Design Document

Software Engineering 2

December 11,2016

Authors:

Claudio Salvatore	Arcidiacono	Matr 879109
Antonio	Di Bello	Matr 878786
Denis	Dushi	Matr 879115

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to explain the architectural decisions and tradeoffs chosen in the design process of the PowerEnJoy system. The document aims to provide a view of:

- The main software components of the system and their interactions
- The architectural styles and patterns used in the design process
- The user interfaces and how the interaction between user and system are managed
- How the design decisions satisfy the requirements previously defined in the RASD

1.2 Scope

The project to be developed is PowerEnJoy, a digital management system for a car-sharing service that exclusively employs electric cars. The system has to allow users to register and log in to the service, to see which car are available and were the cars can be parked. In addition to the basic functionality of a car-sharing service the system has to incentivize virtuous behaviors of the users, like plugging the car to a charging station or leaving the car with enough charge, in order to have as much as possible a self-sustainable service.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Guest: is an user that is not registered yet to PowerEnJoy.
Client: is an user that has completed successfully the registration procedure.
Logged client: is a client that has logged into PowerEnJoy.
Blocked client: is a client that is not allowed to log in to system because he has some debits with PowerEnJoy.
User: a client or a guest.
Virtuous behavior: the client has a "Virtuous behavior" if he performs one or more of the following actions:

- transport at least other two passengers into the car.
- leaves the car with less than 50% of the battery empty.
- plugs the car into the power grid of a charging station.

Rent: defines the period of time that starts when the logged client opens the car and ends when the door of the car are locked due to a "end rent" request of the logged client.
Pin: a personal sequence of four numbers given by the system after the registration, necessary to start the engine.
Map Information: denotes the following information:

- location of available cars.
- location of charging areas.
- for each charging area the number of available charging spots.
- boundaries of safe areas.

Not valid data: syntactically incorrect data (e.g. mail not in the format local-part@domain).
Credentials: email and password used to log into PowerEnJoy.
Valid Credentials: email and password related to a registered user.
Car information Menu: displayed when a client selects a car on the map. In this menu are displayed the information about the car (remaining charge percentage and license plate number)

and a button that can be used to reserve the car.
Safe Areas: areas in which is permitted to the client to end the rent and leave the car.
Valid Payment Information: Credit or prepaid card with at least a minimum amount of money.

1.3.2 Acronyms

- **RASD:** requirements analysis and specifications document
- **UX:** user experience
- **BCE:** boundary-control-entity diagram
- **MVC:** Model-View-Controller pattern

1.4 Reference Documents

- RASD Document
- Assignment AA 2016-2017.pdf
- Sample Design Deliverable Discussed on Nov.2.pdf

1.5 Document Structure

- **Introduction:** This section gives a brief introduction of the Design Document.
- **Architectural Design:** this section is divided as follows:
  1. *Overview:* this part defines the high level components and their interaction
  2. *Component view:* this part provides a definition of the software component of the system and their interaction
  3. *Deployment view:* this part describes how the component previously defined must be deployed.
  4. *Runtime view:* this part provides a set of sequence diagrams to explain the way components interact to accomplish specific tasks
  5. *Component interfaces:* this part gives a definition of the main interfaces that components provide.
  6. *Selected architectural styles and patterns:* this part contains an explanation of which architectural styles and patterns have been used in the design process and why those decisions have been applied.
  7. *Other design decisions*
- **Algorithm Design:** this section defines the most critical and relevant parts of the system using Java code.
- **User Interface Design:** this section provides mockups, UX and BCE diagrams to explain how user and system will interact.
- **Requirement Traceability:** this section explains how the requirements presented in the RASD are satisfied by the design decisions made.

2 Architectural design

2.1 Overview

The high level architecture is composed of three layers.The most external is the client layer, it provides the interfaces that allows user-system interactions: the web application, the mobile application and the car system. The second layer is composed by an application logic server that will handle the users requests and that will send the requested data to the first layer components, this layer will also send modification data requests to the third layer. The third layer is composed by a database and his DBMS, this layer will handle the user data, requests data and cars data.

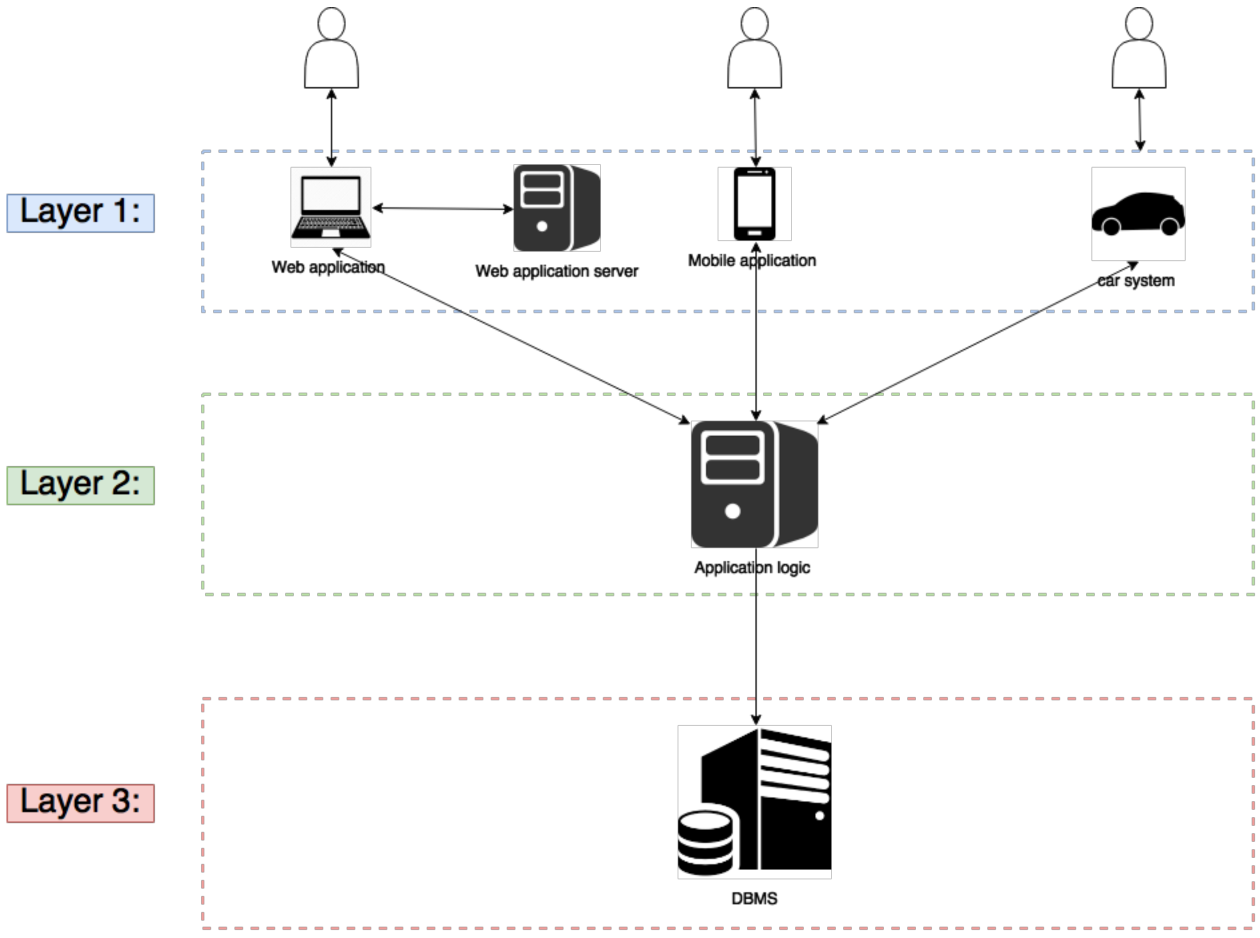


Figure 1: Layers architecture

2.2 High level components and their interaction

In the first level we find two components, the client and the car system. The client is composed of a web application and a mobile application.The web application has to interact with a dedicated server in order to work properly, it should be implemented in java script and HTML and it should have its standalone logic, making to the server only the request for getting data about cars and the map, the request of starting a reservation and the requests of updating the account information. The mobile application should have his standalone logic as well and it will make to the server the same requests as the web application. The car system will be a java application accessible from the car display, through which the user can insert the pin, report damages and end the rent.The request will be send to the server. At the second level there is the server,which should be written in php and should have an event driven architecture because it should handle multiple requests at the same time asynchronously. The server interacts with the client by sending notifications about the payments and requesting periodically the position. The server interacts with the car requiring lock/unlock doors and enable engine.The server also requires information such as the position, the remaining charge, the number of passengers and if the car is plugged or not to a charging station. At the third level we find the DBMS server that handles a SQL database. The server interacts with the DBMS with some fetch data request and with some data update requests.

