Politecnico di Milano

Computer Science and Engineering's master degree course

Department of Electronics and Information

PowerEnjoy
Design Document

Version 1.1

Release Date: 07/02/2017

Customer: Eng. Elisabetta DI NITTO

Authors:

Eng. Marco FERNI	Id. 877712
Eng. Angelo Claudio RE	Id. 877808
Eng. Gabriele TERMIGNONE	Id. 877645

Contents

Chapter 1

Introduction

1.1 Purpose

This document extends the PowerEnjoy's RASD by providing a functional description of the system and more technical details. An overview of the system's design will be given through UML's Component, Deployment and Sequence diagrams, and we'll elaborate more on the User Interface part via the explanation of some already exposed mockups. The interactions between components, the deployment cycle and the runtime behavior of the system will be discussed, aiming to reach a level of description detailed enough for the software development to proceed with an understating of what are the software and hardware choices to be taken and how the system should be built.

1.2 Scope

PowerEnjoy's main goal is to help people move around easier, without having to rely on their personal transport; a secondary goal is the reduction of cities' pollution and acoustic noise. To utilize Power Enjoy, a user needs to successfully complete a registration procedure, in which he's asked to insert his IDs and driving licenses, together with some personal data. The system allows the now registered user to rent a car for a limited amount of time. The user can now start to look for a car by entering either his current position (detected by using his smartphone's GPS) or a specific location, chosen on the map, that he'll need to reach by himself. Frauds mechanism, like the 1 EUR fee, are applied to prevent abuse. The system policies encourage a smarter use of our service, by offering discounts to those who share a trip together. Users are also strongly suggested to leave a car in or near a PowerEnjoy's parking lot.

1.3 Definitions, Acronyms, Abbreviations

- *Cost of the Trip*: raw price of the service calculated only on the base of the duration of the car's usage, before discounts or additional charges are applied.
- *Virtuousness coefficient*: the factor by which to multiply the cost of the trip to get the amount of the bill. Its initial value is 1.
- *Supervisor*: a company employee who work at the Car hub controller.
- *Recharge on site*: a company procedure: a worker is sent to recharge a low car that was parked detached from the power grid.
- *Car recovery*: a worker is sent to retrieve a car that has been forgotten outside a safe area and move that car into in one of these lot.
- *Guest*: a person who is not already registered to the system.
- *User*: a registered customer.
- *RASD*: Requirement Analysis and Specification Document
- *DD*: Design Document
- *DB*: Database
- *ER Diagram*: Entity Relation Diagram

1.4 Reference Documents

- The PowerEnjoy's RASD (Requirement Analysis and Specification Document)
- The project's Assignments PDF
- Old years projects
- The DD standards
- Software Engineerign course slides
- UML Specification Standards

1.5 Document Structure

- **Introduction:** this section introduces the design document. It contains a justification of its utility and indications on which parts are covered in this document that are covered by RASD.
- **Architectural Design:** this section is divided into different parts:
    1. **Overview:** this sections explains the division in tiers of our application
    2. **High level components and their interaction:** this sections gives a global view of the components of the application and how they communicate
    3. **Component view:** this sections gives a more detailed view of the components of the applications
    4. **Deploying view:** this section shows the components that must be deployed to have the application running correctly.
    5. **Runtime view:** sequence diagrams are represented in this section to show the course of the di erent tasks of our application
    6. **Component interfaces:** the interfaces between the components are presented in this section
    7. **Selected architectural styles and patterns:** this section explain the architectural choices taken during the creation of the application
    8. **Other design decisions**
- **Algorithm Design:** this section describes the most critical parts via some algorithms.
- **User Interface Design:** this section presents mockups interaction.
- **Requirements Traceability:** this section aims to explain how the decisions taken in the RASD are linked to design elements.

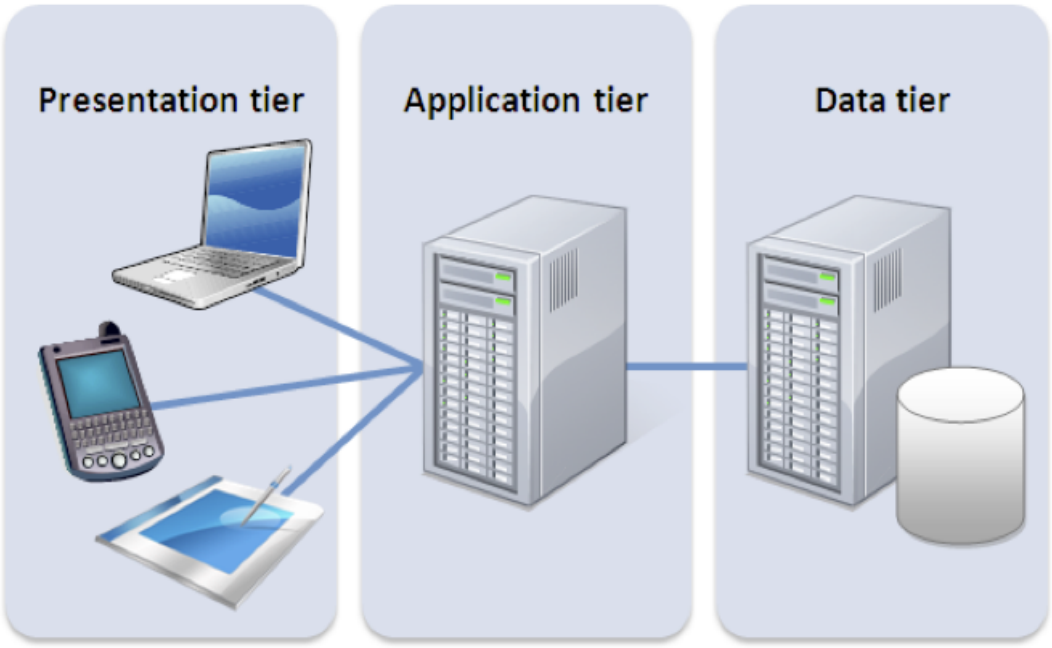
Chapter 2

Architectural Design

2.1 Overview

2.1.1 General Structure

A Three-Tier architecture is used in this system:



More information about the chosen architectural style can be found in the section 2.6.

2.1.2 High level components and their interaction

The following diagram gives a brief introduction of the system by showing the main components of the architecture and their basic relations:

- Central System
    - Application Server
    - DB
- Mobile App
- Car
    - Sensors
    - Display
- Supervisor

The communications between the central server and the mobile devices (user's device and car's devices) are performed using RESTful API through the mobile network (3/4G).

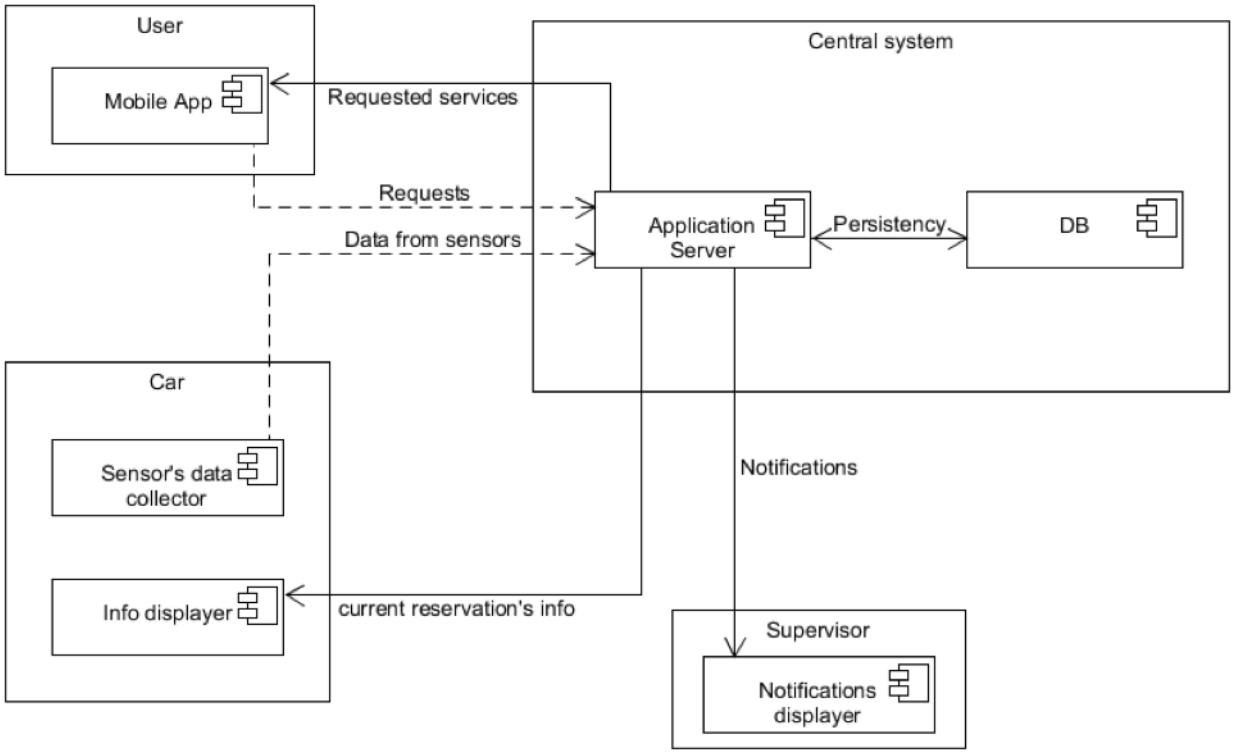


Figure 2.1: *High Level Architecture*

2.2 Component view

NB: the dashed arrows represent the dependence of a component from the instances of the pointed part.

DB run on the same server of the application server, so for simplicity we represented it as a normal component in the component diagram).

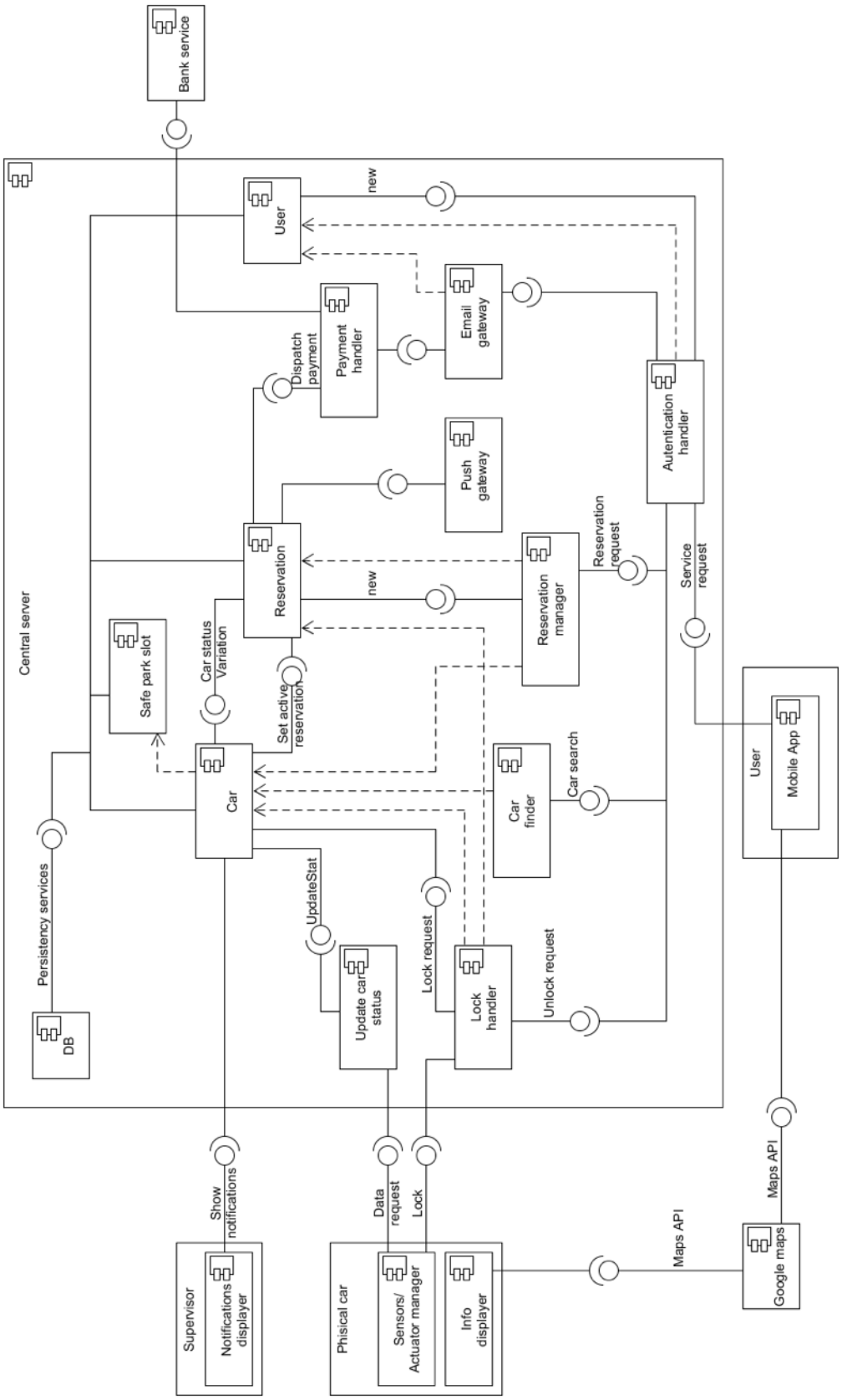
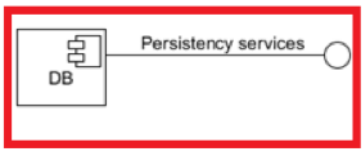


