guest* or *Guest user*: person not registered to the service
- *MVC*: Model View Controller
- *OS*: Operating System, related both to desktop and mobile platforms
- *PIN*: Personal Identification Number
- *RASD*: Requirements Analysis and Specification Document
- *Registered user*: see *User*
- *REST*: REpresentational State Transfer
- *RESTful*: that follows the REST principles
- *Safe area*: set of parking spots where a user can leave a car without penalization
- *User*: person with a valid driving license registered to the service
- *W3C*: World Wide Web Consortium

1.4 Reference documents

The Design Document has been composed following the indications and examples reported in the document ISO/IEC/IEEE 1016-2009, released by W3C, containing provisions for the processes and products related to the engineering of requirements for systems and software products and services throughout the life cycle.

The previous document delivered for this project, the Requirements Analysis and Specification Document, describes fundamental aspects of PowerEnjoy such as domain assumptions, goals, functional and non-functional requirements.

With regards to the course named Software Engineering 2 and held by professors Luca Mottola and Elisabetta Di Nitto (Politecnico di Milano, a. y. 2016/17), the document conforms to the guidelines provided during the lectures and within the material of the course.

1.5 Document structure

This Design Document is composed of five chapters, each of them describing a specific aspect of our project.

- *Introduction*: provides a general overview of this document.
- *Architectural design*: explains the main components, focusing on their interactions, architectural choices and patterns.
- *Algorithm design*: describes the most relevant algorithms utilized in order to optimize the system.
- *User interface design*: shows what is likely to be displayed to the user when using the client-side implementations.
- *Requirements traceability*: indicates how the requirements reported in the RASD map to the design elements described in this document.

Chapter 2

Architectural design

2.1 Overview

2.2 Component view

2.3 Deployment view

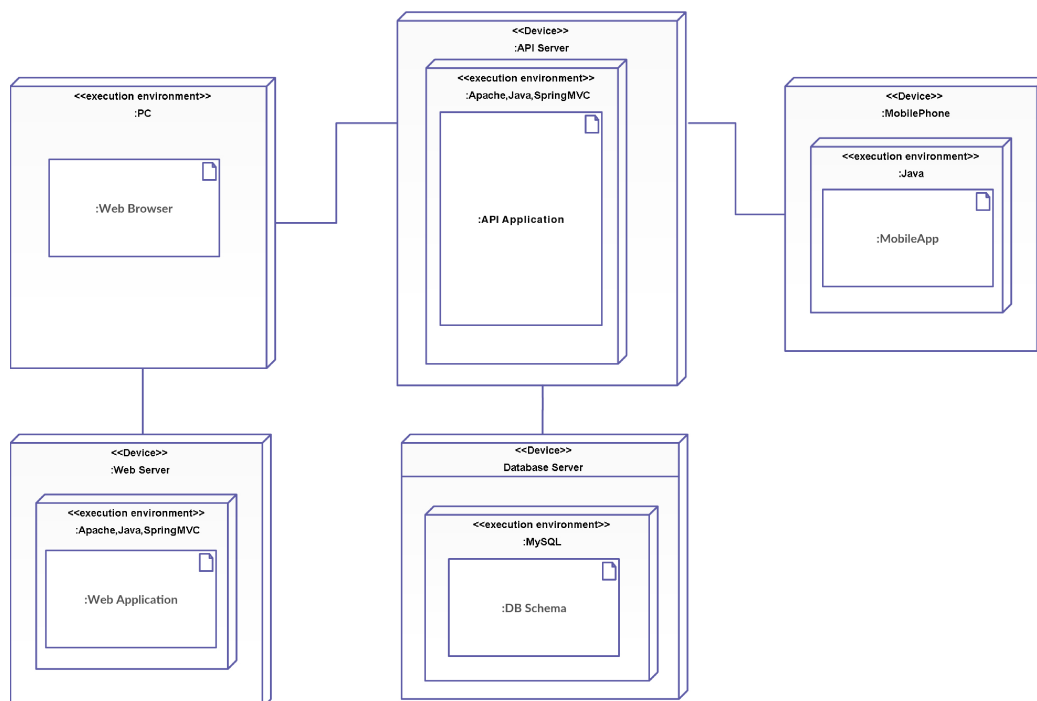


Figure 2.1: Deployment view diagram of the system

2.4 Runtime view

This section describes the system components involved and the related interactions for some of the use cases reported in the RASD.

The map manager is an interface that acts as a router to communicate between the Client/User and with the system. In the run time view sequence diagram, the map manager is not shown mostly in all the diagrams. But in the individual explanation of the sequence diagram, its functionality is explained.

