Politecnico di Milano

Software engineering 2

PowerEnjoy
Design Document v1.1

Alessandro Caprarelli	874206
Roberta Iero	873513
Giorgio De Luca	875598

February 3, 2017

Contents

Chapter 1

Introduction

1.1 Purpose

This document represents the Design Document (DD). The main purpose is to describe the system in terms of its components, providing a detailed description of the high level architecture, the design patterns we are going to use, the analysis of the internal and external interactions of the components. This document is addressed to developers that have to implement the software.

1.2 Scope

The aim of the project PowerEnJoy is to provide an automated service of car sharing. After a registration, a client can hire a car near him/her through the web or mobile application and he/she can enjoy of all the extra services offered. The exact position of the client is determined by the GPS signal of the client's device or it's allowed to manually insert a specific address. So the system displays all the available cars in the client's close area. Then the client could make a reservation of a car, after which the system notifies the client with a message of confirmation with the car identifier. If the reservation procedure successfully ends the chosen car won't be available anymore for other clients. Moreover a client cannot hire more than one car at the same time. After the reservation the client has at most one hour to reach the car, when this time expires the system gives a penalty to the client and the car, previously hired, is available again for other clients. The system allows the client to cancel his/her reservation. When the client reaches the car, he/she can tell the system that is nearby through a specific button in the application and he/she starts

to pay as soon as the engine ignites. During the travel the system supervises the current charge of the car and notifies it to the client through a screen located in the car. The system stops charging the amount of money that the client has to pay when he/she communicates through the application his/her decision to stop the rent. When the car is parked in a safe area and the client exits, the system locks the car automatically and starts the procedure of payment. The client is notified with the result of this procedure through an SMS, including the final fare.

1.3 Definition, acronyms, abbrevations

1.3.1 Definition

- **Guest client:** a person that is not already registered in the system or that has to log in.
- **Registered client:** a person who has valid access credentials to log in the system.
- **System administrator:** privileged user, in charge of managing administration processes and of updating business logic.
- **Reservation:** it is the action performed by a registered client that allow him/her to reserve an available car for maximum one hour.
- **Journey time = travel time:** time elapsed since the user starts the engine to the user parks the car and terminates the journey.
- **Available car:** a car that is not reserved by any user and has enough charge to be rented.
- **Unavailable car:** a car that is already reserved or damaged, so impossible to reserve.
- **Gps navigation:** it is the navigation system that is included in the car on board system. It could be used by the user to find direction to the final destination.
- **Final destination:** address where the user wants to go.
- **Safe area:** the region where is permitted to park and leave a car once the rent is terminated.
- **Power grid station:** the area where it's allowed users to park the cars, leaving them attached to the power grid.

1.3.2 Acronyms

- **RASD:** Requirements Analysis and Specification Document
- **DD:** Design document
- **API:** Application Programming Interface
- **UI:** User Interface
- **MVC:** Model View Controller
- **URL:** Uniform Resource Locator
- **UXD:** User Experience Design
- **BCE:** Boundary Control Entity
- **SMS:** Short Message Service
- **WSS:** Web Socket Secure
- **HTTPS:** HyperText Transfer Protocol over Secure Socket Layer

1.3.3 Abbrevetations

- **[Gn]:** n-goal
- **[Rn]:** n-functional requirement

1.4 Reference documents

- PowerEnjoy specification document (assignment).
- RASD v1.1 .
- IEEE Std 1016tm-2009 Standard for Information Technology - System Design - Software Design Descriptions.

1.5 Document overview

The document is substantially divided in 5 sections:

- Introduction: it provides a general description of the content of the Design Document and some additional information in order to be coherent with the RASD document, previously produced.
- Architecture Design: in this section there are a detailed increasing description of the components of our application, of their architecture and of their internal and external interactions. This section also explains the main design and architectural choices.
- Algorithms Design: this section contains an explanation of the main algorithms that have to be implemented in the software. Pieces of pseudo-code are included in order to give a clear view of those algorithms.
- User Interface Design: this section includes mockups and user experience explained via UX and BCE diagrams.
- Requirements Traceability: this section relates the decisions taken in the RASD to the design components introduced in this document.

Chapter 2

Architectural design

2.1 Overview

This chapter describes the different components of PowerEnjoy and how they interact with each other, starting from an high level view of the system and finally explaining in details the sub-components.

2.2 High level components and their interaction

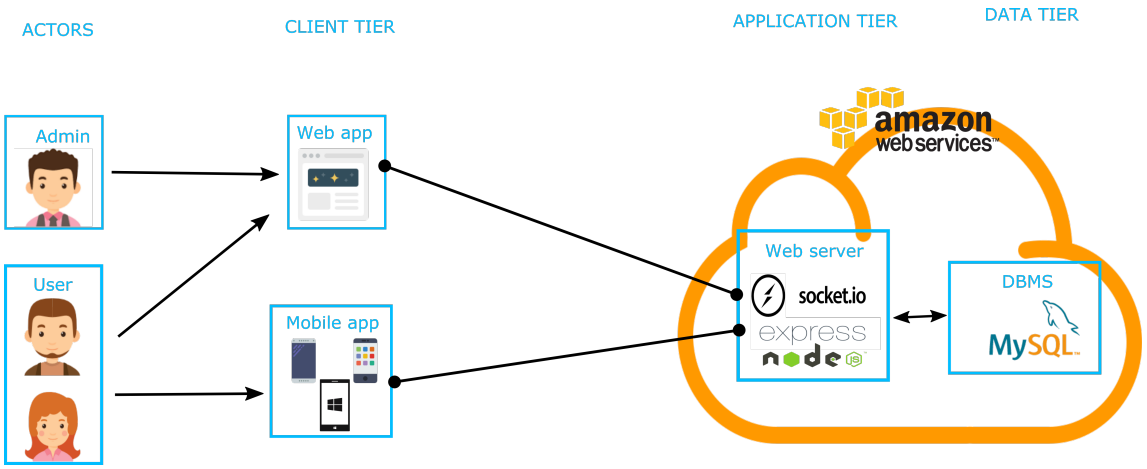

Figure 2.1: High level view of PowerEnjoy

PowerEnjoy is composed by three main components: DBMS, web server and client application. These can be mapped to three tiers in a three-tiers architecture: the data tier is the DBMS, the application tier is the web server and the GUI tier is the client application. The client application provides the UI through which the users can access the service offered by PowerEnjoy.

