

Version: 1.0.0

Release date: 25-11-2016

Riccardo Cattaneo 873647

Fabio Chiusano 874294

PowerEnJoy, Design Document

Table of Contents

1 Introduction 3

1.1 Purpose 3

1.2 Scope 3

1.3 Definitions, Acronyms, Abbreviations 4

1.4 Reference Documents 5

1.5 Document Structure 5

2 Architectural Design 6

2.1 Overview: High level components and their interaction 6

2.2 Component view 6

2.3 Deployment view 6

2.4 Runtime view 6

2.5 Component interfaces 6

2.6 Selected architectural styles and patterns 6

2.7 Other design decisions 6

3 Algorithm Design 7

4 User Interface Design 8

5 Requirements Traceability 9

6 Effort spent 10

7 References 11

# Introduction

## Purpose

This document describes the hardware and software architecture of the PowerEnjoy System. Therefore, it outlines hardware tiers and all parts of the software and how they will work and cooperate together.

In particular, this document contains information about:

* Architecture Design with related pattern used;
* Main components and software interface design;
* Runtime behaviour of the system
* User Interface Design

## Scope

PowerEnjoy is a car-sharing service based on a mobile web application.

It allows the user to see, thanks to the help of an external search-on-a-map handler, where the electric cars are, only if they are close to an address provided by either the user or his/her GPS Location.

Hence, it allows users to reserve an electric car and to get on board when he/she is close to it.

The car software takes into account the minutes of usage of the car, the number of passengers, the battery level and the location of release.

The system then calculates the charges the user for the ride.

The main purpose of the system is to create a new and smart car-sharing service, that incentivize virtuous and green behaviours.

## Definitions, Acronyms, Abbreviations

* DD: Design document.
* RASD: Requirements Analysis and Specification Document.
* JSE: Java Serial Edition.
* JEE: Java Enterprise Edition.

## Reference Documents

## Document Structure

This document specifies the architecture of PowerEnJoy spreading from the general into the specific. Also it describes the architectural decisions and tradeoffs and justifies them. The design was guided by a top-down process approach and the document structure reflects this tactic.

The document is organized as follows:

1. Introduction, provides a synopsis of the architectural descriptions.
2. Architectural design, provides a general description of PowerEnJoy including its  functionality and matters related to the overall system and its design.
3. Algorithmic design.
4. User Interface design.
5. Requirements traceability.
6. Effort spent.

# Architectural Design

## Overview

We propose to make a web app that will give users a comfortable way to use our service. The reasons according to which we have chosen a web app instead of the web site are explained in the “Proposed system” chapter of the RASD document.

The architecture of PowerEnjoy Service has three tier:

* The presentation will be provided to the user thanks to the cooperation between the client and the Web Tier of the Server;
* The Logic will be provided by the Main Server;
* The Persistence functionalities will be provided by a Database Server, which communicate directly to the Main Server.



