POLITECNICO
MILANO 1863

Politecnico di Milano

Software Engineering 2 project

PowerEnJoy

Design Document

Matteo Penco
mat. 875740

Riccardo Pressiani
mat. 874948

Version 1.0 approved

December 11, 2016

Revision History

Revision	Date	Author(s)	Description
0.1	23.11.16	RP	Document created
1.0	11.12.16	RP and MP	Document approved

Hours of work

Matteo Penco	20h
Riccardo Pressiani	20h

Contents

Chapter 1

Introduction

1.1 Purpose

This document provides a high detailed description about the design and the architecture of the PowerEnJoy software. It includes the design information about the components and how they intereact among them, the design of the most complex algorithm and the user interface design of the software to be developed.

This document is inteded for the key design stakeholders, which include managers and the development team.

1.2 Scope

PowerEnJoy is inteded to be a management system for a car-sharing system that exclusively employs electrical powered cars. The system allows users to find all the available cars near a given location which can be their current position or a specific address typed in. The user can book one of the cars available for a limited amount of time. Once the booked car is reached by the user, it can be unlocked from one of the smart devices of the user. After entering a PIN, decided by the user during the registration phase, on the onboard computer, the engine can be started and the rental begins. The cost of the service is calculated on the total duration of the rental multiplied by a fixed rate per minute. The user is continuously informed about the cost of the ongoing rental by the onboard computer. The user can end the rental by parking the car and stopping the engine.

The PowerEnJoy platform is intended to be available on the major mobile devices, such as smartphones and tablets running iOS [1] or Android [2].

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

Onboard computer An onboard computer is intended to be the embedded device integrated in the car system that provide two main functions: on one hand it shows the user basic information about the ongoing rental, on the other hand it is in charge of all the communications related to the state of the car between the car and the PowerEnJoy management system.

Safe area A safe area is an area, within prede

ned edges, in which the users are allowed to park the rented cars. The users is not allowed to park, and therefore end a rental, if he/she is not inside a safe area.

Sign up Sign up is intended to be the registration phase. During this process, the user provides the system the credetials required along with the information about the documents required.

1.3.2 Acronyms

API Application Programming Interface

BCE Boundary Control Entity

PIN Personal Identification Number

RASD Requirement Analysis and Specification Document

UX User Experience

1.3.3 Abbreviations

DB Database

ID Identification

1.4 References

- Assigments document for the first semester project of the Software Engineering 2 course held at Politecnico di Milano by Mottola Luca and Di Nitto Elisabetta [3].
- PowerEnjoy - Requirement Analysis and Specification Document [4].
- IEEE Standard for Information Technology—Systems Design— Software Design Descriptions [5].
- IEEE Systems and software engineering — Architecture description [6].

- IEEE Recommended Practice for Architectural Description of Software-Intensive Systems [7].

1.5 Overview

A general introduction about the PowerEnJoy project has been given in this chapter. In the following chapters the reader will be provided with a more detailed description of the software design and the system architecture.

In Chapter 2, the overall system architecture is presented. Several models and paradigms have been exploited in order to describe the concepts clearly.

In Chapter 3, the model of the most critical aspects of the algorithm is provided. It explains the logical steps of the algorithm even if it cannot be consider as a real implementation.

In Chapter 4, the user experience and its interaction with the system are presented.

In Chapter 5, the requirements traceability will show which software components are needed in order to satisfy the feasibility of the goals described in the RASD.

Chapter 2

Architectural Design

2.1 Overview

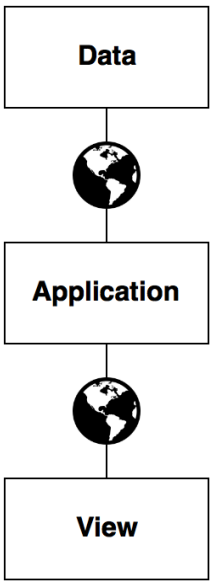

Figure 2.1: The PowerEnJoy system is developed based on a three tiers architecture.

The high level architecture of the PowerEnJoy system is composed by three tiers, corresponding to the view, application and data layer. The desktop application relies on an additional tier that corresponds to the WebServer.

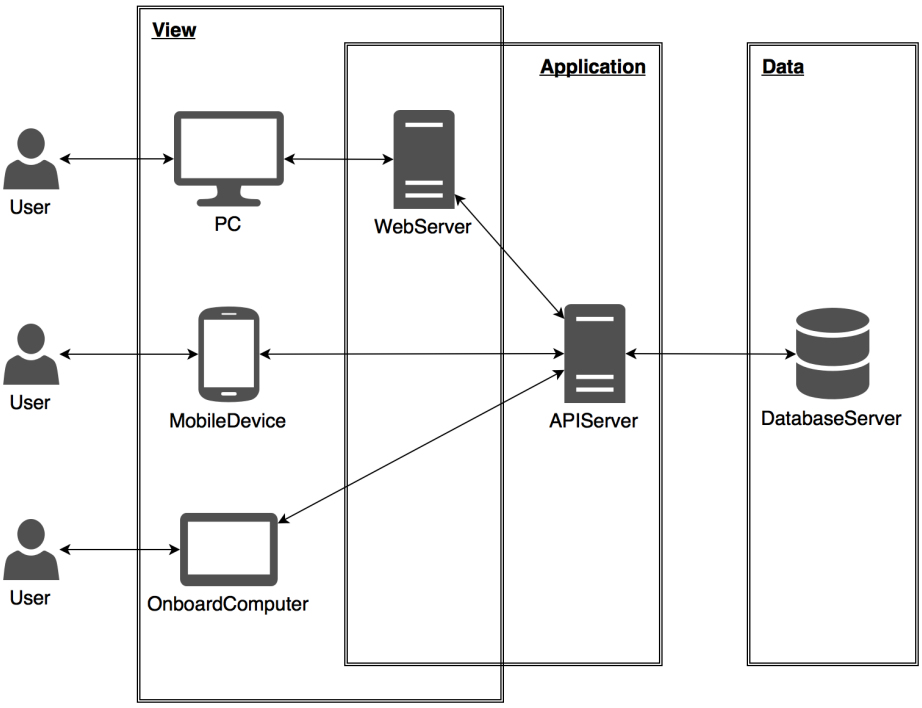

Figure 2.2: The general architecure of the PowerEnJoy system is composed by three types of clients (MobileDevice, PC and OnboardComputer), a WebServer that is shared between the view and the application layer, an APIServer and a database

2.2 High Level Components

From a high level point of view, the PowerEnJoy system is composed by four elements.