The UI makes requests to the web server that is in charge of elaborating those and eventually of providing a valid response. The client application is waiting for events from the web server and it updates the UI according to the data received.

The web server, while is processing the requests, can make call to external API and query the database.

2.3 Component view

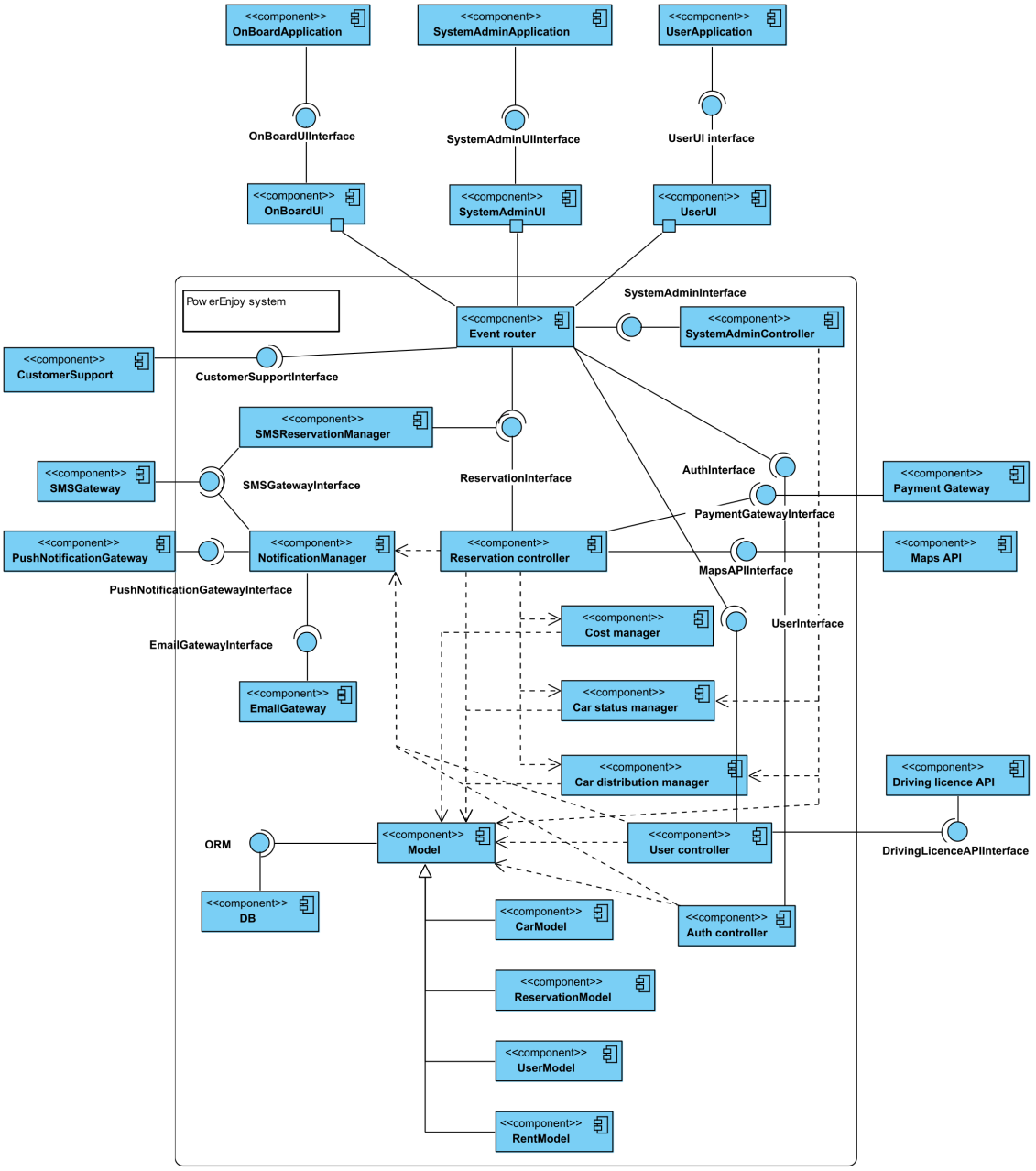

Figure 2.2: Component diagram

The PowerEnjoy system runs a websocket server, as you can see from the high level view diagram, that receives events from the end-user applications.

The EventRouter dispatches the requests to the appropriate Controller that elaborates the request. During this operation the controller can use other components of the system or can make calls to external services.

The ReservationController is the main controller and is in charge of managing all the process that involves a reservation, starting from the submission of a reservation request until the end of the rent. It can be called by either the EventRouter or the SMSReservationManager.

