Finito

Finita mia parte, manca UML o JAVA

Da rileggere ed eventualmente sistemare

Devo fare del lavoro

# **Introduction**

**1.1 Purpose**

This document extends the PowerEnjoy’s RASD document entering more deeply into the technical details.

This document is addressed to developers, aiming to explain the high level architecture, the main document’s components and how they interact with each other’s, the deployment cycle and the runtime behavior.

# **1.2 Scope**

PowerEnjoy is a service aimed to help people move around easier, without having to rely on their personal transport.

A secondary goal is to reduce cities’ pollution and noise.

The system allows users, after a registration where they insert their IDs and driving licenses, to rent a car (near their position (detected by using their smartphone’s GPS) or near a specific location, chosen on the map).

# **1.3 Definitions, acronyms, abbreviations**

# ”Cost of the trip” is the 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” is the factor by which to multiply the cost of the trip to get the amount of the bill. Its initial value is 1.

# ”Supervisor” is a company employee who work at the Car hub controller. ”Recharge on site” is a company procedure: a worker is sent to recharge a low car that was parked detached from the power grid.

# ”Car recovery” is a company procedure: a worker is sent to retrieve a car that has been forgotten outside a safe area and park in one of these.

# ”Guest” is a person who is not already registered to the system.

# ”User” is a registered customer.

# ”RASD” is the acronym of Requirement Analysis and Specification Document

# **Documents Structure**

1. INTRODUCTION
   1. Purpose
   2. Scope
   3. Definitions, Acronyms, Abbreviations
   4. Reference Documents
   5. Document Structure
2. ARCHITECTURAL DESIGN
   1. Overview
      1. General Structure
      2. High level components and their interaction
   2. Component view
      1. DB Component and Interface
      2. Safe park lot Component
      3. User Component and Interface
      4. Authentication handler component and interface
      5. Reservation component and interface
      6. Car Component and interface
   3. Deployment view
   4. Runtime view: You can use sequence diagrams to describe the way components interact to accomplish specific tasks typically related to your use cases
   5. Component interfaces
   6. Selected architectural styles and patterns: Please explain which styles/patterns you used, why, and how
   7. Other design decisions

3. ALGORITHM DESIGN: Focus on the definition of the most relevant algorithmic part

4. USER INTERFACE DESIGN: Provide an overview on how the user interface(s) of your system will look like; if you have included this part in the RASD, you can simply refer to what you have already done, possibly, providing here some extensions if applicable.

5. REQUIREMENTS TRACEABILITY: Explain how the requirements you have defined in the RASD map to the design elements that you have defined in this document.

6. EFFORT SPENT: In this section you will include information about the number of hours each group member has worked towards the fulfillment of this deadline.

