

Luca Scannapieco - 877145

Andrea Pasquali - 808733

Emanuele Torelli - 876210

11/12/2016

PowerEnJoy

Design Document

Table of contents

[1. Introduction 3](#_Toc468551037)

[1.1. Purpose 3](#_Toc468551038)

[1.2. Scope 3](#_Toc468551039)

[1.3. Definitions, acronyms, abbreviations 3](#_Toc468551040)

[1.4. Reference documents 3](#_Toc468551041)

[1.5. Document structure 3](#_Toc468551042)

[2. Architectural design 3](#_Toc468551043)

[2.1. Overview 3](#_Toc468551044)

[2.2. Component view 3](#_Toc468551045)

[2.3. Deployment view 5](#_Toc468551046)

[2.4. Runtime view 6](#_Toc468551047)

[2.5. Component interfaces 8](#_Toc468551048)

[2.6. Selected architectural styles and patterns 9](#_Toc468551049)

[2.7. Other design decisions 9](#_Toc468551050)

[3. Algorithm design 9](#_Toc468551051)

[4. User interface design 9](#_Toc468551052)

[5. Requirements traceability 9](#_Toc468551053)

[6. Effort spent 9](#_Toc468551054)

[7. References 9](#_Toc468551055)

# 1. Introduction

## 1.1. Purpose

## 1.2. Scope

## 1.3. Definitions, acronyms, abbreviations

## 1.4. Reference documents

## 1.5. Document structure

# 2. Architectural design

## 2.1. Overview

The PowerEnJoy service is implemented as a common client-server application, in which the offered services are essentially three:

* User interface
* Application logic
* Database

Each of these logic services is placed in the corresponding physic layer, the result is that we adopted a three-tier architecture.

We provide a mockup to better understand the structure of the PowerEnJoy service.

MOCKUP

The user interface has two different implementations, one is constituted by a web app that can be executed on a modern browser, and the other one is the PowerEnJoy mobile application. Furthermore, during a ride, there is another component with which the user interacts: the screen of the car. On the other hand, the assistance coordinator has an interface built ad hoc to performs his work, in fact the tasks that the coordinator must do are forbidden to the common users.

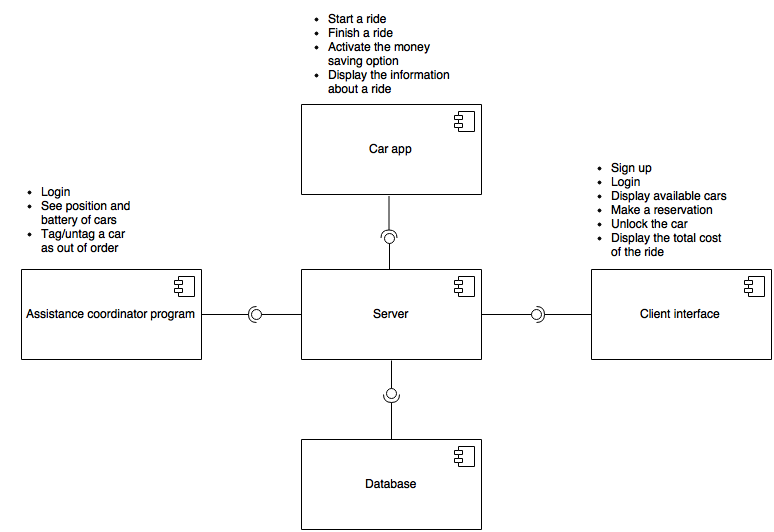
The core of our architecture is the main central server, where is provided the logic necessary to the system to run properly. In addition to the central server we need an external web server to run the PowerEnJoy web app properly.

Separating the layer for the application logic and the database we can ensure a high modularity of the system, and for example the company can decide to move one of the layers (or both) to a cloud service, for example to amazon AWS where it would have dedicated cloud servers with load balance for database and other for application logic on demand.

## 2.2. Component view

In the following diagram we will show the main components of the PowerEnJoy system, presenting them firstly at the highest level as possible.

Note that for each of the three high level components of the Tier 1 (the client app, the car app and the coordinator app) we describe the functionalities that the system must ensure; we derived those functionalities from the use cases diagram presented in the RASD.

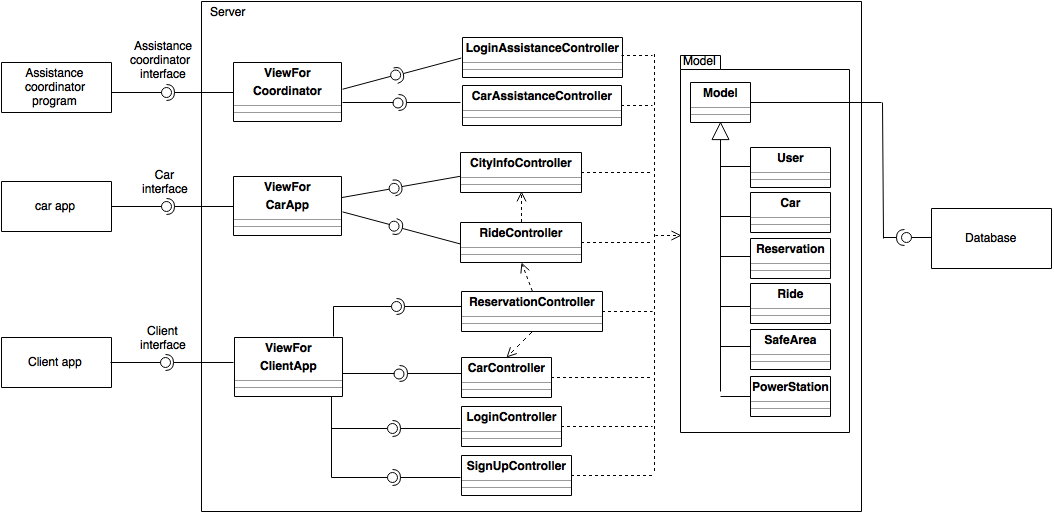


Now we can proceed with a further description of the central server, which is the core of the whole system, by analysing at a software level how it works.

