# Politecnico di Milano

# A.A. 2016-2017

# Software Engineering 2: PowerEnJoy

# **D**esign **D**ocument

*Lorenzo Frigerio (mat. 879203)*

*Martina Perfetto (mat. 808869)*

*Ivan Vigorito (mat. 879204)*

11th December 2016

*Version 1.0*

# Contents

# 1. Introduction

## 1.1. Purpose

This document describes the specific architecture and design of “PowerEnJoy”.

The focus will be on structural and design styles choices, expanding the thread already analysed in RASD document.

The design document in effect, starting from the requirements given in the RASD build a feasible architecture for the application.

## 1.2. Scope

The document will present different level views in order to describe clearly the architecture of the application. In particular, will be presented the component view, both high and low level, the deployment view, the runtime view and a further description of user interface, analysed in its runtime flow.

## 1.3. Glossary

User / registered user: he/she is the client of the service; he/she is able to rent a car in order to travel around the city. He is associated with:

-Name

-Surname

-Other personal information

-Method of payment

-Number of driving licence and expiration date

-Password

Employee/operator: is the one that help users in case of emergency and has the responsibility of managing cars in case of malfunction. Users can call them by using the telephone exchange of the application.

Method of payment: is inserted by the user during the registration phase but can be updated over the time. Only one method is active at once and payment are concluded using services offered by the different companies holding the credit card. An invoice containing all the charges collected is generated monthly.

Car: sometimes referred as vehicle is the means of transport rented by users. It contains a set of sensors that analyse the number of passengers presents on the car, control the charge of the battery and detect when a door is closed. Moreover, it includes a module that transmit this information to the system using the Internet.

Available car: car that is not in use at the moment by any user, has at least 20% of charge and is not reserved by anyone.

Reservation: made by a user that wants to use a car. Has a duration of one hour maximum and is associated with a unique car. Once the user asks to unlock the car the car becomes associated to the user until he decides to end the ride.

Charge: amount of money that users have to pay due to the use of the service. It is immediately calculated by the system after a ride but money is transferred only at the end of the month.

Penalty: fee derived from a bad behaving of the user such as a damage on the car or a fine for exceeding speed limits. The fee will be notified to the user and included in the monthly invoice.

GPS navigation device: system that equip each car and that is able to calculate the exact position of the car and display to the user the route to follow. Its display is also used to show the current fee of the ride and the status of the battery.

Start/stop button: device present in all the electric cars that allow the engine to ignite when the car is unlocked. It also allows the engine to stop when the user wants to get off the car.

Special Safe Area: special parking areas that contain plugs that allow cars to be recharged. They are provided with sensors that detect the number of spots that are

currently used and communicate the number to the system. They are also called power grid stations.

Safe Area: Space included in boundaries that determine where users can park a car. It covers entire metropolitan cities in order to facilitate users to find a park and they may also contain power grid stations. Users cannot terminate a ride while outside from a safe area.

Ride/Rental: it last from when the user picks up the car until when the system stops charging the user. It includes a possible set of temporary stops and the total path travelled by the user.

Distance: It is the lengthiness of road which is covered by the user with the rented car.

Park: is when a user leaves the car and wants to end the rental. At this point the system stops charging the user.

Stop: is when a user leaves the car but wants to resume the ride in the future. The car will be locked by the system that, however, will continue to charge the user for the ride.

### 1.3.1. Acronyms

SPA: Special Safe Areas

SA: Safe Areas

## 1.4. References

- Requirements and Specification Document, RASD
- IEEE Standards for Information Technology Systems, Design Document

## 1.5. Document Structure

The Design Analysis is based on a Top-Down approach; therefore, the Document structure will follow the same path. It will start from the high-level architecture, presenting the main components with their operations and mutual relations (2.2). Then it slides down to a lower level in which the high-level components are decomposed and analysed in detail (2.3).

After that in 2.7 paragraph will be presented the Deployment view of the system that show the execution architecture of the system representing the deployed software and hardware artefacts.

Runtime view presented in 2.8 paragraph will show the behaviour of the system during some typical situation. That will permit to understand in an easier way how the execution flow works.

The lowest level of analysis is reached in paragraphs 2.4 and 2.5 where are described the inner composition of the Interfaces and the Architectural choices which have been made to design the system.

Lastly, will be presented some of the main algorithms that are the fulcrum of the entire system. They will be described through pseudo-code.

Chapter 4 will take the User Interface already presented in RASD and give further information about the design choices. It will also describe some UI flows clearing up the navigation through different web pages or mobile screens. The paragraph about User Experience will get over the mere “Look Requirements” presented in RASD and it will deeply analyse the structure of the Web Application.

The last chapter presents the architectural choices about the Database structure. In particular, will be presented the E-R diagram and the correspondent relational model.

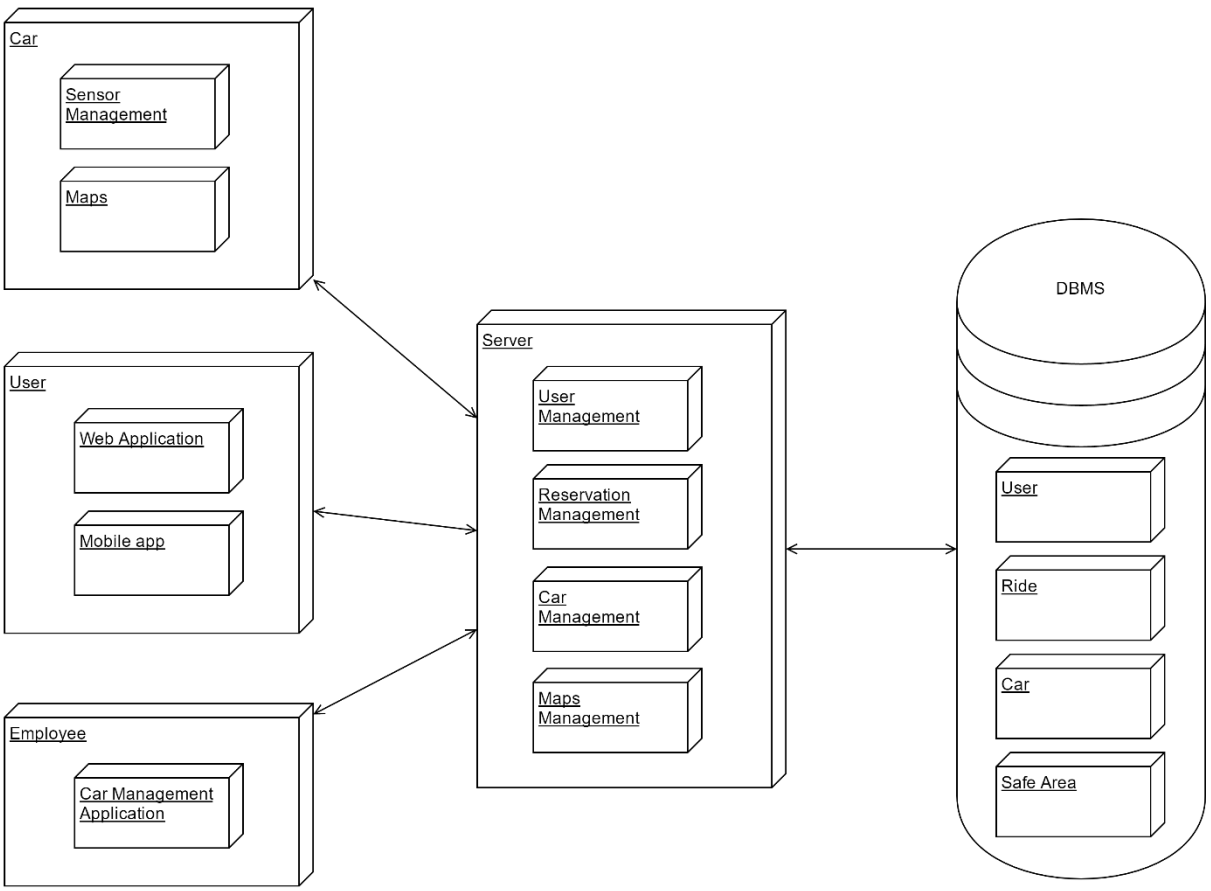
# 2. Architectural Design

## 2.1. Overview

The architectural design of the application is based on three different tiers, which are the CLIENT-tier, the BUSINESS LOGIC-tier and the ENTITY-tier:

- CLIENT-tier: It is represented by the web application and the mobile app used by the users as part of a thin tier. In addition, there is also the employee application that allows operators to manage the maintenance and the problems related to the cars. Lastly there is the application running on the car. This app allows the management of the display present on the car and the sensors used to verify the presence of discount and communicate them to the system.

- BUSINESS LOGIC-tier: It is the core of the application and plays as a mediator between the requests of users and employees and the cars. Moreover, it manages the reservation process and the safe areas.
- ENTITY-tier: It contains the manager of the data. It is responsible of database modification such as the insertion or the retrieving of any kind of data.



## 2.2. High level components and their interaction

The following diagram illustrates a high level architecture of the system. It shows how communication between the different clients and the central system are managed.


CAR-Map Management: this communication allows the cars to retrieve the information about maps; Indeed, the car may need the path to follow in order to apply the “Money Saving Option”. Cars are provided with the latest version of the position of safe and special safe areas that is updated regularly.

CAR-Car Management: this communication grants the possibility for cars to communicate some information to the system. In fact, cars tell the system when all the users are exited from the car and asks to the system the confirmation to lock.

Car Management-CAR: the system has the chance to ask to the car if a discount can be applied and in order to get the position of the car.

Employee-Car Management: the employee has the opportunity to manage the maintenance and all the problem related to the cars. Through a web application they can ask to change the status of a car if under maintenance or check their positions.

