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

Academic Year 2016/2017

Contents

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.

1.3 Definitions, acronyms, abbreviations

- *API*: Application Programming Interface
- *BCE*: Business Controller Entity
- *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
- *UX*: User eXperience
- *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

The system-to-be that will be implemented for PowerEnJoy is made of three main layers: client, application server and database.

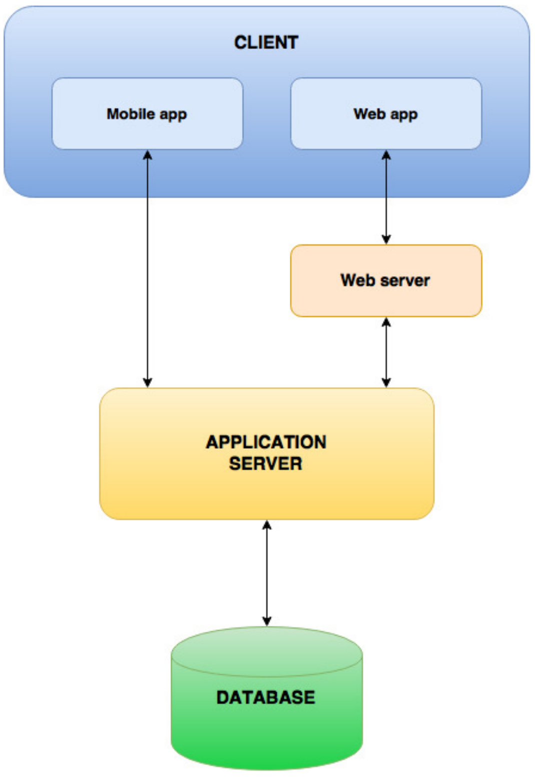

Figure 2.1: Layers of the system

- *Client.* It is the presentation layer and provides two different interfaces, a mobile app and desktop version to be used via web.
  The former is intended for phone and tablets and straightly interacts with the application server layer.
  The latter is a web app, intended then for browsers, which interacts with the web server, a middleware in charge of providing the model of the layout and handling the events coming from the web app.

- *Application server.* It is the logic layer of the system. It includes all the business logic and provides web services so that users can interact with this layer from both the interfaces.

- *Database.* It is the data layer of PowerEnJoy. It interacts only with the application server, receiving from it requests to retrieve, update, create or delete the data stored. It provides the interfaces that allow the application layer to execute these operations.