Client Tier

Presentation Tier

Mobile Web-App

Web Tier

Presentation Tier



Business Tier

Business Logic Tier

Main Server



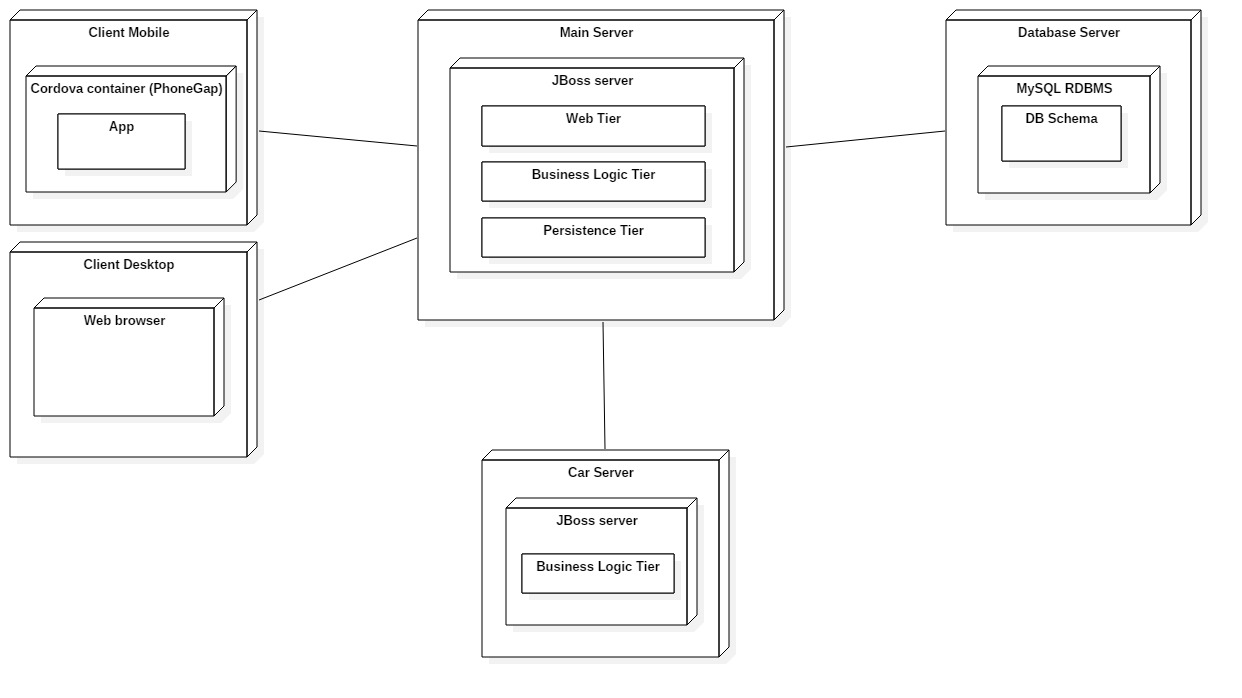
EIS Tier

Database Tier

Database Server

## High Level Components and their interactions

### Overview



* Client Tier, represented by the web app, that will be available to all the major mobile operating systems and will be developed in a way to communicate with the PowerEnjoy servers through RESTful API and HTTP requests, consequently the front-end will lie on the server;
* On the server side, the Main Server contains two software tiers:
  + Web Tier, that manages the communication with the web app.

It receives requests from the app and provides it and updated front-end, thanks to JavaServerPages (JSP);