2.3 Component view

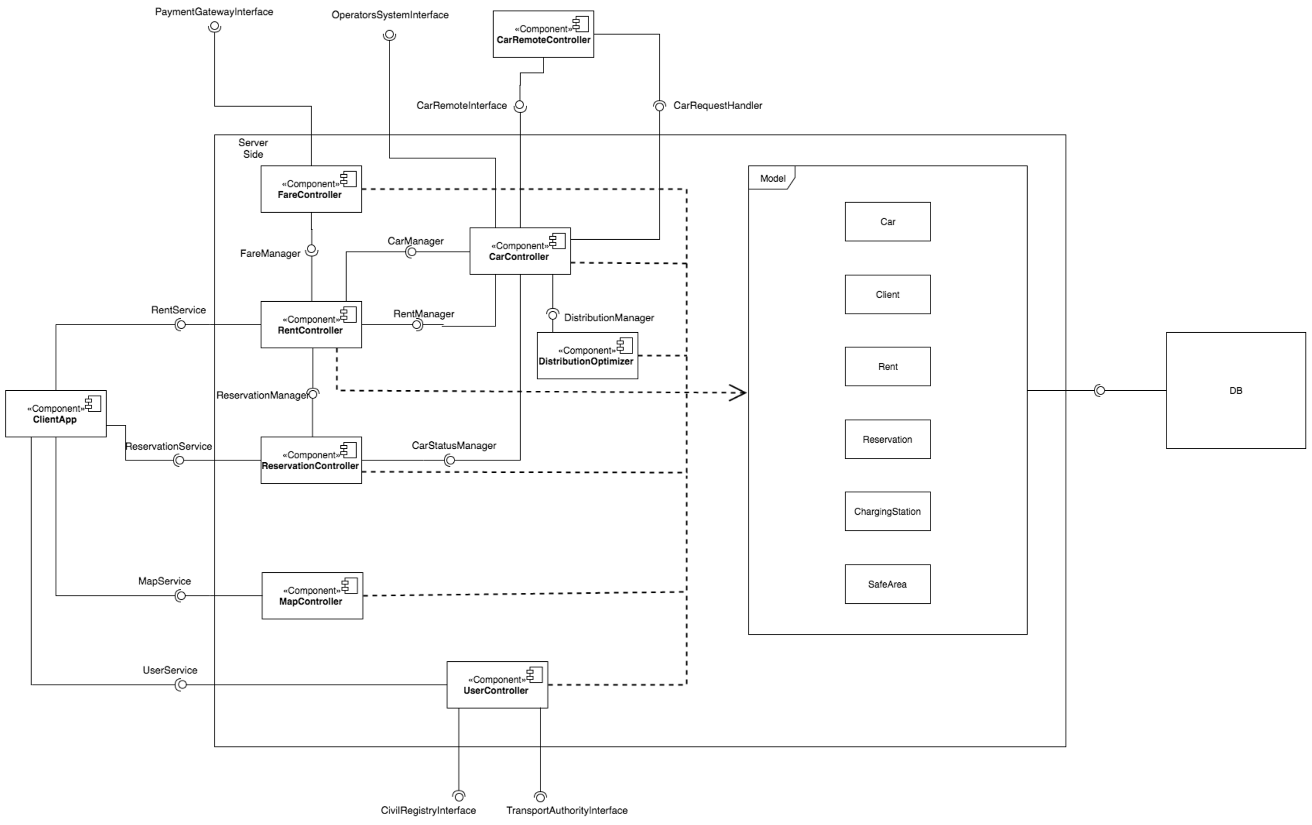


Figure 2: Component diagram.

2.4 Deployment view

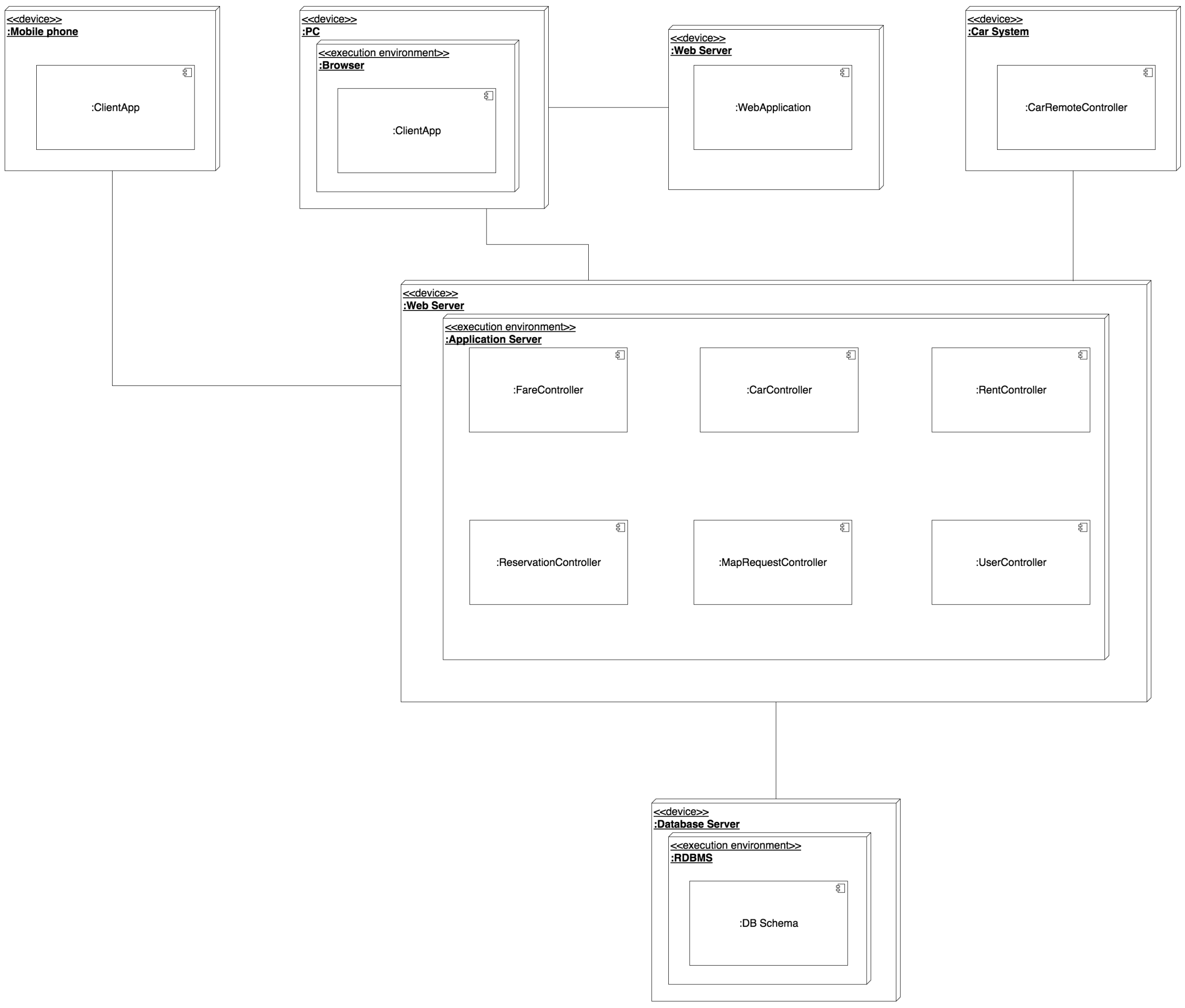


Figure 3: Deployment diagram.