2.2 Component view

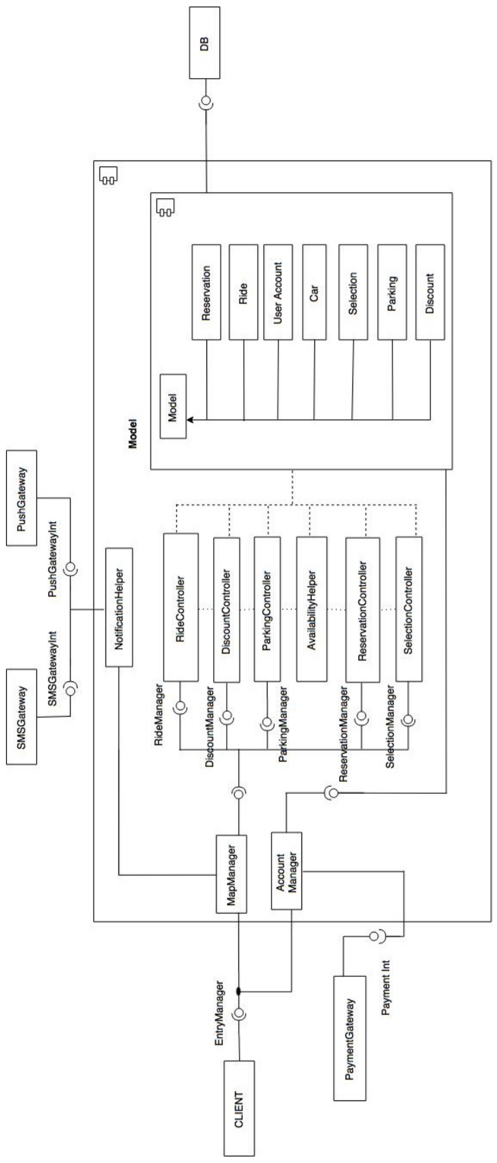

Figure 2.2: Component view of PowerEnJoy

2.2.1 State diagrams

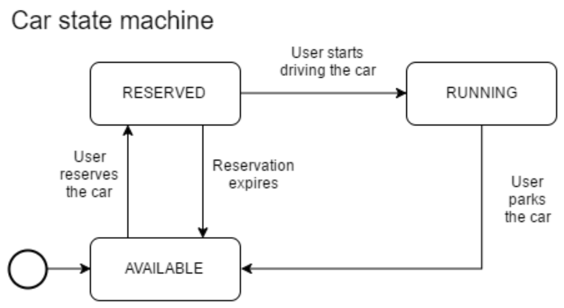

Figure 2.3: State diagram for the class car

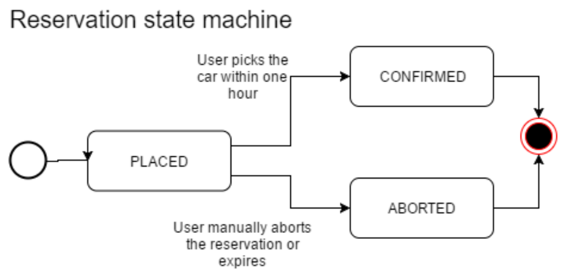
Reservation state machine
User picks the car within one hour
CONFIRMED
PLACED
ABORTED
User manually aborts the reservation or expires

Figure 2.4: State diagram for the class reservation

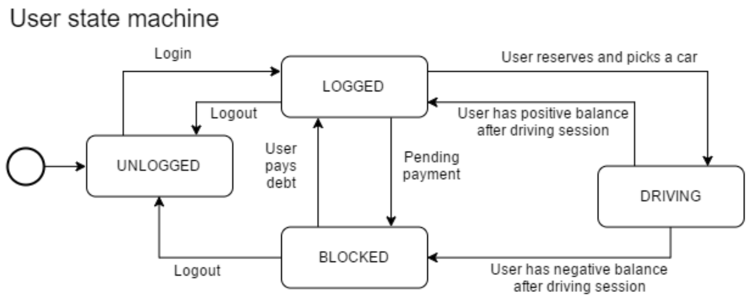
User state machine
Login
LOGGED
User reserves and picks a car
Logout
User has positive balance after driving session
UNLOGGED
User pays debt
Pending payment
DRIVING
BLOCKED
Logout
User has negative balance after driving session

