POWER ENJOY

DD

Design Document

Version: 1.0.2

11th December 2016

Edited by:

Marzia Degiorgi - 878360
Giulia Leonardi - 877491
Valentina Ionata - 808678

Index

Index **2**

1. Introduction **4**

1.1. Purpose 4

1.2 Scope 4

1.3. Definitions, Acronyms, Abbreviations 4

1.3.1 Acronyms 4

1.3.2 Abbreviations 5

1.4. Document Structure 5

2. Architectural Design **6**

2.1. Overview 6

2.1.1 Mobile Overview 6

2.1.1 On-Board Overview 7

2.2 Component View 8

2.3. Deployment View 11

2.4. Runtime View 13

2.4.1 User Log In 13

2.4.2 Make a reservation 14

2.4.3 End a ride selecting Money Saving 16

2.4.4 Notify an operator of a critical situation 18

2.5. Component Interfaces 19

2.6. Selected Architectural Style and Patterns 19

2.6.1 Architectural Style 20

2.6.2 Architectural Patterns 21

2.7. Other Design Decisions 21

3. Algorithm Design **21**

3.1 Find a car 22

3.2 Reserve a Car 22

3.3 Calculate final charge 23

4. User Interface Design **24**

4.1 Mockups 24

4.1.1 User Reservation interfaces 24

4.2 UX diagrams 25

4.2.1 UX: Sign Up and Login Interfaces 26

4.2.2 UX: User and the mobile interface 27

4.2.3 UX: Operator and the mobile interface 28

4.2.4 UX: User and the on-board interface 29
4.3 BCE diagrams 30
4.3.1 BCE: User mobile application 30
4.3.2 BCE: Operator mobile application 31
4.3.3 BCE: On-Board application 31
5. Requirements Traceability 32
6. Effort Spent 39
6.1 Giulia Leonardi 39
6.2 Marzia Degiorgi 39
6.3 Valentina Ionata 39
7. References 40
8. Revision History 40

1. Introduction

1.1. Purpose

The purpose of the “Design Document” is to provide a description of the structural design and the architectural decisions of the “Power Enjoy” system, fully enough to allow an understanding of what is to be built and how is expected to be built .
This document is addressed to all developers and programmers who have to implement the actual software.

1.2 Scope

The aim of the project is to create an efficient platform for car sharing, with additional prerogatives respect the already existing ones. Thanks to bluetooth connection, unlock the car will result easier and faster; the “start and stop” button, settled in each car of the “Power Enjoy” company, allows to avoid the use of the key; the QR Code, which links each passenger with his reservation, permits not to lose time with errors of reading or typing: just placing your own QR Code near the cam of the on-board device, the system will recognize and authenticate you. All the car sharing basic functionalities are guaranteed: registration of a new user and consequent login, localization of the cars available and reservation of the chosen one.

1.3. Definitions, Acronyms, Abbreviations

1.3.1 Acronyms

- OS : Operating System
- UI: User Interface
- API: Application Programming Interface
- REST: Representational State Transfer
- DB: Database
- JSON: JavaScript Object Notation
- OTC: One Time Code
- HTTP: HyperText Transfer Protocol
- MVC: Model View Control
- GUI: Graphical User Interface

1.3.2 Abbreviations

Abbreviation	*Definition*
G[i]	Identifier of the goal ‘i’
DD[i]	Identifier of the design ‘i’
FR[i]	Identifier of the functional requirement ‘i’
NFR[i]	Identifier of the non functional requirement ‘i’
NDD[i]	Identifier of the design associated to a non functional requirement ‘i’

1.4. Document Structure

This document is essentially structured in six parts:

1. In this **first section**, we provide a general introduction about the purpose and the scope of the document .
2. The **second section** includes the full design description in detail through UML diagrams.
3. In the **third section**, there is the algorithm design where we describes different situations through the pseudocode.
4. In the **fourth section,** we shows the User interface through mockups, UX and BCE diagrams.
5. In the **fifth section**, we match the design choices with the requirement presented in the RASD document.
6. Finally in the **sixth section,** we presented the team effort and references.

2. Architectural Design

2.1. Overview

We chose for our independent “Power Enjoy” System a three-tier architecture, like show below:

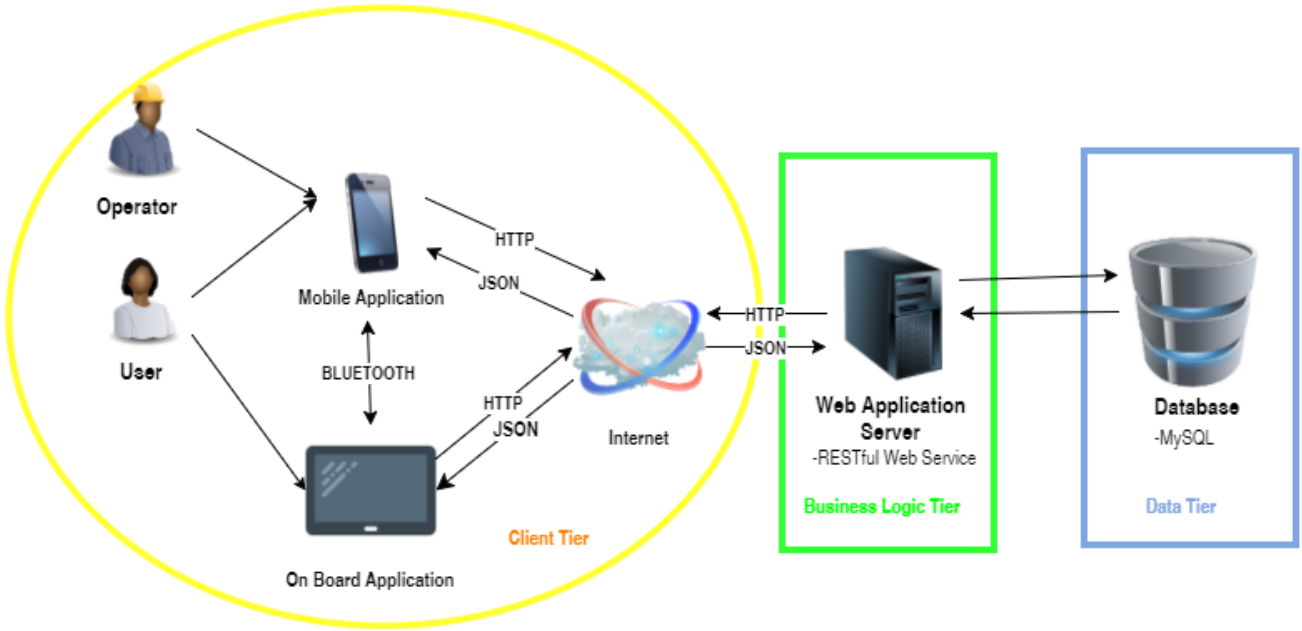

Figure 0: General Design Architecture of the System

2.1.1 Mobile Overview

The best choice for our mobile application is a native approach. The choice about the native application, instead of a mobile web app, is due to the need to interface with the device’s native features, information and hardware, as the bluetooth connection, or the GPS one. As the app is basically native, each mobile application development platform will have its own native programming language: Java for Android, Objective-C for iOS and Visual C++ for Windows Mobile. The RESTful API usage allows us to build the application on various platforms and to simplify the architectural design structure, so that each element should not have its own business logic.

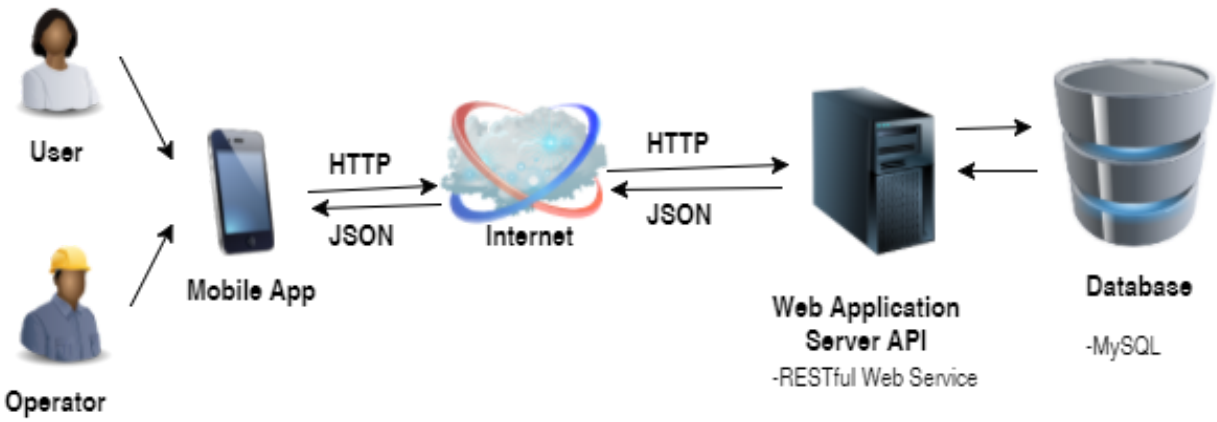

Figure 1: General Architecture for mobile application

2.1.1 On-Board Overview

In the following architectural Design we present the interaction of the devices with the Power Enjoy System.
The User communicates with the system through a Wifi/3G connection to use all the mobile application services as described in the Figure 1. Then, to unlock the car it will be used a Bluetooth channel that, as described in the RASD, allows a simple communication to unlock the car and to acquire User information.
The On-Board device uses a Wifi/3G channel to share vehicle data such as battery, diagnostic information, trip data, position. The same channel is used by the system to upload new software, updating and adding a new service on the car's device.

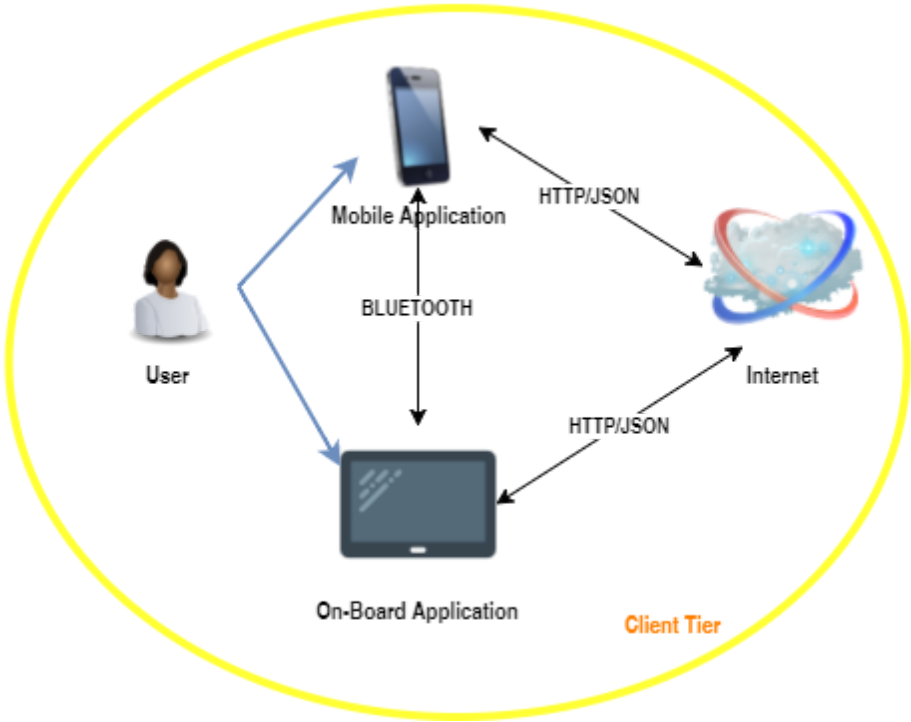

Figure 2: General Architecture for the interaction with On-Board Device

2.2 Component View

In this section, we focused on the physical components of Power Enjoy independent system, following the overview division.