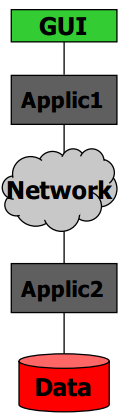
7. REFERENCES

# **2. Architectural Design**

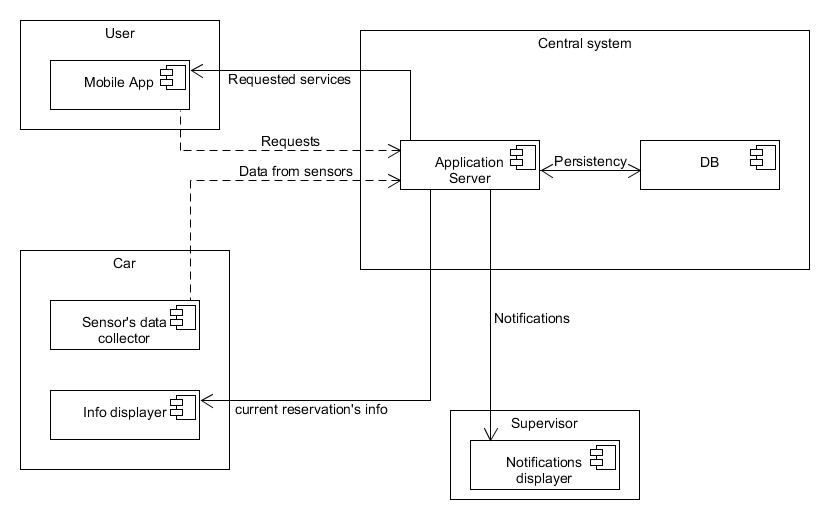
**2.1 Overview**

**2.1.1 General Structure**

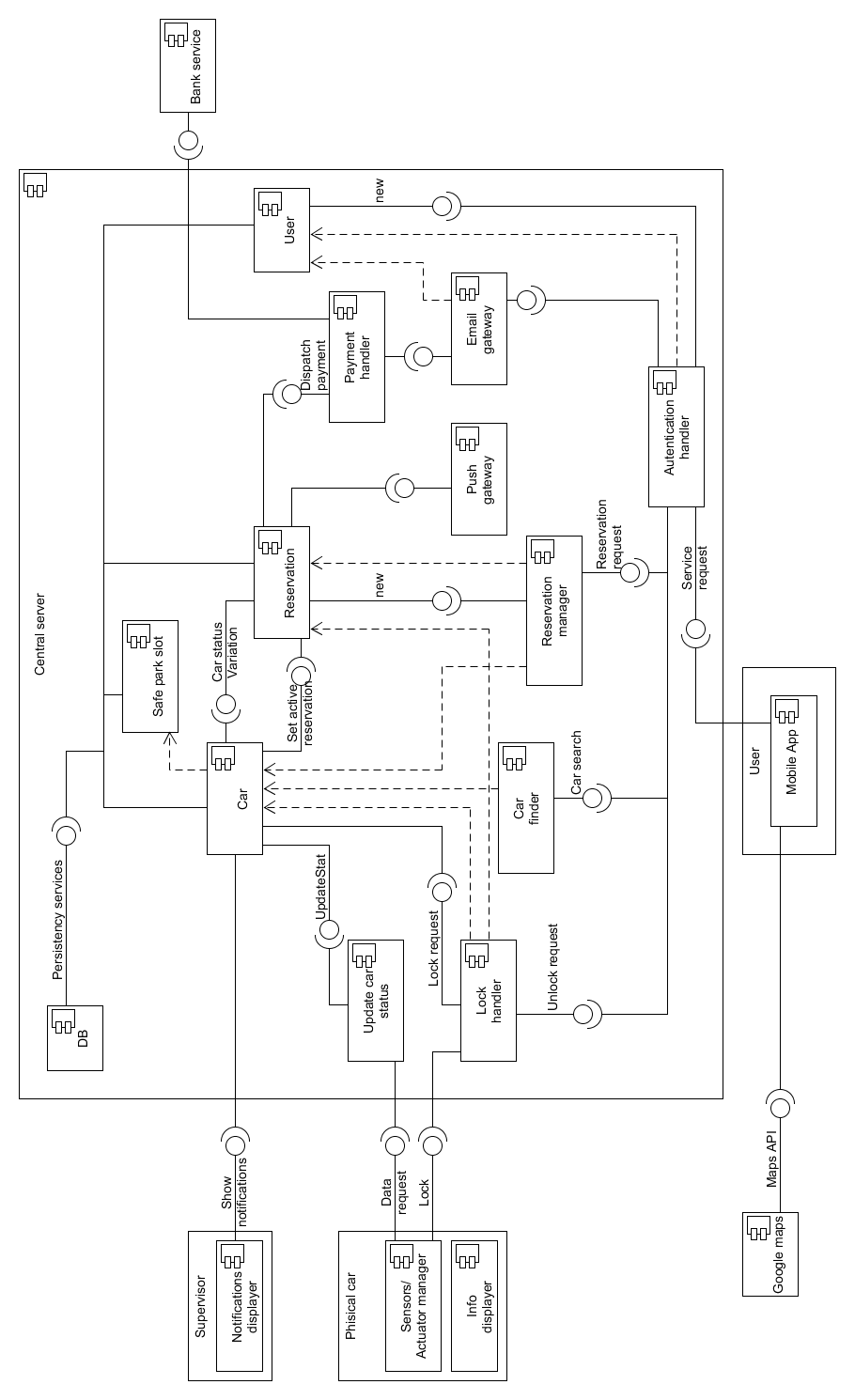
2 level C/S tier (distributed logic)