Each and every component are separated for specific functionality. This is done in order to achieve better performance and re-usability of the code.

Selection of available car

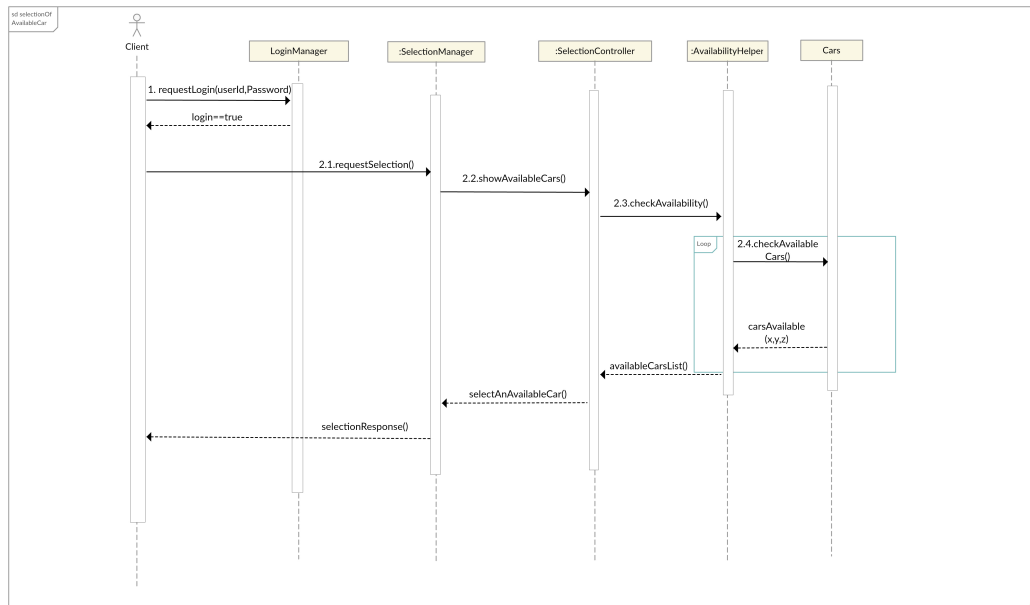


Figure 2.2: Sequence diagram for the selection of the available car

The Client/User wants to select an available car. Only after successful login, the Client/User will be able to select the car. A Login manager takes care of this credential verification. After successful verification of the user credentials, the Client/User can send a request to select an available Car through the map manager which transfers the request to the selection manager. This indeed interacts with the selection controller to display the list of available cars.

The controller interacts with another component called availability helper. This availability helper communicates with all the cars and checks which cars are available. This check happens in a loop continuously. After finding the list of available cars, then the response is transferred selection controller and finally it is transferred to the Client/User. After this, the user/client can select only one available car for the reservation.