The Power Enjoy Server is the main component of the System, it receives requests of services from the client side (On-Board/Mobile application), so the web Server elaborates the HTTP request and address the message to the application server that provides a response in JSON accessing the database informations.

The components like notification manager and payment manager such as Push request for Operators, email confirmation for users, or payments can be asynchronous. All the other services requires synchronization between the client and server.

The component design structure is showed in detail below:

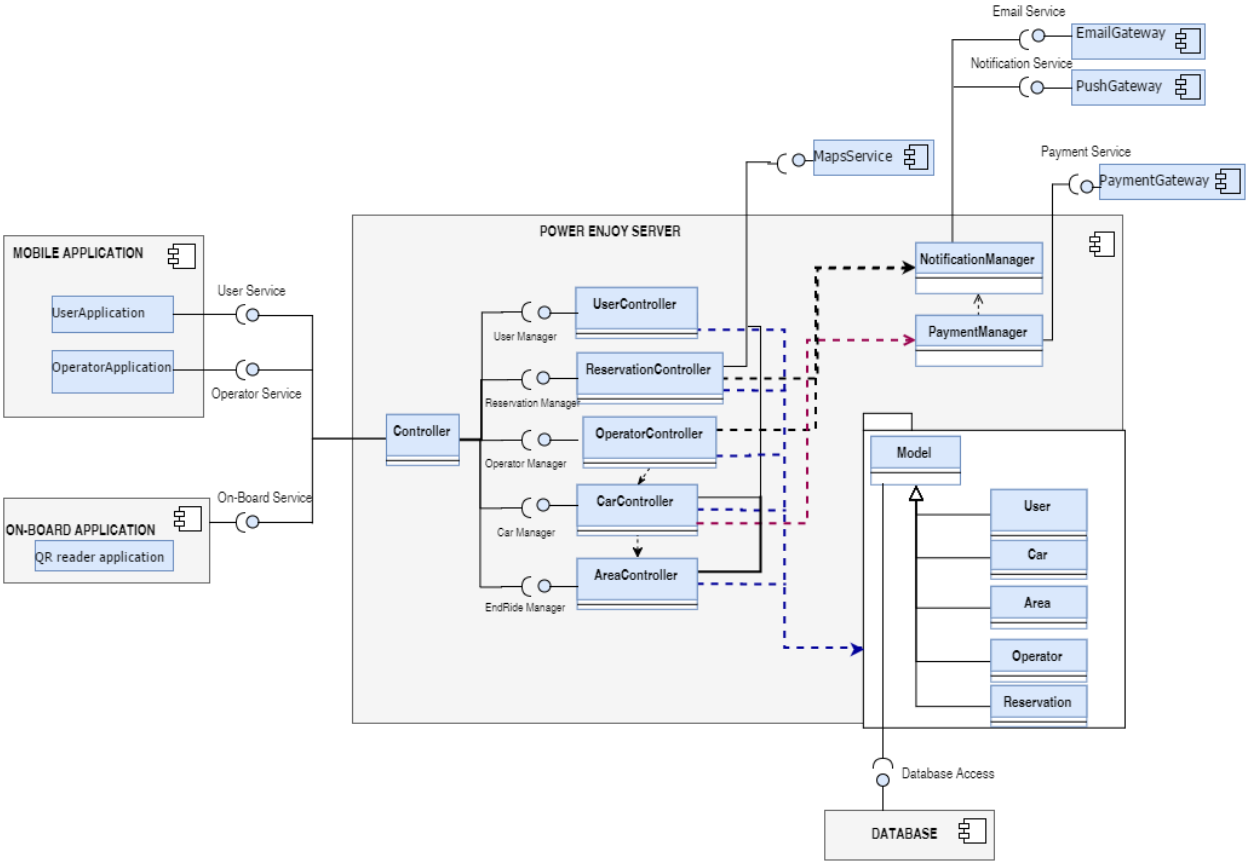

Figure 3: Component View Diagram

Database: it is used to store all datas such as credentials, reservation informations and so on.
Controller: manage the incoming service and addresses the request to the correct controller.
User Controller: check all information about User during the access,check the requesting of a service. e.g modifying profile information, payments and also the credential to register in the system.
Reservation Controller: manage the reservation proper controls such as reservation updating, available vehicles.
Car Controller:it is the On board controller that monitor the following states of the car:

- Availability of cars.
- End of the ride.
- Used options on the cars(money saving, discounts, charges).
- Emergency situation(low battery,damages,parked in an outside area).

Operator Controller: manage the access of operator and notify them if there are problem with a car in their area.
Area Controller: manage the position of cars and parking areas for money saving option.
Notification Manager: manage the notification to users and operators, and also payments.
Email gateway: send email to Users for any kind of advertisement.
Maps service: external service used for Maps.

Push gateway: send push notification in the operator application or User application.
Payment gateway: apply payment to Users.
Model: representation of the world linked to the datas.

2.3. Deployment View

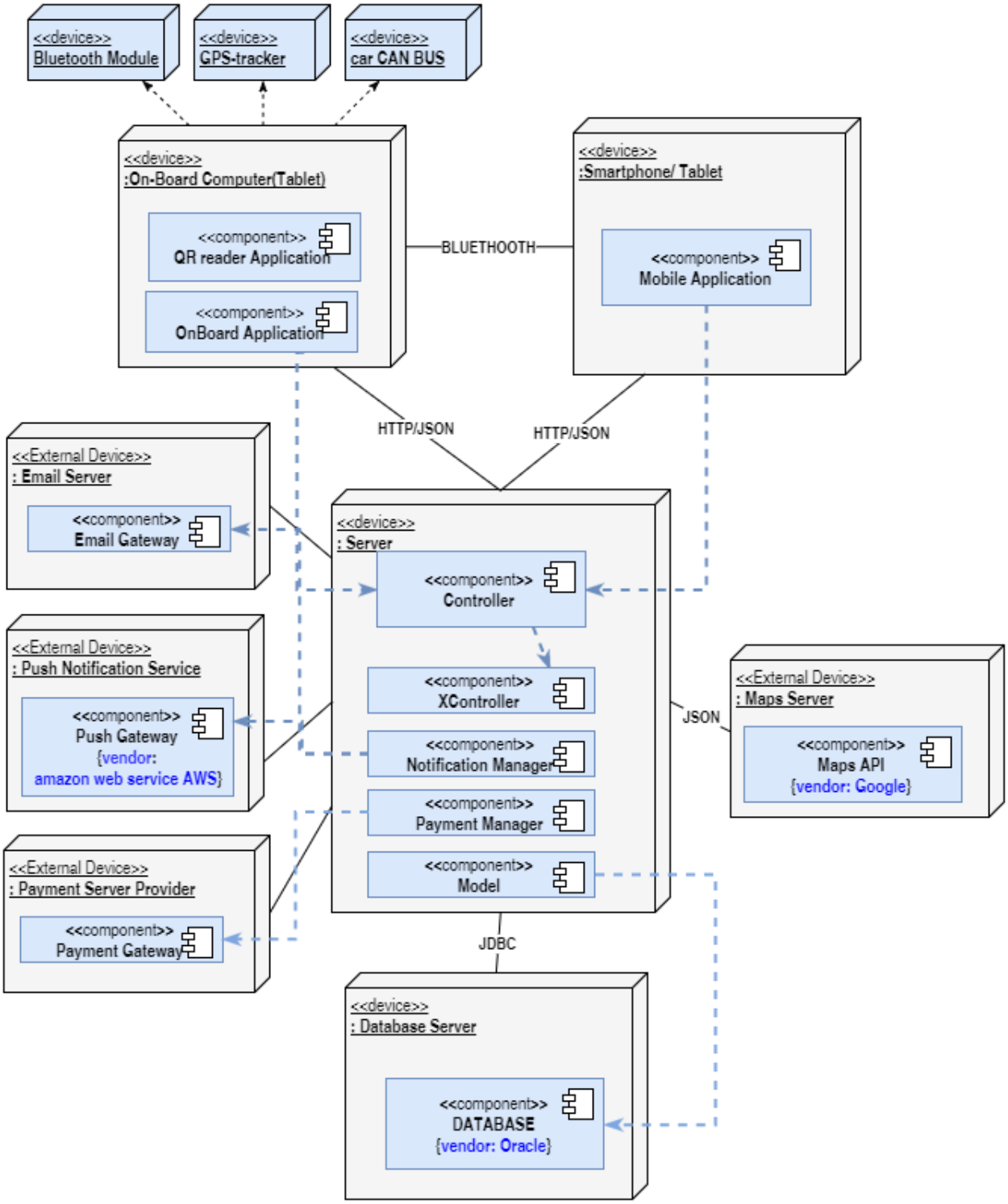

Figure 4: Deployment View Diagram

The **CAN BUS device** for vehicles allows a real time control between all the smart electronic devices in the car and allows also their communication. Then, this device permits to collect datas about the car state. It is useful for monitoring the parameters of cars.
The **X Controller** in the deploy view are all the controller component specified in the previous diagram.