It uses some internal and external components in order to fulfill all the received requests:

- it uses an external PushNotificationGateway in order to send real-time push notifications to users' mobile app.
- it uses an external SMSGateway in order to send SMS notification to users' mobile phone.
- it uses an external PaymentGateway in order to complete a payment after a rent is finished.
- it uses an external MapsAPI in order to locate users and cars into a map. In addition it uses the navigation API, that are part of the Google Maps API, in order to suggest the right journey to the user in case of 'save money' option.
- it uses the internal DiscountManager in order to calculate the discount to apply at the end of a rent.
- it uses the internal CarStatusManager and CarDistributionManager in order to check whether a car is available or not.

The AccountController is in charge of the process of registration and modification of user's profile and settings. It uses an external Driving licence API, provided by the 'Motorizzazione Civile', in order to check if a driving licence is valid or not.

The AuthController is in charge of checking the validity of the credentials during the login process.

All the controllers use instances of Model in order to retrieve data from the DB.

2.4 Deployment view

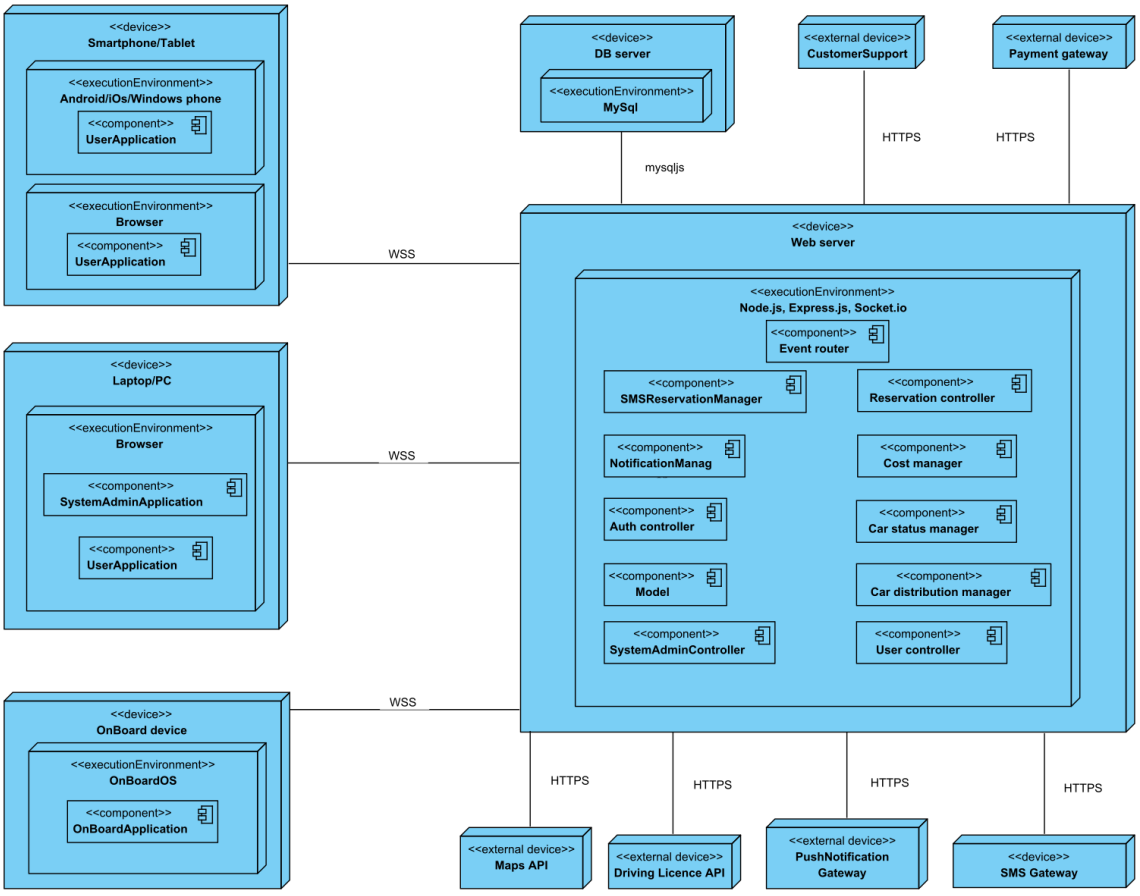

Figure 2.3: Deployment diagram

In this section the components defined in the previous section are placed within a device. The following diagram explains where each component is and how different devices communicate between them.

JSON is used to encapsulate data inside a standard structure and only JSON messages are exchanged between our internal components. This choice makes easier the exchange of messages, even complex objects, among components built with different technologies.

2.5 Runtime view

In this section are presented some sequence diagrams that explain the interactions among components showed in Component and Deployment diagrams.