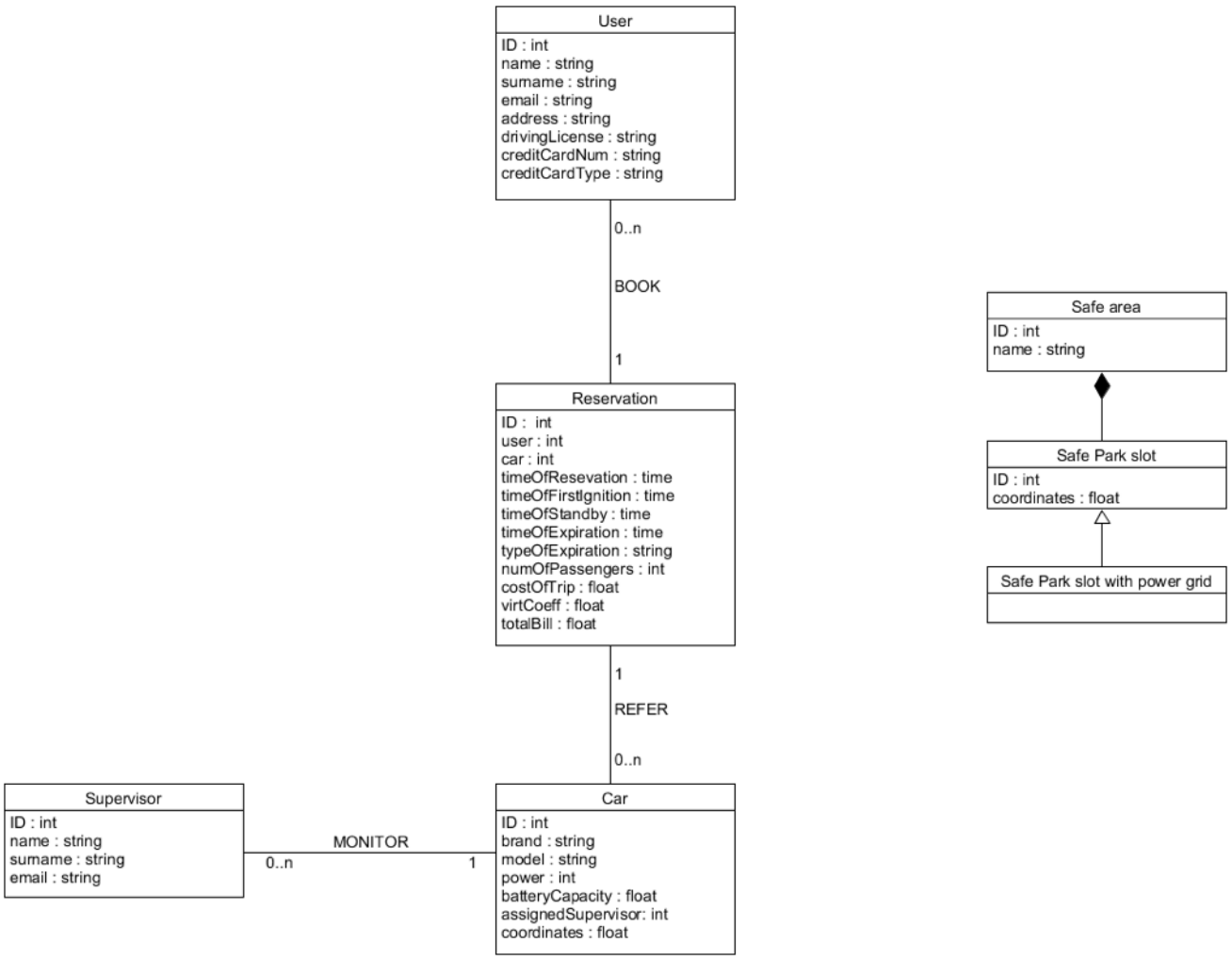
Figure 2.2: *Component Diagram*

2.2.1 DB Component and Interface

This component represents the Database used by the application server to store persistent data.

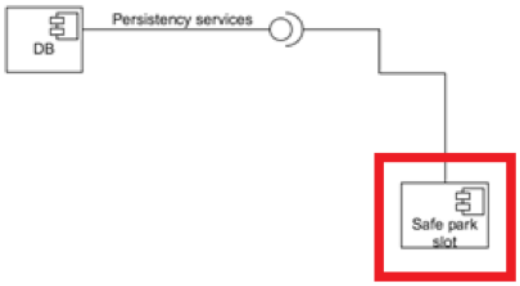


The Database's structure is explained in the following ER Diagram:



The interface *"persistency services"* represents all the methods that other components use to communicate with the database; these methods have to guarantee the atomicity, consistency and security of the transactions.

2.2.2 Safe park lot Component



This component represents a simple entity bean; it's only meant to represent the entities *"Safe park slot"* of the database.

2.2.3 User Component and Interface

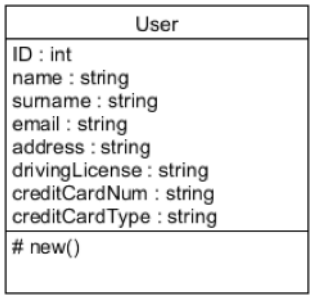


Figure 2.3: *class representation*

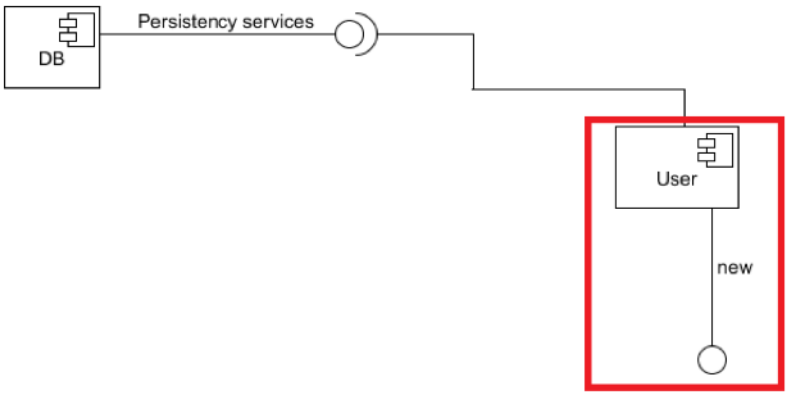


Figure 2.4: *component*

This component is an entity bean too. The *"new"* interface makes possible to create new *"User"* object, for example when a new customer registers to the application; it is up to the component to send the new information to the database.

2.2.4 Authentication handler component and interface

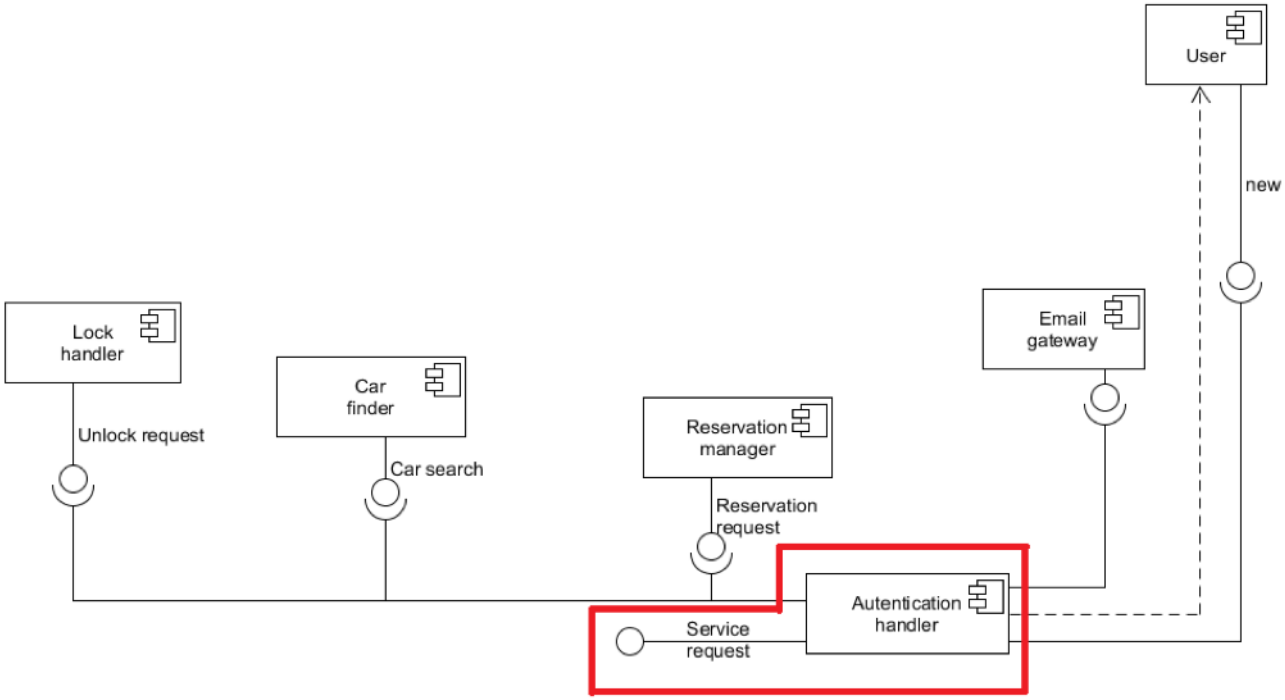


Figure 2.5: *component*

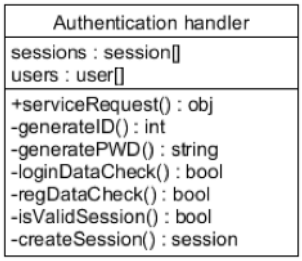


Figure 2.6: *class representation*

This component has the purpose of verifying the correctness of the data submitted by the user during the login phase, and to keep track of the user's session; another task of this component is to verify that each request coming from a user is made during a valid session: if this is true, the component manages to call the right method.