2.4. Runtime View

2.4.1 User Log In

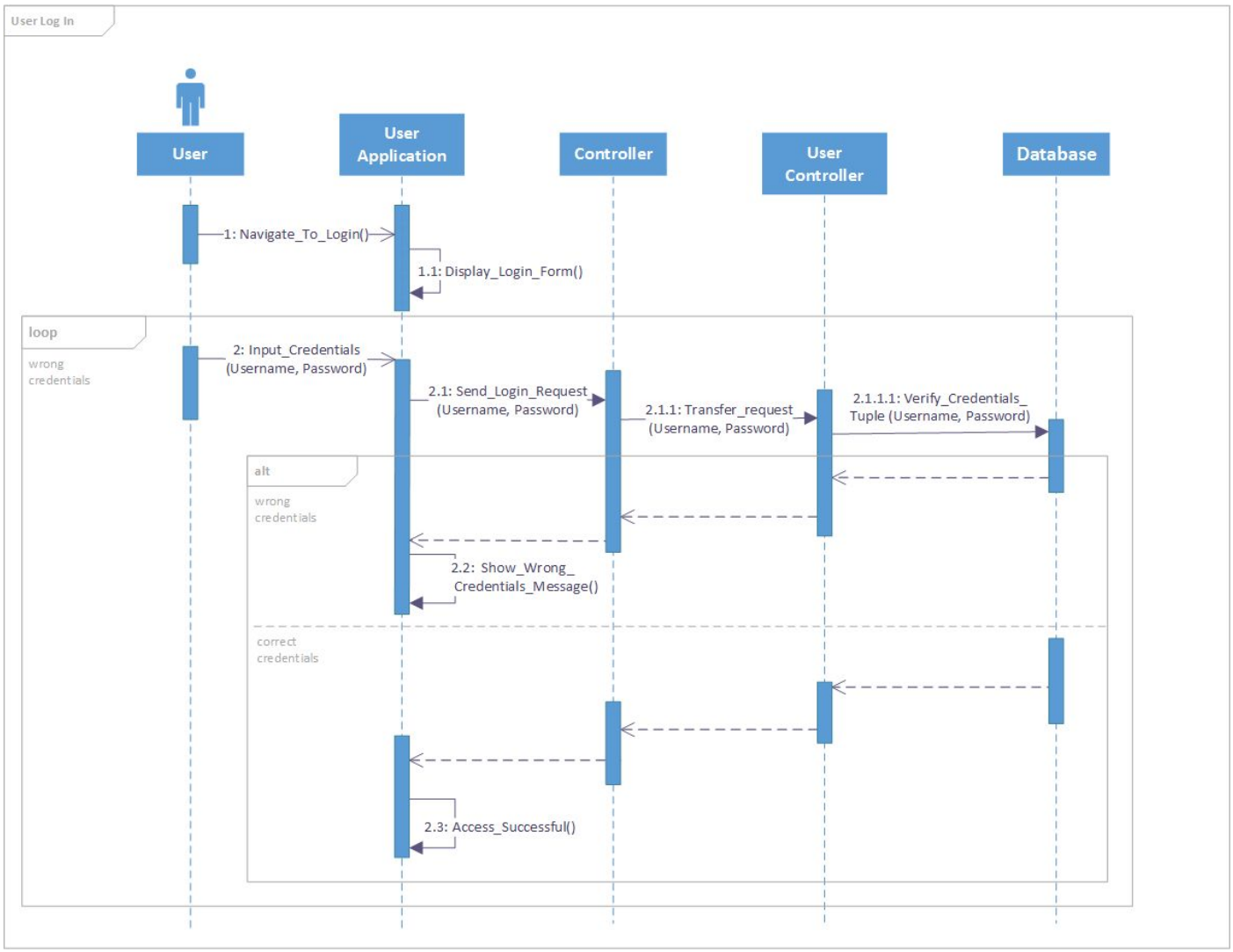

Figure 5: User Log In

This sequence diagram describes how a User (already registered in the system) can log in the Power EnJoy application. When the User inputs his credentials, a login request is sent to the Controller that takes in charge of forwarding this to the UserController. The system checks the correctness of the informations comparing them with database tuples: if the credentials are wrong, an error message is displayed and the User has to insert his credentials again, otherwise he can correctly log in the application.

2.4.2 Make a reservation

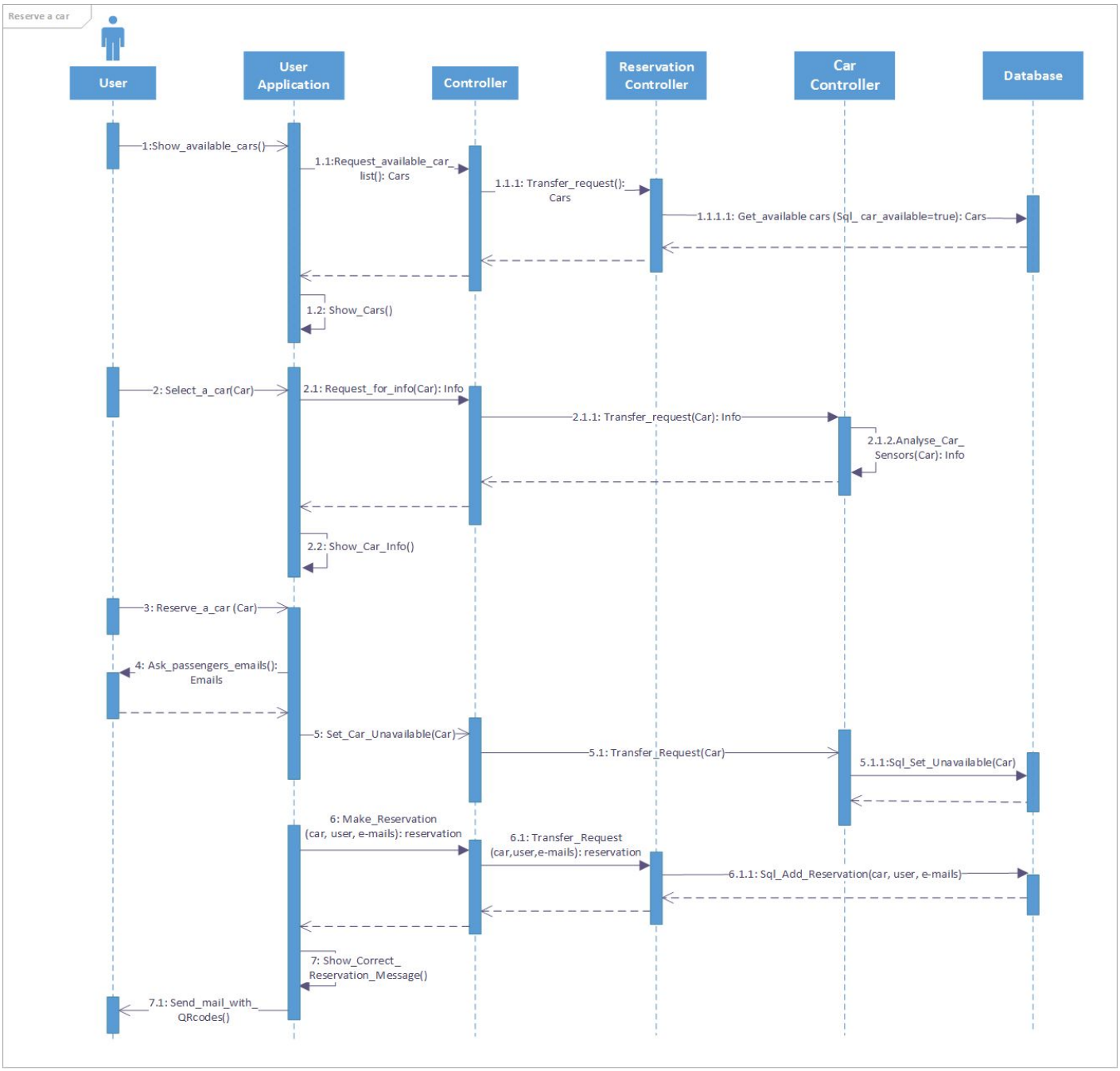

Figure 6: Reserve a car

This sequence shows how a User can reserve a car choosing between the available cars in the map. First of all the User chooses to see all the available cars near his position, next the request is sent to the controller that forwards it to the reservation control, that takes care to ask the database to select only tuples with available cars. After that, the system displays the map with all available cars, after accessing the Map Service. The User can select a car in the map so that the system can show him the car informations, after having analysed the sensors. After all the User can decide to reserve a car, so the System ask him to insert the email of all passengers to complete the reservation. The system deals with changing the state of the car to unavailable and memorizing the reservation in the database (including the user, the car and the emails of passengers). Finally the System sends to the User an email with the QR codes.

2.4.3 End a ride selecting Money Saving

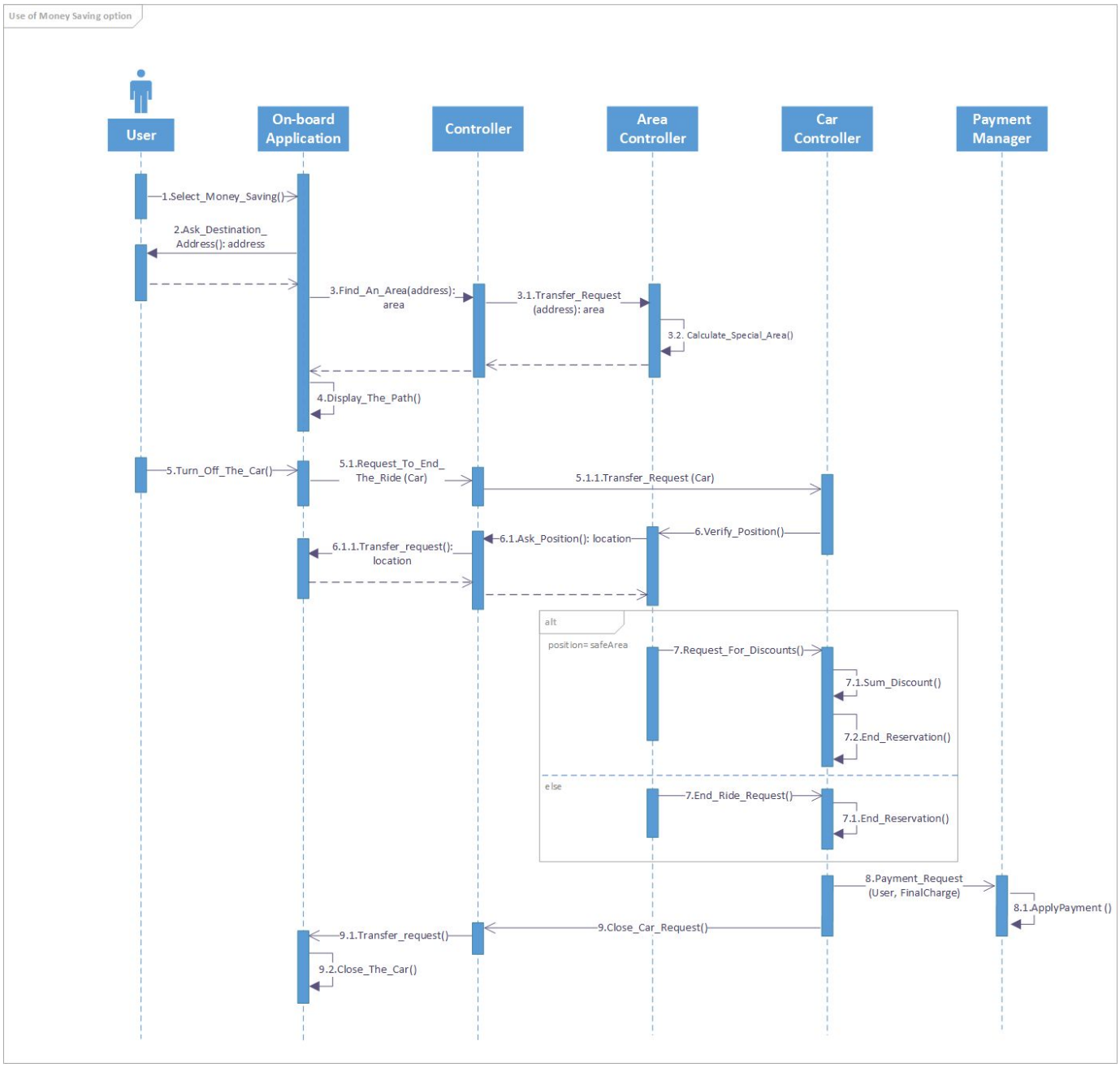

Figure 7: Use of Money Saving Option

In this sequence we can see how a User can use the Money Saving option. If the User selects this option, the mobile app asks him to input the destination address: in this way the Area controller can use it to calculate the special safe area; than this controller access the Map Service and returns a map with the path to the destination to the mobile application, so that it can show that to the User. When the User selects to end the ride, a request is sent to the Car Controller that asks the position to the sensors of the car just to analyse if the car is parked in the selected safe area. If the position and the location of the special area are the same, a discount is added to the price of the ride, otherwise the current reservation is simply closed. Finally the car controller applies the payment through the payment manager and the car is closed.