The mobile and the desktop application are included in the *Client* component. The role of this component is to allow the user to be able to book and rent cars, which are the main functions of PowerEnJoy. This is possible thanks to the communication that the clients establish with the application server.

The *Application* is the component in which all the logic of the system is executed. The calls to this component, coming from the clients, are routed to the dedicated controller that will be in charge to execute them.

The application level is stateless. For this reason a *Database* is needed to store all the information about the state of the service.

The *Onboard Computer* is the system component that allow all the communication about the car, between the car itself and the application level.

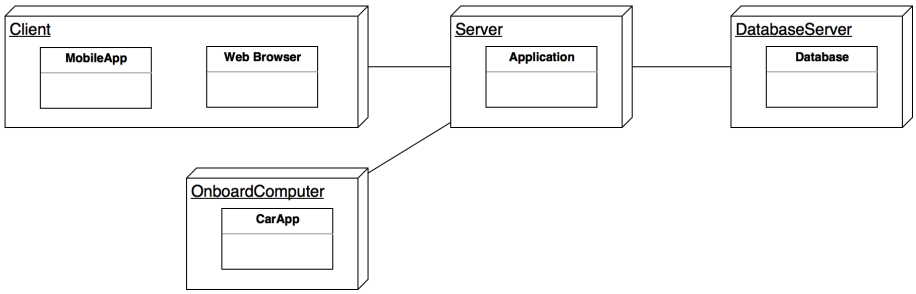

Figure 2.3: The picture above represents the high level model of the PowerEnJoy system infrastructure.

2.3 Component View

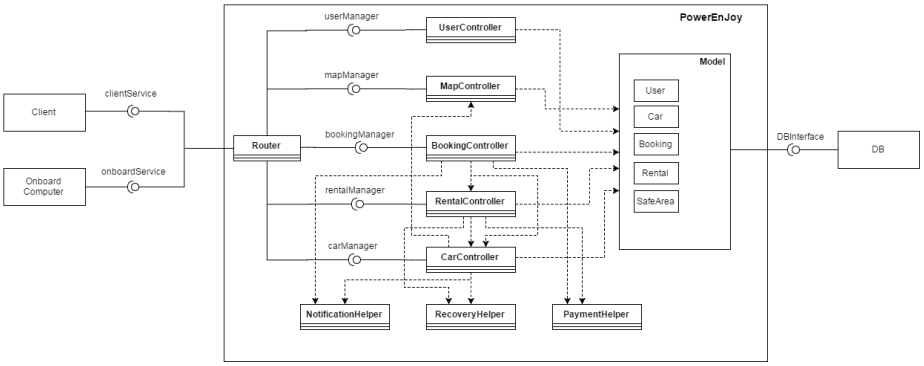

Figure 2.4: The picture above shows system's components, provided and required interfaces and relationships between them.