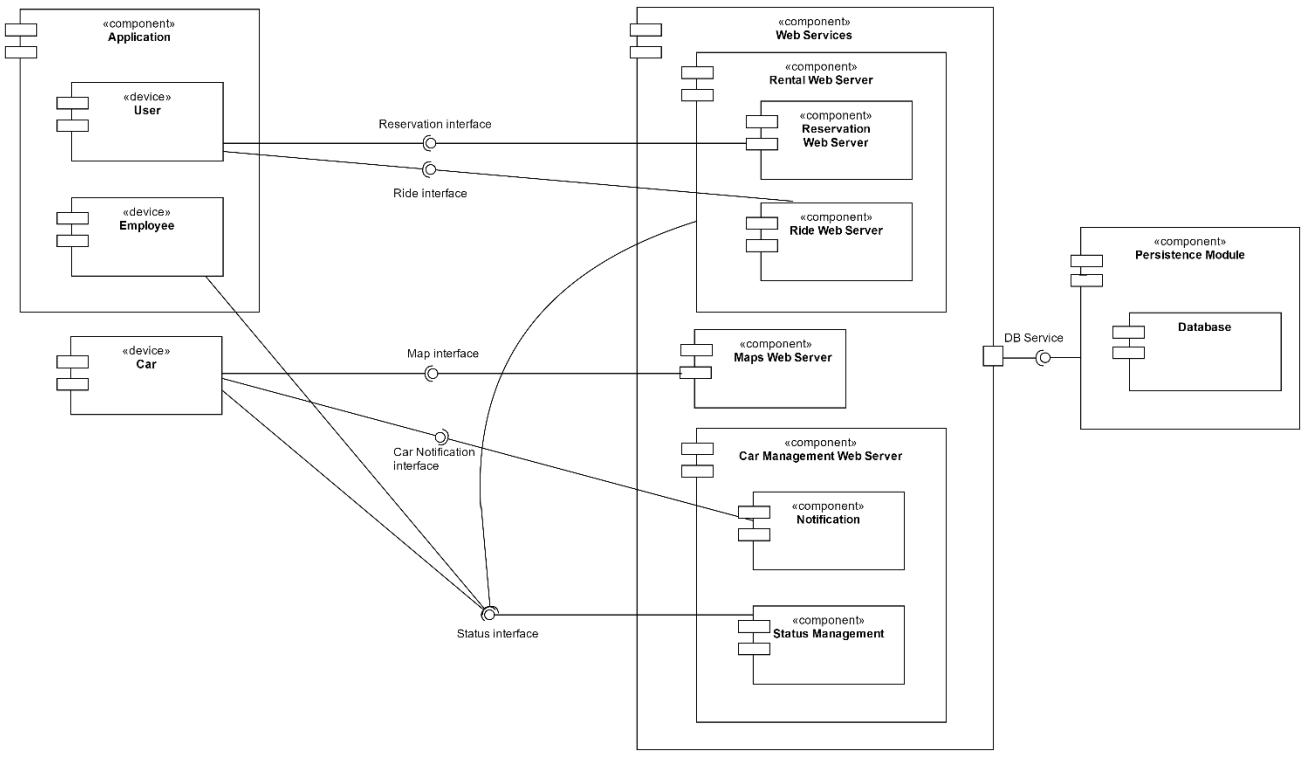
User-User management: users can ask to the user manager to perform a login or to update their personal information. Moreover, they can insert or update the method of payment in case it is expired.

User-Reservation management: after the login the user can ask for a reservation by selecting on the available cars proposed by the system. In addition this block performs the operations related to the rides such as the begin of a ride (unlockCar()) e its end (endRental()).

## 2.3. Component View

Now the High-Level Components view presented above will be decomposed and analysed in detail. In particular, they will be highlighted the subcomponents and how they behave. A component behaviour is described by provided and required interface, and its relation with other components.

## 2.4. Component Diagram: Web Services



The component diagram is constituted of three macro components:

APPLICATION: it contains all the components related to the web application and the mobile app used by the user and the Car Management Application of the employees. It is mainly designed as a presentation component with very little logic. In fact, it allows user to make request to the system and to display to them all the notifications. The employee component offers a different user experience in order to allow them to manage cars in case of problems.

CAR: it is the component installed on each car that manages its functioning. It is composed of a microprocessor that administer a set of sensors placed on the car. The information collected are sent to the system through a LTE modem connector. Moreover, it manages the display of the car that shows the path to follow and the information related to the ride and the status of the car (e.g. the level of the battery).

WEB SERVER: It is the main macro component that manages most of the logic of our system. Its subcomponents are:

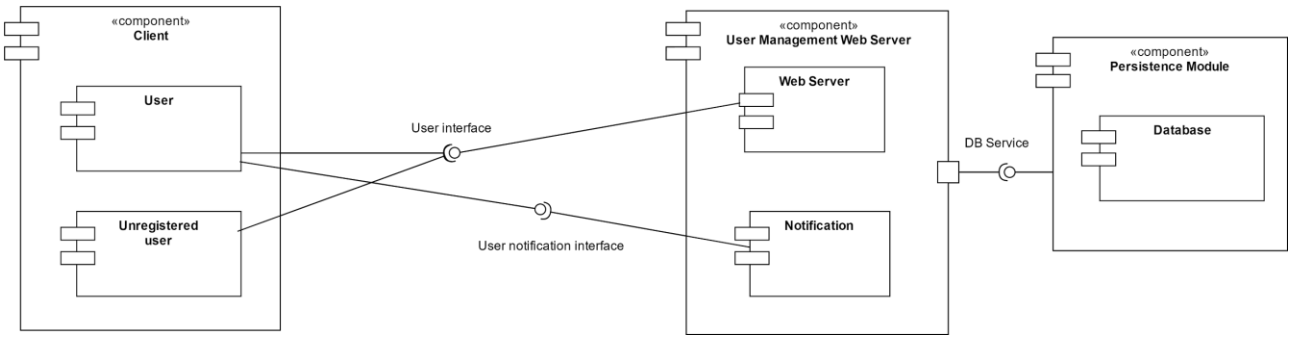
- Rental Web Server: it is the component that administer the entire process of the rental: from the ride reservation to the end of the ride. It is composed of two sub components:
- Reservation Web Server: It offers its interfaces to clients that want to check the available cars and book a reservation.
- Ride Web Server: It is used by client in order to begin a ride or terminate it and it ask to the persistent module to update the status of the rides.
- Maps Web Server: It manages the maps of the safe and special safe areas and can check if a car is inside or not in one of it. Moreover, cars daily update their maps so that user can enjoy the better experience and can correctly spot safe and special safe areas.
- Car Management Web Server: It is the component dedicated to the communication with cars and the management of its information. It is composed of two sub components that lead these type of communications:
- Notification: it is the component that push notification to the car in order to send requests when necessary. It is designed as a separate component in order to avoid that the user application could directly ask to it to make request to the

car. As a result, all the communication is filtered by the rental component that manipulate the requests and select the right car to forward the message.

- Status Management: It provides the interface for other components that want to update the status of a car. The employees can directly change the status in case of problem, while users that want to terminate the rental can ask, through the rental component, to end the ride.

PERSISTENCE MODULE: it offers an interface in order to communicate to the database safely. It is used by most of the components of the web service that, in this way, do not rely on a specific DBMS.

## 2.5. Component Diagram: User Management




In order to clarify the management of the users the user management web server, that is a subcomponent of the web server, is proposed here.

USER MANAGEMENT WEB SERVER: it is the controller of the personal information about the users, and it provides the function to update them:

- Web Server: It is the component that generates the web pages for the clients; It allows the login, and the update of the personal information.
- Notification: It contains the logic for pushing notification to the client. In some cases, such as the sending of the monthly invoice, this component is asked to recap via E-Mail all the costs charged to a specific user. In addition, can notify also in case of penalty due specified road traffic offences or problems with the payment method.

## 2.6. Component Interfaces

In a system each component interacts with other elements inside the system. The communication between components is managed using interfaces. Every interface has a set of methods, which could be called by another component in order to perform specific actions. The components can avoid to know how the operations are implemented in each element.

Here is presented a description of each interface presented in the component view, in order to explain the interactions between components.

- ***DB Service***

    - +insertUser(User usr : void)
    - +insertReservation(Reservation reserve : void)
    - +insertEmployee(Employee empl : void)
    - +insertRide(Ride rid:void)
    - +insertCar(Car car:void)
    - +modifyEmployee(Employee empl : void)
    - +modifyUser(User usr : void)
    - +updateStatus(Status stat, Car car:void)

The set of methods defined above are used so as to insert values and query inside the DBMS. With these methods, it is possible to add or modify query inside databases. They take the entities classes, with every detail related, that will be decomposed in basic type pieces of information and inserted in the database.

- ***Map Management***

    - +calculateRoute(String departure, String destination)
    - +isInsideSafeArea(Car car, GPSData local_car)

This set of methods concerning all the things which the service needs concerning the problems related to the map.

- ***Reservation***

    - +reserveCar(String username, Car car, Date reservation_date, Time reservation_time : Bool confirmation)
    - +searchAvailableCar(GPSData local)
    - +searchAvailableCar(String departure_address)
    - +unlockCar(Car car, String username, GPSData local_user)

This interface allows users to rent a car from his web-application. The interface for each method need some information passed as parameters such as, the client username, the car selected or GPSData. The "unlockCar" method is different from the method inside the car management interface, that because first of all the user ask to the "Reservation" component to unlock the car, which will analyse the position between the user and the car. After that the "Reservation" component will ask to the "Car management" to unlock the specific car.

- **Ride**
    - +beginRide(Car car, Reservation reserv)
    - +endRental(Car car)

The ride interface provides the methods to manage the ride of a reservation. The system with these methods has all the information in order to track the car and define the final invoice of rental.

- **Status**

    - +lockCar(Car car)
    - +unlockCar(Car car)

- +getCarPosition(Car car)
- +updateStatus(Car car, Status stat)

This set of methods is the interface of status component. It provides all the methods in order to change and set the car status.