2.4.4 Notify an operator of a critical situation

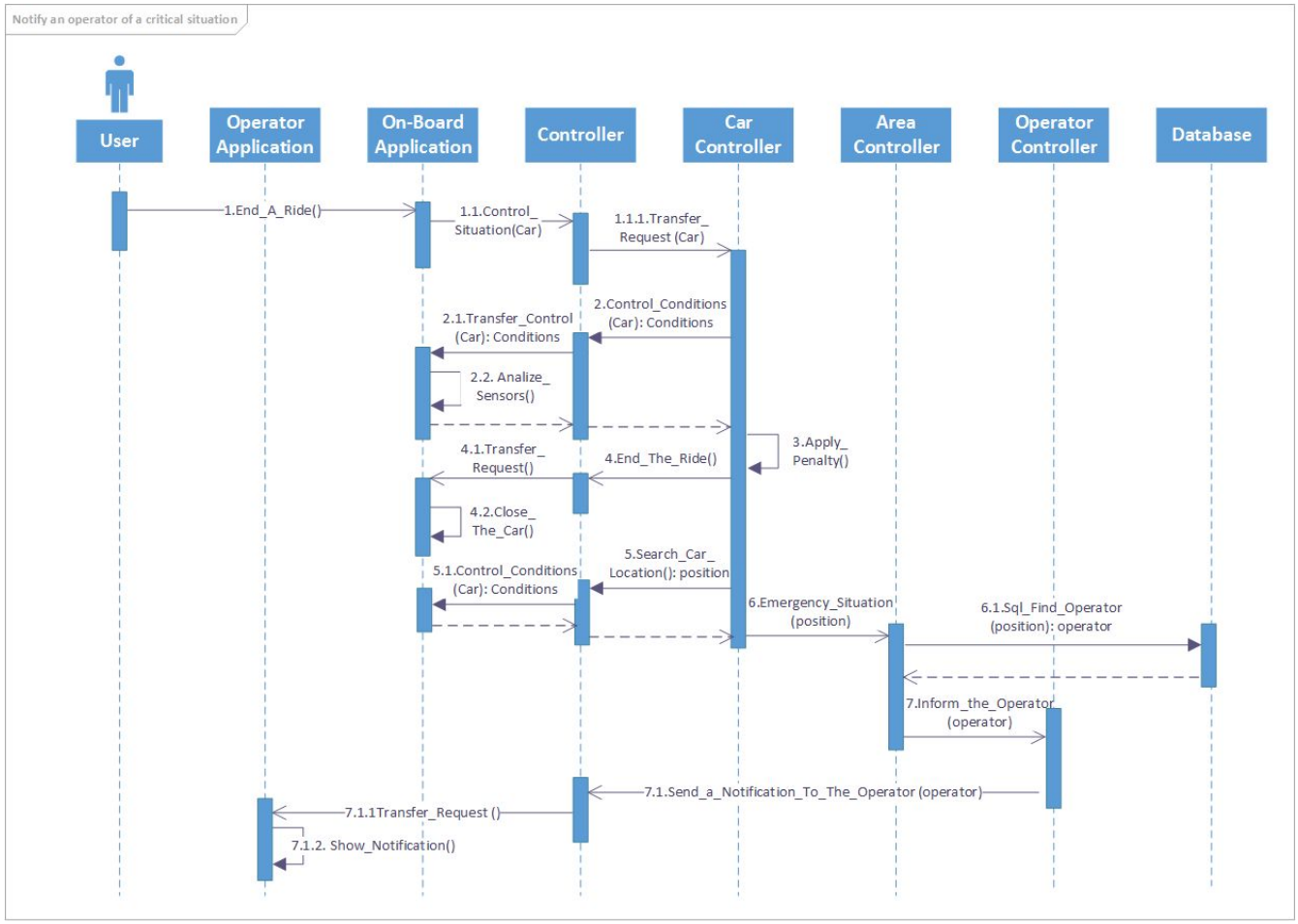

Figure 8: Notify an operator of a critical situation

This sequence diagram describes a situation in which the operator has to intervene on a car. When a user chooses to end the ride, the on-board app sends a request to the car controller to check the situation of the car. If the condition needs an intervention of the operator, a penalty is charged to the User and the car is closed. After that, sensors are analysed to find the position of the car: in this way the system can check in the database the operator that is associated with this area and can transfer to the Car Controller the request for maintenance. The controller sends a request to the operator application that displays a notification to the User.

2.5. Component Interfaces

As shown in the deployment view, our server has a web part and an application part. The web Server has the task to interpret the HTML requests from the client and to address them to the Application Server. So here we can distinguish three different interfaces depending on the service required (User, On Board, and Operator) on the client side. Then we have an application server that is the core of our server and elaborates the requests. At least we show the external interface services provided to the system.

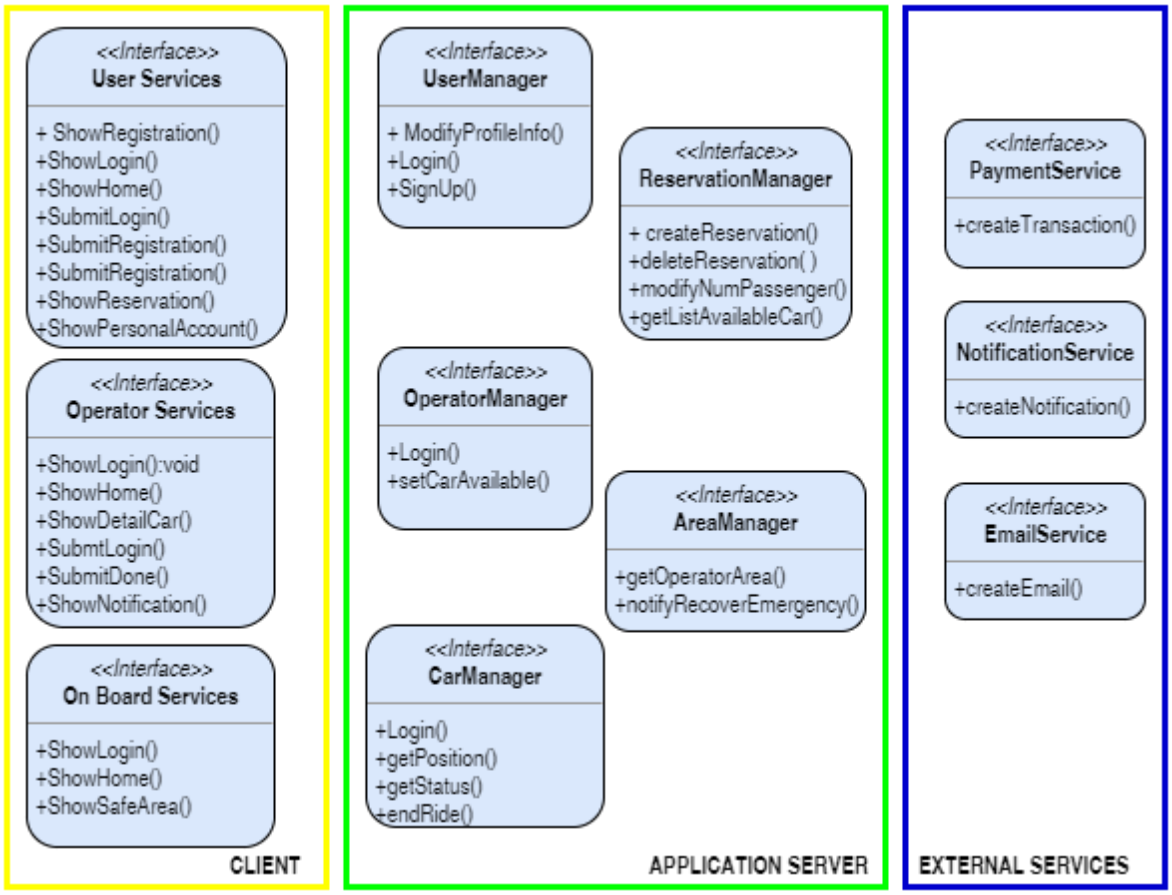

Figure 9 : Component Interface View

2.6. Selected Architectural Style and Patterns

The Power Enjoy system is structured as a three-tier architecture, that is a client-server software architecture pattern in which we can distinguish:

- ➔ **Presentation Tier** that is the user interface (client side).
- ➔ **Business Logic Tier** that makes logic decisions, coordinates the application, processes commands and performs calculation.
- ➔ **Data Tier** that is where the informations are stored.

The main reasons we chose a three-tier architecture are the following:

- ➔ It is possible modifying a single tier without affecting the others.
- ➔ It is possible using different sets of developer for each tier.
- ➔ The Database is more secure because the Client does not have a direct access.
- ➔ When one tier fails, there is no data loss, because you are always secure accessing the other tiers.

2.6.1 Architectural Style

- ❑ **RESTful API:**

  it is an architectural style for distributed systems that manipulates data, resources and information through HTTP protocol from client to server and the content is returned through JSON that represent the simple data structure of the response.

  An example of the commands used by HTTP:

Resource	GET	POST	PUT	DELETE
/reservations	list of all reservations	create new reservation	update all reservations	delete all reservations
/reservations/0001	show the specific reservation	error	if exists, update the reservation, else error	delete the corresponding reservation

Client support only limited methods of HTTP request, in the example above they can only create a new reservation (POST /reservations) ,update their reservation, show it and delete it (GET,PUT,DELETE /reservations/0001).
The security concern is managed defining the session of a user as a resource.
This solution allows to the server to change resources when it wants and also, it is a good way to organize class of datas.
The main advantages of REST are:

- ➔ separates completely the User interface from the Server, and so improves the scalability, allows to develop components independently, and increase the portability of the interfaces to other kind of platform.
- ➔ Visibility, reliability and scalability.
- ➔ independency from platforms and languages.

- **Client-Server:**

  The client side includes both mobile application and the on-board application. As we said before, this is a thin application to guarantee high performance and to keep things simple.
  The server side includes the application server and the DB.
  As said previously, the main reason to choose a Client-Server Architecture is to manage different things independently in order to increase the scalability of the System.

2.6.2 Architectural Patterns

- **MVC Pattern:**

  This pattern divides the components of the System in three part:

  - View that is a GUI for the user on the client side
  - Controller to manage incoming request and services on server side
  - Model to manage the datas of the system on the server side.

  The MVC pattern provides a clear distinction between the different application modules, and also increases the flexibility, and reusability of each component such as we can update only the GUI interface without modifying control and model.

- **State Pattern:**

  This pattern can change the behavior of an object in real time such as the car state in our system, as we showed in the Statechart Diagram in the RASD.

2.7. Other Design Decisions

We chose to use different external open services in our design decisions, such as a map service like Google Maps, a notification service(push and email) and payment gateway service.

3. Algorithm Design

We provide here some examples of the main functions to be developed.

3.1 Find a car

Function `findACar`(*location*: Coordinates)
Data *location,* coordinates of the address inserted by the User or of his current position
Result *listOfCarsAvailable,* a list of objects Car located within a certain radius distance from the location chosen, if the address inserted or the recorded position is valid, ***null*** otherwise

```
    Foreach car in listOfCars do
        if car.available() then
            if Map.calculateDistance(car.getPosition() , location) <= radiusDistance
            then
                listOfCarsAvailable.add(car)
            end
        end
    end
return listOfCarsAvailable;
```

The "*findACar"* function allows the User to find the cars available around him or around the location he inserted. This functions scans the list of all the car registered on the database and, if they result to be available (not yet reserved) and located within a certain distance from the position chosen by the User, returns them as a list of car available. The cars belonging to it will be then shown graphically on the mobile interface, by means of rounded arrows arranged in points of the map corresponding to their real coordinates.