2.4 Deployment View

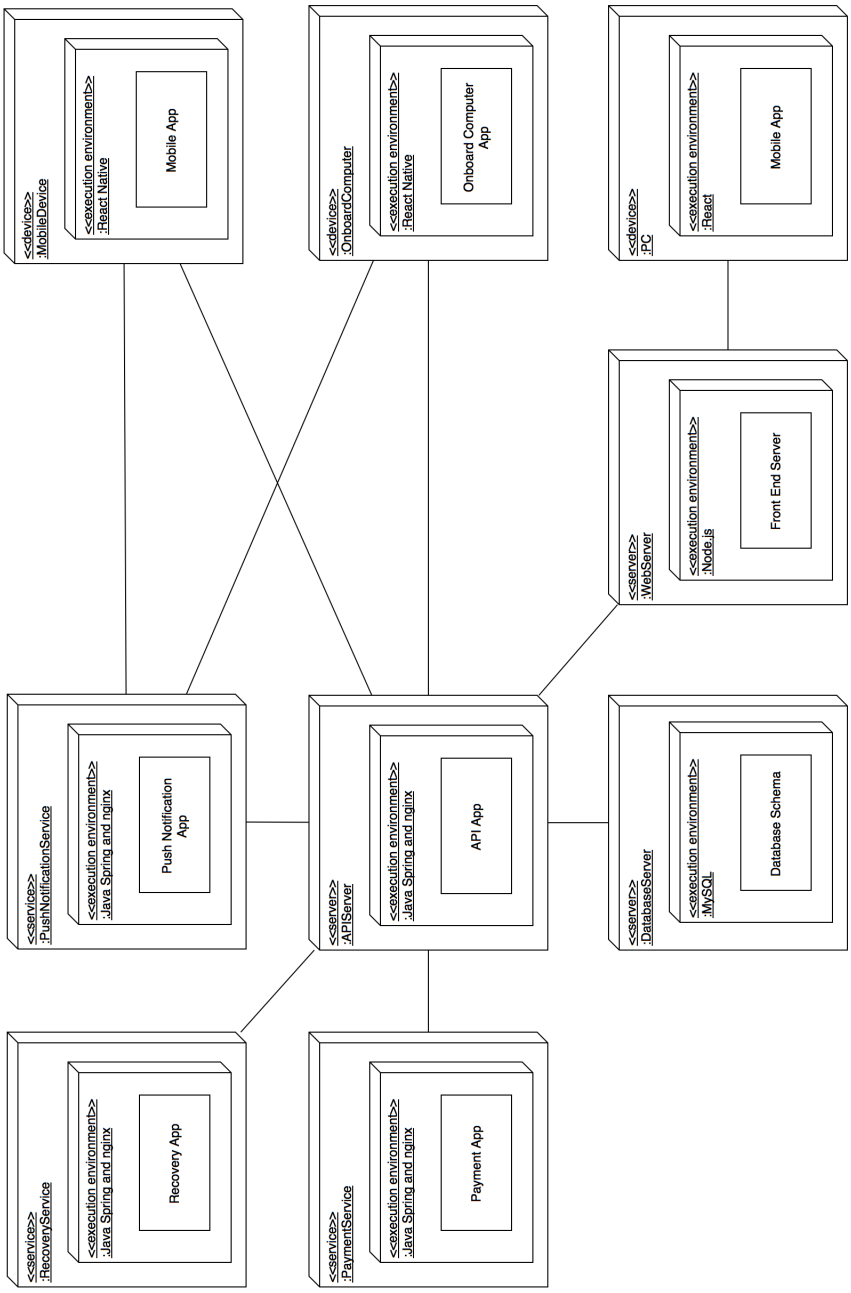

Figure 2.5: The picture above provides an overall description of the system architecture components and their interactions.

From an high level point of view, the deployment view shows the three tiers of the system architecture: client (MobileDevice, PC, OnboardComputer), application (APIServer, WebServer and Services) and data (DatabaseServer). There are some new aspects that have been introduced in this diagram and which have not been mentioned previously. For this reason, further details about them will be given below.

The WebServer is a useful component with respect to the desktop web application. The WebServer contains all the static content of the web pages. This pattern allows the clients to render only the dynamic content of the web pages, such as dynamic data arriving from the APIServer and animations.

The Services, e.g. payment, recovery and notification, provide the system with APIs that can be used to invoke the functions required. For example, in order to execute a payment, the system will call a specific end-point of the PaymentService with the necessary information about the payment requested.

Below, the list of frameworks mentioned in the deployment diagram is provided.

Back End

- Java Spring [8]
- nginx [9]
- Node.js [10]
- MySQL [11]

Front End

- React [12]
- React Native [13]

