Design Document

Simone Mosciatti & Sara Zanzottera

December 4, 2016

Contents

1 Introduction 3
- 1.1 Purpose . . . 3
- 1.2 Scope . . . 3
- 1.3 Definitions, Acronyms, Abbreviations . . . 3
- 1.4 Reference Documents . . . 5
- 1.5 Document Structure . . . 5

2 Architectural Design 6
- 2.1 Overview . . . 6
- 2.2 Interface . . . 6
- 2.3 Components . . . 8
- 2.4 Deploying . . . 11
- 2.5 Logical Deploying . . . 11
- 2.6 Deploy . . . 12
  - Reason Technology Choice . . . 13
- 2.7 The rest . . . 13
- 2.8 Component View . . . 15
- 2.9 Deployement View . . . 15
- 2.10 Runtime View . . . 15
- 2.11 Component Interfaces . . . 15
- 2.12 Architectural Styles and Patterns . . . 15
- 2.13 Other Design Decisions . . . 15

3 Algorithm Design 16

4 User Interface Design 17

5 Requirements Traceability 18

6 Conclusions 19
- 6.1 Tools used . . . 19
- 6.2 Hours of work . . . 19

1 Introduction

1.1 Purpose

This Design Document aims to provide to everyone involved in the actual development of the application specific insights about the structure of PowerEnJoy, its acthitecture's details, the desing patterns we chosed to implement, but also some details about its high level components, their interactions and general behavior.

1.2 Scope

PowerEnJoy is a digital management system for car sharing that exclusively employs electric cars to provide its service. The system provides all the functionalities normally provided by a car sharing service: registering to the service, find the location of nearby available cars, reserve cars up to a short amount of time, unlock the chosen car once found, ride it and then park it in a safe area, when it will be automatically locked and the fee paid.

In addition, the system gives bonuses and penalities in term of discounts or overprices depending on the behavior of the user, in order to promote virtuous behaviors.

PowerEnJoy is therefore a inherently distributed system, based on a central server interactions with many distributed nodes. In detail the system can be divided into four main parts:

- a public app, used by customers to access the service
- a centralized backend that provides the service
- the cars' onboard system, that communicates only with the centralized backend
- a reserved fronted, used exclusively by the staff members to better organize their job

All these four components will be examined in more detail in the subsequent sections of the document.

1.3 Definitions, Acronyms, Abbreviations

RASD Requirements and Specification Document.

DD Design Document.

User A customer of PowerEnJoy using the service.

Staff Operator An employee of PowerEnJoy which takes care of the cars.

Ride The action of getting onboard of a PowerEnJoy car, start its engine, drive to destination and park.

Running Time The time an user spends using the PowerEnJoy service.

Issue Any problem a car may incur in, or a user may face while using the service.

Nearby Cars Cars located within a maximum distance to a specific position.

Nearby Issues Issues that are affecting cars close to a specific position.

Booking (Reservation) The act to reserve a car for a limited amount of time for future use by a user.

Reservation's maximum time The maximun amount of time a car can be reserved.

Driver Whoever is driving a regularly booked PowerEnJoy car.

Passenger Whoever is in inside a PowerEnJoy car but is not the driver.

Driving License The state's issued driving license of the user.

Notification A form of comunication where the user is actively notified of some event.

Issue Report An incoming notification that states a car incurred in an issue.

Fine A fine issued by the local law enforcing officers to a user while driving a PowerEnJoy car.

Pending Bills Bills that an user still need to pay to PowerEnJoy .

Safe Area An parking area, predefined by the company, where is possible to safely park the cars of the PowerEnJoy fleet.

Battery Charge The amount of charge that is kept inside the car's battery.

Charging Station Dedicated areas where is possible to plug the PowerEnJoy cars to charge their batteries.

Car's Onboard System The controll system of the car that is able to exchange data with the central system and to relevate operation parameters.

Customer's App An implementation of the system frontend tailored to the need of the customers.

Operator's App An implementation of the system frontend tailored to the need of the staff.

Central System The central system for PowerEnJoy . All the command and all the data are streamed, analyzed and used here.

Credentials Pair {Username, Password} necessary to access the PowerEnJoy system.

GPS : Global Positioning System is a global navigation satellite system (GNSS) that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites.

System's Frontend The interface provided to the user of the PowerEnJoy system.

System's Backend The whole technical infrastructure necessary to PowerEnJoy .

1.4 Reference Documents

- *Assignments AA 2016-2017.pdf* (Assignments document given by the teacher)
- *Sample Design Deliverable Discussed on Nov. 2.pdf* (Sample document provided by the teacher)

1.5 Document Structure

1. **Introduction**

   This sections aims to explain the purpose and the scope of the document, introducing the reader to subsequent sections of the document itself.

