



**POLITECNICO**  
MILANO 1863

**Software engineering 2 Project**

Design Document

**PowerEnJoy**

Perugini Alex	876359
Re Marco	873564
Scotti Vincenzo	875505

# Table of Contents

- 1. Introduction**
  - 1.1. Purpose
  - 1.2. Scope
  - 1.3. Glossary
    - 1.3.1. Definitions
    - 1.3.2. Acronyms
    - 1.3.3. Abbreviations
  - 1.4. Reference documents
  - 1.5. Document structure
- 2. Architectural design**
  - 2.1. Overview
  - 2.2. Component view
    - 2.2.1. Car
    - 2.2.2. App
    - 2.2.3. Server
  - 2.3. Deployment view
  - 2.4. Runtime view
    - 2.4.1. Reservation
    - 2.4.2. Unlock
    - 2.4.3. Ride
    - 2.4.4. Notification
  - 2.5. Selected architectural styles and patterns
    - 2.5.1. Architectural styles
    - 2.5.2. Patterns
- 3. Algorithm design**
  - 3.1. Reservation creation
  - 3.2. Nearest operator
- 4. User Interface design**
  - 4.1. UX diagram
  - 4.2. BCE diagram
  - 4.3. User Interface mockups
    - 4.3.1. Unlogged user interface
    - 4.3.2. Operator interface
    - 4.3.3. Client interface
- 5. Requirements traceability**
- 6. Used tools**
- 7. Hours of work**

# 1 Introduction

## 1.1 Purpose

This document describes with more technical details the design of the PowerEnJoy system presented in the RASD and provides information about the architecture of the system, the analysis of some important algorithms and a description of the user interface.

This document also shows how the requirements presented in the RASD are accomplished by the proposed architecture.

## 1.2 Scope

The scope of the system is to provide to the PowerEnJoy company all the requested functionalities in an efficient and reliable way in order to decrease the costs on the long period.

The system allows users to register and become clients of the company, whose service is available almost 24 hours per day.

The system manages the services provided and the client's requests in the most effective way to reduce the operators to the minimum necessary and so also decrease the total costs.

## 1.3 Glossary

### 1.3.1 Glossary

Charging Area/Station	Subset of safe area where is possible to plug the car to a power grid to charge its battery
Client	User of PowerEnJoy registered to the system with normal log in elements
Driver	User sitting in the car that has the control of the vehicle, can be both a client or an operator
Driving Licence	Category B licence needed to drive the cars provided by PowerEnJoy
Money Saving Option	Functionality provided through the navigation system that enlights the nearest charging area to the client destination with free plugs
Operator	Employee of the company that provides the PowerEnJoy service, he is in charge of managing the cars according to the instructions he receives from the system
Passenger	Person inside the car that is not on the driver seat
Payment Method	The payment methods are all the methods accepted by the external payment system that the system will

	interface with
Power Grid	Grid located at each power station, it has plugs to charge the PowerEnjoy cars
Punishment	Fee on the charge for the ride
Ride	Entire operation of driving the car from the moment the engine is ignited to the moment the car is left parked in a safe area
Reward	Discount on the charge for the ride
Safe Area	Area defined by a set of GPS positions in which is possible to park the PowerEnjoy cars
Special login elements	Email and password gave to operators to authenticate as employees during the worktime

### 1.3.2 Acronyms

GPS	Global Positioning System
RASD	Requirement Analysis and Specification Document
DD	Design Document
DB	Data Base
DBMS	Data Base Management System
RDBMS	Relational Data Base Management System
UX	User Experience
BCE	Boundary Control Entity

### 1.3.3 Abbreviations

App	Application, in this case refers to the software installed on smartphones
-----	---

## 1.4 Reference Documents

1. Project description: Assignments AA 2016-2017.pdf
2. Example documents:
  - Sample Design Deliverable Discussed on Nov. 2.pdf

## 1.5 Document Structure

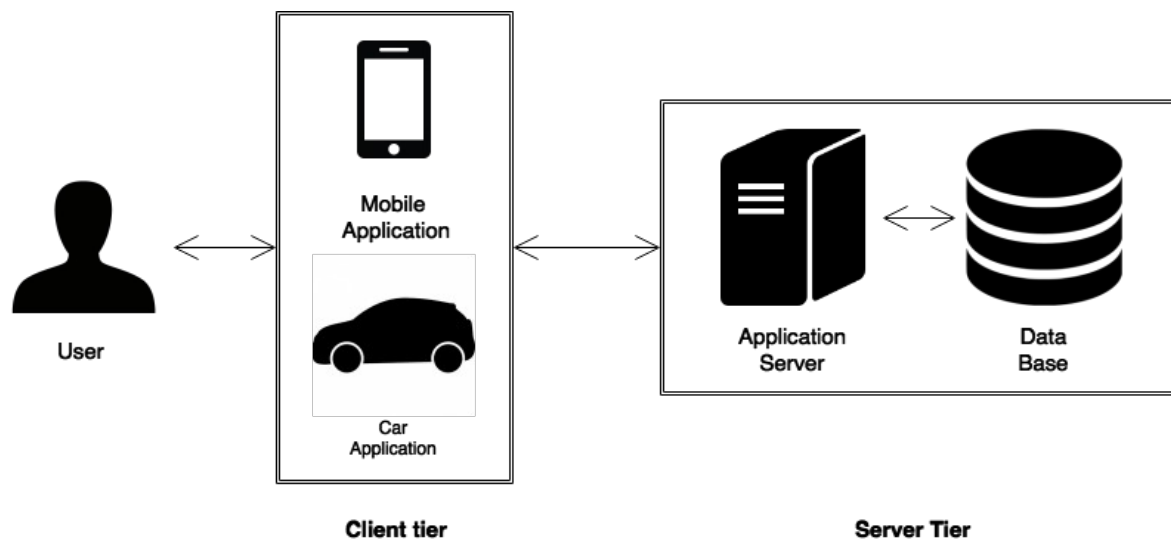
The document is organized as follows:

- Section 1: Introduction, provides a general description of the environment, introduced by the RASD, that will drive the choices in the design of the software.
- Section 2: Architectural design, contains a sequence of descriptions for the intended software and all its components with their interaction, each description increases in the level of the details and departs from the initial abstraction.
- Section 3: Algorithm design, contains an informal and high level description of the more useful and important algorithms.
- Section 4: User Interface design, contains a graphic representation for the UI accompanied with a detailed description of its functionalities and how they are performed.
- Section 5: Requirements traceability, contains the explanation of how the requirements described in the RASD map to the key elements of the derived design.
- Section 6: Used tools, contains the list of the tools used to realize the DD and the scope they were used for.
- Section 7: Hours of work, report of work time for each member.

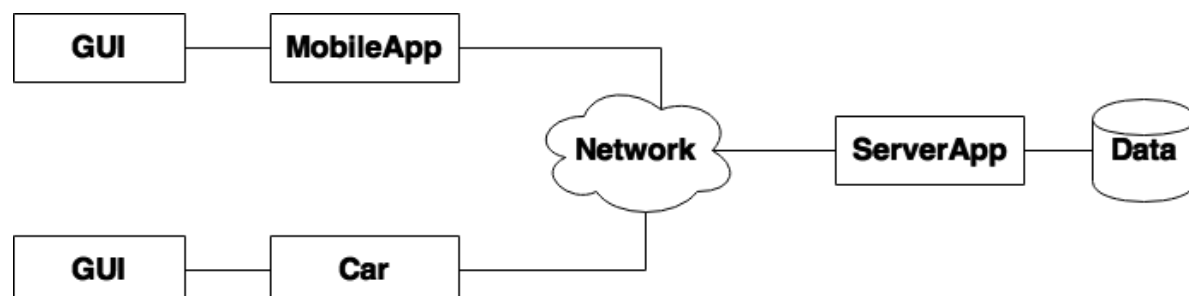
## 2. Architectural design

### 2.1 Overview

The PowerEnJoy system has a Client/Server architecture, this kind of architecture is necessary to develop a mobile application service in which the smartphone sends requests to a server that provides answers to the users and manages all the functionalities. In particular the chosen Client/Server architecture is a Two-tier architecture because it's simpler than a Three-tier architecture that would bring more complexity without any relevant advantage.