- **User Manager notification**

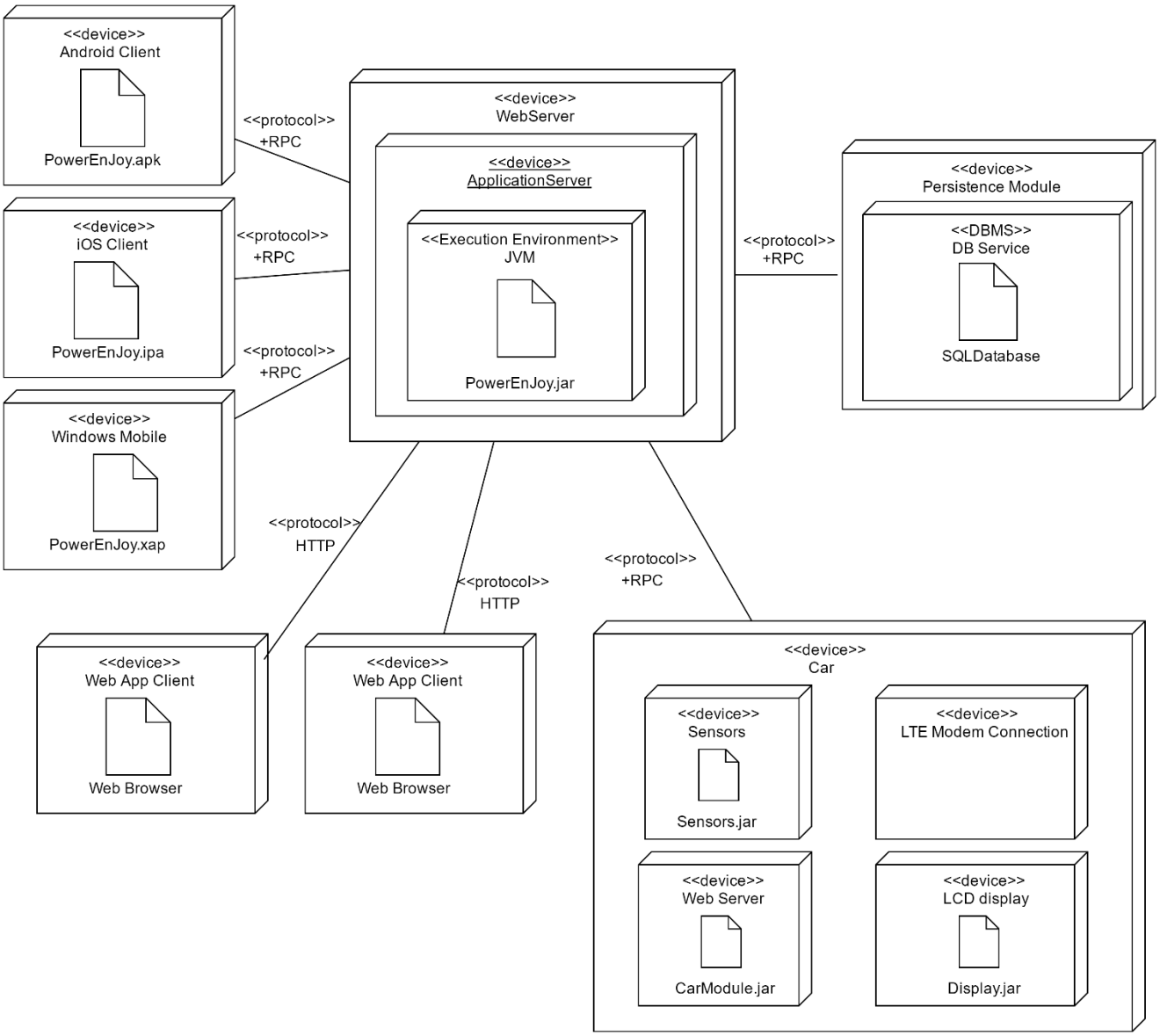
  - +sendEmail(String email, String event_type, String user_destination, String password)
  - +sendNotification(String username, String event_type, String event_id)

This set of methods is created in order to send notifications and emails from the server to the users. The first method is used also as requested inside requirements in which it is asked to send an email with his password as soon as the user fulfilled the registration forms.

- **Car**
  - +lock()
  - +unlock()
  - +askPosition()
  - +rideInfo()

The car provides an interface callable by the notification component of the car web server. This interface allows the basic behaving of a car such as the lock and the unlock. Moreover through the interface it is possible to retrieve information about the car position and about the ride (e.g. if a discount is abblicable)

## 2.7. Deployment View




The diagram illustrates the deployment of the physical nodes of the application and the different technologies used to communicate between different nodes. Thanks to the three-tier architecture the system can be easily scalable and the web server can be replicated in case of a significant amount of requests without losing coherence. The diagram is composed as:

Mobile devices: All the mobile devices can communicate to the central web server through a and each one has a specific application designed for the different operating systems existing on the market.

Web App Client: Clients can perform reservation and update their personal information also through the browser. In this case an HTTP protocol is used in order to send GET and POST request for retrieving information or sending the chosen car to the central server. The presentation layer is implemented using HTML5 for a more entertaining using experience and JavaScript with AJAX for the asynchronous calls.

Car: Standard electric cars are equipped with a set of devices in order to make them suitable for the scope of PowerEnJoy. These devices are mainly:

Sensors: that consist of an application that manage the sensors already present on an electric car.

Web Service: that manage the request of the server, such as the information about the ride or the current position.

LTE Modem Connection: that is installed on each car in order to guarantee a stable internet connection and make the car reachable by the server.

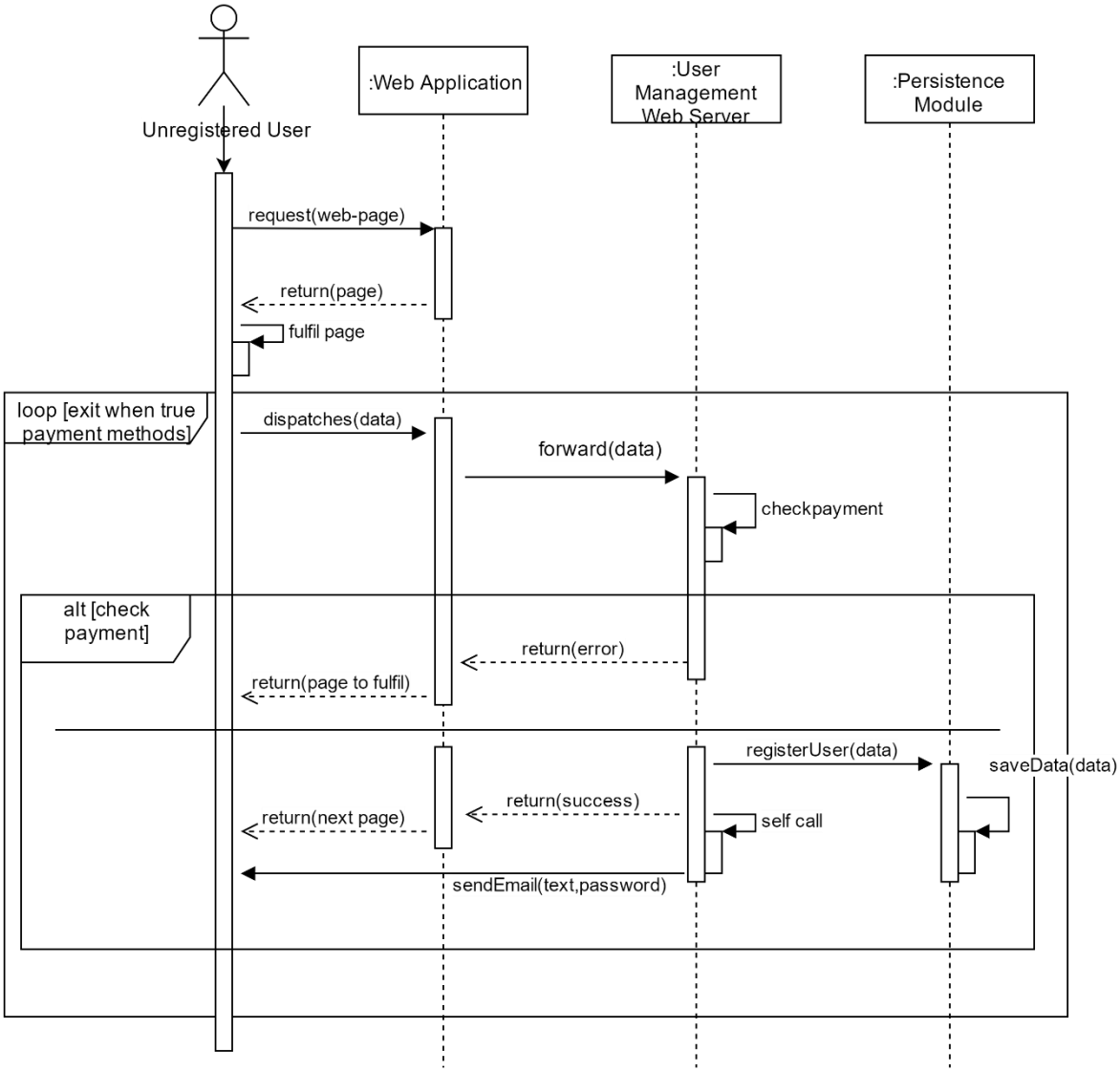
WebServer: It is the main node of the application and works with a Java Virtual Machine installed on it. In fact, the logic of the application is written in Java and this server manages the requests from the clients through different servlets.

Persistence Module: It is the server where the database is stored and it is provided with an application that offers its interface to the Web Server in order to make the retrieval of data independent to the DBMS used. However, in this case a SQL database is used because of its reliability. This module implements a Facade pattern to make the access to the database unique and a repository pattern to uncouple the object-oriented logic used by the business logic from the tuples of the database.

## 2.8. Runtime View

The runtime view will be represented in form of sequence diagrams that illustrates how the components of the application interact one wo each other. There will be proposed the main five case of the application: Sign Up, Make a reservation, Begin the ride, Complete the ride, Employee car management.