2.5 Runtime view

Reserve a car

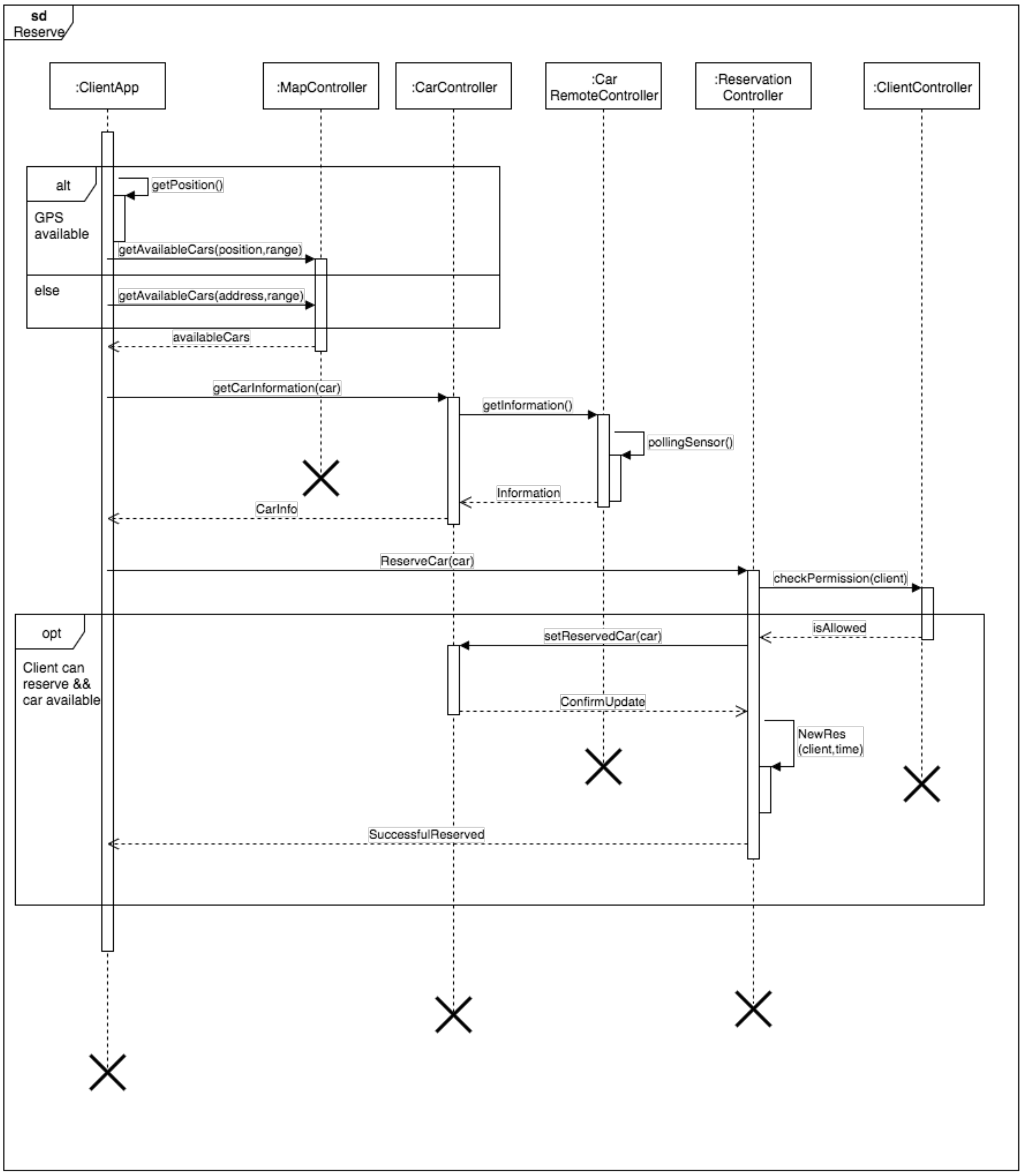


Figure 4: Runtime diagrams for reserve a car.

Rent a car

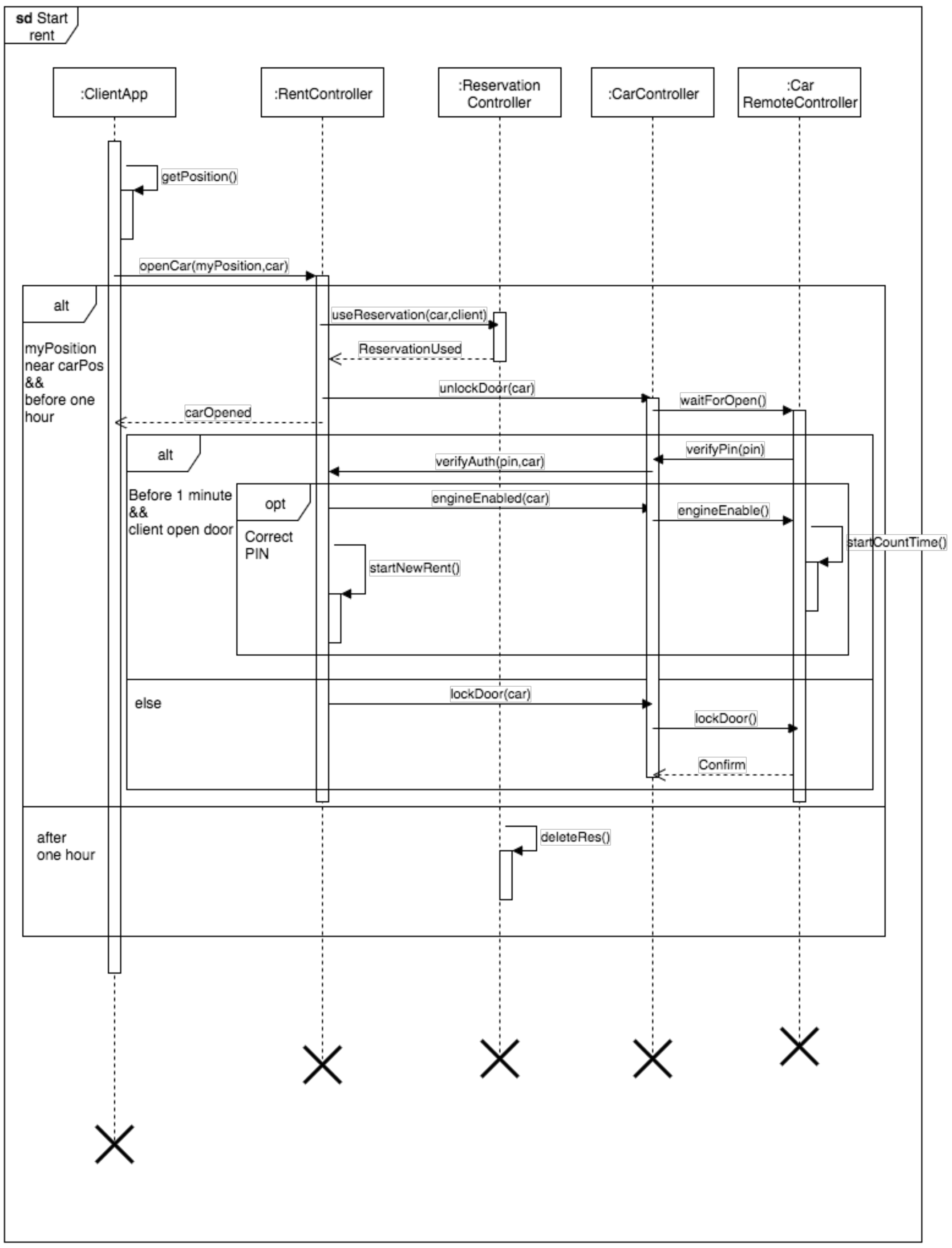


Figure 5: Runtime diagrams for start a rent.