The last task of this component is to manage the registration of new users: after having verified the correctness of the data submitted, the *"authentication handler"* calls the *"new"* method of the component *"user"*, and uses the *"email gateway"* component to send the confirmation email containing the password to the new user.

2.2.5 Reservation component and interface

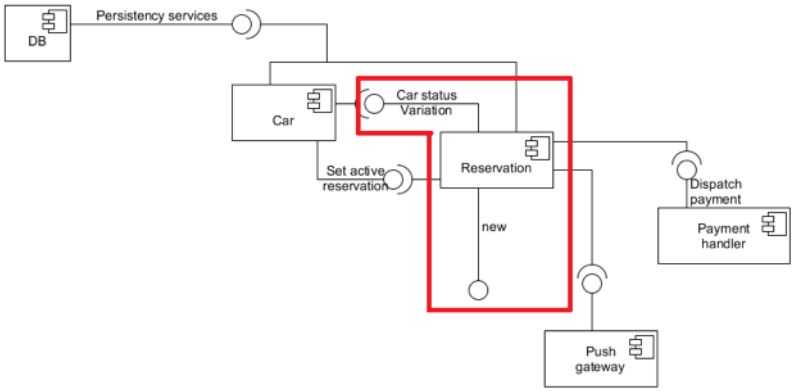


Figure 2.7: *component*

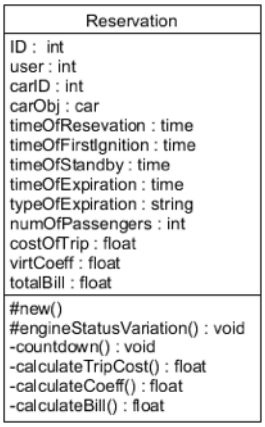


Figure 2.8: *class representation*

This component contains all the information about a certain reservation that are needed for the runtime functionality of the system.

The *"new"* interface makes possible to create new *"reservation"* object. The method associated with this interface is responsible to initialize part of the object's attribute, to launch a 60 minutes' countdown, and to use the *"set active reservation"* interface of the car object to which the reservation is associated. The *"Car status variation"* is used to notify the reservation object about the changing of specific attributes on the state of the Associated car; this is useful because in this way the reservation can detect when the first engine's ignition occurs, so the system can register the number of passenger, or when a car is parked (in a safe area or not), in order to correctly launch the countdown, or ultimately, when a parked car is turned back on, in order to delete the running countdown. The private method *"countdown"* is used to determine when a reservation should expire: when it terminates, it sets the *"timeOfExpiration"* and *"typeOfExpiration"* attributes of the object, use the *"set active reservation"* interface of the car object in order to set the *"actualReservation"* attribute of the car to NULL, and launch sequentially the three method *"calculateTripCost"*, *"calculateCoeff"* and *"calculateBill"*; after that, it calls the *"payment handler"* component. The last thing done by the method if it reaches the end of the countdown, is to send the information about the expired reservation to the database. The *"push gateway"* component is used to send specified information about the reservation to the user's app and to the car's monitor.

2.2.6 Car Component and interface

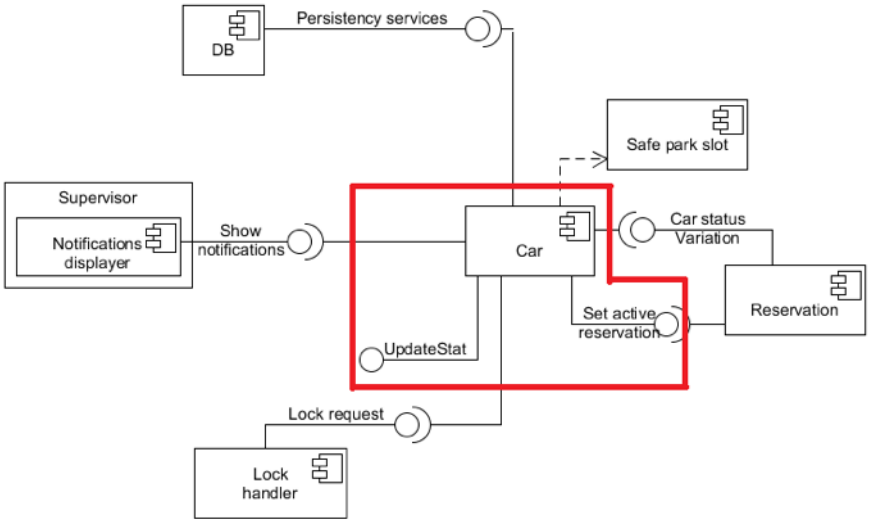


Figure 2.9: *component*

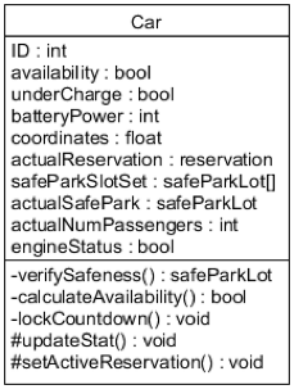


Figure 2.10: *class representation*

The *"car"* component represents the entity "car" of the database, but contains also other information that are needed for the runtime functionality.

The *"updateStat"* interface permits to an external component to keep the car object's attributes up to date; it is also responsible to call the *"verifySafeness"* and *"calculateAvailability"* methods.

The *"set active reservation"* method is used by a new reservation to associate itself to the reserved car; it is also used by a reservation when it is about to expire, to set the *"actualReservation"* attribute of the associated car to null. It is also responsible to call the *"calculateAvailability"* methods.

When the methods associated with this interfaces are called, they perform various checks on updated data end eventually perform calls to another component:

Engine switched off and no passenger on board ⇒ car.lockCountdown + car status variation

Engine switched on ⇒ car status variation

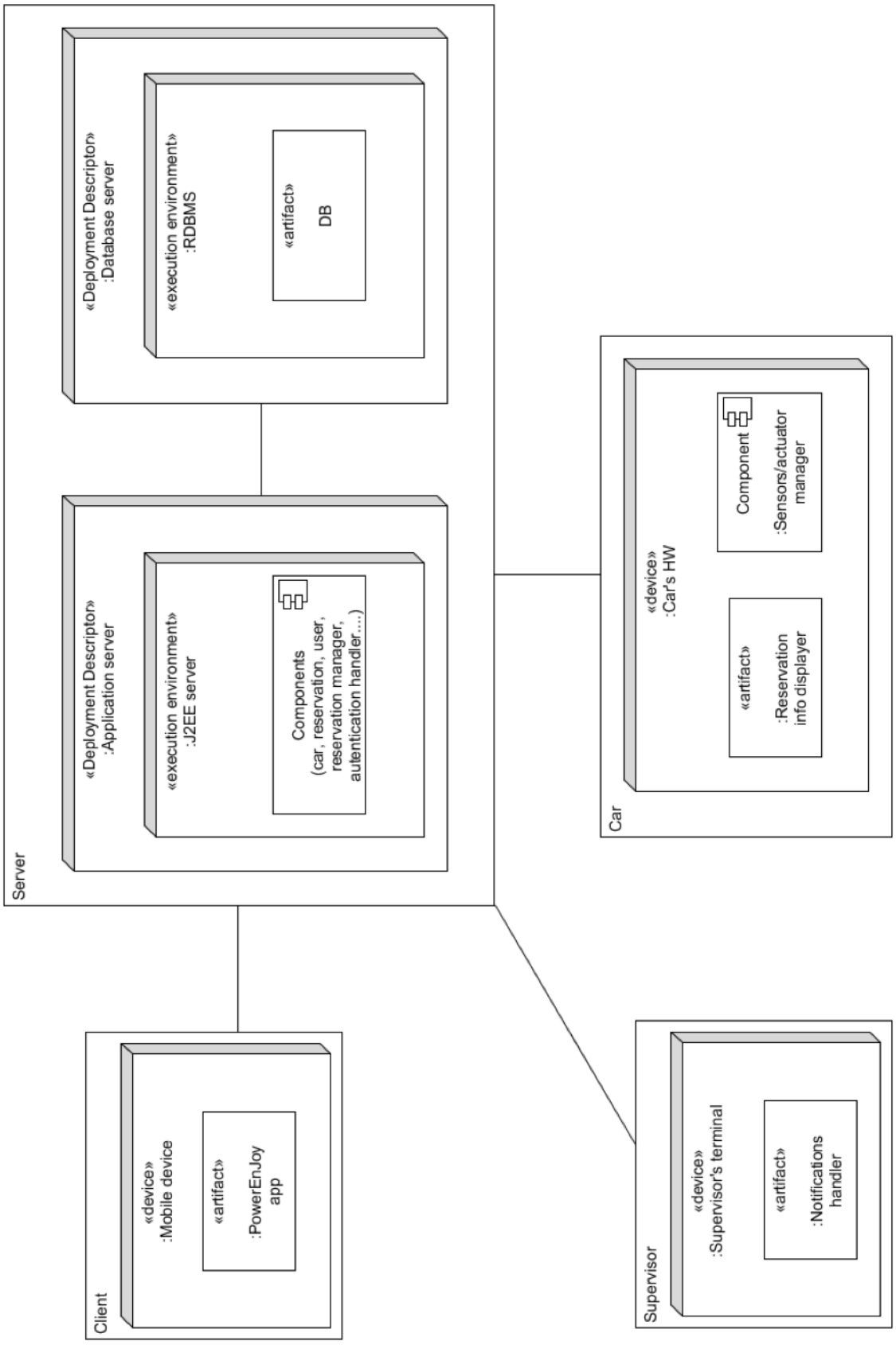
Reservation Expired && battery power < 20% && car not under charge ⇒ show notification

Reservation Expired && car is not in safe area ⇒ show notification

The *"lockCountdown"* method perform a 60 seconds countdown at the end of which it will be perform a lock request.