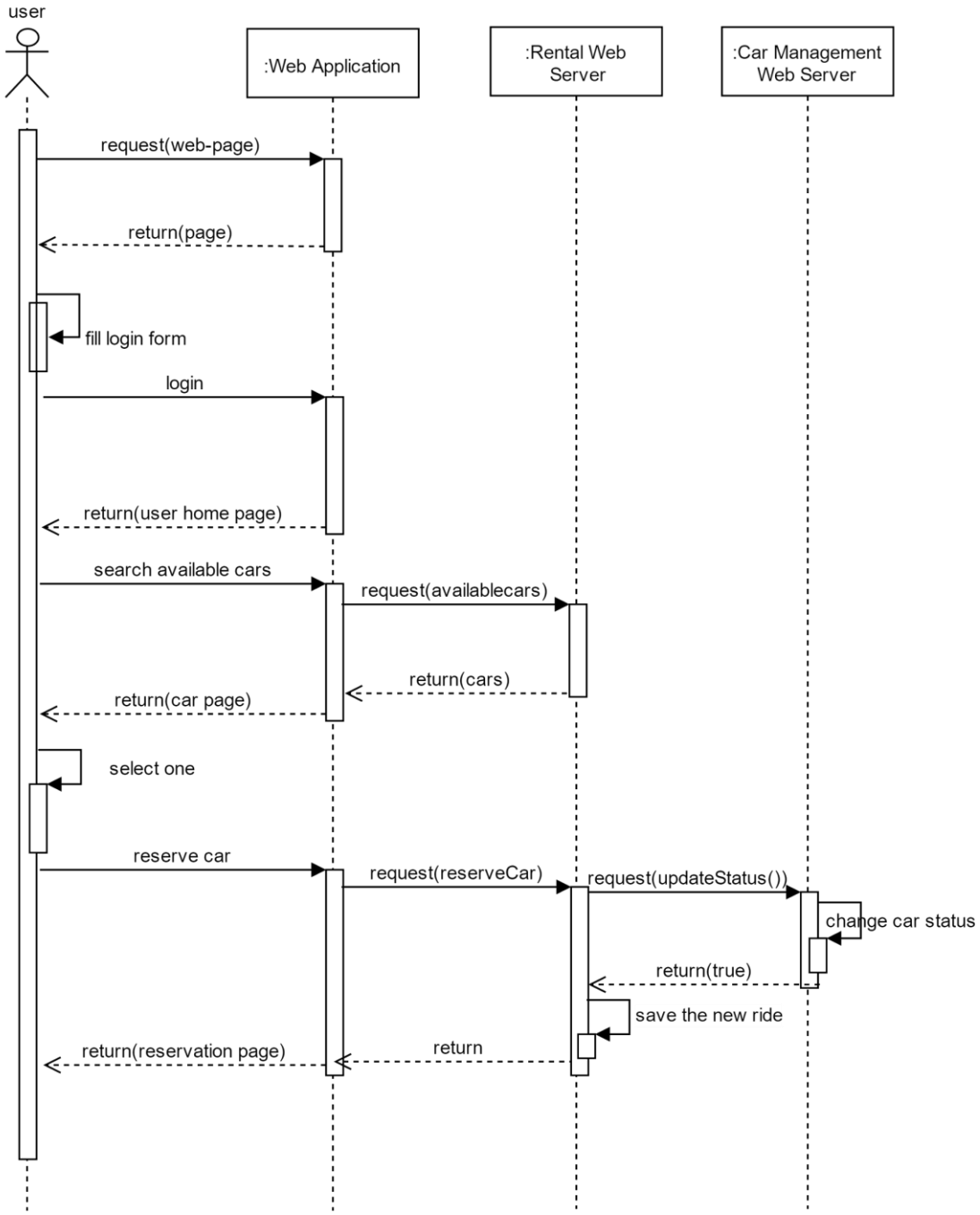
*Sign Up of a new User*




The following Sequence Diagram represents the Sign Up functionality. First of all, an Unregistered user reaches the page, where a web application will provide the data and pages to the user in order to register.

When the forms concerning the register process, are fulfilled the application server turn in this information to the User Management Web Server. If the payment method is true, it will provide the next page and It will send the password with the mail. In order to check the validity of payment methods the system use some external APIs not provided by PowerEnJoy.

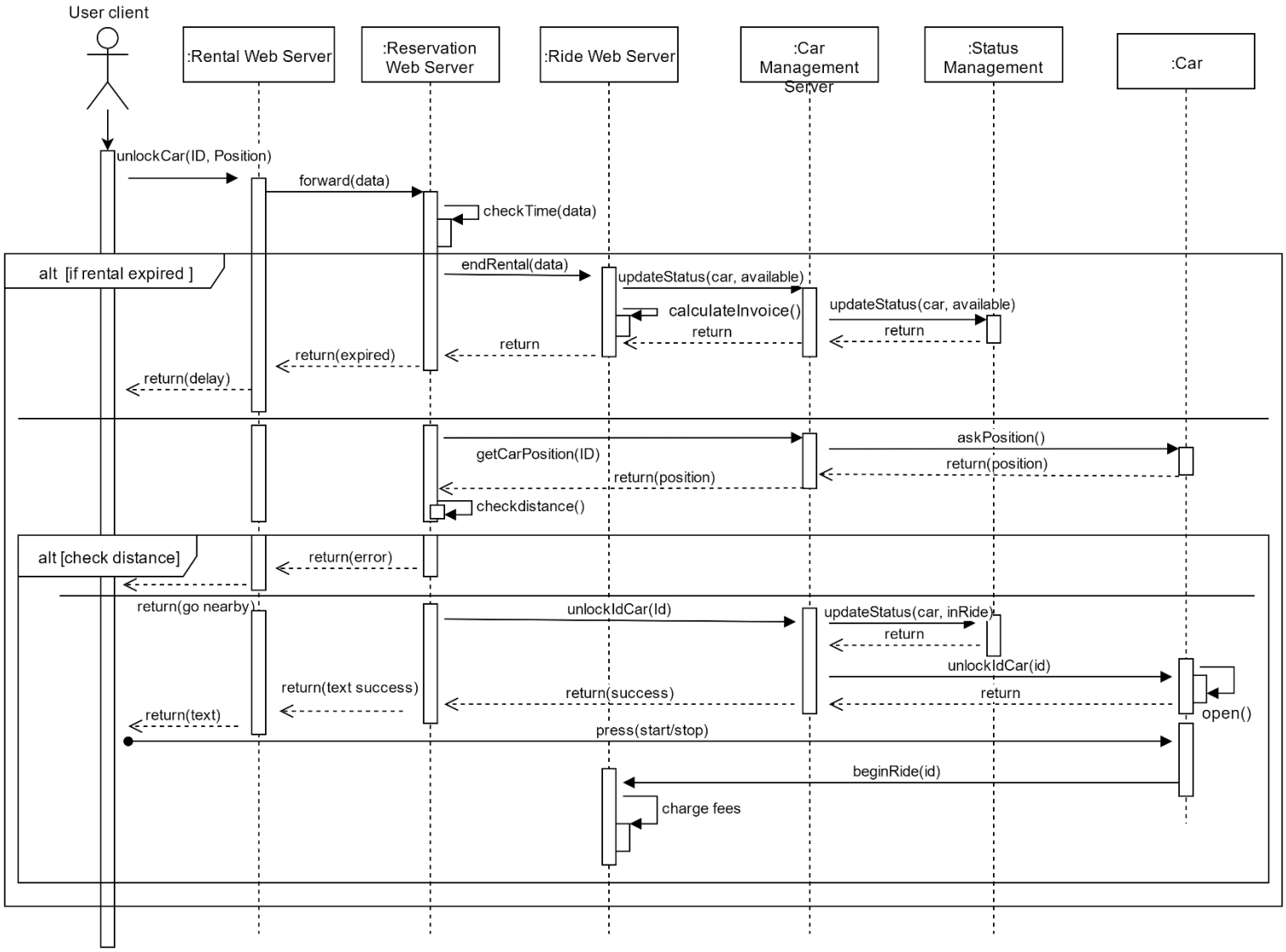
*Make a reservation*



The sequence diagram explains how a user searches a car and reserves it.

First of all the user inserts its credential and logs into the web site, with login() so to be redirected to the homepage, after that, he inserts the address and the radius for the research, with searchAvailableCars(), then the page with available cars is loaded, and after he can select one of them and reserve it, with reserveCar(), and a new reservation will start with the user and the selected car.