2.5 Runtime View

Log in

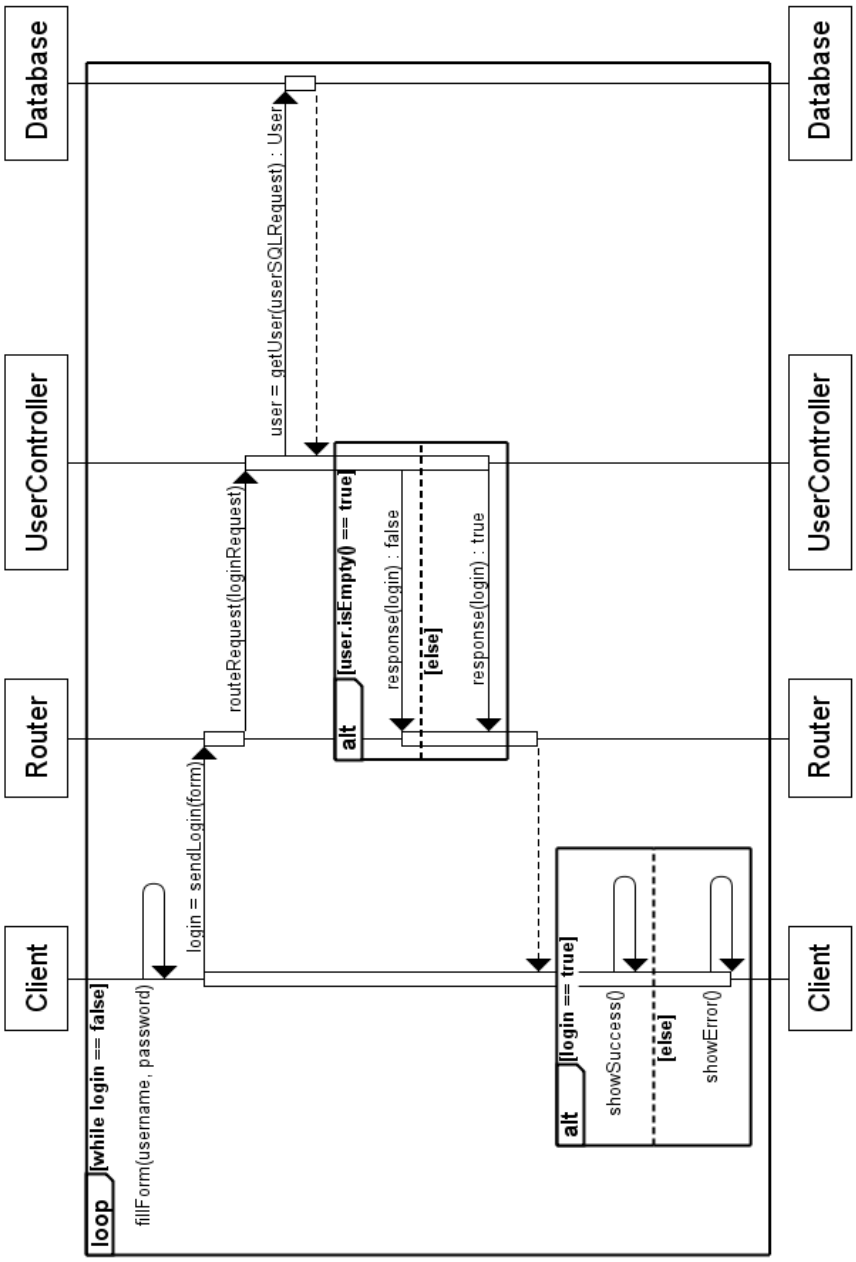

Figure 2.6: Runtime View - Log in

Book a car

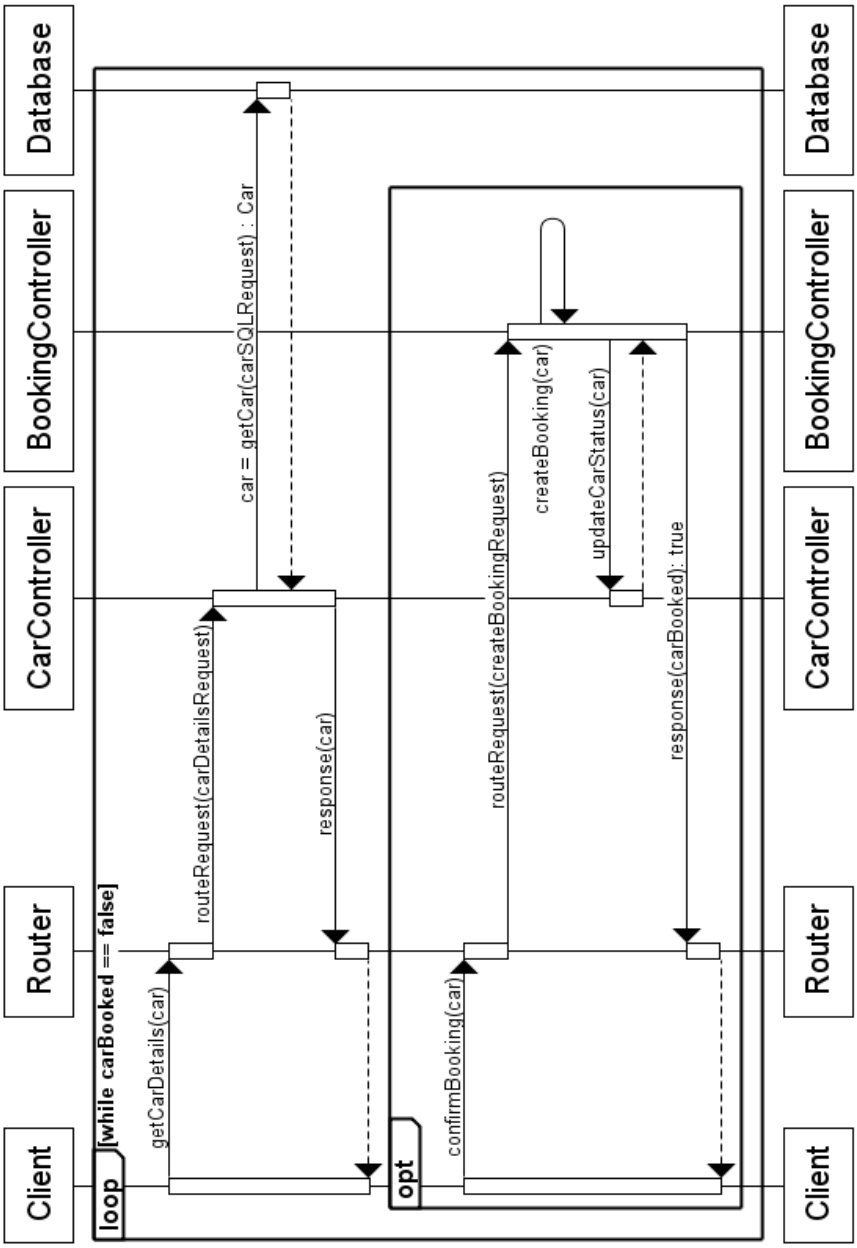

Figure 2.7: Runtime View - Book a car

Cancel booking (elapsed time)

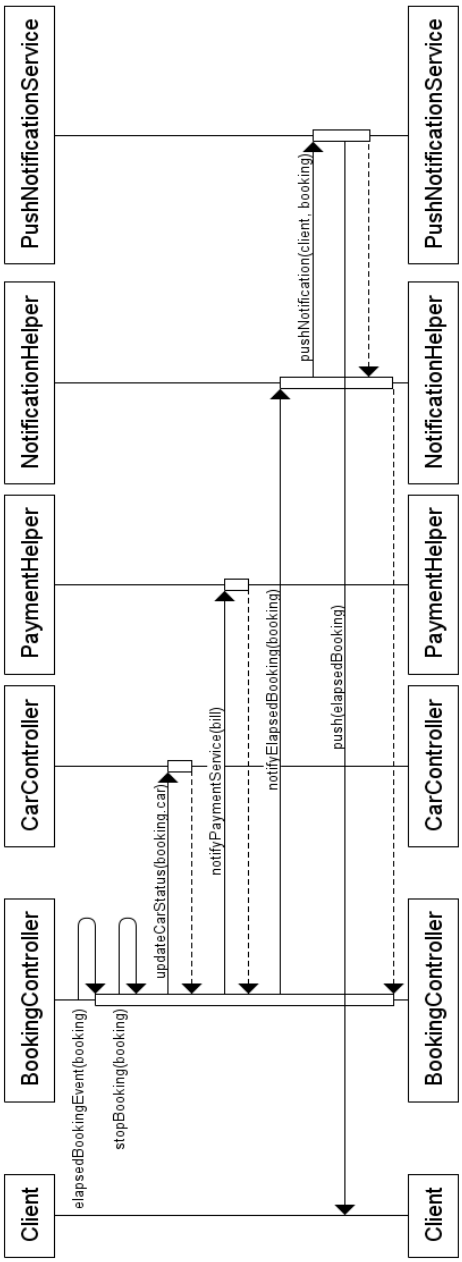

Figure 2.8: Runtime View - Cancel booking (elapsed time)