Figure 2.5: State diagram for the class user

2.3 Deployment view

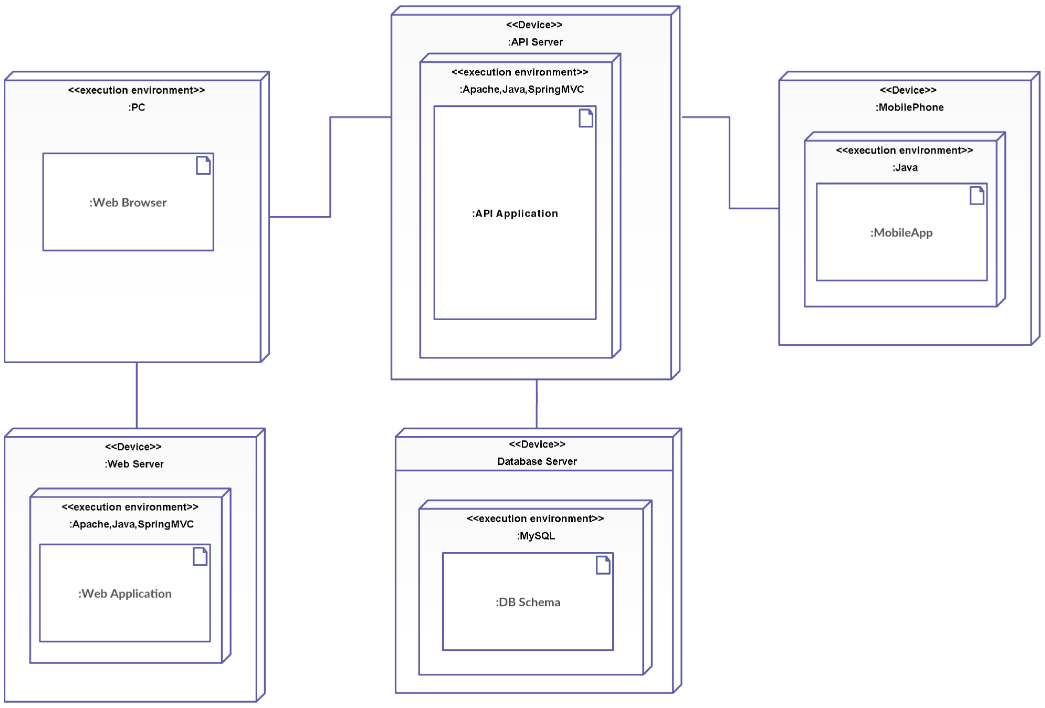

Figure 2.6: Deployment view diagram of the system

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

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

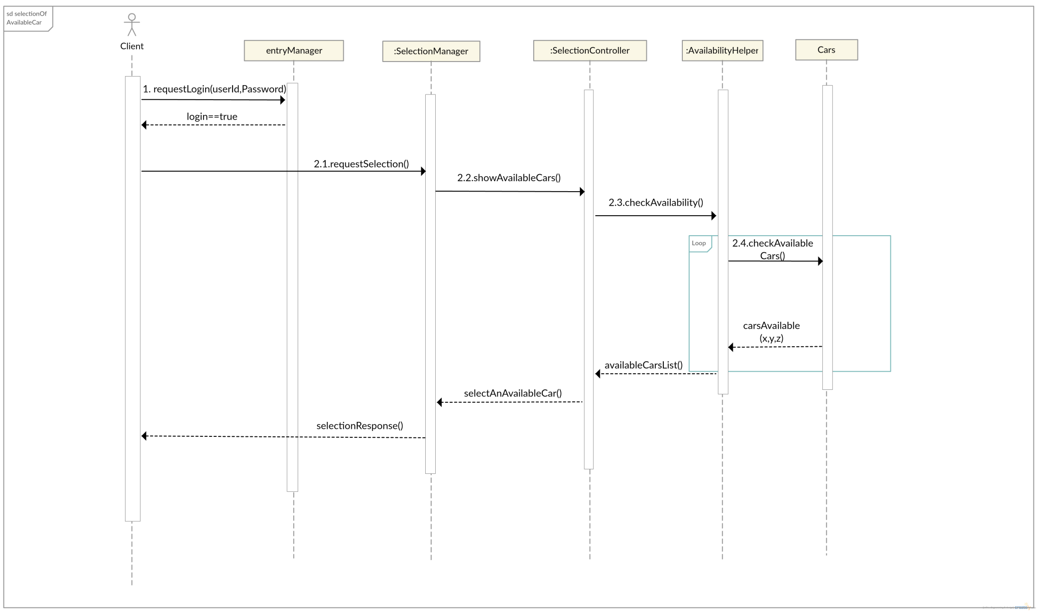

Figure 2.7: Sequence diagram for the selection of the available car

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

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

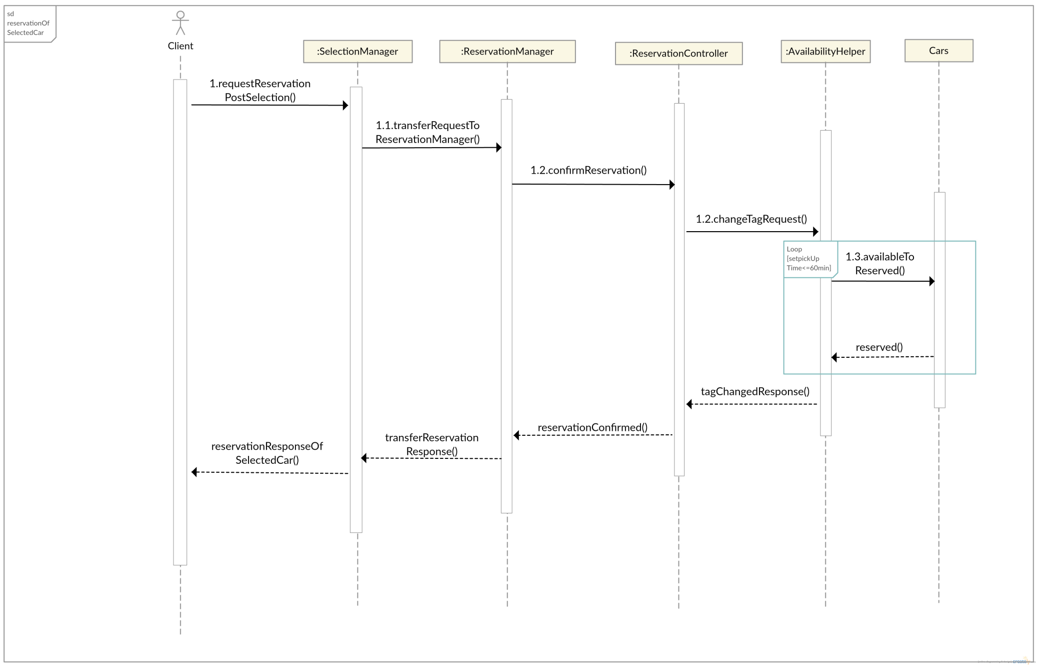
sd reservationOf SelectedCar
Client
:SelectionManager
:ReservationManager
:ReservationController
:AvailabilityHelper
Cars
1.requestReservation PostSelection()
1.1.transferRequestTo ReservationManager()
1.2.confirmReservation()
1.2.changeTagRequest()
Loop [setpickUp Time<=60min]
1.3.availableTo Reserved()
reserved()
tagChangedResponse()
reservationConfirmed()
transferReservation Response()
reservationResponseOf SelectedCar()

Figure 2.8: Sequence diagram for the reservation of the selected car

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

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.