Reservation of selected car

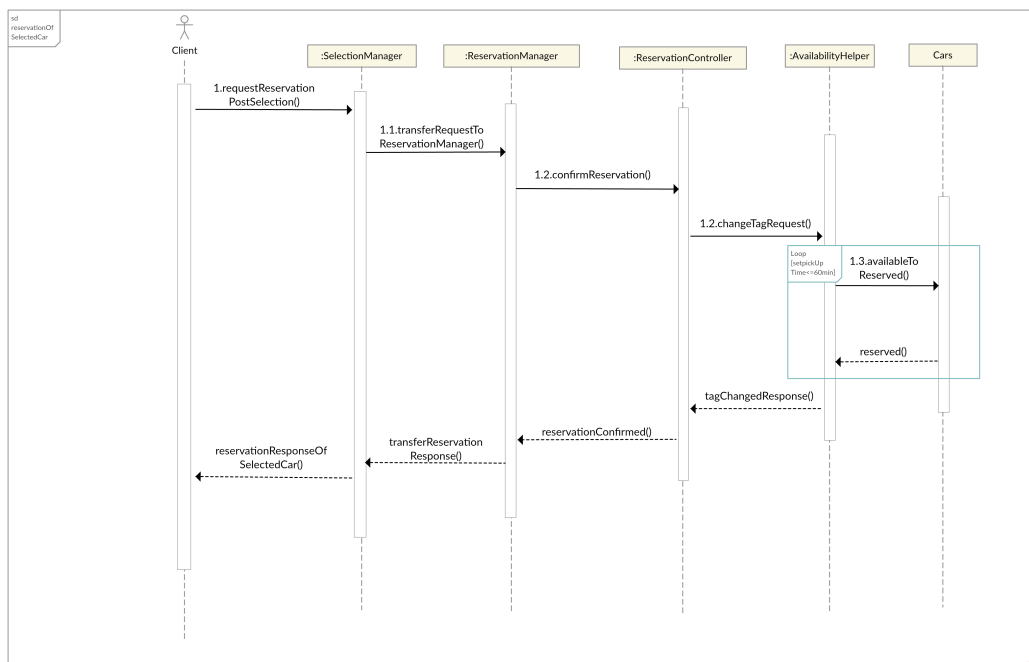


Figure 2.3: Sequence diagram for the reservation of the selected car

The reservation confirmation of a car can happen only after selection of the available car. So the sequence of the user's request for the reservation starts from the selection manager. It transfers the request to the reservation manager for the confirmation. The reservation controller is a component that sends another request to the availability helper to change the tag of the car selected.

The availability helper interacts with the car and changes the tag as "reserved". The availability helper starts a counter timer immediately after changing the tag by setting a time for 60 minutes. This tag will hold reserved only for upto 60 minutes. Now, the response is sent back to the reservation controller which is finally sent to the Client/User as a confirmed reservation.

Selection of special parking areas

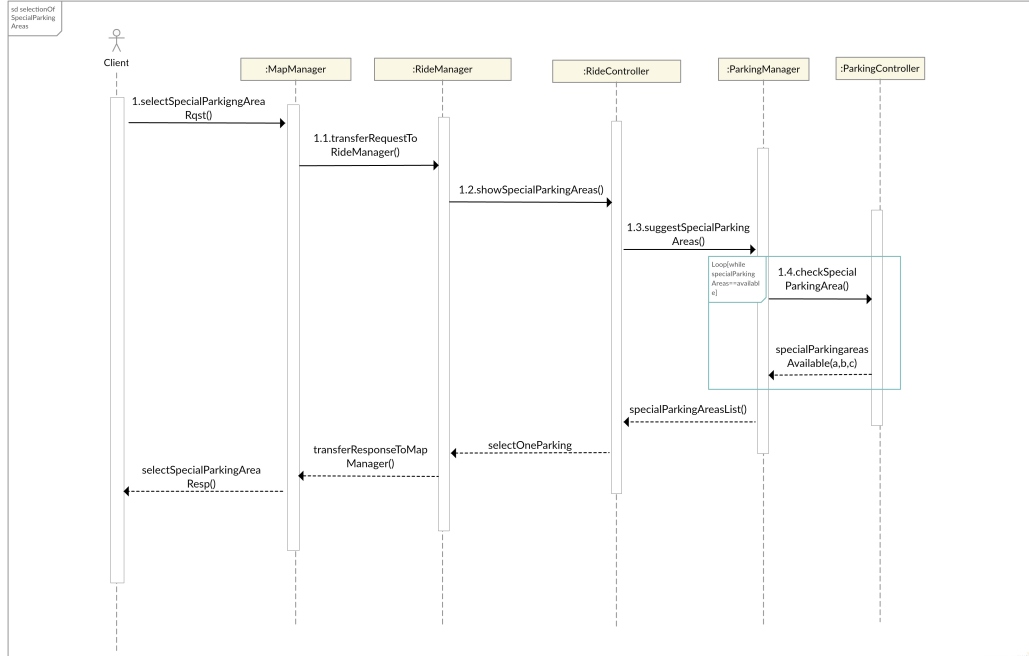


Figure 2.4: Sequence diagram for the selection of special parking areas

The Client/User requests for a special parking area to get a discount. So, he/she selects the option during the ride to the system through the map manager. The map manager acts as an interface and transfers the request to the ride manager. The ride manager provides an interface to communicate with the ride controller during the ride.

This component asks to display the available list of special parking areas nearby. So, the parking manager accepts this request while the parking controller checks the availability of special parking areas and sends the list of available special parking areas to the Parking manager. The parking controller does a check continuously in a loop. If there are no special parking areas available, then it sends an response with result as zero and suggests list of safe parking areas to the parking manager. This requests are finally sent as a response to the user/client.