Unlock car doors

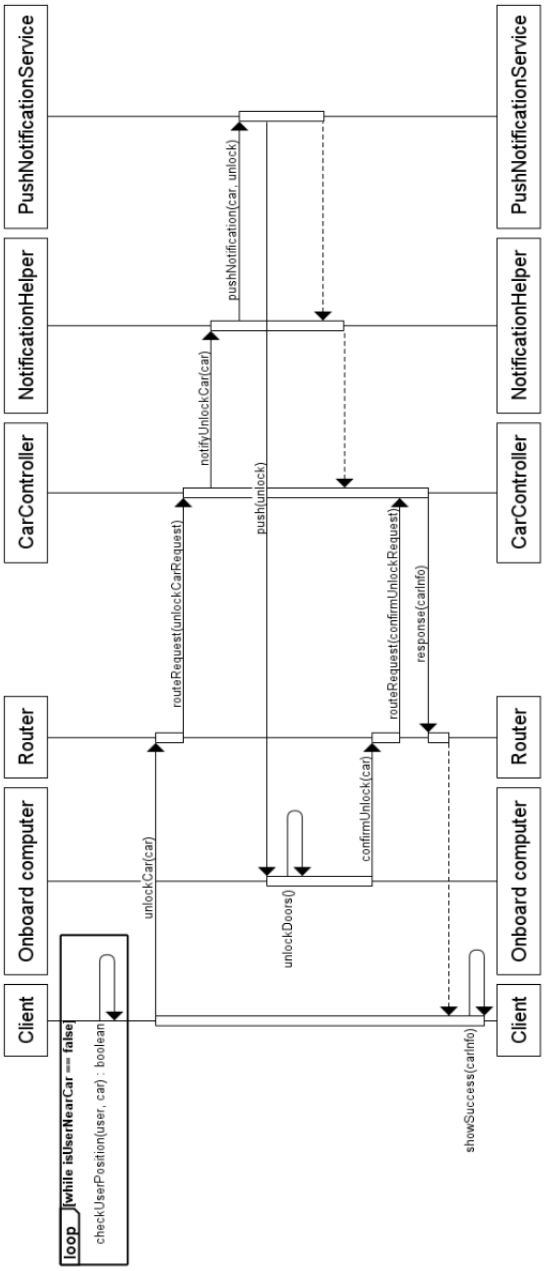

Figure 2.9: Runtime View - Unlock car doors

Begin rental

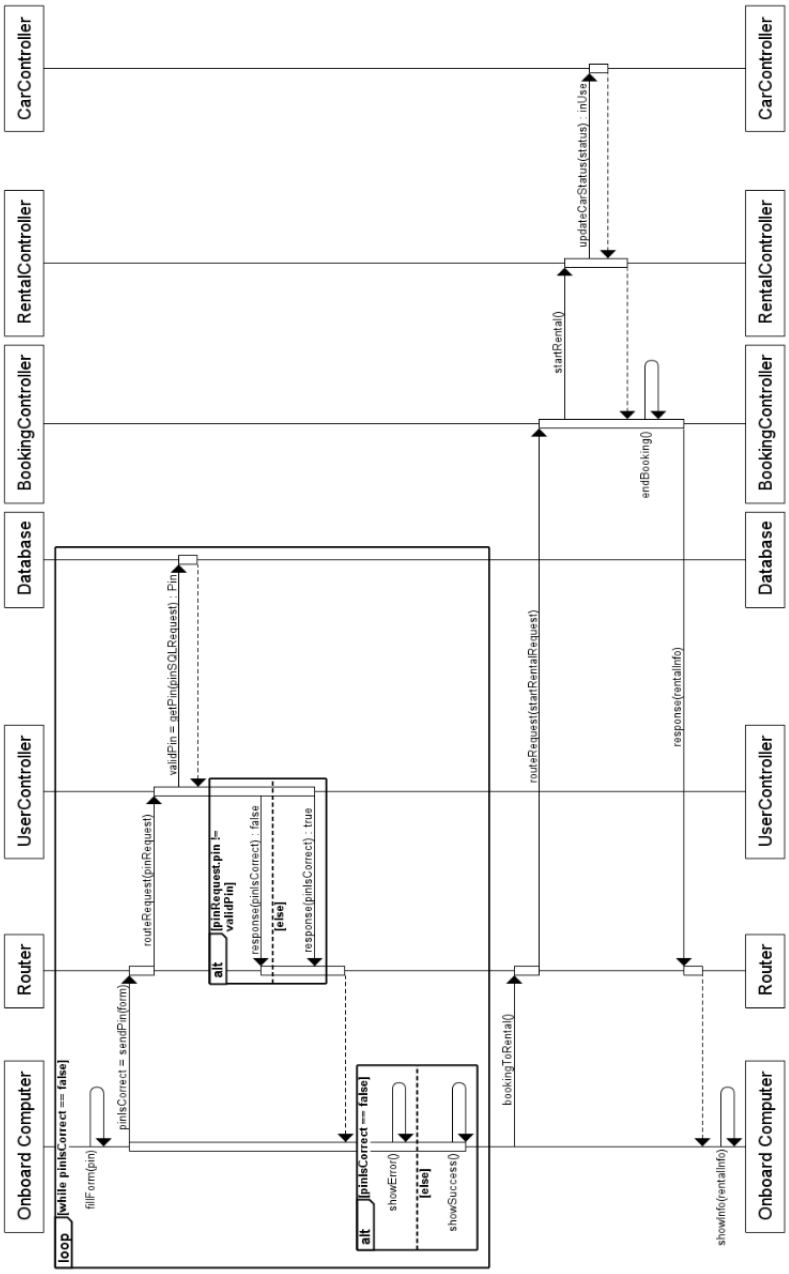

Figure 2.10: Runtime View - Begin rental

End rental

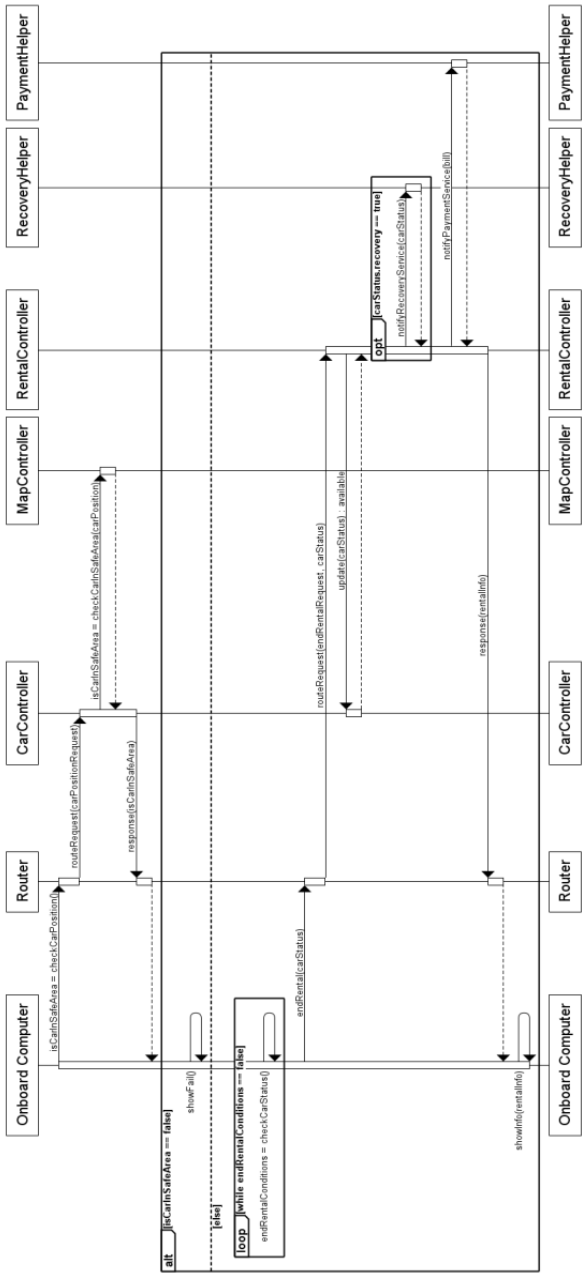

Figure 2.11: Runtime View - End rental

2.6 Component Interfaces

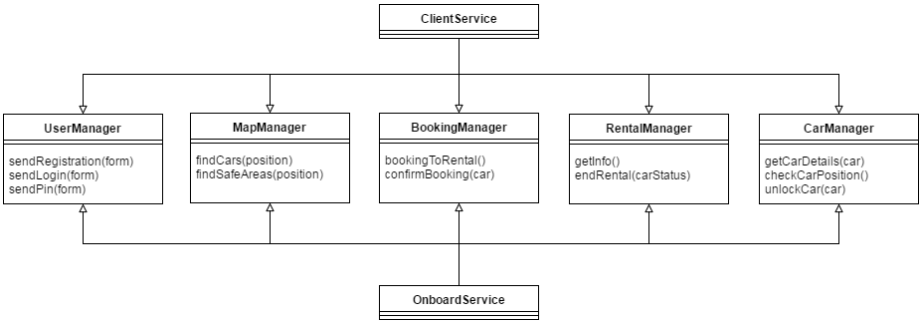

Figure 2.12: The picture above shows system's component interfaces, focusing on methods which can be called externally from user and onboard services.

2.7 Architectural Styles and Patterns

MVC The entire system relies on the Model View Controller pattern. The view corresponds to the applications and the onboard computer, the APIServer implements the controllers while the model is represented by the database.

Client-server All the communications among the entities of the architecture are based on the client-server pattern. A request is sent from the client that will receive a correspondent response synchronously or asynchronously.

Chapter 3

Algorithm Design

Here it's represented the discount calculation, after rental's end, using Java code. Only significants classes, fields and methods are represented for this scope.

```
public class Car {

  /**
   * batteryLevel and position of the car
   * are calculated after the end of the rental
   */
  private int batteryLevel;
  private Position position;

  public int getBatteryLevel() {
    return batteryLevel;
  }

  public void setBatteryLevel(int batteryLevel) {
    this.batteryLevel = batteryLevel;
  }

  public Position getPosition() {
    return position;
  }

  public void setPosition(Position position) {
    this.position = position;
  }

}
```

```
public class Position {
  private float latitude;
  private float longitude;

  public Position(int x, int y){
    this.latitude=x;
    this.longitude=y;
  }

  public float getLatitude() {
    return latitude;
  }

  public float getLongitude() {
    return longitude;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    Position other = (Position) obj;
    if (
      Float.floatToIntBits(latitude) !=
      Float.floatToIntBits(other.latitude)
      )
      return false;
    if (
      Float.floatToIntBits(longitude) !=
      Float.floatToIntBits(other.longitude)
      )
      return false;
    return true;
  }


}
```

```
public class Rental {

  private Car car;