The client is not thin, in fact the application logic is split on the server and the mobile application (even if the part on the server is bigger and accomplishes more functionalities than the part on the client), this choice allows to separate the logic which is strictly connected to the smartphone functionalities and the general management of the whole system. In this way the service could be extended with a web application more easily because the application logic on the server and the mobile application logic are decoupled. The system also needs a software running on the cars to get information about them and about the world, in this way the server can control and manage them.



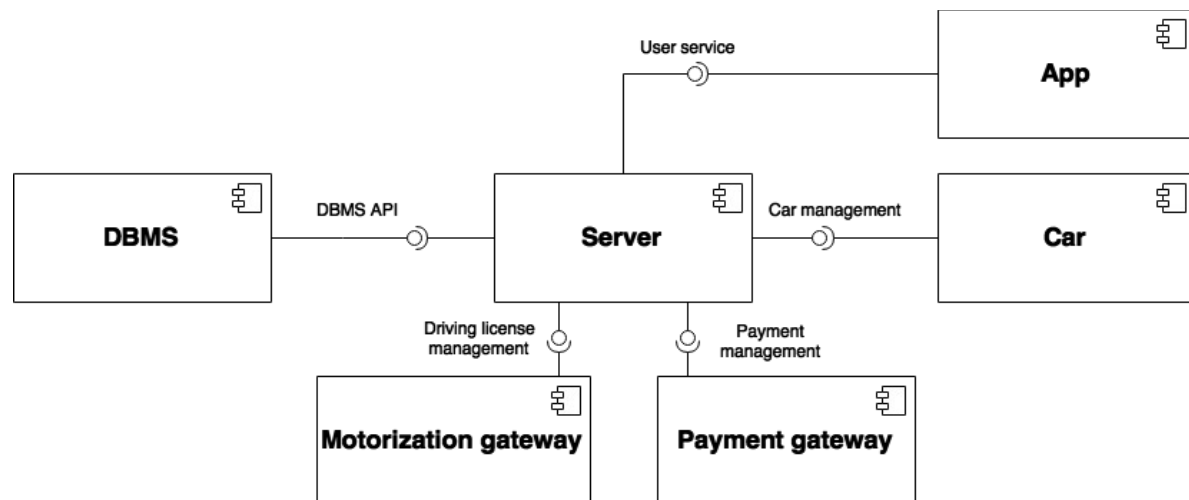
## 2.2 Component view

The architecture of the system has four main components: server, app, cars, and the DBMS. In addition to those, there are also two gateways that allows the server to connect to external services providers.

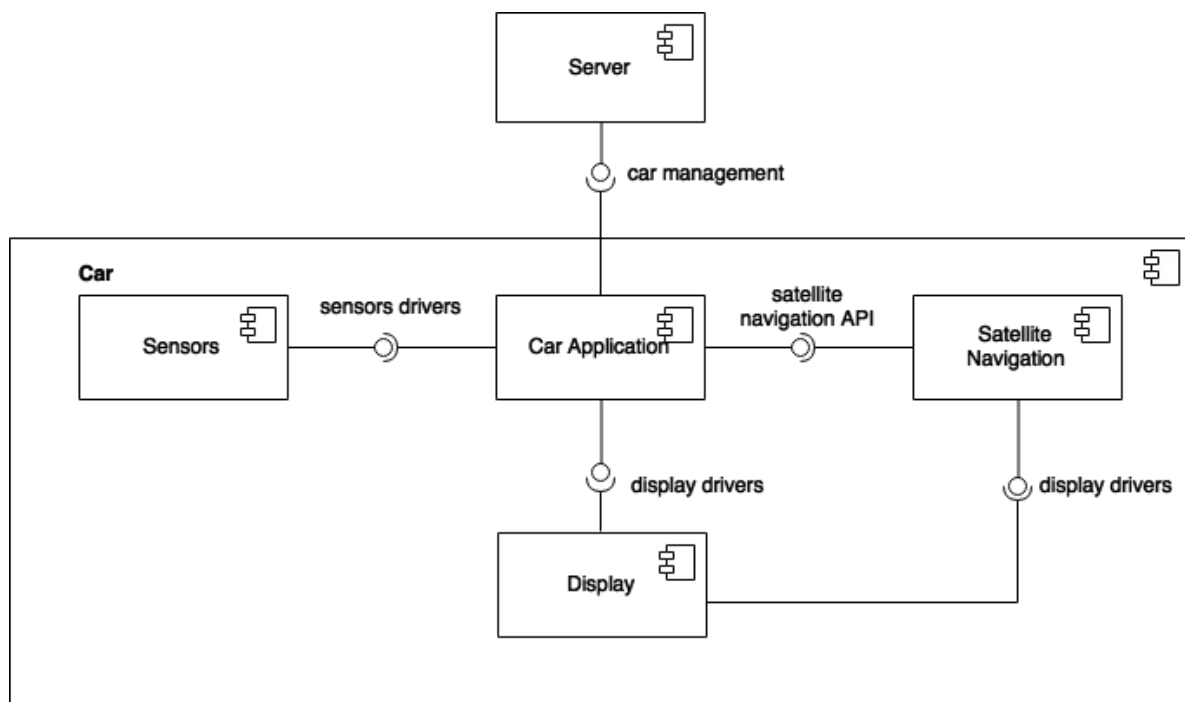
The server manages all the requests coming from clients (both from smartphone and cars) and send requests to operators to have all the services working properly. It also controls all the cars (unlock, lock, check position...).

The app makes clients interact with the server to access some of the services of PowerEnJoy (search and reserve a car, see payment history, unlock the car and so on). It also allows operators to interact with server, receiving notifications, signaling if they take it in charge and the conclusion of the intervention.

The motorization gateway and the payment gateway are used by the server to check information inserted by users, the former for the license id, the latter for the validity of the data inherent the payment method chosen by the user.



### 2.2.1 Car



The architecture of the car is composed by sensors which give information about the number of passengers in the car to the Car Application.

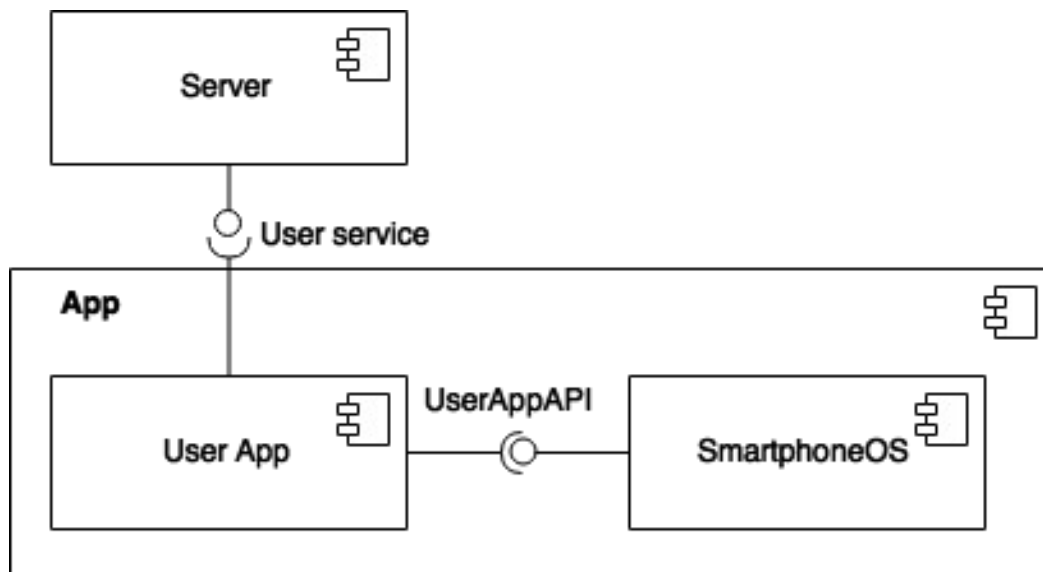
The satellite navigation is connected to the car application to take information about the navigation path to provide.

The display shows to the user information taken from the car application (such as the actual charge and the buttons) and from the satellite navigation (destination to reach and path to follow).