****

**2.1.2High Level Components and their interaction**

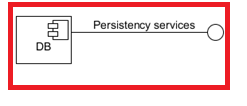


**2.2 Component View**

****

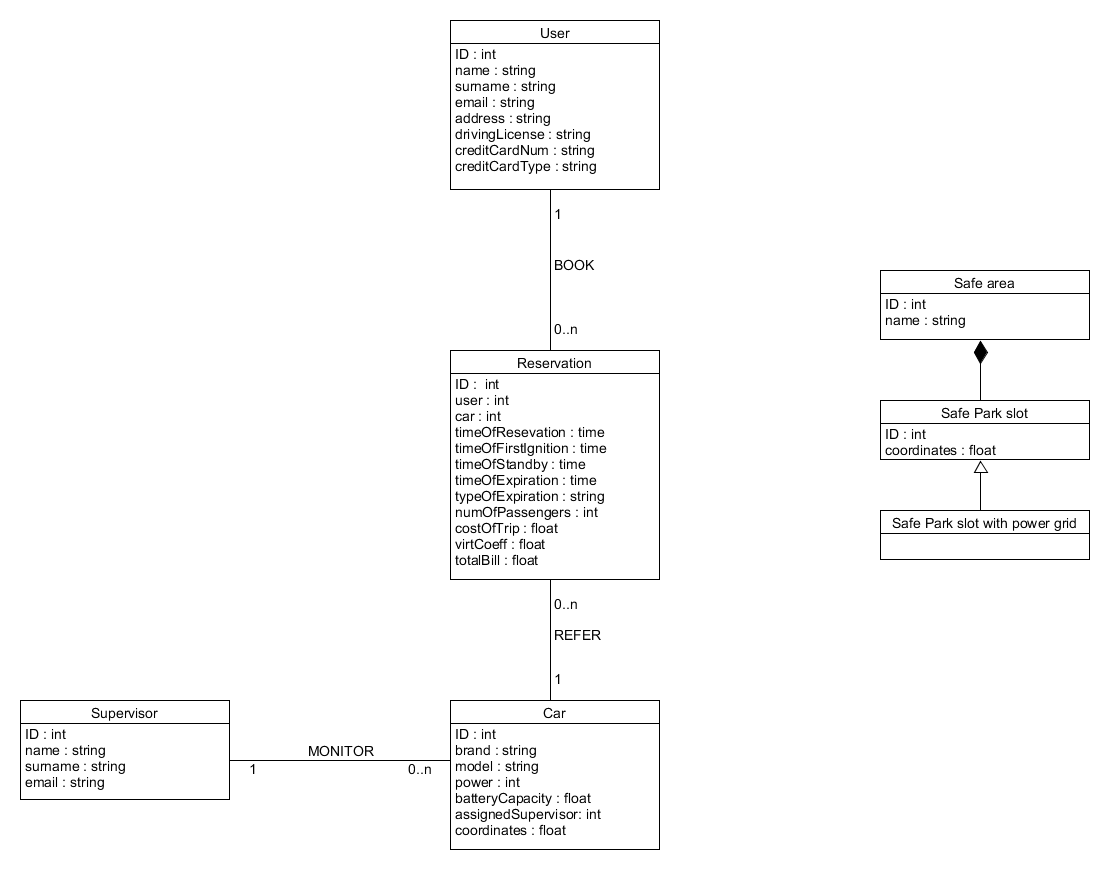
*Component diagram*

**2.2.1 DB Component and interface**

****

*Component*

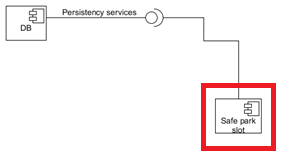
The component represents the Database used by the application server to store persistent data.



*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.