* + Business Logic Tier, that elaborates requests from client thanks to the embedded logic and manages interactions with Database, thanks to the Java Percistence API (JPA);
* Database Tier, that contains and manages persistent data in an efficient way.

Of course, the electric cars must be able to communicate with the server, so they must be provided with an Internet connection and an on-board computer that must be able to run Java software. Cars will have small logic, mainly dedicated to communicate sensors data with Logic Tier of the Main Server.

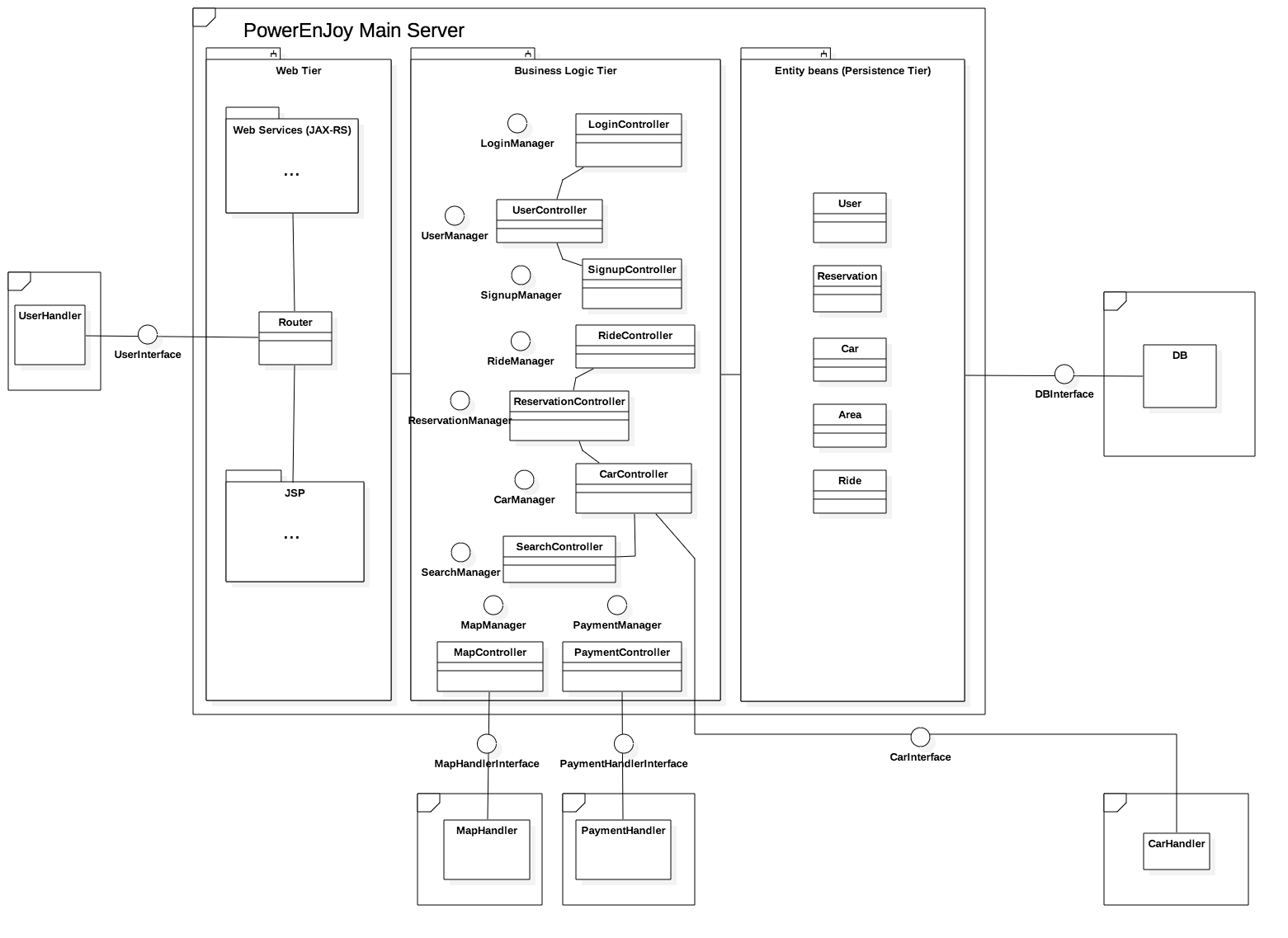
The Main Server will also take advantage of external handler, such as Payment Handler and Search-on-a-map Handler, in order to manage successfully functionalities provided to users.