The car application is in charge of communicating with the application server of the system, exchanging data about position, charge, payment and so on.

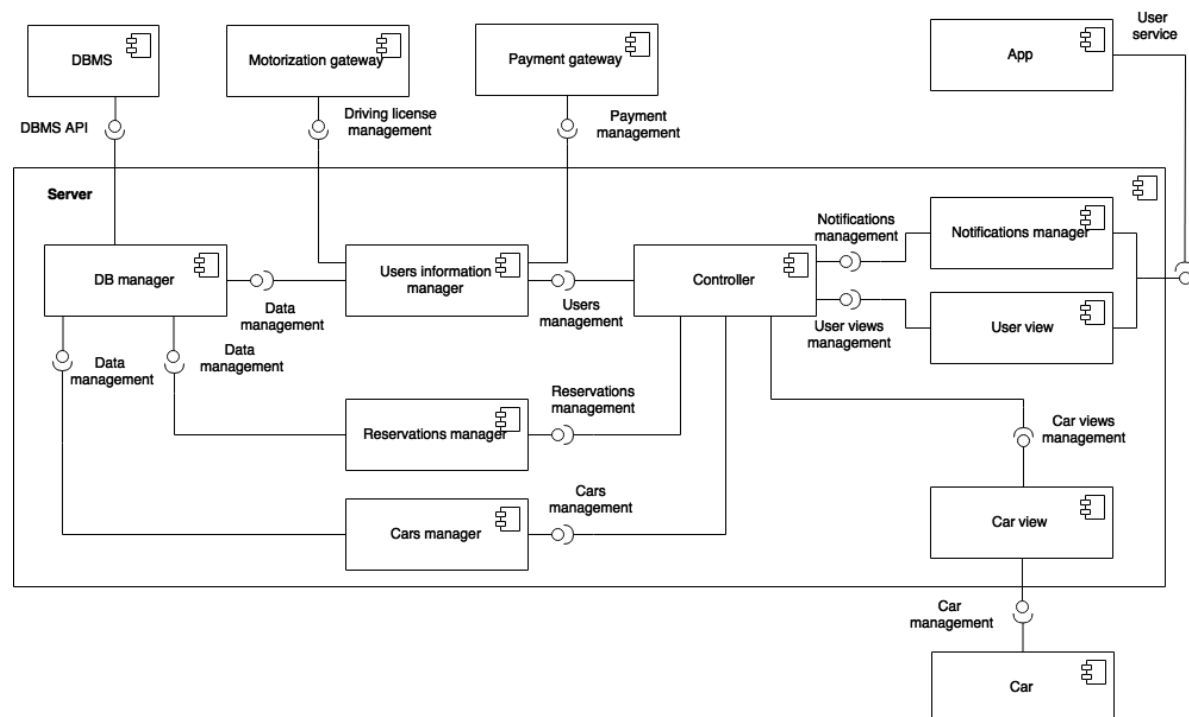


### 2.2.2 App



The architecture of the app results in a simple implementation, in fact it manages the connection to internet and provides all the functionalities using the instruments provided by the OS of the smartphone. In particular the user app has different interfaces and provides different functionalities for the clients and the operators.

### 2.2.3 Server



The architecture of the server is based on MVC pattern, in fact there is a controller component which manages the interactions between components that are part of the model and other ones that are part of the view. In particular notifications manager, user view and car view represent the view, they are all in charge to communicate with the external world receiving and providing informations. Users information manager, reservations manager, cars manager and DB manager (also with the external DBMS) represent the model, they solve different tasks providing each one different functionalities. The users informations manager manages all the informations regarding users, it also communicates with motorization gateway and payment gateway to verify users informations. The reservations manager creates and deletes reservations according to the constraints defined in the RASD. The Cars manager takes care of all the data regarding cars (checks their position, state, battery level...) and activates functionalities (lock, unlock, save money...) when requested. Lastly the DB manager is in charge of creating queries in order to update the DB when the state of things changes.

## 2.3 Deployment view

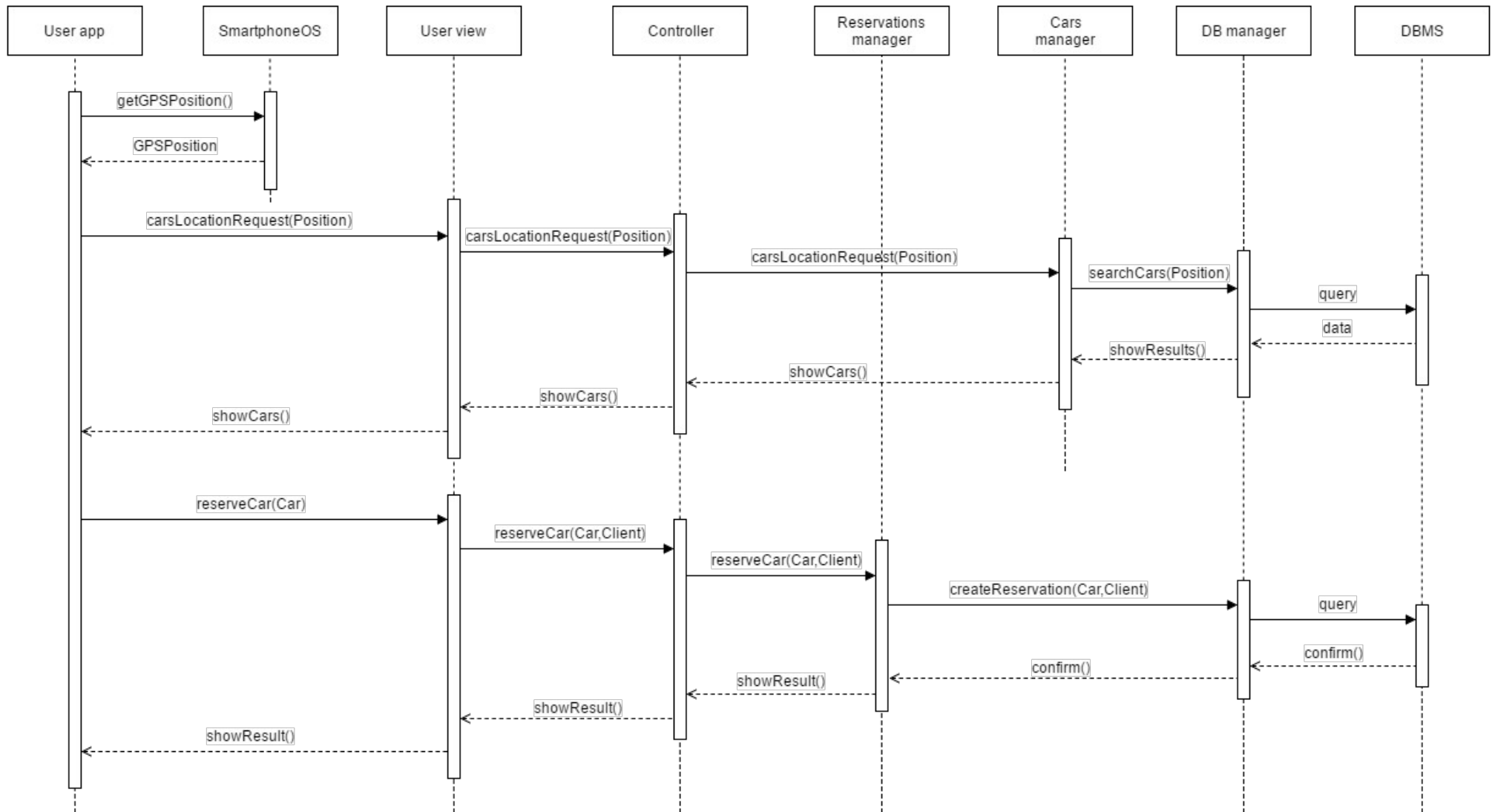
### 2.3.1 Deployment diagram



The main nodes derived by the deployment diagrams reflects the results illustrated in the component view and the intended two tier architecture. In fact the two devices for database and application server will respectively host the relational database management system, containing the database schema, and the PowerEnJoy server application, containing the software interfaces that communicates with the RDBMS, thus represents the tier for the server. The other devices are the ones which communicate with the server, they send requests and use the installed software to provide all the functionalities illustrated in the RASD. As represented there are a mobile application and the car system that together take the role of the client tier.