  private int duration;
  private double amountToPay;

  public Rental(Car car, int n, double p){
    this.car=car;
    this.duration=n;
    this.amountToPay=p;
  }

  public Car getCar() {
    return car;
  }

  public int getDuration() {
    return duration;
  }

  public double getAmountToPay() {
    return amountToPay;
  }

  public void setAmountToPay(double amountToPay) {
    this.amountToPay=amountToPay;
  }
}
```

```
public class SafeArea {

  private Position position;
  private boolean powerGridPresence;

  public SafeArea(Position position, boolean powerGridPresence){
    this.position=position;
    this.powerGridPresence=powerGridPresence;
  }

  public Position getPosition() {
    return position;
  }

  public boolean isPowerGridPresence() {
    return powerGridPresence;
  }

}
```

```java
import java.util.Set;

public class Discount {

  private Rental rental;

  public Discount(Rental rental){
    this.rental=rental;
  }

  /**
   * @param atLeastTwoPassengersTime is calculated by the car
   * and indicates the time during which at least two passengers
   * were present in rental's car
   */
  public double discountPassengers(int atLeastTwoPassengersTime){
    if(atLeastTwoPassengersTime<rental.getDuration()) return 0;
    else return rental.getAmountToPay()*0.1;
  }

  /**
   * we suppose that car's batteryLevel belongs to a range
   * between 0 and 100
   */
  public double discountBattery(){
    if(rental.getCar().getBatteryLevel()<=50) return 0;
    else return rental.getAmountToPay()*0.2;
  }

  /**
   * @param map is the set of safeAreas
   * @param charging is true if rental's car has been plugged
   *        to the power grid
   */
  public double discountCharging(
    Set<SafeArea> map,
    boolean charging
  ){
    if(charging==false) return 0;

    for(SafeArea a: map){
      if(
        a.getPosition().equals(rental.getCar().getPosition())==true &&
        a.isPowerGridPresence()==true
      ) return rental.getAmountToPay()*0.3;
    }
```

```
    return 0;
  }

  public void calculateDiscounts(
    int atLeastTwoPassengersTime,
    Set<SafeArea> map,
    boolean charging
  ){
    double totalDiscount=
      this.discountPassengers(atLeastTwoPassengersTime)+
      this.discountBattery()+
      this.discountCharging(map, charging);

    rental.setAmountToPay(rental.getAmountToPay()-totalDiscount);
  }

}
```