*Begin the ride*



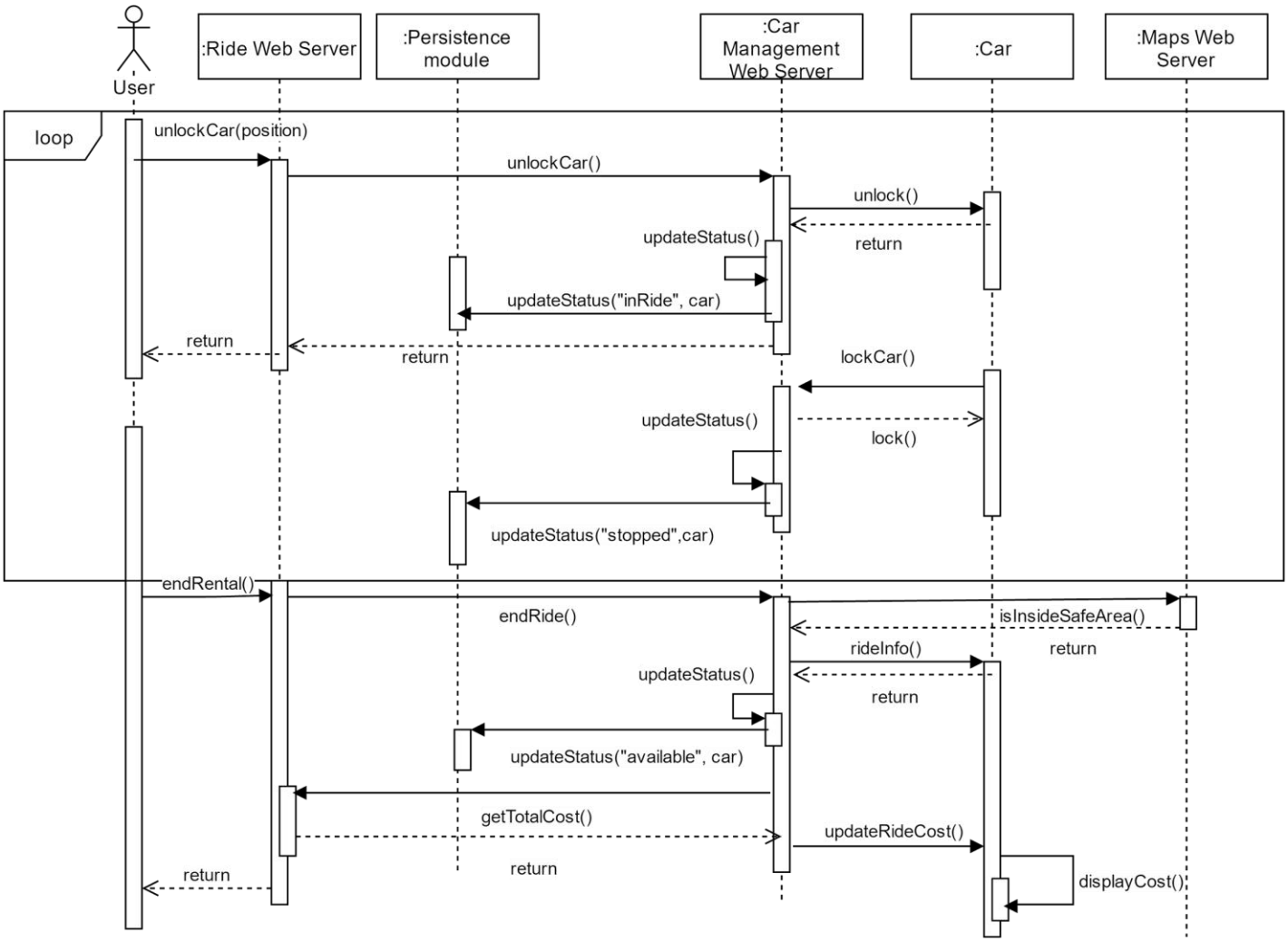

The sequence diagram can be divided in 3 different situations. It is like a sequence of if-else clause, in which only the fulfilment of all the conditions can unlock the car for the user.

In the first if-clause the system controls if the user has reached the car in time (within one hour). If he fulfilled the condition, the system checks the distance between the user and car positions.

If the user is nearby the car, the system will unlock the car for the user. As we can see, all the interactions with car go through the “car management server”.

Finally, when the user presses the “start/stop” button, the car warns the “Ride Web Server” in order to start the count of the finally invoice for this reservation.

*Complete the ride*



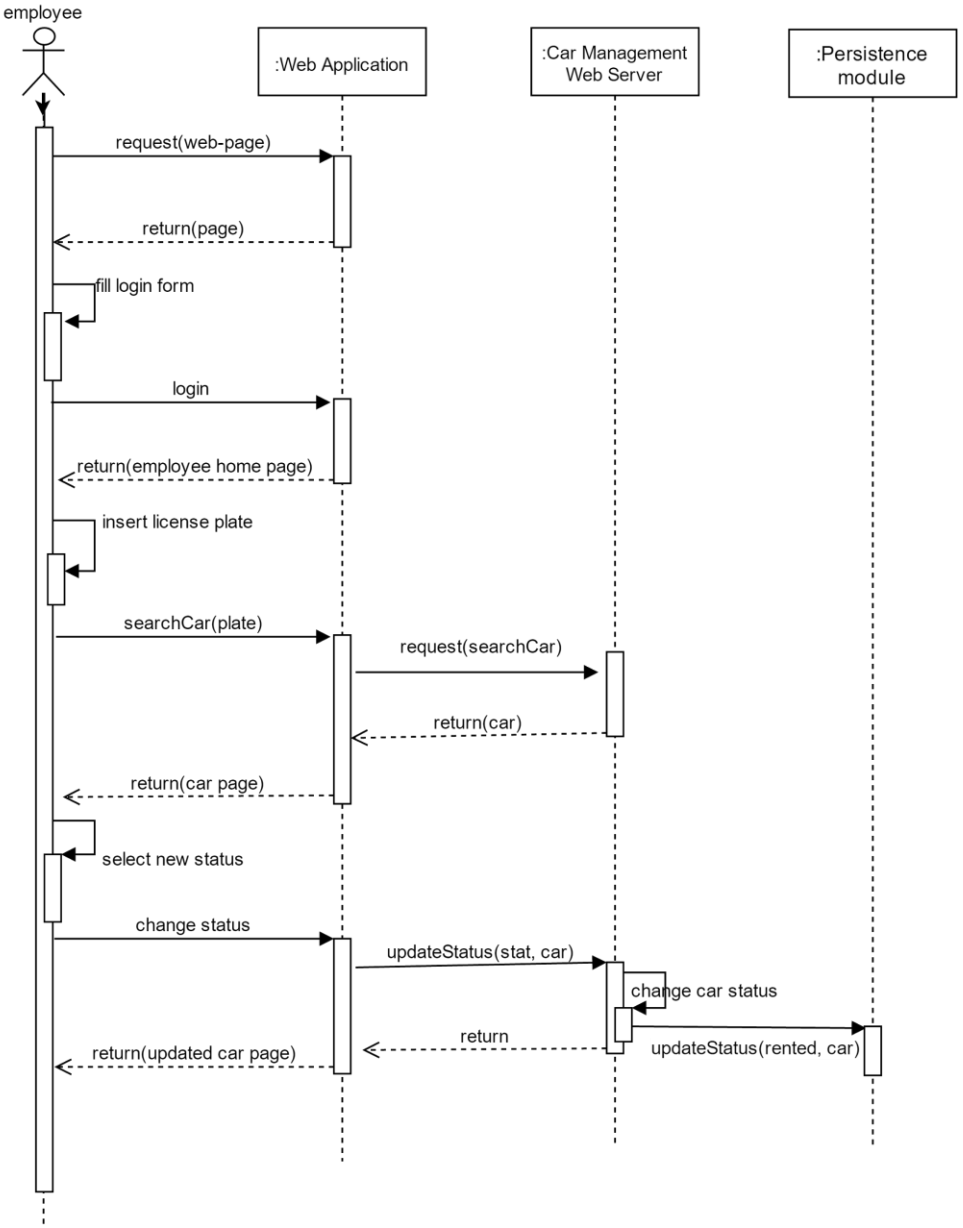

The sequence diagram reveals the structure of the communication between the main objects of our application during a ride. Firstly, a user, through the mobile app asks to the system to begin or resume a ride calling the beginRide() method. If the information provided are correct the car is unlocked and the status of the car is updated to “inRide”.

Now, the user can freely use the car and exploit the service offered by PowerEnJoy until he decides to have a break. As soon as all the passengers has left the car, this information is notified to the system that lock the car and update its status in “Stopped”.

This process goes over until the user asks to end the rental; at this point, the system, after having checked if the position is inside a safe area, asks to the car the information about the ride such as the duration and the possible discounts.

The status is updated again in order to set the car available for other users and the final cost of the ride is calculated. Finally, the calculated cost is sent to the car that shows the amount over its display.

*Employee car management*



The sequence diagram explains how an employee searches a car and changes its status.

First of all, the employee inserts its credential and logs into the web site, with login(), and he is redirected in his homepage, after that, he inserts the license plate of the car he wants to search, with searchCar(), then the specific car page is loaded, and after he can select a new status and change it, with updateStatus(). The updateStatus method will both change the car status inside the "Status Management" and the DBMS.

## 2.9. Selected architectural styles and patterns

All the application architecture is built on a client-server structure, and more specifically this is the three-tier architecture (presentation layer, application layer and database layer). In this multi layered architecture there is a thin client, which only displays information related the service (the group of pages of the website, no operations are executed on the client side) and a fat server which controls the application's functionality by performing detailed processing. The data tier includes the data persistence mechanisms and the data access layer that encapsulates the persistence mechanisms and exposes the data.

Each macro component is divided in sub-components so as to create a set of different atomic modules easy to use, but also they manage to reduce the system complexity.

The components exchange information with others using the RPC (Remote Procedura Call).

During the defining of application design, it was decided to introduce some design patterns very useful, because they represent solutions to common problem.

***Thread Pool***

Due to the relevant number of requests from different users at the same time a thread pool pattern is necessary in the management of these requests to the database. In this way, different requests are managed with the same connection in order to improve performances.

In this case, both the users and the employees will perform calls to the database so a thread pool makes the system scalable in an easier way.

### *Read-Write Lock*

This is a pattern to manage concurrency: it is a synchronization primitive that solves one of the read-write problems in order to avoid data corruption or simultaneous operations.

### *Repository*

In this pattern the sub-system components, which the system is decomposed, access and modify the same data structure, called Repository. The single components are independent among them, and they interact only with the repository. With this pattern the managing of data is though with an object oriented view, but on the other side of repository interface there is a query logic. The pattern is used in the Persistence Module to separate the business logic from the data access layer.

### *Façade*

The Façade pattern is used to hide the details of complex blocks of components with different interfaces in order to simplify the communication between these blocks and the rest of the architecture. In the project this pattern is used to allow the communication between the web server and the database.

# 3. Algorithm Design

## *Money Saving Option*

The money saving option can be activated by a user before starting the ride. If the registered user enables “Money Saving Option”, the user can input a final destination and the system provides information about the Special Safe Area where to leave the car to get a discount. This station is determined to ensure a uniform distribution of cars in the city and depends both on the destination of the user and on the availability of power plugs at the selected stations.

*Pseudo-Code:*

```
*user activates Money Saving Option (MSO) and inserts a destination*

nearbySpecialSafeArea = [ ]
destination = requestMSO.getDestination();
nearbySpecialSafeArea = system.getListNearbySpecial(String destination);
minNumCars = nearbySpecialSafeArea[0].getNumberOfCars()
specialSafeAreaSelected = null
for specialSafeArea in nearbySpecialSafeArea:
if( specialSafeArea.getNumberFreePlug() > 0)
        if(specialSafeArea.getNumberOfCars() <= minNumCars)
            minNumCars = specialSafeArea.getNumberOfCars()
            specialSafeAreaSelected = specialSafeArea

requestMSO.setParkArea(SafeArea specialSafeAreaSelected)
return 0
```

## *Cars Density*

One of the main challenge that the application has to deal with is the distribution of cars inside safe areas. Indeed, a concentration of too many vehicles in the same area may lead to a bad user experience. The solution is to provide to employees a tool that evaluate the density of cars inside a safe area and spot dense zone. The employee chose the safe area and the radius where to perform the analysis. All the parked cars of the safe area are collected and for each one the position is retrieved.