3.2 Reserve a Car

Function `reserveACar`(*carSelected*: Car, *user*: User)
Data *carSelected,* the car selected by the User
Result *true* if the operation has a successful conclusion, *false* if some errors occur

```
    Foreach car in listOfCars do
        if car == carSelected then
            reservation= new Reservation(user, car, Time.getCurrentTime() )
            car.setUnavailable()
            user.sendBookingConfirmation( user.getEmail())
        end
    end
return;
```

The "*reserveACar"* function allows the User to reserve the car chosen. It receives as parameters an instance of the class User, corresponding to the user wants to make the reservation, and an instance of the class Car, which represents the Car which has to be reserved. The function identifies the car selected and creates a new reservation.

3.3 Calculate final charge

Function `calculateFinalCharge` (*reservation*: Reservation)
Data *reservation,* the reservation about which we want to calculate the charge
Result *charge,* the final charge of the ride, calculated taking in account the discounts
float basicCharge =
Time.*calculateInterval*(*reservation*.`getReleaseTime()`,
reservation.`getPickUpTime()`)* *chargePerMinute*
if *reservation*.`moneySavingMode()` **and**
reservation.`getCar().getPosition().equalTo(` reservation.`moneySavingSafeArea().getPosition())` **then**
charge = *basicCharge* * (1 - 1 * *percentageDiscountMoneySaving)*
else if *reservation*. `getCar().getNumberOfPassengerAuthenticated()` >1 **then**
charge = *basicCharge* * (1 - 1 * *percentageDiscountPassengers)*
else if *reservation*. `getCar().getBatteryLevel()` >= 50 **then**
charge = *basicCharge* * (1 - 1 * *percentageDiscountBatteryLevel)*
else if *reservation*. `getCar().getPosition().isInSafeArea()` **and**
reservation. `getCar().batteryPlugInserted()` **then**
charge = *basicCharge* * (1 - 1 * *percentageDiscountRecharge)*
else if *reservation*. `getCar().getBatteryLevel()` <= 80 **or**
reservation. `getCar().getPosition().isMoreThan3KmFar()` **then**
charge = *basicCharge* * (1 + 1 * *percentagePenalty)*
else
charge = *basicCharge*

return charge;

The "*calculateFinalCharge"* function allows the system to calculate the charge to debit to the User. This function verifies the truthfulness of certain statements relating to the Driver's accuracy of use, and, consequently, it applies or not some discounts or penalties to the "*basicCharge",* the cost of the ride based just on the time of use of the car.

4. User Interface Design

4.1 Mockups

Mockups are already shown in the Requirements Analysis and Specification Document, section 2.1.2 and following. Here we extend some crucial points.

4.1.1 User Reservation interfaces

Figure 10 : Mockup Reservation with passengers flow

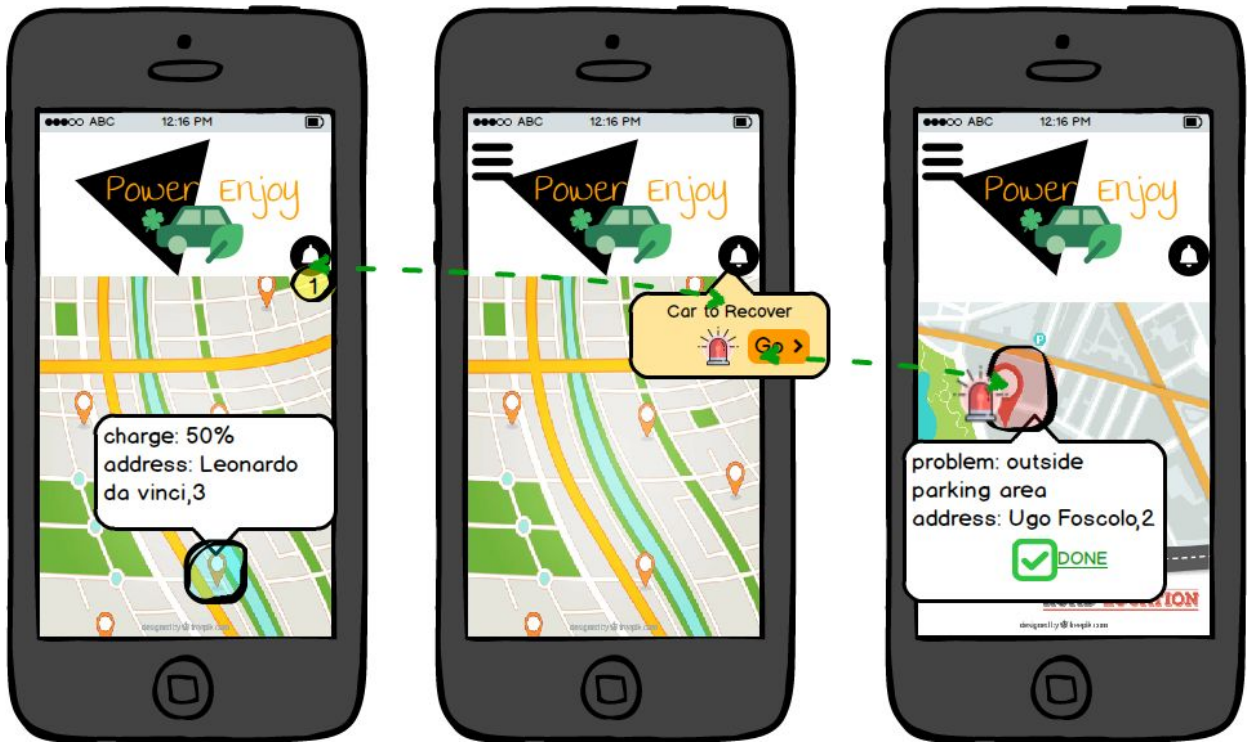

Figure 11 : Mockup Operator flow

4.2 UX diagrams

Below, the User Experience (UX) diagrams, to show the basic and most important actions which can be performed by a User or an Operator.

In case an error occurs during the compiling of the “input forms” made by the User, the input form will be reloaded and the User will be asked to fill the fields again. Clicking on a “choice button” (for instance, the ones which represent the payment method, in the registration form) implies a graphic highlighting of it and it is not necessary the reloading of the screen.

Some functions are described in the diagrams, with the only aim of giving a sample of the basic functions to code; the names given to them are completely arbitrary.