### Technologies used

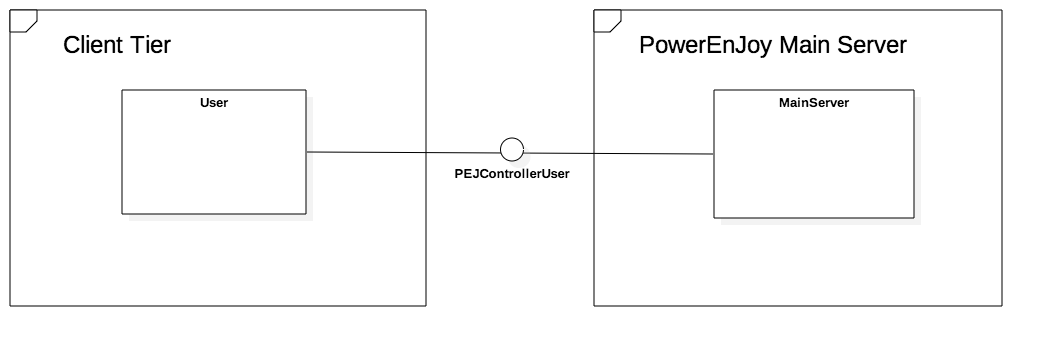
* The web app will be implemented using PhoneGap framework:
  + Interactions with the Server will be done via HTTP calls, using RESTful services offered;
* The Main Server will be implemented with Java Enterprise Edition (JEE):
  + Communication with the web app will be possible thanks to Java API for RESTful Web Services (JAX-RS);
  + The front-end functionalities, that clients need, will be provided with Java Server Pages (JSP);
  + Business Logic will be implemented using Session Java Beans, and communication with WebTier will be managed with the EJB Container;
  + Data management and interaction with Database Server will be implemented thanks to Java Persistence API (JPA), and will be composed of Entity Java Beans;
* For the Database Server we will use MySQL.