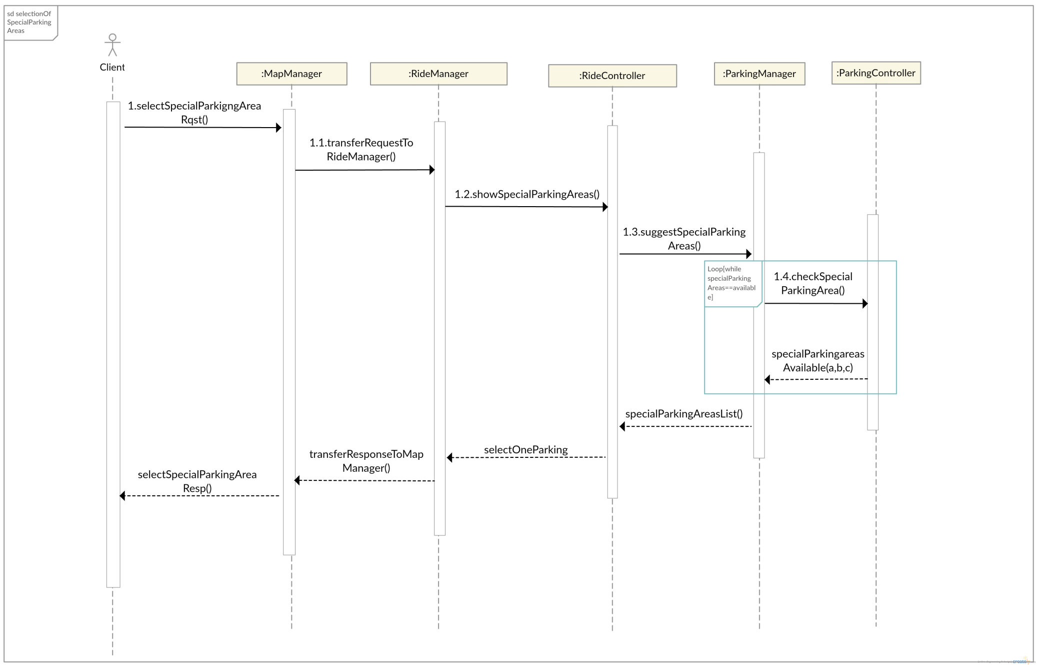
sd selectionOf SpecialParking Areas
Client
:MapManager
:RideManager
:RideController
:ParkingManager
:ParkingController
1.selectSpecialParkigngArea Rqst()
1.1.transferRequestTo RideManager()
1.2.showSpecialParkingAreas()
1.3.suggestSpecialParking Areas()
Loop[while specialParking Areas==available]
1.4.checkSpecial ParkingArea()
specialParkingareas Available(a,b,c)
specialParkingAreasList()
selectOneParking
transferResponseToMap Manager()
selectSpecialParkingArea Resp()

Figure 2.9: Sequence diagram for the selection of special parking areas

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 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 Selected architectural styles and patterns

As described before, our application will be divided into three main layers:

- Database (for the storing of permanent data)
- Application Server (for the business logic)

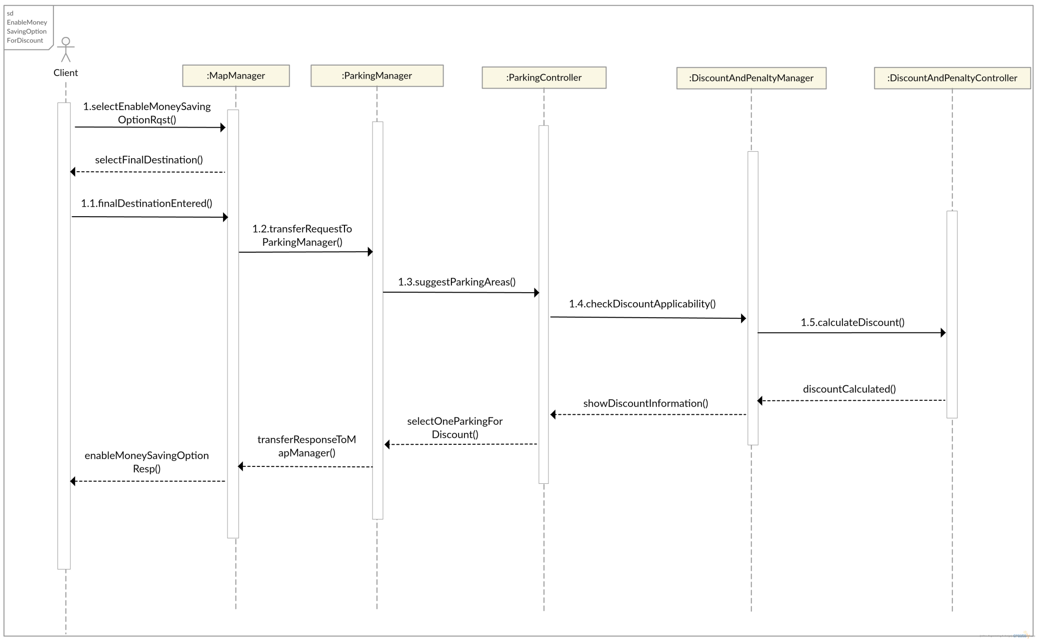
sd EnableMoney SavingOption ForDiscount
Client
:MapManager
:ParkingManager
:ParkingController
:DiscountAndPenaltyManager
:DiscountAndPenaltyController
1.selectEnableMoneySaving OptionRqst()
selectFinalDestination()
1.1.finalDestinationEntered()
1.2.transferRequestTo ParkingManager()
1.3.suggestParkingAreas()
1.4.checkDiscountApplicability()
1.5.calculateDiscount()
discountCalculated()
showDiscountInformation()
selectOneParkingFor Discount()
transferResponseToM apManager()
enableMoneySavingOption Resp()