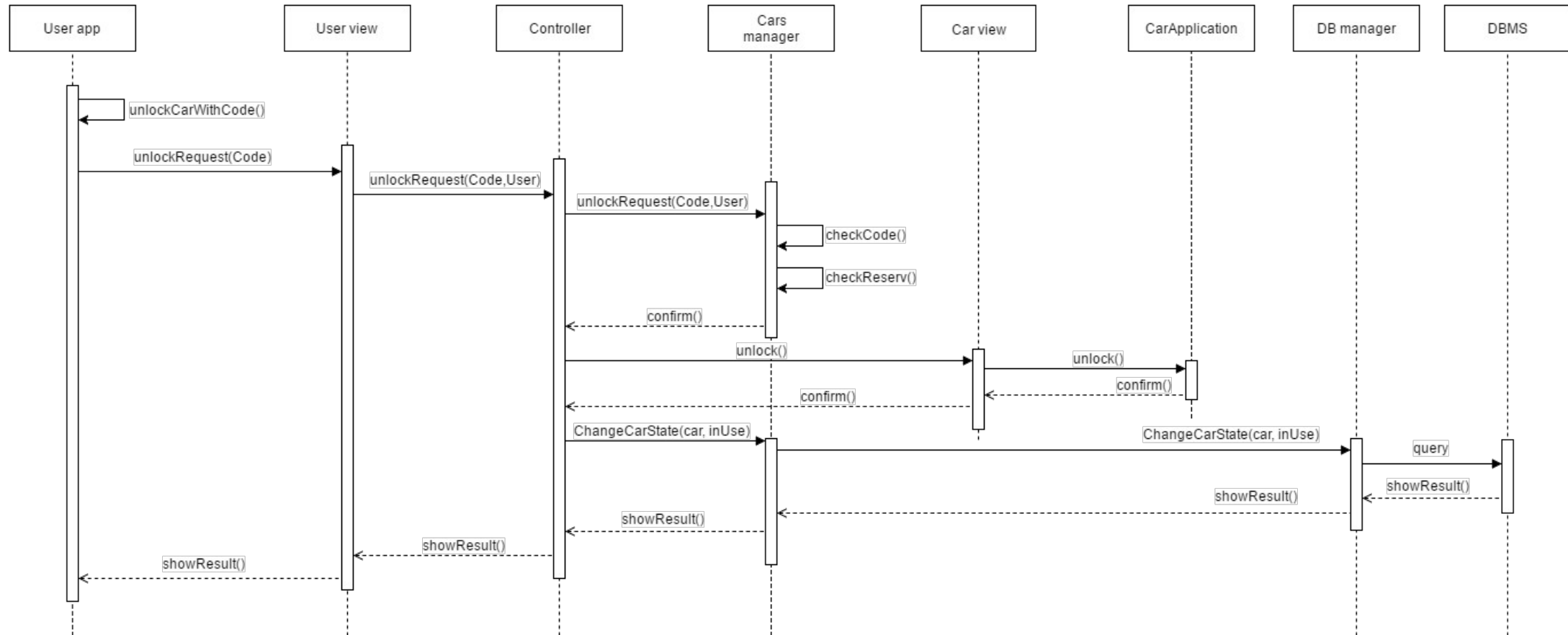
## 2.4 Runtime view

### 2.4.1 Reservation



This diagram shows how a car is reserved, assuming that the client is already logged in the system. The app asks for the current GPS position to the smartphone OS, then asks to the server the available cars near that position. The request is forwarded through various components until reaches the DB manager which retrieves the required informations from the DB and sends the answer which is forwarded back to the app. The client is now able to select a car and reserve it, the reservation request is sent to the server and reaches the cars manager which, through the DB manager, creates the reservation in the DB and changes the state of the car to 'reserved' (these two operations are all included in `createReservation(Car,Client)` method). Once completed these operations the cars manager sends back a confirmation message which reaches the app.

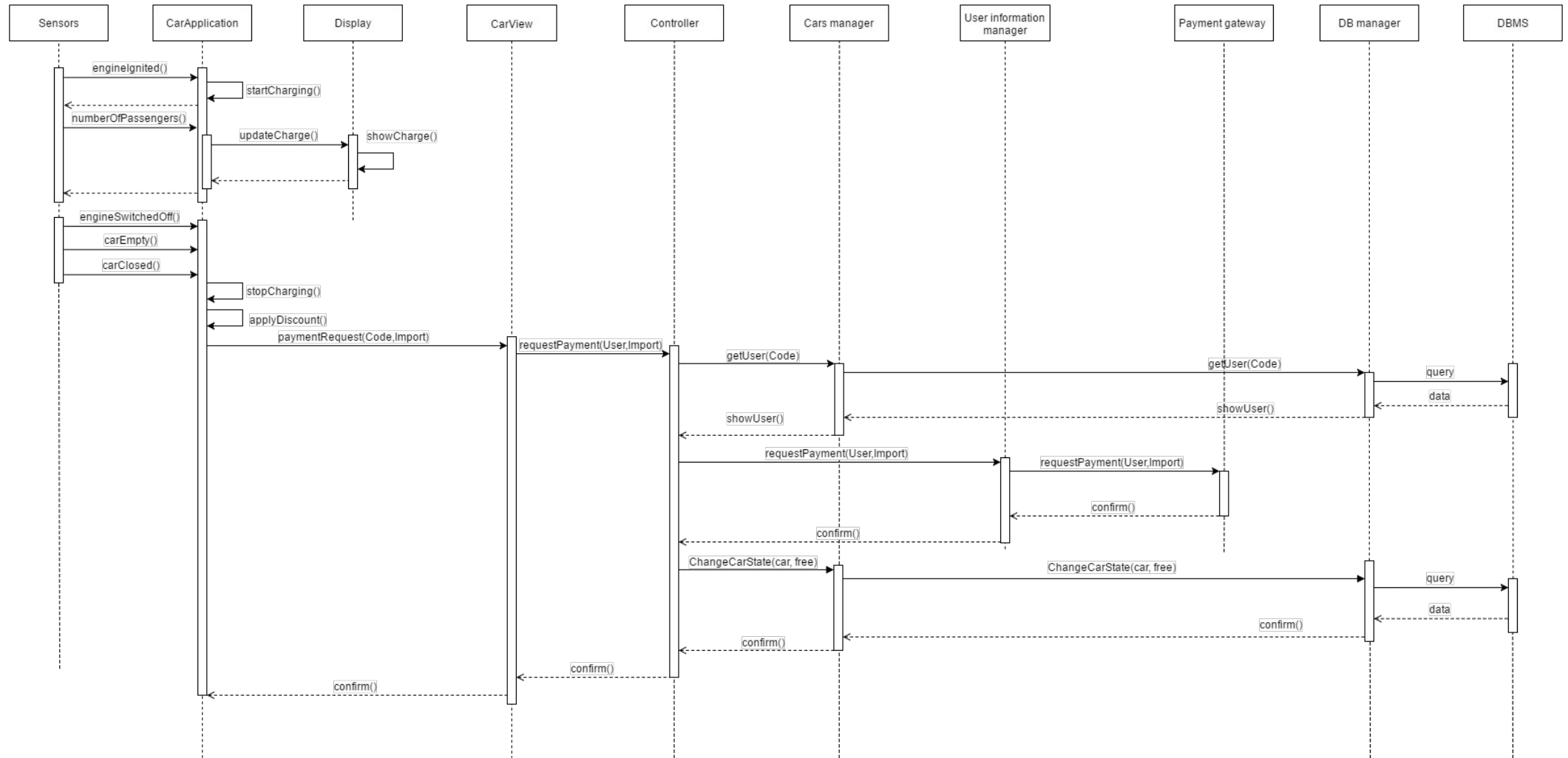
## 2.4.2 Unlock



This diagram shows how a car is unlocked assuming that the client is already logged in the system and is near the reserved car. The client selects within the mobile app the 'Unlock with code' option and he/she inserts it. The mobile app sends a request to unlock the car which is forwarded until it reaches the cars manager. This component, after checking if the code is actually associated to a car reserved by the client, sends a confirmation to the controller. The controller then uses the car view to send an unlock command to the car application which unlocks the car and sends back a confirmation message. The cars manager can now modify the state of the car as 'in use' (through the DB manager). After completing these operations a confirmation message is sent back to the application.