End a rent

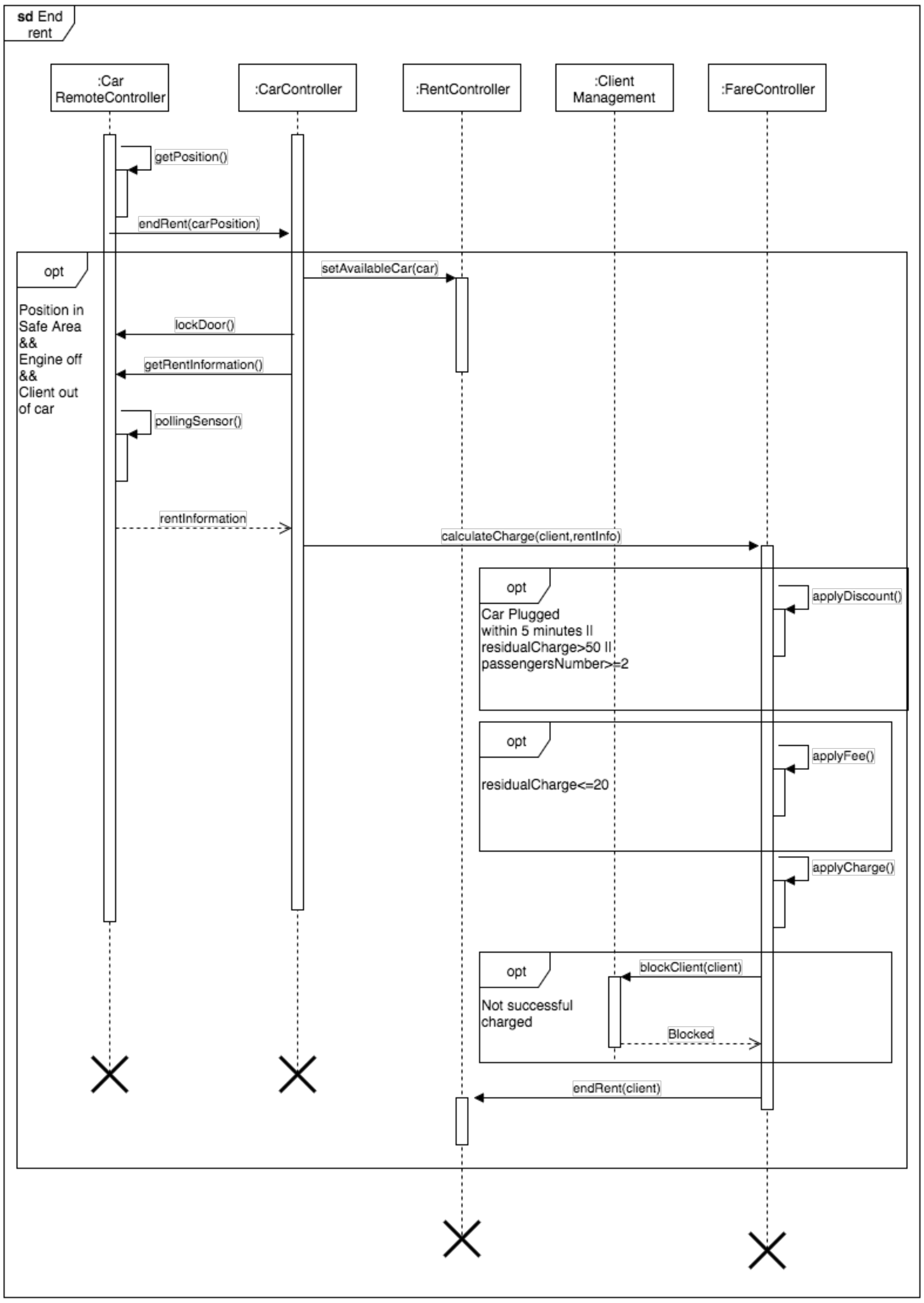
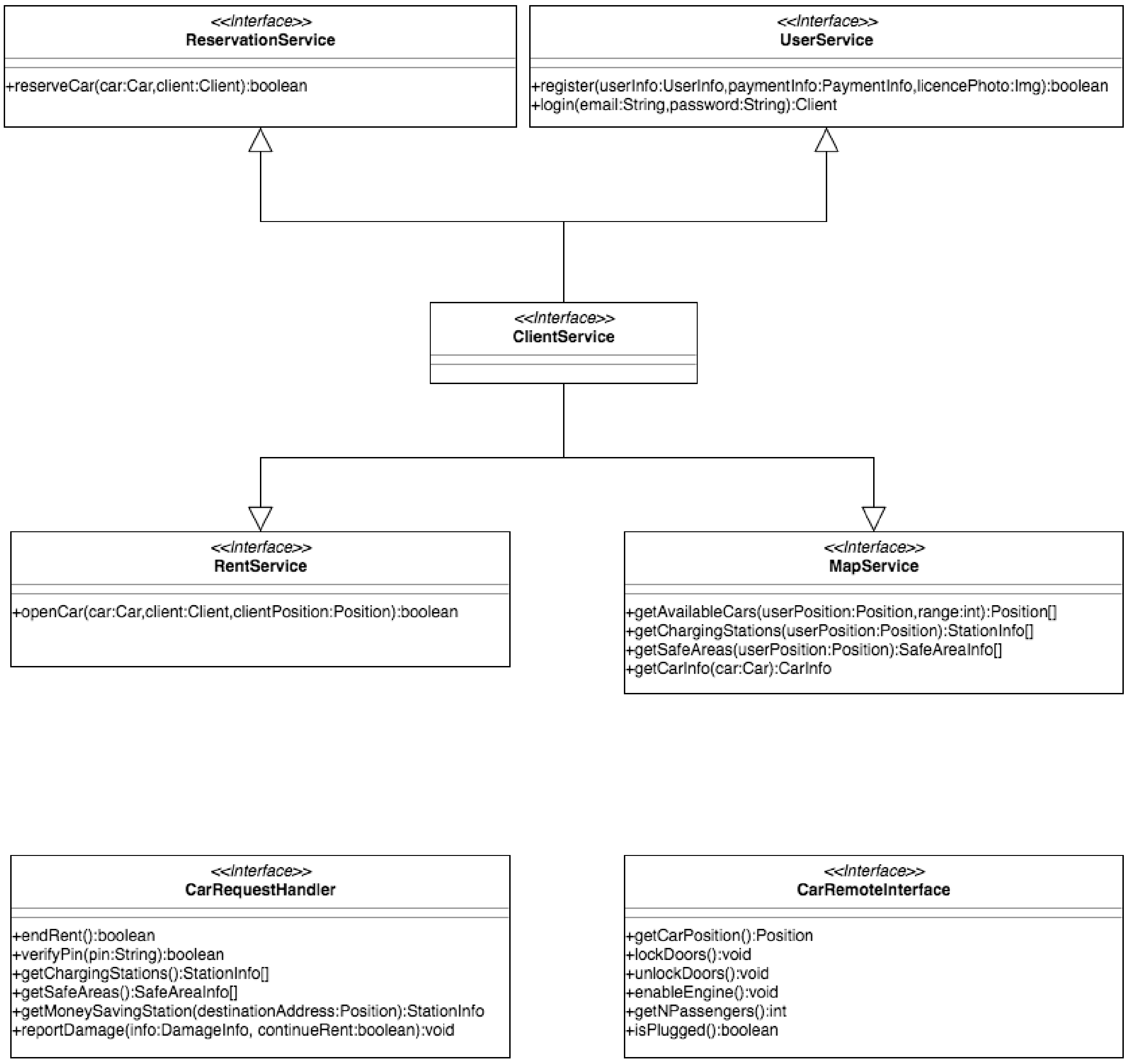


Figure 6: Runtime diagrams for end a rent.