Through the interface of the CarManagement all the available cars placed nearby (distance below radius) are collected; as available cars are intended all cars that are not under maintenance or plugged. In fact, plugged cars are not displayed to users in order to allow them to recharge completely without damaging the expensive battery.

Then, the system notifies to the employee the density of the area around each car so that the operator can discuss about moving some cars in a better position to improve the user experience.

*Pseudo-Code:*

```
AnalyseCarsPosition(SafeArea safeArea, int radius){

for car in safeArea.getParkedCars():

 carPosition = CarManagement.getCarPosition(car)

 availableCars = rentalManagement.searchAvailableCar(carPosition, radius)

density = availableCars.length()/( PI * radius ^2 )

print(“The area around car: ”+car.getLicencePlate()+”placed in+“+carPosition+”  has a density of “+ density)

}
```

### *Final cost of a ride*

The last thing to calculate on a ride is the final cost. If the user does not unlock the car in one hour from the reservation, expired is set "True", he will be charged a fee of 1 EUR. When the user presses the start/stop button, the engine ignites and the system starts charging user for a fee of 0,25 EUR/min.

Then there are some discounts or extra-payments:

- -10% if the car says that the discountForGroup is applicable (the number of passengers must be more than one for at least the 50% of the ride)
- -20% if the car is left with more than 50% of battery
- -30% if the car is parked in a special safe area and plugged in the power grid
- +30% if the car is parked at more than 3000 meters away from the nearest grid station and with less than 20% of battery
- -40% (cannot be combined with other discount) if the user uses the money saving option

*Pseudo-Code:*

```
int totalCost(MSO requestMSO, Boolean expired, Car car)
cost=0
nearbySafeArea = system.getListNearbySafe(GPSData car.getPosition() )
nearbySpecialSafeArea = system.getListNearbySpecial(GPSData car.getPosition() )
discountForGroup=false
if (expired)
        cost=1
        return cost
cost=duration*0,25

if ( car.getDistance(requestMSO.getParkArea())<40 AND requestMSO.active()
//if the distance between a car and a special safe area is below 40m the car is considered
//inside the SPA
        AND  car.isPlugged() )
                cost=cost*0,60
```

```
            return cost


for specialSafeArea in nearbySpecialSafeArea:
        if (car.isPlugged() AND car.getDistance(specialSafeArea)<40)
            cost=cost*0.7
discountForGroup=car.askIfApplicable()      //ask if the number of passengers is above 1
if (discountForGroup)
        cost=cost*0,9
if (car.levelBattery() > 50)
        cost=cost*0,8
if (car.levelBattery() < 20 && car.distance(nearbySpecialSafeArea)>3000)
        cost=cost*1,3
return cost
```

# 4. User Interface

## 4.1. Design Overview

The main idea during the developing of user's interface is to create an application (web and mobile) which is easy and immediate for the user.

As we can see, the user interface is designed to meet these requirements. From the mock-ups presented in RASD, here it is presented the links between the single interfaces, in order to make the procedures intuitive and as easiest as possible.

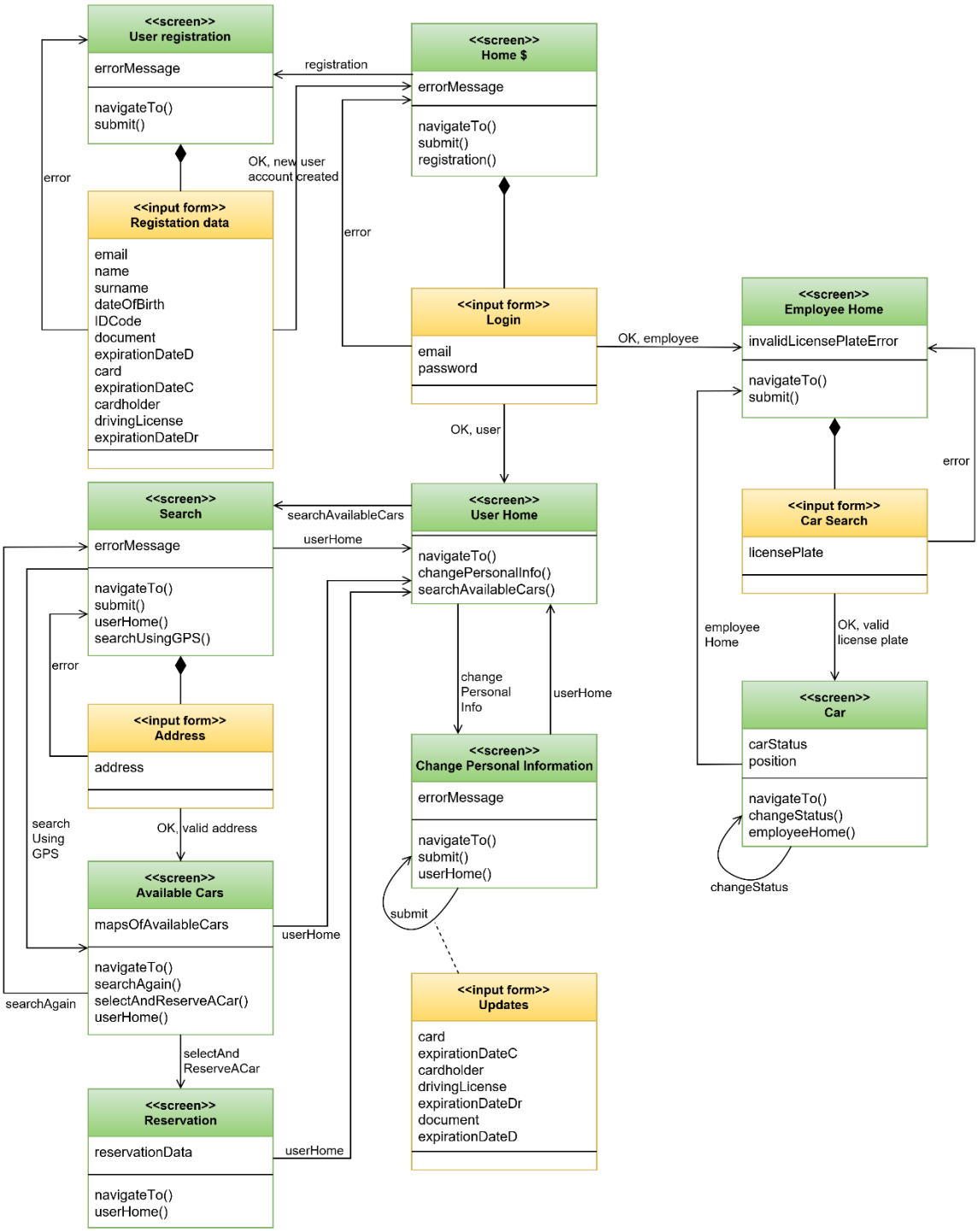
## 4.2. User interface and navigation flow

Here it is presented a diagram which depicts how pages are linked among them. In this diagram it is also described: the input forms, different pages and how the navigation through the website is structured.

The diagram is presented as a class diagram, and the symbols have the same meaning as if they were used in that kind of diagram. The tag <<page>> means that the class represent a web page, and <<form>> identifies an input form contained in a specific page.

The home page structure is the same for employee and users. The unregistered users can register themselves through a specific page, reachable from the home page. The registered user can access to the rental service, so every register user is able to reach pages for which he has permissions.

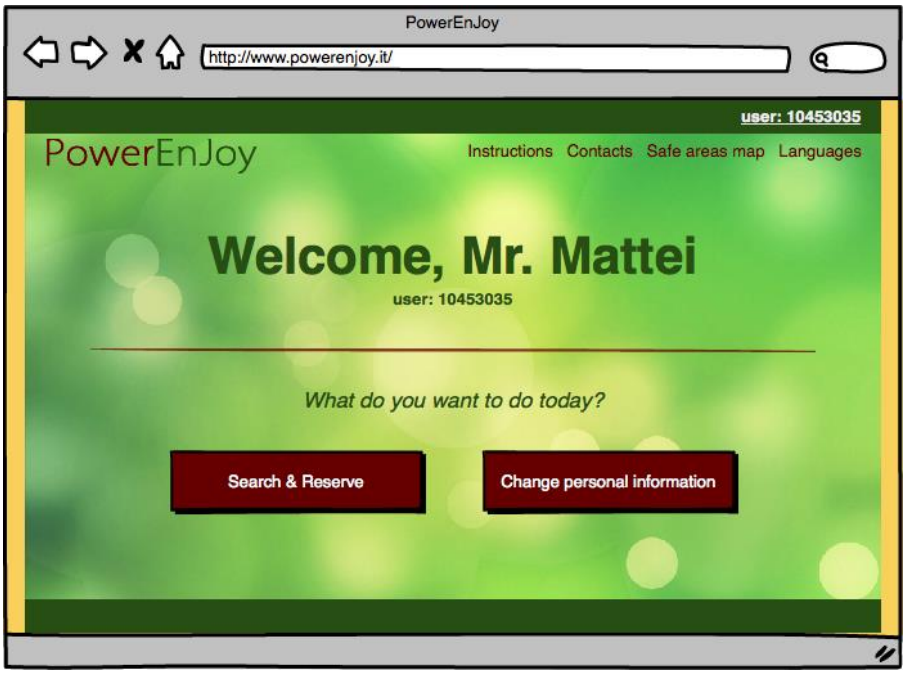
Here are presented only direct flows, associations representing cancellations or links to previous or home pages are omitted in order to simplify the reading of the diagram.




## 4.3. Further preview of the UI

Here we present other mock-ups, which are not putted inside RASD. These mock-ups are presented with a brief description. They have been created in order to clarify some detailed aspects.