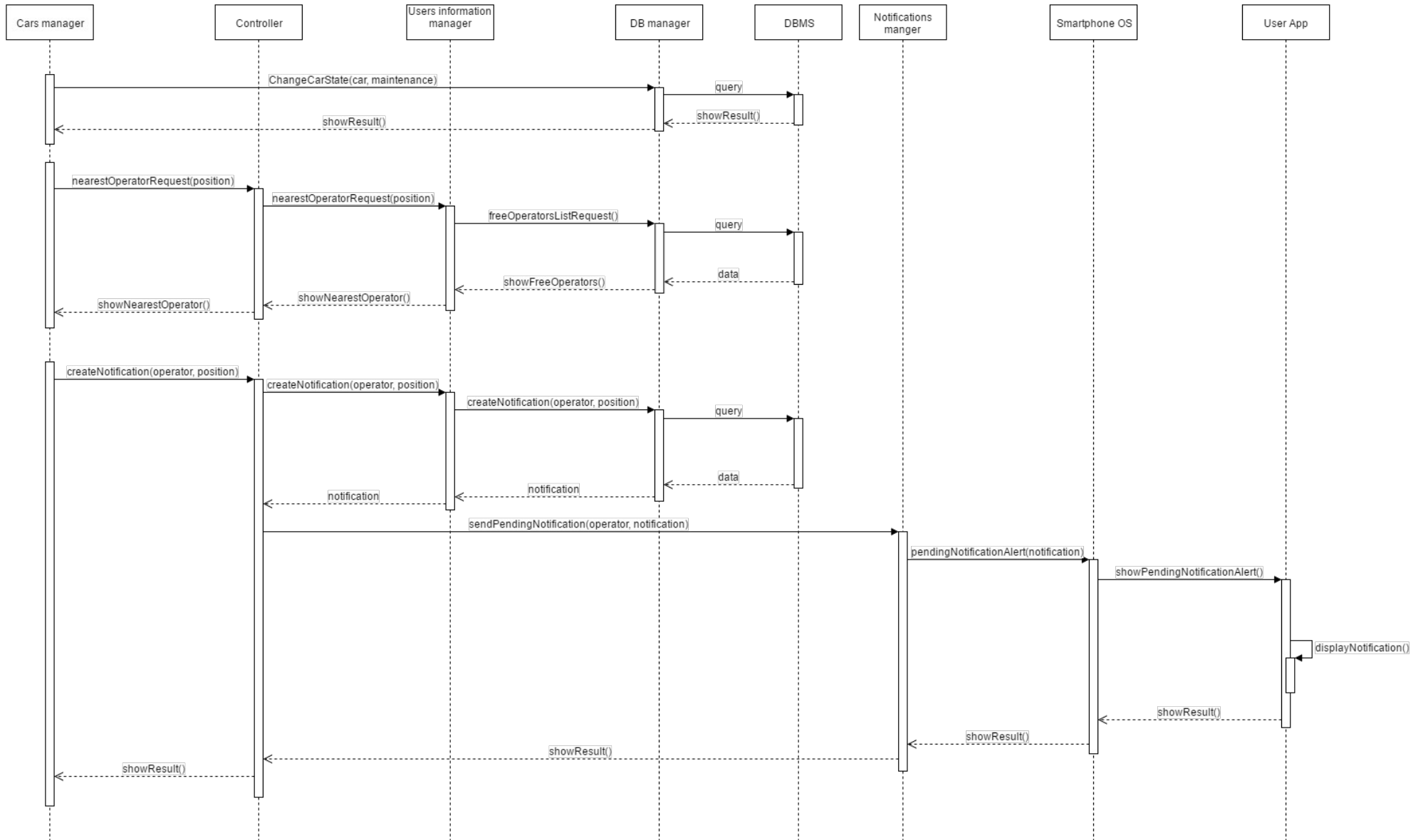


## 2.4.3 Ride



This diagram shows how a ride is registered assuming that the car has already been unlocked and the client is inside. When the engine is turned on a sensor notifies the car application which starts charging the client, then it keeps updating the charge and showing it through the display. When the engine is switched off, the car is empty and the doors are closed each sensor notifies it to the car application, when all these three facts are verified the ride is considered finished so the charge is stopped. When the sensors send the information that the car has been closed they also include informations about the state of the car: position, battery level and 'charging' (true or false) so the car application is able to calculate any discount and apply it, after that it sends the payment request to the server. The car view receives the payment request and moves it to the controller that identifies the user through the cars manager. Then the controller forwards the payment request to the payment gateway using the user account manager, when the confirmation message comes back to the controller it interacts with the DB (through cars manager and then DB manager) to create the new payment in the payment history and set the car state as 'free'.

## 2.4.4 Notification



This diagram shows how the operators are notified by the system when their intervention is needed. The cars manager checks if any car needs maintenance, if so, it first send a request to the DB manager in order to change the car state to 'maintenance', then asks for the operator which is nearest to the car. The user information manager retrieves the requested information from the DB and sends the answer. Now the cars manager is able to request the creation of a notification. Once the notification is created the controller sends it to the notification manager which is in charge of forwarding it to the smartphone app of the operator.

## 2.5 Selected architectural styles and patterns

### 2.5.1 Architectural styles

Architecture	Client-Server (2-tiers)
Use	Component Division: <ul style="list-style-type: none"> <li>- Car, Smartphone (Client)</li> <li>- Application Server, DataBase (Server)</li> </ul>
Reason	The reasons are already explained in the part 2.1 of this document.

Architecture	MVC
Use	Internal server structure: <ul style="list-style-type: none"> <li>- User Information Manager, Reservation Manager, Cars Manager (Model)</li> <li>- Controller (Controller)</li> <li>- Notification Manager, User View, Car View (View)</li> </ul>
Reason	Decouple view and models to provide independent modification, while maintaining the connection between the 2 parts.

### 2.5.2 Patterns

Pattern	Publisher Subscriber
Use	App information update: <ul style="list-style-type: none"> <li>- Mobile App (Subscriber)</li> <li>- User View (Publisher)</li> </ul>
Reason	Message dispatch for cars information with hybrid filtering on content and topic to recognize addressed user.

Pattern	Publisher Subscriber
Use	Notifications dispatch: <ul style="list-style-type: none"> <li>- Mobile App (Subscriber)</li> </ul>

	- Notification Manager (Publisher)
Reason	Message dispatch for notification with content filtering to recognize the addressed operator.

Pattern	Proxy Pattern
Use	DB Communication: <ul style="list-style-type: none"> <li>- User Information Manager, Reservation Manager, Cars Manager (Client)</li> <li>- DB manager (Proxy)</li> <li>- DBMS (Server)</li> </ul>
Reason	Prevent direct access to DB, in order to increase security.

Pattern	Singleton
Use	DB manager (Singleton)
Reason	Provide a global point of access to the DB manager and subsequently to the DB.

Pattern	Wrapper
Use	Payments: <ul style="list-style-type: none"> <li>- Payment Gateway (Wrapper)</li> <li>- User Information Manager (Client)</li> </ul>
Reason	Provide a simplified interface usable by the User Information Manager to perform the payments.

Pattern	Wrapper
Use	License check: <ul style="list-style-type: none"> <li>- Motorization Gateway(Wrapper)</li> <li>- User Information Manager (Client)</li> </ul>
Reason	Provide a simplified interface usable by the User Information Manager to perform the licence informations check.

Pattern	Observer-Observable
Use	View-Controller connection: <ul style="list-style-type: none"> <li>- Controller (Observer)</li> <li>- Notification Manager, User View, Car View (Observer, Observable)</li> </ul>

	<ul style="list-style-type: none"> <li>- User Information Manager, Reservation Manager, Cars Manager (Observable)</li> </ul>
Reason	Realize MVC implementation.