2.6 Component interfaces



<<Interface>>
ReservationService
+reserveCar(car:Car,client:Client):boolean
<<Interface>>
UserService
+register(userInfo:UserInfo,paymentInfo:PaymentInfo,licencePhoto:Img):boolean
+login(email:String,password:String):Client
<<Interface>>
ClientService
<<Interface>>
RentService
+openCar(car:Car,client:Client,clientPosition:Position):boolean
<<Interface>>
MapService
+getAvailableCars(userPosition:Position,range:int):Position[]
+getChargingStations(userPosition:Position):StationInfo[]
+getSafeAreas(userPosition:Position):SafeAreaInfo[]
+getCarInfo(car:Car):CarInfo
<<Interface>>
CarRequestHandler
+endRent():boolean
+verifyPin(pin:String):boolean
+getChargingStations():StationInfo[]
+getSafeAreas():SafeAreaInfo[]
+getMoneySavingStation(destinationAddress:Position):StationInfo
+reportDamage(info:DamageInfo, continueRent:boolean):void
<<Interface>>
CarRemoteInterface
+getCarPosition():Position
+lockDoors():void
+unlockDoors():void
+enableEngine():void
+getNPassengers():int
+isPlugged():boolean

Figure 7: Component interface diagram.

2.7 Selected architectural styles and patterns

Overall architecture

Our application will be divided into 3 tiers:

1. Database handled by a DBMS
2. Application logic
3. Thin Client (a simple interface to the application logic layer)

Design Patterns

MVC: The model-view-controller is widely used in our application.The model is represented by the database, the controller is our application logic and the view is the web app and the mobile app.

Client-Server The Client Server Paradigm is used in order to maximize the simplicity of our application.The client is represented by the web application and the mobile application and the server is represented by our application logic and the DBMS

2.8 Other design decisions

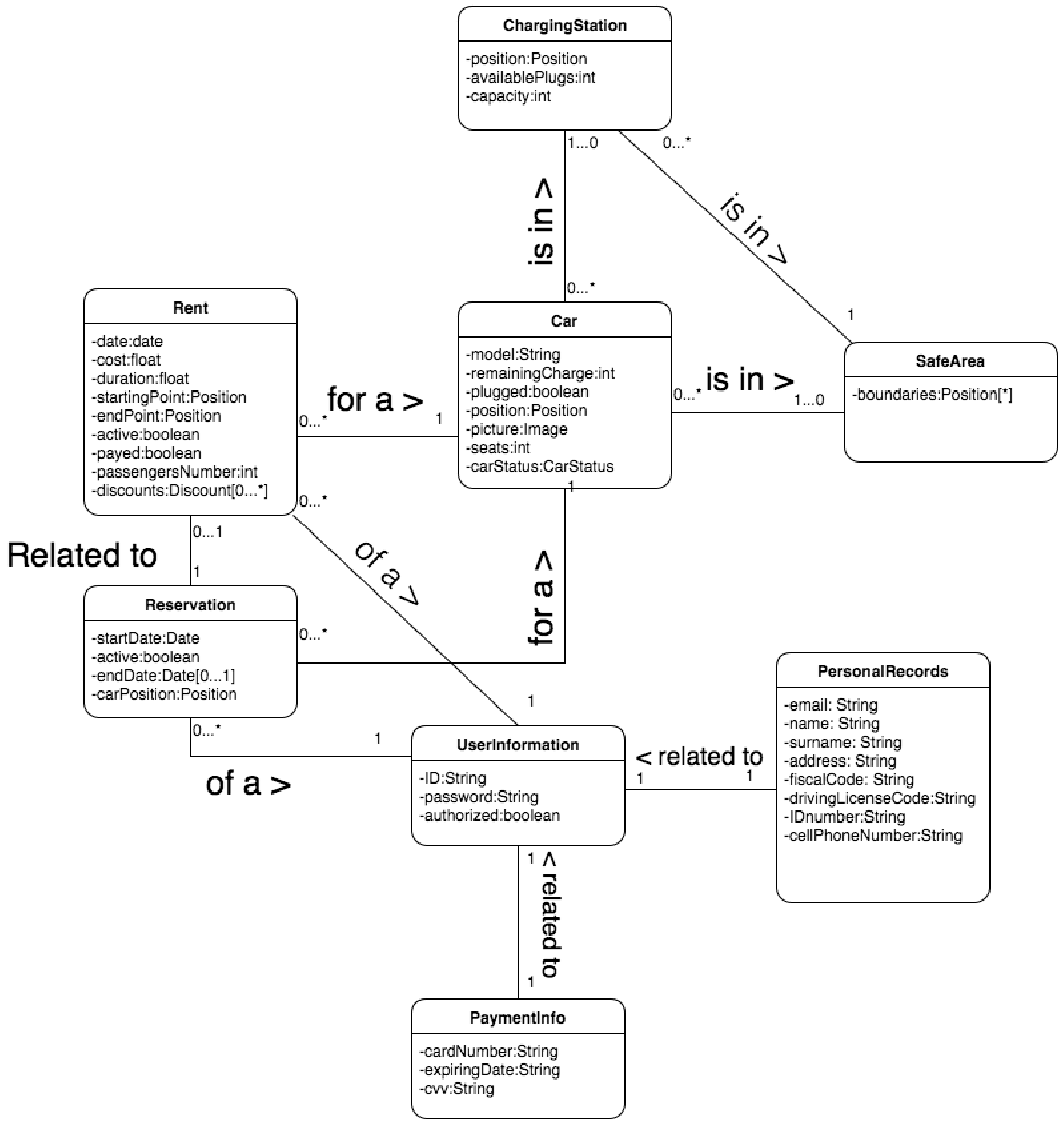


Figure 8: ER diagram.

3 Algorithm design

Here there is a little view (for example written in Java) of some important algorithm used in the system. Here below, it is shown how the system indicates a charging station near the destination in order to guarantee a uniform distribution of cars in the city.

```java
/**
 * Example of some important classes.
 **/

package com.sweng.powerenjoy.routeOptimizer;

import java.util.Math;

public class DistributionOptimizer {

     /**
     * Calculate the charging station where the client should
     * release the car to obtain the "money saving" discount.
     *
     * @param destination  The final destination of the client.
     * @throw NotAvailStationException  If there aren't near
     *                                  stations have a free plug.
     * @return    The position of the recommended charging station.
     */
    static Position RecommendPosDiscount(Position destination)
       throws NoAvailStationException {

        int distance;
        int minCarNumber=Integer.LIMIT_MAX;
        int carNumber;
        ChargingStation rightStation;

        for (ChargingStation station : allChargingStations){
            distance=destination.calcDistance(station.getPosition());
            if(distance<MAX_DISTANCE){
                carNumber=station.getNumberCarPlugged();
                if(carNumber<minCarNumber &&
                   station.getNumberAvailablePlug()>2){
                    rightStation=station;
                }
            }
        }
        if (rightStation==null)
            throw new
               NoAvailStationException(destination,MAX_DISTANCE);
        return rightStation.getPosition();
    }

}


public class ChargingStation {

    private Position position;
    private int numberPlug;
    private Vector<Car> cars;
    //...
    public int getNumberCarPlugged(){
        return numberPlug-this.getNumberAvailablePlug();
    }

    int getNumberAvailablePlug(){
        int n=numberPlug;
        for(Car car : cars){
        /**
        * For future optimization we can consider the plug
        * attacched to a reserved car as free.
        * ... && car.getStatus!=Status.RESERVED)
        **/
                if (car.isPlugged())
                n--;
        }
        return n;
    }

}

/**
 * One address corresponds to a set of coordinates.
 * There aren't two different position with the same coordinates.
 **/
public class Position {

    private Coordinate coordinate;
    private String addressName;
    private int civicNumber;

    public int calcDistance(Position p){
        int x1,x2,y1,y2,d;
        x1=coordinate.getRelativeX();
        y1=coordinate.getRelativeY();
        x2=p.getRelativeX();
        y2=p.getRelativeY();
        d=sqrt( pow(x1-x2,2) + pow(y1-y2,2) );
        return d;
    }

}
```

4 User interface design

4.1 Mockups

Users can interact with our system through the mobile application, the website and the display located inside the car.