4.2.1 UX: Sign Up and Login Interfaces

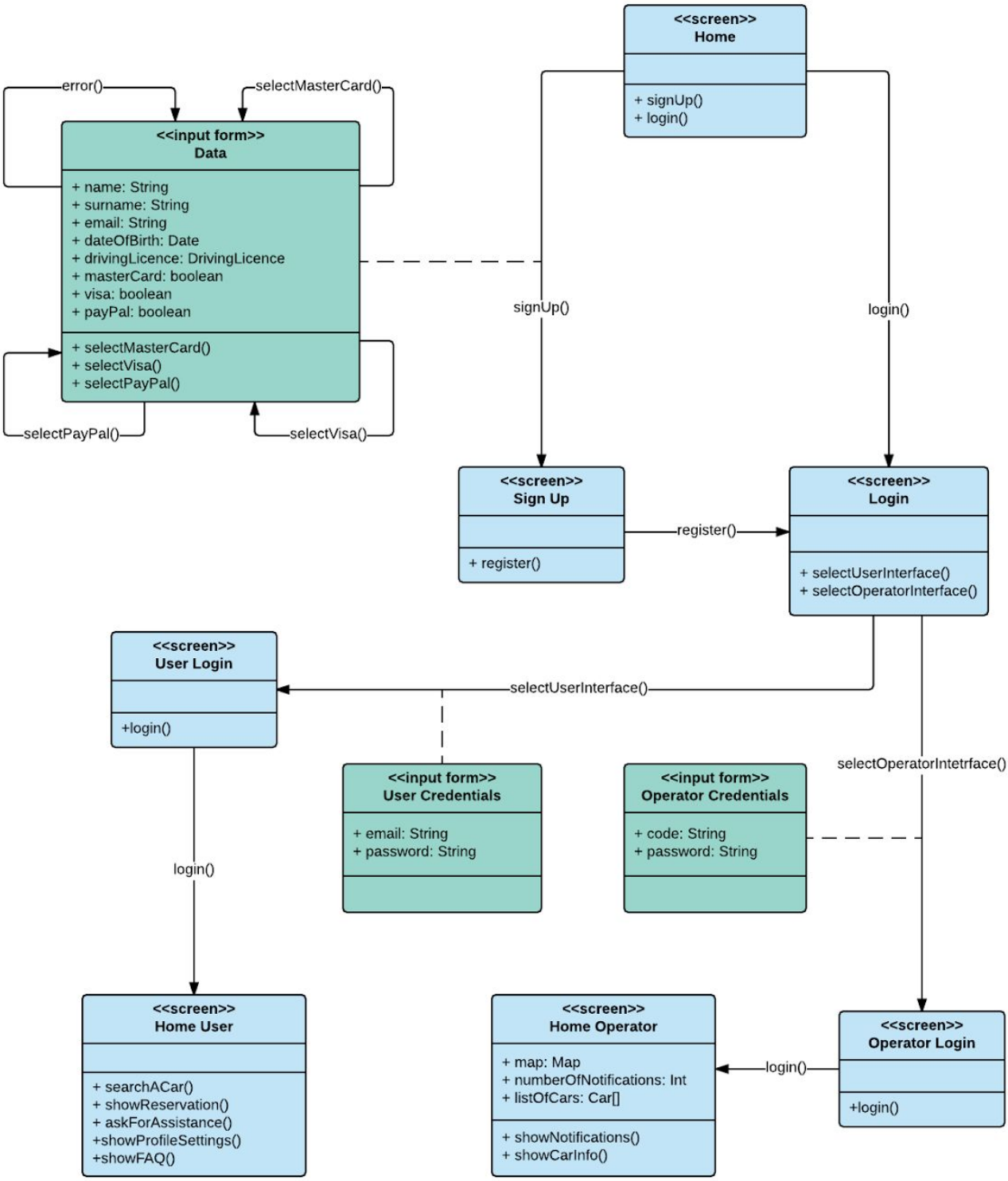

Figure 12: UX sign up and login

4.2.2 UX: User and the mobile interface

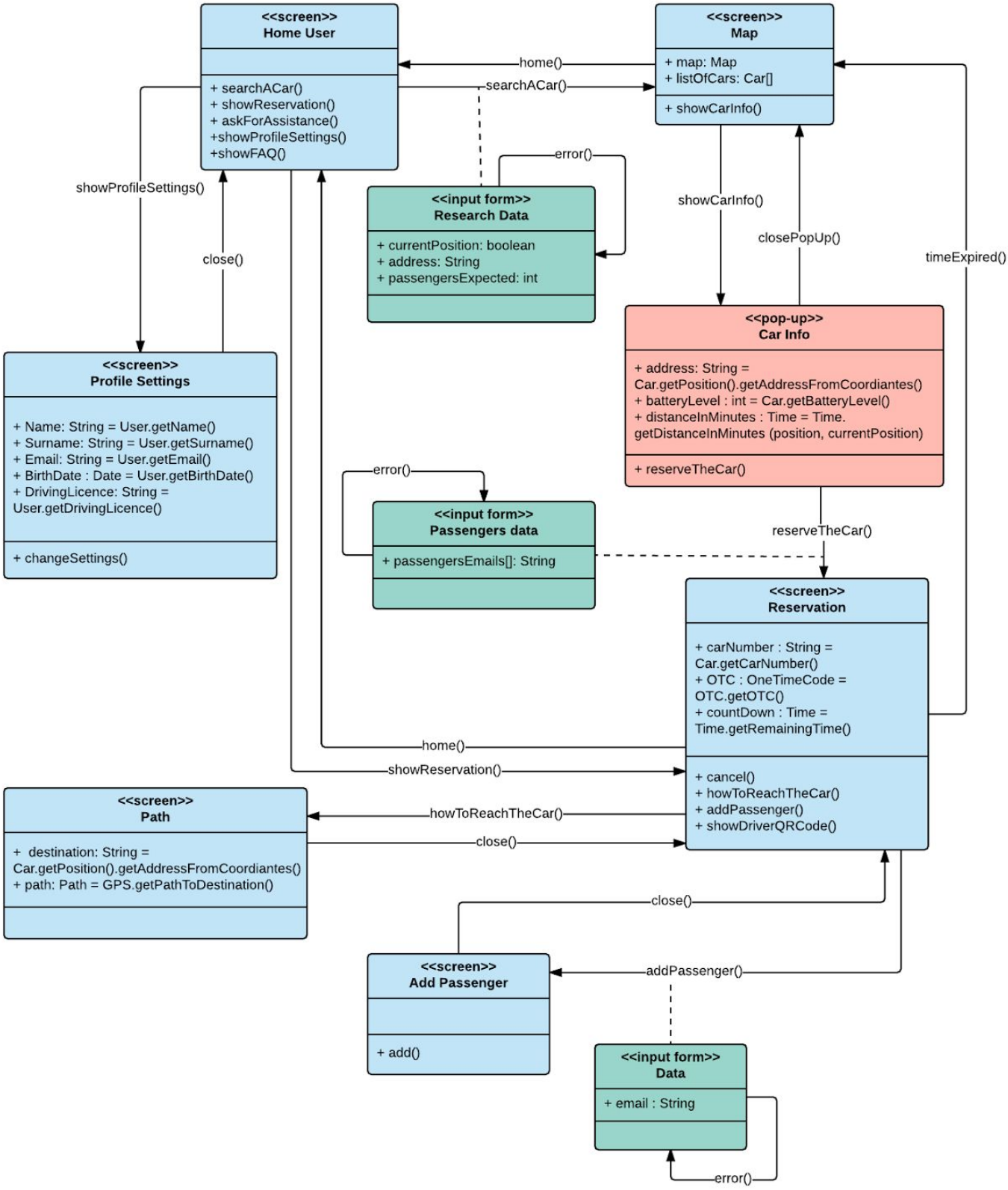

Figure 13: UX user mobile

4.2.3 UX: Operator and the mobile interface

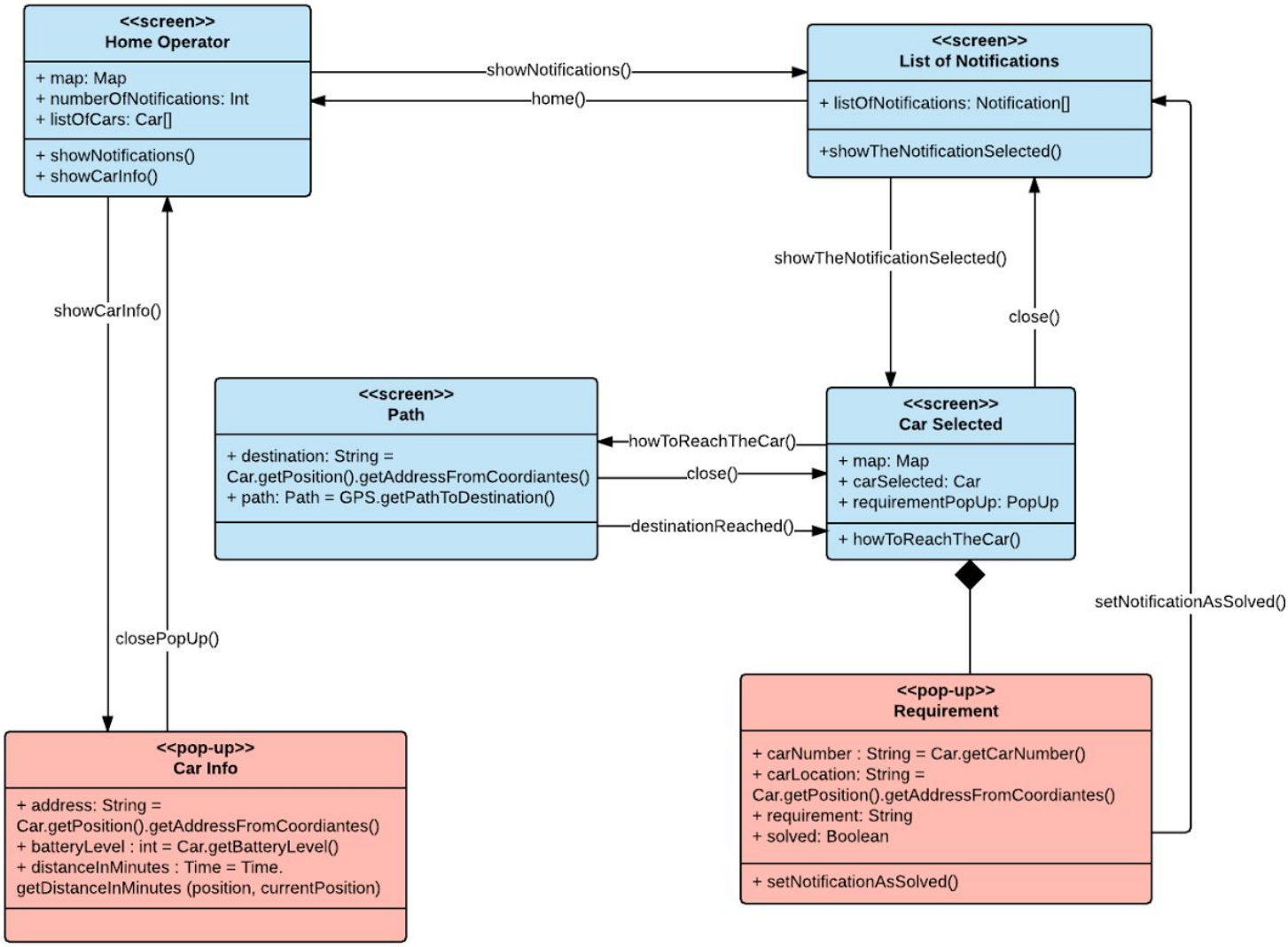

Figure 14: UX operator mobile

4.2.4 UX: User and the on-board interface

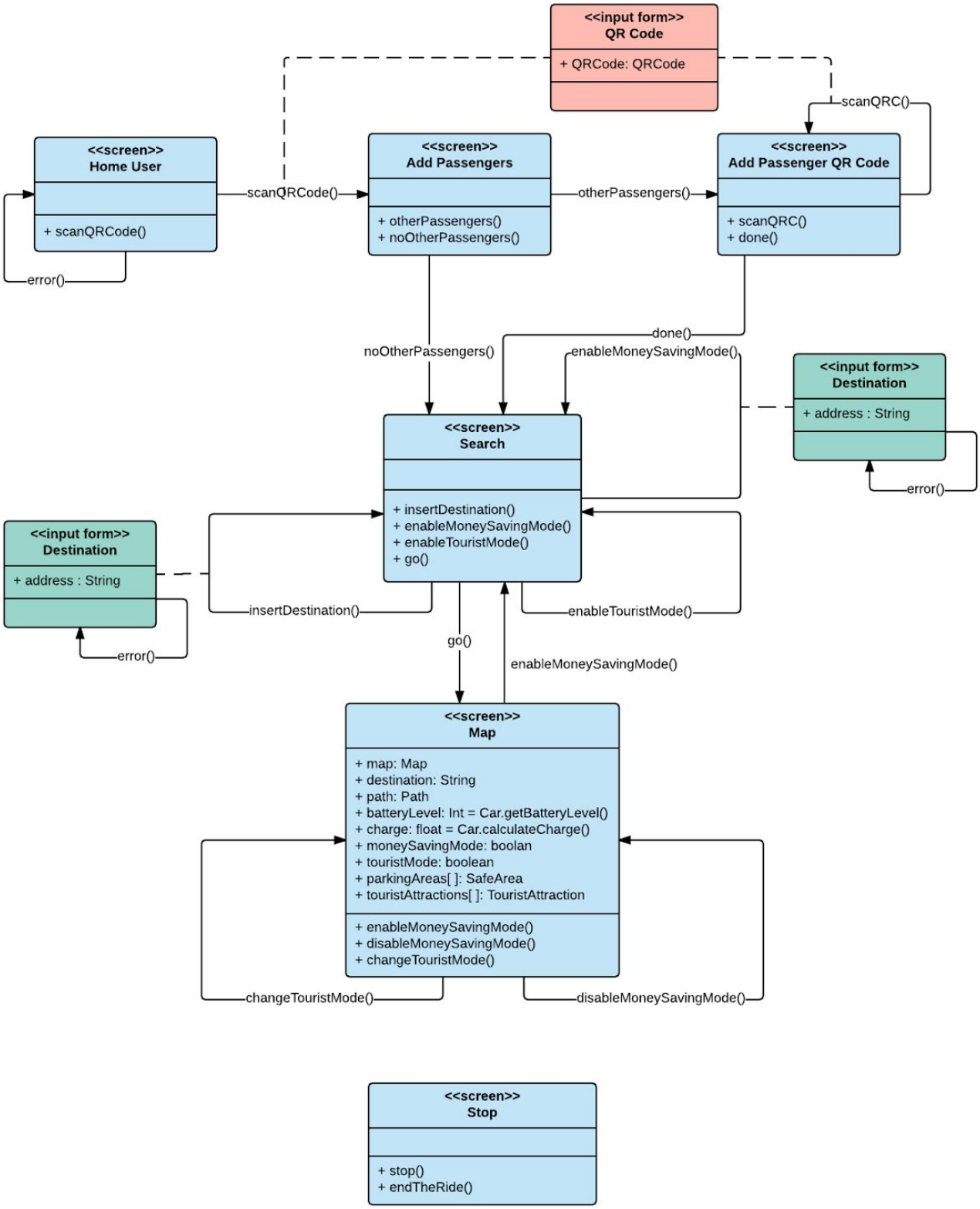

Figure 14: UX user on-board device

4.3 BCE diagrams

We inserted Boundary Control Entity diagrams to show the functionalities of all the mobile and on-board devices of our system. This chart is also useful to show all the interactions between classes, in agreement with what is described in the previous diagrams (Class, UX, Sequence).