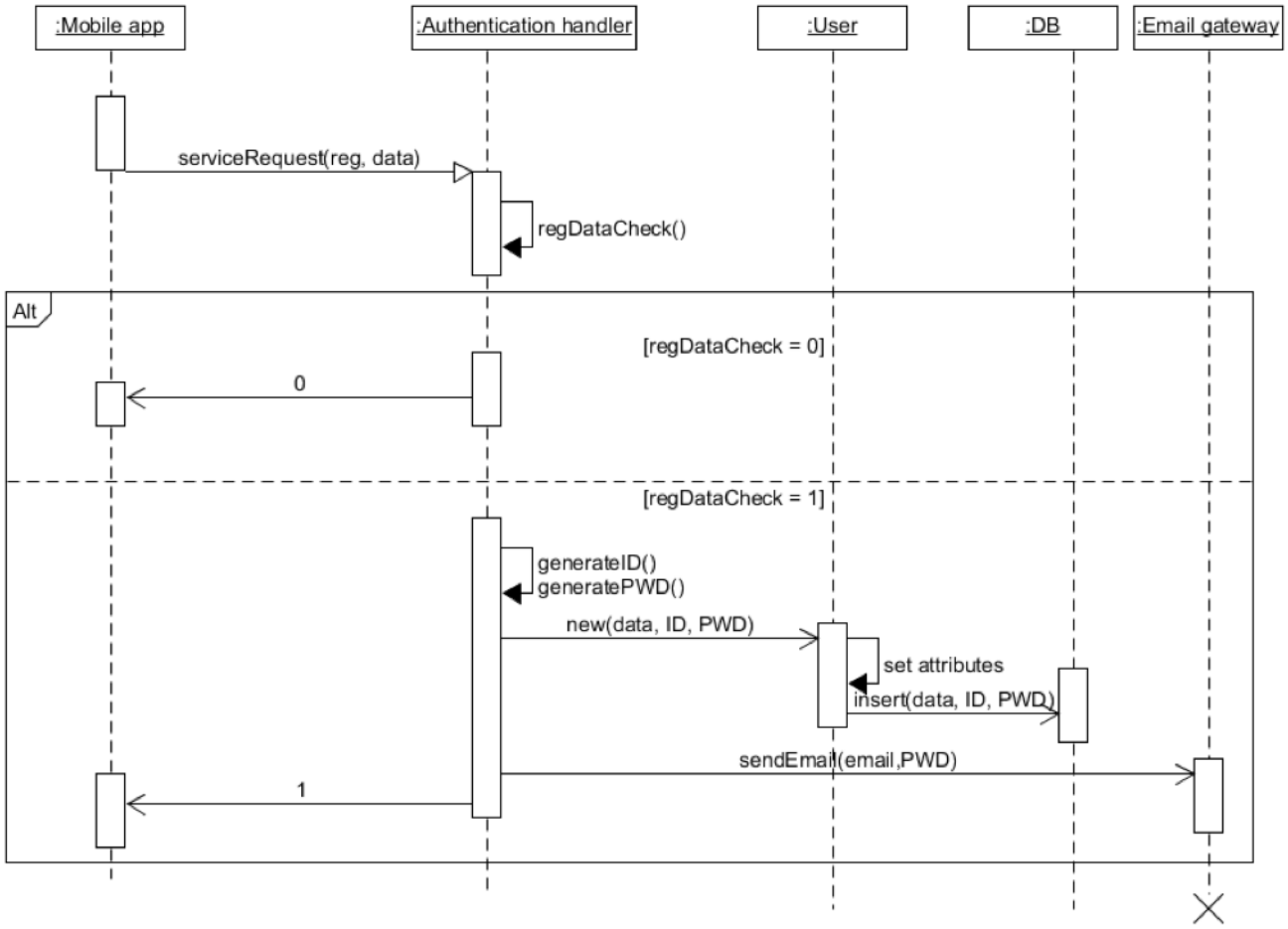
2.3 Deployment view

The folliwing depicts the deployment diagram of the PowerEnjoy system.

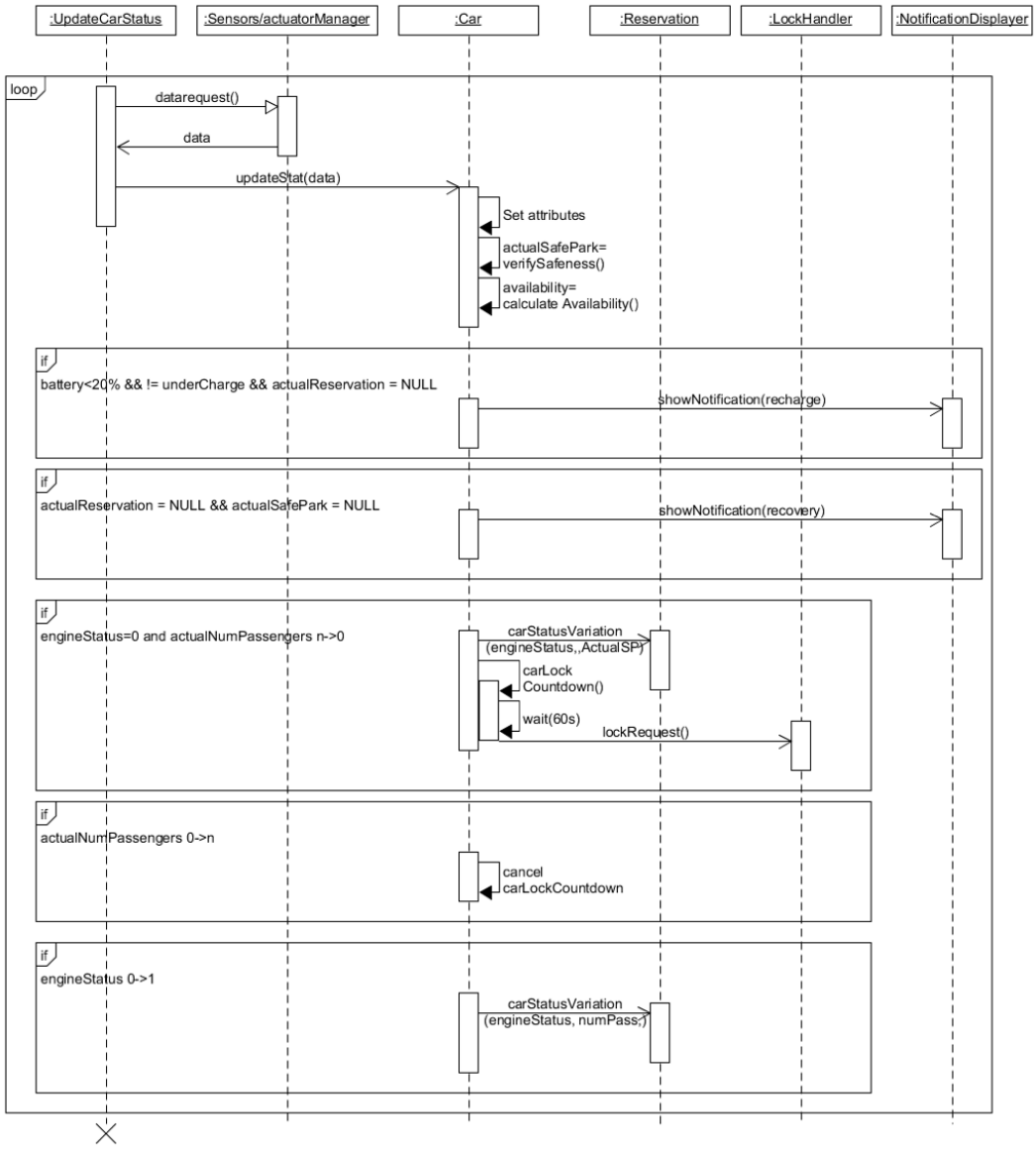

Server
«Deployment Descriptor»
:Database server
«execution environment»
:RDBMS
«artifact»
DB
«Deployment Descriptor»
:Application server
«execution environment»
:J2EE server
Components
(car, reservation, user,
reservation manager,
autentication handler....)
Car
«device»
:Car's HW
Component
:Sensors/actuator
manager
«artifact»
:Reservation
info displayer
Client
«device»
:Mobile device
«artifact»
:PowerEnJoy
app
Supervisor
«device»
:Supervisor's terminal
«artifact»
:Notifications
handler

2.4 Runtime view

2.4.1 Registration Sequence



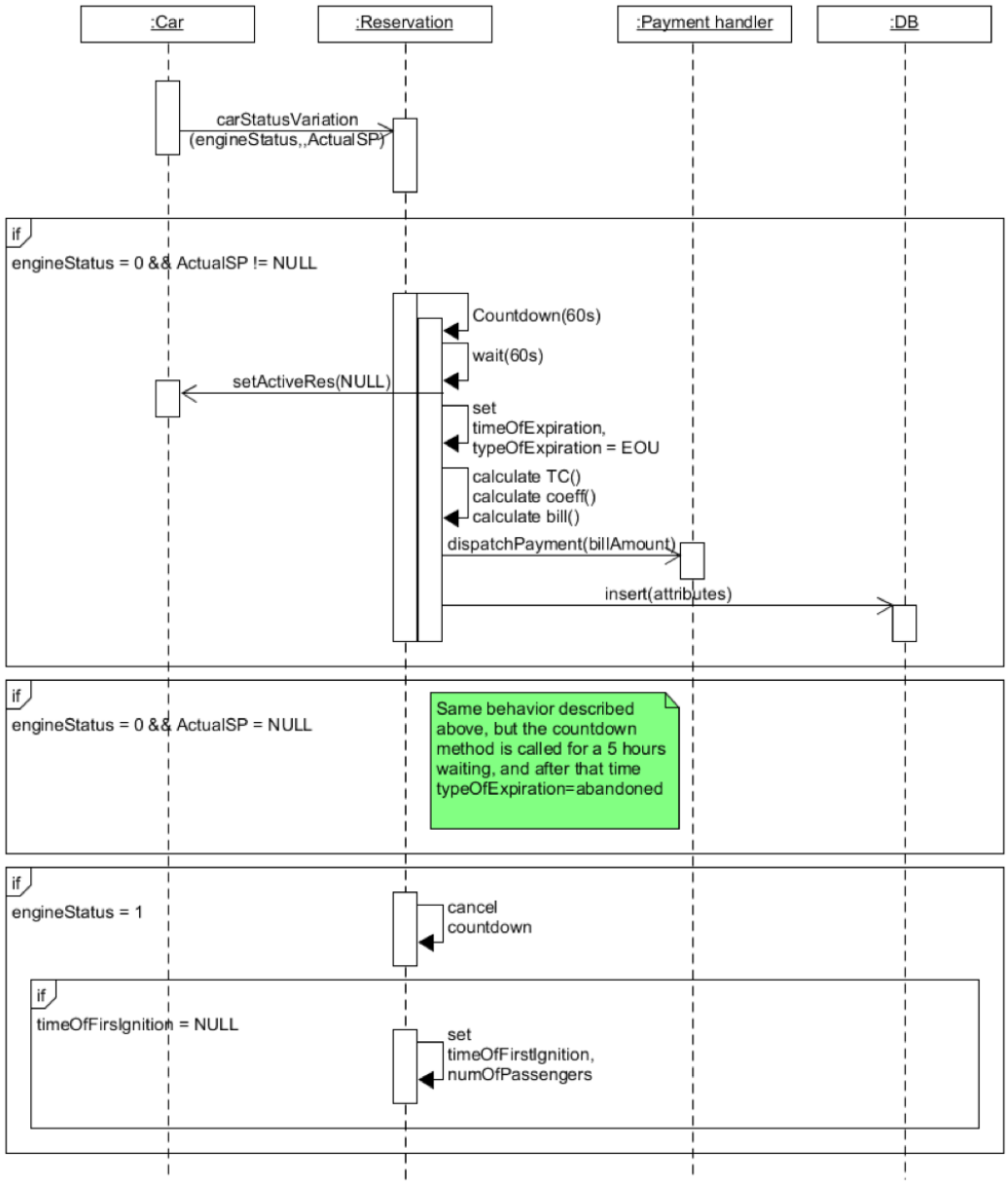
2.4.2 Update car status sequence



The *"carStatusVariation"* call will be explained in the next sequence diagram

2.4.3 Car status variation Sequence

This sequence diagram shows how variations of car's status influence the reservation.



2.5 Component interfaces

All the relevant interfaces have been already discussed in the section B.