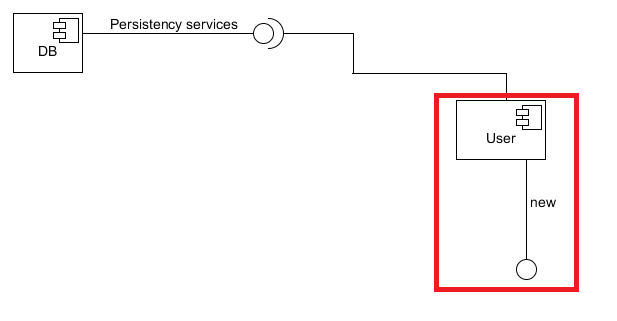
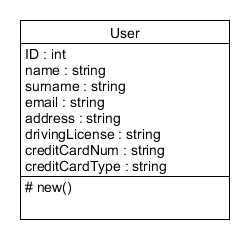
**2.2.2 Safe park lot Component**



*Component*

This component represents a simple entity bean, it’s only meant to represent the entities “Safe park slot” of the database

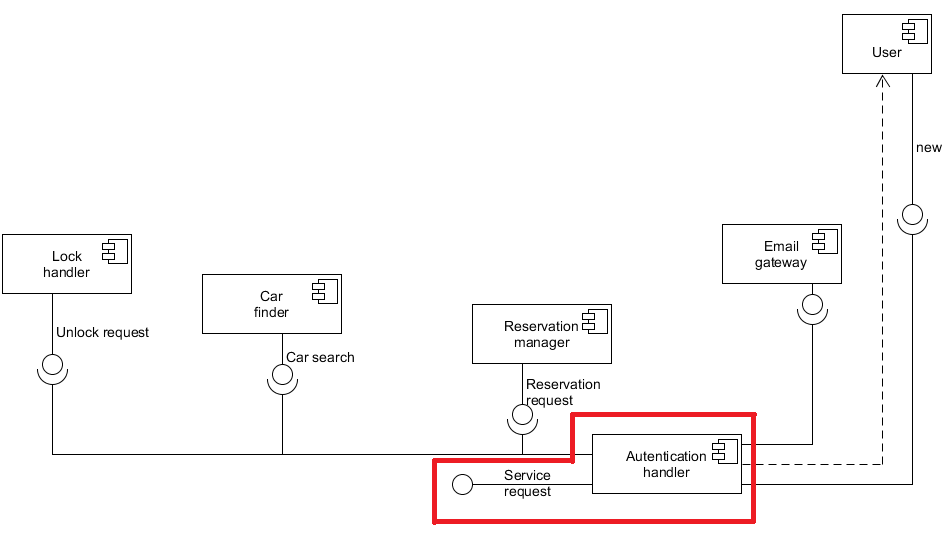
**2.2.3 User Component and interface**



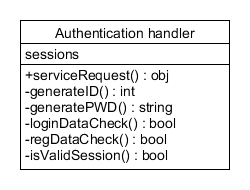
*Component class representation*

This component is an entity bean too; in addition, 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 Autentication handler component and interface**