Figure 2.10: Sequence diagram for the activation of the money saving option

- Client (for the interaction between user and system)

RESTful API with JSON used by clients (both mobile apps and web browsers) to interact with the application layer. Users have to authenticate through specific API calls via HTTP basic authentication for each request.

As regards the patterns for designing PowerEnJoy, the Model-View-Controller one has been widely taken into account. MVC is a software design pattern for implementing user interfaces on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways this is presented to the user.

Chapter 3

Algorithm design

In this chapter we are going to present two of the algorithms thought for our system. The language used to describe them is based on Java, with the sporadic addition of pseudocode and comments for the least interesting parts of the code.

3.1 Discounts and penalties

```
Double discountsAndPenalties(Double rideCost) {
    Double discount = 0.0, penalty = 0.0;

    if (totalPassengers >= 3) {
        discount = 0.1; // 10% discount
    }

    if (batteryStatus >= 0.5) {
        discount = 0.2; // 20% discount
    }

    if (car.isCharging()) {
        discount = 0.3; // 30% discount
    }

    rideCost -= rideCost * discount;

    // distances are expressed in meters
    Double currentDistance, minDistance = Double.MAX_VALUE;
    for (PowerStation ps : PowerStation.listAll()) {
        currentDistance = distance(car.getPosition(),
         ↪  ps.getPosition());
        if (currentDistance < minDistance) {
            minDistance = currentDistance;
        }
    }

    if (minDistance > 3000 || batteryStatus < 0.2) {
        penalty = 0.3; // 30% penalty
    }

    rideCost += rideCost * penalty;

    return rideCost;
}
```

3.2 Money Saving Option

```
PowerStation moneySavingOption(Position destination) {
    long[] density;
    int[] distances;
    PowerStation[] stations;
    int distance;

    for (i = 0; i < zone.size; i++) {
        density(i) = zone[i].parkedCar.size / zone[i].area;
    }

    for (i = 0; i < zone.size; i++) {
        distance = distance(zone[i], destination);

        if (avg(density) - density(i) > threshold1 && distance
         ↪  < threshold2) {
            distances.add(distance);
        }
    }

    return stations[indexOf(min(distances))];
}
```

Chapter 4

User interface design

4.1 Mock-ups

We have already presented the mock-ups for mobile and desktop versions of the app in section 2.2.1 of the Requirements analysis and specification document.

4.2 UX diagrams

In this section UX diagrams for some of the relevant operations will be showed in order to better describe the user interaction with the system.