2.5.1 Login

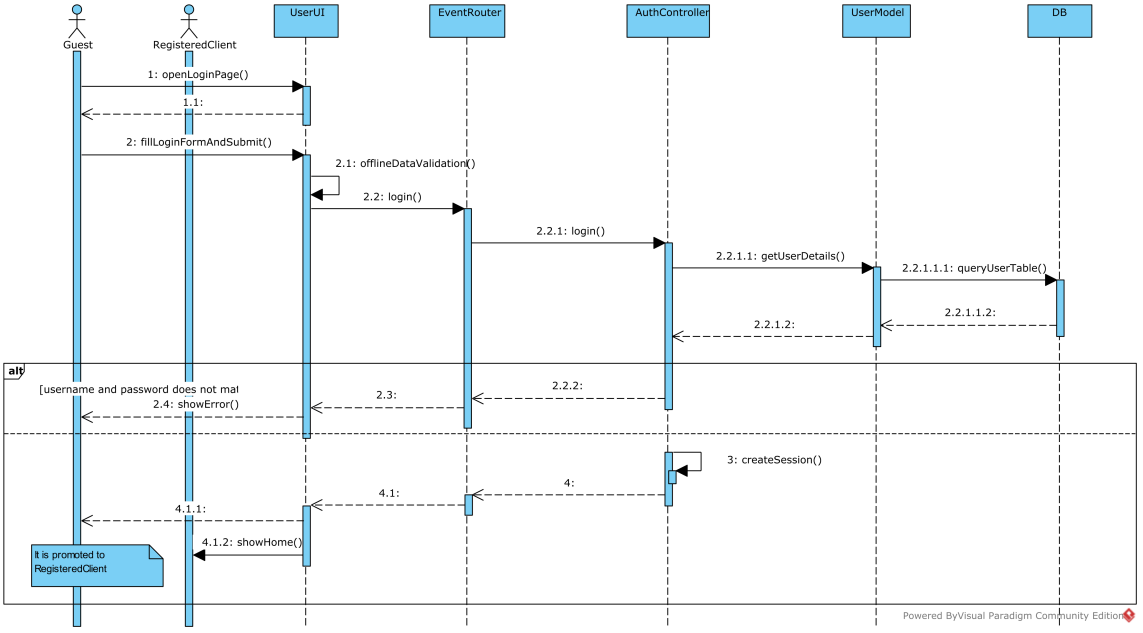

Figure 2.4: Login sequence diagram

2.5.2 Reservation through mobile app

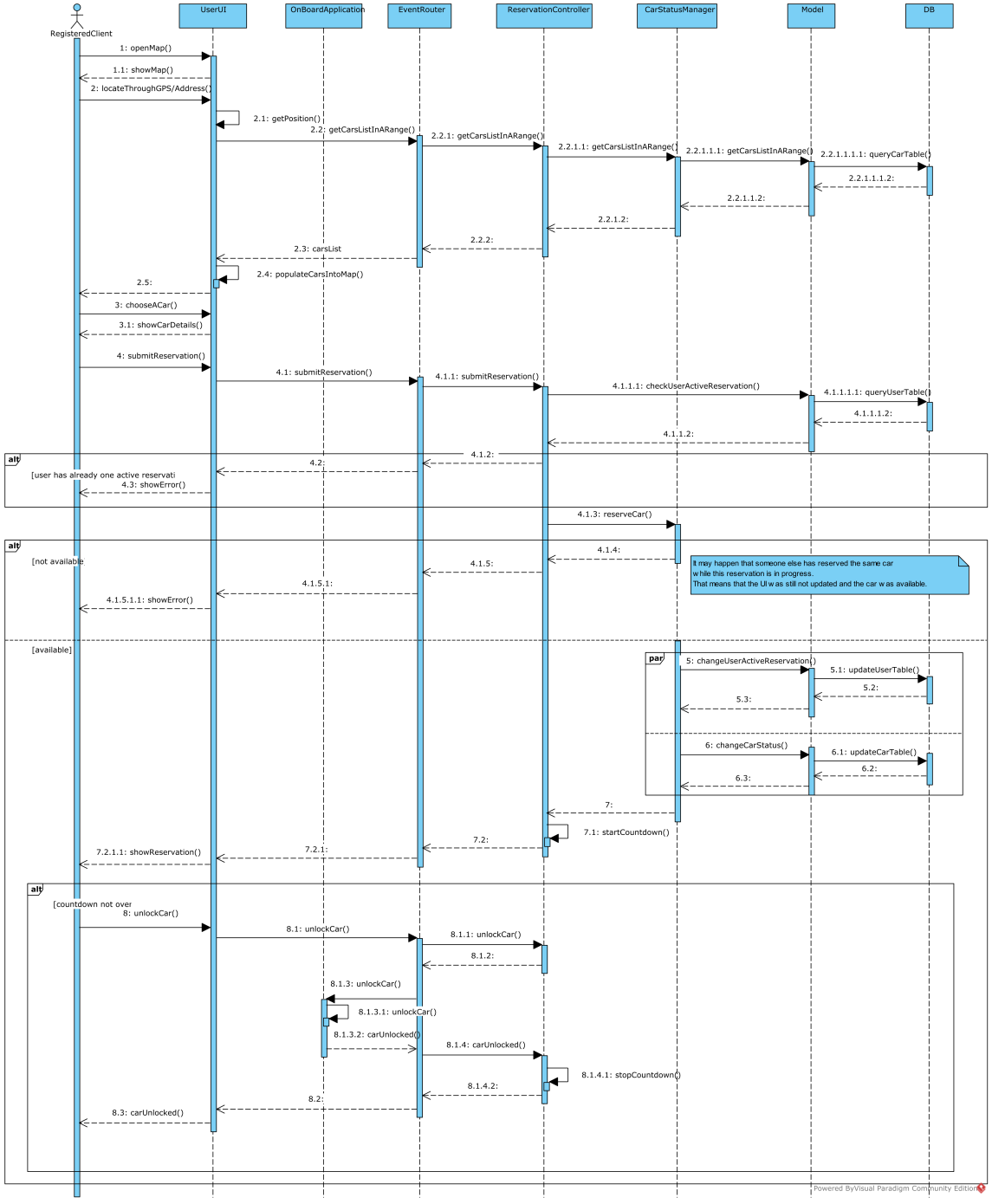

Figure 2.5: Reservation sequence diagram

2.5.3 Usage of on board application

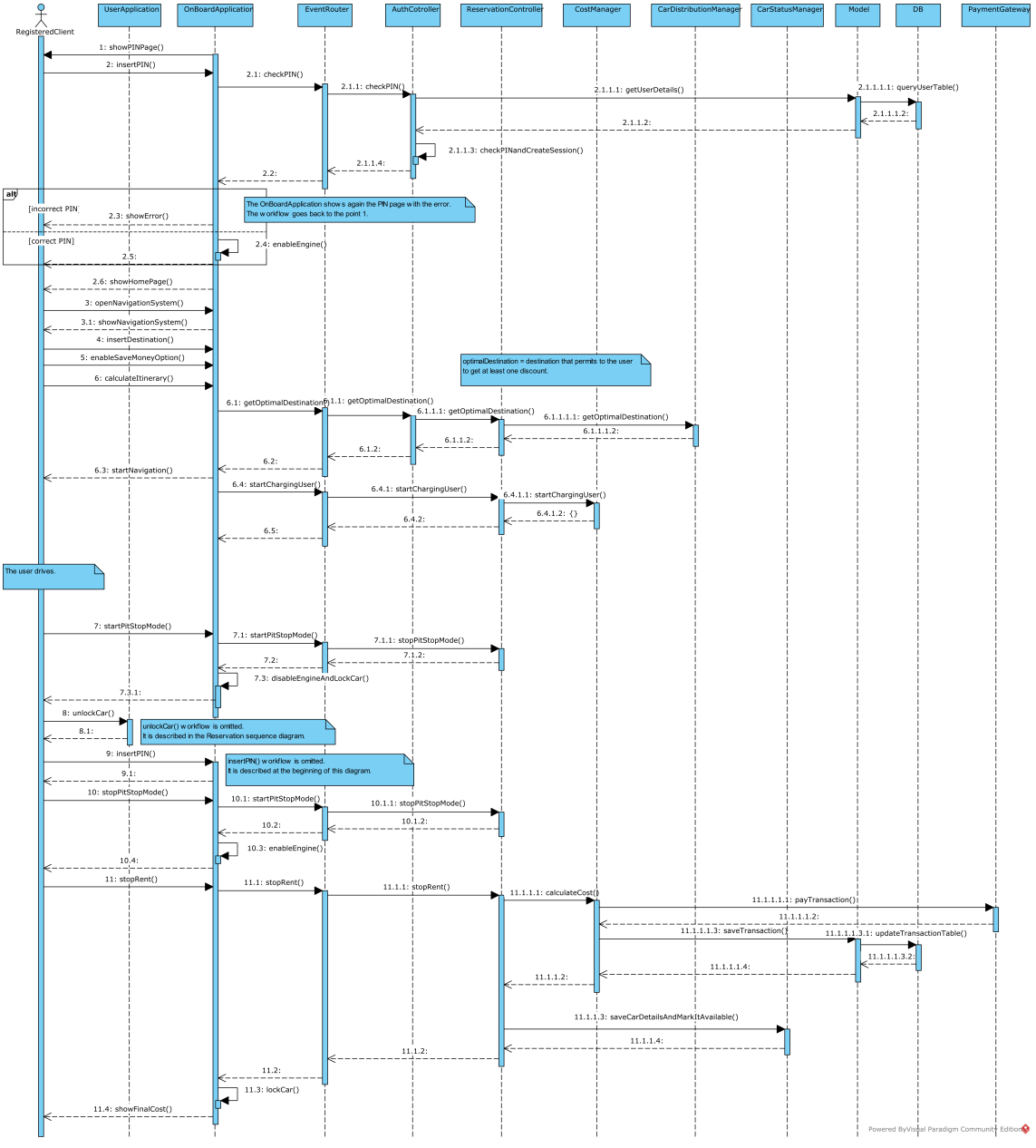

Figure 2.6: Usage of on board application sequence diagram

2.6 Component interfaces

In this section all the interfaces, previously shown in the Component Diagram, are analyzed.

2.6.1 UserUI interface

This interface provides an entry point to the system for Guest users and Registered Clients, through web/mobile application.

The following list contains the main methods provided by this interface:

- showHome()
- showLogin()
- showSingUp()
- showAccount()
- showReservationPage()
- showMap()
- showCarDetails()
- submitRegistration()
- submitLogin()
- submitReservation()

2.6.2 SystemAdminUIInterface

This interface provides an entry point to the system for the System Administratror.

The following list contains the main methods provided by this interface:

- showHomeAdmin()
- showAdminLogin()
- showAdminAccount()
- showMenu()
- showDataEditor()

- submitAdminLogin()

2.6.3 OnBoardUIInterface

This interface provides an entry point for Guest users and Registered Clients, through the on board application.

The following list contains the main methods provided by this interface:

- showEnterPinScreen()
- showOnBoardHome()
- showNavigationSystemHome()
- submitAddress()
- submitSaveMoneyOption()
- submitTerminateRent()
- submitPitStopMode()
- stopPitStopMode()
- submitPin()
- submitAskForHelp()

2.7 Selected architectural styles and patterns

This section exposes the architectural decisions for the development the system of Power Enjoy and the reasons related to each choice.

2.7.1 Overall architecture

Service Oriented Architecture

SOA fits perfectly the needs of the system of Power Enjoy, starting from the calls to external services to the communication between the different parts of the data and business models, the involved devices and each architectural layer.

Client - server architecture

PowerEnJoy is an on-demand service usable by the client through website, smartphone app and SMS. The architecture that best complies with this context is the server-client one: besides all benefits of the communication between the devices, it's the most suitable architecture also for its maintainability and scalability. The interfacing with different types of devices is completely horizontal to the server side, permitting an easier implementation and management of the workflow of the system.

2.7.2 Design patterns

MVC

The Model-View-Controller paradigm is largely used in systems where a data model interacts with different clients, like PowerEnJoy: separating all components with respect to their aim permits to developers a better management of interactions between them.

Furthermore it is possible to map the 3 tier architecture with the MVC. The application and data tiers represent the controller and model components while the client tier represents the view component.