2.6 Selected architectural styles and patterns

As previously introduced, a Three-Tier architecture is used in this project:

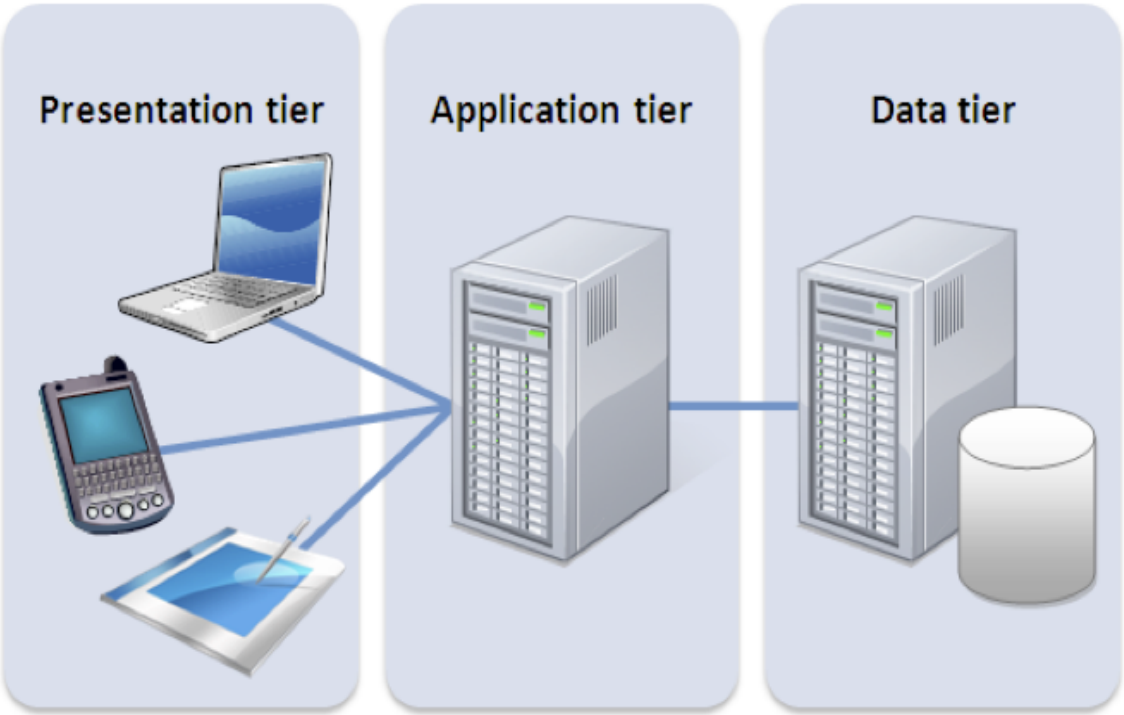


Figure 2.11: *Three Tiers Architecture*

1. Presentation Tier (Client Tier)
    a. Composed mainly by:
        i. The be the app used by the user to find and rent a car
        ii. The screen in the rented car and
        iii. Possibly, the tools used by the Supervisor.

   b. It's the layer that interacts with the server and displays useful information.

2. Application Server

   a. Where all the computations (reservations, locking/unlocking cars, payment) occur.

3. Data Tier

   a. MySQL DBMS

Elaborating more, a three tier architecture like the one described here, represents a situation where the storage logic (database) is separate from the business logic (application) and from the presentation.

This kind of separation of duties allows for:

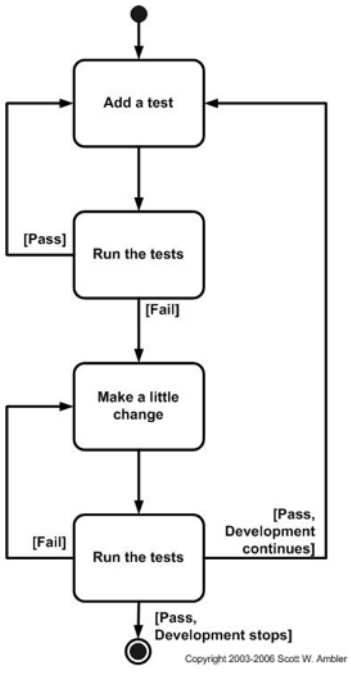
1. An easier implementation of the system
2. An efficient scalability (future proof architecture). We could expand the DB, should we need a bigger one, change the UI, etc. with a low impact on the untouched tiers.
3. It is also a matter of security, allowing for a better isolation of different processes.

The system can then be described as having an object-oriented architecture:

- All components are object
- Connectors are messages and method invocations
- The internal representation is hidden from other objects

That gives us all the benefits of the object-oriented paradigm, with focus on the scalability (as explained before), reusability and testability.

The system is meant to be developed with a test driven development (TDD): all the goals discussed in the RASD document (and in the following pages), will be converted into test cases and then programmers will code aiming to pass those tests.



The software will then evolve adding more goals/tests.

As always, integration tests need to pass with the addition of every new module.

The tools used in the testing process are not discussed in this document.

In addition, the approach used in this system is a top-down one: we started from the agreed specifications, we discussed the master features needed (the high-level structure of the system) and then we started to work down to detailed decisions about low-level constructs and individual algorithms.

Finally, a system like this one must comply with a number of QoS metrics like:

- **Reliability:** what if the user rents a car, parks it in a shopping mall car park and then, after buying a lot of frozen foods, it finds out that the service isn't working and the car won't open?
- **Availability:** the service must be continuously available to the user; we can't allow downtimes in a competitive business like this one.
- **Usability:** a not so user friendly app (or car's display) will make us lose the customer or we'll need to invest a lot of resources in Public Relations and Customer Assistance.

2.7 Other design decisions

- The smartphone app will interact to the user via Push Notifications.
- All kinds of mobile devices are supported (Android, iOS and Windows Phone).
- Google Maps, with its API, is used in the app, allowing the user to:

  - locate a car (using the Google Maps' plotting capability) in the world before a reservation
  - save money, when the saving option is chosen, thanks to the Route Suggestion Mode that draws an ideal path from his position to the parking lot minimizing his expenses.

- A premium GM's plan is expected to be used, due to the high volume of API requests needed for this app to correctly function.

Chapter 3

Algorithm Design

There will be shown 3 of the functions that concern the system
The language used for the functions is java.

3.1 CalculateCoeff function

The function that computes the virtuosness coefficient could be:

```
private float CalculateCoeff() {
    if (this.GetEngineStatus == false)
        /*
            If the system detects the user took at least two other passengers
            onto the car, the system applies a discount of 10% on the last ride.
        */
        if (this.numOfPassengers == 3) {
            this.virtCoeff =- 0.1;
        }

        /*
            If a car is left with no more than 50% of the battery empty,
            the system applies a discount of 20% on the last ride.
```

```
*/
if (this.carObj.GetBatteryPower() > 50) {
    this.virtCoeff =- 0.2
}

/*
    If a car is left at special parking areas
    where they can be recharged and the user takes care
    of plugging the car into the power grid,
    the system applies a discount of 30% on the last ride.
*/
if (this.carObj.GetUnderCharge() == true) {
    this.virtCoeff =- 0.3
}

/*
    If a car is left at more than 3 KM
    from the nearest power grid station
    or with more than 80% of the battery empty,
    the system charges 30% more on the last ride
    to compensate for the cost required
    to re-charge the car on-site.
*/
if (this.carObj.GetBatteryPower() < 20
 || safearea.SearchNearestPowerGrid(
 this.carObj.GetCoordinates()) > 3000) {
    this.virtCoeff =+ 0.3;
}
```

```
    }
    return this.virtCoeff;
}
```

3.2 CalculateAvailability function

The function that updates the availability of a car could be:

```
protected Bool CalculateAvailability() {
    if (GetActualReservation(() == null
     && GetBatteryPower() > 20
     && GetActualSafePark() ! = null) {
        SetAvailability() = true;
    } else {
        SetAvailability() = false;
    }
        return GetAvailability();
}
```

3.3 ShowNotification function

The function that shows to a supervivor a car could be:

```
protected void ShowNotification() {
   if ((GetActualReservation().GetTypeOfExpiration() == "expired")
    &&(GetActualSafePark().GetCoordinates() ! = GetCoordinates()
     || (GetBatteryPower() < 20 && GetUnderCharge() == false)) {
      NotifySupervisor();
   }
}
```

Chapter 4

User Interface Design

The user interface has been designed in according with the *MVC* design pattern.

4.1 My Reservation Interface

4.1.1 Registration Flow

These are the main steps a user has to take to register on the application.

Step 1 of 4: Open App PowerEnjoy

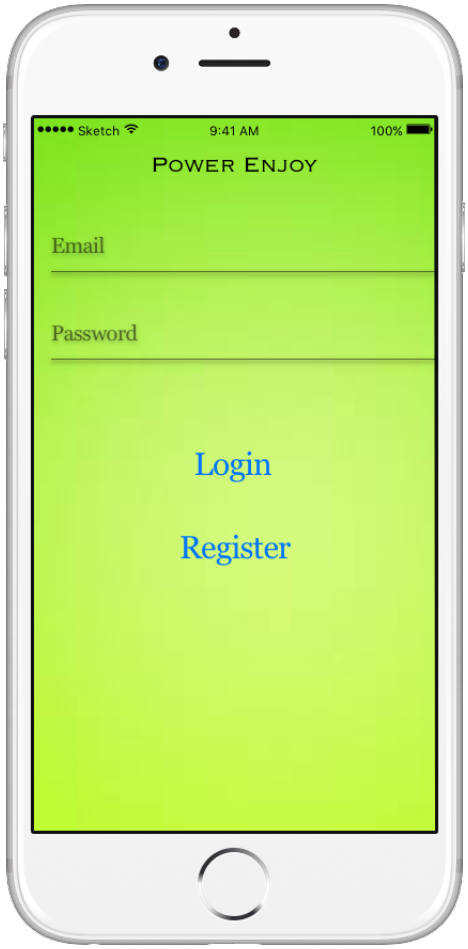
DESCRIPTION

The user opens the PowerEnjoy app from the HomePage of his smartphone.

COMPONENTS INVOLVED

Mobile app.

Step 1 of 4: Open App PowerEnjoy



DESCRIPTION

The user presses the *Register* button.

COMPONENTS INVOLVED

Mobile app.

Step 2 of 4: Fill in data in first registration form

DESCRIPTION

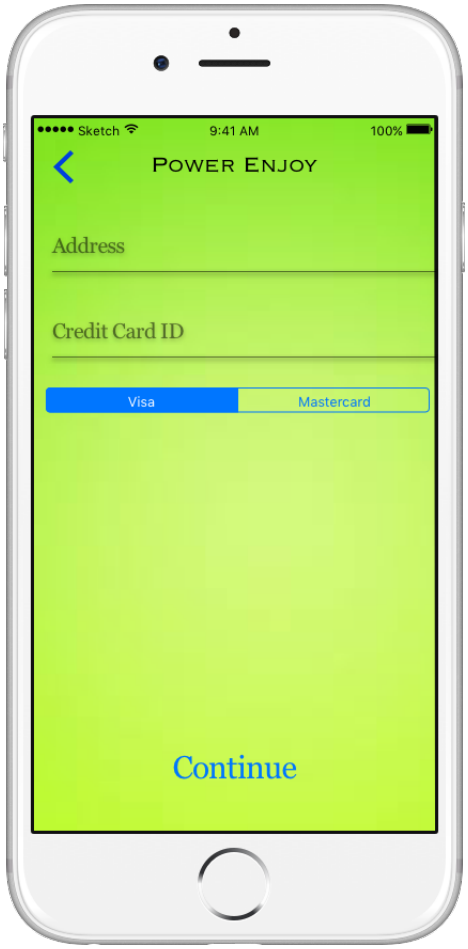
The user inserts the personal data and presses on the *Continue* button.

If one of the information inserted is not valid, it will be shown an error message, and all the invalid fields, it will be highlighted in red.

COMPONENTS INVOLVED

Mobile app.

Step 3 of 4: Fill in data in second registration form



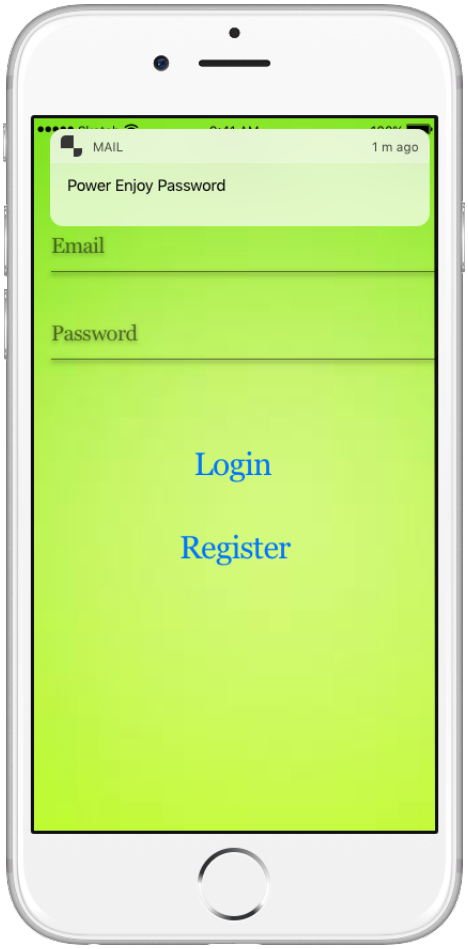
DESCRIPTION

The user inserts here his address, his credit card ID (specifying the ID's type) and whereupon he presses the *Continue* button.