*Component*

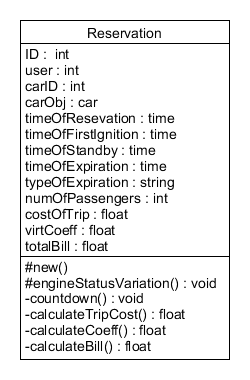


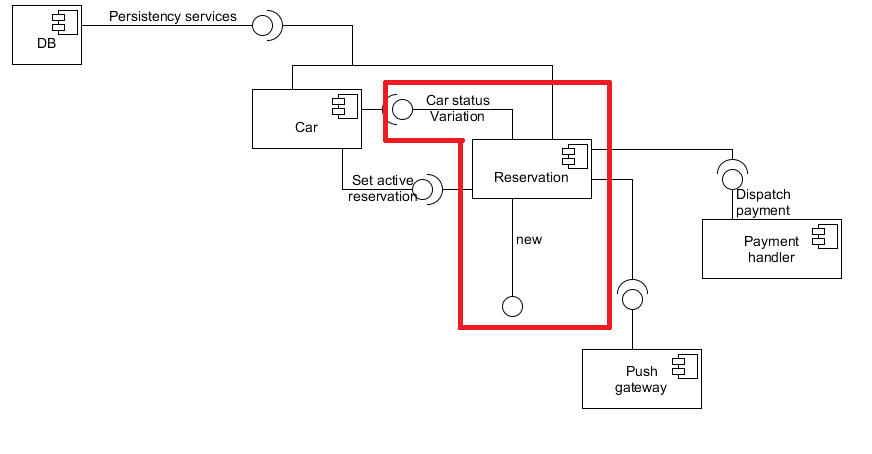
*class representation*

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**



****

*Component class representation*

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

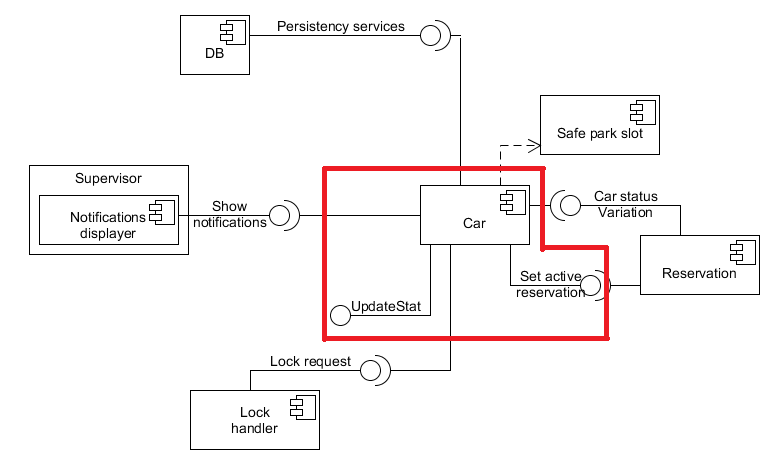
The “new” interface makes possible to create new “reservation” object. The method associated with this interface is responsible to initialize part of the object’s attribute, to launch a 60 minutes’ countdown, and to use the "set active reservation" interface of the car object to which the reservation is associated.

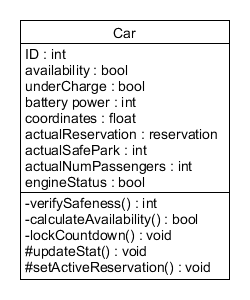
The "Car status variation" is used to notify the reservation object about the changing of specific attributes on 'state of the Associated car; this is useful because in this way the reservation can detect when the first engine’s ignition occurs, so the system can register the number of passenger, or 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 “timeOfEpiration” 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**