View available car information in mobile application

This mockup shows how the information of an available car (charge level and distance from the client) can be displayed in the mobile app. The logged client has the possibility to reserve the car by tapping on the "Reserve" button.

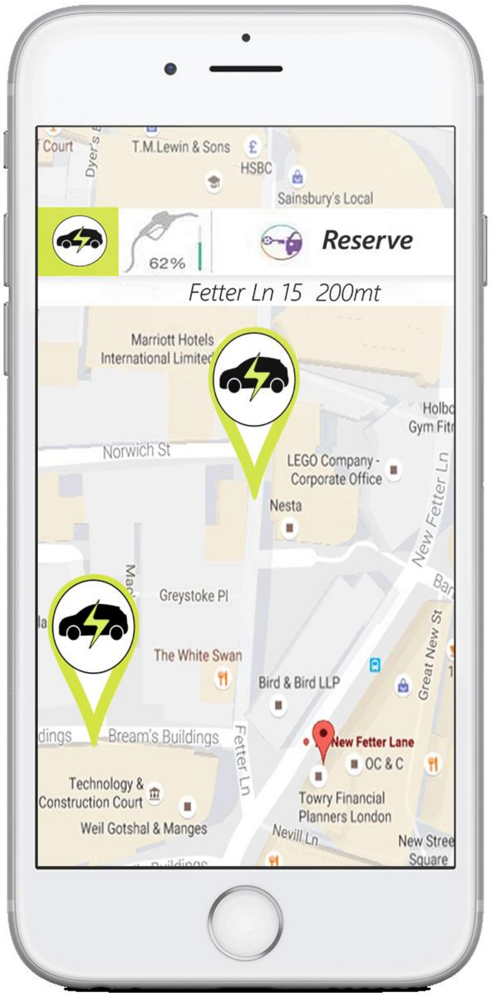


Figure 9: View available car information in mobile application.

View map on the website

This mockup shows the map that a generic user (guest or client) can see on the PowerEnJoy website. On the map are displayed the boundaries of the safe areas in which a client can leave the car and the charging stations with the number of available charging spots. The user can insert an address in order to see the available cars within a certain distance.

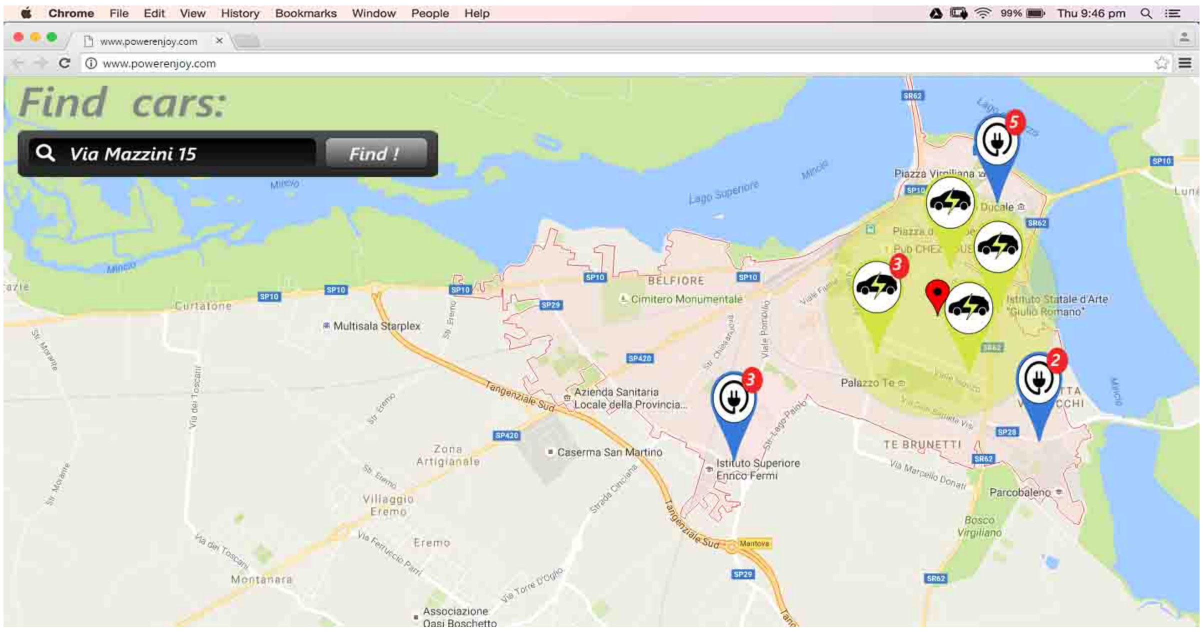
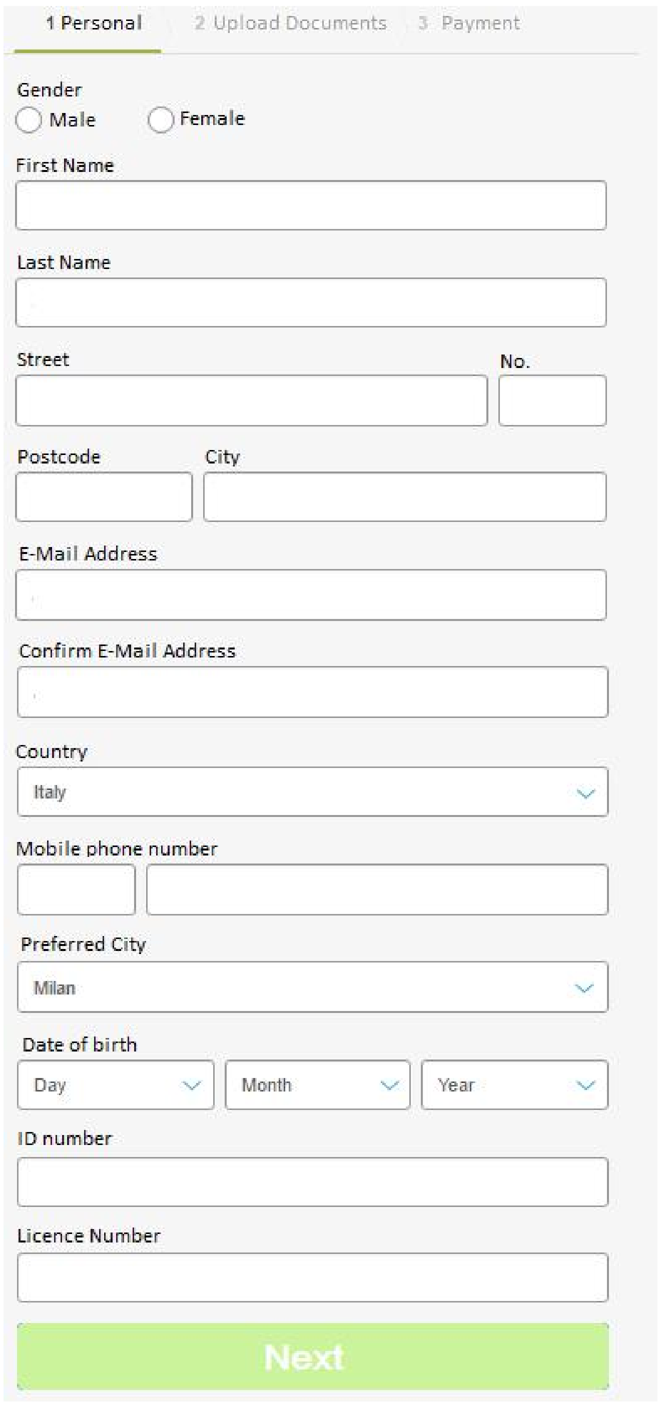


Figure 10: View map on the website.