Singleton

The singleton patterns restrict the number of instantiations of a class to one. Creating singletons in Node.js is pretty straightforward using 'require'. It does not matter how many times you will require this module in your application; it will only exist as a single instance.

Callback

Callbacks are the essence of Node.js. They enable a balanced, non-blocking flow of asynchronous control across modules and applications.

Node.js relies on asynchronous code to stay fast.

Middleware

Middlewares are the core idea of Express.js. The concept is poweful and simple: the output of one function is the input for the next.

Chapter 3

Algorithm design

In this chapter are presented some solutions for the main critical parts of the system.

3.1 getCarsListInARange

```
Data: w: the user requesting reservation
Result: A list of car available in a range
Car[] availableCars;
Position pos ← w.getPosition();
if pos!=null then
    int range ← w.getRange();
    foreach c in CarStatusManager.getCarsInRange() do
        if c.rentable() then
            availableCars.add(c);
        end
    end
end
return availableCars;
```

Algorithm 1: Function getCarsListInARange

3.2 submitReservation

Data: w: the user requesting reservation
c: the car choosen by w
Result: true if reservation has been performed successfully, otherwise false
if *!w.hasAlreadyActiveReservation()* **then**
  if *c!=null* **then**
    if *CarStatusManager.reserveCar(c)* **then**
      Request r;
      startCountdown();
      r.user ← w;
      r.car ← c;
      return true;
    end
  end
else
  return false;
end

Algorithm 2: Function submitReservation

3.3 calculateDiscount

Data: w: the user that used a car
r: the rent instance, just concluded, related to u
Result: a percentage of the discount
float discount ← 0;
Car c = r.getReservation().getCar();
if *c.getBatteryStatus()>50* **then**
  discount ← discount + 0,20;
else
  Position posCar = c.getPosition();
  if *APIMaps.getDistanceFromNearestSA(posCar)>3 and c.getBatteryStatus()* *<= 0,20* **then**
    discount ← discount - 0,30;
  end
end
if *c.getPeopleLastRide()>1* **then**
  discount ← discount + 0,10;
end
if *c.isPluggedIn()* **then**
  discount ← discount + 0,30
end
return discount;

Algorithm 3: Function calculateDiscount

Chapter 4

User interfaces design

4.1 Mockups

Some mockups have been already included in the RASD document of PowerEnJoy, in section 3.1.1. In addition to them we provide other mockups to show other functionalities of the application.

4.1.1 Activation of the car

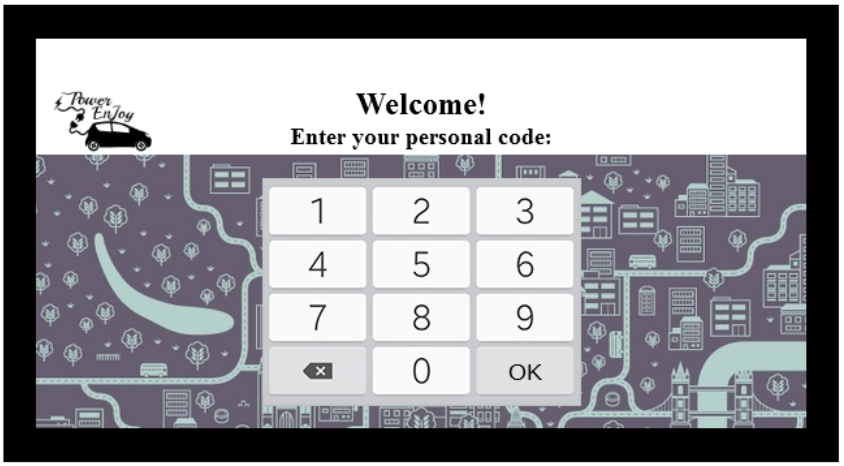

Figure 4.1: Activation of the car mockup

Once in the car the Client has to insert his/her personal code in order to activate the car. If she/he doesn't insert the correct pin he/she won't be able to ignite the engine.

4.1.2 Additional functionalities

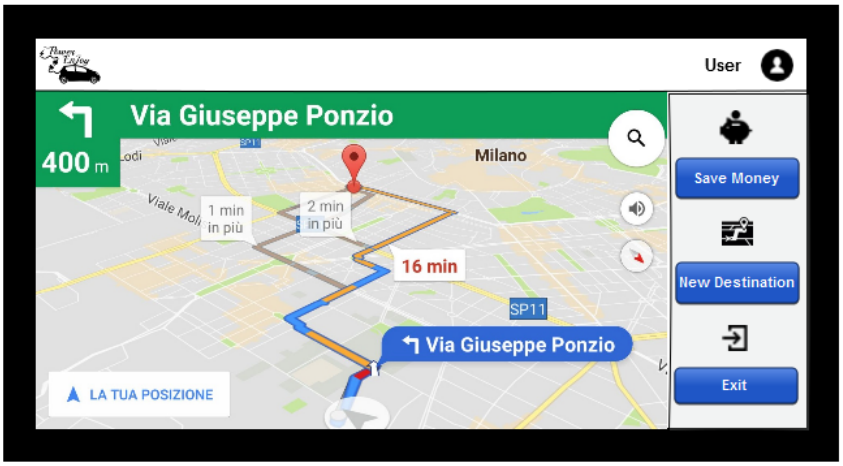

Figure 4.2: Additional functionalities mockup

When the engine ignites, the GPS navigation device offers the Client the opportunity to use the navigation system and in addition to enable the save money option, in order to obtain a discount.