*Component 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

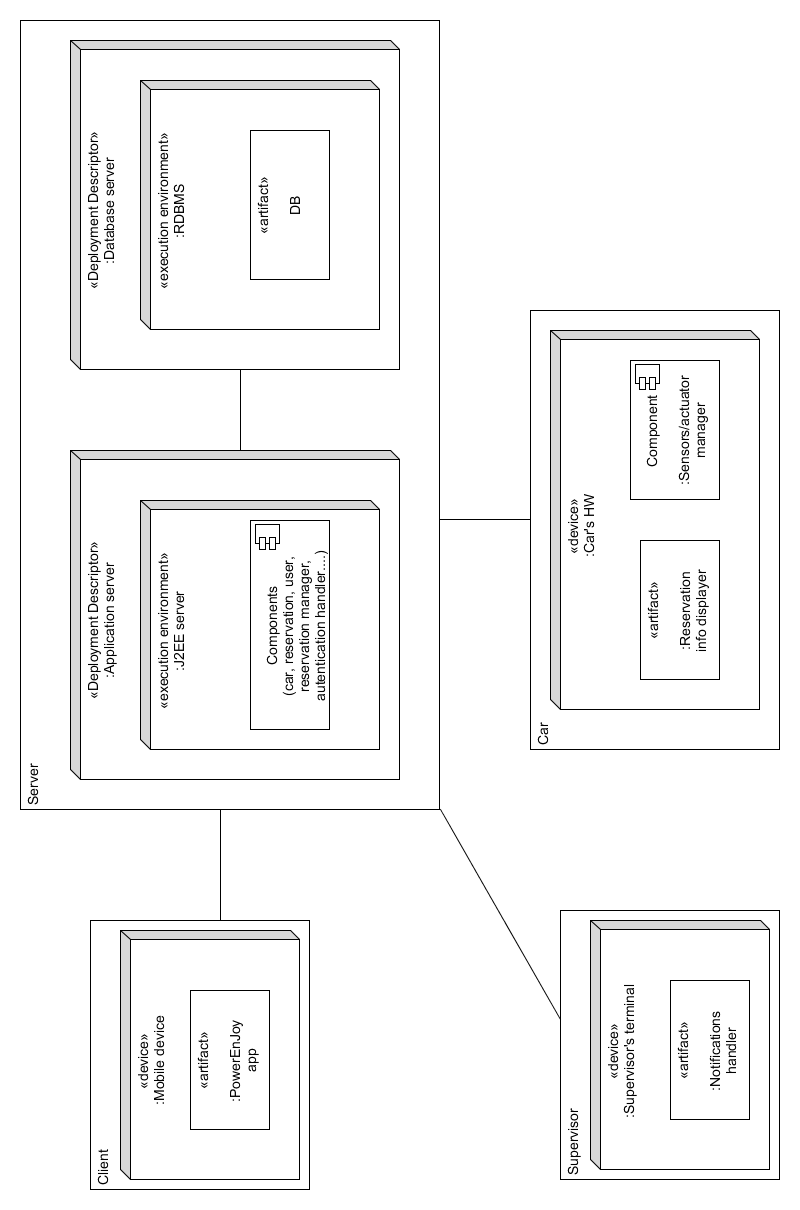
Engine switched on => car status variation

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

Reservation Expired && car is not in safe area => show notification

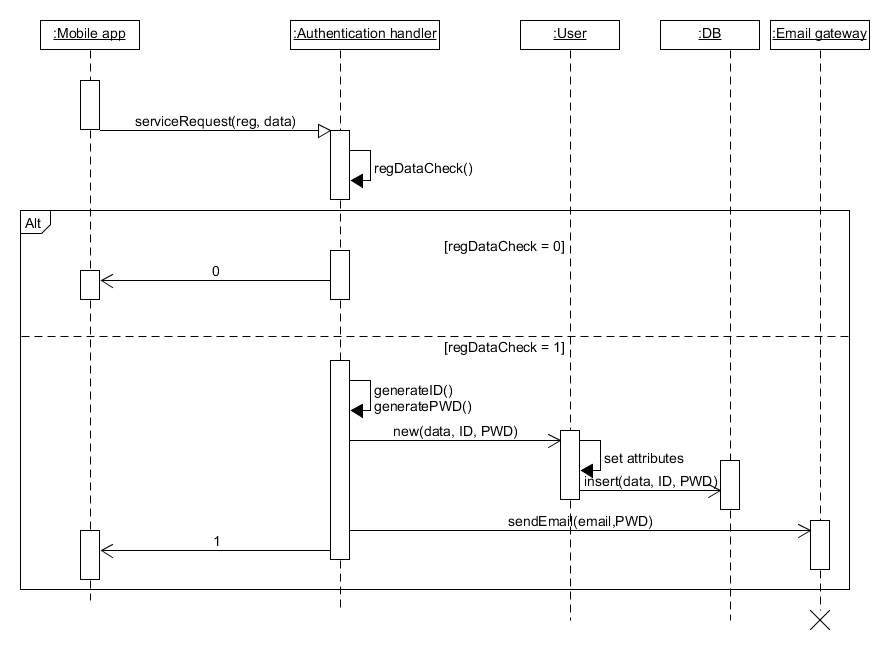
The “lockCountdown” method perform a 60 seconds countdown at the end of which it will be perform a lock request.