COMPONENTS INVOLVED

Mobile app.

Step 4 of 4: Receive password via email address



DESCRIPTION

Up to now all the information filled in the previous screen are treated locally.

Now the information will be send to the application server that will store the user's data in the DB.

Whereupon the registration is completed and the user receives the password of his new account from *Email gateway* component.

COMPONENTS INVOLVED

Mobile app, Authentification handler, User, DB and email gateway.

4.1.2 User Trip

These are the main steps a user has to take to rent a car.

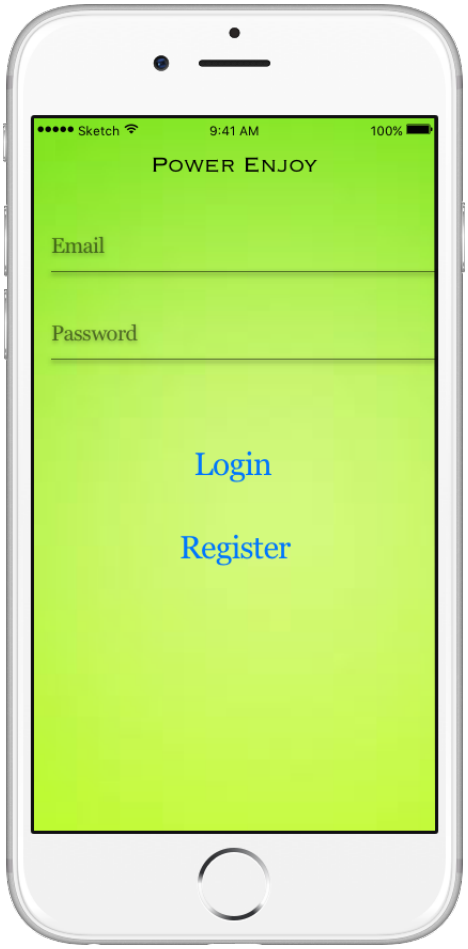
Step 1 of 6: Open App PowerEnjoy

DESCRIPTION

COMPONENTS INVOLVED

Mobile app.

Step 2 of 6: Make the Login to the system



DESCRIPTION

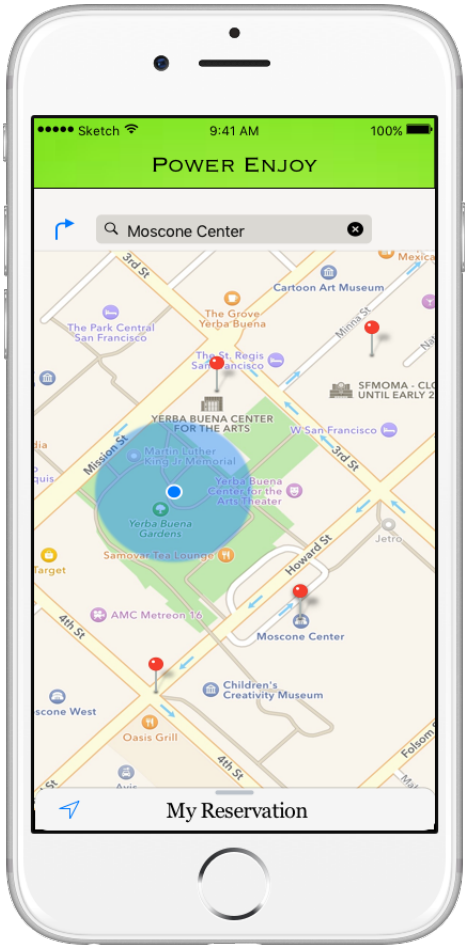
The user fills in the Email and Password, and whereupon he clicks the *Login* button.

The application server receives the login credentials and if they are right, it sends a feedback to the app to allow user to start using the app.

COMPONENTS INVOLVED

Mobile app, Authentification handler, User and DB.

Step 3 of 6: Search a car in the map and reserve it



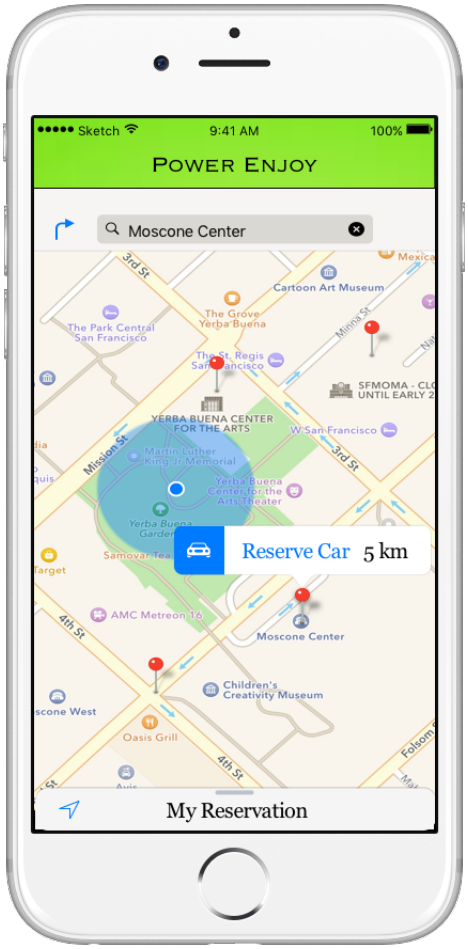
DESCRIPTION

The user can select one of the car available in the map.

COMPONENTS INVOLVED

Mobile app, Google Maps, Car Finder and DB.

Step 3 of 6: Search a car in the map and reserve it



DESCRIPTION

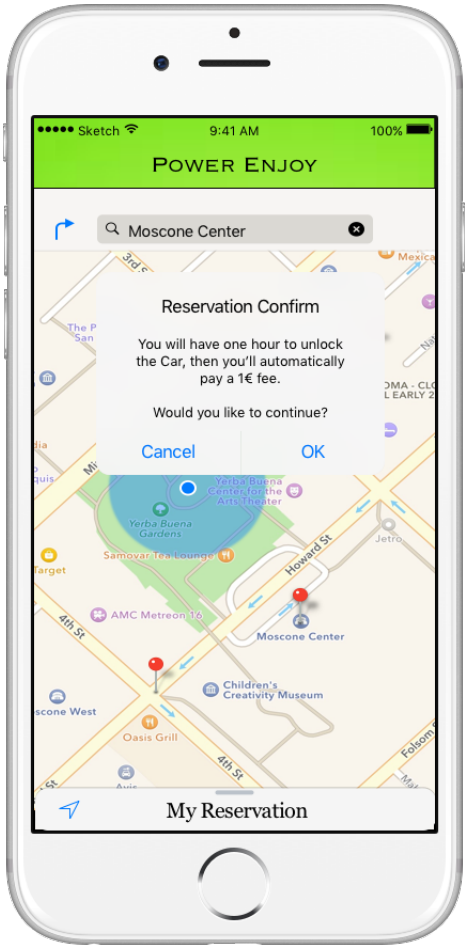
Once a car is selected, the user can reserve it.

The user can see the distance from his current position.

COMPONENTS INVOLVED

Mobile app, Google Maps, Reservation Manager and DB.

Step 3 of 6: Search a car in the map and reserve it



DESCRIPTION

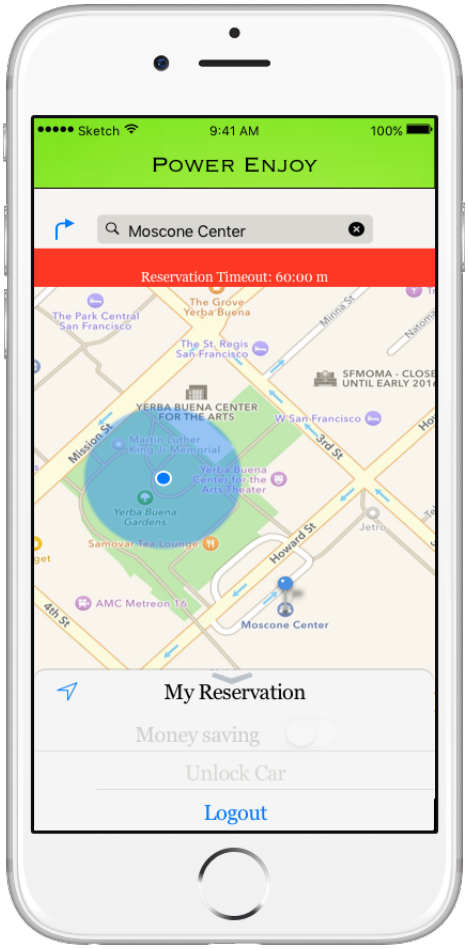
The user must confirm the reservation.

If so, the request'll be handle by the *Reservation handler* component of the application server that'll stores the registration in the DB, and it'll prevent that another user could reserve the same car.

COMPONENTS INVOLVED

Mobile app, Google Maps, Reservation Manager and DB.

Step 4 of 6: Reach the car



DESCRIPTION

The user has to reach the car in 60 minutes.

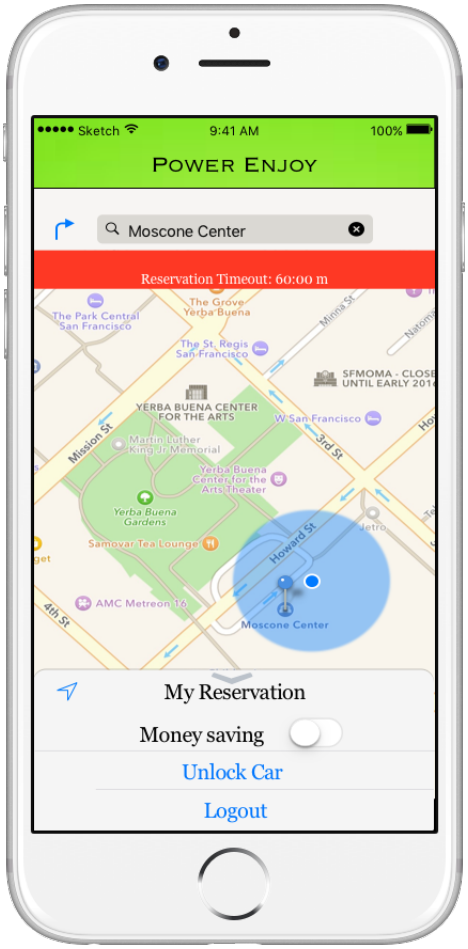
A notification shows the remaining time.

As long as the user is 100 m farther from the car, he can't unlock it.

COMPONENTS INVOLVED

Mobile app, Google Maps and Push gateway.

Step 4 of 6: Reach the car



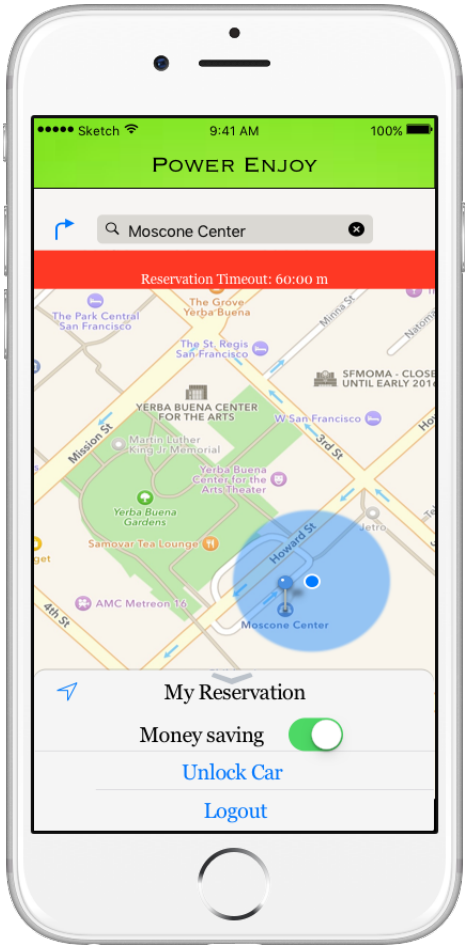
DESCRIPTION

As soon as the user is less distance than 100 m than the car, they can unlock the car.