4.1.3 Terminate rent

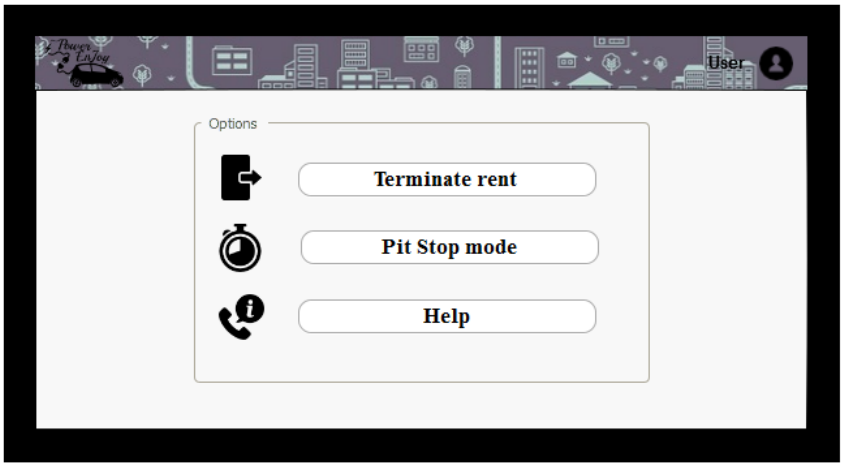

Figure 4.3: Terminate rent mockup

The GPS navigation device shows as default this screen, allowing the Client to terminate the rent or to choose the 'Pit Stop' mode. If the Client is using the navigation system, he/she can reach this screen with a click on the exit button(See 4.1.2).

4.2 UX diagrams

The following UX diagram describes paths that a user can follow. Starting from the homepage of the application (both web and mobile application), a user can choose to do a registration or to login into the system, if he/she has already valid access credentials. In both cases if an error occurs, for example due to mistaken information, there is an error screen that report to the user that something goes wrong and redirect him/her to a safe screen to redo all the operations. Once a Registered Client is logged into the system he/she can reserve a car, choosing the address in which he/she wants to look for a car or deciding to be located through the GPS signal of the device. Then the Registered Client is directed to the Map screen in which he/she can choose a car to rent and once the decision is made he/she return to the Reservation screen. Then he/she can check the Show Reservation screen, in which there are a recap of the reservation and the possibility to cancel it.

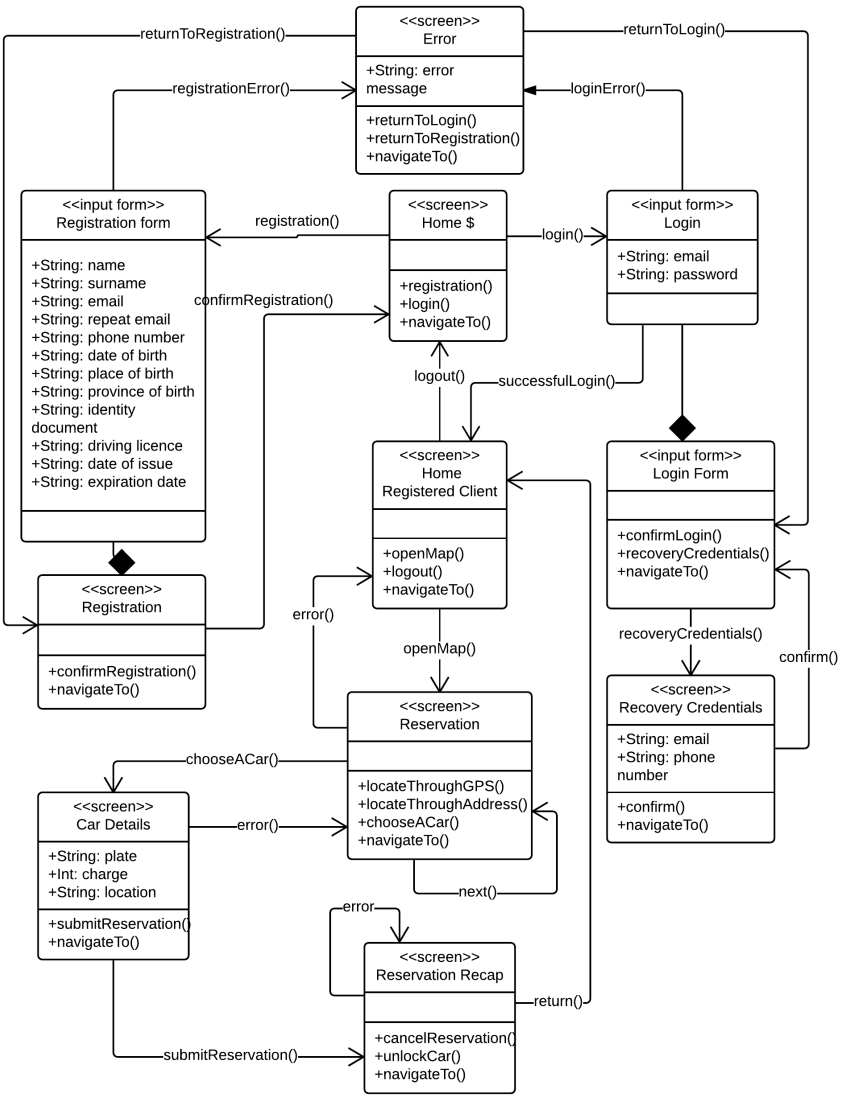

Figure 4.4: UX diagram

4.3 BCE diagrams

The following diagram represents the BCE diagram and shows how a user action, performed on its UI, is managed within the component of our system.