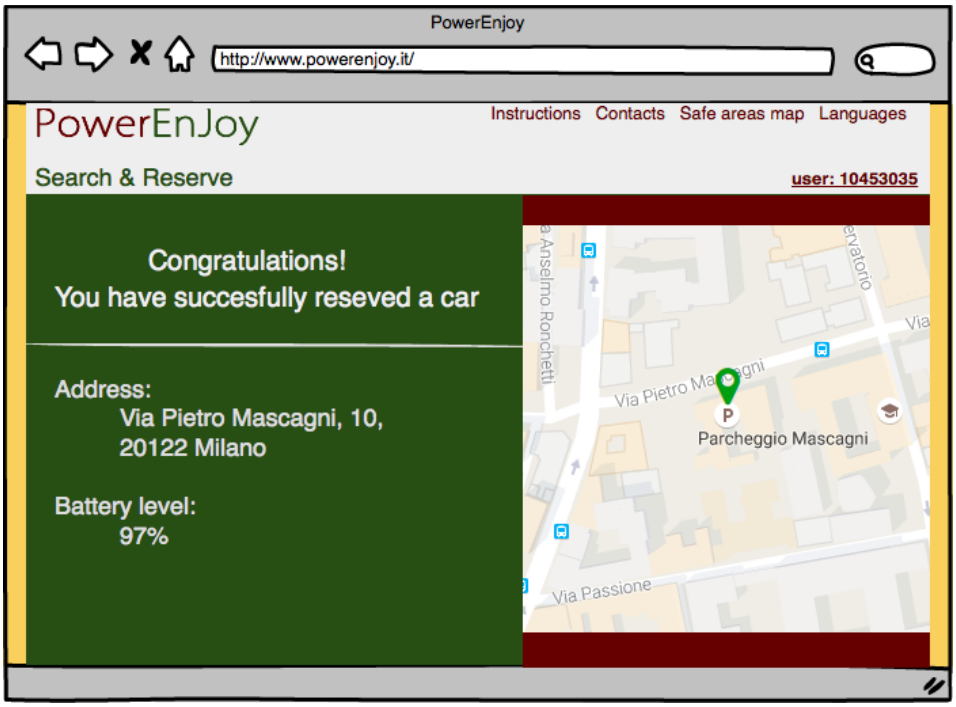
***Welcome page registered users;*** the register user will see this page after the login.




***Employee panel;*** the employee with this panel is able to change the status of a car in case of issues

***Welcome page employees,*** employees will see this page after login


***After Reservation page,*** the user after the reservation will see a brief summary



## 4.4. User Experience

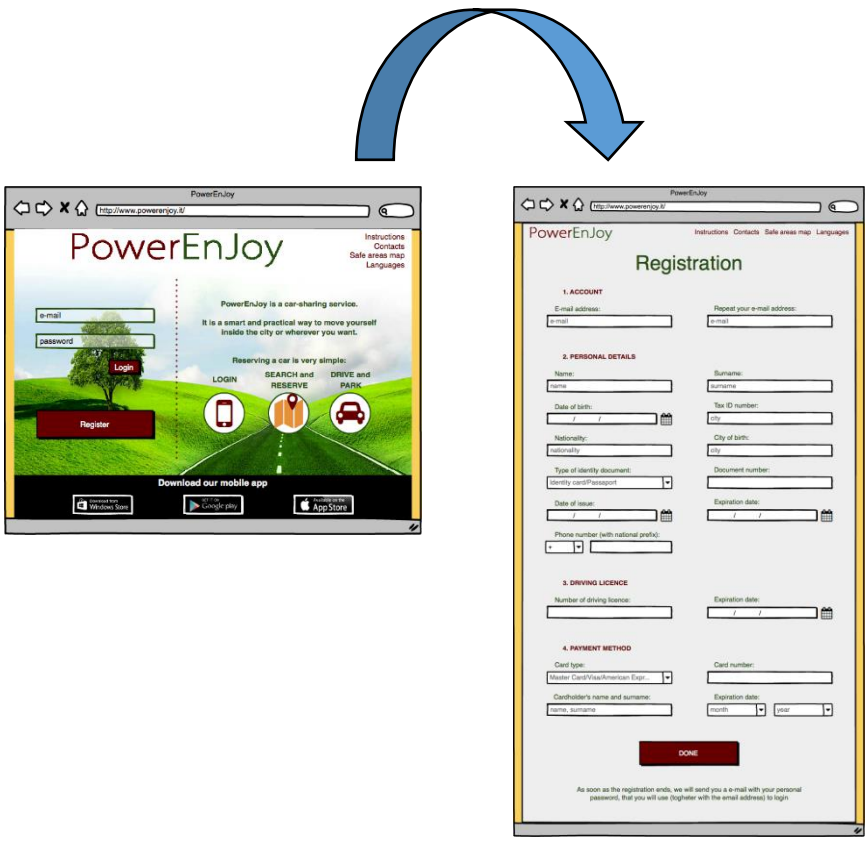
In order to make the service as immediate as possible the sequence of actions which need to be performed so as to reserve a car are few and very easy. Inside the home page the Unregister user is able to create a profile in PowerEnJoy.

Every option is fully described in the RASD section related to Use Cases, and the navigation uses only buttons to navigate through the pages, and forms to insert data.

There are shown some page flows based on the mock-ups already presented in RASD, in order to fully describe how pages are linked, and in which way the navigation is structured.

### 4.4.1. Sign Up process

When a new User reaches the page, he can Sign up his profile inside PowerEnJoy. Inside the home page there is a signup button: in the following page he will find the registration form. The new user needs to register himself in the website in order to use the website features such as make a reservation.



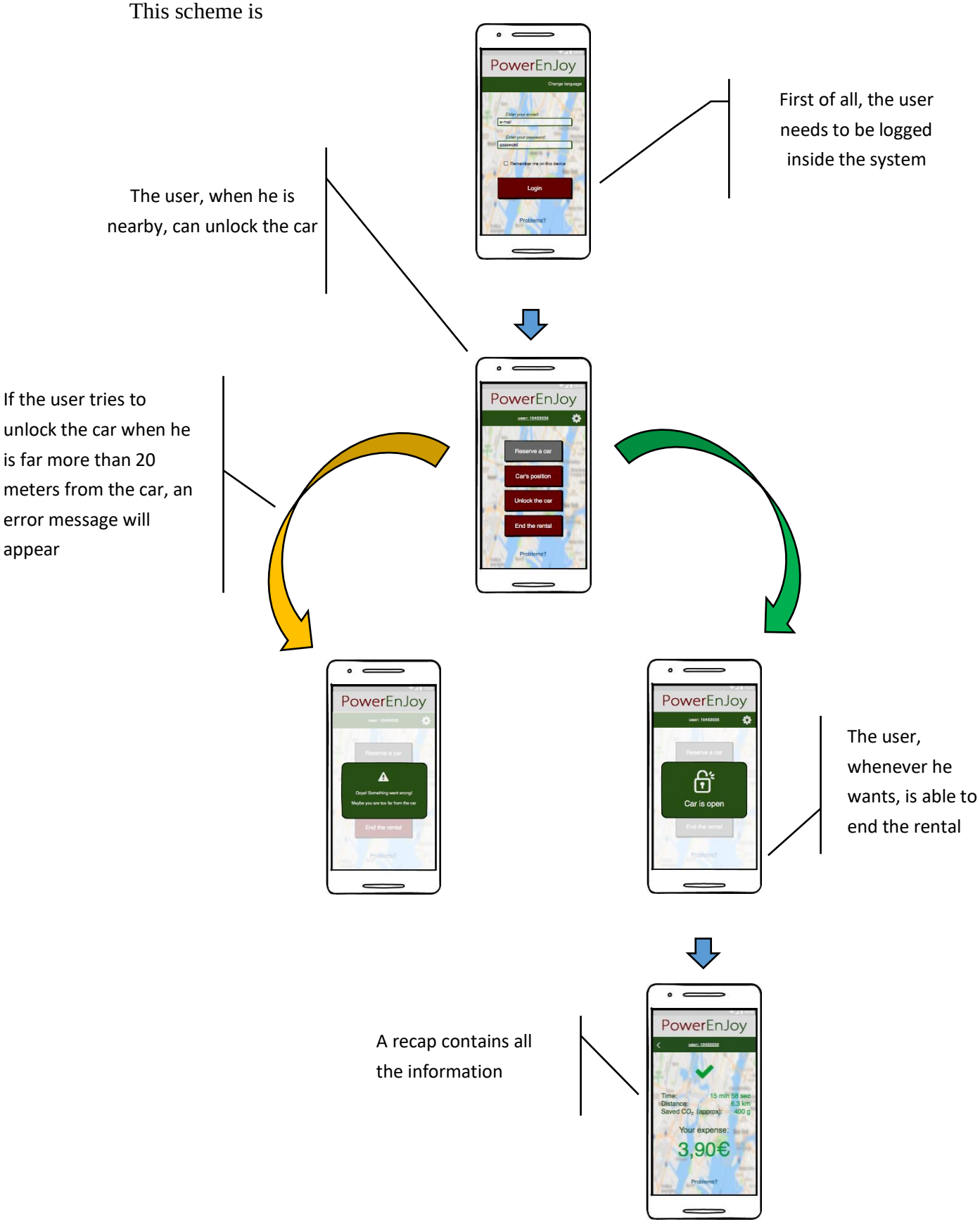
### 4.4.2. Reserve a Car

A registered user can reserve a car from the website. First of all, he needs to be registered so as to access to the website with his profile. After that the user access to "Search & Reserve" page, in which the user can rent a car selecting the radius and where he needs the car. The system will show a page referring to user's preference defined. In this page the user can select the car, which better fits his needs. Finally, a page of confirmation page will be shown, where the user can read a resume, or go back to the home page.

## 4.4.3. Unlock a car and end rental

Here It is described the sequence of actions concerning the phase of unlocking the car and end the rental with the application.