## Component view

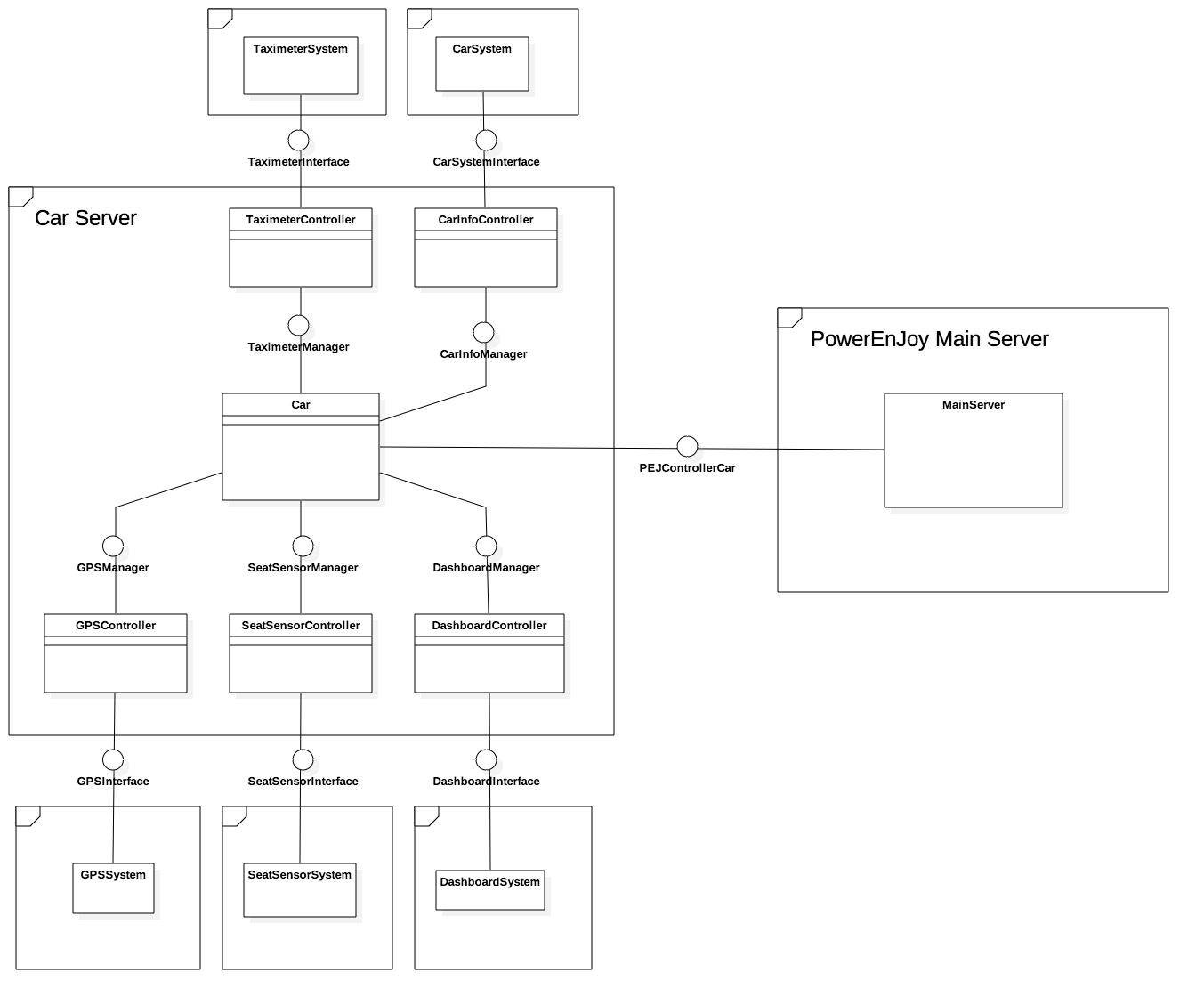
### Main server component view



### User component view



### Car component view



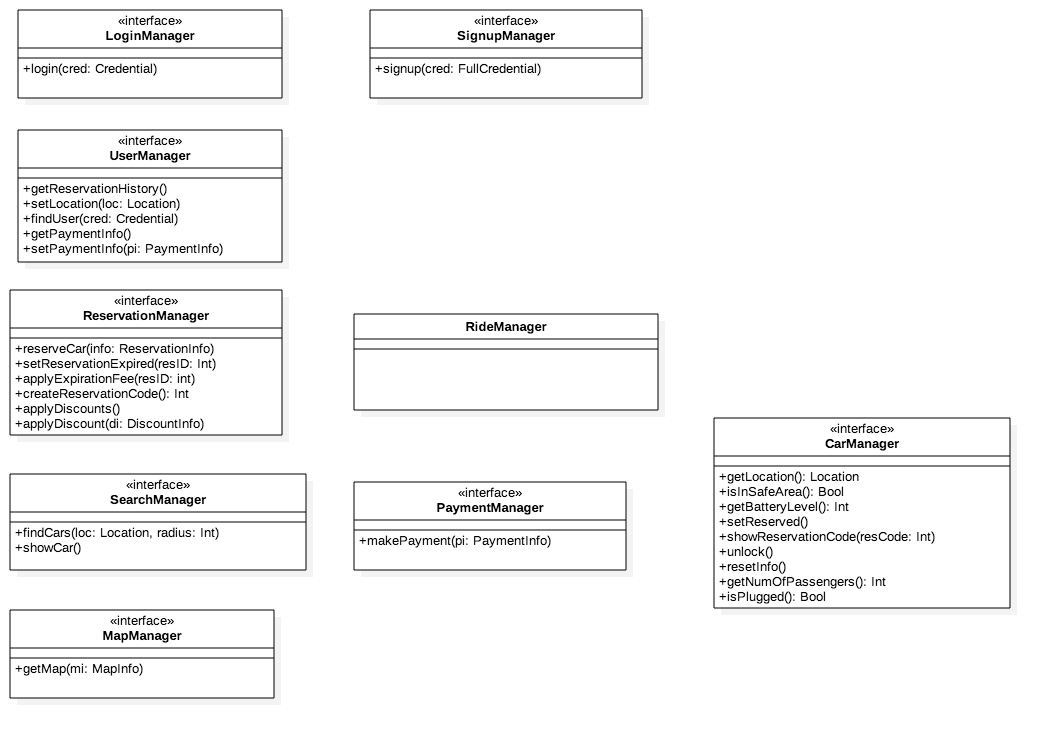
## Deployment view

## Runtime view

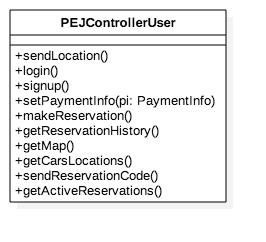
Looooots of sequence diagrams.