Pattern	Observer-Observable
Use	Sensors signals recognition: <ul style="list-style-type: none"> <li>- Car Application (Observer)</li> <li>- Sensors (Observable)</li> </ul>
Reason	Provide a connection between the application inside the car and the sensors.

Pattern	Wrapper
Use	Satellite navigation: <ul style="list-style-type: none"> <li>- Car Application (Client)</li> <li>- Satellite Navigation (Wrapper)</li> </ul>
Reason	Provide a simplified interface usable by the Car Application to interact with the Satellite Navigation.

### 3. Algorithm design

Here follows 2 samples for the intended algorithms that will be used for the creation of a reservation and in the research for nearest free operator. We do not report complete code.

#### 3.1 Reservation creation

```
public Message reserveCar(Car car, Client client){
    //[check correctness of parameters]
    Reservation newReservation;
    try{
        newReservation = reservationsManager.createReservation(car, client);
    }
    catch(Error error){
        return createErrorMessage(error);
    }
    return createReservationMessage(newReservation);
}
```

```
public Reservation createReservation(Car car, Client client) {
    if (!car.reservable())
        //[error message]
    List<Reservation> activeReservations;
    //[initialization]
    boolean alreadyOneReservation = false;
    for (Reservation r : activeReservations) {
        if (r.getClient().equals(client))
            alreadyOneReservation = true;
    }
    if (!alreadyOneReservation) {
        Reservation reservation = new Reservation(car, client);
        carsManager.changeCarState(car, new ReservedCarState);
        startTimer(reservation);
        return reservation;
    } else
        //[error message]
}
```

## 3.2 Nearest operator

```
public class Haversine{
    public static final float R = 6372.8; // In kilometers

    //Calculates the distance between two coordinates
    public float haversine(Position pos1, Position pos2) {
        float lat1, lon1, lat2, lon2;
        lat1 = pos1.latitude;
        lon1 = pos1.longitude;
        lat2 = pos2.latitude;
        lon2 = pos2.longitude;
        float dLat = Math.toRadians(lat2 - lat1);
        float dLon = Math.toRadians(lon2 - lon1);
        lat1 = Math.toRadians(lat1);
        lat2 = Math.toRadians(lat2);

        //This formula is used to calculate the distance between two points situated
        on //a spherical surface
        float a = Math.pow(Math.sin(dLat / 2),2) +
            Math.pow(Math.sin(dLon / 2),2) * Math.cos(lat1) * Math.cos(lat2);
        float c = 2 * Math.asin(Math.sqrt(a));
        return R * c;
    }
}

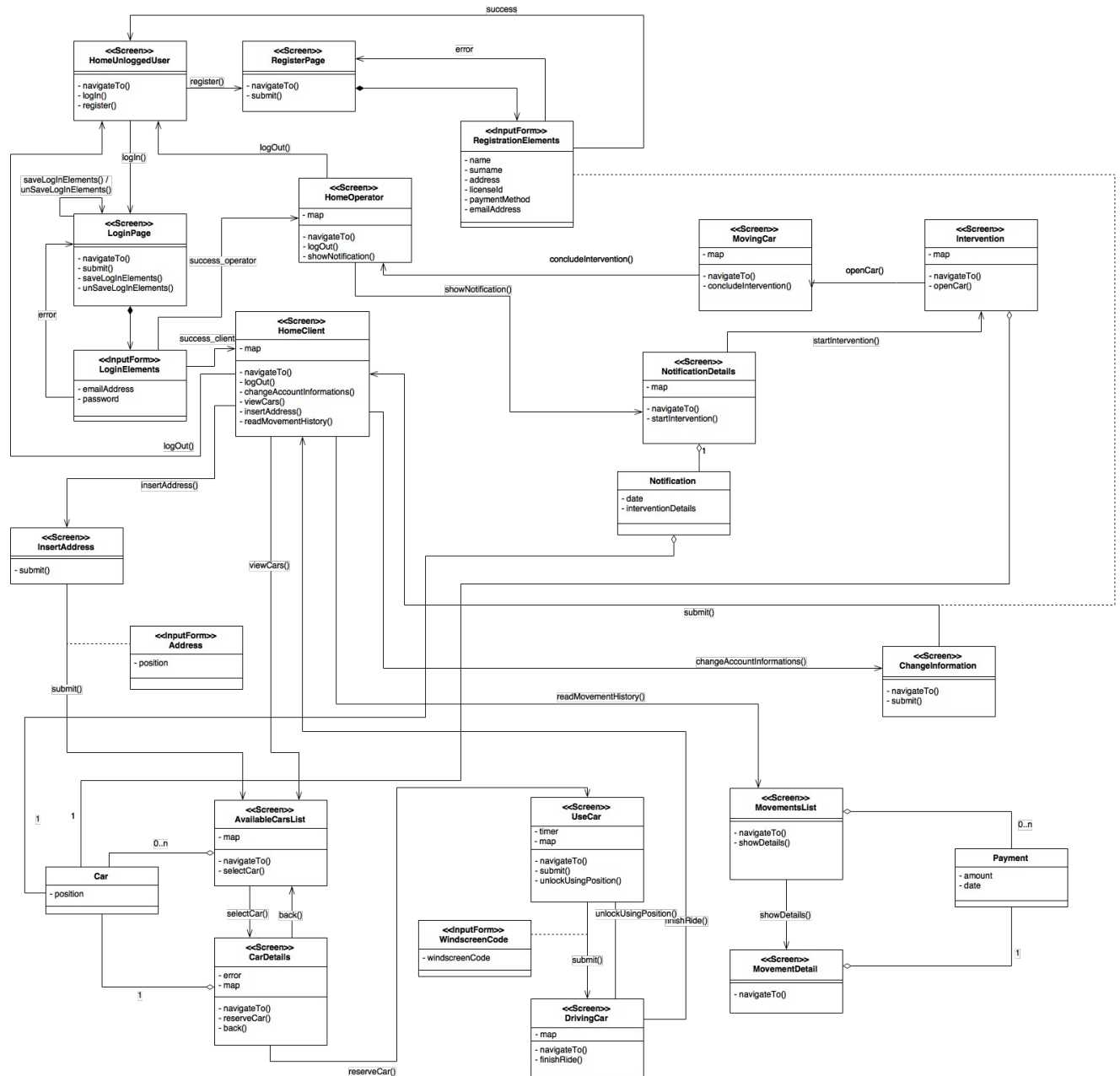
//previously it is checked if there are no free operators, in such a case the notification is
//enqueued. This allows us to assume that when this method is called there is at least one
//free operator
public Operator nearestOperator(Car car, List<Operator> workingOperators){
    Operator nearestOperator;
    float minimumDistance;
    for(Operator o : workingOperators){
        if(!acceptedNotification(o)){ //operator is not already working on a notification
            float distance = Haversine.haversine(car.position, o.position);
            if(nearestOperator == null || minimumDistance > distance){
                nearestOperator = o;
                minimumDistance = distance;
            }
        }
    }

    return nearestOperator;
}
```