Chapter 4

User Interface Design

4.1 Overview

In this section, thanks to UX and BCE diagrams, there are represented main actions which can be performed by user and it's shown how these actions are managed by the system and how they are linked with system's model. Instead, with regards to mockups, they are already represented in RASD's section 2.1.

4.2 UX diagrams

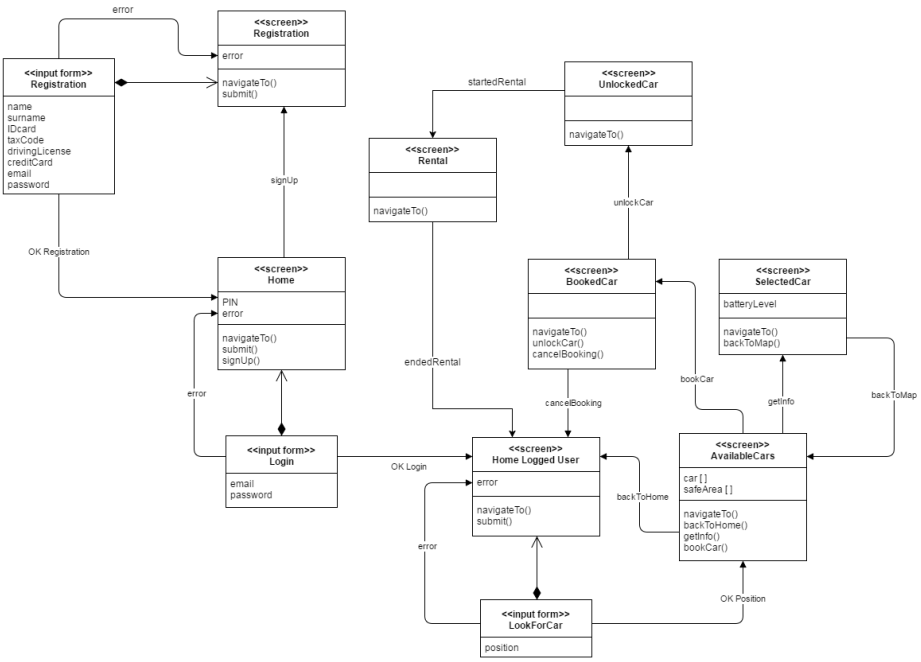

Figure 4.1: UX mobile/desktop.

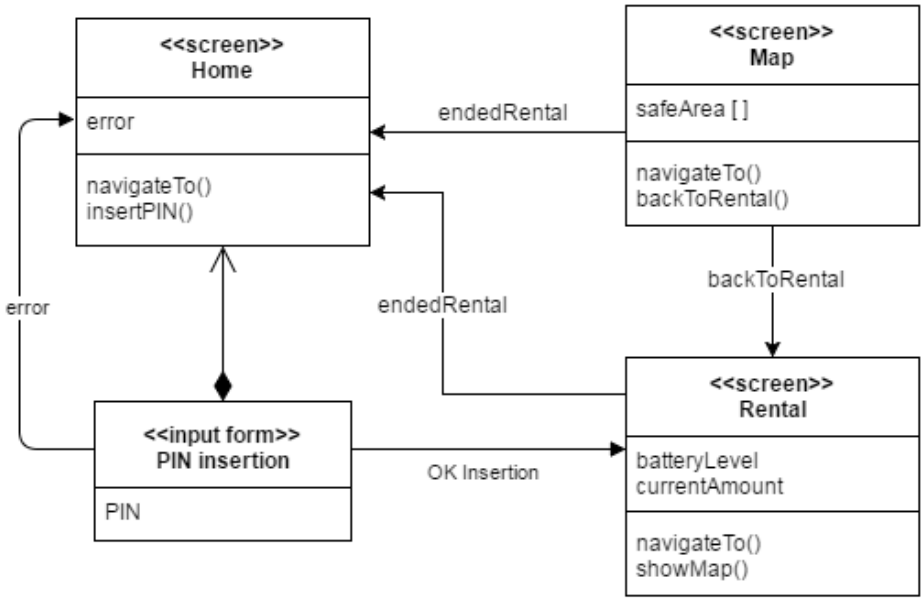

Figure 4.2: UX car's onboard computer.