Enable money saving option for discount

The Client/User wants to select/enable money saving option for availing a discount. In order to do that, he/she must select the final destination through the map manager. The map manager asks to select the final destination

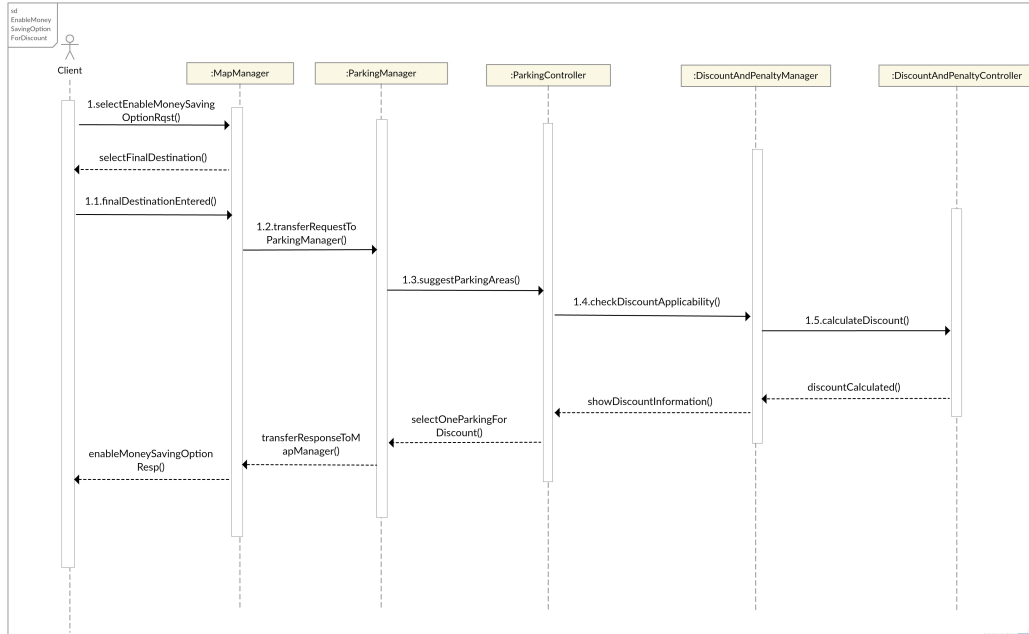


Figure 2.5: Sequence diagram for the activation of the money saving option

immediately after selecting the money saving option. After entering the final destination, the request is transferred to the Parking controller through the parking manager to select the parking areas that have a discount.

To check to discount applicability, the parking controller communicates with the discount and penalty manager. The discount controller checks if the parking area is applicable for a discount and calculates the discount. This discount information along with the parking area is sent as a response to the Client/User through the map manager.

2.5 Component interfaces

2.6 Selected architectural styles and patterns

2.7 Other design decisions

Chapter 3

Requirements traceability

Each requirement described in the RASD maps to some of the components reported in this document; below the reader will find the list of components involved for every goal of our service.

- [G1] Registration of a user to the system
 - Login Manager
- [G2] Finding the locations of the available cars
 - Map Manager
 - Availability Helper
 - Cars
- [G3] Reservation of a car
 - Map Manager
 - Selection Manager
 - Reservation Manager
 - Reservation Controller
 - Availability Helper
 - Cars
- [G4] Expiration of reservation and penalization
 - Map Manager
 - Selection Manager
 - Reservation Manager

- Reservation Controller
 - Availability Helper
 - Discount and Penalty Manager
 - Discount and Penalty Controller
- [G5] Entry of registered user into the car
 - Map Manager
 - Ride Manager
 - Ride Controller
 - Car
- [G6] Start charging and notifying the registered user
 - Map Manager
 - Ride Manager
 - Ride Controller
 - Discount and Penalty Manager
 - Discount and Penalty Controller
 - SMS Gateway Interface
 - SMS Gateway
- [G7] Stop charging the registered user and lock the
 - Map Manager
 - Ride Manager
 - Ride Controller
 - Parking Manager
 - Parking Controller
 - Discount and Penalty Manager.
 - Discount and Penalty Controller
- [G8] Safe areas for parking the reserved cars
 - Map Manager
 - Ride Manager
 - Ride Controller

- Parking Manager
 - Parking Controller
- [G9] Detection of extra passengers and applying discount
 - Map Manager
 - Ride Manager
 - Ride Controller
 - Discount and Penalty Manager
 - Discount and Penalty Controller
- [G10] Detection of the battery status and applying discount
 - Map Manager
 - Ride Manager
 - Ride Controller
 - Discount and Penalty Manager
 - Discount and Penalty Controller
- [G11] Detection of special parking areas and applying discount
 - Map Manager
 - Ride Manager
 - Ride Controller
 - Parking Manager
 - Parking Controller
 - Discount and Penalty Manager
 - Discount and Penalty Controller
- [G12] Checking parking and battery constraints and penalization
 - Map Manager
 - Ride Manager
 - Ride Controller
 - Parking Manager
 - Parking Controller
 - Discount and Penalty Manager
 - Discount and Penalty Controller