## Component interfaces

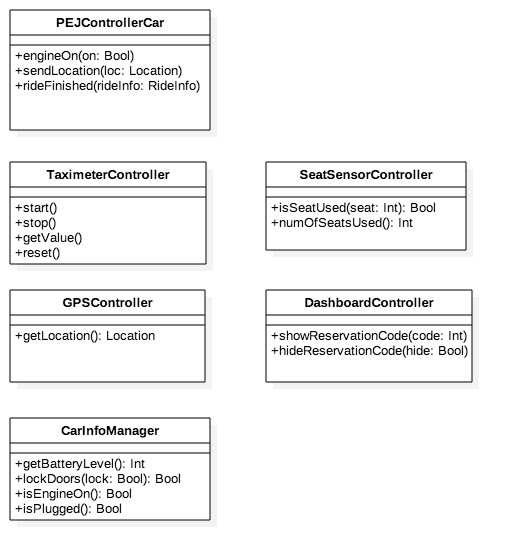
### Main server component interfaces



### User component interfaces



### Car component interfaces



## Selected architectural styles and patterns

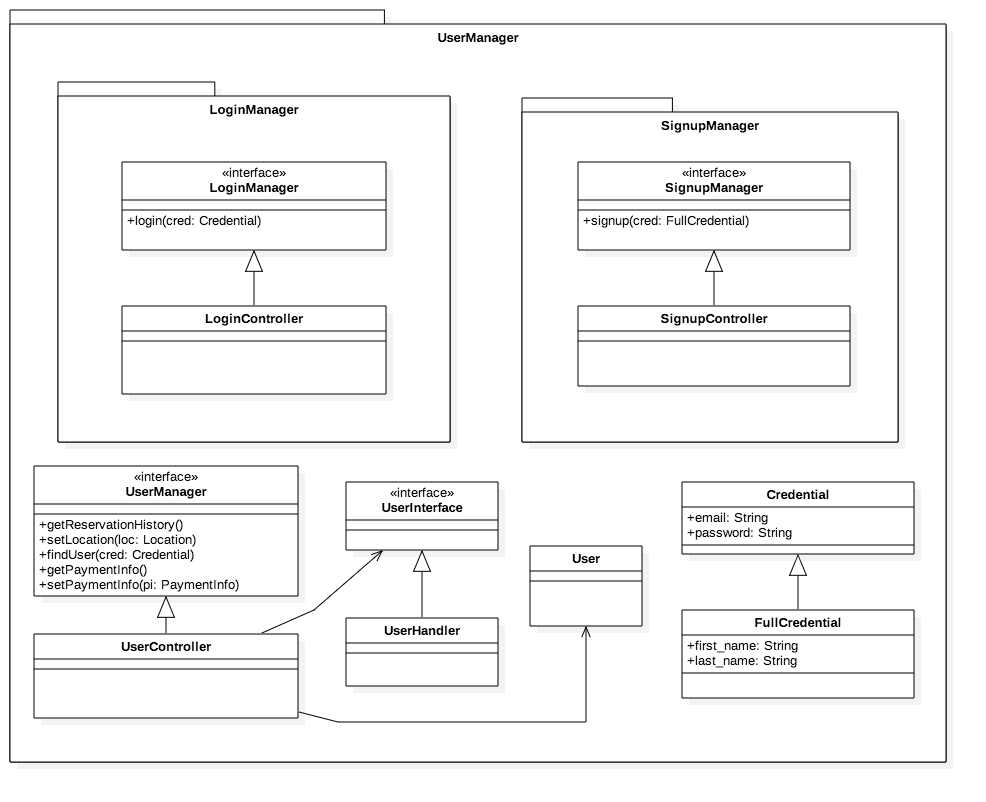
Three tiers, event-based, MVC…

## Other design decisions

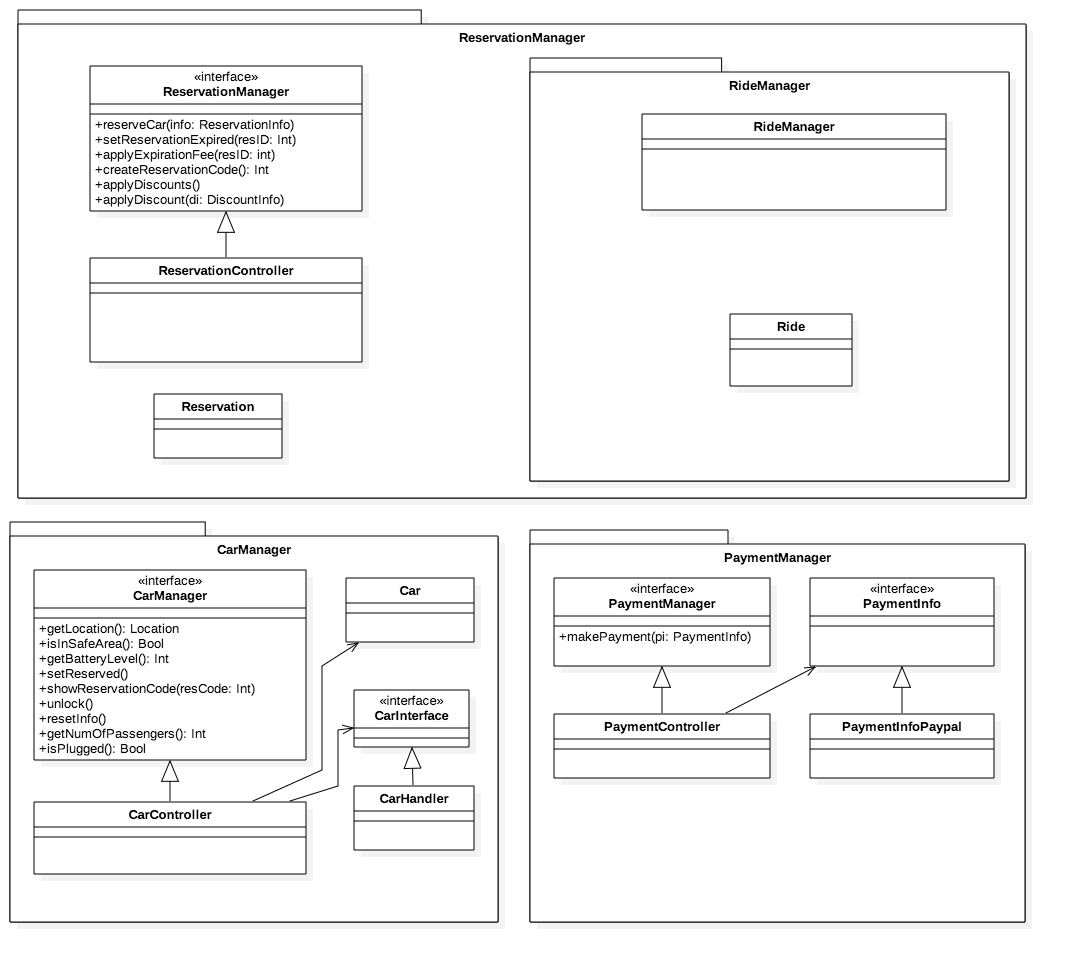
Maybe data model (class diagrams, Object Relationship diagrams) and data storage (ER diagrams, SQL stamements for tables creation).