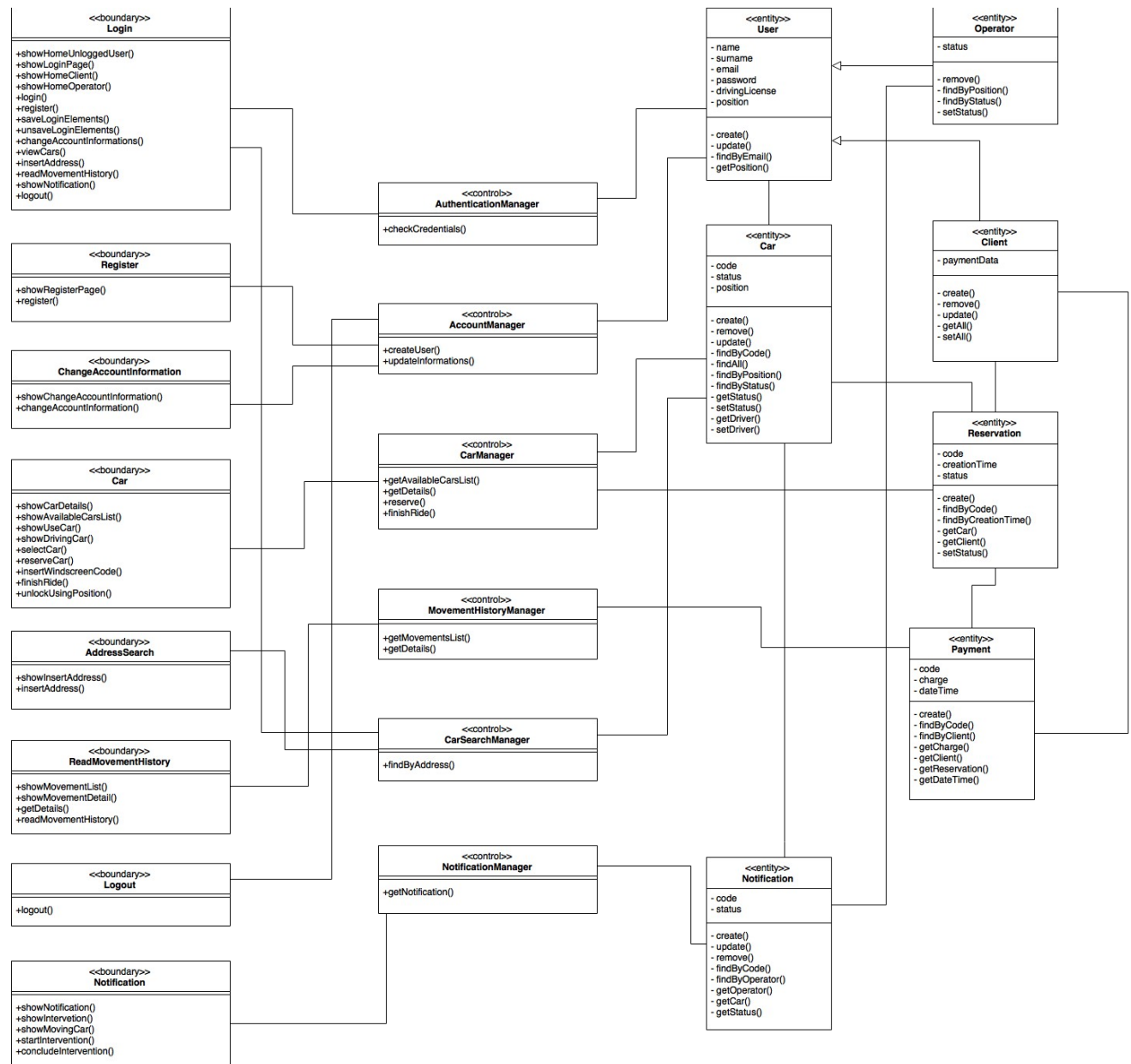


## 4. User interface design

### 4.1 UX diagram




## 4.2 BCE diagram

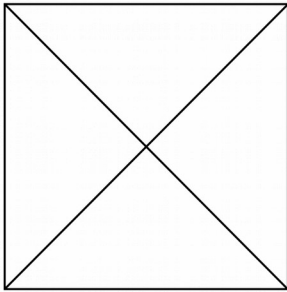
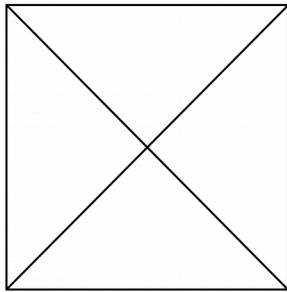
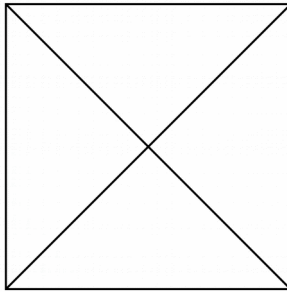


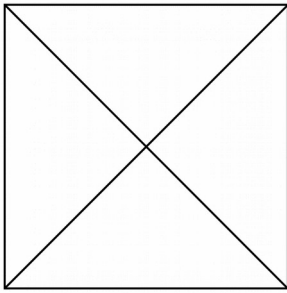
## 4.3 User Interface mockups

### 4.3.1 Unlogged user interface

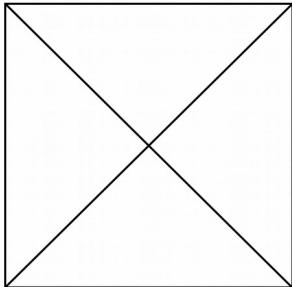
  <div>Log In</div> <div>Register</div>	Register	Log In
	<div>Name</div> <div></div> <div>Surname</div> <div></div> <div>Email</div> <div></div> <div>Address</div> <div></div> <div>License ID</div> <div></div> <div>Payment method</div> <div></div> <div>Register</div>	<div>Email</div> <div></div> <div>Password</div> <div></div> <div>Log In</div> <div><input type="radio"/> Keep me Logged In</div>

### 4.3.2 Operator interface

Home	Notification	Intervention
	 <div>INTERVENTION POSITION: position_A REMAINING BATTERY: battery_level TIME TO REACH: time DATE: dd/mm/yyyy DETAILS: details</div>	
<div>View notification</div>	<div>Start intervention</div>	<div>Open car</div>
<div>Log Out</div>		

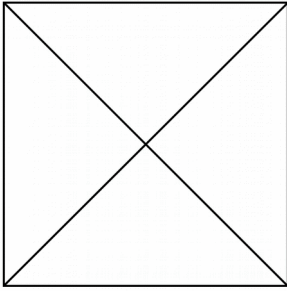
Moving car

<div>Conclude Intervention</div>

### 4.3.3 Client interface

Home	Chage account info	Movements hisotry
<div></div> <div><a href="#">View cars</a></div> <div><a href="#">Insert address</a></div> <div><a href="#">View movements history</a></div> <div><a href="#">Change account info</a></div> <div><a href="#">Log Out</a></div>	<div><div>Name</div><input type="text"/></div> <div><div>Surname</div><input type="text"/></div> <div><div>Email</div><input type="text"/></div> <div><div>Address</div><input type="text"/></div> <div><div>License ID</div><input type="text"/></div> <div><div>Payment method</div><input type="text"/></div>	

[Change](#)

## Car

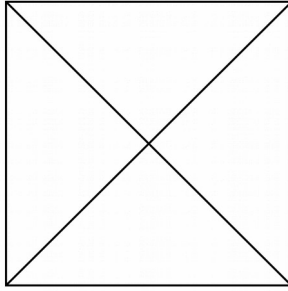


CAR\_1  
POSITION: position\_A  
REMAINING\_BATTERY: battery\_level  
TIME\_TO\_REACH: time

Reserve

Back

## Use car



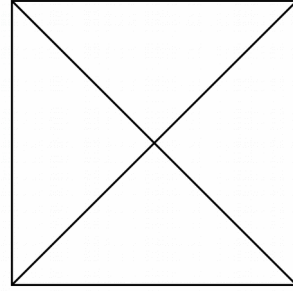
Remaining time: mm/ss

Unlock with position

Windscreen code

Unlock with code

## Driving car



## 5. Requirements traceability

This document is made to design a software which functionalities have been described in the RASD. The list that follows associates the requirements of the system with the component of the architecture described above that accomplishes it.

[G1] Allows clients to register in the system

- The system must accept the registration only if all the required fields are filled
- The system must save client's data in the database
- The system must send a request to the motorization to check the validity of the driving license
- The system must send a request to the payment system to check the validity of the payment method
- The system must show an error message if the inserted data are wrong or the fields are filled incorrectly

The User App will offer to the client the interface to fill the registration form, once the form is completely filled it is sent to the server. Information on the server are received by the User View and passed to the User Informations Manager via the Controller. Here the User Informations Managers will check informations correctness, in particular will use the Motorization Gateway and the Payment Gateway to control respectively the validity of the driving license and the validity of the payment method. In the case of one of the informations results incorrect or not valid the previous sequence of components will be retraced in order to inform the user of the error, otherwise user informations will be passed to the DB manager to store them inside the DataBase.

[G2] Allows users to login

- The system must show an error message if the credentials are wrong
- The system must permit client to access to clients' area if the credentials are correct

In the same way of the registration the login elements are passed to the User Information Manager starting from the User App, there the User Information Manager uses the DB manager to retrieve the informations regarding the account from the DataBase, checks the correspondence between sent and saved informations and, in case of success send the success message and informations to the User App, while in case of incorrect informations sends an error message.