4.3 BCE diagram

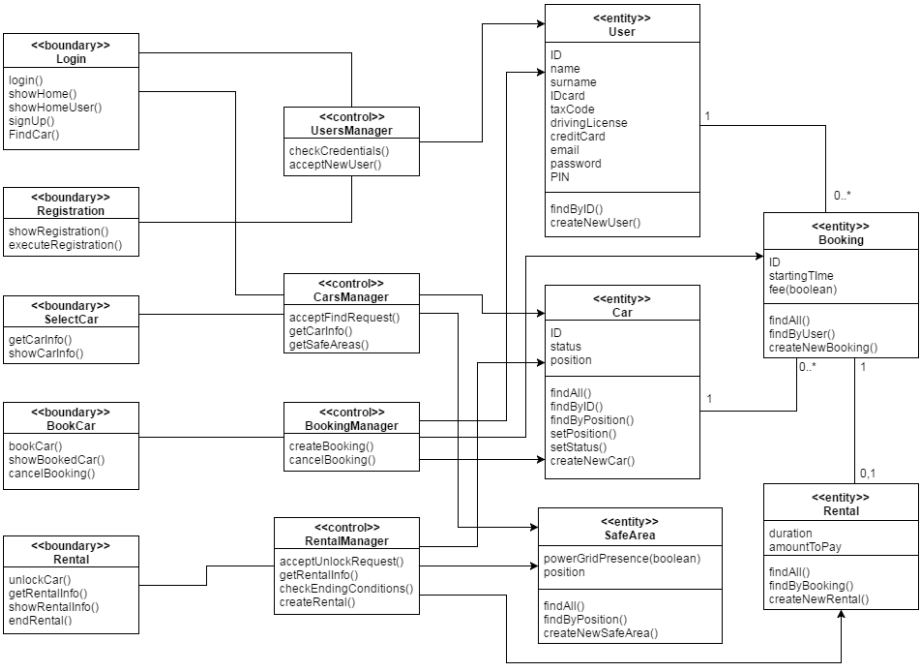

Figure 4.3: BCE diagram.

Chapter 5

Requirements Traceability

The component design of the system has been developed in order to cover all goals and requirements provided in the RASD. Below, the software component needed to guarantee each goal feasibility are reported. The overall component design and interactions are shown in Figure 2.4.

[G1] User is able to register in the system.

- Client app
- UserController

[G2] User registered receives a PIN to insert in the reserved car's onboard computer in order to drive it.

- Client app
- UserController
- NotificationHelper

[G3] User is able to log in the system.

- Client app
- UserController

[G4] The user is able to find the location of all the available cars near is current position or near a specified address.

- Client app
- MapController
- CarController

[G5] User is able to book a car for up to an hour before picking it up.

- Client app

- CarController
- BookingController

[G6] User is able to check the battery level of a car before booking it.

- Client app
- CarController

[G7] User is able to cancel a booking of a car.

- Client app
- BookingController

[G8] If a car is not picked up within one hour from the booking the reservation expires and the user pays a fee of 1 EUR.

- CarController
- BookingController
- PaymentHelper
- NotificationHelper

[G9] User is able to open the reserved car.

- Client app
- CarController
- NotificationHelper

[G10] The user can start a rental by inserting his/her PIN and then by igniting the engine.

- Onboard computer
- UserController
- BookingController
- RentalController
- CarController

[G11] When a user is driving a car, the information about the ongoing rental is displayed by the onboard computer. The displayed information includes: the battery level of the car, the actual cost of the ongoing rental and a map that can be used as a GPS navigation device of to find both safe areas and special parking areas.

- Onboard computer
- RentalController
- CarController

- MapController

[G12] The rental ends when the user parks in a safe area and exits the car.

- CarController
- MapController
- RentalController
- RecoveryHelper
- PaymentHelper
- NotificationHelper

[G13] If the user shares the rented car with at least two passengers during the whole duration of the rental, he/she is eligible for a 10% discount on the total cost of that rental.

- CarController
- RentalController
- PaymentHelper

[G14] If the user leaves a car with the battery level higher than 50%, he/she is eligible for a 20% discount on the last ride.

- CarController
- RentalController
- PaymentHelper

[G15] If the car is left within one of the special parking areas and the user takes care of plugging the car into the power grid, he/she is eligible for a 30% discount on the last ride.

- CarController
- MapController
- RentalController
- PaymentHelper

[G16] If the car is left at more than 3KM from the nearest specia parking area or with the battery charge lower than 20%, the user will pay an extra fee that will correspond to the 30% of the cost of the last ride.

- CarController
- MapController
- RentalController
- PaymentHelper

[G17] In case of low battery of a parked car, damaged car or empty battery, the car becomes unavailable until is recovered by an external company in order to fix all problems, depending on the situation, written above.

- CarController
- MapController
- RecoveryHelper
- NotificationHelper

The Router component and the Database are never mentioned above because the former is needed everytime a request is done by the clients to the server, while the latter is needed everytime a change in the system state happens.

List of Figures

Bibliography

[1] Apple Inc. iOS. Available at `http://www.apple.com/ios/` (November 26, 2016).

[2] Google Inc. Android. Available at `https://www.android.com/` (November 26, 2016).

[3] L. Mottola and E. Di Nitto. *Assignments AA 2016-2017*. Politecnico di Milano.

[4] M. Penco and R. Pressiani. PowerEnjoy - Requirement Analysis and Specification Document. Available at `https://github.com/rpressiani/se2-project-2016/blob/master/releases/RASD.pdf` (November 27, 2016).

[5] ISO/IEC/IEEE Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46, Dec 2011.

[6] IEEE Standard for Information Technology–Systems Design–Software Design Descriptions. *IEEE STD 1016-2009*, pages 1–35, July 2009.

[7] ISO/IEC Standard for Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems. *ISO/IEC 42010 IEEE Std 1471-2000 First edition 2007-07-15*, pages c1–24, July 2007.

[8] Pivotal Software. Spring. Available at `https://spring.io` (December 9, 2016).

[9] NGINX Inc. nginx. Available at `https://www.nginx.com` (December 9, 2016).

[10] Node.js Foundation. Node.js. Available at `https://nodejs.org` (December 9, 2016).

[11] Oracle Corporation. MySQL. Available at `https://www.mysql.com` (December 9, 2016).

[12] Facebook Inc. React. Available at `https://facebook.github.io/react/` (December 9, 2016).

[13] Facebook Inc. React Native. Available at `https://facebook.github.io/react-native/` (December 9, 2016).