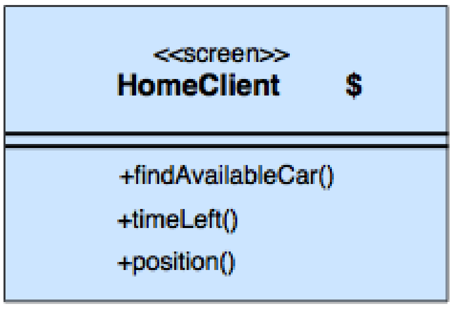

Figure 4.1: UX user with active reservation

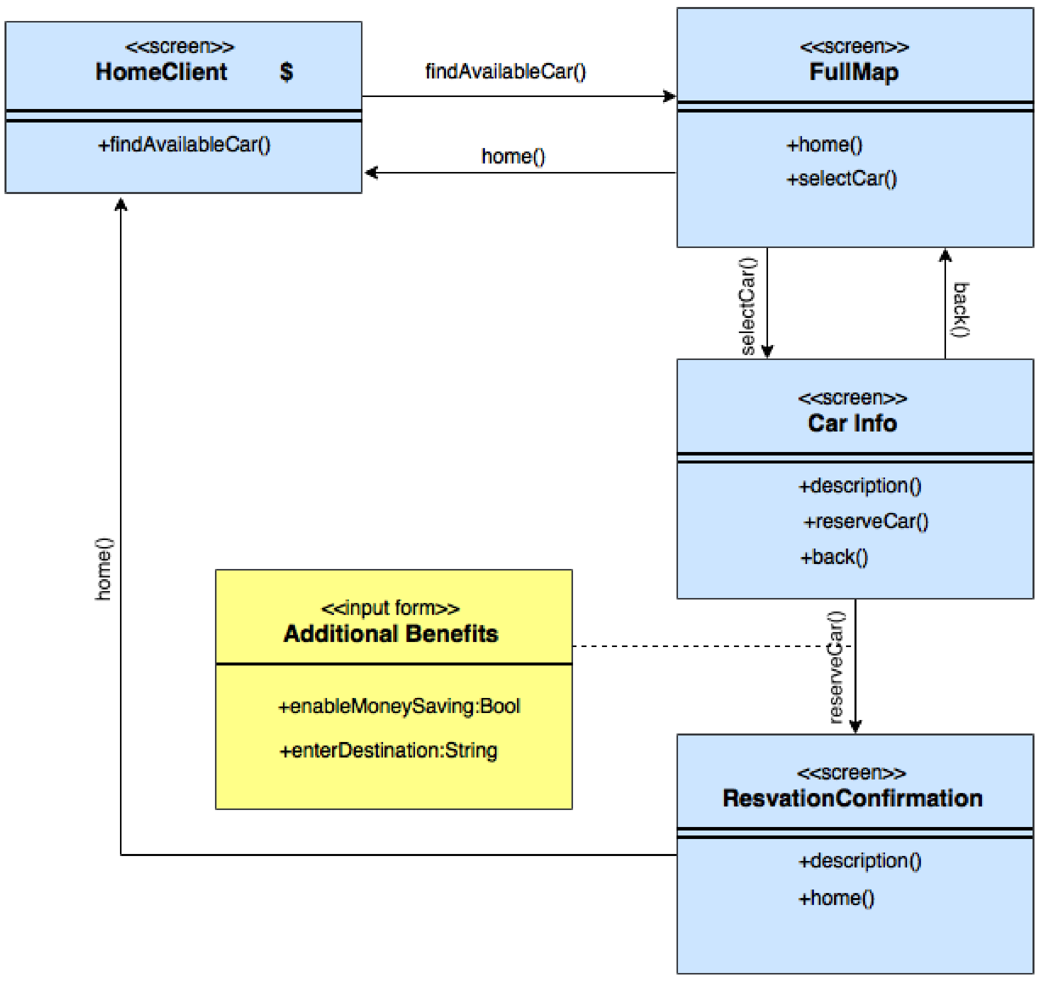

Figure 4.2: UX user with no reservation

4.3 BCE diagram

The figure 4.3 shows the Business-Controller-Entity diagram for users that access to the homepage not having an active reservation of a car. A BCE diagram is very useful when the MVC pattern is used as arch