COMPONENTS INVOLVED

Mobile app, Google Maps, Lock Manager and Push gateway.

Step 4 of 6: Reach the car



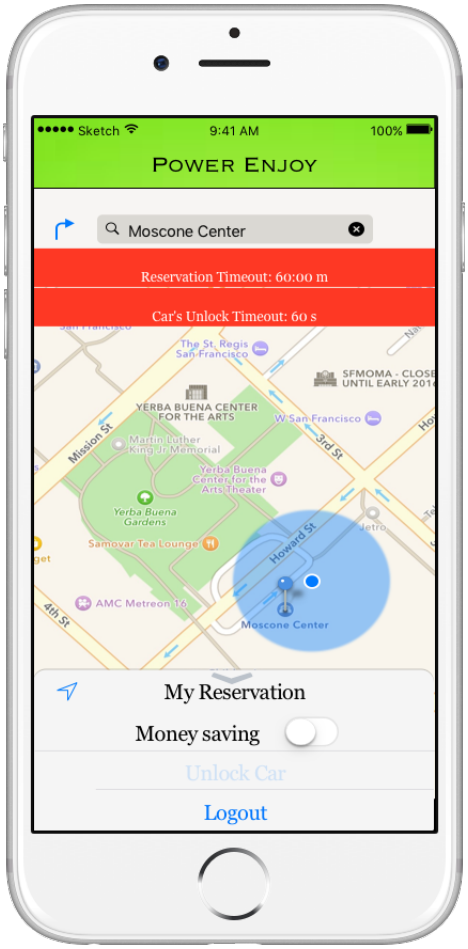
DESCRIPTION

Eventually before unlocking the car, the user should enable the *Money Saving* option.

COMPONENTS INVOLVED

Mobile app, Google Maps, Lock Manager and Push gateway.

Step 5 of 6: Unlock the car and start the trip



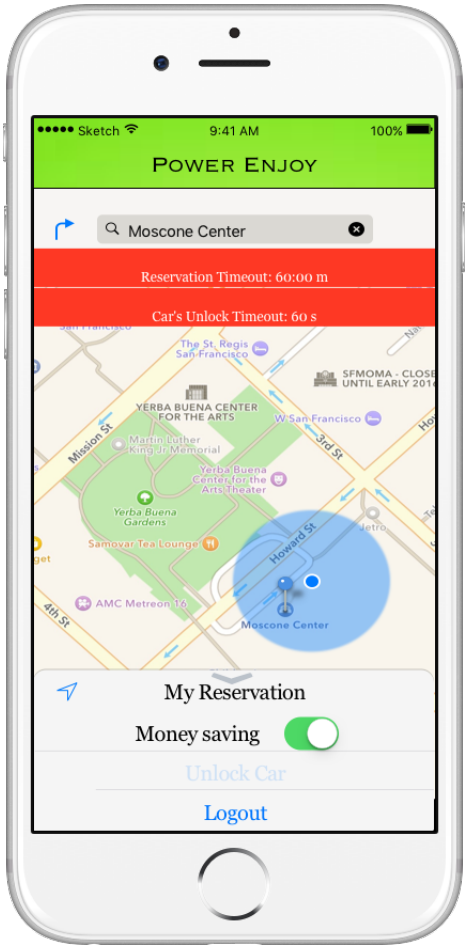
DESCRIPTION

As soon as the user unlocks the car, he has 60 seconds to enter into the car.

COMPONENTS INVOLVED

Mobile app, Google Maps, Lock Manager and Push gateway.

Step 5 of 6: Unlock the car and start the trip



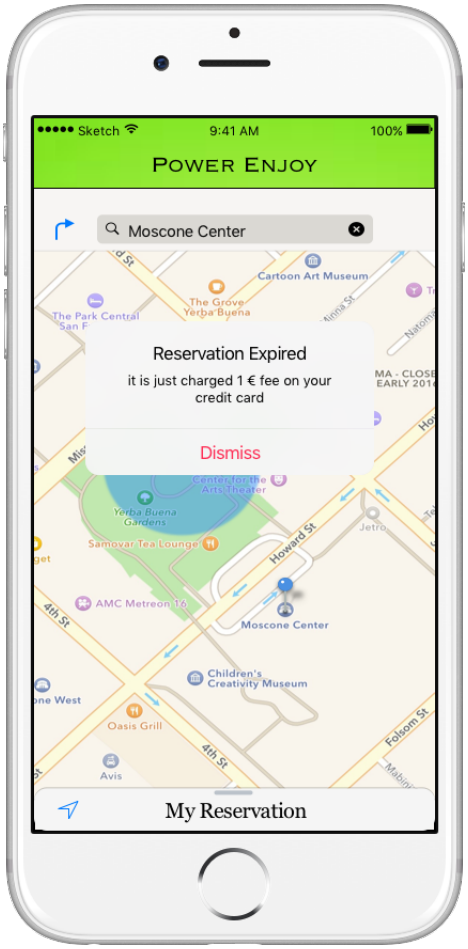
DESCRIPTION

Also in this case, the user can enable *Money Saving* option before he gets into the car.

COMPONENTS INVOLVED

Mobile app, Google Maps, Lock Manager and Push gateway.

Step 5 of 6: Unlock the car and start the trip



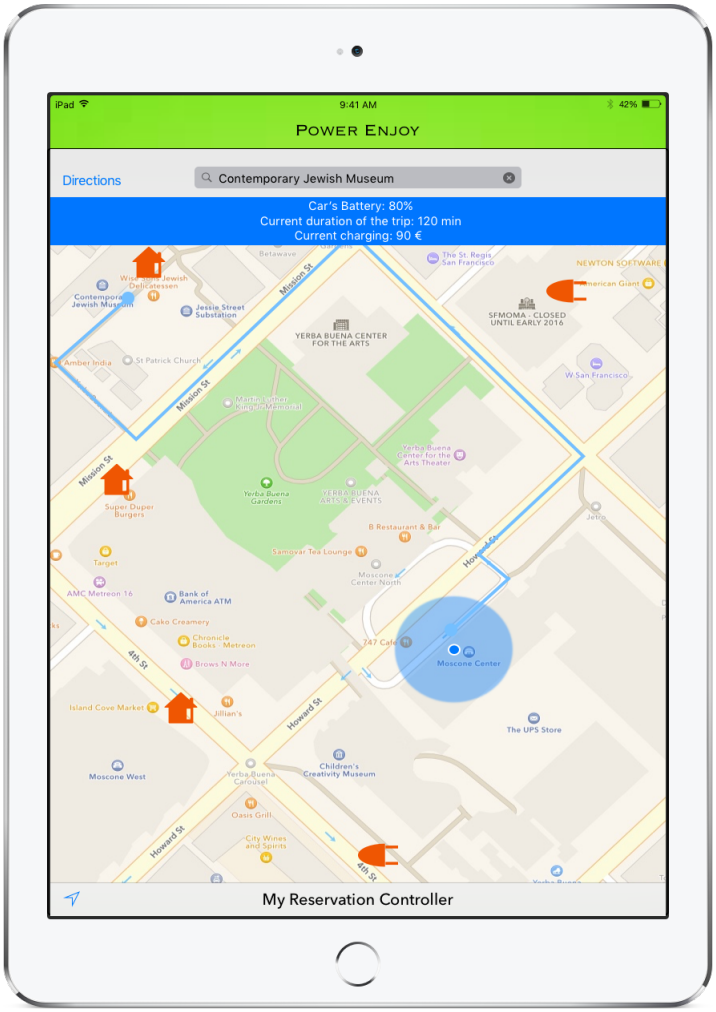
DESCRIPTION

If the user doesn't get in the car in 60 seconds, the reservation will expire and they will need to pay a 1 EUR fee.

COMPONENTS INVOLVED

Mobile app, Google Maps, Reservation Manager, Lock Manager and Push gateway.

Step 5 of 6: Unlock the car and start the trip



DESCRIPTION

The user can now set his destination by using the car's display.

The safe areas are visible on the map.

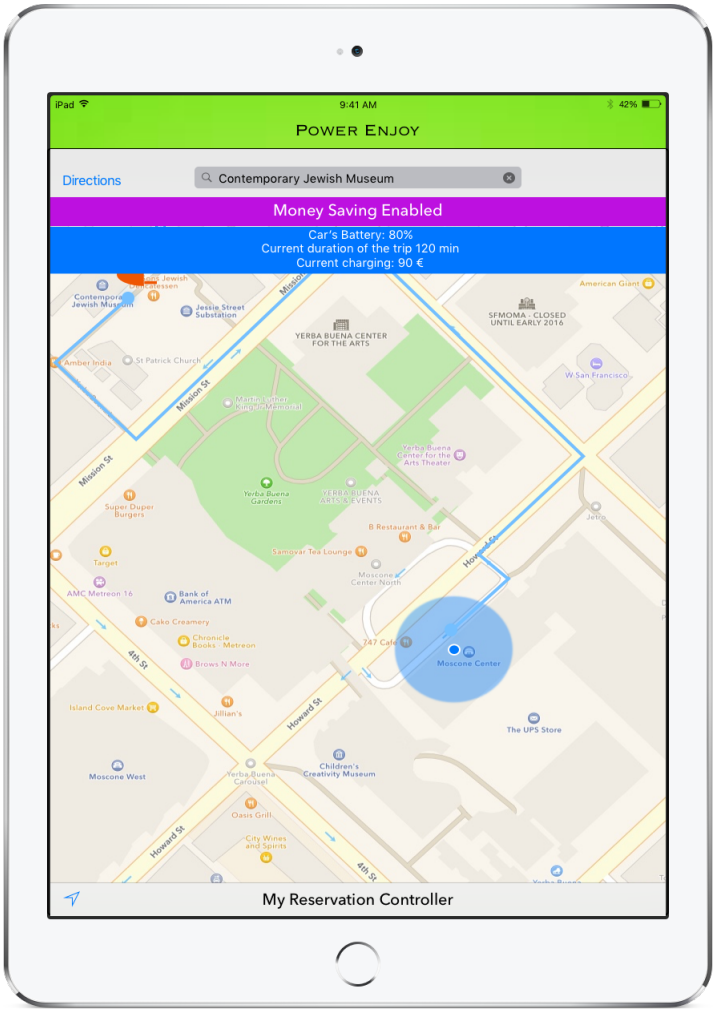
The orange home symbol depicts a safe area without power grid.

The orange plug symbol depicts a safe are with power grid.

COMPONENTS INVOLVED

Mobile app, Google Maps, Info displayer, Safe car slot, DB, Update car status, Sensors/Actuator manager and Push gateway.

Step 5 of 6: Unlock the car and start the trip



DESCRIPTION

The user can see in the screen the trip's info like car's battery, the current duration of the trip and the current charging.