The various functionalities are exploited by multiple controllers, each of them provides some methods.

Each controller provides an interface to a view, in fact there are three views one for high level component of the Tier 1. The views are important to guarantee the correct forwarding of messages from the client, so that the correct method of the correct controller is invoked.

In the server there is also a model that is a representation of the database server data structure.



* Spiega il database (class diagram)

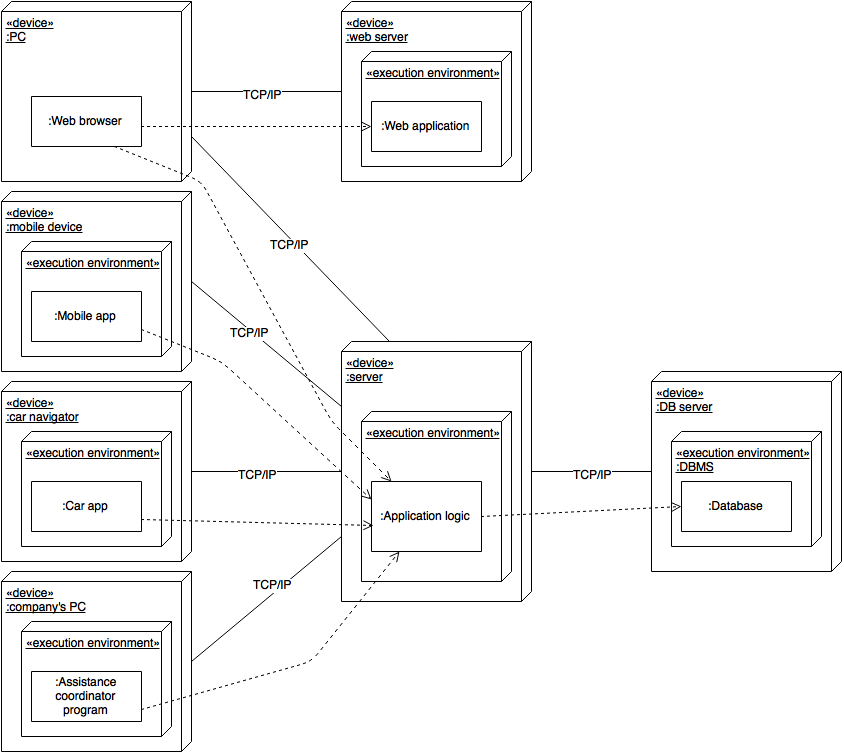
CLASS DIAGRAM DB

## 2.3. Deployment view

-Spiega a cosa serve

-Spiega perché TCP IP

-spiega del wer server



## 2.4. Runtime view



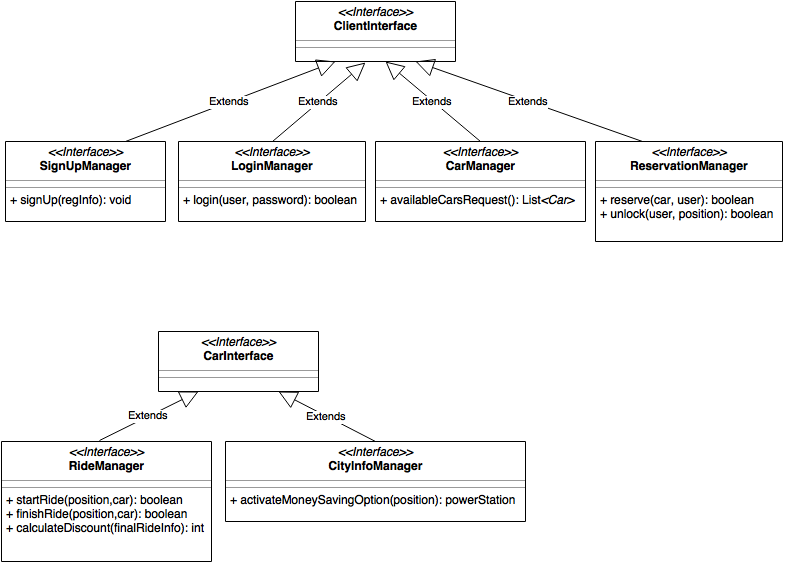
The client sends the reservation request to the reservation manager through one of the method offered by its view. The resource manager needs to ask to the car manager whether the reserved car is still available: if it is, a new reservation will be instantiate by the reservation manager that will also set the car and the user of the new reservation and start the reservation timer. If the car is no longer available an error will be notified to the clientApp.



Once the client is at most five meters distant from the car he is reserved, he can send the request to unlock the car to the sever. This request is handled by the reservation manager who checks whether the client is actually close to the car. If this check goes well the reservation manager declares expired the reservation and unlocks the car. Then informs the ride manager that a reservation has just turned into a Ride. The ride manager is in charge of instantiating the new Ride and “waking-up” the CarApp on board the interested car. The new Ride will be instantiate only once the user will ignite the car. If the reservation manager detects that the user is more than five meters far from the car an error will be notified to the ClientApp.

## 2.5. Component interfaces

Due parole introduttive



## 2.6. Selected architectural styles and patterns

-mvc server

## 2.7. Other design decisions

inventa

# 3. Algorithm design

# 4. User interface design

# 5. Requirements traceability

# 6. Effort spent

# 7. References