Registration form

The mockup above shows the registration procedure that a guest has to complete in order to become a client and have access to the PowerEnJoy car sharing.



Gender
Male
Female
First Name
Last Name
Street
No.
Postcode
City
E-Mail Address
Confirm E-Mail Address
Country
Italy
Mobile phone number
Preferred City
Milan
Date of birth
Day
Month
Year
ID number
Licence Number

Figure 11: Registration form.

Insert pin on the car display

This mockup shows the form in which the client has to insert his pin. After inserting the correct pin the client can start the engine of the car.

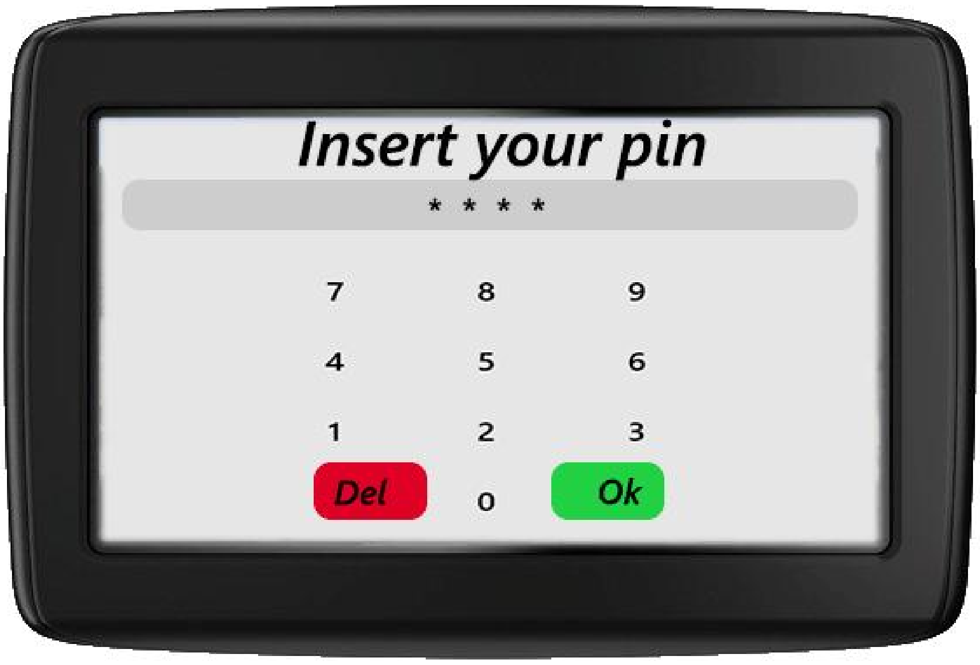


Figure 12: Insert pin on the car display.

4.2 UX Diagrams

UX diagram mobile app

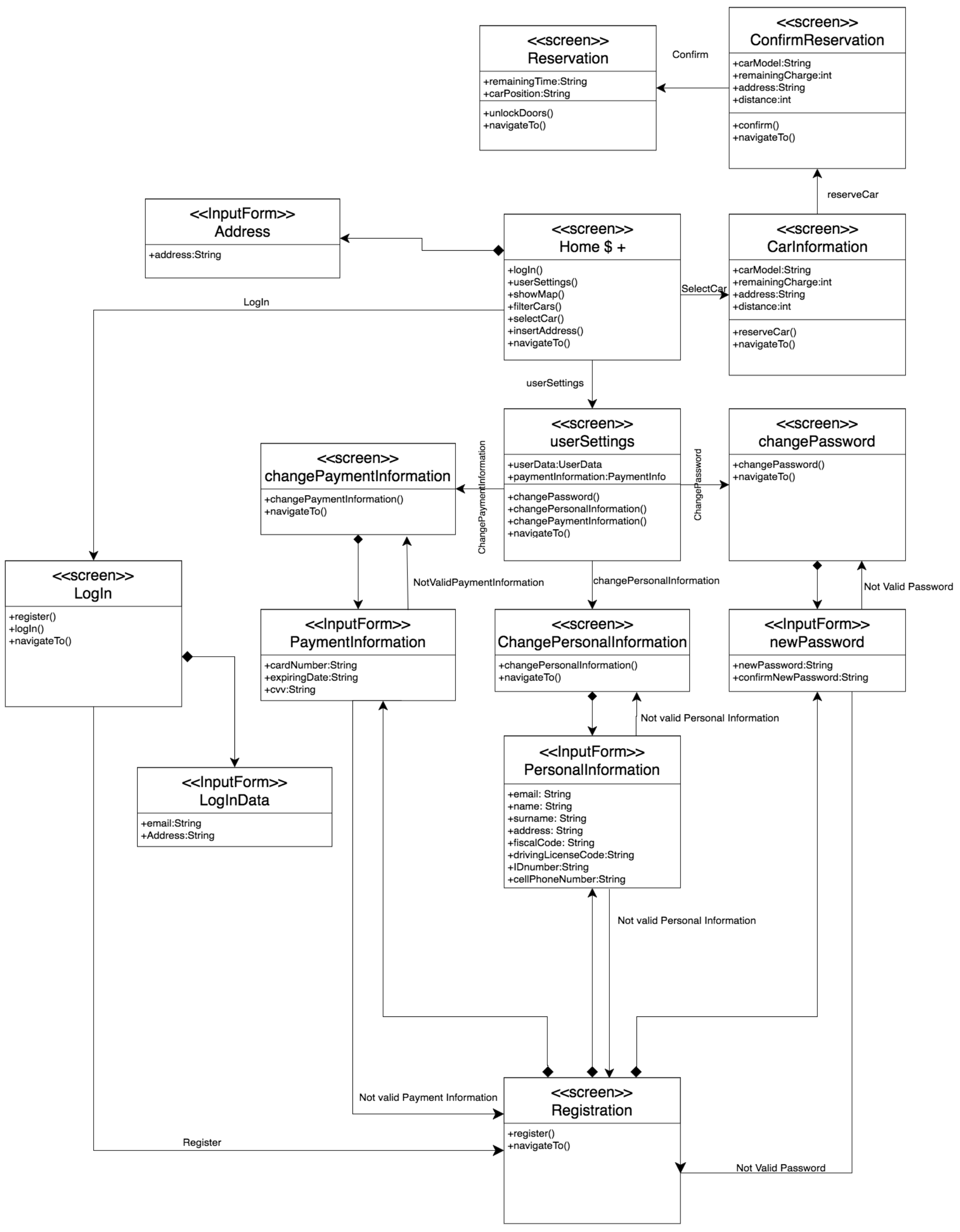


Figure 13: User Experience diagram app.