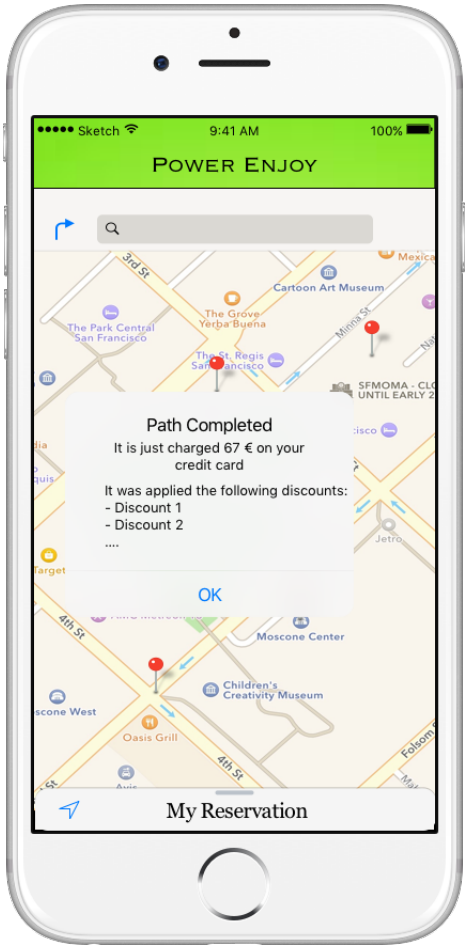
A status text notifies whether the *Money Saving* option is enabled.

If *Money Saving* is enabled the user can see only the safe area computed by system according to user's destination.

COMPONENTS INVOLVED

Mobile app, Google Maps, Info displayer, Safe car slot, DB, Update car status, Sensors/Actuator manager and Push gateway.

Step 6 of 6: Reach the destination and exit



DESCRIPTION

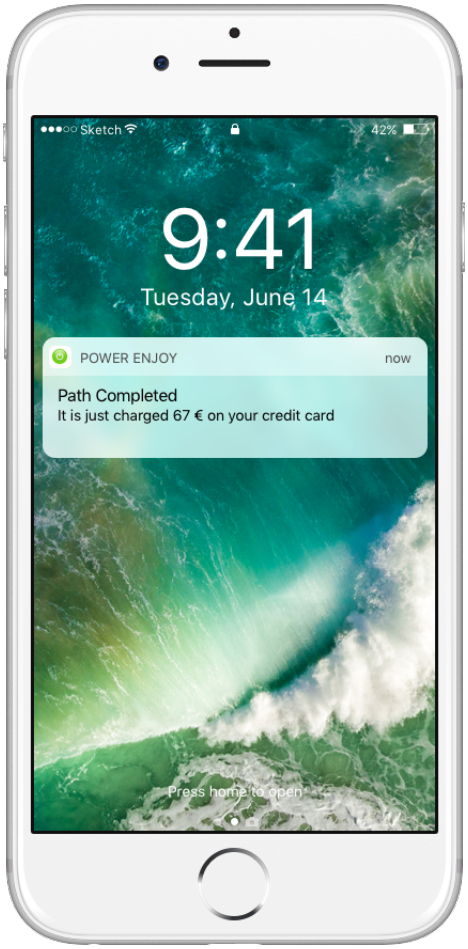
As long as the user reaches a safe area, parks into it and get out of the car, a push notification will inform him about the amount charged and eventually about the discounts applied.

COMPONENTS INVOLVED

Mobile app, Reservation Manager, DB, Payment handler, Email gateway, push gateway, User, Bank service, Sensors/Actuator manager, Update car status, Safe

car slot, Push gateway and Lock handler.

Step 6 of 6: Reach the destination and exit



DESCRIPTION

This is a second type of push notification.

COMPONENTS INVOLVED

Mobile app, Reservation Manager, DB, Payment handler, Email gateway, push gateway, User, Bank service, Sensors/Actuator manager, Update car status and Safe car slot and Lock handler.

Step 6 of 6: Reach the destination and exit

DESCRIPTION

This is a third type of push notification.

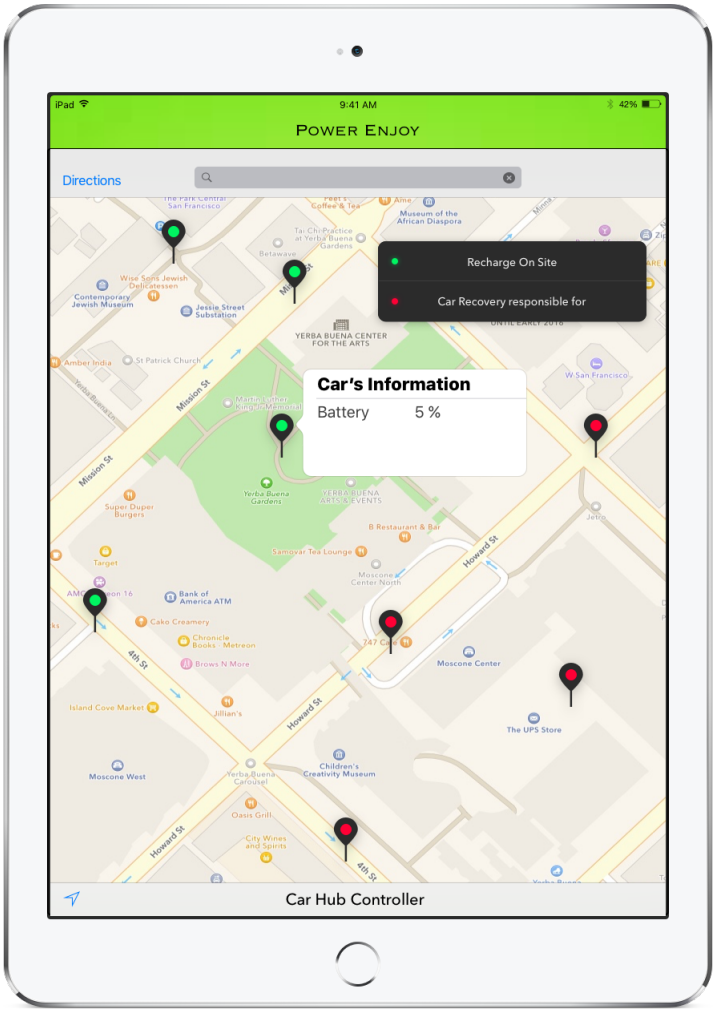
COMPONENTS INVOLVED

Mobile app, Reservation Manager, DB, Payment handler, Email gateway, push gateway, User, Bank service, Sensors/Actuator manager, Update car status and Safe car slot, Push gateway and Lock handler.

4.2 Car Hub Controller Interface

A possible supervisor will be able to perform the *Recharge On Site* and *Car Recovery* tasks in the following way.

4.2.1 A supervisor selects a car to recharge

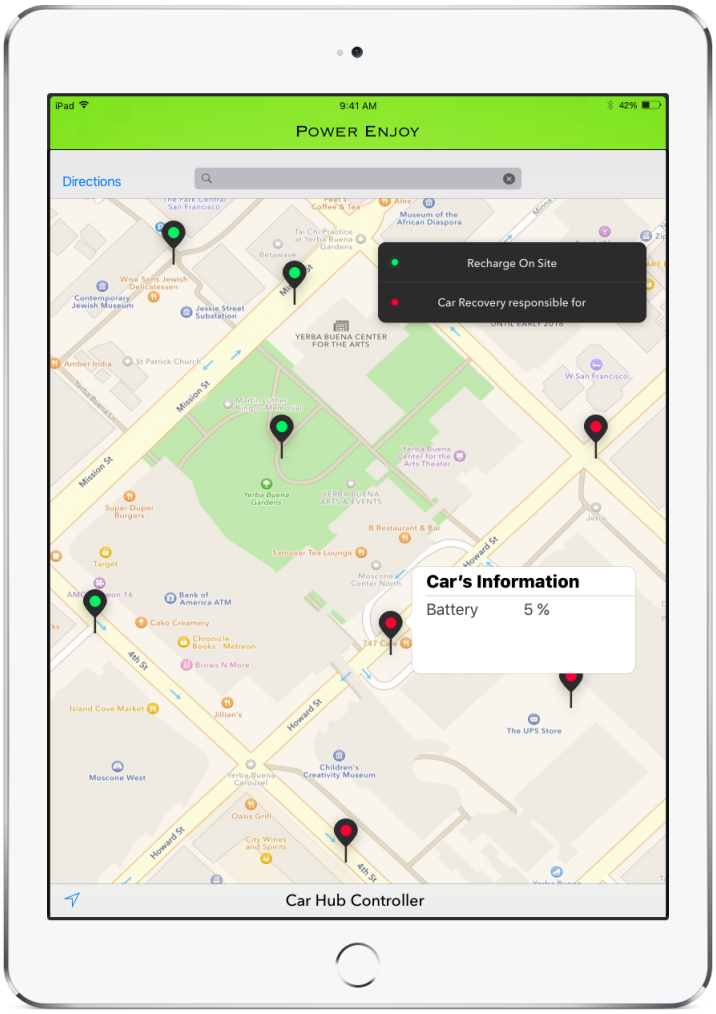


DESCRIPTION

COMPONENTS INVOLVED

Notification Displayer, Car and DB.

4.2.2 A supervisor selects a car to recovery



DESCRIPTION

COMPONENTS INVOLVED

Notification Displayer, Car and DB.

Chapter 5

Requirements Traceability

- [G1]: Ensure system's accessibility.
- [G2]: Supervisors must be able to check cars' status.
- [G3]: Supervisor should be able to dispatch *"recharge on site"* correctly.
- [G4]: Supervisor should be able to dispatch *"car recovery"* correctly.
- [G5]: Guarantee the correctness of each car's *"availability state"*.
- [G6]: Allow user to find available cars within a certain distance from a specified place.
- [G7]: Allow user to reserve a single car.
- [G8]: Discourage fake and too long reservation.
- [G9]: Allow the user who reserved the car to see information about his reservation.
- [G10]: Allow only the user who reserved the car (and his passenger) to access it.
- [G11]: Guarantee the correctness of the *"cost of the trip"*.
- [G12]: Guarantee the correctness of the *"virtuousness coefficient"*.

- [G13]: Guarantee the correctness and the payment of the final bill.
- [G14]: Support the users saving money.

GOAL	**Component**	**Entity**
G1	Mobile App, Authentication handler	User, Central server
G2	Notifications displayer, Car	Supervisor, Central Server
G3	Notification displayer	Supervisor
G4	Notification displayer	Supervisor
G5	Update car status, Car, Reservation, Safe park slot, Sensor/actuator manager	Central server, Physical car
G6	Mobile app, Authentication handler, Car finder, Google maps API	Central server, User, External
G7	Mobile app, Authentication handler, Reservation manage Reservation	Central server, User
G8	Reservation manager	Central server
G9	Mobile app, Authentication handler, Reservation manager, Reservation	Central server, User
G10	Mobile app, Authentication handler, Lock handler, Sensor/actuator manager	Central server, User, Car
G11	Reservation, Car	Central server
G12	Reservation, Car	Central server
G13	Reservation, Car Payment handler	Central server
G14	Info displayer, Google maps API	Car, External

Chapter 6

Effort Spent

Date	Marco (h)	Angelo (h)	Gabriele (h)
16/11/2016		0.5	
21/11/2016			1
22/11/2016			1
29/11/2016	6	4	4
30/11/2016	5		3
01/12/2016	0.5		6
02/12/2016	5		5
03/12/2016	1	1.5	
04/12/2016	5		
06/12/2016	3	3	3
08/12/2016	3	5	
09/12/2016		2	
10/12/2016		8	
11/12/2016	2	5	
TOTAL	30.5	29	23

Chapter 7

References

7.1 Tools Used

- **Sketch:** for the mokups' graphical rapresentation
- **MS Word:** as a shared text editor
- **Github:** to publicize our work
- **Texpad:** to design DD with LaTeX
- **Photoshop**
- **UmlLet**
- **Sublime Text:** to write algorithms

7.2 ChangeLog

Version 1.0 - First Release.

Version 1.1 - Fix mockups and images look & feel and add components involved in mockups.