**2.3 Deployment View**

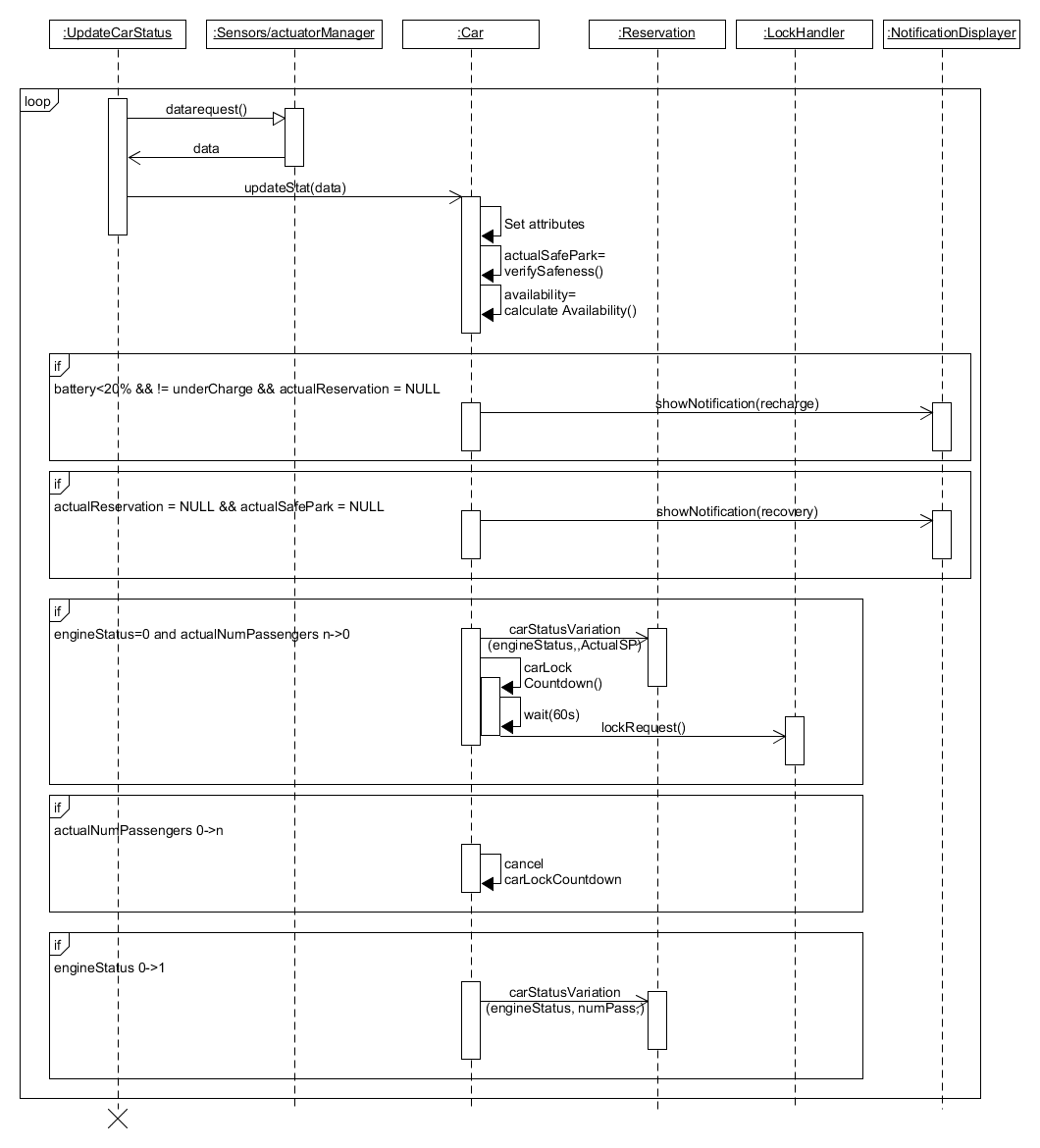
*Deployment view*

**2.4 Runtime View (Sequence)**

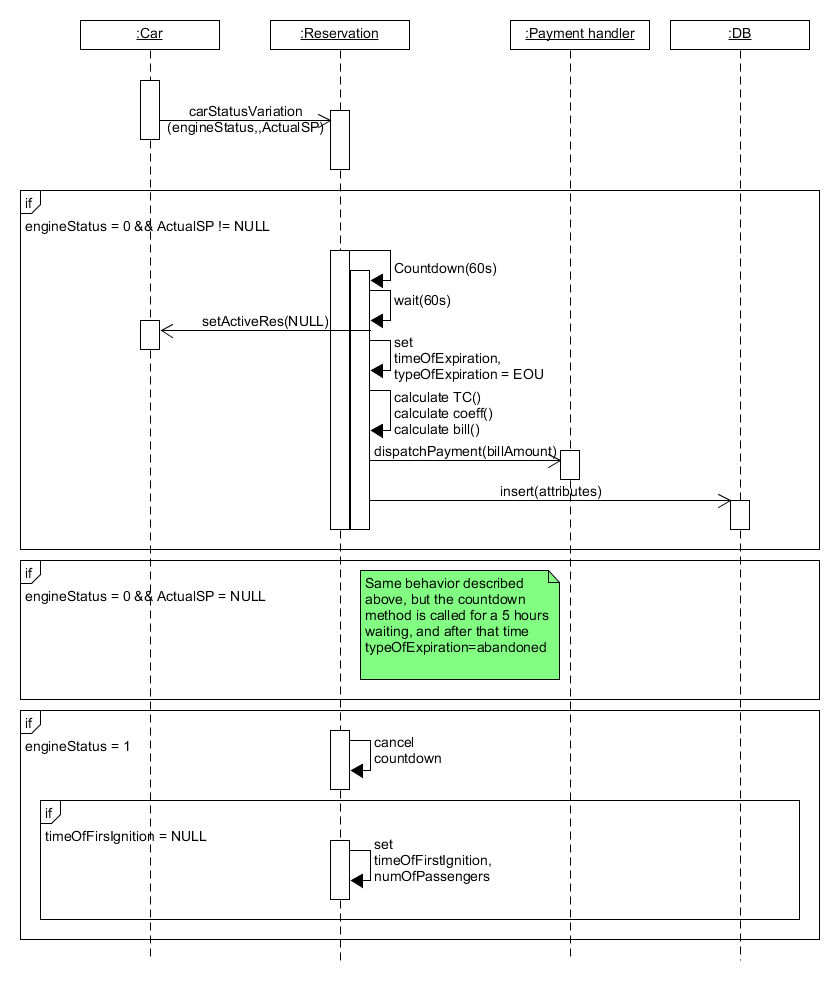
**2.4.1 Registration Sequence**

****

**2.4.2 Update car status sequence**

****

2.4.3 Car status variation Sequence



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

Inserire I sequence con una breve descrizione

**2.5 Component interfaces**

**2.6 Selected architectural styles and patterns**

**2.7 other design decisions**

# **3. Algorithm Design**

Parte di JAVA

# **4. User Interface Design**

?? spiegare con il mockup?

# **5. Requirements Traceability**

* [Goal 1]: Ensure system’s accessibility
* [Goal 2]: Supervisors must be able to check cars’ status
* [Goal 3]: Supervisor should be able to dispatch ”recharge on site” correctly
* [Goal 4]: Supervisor should be able to dispatch ”car recovery” correctly
* [Goal 5]: Guarantee the correctness of each car’s ”availability state”
* [Goal 6]: Allow user to find available cars within a certain distance from a specified place
* [Goal 7]: Allow user to reserve a single car
* [Goal 8]: Discourage fake and too long reservation
* [Goal 9]: Allow the user who reserved the car to see information about his reservation
* [Goal 10]: Allow only the user who reserved the car (and his passenger) to access it
* [Goal 11]: Guarantee the correctness of the ”cost of the trip

# **6. Effort Spent**

Marco:

29/11/16: 6h (4 coop, 2 alone)

30/11/16: 5h (2 coop, 3 alone)

1/12/16: 0.5h

2/12/11: nya

Gabriele:

21/11/16: 1h

22/11/16: 1h

29/11/16: 4h

30/11/16: 3h

01/12/16: 6h

02/12/16: 1h

# **7. References**