4.3.1 BCE: *User mobile application*

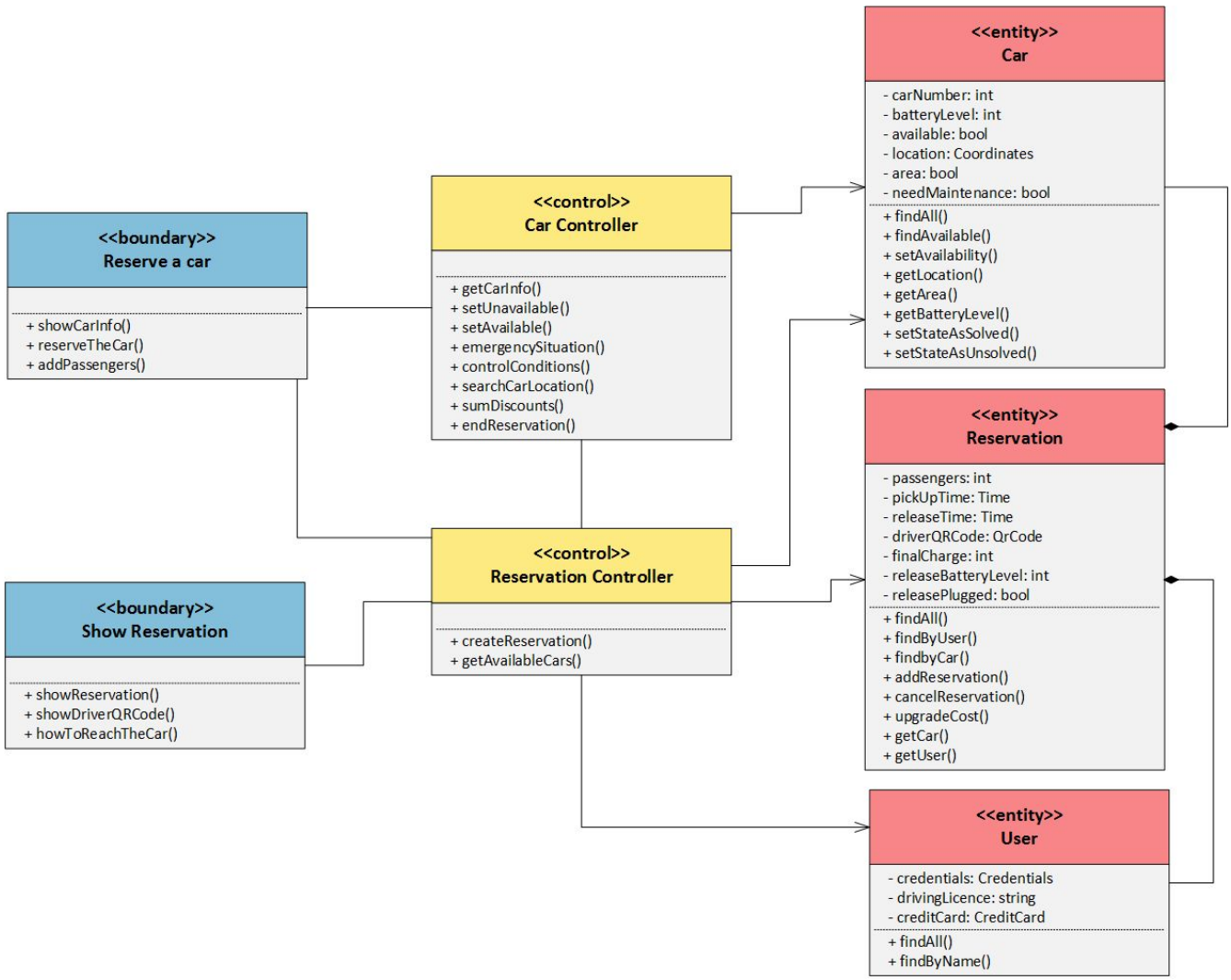

Figure 15: BCE User mobile application

4.3.2 BCE: *Operator mobile application*

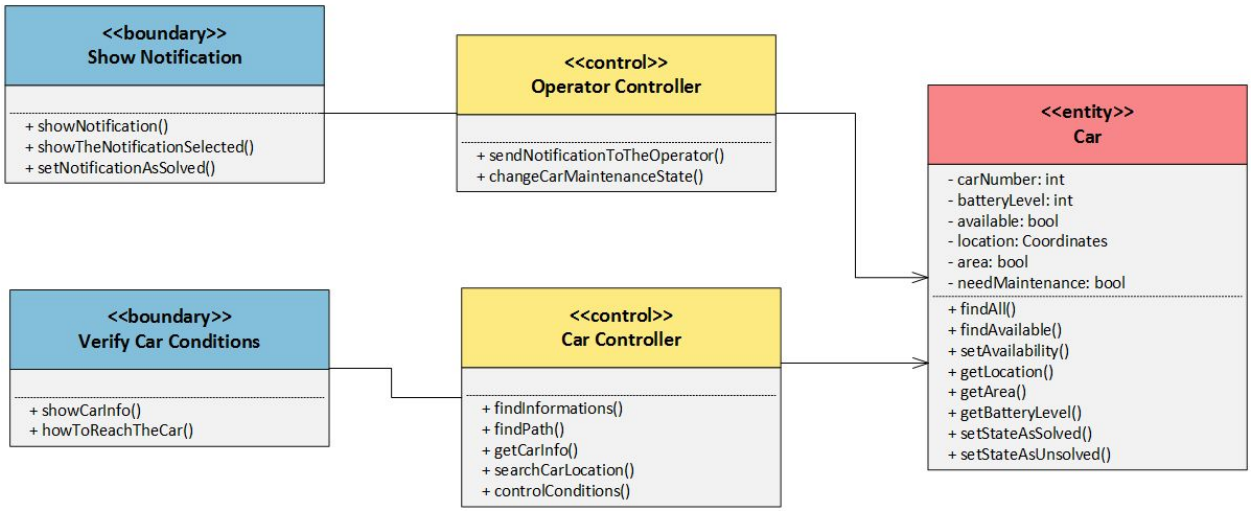

Figure 16: BCE Operator mobile application

4.3.3 BCE: *On-Board application*

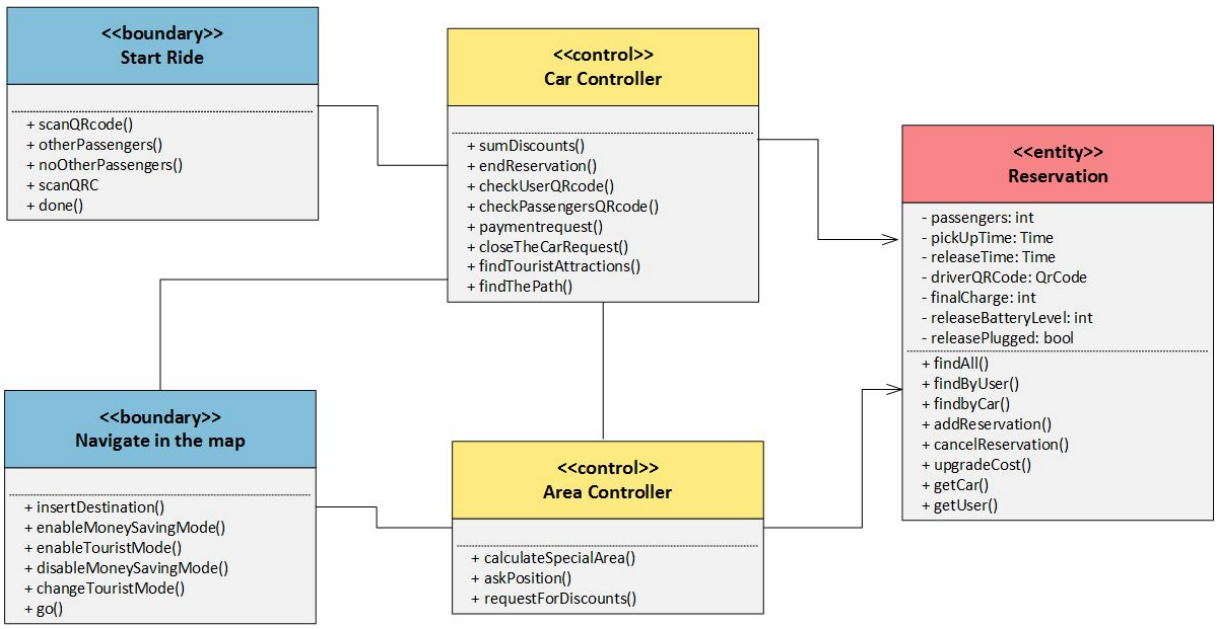

Figure 17: BCE On-Board application

5. Requirements Traceability

[G1] Allows Guests to sign up providing their credential.

- **[FR 1.1]** The system must verify that the email address has not to be registered yet on the DBMS.
    - [DD 1.1.1] The **UserController** component and Interface verifies through the **database** that the email address is not stored.
- **[FR 1.2]** The system must verify that the payment method inserted is valid.
    - [DD 1.2.1] The **UserController** and Interface that manages the registration and the access to the Server checking the correctness of all input informations.
- **[FR 1.3]** The system allows Guests to register themselves, by inserting their personal informations.
    - [DD 1.3.1] The **Controller** and the associated interface provides a form to the user and the possibility to submit it, the **UserController** validate it before add the new User to the **DB** through the **Model**.
- **[FR 1.4]** Guests must be able to see only the “Sign up and Log in” layout.
    - [DD 1.4.1] The **Controller** with the associated interface for mobile services, show the homepage.
- **[NFR 1.1]** The System must store passwords in a safe way.
    - [DDN 1.1.1] The **DataBase** component that stores in a secure way username,password through hash operation and all the other informations about the new User. To guarantee that we chose a **three-tier architecture**.
- **[NFR 1.2]** The email address used for the registration must be formally correct.
    - [DDN 1.2.1] The **UserController** checks also the validity of the email address before submitting the form.

[G2] Allows Users to login.

- **[FR 2.1]** The system must verify that the entered username and password are already registered into the DBMS.
    - [DD 2.1] The **UserController** that manages the login and access the Database to check the correctness of username and password to authenticate the User and create a connection from Client to Server.
- **[FR 2.2]** If username and password are correct , the system allows the User to log in.
    - [DD 2.2] after the response from the application Server, the Web Server guarantee the access through the **Mobile Service interface** submitting the login and showing the homepage.

[G3] Allows Users to find the locations of available cars within a certain radius distance from their current location or from a specified address.

- **[FR 3.1]** The system must know the position of all the cars that are available.
    - [DD 3.1.1] **GPS tracker**
- **[FR 3.2]** The system must show the Users the position of the available cars within a certain radius distance from their position.
    - [DD 3.2.1] Access the **GPS** information of the User through the application Policy.
    - [DD 3.2.2] **Reservation Controller** component checks the status of the car through the **DB** and shows the available cars in a certain radius from the position of the User.
- **[FR 3.3]** The system must show the Users the position of the available cars within a certain radius distance from the address inserted by him/her.
    - [DD 3.3.1] **Reservation Controller** get the coordinates of the address inserted and checks the status of the car through the **DB** showing only the available cars in a certain radius from this address.
- **[NFR 3.1]** The GPS of the cars must be on 24 hours a day.
    - [DDN 3.1.1] The **GPS Tracker.**

[G4] Allows to reserve a car for up to one hour.

- **[FR 4.1]** Since the User pushes the reservation button, the system must set as unavailable the reserved car for all the other Users.
    - [DD 4.1.1] Submit the reservation through the Mobile Service Interface and then the **Reservation Controller** create a new reservation object and stored to the DB.
    - [DD 4.1.2] The **Car** component associated status is set unavailable.
- **[FR 4.2]** The system must allocate an OTC to the reservation.
    - [DD 4.2.1] The **reservation** component generates an OTC when it is created and associates it to the User in the DB.
- **[NFR 4.1]** The system must be able to count down the minutes since the reservation, up to one hour.
    - [DDN 4.1.1] The **reservation** component generates an expiration time associated to the User in the DB.
- **[NFR 4.2]** The OTC is delivered to the User in a safe way.
    - [DDN 4.2.1] The **Notification Manager** component through an **email gateway** send the OTC code.