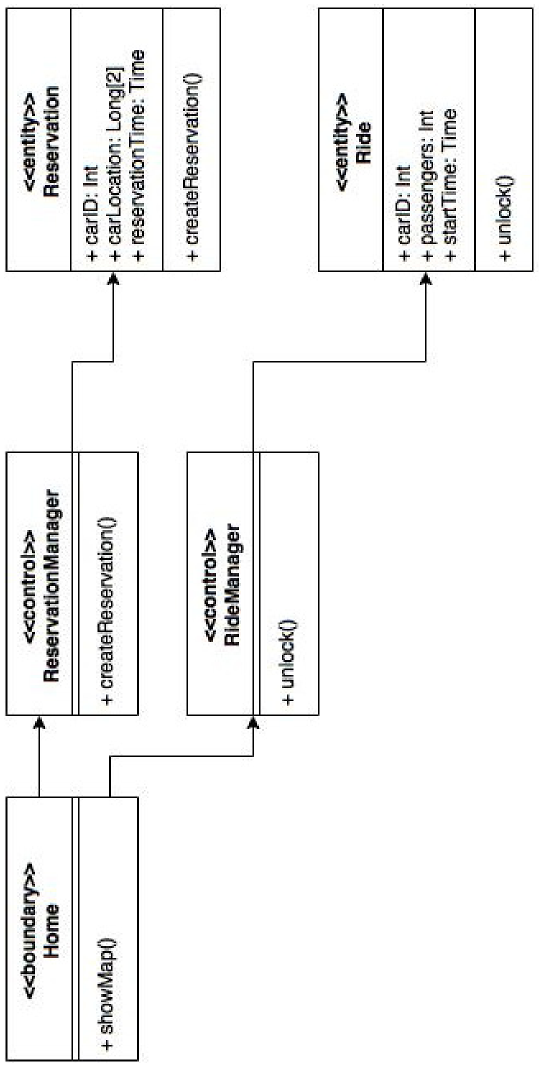

Figure 4.3: BCE user with no reservation

Chapter 5

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

Chapter 6

Effort spent

6.1 Francesco Fabiani

- 21/11/2016: 1h
- 23/11/2016: 2h
- 26/11/2016: 1h
- 28/11/2016: 1h30min
- 30/11/2016: 2h30min
- 02/12/2016: 2h
- 04/12/2016: 1h30min
- 05/12/2016: 2h30min
- 06/12/2016: 1h
- 07/12/2016: 2h
- 08/12/2016: 2h30min
- 10/12/2016: 2h
- 11/12/2016: 1h30min
- 12/12/2016: 3h

6.2 Jagadesh Manivannan

- 20/11/2016: 1h30min
- 21/11/2016: 1h
- 22/11/2016: 1h30min
- 24/11/2016: 1h
- 25/11/2016: 2h
- 27/11/2016: 2h
- 29/11/2016: 1h30min
- 01/12/2016: 3h
- 03/12/2016: 1h30min
- 04/12/2016: 2h
- 06/12/2016: 2h
- 07/12/2016: 1h30min
- 09/12/2016: 2h
- 10/12/2016: 3h
- 11/12/2016: 2h

6.3 Niccolò Pozzolini

- 21/11/2016: 1h
- 22/11/2016: 1h30min
- 24/11/2016: 2h
- 26/11/2016: 1h30min
- 27/11/2016: 2h
- 29/11/2016: 2h
- 30/11/2016: 2h30min

- 02/12/2016: 1h
- 03/12/2016: 1h
- 04/12/2016: 2h30min
- 06/12/2016: 1h
- 08/12/2016: 1h
- 09/12/2016: 1h30min
- 10/12/2016: 2h
- 11/12/2016: 2h
- 12/12/2016: 1h30min

Chapter 7

References

7.1 Used tools

The tools we used to create this DD document are:

- *Creately.com*: for design of sequence diagram of Runtime view,Deployment View.
- *Draw.io*: version control for the development of this document.
- *GitHub*: for design of the UML diagrams-component view, UX diagram.
- *TeXstudio*: drafting of this document.

Chapter 8
Changelog

8.1 v1.1

- Updated the component view
- Updated the runtime view of selection of available car