This scheme is



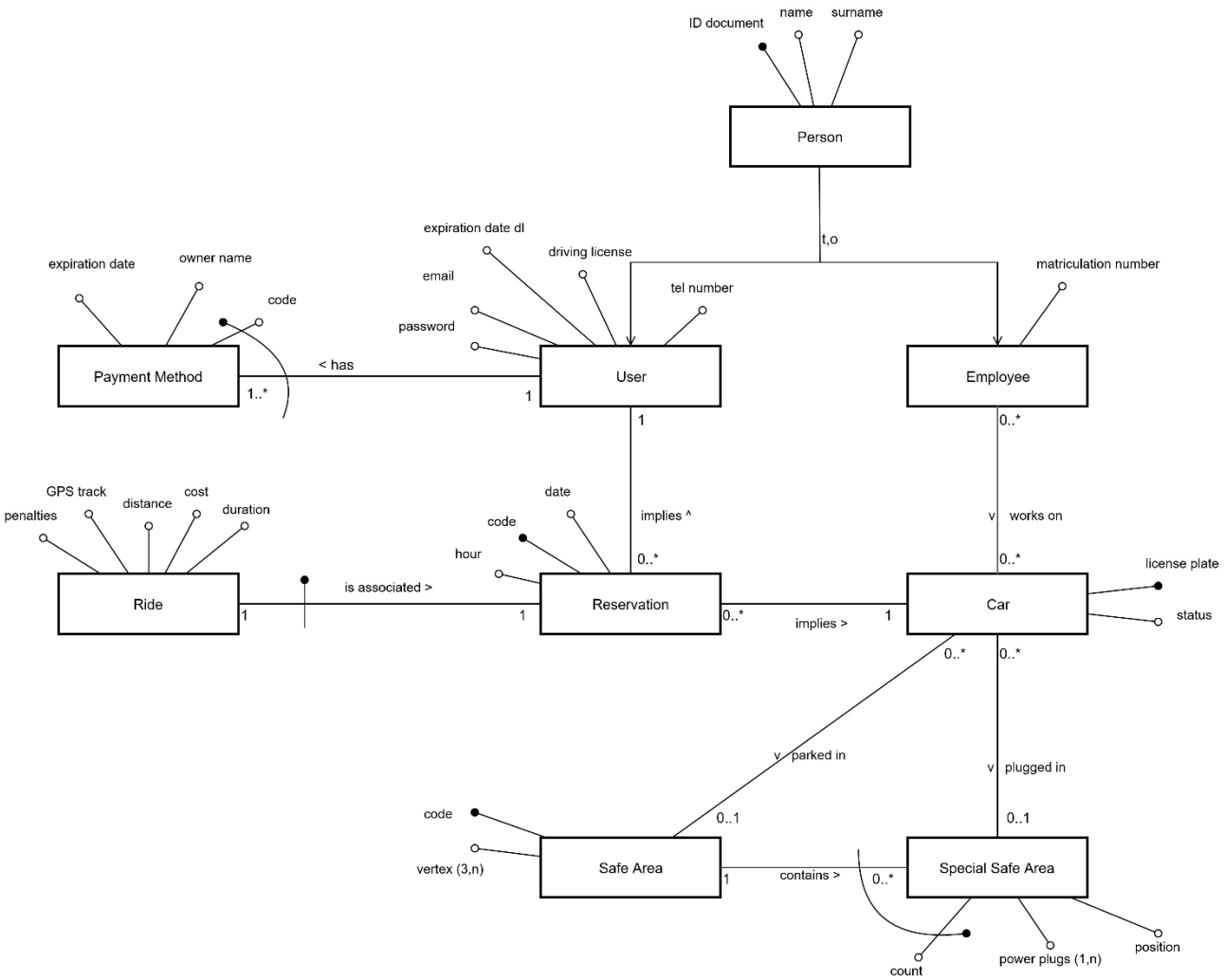
# 5. Entities Architecture

## 5.1. E-R Diagram

In order to clarify the data structures used in the application, here is reported the ER (Entity-Relationship) diagram for the application database.

Every entity has a primary key, a field already suitable or inserted that is fundamental to identify every record of the database; in this way there are no weak entities.

Every relation between entities is represented with its cardinality, and the diagram uses the UML standard for entities and attributes, becoming clear to read and self-explaining.



## 5.2. Relational Model

From the ER diagram we can easily find out the Relational model:

- User(IDDocument, Name, Surname, Email, Password, DrivingLicense, ExpirationDate, TelNumber)
- Employee(IDDocument, Name, Surname, MatrNumber)
- PaymentMethod(CardCode, *UserID*, OwnerName, ExpirationDate)
- Reservation(ResCode, *UserID*, *Car*, Date, Hour)
- Ride(*ResCode*, Duration, Cost, Distance, GPSTrack, Penalties)
- Car(LicensePlate, Status)
- SafeArea(Code, Vertex)
- SpecialSafeArea(*SafeArea*, Count, PowerPlugs, Position)
- Work(*Employee*, *Car*)
- Parking(*Car*, *SafeArea*)
- Plugging(*SafeArea*, *Count*, *Car*)

Foreign keys are all the attributes in *italic*, and primary keys are the underlined.

Every entity has a primary key (that can be a couple too) that identifies uniquely every other attribute, and every relation is identified by the foreign keys belonging to the entities involved. The 1to1 and the 1toN relation has been absorbed by other entities. We wanted to keep both Ride and Reservation, for a more ordered model.

# 6. Requirements Traceability

***[G1] Users must be able to register to the system by providing their credentials and payment information***

[R1] [R2]: The integrity of the Database allows to guarantee the uniqueness of usernames.
[R3] [R4] [R8]: Specific pages of the web application and the mobile apps permit to register and access to the platform
[R5] [R6]: The system is provided with a notification centre that can communicate to users in case of necessity and manage all the e-mail involved in the communications
[R7]: The system exploit the services offered by the different credit cards producers in order to verify the validity of a credit card.

***[G2] Registered users must be able to find the locations of available cars within a certain distance from their current location or from a specified address.***

[R1] [R3]: The Car Management component can ask to car their position; in addition, each car can retrieve its position and send it as an answer using the LTE module present on the car.
[R2]: Cars are provided with sensors managed by the main component management of the car that can answer to requests by the Car Management of the system.
[R4]: a page of the web (Available Cars) application allows user to search for cars; the request is managed by the Reservation Management of the system.
[R5]: The Database contains all the information in regard to safe areas that is managed by the Maps Management.

***[G3] Users must be able to reserve cars***

[R1]: a specific page (Reservation) grant the possibility to select one car among the ones available.
[R2]: The Database contains all the information about past and actual rentals that are accessible through the interface offered by the Persistent Module

[R3] [R4]: The Reservation Management verify that users cannot rent another car while performing a rental. Moreover, only available cars can be rented.
[R5]: The Reservation Management check that the user that is completing a reservation has already paid the last monthly invoice.

***[G4] A user must be able to enter in the car reserved by him when nearby***

[R1]: The car module present on each car provide an interface that allows the Car Management to lock and unlock a car.
[R2]: The user sends his position together with the request to unlock the car and the system is able to retrieve the position of each car. At this point the Ride Management is able to calculate the distance between the two devices and verify if it is below 20m
[R3]: The LTE modem connection present on each car allows the connection to the different web servers. The mobile application can access to the system thanks its interfaces and perform different requests.
[R4]: Users take advantage of the GPS module present on their device in order to provide a correct position.

***[G5] The user is able to know the current charge of the ride***

[R1]: each car has an interactive display that is managed by the Car module present on each car.
[R2]: The Car Module can ask through the interface offered by the Car Management to begin the ride.
[R3]: The Car Module is able to calculate dynamically the amount of the fee

***[G6] The system is able to calculate the final cost of the rental***

[R1] [R2]: each car is provided with sensor that are used to retrieve information about the ride that are then sent to the Car Management.
[R3]: The Database contains the information about the fact that a car is currently plugged or not; in addition, it stores the special safe area where is plugged.

***[G7] The system is able to know when a ride ends***

[R1]: weight sensors placed under the seats reveals the number of passengers present on the car. If the number is 0, the Car Module sends this information to the Car Management.
[R2]: Map Management can check if a position is inside a safe area.
[R3] [R4]: the car Module can communicate with the system in order to confirm lock of it.

***[G8] The system is able to manage the cars***

[R1]: The Car Management Application (employee app) can take advantage of the interface offered by the Car Management in order to update the status of a car.
[R2]: the mobile application contains the number to call in order to contact the telephone exchange of the system in case of problem.
[R3]: cars and the system can communicate over the Internet through the provided interfaces.
[R4]: Car Management Application can check the position of the cars and notify employee if a car needs to be moved.

***[G9] The user is able to update his profile and payment methods***

[R1]: a specific page (Change Personal Information) allows users to update their personal information on the Database.
[R2]: The system can use services offered by the different credit cards to check the validity of a credit card

# 7. Hours of work

*Frigerio Lorenzo:*

- 20/11/16: 4h
- 21/11/16: 2h
- 22/11/16: 3h
- 27/11/16: 3h
- 29/11/16: 3h
- 4/12/16: 4h
- 5/12/16: 2h
- 6/12/16: 3h
- 9/12/16: 3h
- 10/12/16: 2h
- 11/12/16: 3h

*Perfetto Martina:*

- 19/11/16: 2h
- 20/11/16: 4h
- 21/11/16: 2h
- 26/11/16:2h
- 27/11/16: 4h
- 28/11/16: 2h
- 29/11/16: 4h
- 4/12/16: 4h
- 5/12/16: 2h
- 6/12/16: 3h
- 11/12/16: 4h

*Vigorito Ivan:*

- 20/11/16: 4h
- 21/11/16: 2h
- 27/11/16: 4h
- 28/11/16: 2h
- 29/11/16: 4h
- 4/12/16: 4h
- 5/12/16: 2h
- 6/12/16: 3h
- 9/12/16: 4h
- 10/12/16: 2h
- 11/12/16: 2h

# 8. References Used tools

The tools we used to create this DD document are:

- *Github*: for version controller
- *STARUML*: UML sequence
- *Draw.io*: general Diagrams
- *Balsamiq Mockups3*
- *Microsoft Word 2015*