- **[NFR 4.3]** The QR code is delivered to the User and to the Passengers in a safe way.
    - [DDN 4.2.3] The **Notification Manager** component through an **email gateway** send the QR code.

[G5] For a reservation whose duration exceeds the prearranged time, the system charges a fee.

- **[FR 5.1]** If after one hour the User has not opened the car yet, the system has to charge him a fee of 1€.
  - [DD 5.1.1] The **Car controller** checked if at the expired time the car was logged by the User.
  - [DD 5.1.1] The **Car controller** through the **Payment Manager** charge the user with a **payment gateway**.
- **[FR 5.2]** When the reservation time is up, the system must set the car available again.
  - [DD 5.2.1] The **Car controller** set the status available again and the reservation is deleted.

[G6] Allows the only User who reserved the car earlier to unlock it, when in proximity.

- **[FR 6.1]** The system must allow the User to find the car bluetooth signal.
- **[FR 6.2]** The system must allow the User to connect to the car bluetooth signal.
  - [DD 6.1.1 / 6.2.1] The **Bluetooth module** installed on the car.
- **[FR 6.3]** The system must verify that the OTC inserted by the User matches with the one expected.
  - [DD 6.3.1] The **Car Controller** checks that the OTC is the same of the OTC associated in the DB.
- **[FR 6.4]** The system must unlock the car, if the OTC matches.
  - [DD 6.4.1] The **Car Controller** authorized the User and unlock the car.
- **[NFR 6.1]** The car bluetooth signal is available 24 hours a day.
  - [DDN 6.1.1] **Bluetooth module** integrated is always power up.

[G7] Allows the User to power up the reserved car.

- **[FR 7.1]** The system must check if the QR code of the Driver matches with the one allocated to him/her during the reservation process.
  - [DD 7.1.1] The **OnBoard Application** component uses a **QR reader** to scan and recognize each QR code, and sees if the QR code associated to the user in the DB match.
- **[FR 7.2]** The system must check the QR code of all the passengers to apply a price reduction of 10%, if there are at least two other passengers into the car.
  - [DD 7.2.1] The **OnBoard Application** asks the User to insert QR code of passengers and the **Car controller** compares with the codes in the DB.
  - [DD 7.2.2] The **Car Controller** adds a discount associated to that **User** in the DB.

- **[FR 7.3]** The system must allow the User to power up the car just pushing the “Start and Stop” button, if the User has been already authenticated.
    - [DD 7.3.1] The **Car Controller** after the authentication allows the user to start.

[G8] Charges the User from the moment he/she starts up the engine for a given amount of money per minute, up to the moment when he exits the vehicle and also notifies the User of the current charges real time.

- **[FR 8.1]** When the User leaves the car, the system must ask the User if he wants to stop or to end the ride.
    - [DD 8.1.1] The **CAN BUS** allows the **OnBoard Application** to know exactly when the car stops and it shows the two possibility.
- **[FR 8.2]** The system has to show on the screen the current charges real time.
    - [DD 8.2.1] The **OnBoard Application** calculates in real time the amount of money from the moment that the Car start to when it will end the ride.
- **[FR 8.3]** When the User ends the ride the system must charge him for a given amount of money per minute, applying the related discounts and penalties.
    - [DD 8.3.1] The **Car Controller** receives from the **OnBoard Application** the request to end the ride of that precise car, so it charge the User through the **Payment Manager,** also if there are discount or fee associated to the **User** applies them.

[G9] Notifies the User of the current battery level of the car real time.

- **[FR 9.1]** The system must show the current charge of the car battery on the screen of the on-board computer.
    - [DD 9.1.1] The **CAN BUS** allows the **OnBoard Application** to know exactly when the car values and shows that to the User during ride.
- **[FR 9.2]** The System must show an alarm when the battery is particularly low, and show the nearest safe areas from the current position of the car.
    - [DD 9.2.1] The **OnBoard Application** shows an alarm if the battery is low and shows the nearest safe area through the **Map service component.**

[G10] If required, allows the User to receive informations about the main tourist attractions in proximity, while driving.

- **[FR 10.1]** The system must show on the map of the on-board device the most fascinating and popular tourist attractions, as museums, exhibitions, monuments or buildings interesting from an architectural point of view, while he/she is moving in proximity of them.

    - [DD 10.1.1] The **Maps** service on the **OnBoard Application** allows to see the tourist attraction mode.
    - [DD 10.1.2] The **GPS Tracker.**

- **[FR 10.2]** If the User clicks on an attraction, the system has to display a brief description of it.
    - [DD 10.2.1] The **Maps** service on the **OnBoard Application** allows to see images and brief description on a selected tourist attraction.
- **[NFR 10.1]** The system must update its maps every day.
    - [DN 10.1.1] **Google Maps** is updated every day.

[G11] Allows the User to locate the nearest safe areas.

- **[FR 11.1]** The system has to show on the map of the on-board device the safe areas available, as long as the User is driving within a certain distance from them.
    - [DD 11.1.1] The **Maps service component** on the **OnBoard Application** allows to personalize maps adding the safe area points, through the **Area Controller** so that the user can see them.
- **[FR 11.2]** In case of emergency, the system has to notify immediately the problem to the User and to point him/her the nearest safe areas where to stop and let the car be repaired.
    - [DD 11.2.1] The **OnBoard Application** alerts the user showing through the **Map** the nearest safe area.
- **[FR 11.3]** If the User choices the "Money Saving Mode", the system must show, on the screen of the on-board device, all the safe areas in proximity of the Driver's route, until the "Money Saving Mode" is enabled.
    - [DD 11.3.1] The **map Service** on the **OnBoard Application** shows only the safe area with free parking lot in order to redistribute cars on the territory. This is possible also for the **Area controller** component.

[G12] Locks automatically when the User closes the car door, after exiting the vehicle.

- **[FR 12]** The system must lock the car after the User has expressed the will of ending or stopping temporarily the ride, has left the vehicle and has closed the doors.
    - [DD 12.1] When the **OnBoard Application** is stopped or paused automatically locks the doors after a certain amount of second.

[G13] Allows operators to know exactly when cars need to be recharged or an intervention of

different nature.

- **[FR 13.1]** The system must monitor the condition of the car, in order to be able to see if the car needs to be charged or repaired.
    - [DD 13.1.1] The **OnBoard Application** through the **CAN BUS** monitors the parameter of the car, if an emergency occurred sends a request to the **Operator Controller** that through the Area Controller get the associated operator of that area and send a push notification, using the **notification manager** component.
- **[FR 13.2]** The system has to send a request for intervention to the operator to whom was assigned the area in which the car is.
    - [DD 13.2.1] The **Operator Controller** through the **Area Controller** get the associated operator of that area and send a push notification, using the **notification manager** component.

[G14] Allows the User to add passengers when they reserve a car.

- **[FR 14.1]** The User must be able to add at most 4 passengers to the reservation.
    - [DD 14.1.1] The **User Service Interface** provides a form that allows user to add email addresses of passengers for the reservation.
- **[FR 14.2]** The system must send a QR code to all the passengers, in order to be recognized for a certain the ride.
    - [DD 14.2.1] The **User Service Interface** submit the request and the **Reservation controller** send a QR through the **notification Manager**.
    -

[G15] Allows the User to delete a reservation.

- **[FR 15]** After the reservation, the system must permit the User to delete it.
    - [DD 15.1] The **User Service Interface** shows the reservation that can be delete by the User. So the **Reservation controller** delete the reservation made also in the DB.

[G16] Allows the User to modify their profile information.

- **[FR 16]** The System must enable the User to change its personal data, that could be obsolete.
    - [DD 16.1] The **User Service Interface** shows the profile page and the user can submit the new information that the **User controller** checks and if they are correct it saves them in the system.

[G17] Reward the User for a good behaviour with discounts.

- **[FR 17.1]** The system must apply a discount of 10%, if the passengers are more or equal than 3.
    - [DD 17.1.1] If the **On Board Application** and the associated interface logged at least three passenger than the **Car controller** register the discount that can be applied at the end of the ride.
- **[FR 17.2]** The system must apply a discount of 20%, if the car is left with more than 50% of battery.
    - [D 17.2.1] If at the end of the ride **Car controller** checks that battery has the condition of discount and apply it for the **Payment Manager.**
- **[FR 17.3]** The system must apply a discount of 30%, if the User brings the car in a safe area and puts it in charge.
    - [DD 17.3.1] At the end of the ride **Car controller** through **Area Controller** checks that the car is in a safe area and also through the **On Board Application** can check if the car is charging**.**
- **[FR 17.4]** If the User parked in an area using "Money Saving" option, the system must apply a discount.
    - [DD 17.4.1] At the end of the ride **Car controller** through **Area Controller** checks that the car is in a safe area of Money Saving option and apply a discount through the **Payment Manager.**

[G18] Penalize the User for a improper behaviour with additional charges.

- **[FR 18.1]** The system must apply a penalty of 30%, if the car is left with less than 20% of battery.
    - [DD 18.1.1] If at the end of the ride **Car controller** checks that battery has the condition of charging the User and applies it through the **Payment Manager.**
- **[FR 18.2]** The system must apply a penalty of 30%, if the car is left at more than 3 km from the nearest station.
    - [DD 18.2.1] At the end of the ride **Car controller** through **Area Controller** checks where is the Car, if it is in an outside area, the **Payment Manager** applies an extra charge to the User.
- **[FR 18.3]** The system must charge the associated User for every exceptional penalty due to that User.
    - [DD 18.3.1] The **Car controller** checks the situation at the end of every ride.

6. Effort Spent

6.1 Giulia Leonardi

- 28 Novembre: 2h
- 29 Novembre: 4h
- 30 Novembre: 4h
- 1 Dicembre: 3h
- 2 Dicembre: 5h
- 3 Dicembre: 2h
- 4 Dicembre: 1h
- 5 Dicembre: 3h

6.2 Marzia Degiorgi

- 18 Novembre: 3h
- 19 Novembre: 1h
- 25 Novembre: 3h
- 26 Novembre: 3h
- 28 Novembre: 4h
- 29 Novembre: 3h
- 30 Novembre: 6h
- 1 Dicembre: 2h
- 2 Dicembre: 4h
- 5 Dicembre: 3h

6.3 Valentina Ionata

- 26 Novembre: 1h
- 27 Novembre: 3h
- 28 Novembre: 4h
- 29 Novembre: 4h
- 30 Novembre: 6h
- 1 Dicembre: 3h
- 2 Dicembre: 3h
- 3 Dicembre: 4h
- 4 Dicembre: 2h
- 5 Dicembre: 4h
- 8 Dicembre: 1h

7. References

- “Green Move: A platform from highly configurable, heterogeneous electric vehicle sharing”, Andrea G. Bianchessi, Gianpaolo Cugola, Simone Formentin, Angelo C. Morzenti, Carlo Ongini, Emanuele Panigati, Matteo Rossi, Sergio M. Savaresi, Fabio A. Schreiber, Letizia Tanca, Edoardo G. Vannutelli Depoli, 2014.
- Sample Design Derivable Discussed on Nov.2.pdf
- Architecture and design in practice.pdf

8. Revision History

Version	Date	Description
v 1.0.0	11/12/2016	Initial Release
v 1.0.1	14/01/2017	Changes on the Deployment Diagram and Minor Fixes
v 1.0.2	03/02/2017	Formal updates