## Extra: class diagrams for main server components

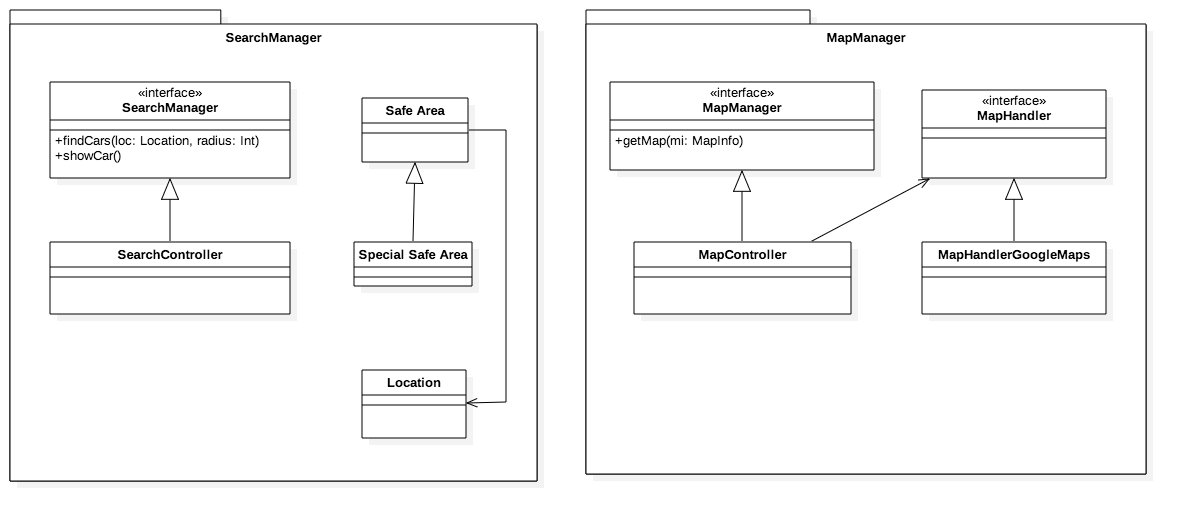
### User related



### Reservation related



### Search related



# Algorithm Design

- GPS usage (not sure)

- Search for near cars:

1. App asks server for near cars and sends it its location;
2. Server retrieves cars locations quickly thanks to some spatial data structures (https://en.wikipedia.org/wiki/Spatial\_database);
3. Server sends the locations to the app;
4. App is happy.

- Search for near parkings: similar to search for near cars.

# User Interface Design

# Requirements Traceability

The following tables show how each requirement specified in the RASD document is satisfied by the architecture proposed. The *components* and *description* columns give and idea of all the functions involved.

* [G1] *Users can see and select an available car close to him, or close to a specified address, and reserve it for up to one hour before they pick it up.*

|  |  |  |
| --- | --- | --- |
| REQUIREMENT | COMPONENTS | DESCRIPTION |
| R1: The system has to detect if a car is parked in a Safe Area | Car server:   * GPSController   Main server:   * CarController | Car server:   * GPSController.getLocation   Main server:   * CarController.getLocation, * CarController.isInSafeArea |
| R2: The system has to detect the battery level of each car | Car server:   * CarInfoController   Main server:   * CarController | Car server:   * CarInfoController.getBatteryLevel   Main server:   * CarController.getBatteryLevel |
| R3: The system has to detect car position and display it on the map | User browser/app:   * PEJControllerUser   Car server:   * GPSController   Main server:   * MapController, * CarController, * SearchController | User browser/app:   * PEJControllerUser.getMap * PEJController.getCarsLocations   Car server:   * GPSController.getLocation   Main server:   * MapController.getMap, * CarController.getLocation, * SearchController.showCar |
| R4: The system has to be able to identify the location of a user or through his/her GPS, if he/she gives the consent, or through some input | User browser/app:   * PEJControllerUser | User browser/app:   * PEJControllerUser.sendLocation |
| R5: The system has to provide a list of available cars close to a given address | Main server:   * SearchController | Main server:   * SearchController.findCars |
| R6: The system has to give the possibility to reserve a car at most by one user at a time | User browser/app:   * PEJControllerUser   Main server:   * ReservationController, * CarController   Car server:   * DashboardController | User browser/app:   * PEJControllerUser.makeReservation   Main server:   * ReservationController.reserveCar, * ReservationController.createReservationCode, * CarController.setReserved   Car server:   * DashboardController.showReservationCode |
| R7: The system has to mark the reservation as expired for a car after one hour if the user has not picked it up | Main server:   * ReservationController (+ JEE Timer Service) | Main server:   * ReservationController.setReservationExpired |
| R8: The system has to apply a fee of 1€ if the reservation has expired | Main server:   * ReservationController | Main server:   * ReservationController.applyExpirationFee |

* [G2] *Users can get in a car only if they are near it and they reserved it.*

|  |  |  |
| --- | --- | --- |
| REQUIREMENT | COMPONENTS | DESCRIPTION |
| R1: Cars, if reserved, should show a reservation code on their dashboard | Main server:   * CarController   Car server:   * DashboardController | Main server:   * CarController.showReservationCode   Car server:   * DashboardController. showReservationCode |
| R2: The system has to unlock the car if the user that reserved it sends the reservation code to the system | User browser/app:   * PEJControllerUser   Main server:   * CarController   Car server:   * CarInfoController | User browser/app:   * PEJControllerUser.sendReservationCode   Main server:   * CarController.unlock   Car server:   * CarInfoController.lockDoors |