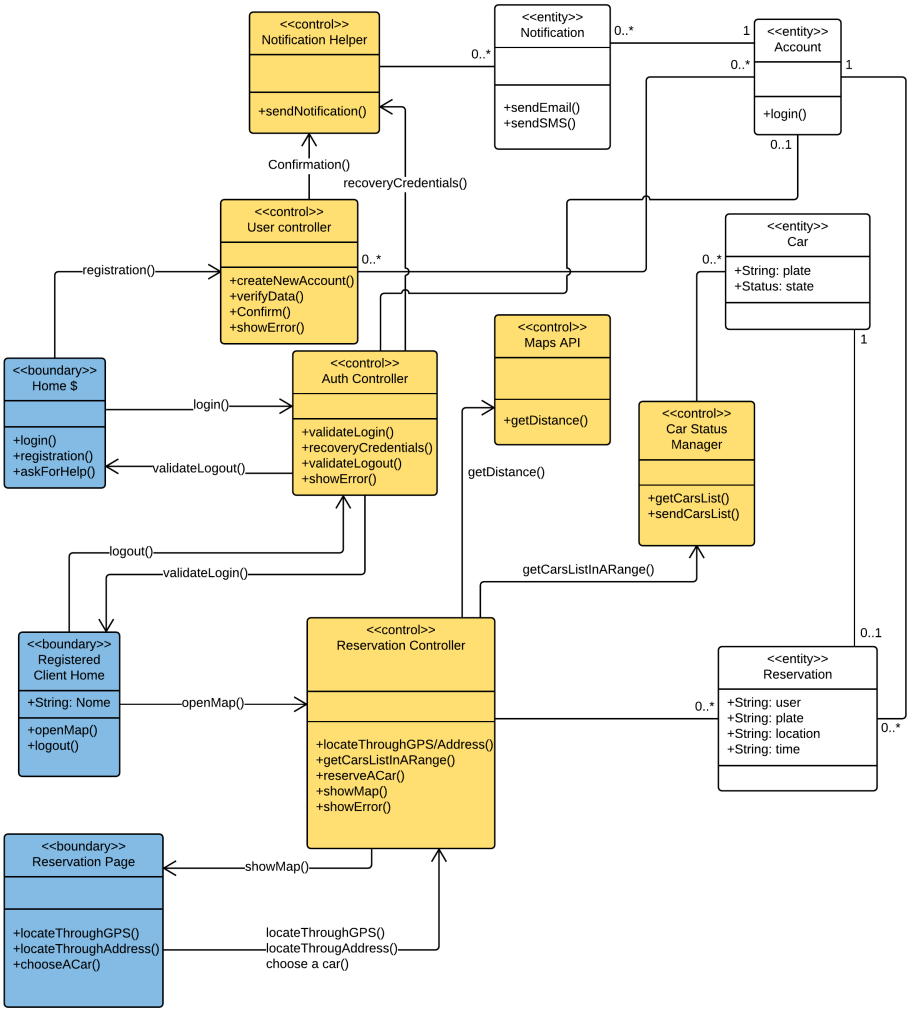

Figure 4.5: BCE diagram

Chapter 5

Requirements traceability

The following table maps the requirements identified in the RASD document (section 3.2) to the design decisions taken in this DD.

Requirements	Design decisions
[**G1**] - The Registered Client can hire a car through web/mobile application	• UserApplication • ReservationController • CarStatusManager • OnBoardApplication • CostManager • PaymentGateway • CarDistributionManager • AuthController
[**G2**] - The Registered Client can hire a car through an SMS.	• SMS ReservationManager • ReservationController • CarStatusManager • OnBoardApplication • CostManager • PaymentGateway • AuthController • CarDistributionManager
[**G3**] - The Guest Client can register himself/herself into the system as Registered Client.	• UserApplication • UserController • NotificationManager • PaymentGateway

[**G4**] - Registered Clients can login into the system.	• UserApplication • AuthController • UserController • NotificationManager
[**G5**] - The Logged Client can manage his/her sensible data.	• UserApplication • UserController • NotificationManager
[**G6**] - The Registered Client can manage his/her requests of hiring.	• UserApplication • UserController • ReservationController • SMSReservationManager • CarStatusManager • NotificationManager
[**G7**] - Ensure the Registered Client the possibility to receive a discount on his/her last ride.	• OnBoardApplication • CostManager • ReservationController • CarDistributionManager • CarStatusManager
[**G8**] - Ensure a uniform distribution of cars in the city.	• OnBoardApplication • CarDistributionManager • ReservationController
[**G9**] - Guarantee to have always a minimum number of cars in the system with enough charge to be hired.	• SystemAdminApplication • CarStatusManager • SystemAdminController
[**G10**] - Allow the system administrator to update and check data in the database of the system.	• SystemAdminApplication • SystemAdminController • AuthController
[**G11**] - Guarantee a correct interoperability of the system with external services.	• SMSGatewayInterface • PushNotificationGatewayInterface • PaymentGatewayInterface • MapsAPIInterface • DrivingLicenceAPIInterface
[**G12**] - Ensure that a Registered Client's bad behaviour is punished with the application of some penalties.	• ReservationController • CostManager • OnBoardApplication • CarDistributionManage

Chapter 6

Used tools and working hours

6.1 Used tools

The tools used to create this document are the following:

- MikTex 2.9: to format the document using LaTex.
- Pencil: to create mookups.
- Visual paradigm: to create componet, deployment and sequence diagrams.
- LucidChart: to create UX and BCE diagrams.

6.2 Working hours

	Alessandro	Roberta	Giorgio
29/11	0.5	0.5	0.5
30/11	2	4	2
1/12	2	4	2
2/12	3	3	3
3/12	2		2
4/12	3		2.5
5/12	3	3	3
6/12	6	6	3
7/12		1	2
8/12		1	1
9/12	3	3	3
10/12	4	1	1
11/12			
Total	28.5	26.5	25

Changelog

v1.1

- Fixed typos in in component and deployment diagrams.