[G3] Allows clients to find available cars

- The system must show the map with the available cars nearby if the GPS is turned on
- The system must show the available cars near the position inserted by the client
- The system must show an error message if the inserted position is not valid
- The system must refresh the map if a car becomes unavailable or if a car becomes

available

The client can search for available cars through User App, the search can be based on GPS or inserted position, in both cases the Controller starts the procedure that proceeds as described in the section 2.4.1 of this document.

[G4] Allows clients to reserve only one car at a time

- The system must show a confirmation message if the reservation has been completed successfully
- The system must show an error message if the client has already reserved another car
- The system must save the time at which the reservation has been carried out

Every time a client makes a registration this is stored in the DB using the Reservation Manager. In the case a client tries to reserve more than one car at the time the Reservation Manager will simply prevent him from doing so, in fact before creating every reservation it uses the DB manager to find active reservations made from the client that is trying to make the new reservation.

[G5] Checks reservation time limit

- The system must delete the reservation if the car isn't picked up within one hour
- The system must mark a car as available when its reservation expires

Once a reservation is started the Reservation Manager inside the Server starts a timer. In case that the timer finishes before the car is picked up, the Reservation Manager first deletes the active reservation and marks the car as available again through the DB manager and then warns the Controller about the event.

[G6] Allows clients to get into the reserved car

- The system must let the client choose if he/she wants to unlock the car according to his/her position or inserting the code on the windscreen
- The system must unlock the car if the client has chosen the position option and the detected position is near the position of the car
- The system must unlock the car if the client has chosen the code option and the inserted code is correct
- The system must show an error message if the client has chosen the code option and the inserted code isn't correct
- The system must show an error message if the client has chosen the position option and the detected position isn't near the position of the car

When a user tries to unlock the car using the User App a message is sent to the Controller passing through the User View so that the Cars Manager can unlock the car communicating with it through the Car View, the whole procedure with the detailed calls is described in the part 2.4.2 of this document.



[G7] Charges the client according to the duration of the ride

- The system must start charging the client when the engine ignites
- The system starts charging the client anyway if he doesn't ignite the engine within one minute
- The system must increase the charge every minute

Once the car gets unlocked the Car Application starts a timer of one minute waiting the client to ignite the engine. If the timer finishes or if the client ignites the engine before the Car Application start calculating the charge updating it every minute till the end of the ride. once the ride is ended it informs the server about the total charge sending the data to the Car View that starts the payment procedure through the Controller.

[G8] Allows the clients to know the actual charge

- The system must show the charge for the ride on the screen inside the car
- The system must update the actual charge during the ride as stated in the point 3 of [G7]

As stated for the goal [G7] is duty of the Car Application to calculate the actual charge, while doing so, it will display the current value on the screen of the satellite navigation

[G9] Identifies correctly the duration of a ride

- The system must be able to recognise whether a car is in a safe area or not
- The system must display an error message on the screen of the satellite navigation if the car is switched off outside a safe area
- The system must advice with a beeper that the door of the car has been opened in an unsafe area
- System must lock the car if the ride is finished
- System must stop the charging when the ride is finished and the car is in a safe area
- System must consider a ride finished if and only if the car is left empty inside a safe area

When the user stops the engine the Car Application sends the information to the Car View inside the server together with the position retrieved by the Sensors, then the Car View informs the car Manager Using the Controller. The Car Manager controls if the position is inside a safe area or not using the informations from the DataBase retrieved by the DB manager. If the car is inside a safe area waits from the Car Application the information for the empty car, then it updates the DB using the DB manager and informs the Controller. Otherwise it informs the Car Application, via Controller and then Car View, that the car is Outside a safe area. At this point the Car Application uses the Display to warn the client about the situation. If the Car Application receives from the Sensors the information about an opened door it starts beeping. If the Sensors informs the Car Application that the door is closed and the car is empty the Car Application locks the car and informs the Car Manager in the server of the situation so it can start the maintenance procedure as

described by the 2.4.4 of this document. Instead if the Sensors registers a closed door but not an empty car the drive resumes to the normal status and the Car Manager is informed by the Car Application, using the Car View as always.

[G10] Rewards or punishes clients according to their behavior

- System must charge a fee of 1€ to the client in case the car he reserved is not picked up within an hour from the reservation
- System must apply a 10% discount on the total cost of the ride if the client takes at least other two passengers
- System must apply a 20% discount on the total cost of the ride if the car is left with no more than 50% of the battery empty
- System must apply a 30% discount on the total cost of the ride if the car is left at a power station and the client takes care of plugging the car into the power grid
- System must apply a 30% additional fee on the total cost of the ride if the car is left at more than 3 KM from the nearest power grid station or with more than 80% of the battery empty
- System must inform the clients of their rewards/punishments through the app

Once a ride is finished and the Car Application calculates the eventual discount or fee on the total cost and sends the complete informations about the ride to the Car View that informs the Car Manager using the Controller, the Car Manager gives the informations to the Controller that informs the User information Manager of the payment. The User Informations Manager retrieves the informations payment information from the DB using the DB manager and uses the Payment Gateway to execute the payment. When the payment procedure is completed the ride is added in the DB using the DB manager, this procedure is shown in the part 2.4.3 of this document.

[G11] Notifies operators when their intervention is needed

- The system must alert the nearest operator when a car is left for more than one minute empty and switched off out of a safe area
- The system must notify the nearest operator when a car is damaged or has a breakdown
- System must lock the car while waiting the operator
- System must let only the notified operator enter inside the car
- The system must notify the nearest operator when a car is left for more than one minute with more than 80% of the battery empty informing him/her if there is a reachable charging station or if it's necessary to recharge on-site
- The system must notify the nearest operator 3 minutes after a car is left in a charging station without the plug
- System must add the notification in a queue if there are no available operators

When the Car Management receives from the Car View, in turn informed by the Car Application, the information about a car that needs maintenance it reacts according to the case: if the car results left outside a safe area or left with more than 80% of battery empty a 1 minute timer starts, if the car results in a charging station without the plug a 3 minutes timer starts, in case the timer finishes the Car Manager starts the notification procedure

described in the section 2.4.4 of this document. The notification procedure starts immediately if the car results damaged or had a breakdown.

[G12] Allows client to use money saving option

- System must display 'Money Saving' button on the screen of the car
- System must require the destination to the client when the money saving option has been selected
- System must calculate and sort charging areas near to the destination
- System allows client to select the desired charging station
- System provide assistance to navigation to reach the destination
- System displays an error message if the provided address for the destination is wrong

The Car Application shows an icon for the save money option on the Display inside the car, if the icon gets clicked the user will be asked to insert the destination address, when the address is inserted the Car Application will require to the Car View in the server the list of nearest charging areas. The Car View uses the Controller that subsequently uses the DB manager to retrieve informations about charging areas, identifies a list of nearest and sends it to the Car Application. At this point the Car Application shows the possible choices on the display and when the user selects one destination the Car Application starts the Satellite Navigation to show to the driver the path to follow.

## 6. Used tools

- Draw.io Diagrams: UML, Use Case, Sequence, State Chart diagrams
- Google Drive: documents sharing
- Google Docs: word processor, concurrent work platform
- GitHub: control version
- Photoshop CC: UI design

## 7. Hours of work

Perugini Alex:

- 29/11/16 30m
- 30/11/16 3h 30m
- 01/12/16 2h
- 02/12/16 4h
- 07/12/16 2h 30m
- 08/12/16 2h 30m

Re Marco:

- 25/11/16 45m
- 27/11/16 1h
- 30/11/16 3h 30m
- 02/12/16 1h
- 03/12/16 2h 30m
- 04/12/16 2h 15m
- 05/12/16 1h
- 06/12/16 3h 30m
- 08/12/16 1h 30m

Scotti Vincenzo

- 25/11/16 40m
- 28/11/16 1h30m
- 29/11/16 1h
- 30/11/16 3h 30m
- 02/12/16 1h 30m
- 03/12/16 1h 10m
- 05/12/16 2h 30m
- 06/12/16 4h
- 09/12/16 3h