* [G3] *Users should pay proportionally to minutes they have used the car, and they should see in real time the amount of the bill.*

|  |  |  |
| --- | --- | --- |
| REQUIREMENT | COMPONENTS | DESCRIPTION |
| R1: The system has to reset trip information when a user get on the car | Main server:   * CarController   Car server:   * TaximeterController * DashboardController | Main server:   * CarController.resetInfo   Car server:   * TaximeterController.reset * DashboardController.hideReservationCode |
| R2: The system has to be able to understand when the car engine ignites | Car server:   * PEJControllerCar | Car server:   * PEJControllerCar.engineOn |
| R3: The system has to start charging the user when the car engine ignites | Car server:   * TaximeterController | Car server:   * TaximeterController.start |
| R4: The system has to display the current charge | TaximeterSystem | The taximeter system is configured to always show its current value when active |
| R5: The system has to identify when a car is parked in a safe area | Main server:   * CarController   Car server:   * GPSController | Main server:   * CarController.getLocation * CarController.isInSafeArea   Car server:   * GPSController.getLocation |
| R6: The system has to identify when there is no one sit in the driver’s seat | Car server:   * SeatSensorController | Car server:   * SeatSensorController.isSeatUsed |
| R7: The system has to stop charging the user when the car is parked in a safe area and there is no one sat in the driver’s seat | Car server:   * SeatSensorController, * GPSController, * CarInfoController, * TaximeterController * PEJControllerCar | Car server:   * SeatSensorController.isSeatUsed, * GPSController.getLocation, * CarInfoController.isEngineOn, * TaximeterController.stop * TaximeterController.getVaule, * PEJControllerCar.rideFinished |

* [G4] *Users can register to the system and have their personal area.*

|  |  |  |
| --- | --- | --- |
| REQUIREMENT | COMPONENTS | DESCRIPTION |
| R1: The system has to provide log-in functionalities to the users:   * The user should be able to check his/her active reservations | Main server:   * LoginController, * UserController   User browser/app:   * PEJControllerUser | Main server:   * LoginController.login, * UserController.findUser   User browser/app:   * PEJControllerUser.login * PEJControllerUser.getActiveReservations |
| R2: The system has to provide sing-up form to users:   * The system has to check that there are not two users with the same username * The system has to store the password and personal information of every user * The system has to provide the possibility to enter a payment method * The system has to check if the payment method provided by the user is valid and usable | Main server:   * SignupController, * UserController, * PaymentController   User browser/app:   * PEJControllerUser | Main server:   * SignupController.signup, * UserController.findUser, * PaymentController.setPaymentInfo   User browser/app:   * PEJControllerUser.signup |
| R3: The system has to provide the possibility to change personal information or payment methods even after the registration | Main server:   * UserController, * PaymentController   User browser/app:   * PEJControllerUser | Main server:   * UserController.findUser, * PaymentController.setPaymentInfo   User browser/app:   * PEJControllerUser.setPaymentInfo |
| R4: The system has to provide the possibility to each user to see the personal “Reservation History” | User browserapp:   * PEJControllerUser   Main server:   * UserController | User browser/app:   * PEJControllerUser.getReservationHistory   Main server:   * UserController.getReservationHistory |

* [G5] *Virtuous behaviours by users should be incentivized.*

|  |  |  |
| --- | --- | --- |
| REQUIREMENT | COMPONENTS | DESCRIPTION |
| R1: The system has to apply a discount of 10% on the final bill if there were at least three passengers on the last ride:   * The system has to identify and store how many passengers there were on the car in the last ride | Main server:   * ReservationController, * CarController   Car server:   * SeatSensorController | Main server:   * ReservationController.applyDiscounts * ReservationController.applyDiscount, * CarController.getNumOfPassengers   Car server:   * SeatSensorController.numOfSeatsUsed |
| R2: The system has to apply a discount of 20% on the final bill if the car is left with at least 50% of battery level   * The system has to be able to identify the battery level of the car | Main server:   * ReservationController, * CarController   Car server:   * CarInfoController | Main server:   * ReservationController.applyDiscounts * ReservationController.applyDiscount, * CarController.getBatteryLevel   Car server:   * CarInfoController.getBatteryLevel |
| R3: The system has to apply a discount of 30% on the final bill if the car is left plugged-in in a Special Safe Area:   * The system has to identify if the car is plugged-in | Main server:   * ReservationController, * CarController   Car server:   * CarInfoController | Main server:   * ReservationController.applyDiscounts * ReservationController.applyDiscount, * CarController.getLocation * CarController.isPlugged   Car server:   * CarInfoController.isPlugged |
| R4: The system has to apply an extra-charge of 30% on the final bill if the car is left at least 3Km from the nearest Special Safe Area and the battery level is less than 30%:   * The system has to be able to calculate the distance between the actual position of the car and the nearest Special Safe Area | Main server:   * ReservationController, * CarController   Car server:   * GPSController * CarInfoController | Main server:   * ReservationController.applyDiscounts * ReservationController.applyDiscount, * CarController.getLocation   Car server:   * GPSController.getLocation * CarInfoController.getBatteryLevel |

# Effort spent

# References