UX diagram car

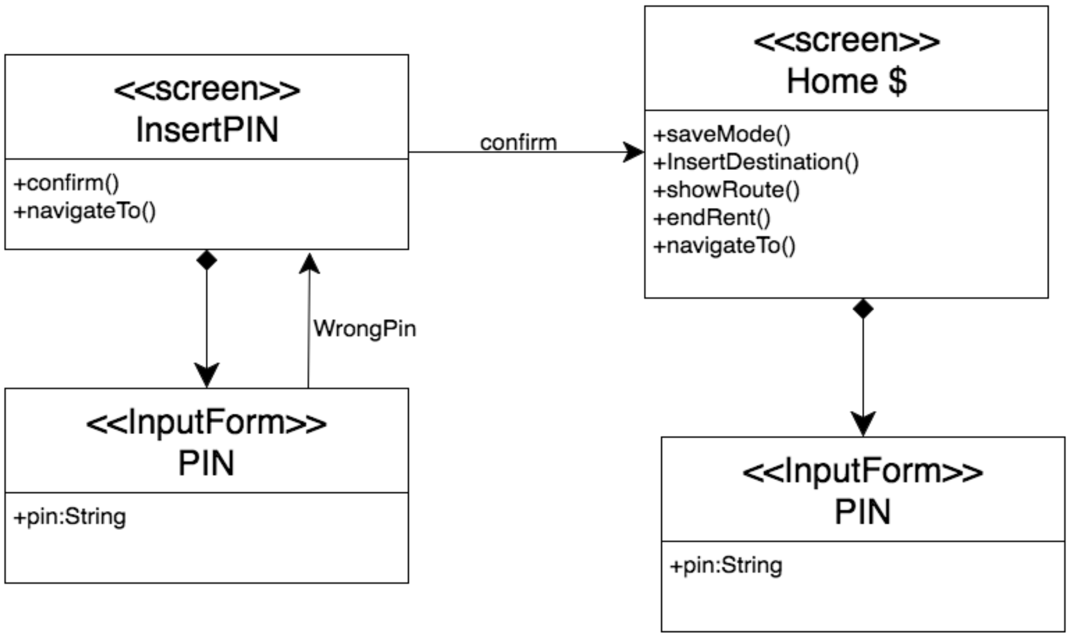


Figure 14: User Experience diagram car.

4.3 BCE Diagrams

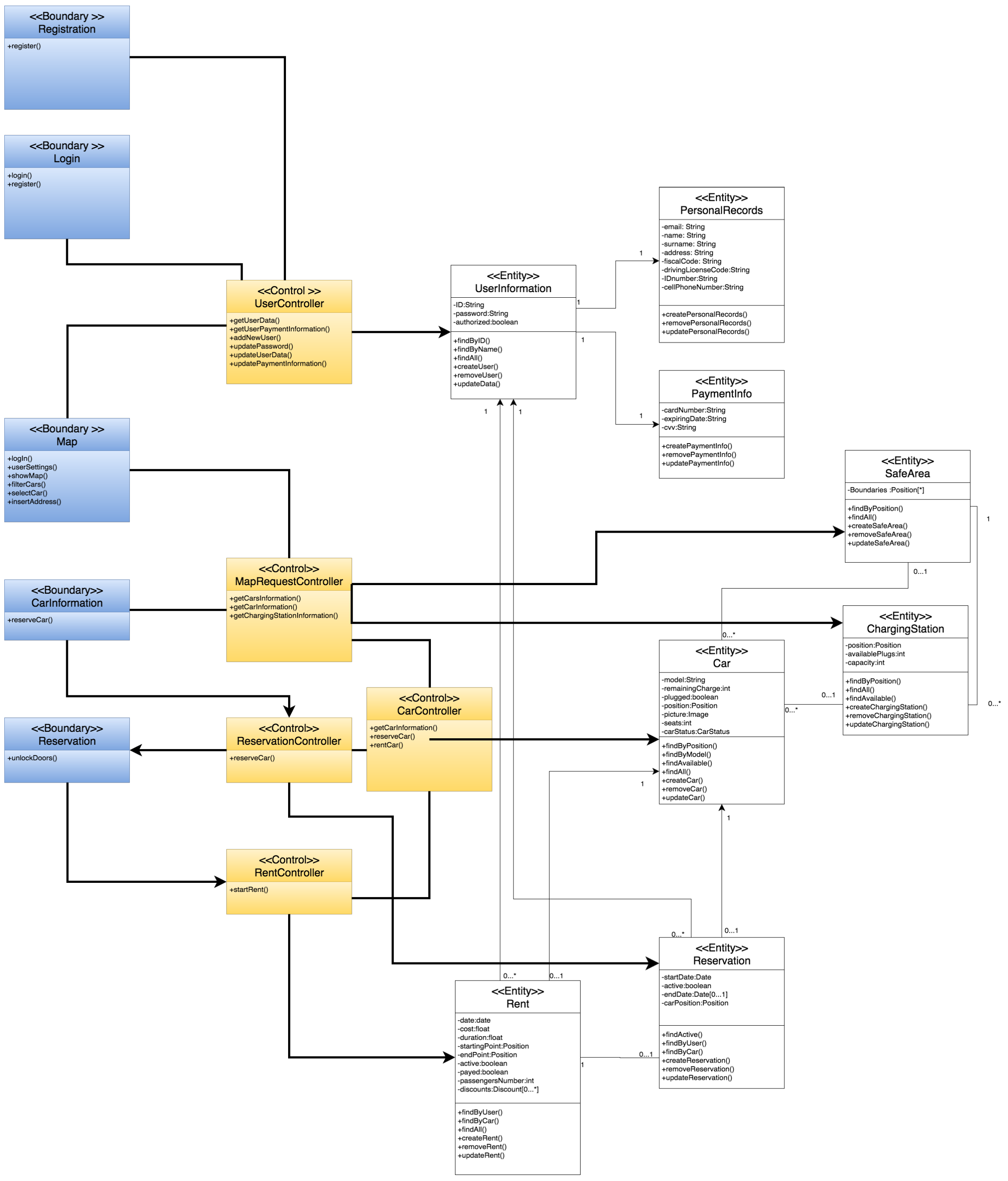


Figure 15: BCE diagram.

5 Requirements traceability

The design decisions explained in this document have been made in order to satisfy all the goals and requirements presented in the RASD. Below, we explain which components of the system have been chosen to satisfy our goals.

G1 A client can register to the system by providing an e-mail, valid payment information and a photo of his driving license.

- ClientApp
- UserController

G2 A client can log in to system.

- ClientApp
- UserController

G3 Allows clients to obtain information about available cars(position and remaining charge), safe areas(boundaries) and charging stations (position and available plugs).

- ClientApp
- MapController

G4 Allows clients to reserve a car that fits most their needs.

- ClientApp
- ReservationController

G5 A client can start the rent opening a car that has reserved previously when he/she is in the near by.

- ClientApp
- RentController

G6 During the rent a client can display the amount of money charged.

- CarRemoteController

G7 Guarantee as many available cars as possible encouraging clients to have a virtuous and eco-friendly behavior applying fees and discounts.

- FareController

G8 Allows clients to end the rent and leave the car in any safe area.

- CarRemoteController
- CarController
- RentController

G9 Clients can report eventual damages made by users that have driven the car before.

- CarRemoteController
- CarController

G10 Clients can set the money saving option.

- CarRemoteController
- CarController
- DistributionOptimizer

6 Effort spent

During the whole project the team worked with the following schedule:

Claudio Salvatore Arcidiacono

- 29 November : 2 hours
- 7 December : 15-18 3 hours
- 8 December : 15-22 5 hours (2hours of pause)
- 9 December : 14-18 4 hours

Total hours: 14 hours.

Antonio Di Bello

- 29 November : 2 hours
- 5 December : 3 hours
- 6 December : 3 hours
- 8 December : 1 hour
- 9 December : 4 hours
- 10 December : 2 hours

Total hours: 15 hours.

Denis Dushi

- 29 November : 2 hours
- 6 December : 3 hours
- 7 December : 3 hours
- 8 December : 6 hours
- 10 December : 3 hours
- 11 December : 1 hours

Total hours: 18 hours.

7 References

7.1 Software and tool used

The tools we used to create this DD document are:

- Overleaf (for latex writing in parallel)
- Photoshop (for mockups)
- Draw.io (for UML diagrams)

List of Figures