2. **Architectural Design**

   This sections will explain the main architectural decision we made.

3. **Algorithm Design**

   In this section we focus on the most critical code section and we provide an in-depth analysis of how they should be structured, eventually providing pseudocode for them.

4. **User Interface Design**

   In this section we carry on the UX design with the help of UX and BCE diagrams, eventually completing them with updated and extended application mockups.

5. **Requirements Traceability**

   In this section we map the requirements stated in the RASD to the actual component or processes that fulfill these requirements.

6. **Conclusions**

   In this section we enumerate the tools we used to redact this document, the hours of work spent by each group member and the (eventual) revision history of the document itself.

2 Architectural Design

The overall design process has been carried in a bottom-up approach, starting from the analysis of the requirements moving upwards to the definition of the higher level components of the system. In the following sections we provide more details on the designed architecture.

2.1 Overview

The overall design process started from the interface between the world and the machine. Given our goals we identify what interface the machine should provide in order to accomplish such goals.

Once the interface have been identified we proced to organize those interface in components having care to respect the single responsability principle in order to provide highly decouple components.

Once the components are defined we proced to deploy them on the right part of our system.

Once the components are logically deployed on different parts of the system we started to reason about what technology use to actually implement them.

The rest of this section will follow our mental flow, guiding the reader in our reasoning and finally providing the overal design.

2.2 Interface

Starting from our goals we enumerate what interface should be define.

REGISTRATION

REGISTER The user provide it's personal information including license number and billing information and then is register to the system.

VALIDATION The system must validate the information provided.

LOGIN

LOGIN Provided the correct information the user is logged into the system and now has the possibility to book cars, unlock cars, paying riding, etc.

LOOKUP

LOOKUP A logged user is able to locate and visualize the position of cars.

BOOK

BOOK A logged user is able to book a cars.

UNBOOK

UNBOOK A logged user is able to cancell a prenotation previously made.

UNLOCK

UNLOCK A logged user in proximity of a car booked at its name is able to unlock it.

POSITION The system must be able to locate the user.

UNLOCK_CAR The system must be able to unlock a car.

RIDE

RIDE User can drive the car to their destination.

READ_LICENSE The system must be able to acquire information about the user driving license and to decide to let the user start the engine or not.

SHOW_INFORMATION The system must be able to show the user some information such as nearby safe areas or nearby power stations.

SAFE_AREAS

SHOW_INFORMATION The system must be able to show the user nearby safe ares.

UNSAFE_PARKING

TURN_OFF The system is able to turn off a car left is an unsafe area.

LOCK_CAR The system is able to lock a car left parked in an unsafe area.

POWER_STATION

SHOW_INFORMATION The system must be able to show the user nearby power stations.

CAR_PLUGGED The system must be able to detect whenever a car is plugged to a power station.

CHANRGE

CALCULATE_FEE The system is able to calculate the total fee that the user must pay.

SEND_FEE The system is able to communicate to the user the total cost of the ride.

PAYMENT

PAY The system ask the user to pay a bill.

FIND_ISSUES

RAISE_ISSUE The user is able to raise an issue about a car.

SUPPORT

TAKE_CHARGE The system allow operator to take charge of certain issues.

SET_STATUS The operator is able to change the status of an issues.

2.3 Components

Given the interface we have identify above we have organize such interfaces in the following components.

All the component are responsable to return meaningfull error messages in case of errors.

USER_MANAGER

Responsability Manage the users.

USER/Register

Responsability the functionality is responsible to register a new user into the system.

Input It need as input information from the user such as:

- Name
- Lastname
- Password
- Email
- License ID
- Credit Card informations

Output in case of success it provide as output the ID of the user just created.

USER/Login

Responsability the functionality allow the user to log into the system.

Input The component require as input the email and the password.

Output It log the user into the system.

LOCATION

Responsability This component takes care of locate elements.

LOCATION/AvailableCar

Responsability the functionality allow the user to retrive the position of the available car.

Input Geographically coordinates and radius of the research.

Output The set of available cars inside the circle of radius provided centered to the coordinated provide.

LOCATION/Areas

Responsability the functionality allow the user to retrive the position of the areas of interest, such as power station or safe parking area.

Input Geographically coordinates and radius of the research.

Output The set of area of intereste inside the circle of radius provided centered to the coordinated provide.

LOCATION/IssuesCar

Responsability the functionality allow the user to retrive the position of the car with some issues.

Input Geographically coordinates and radius of the research.

Output The set of cars with issues inside the circle of radius provided centered to the coordinated provide.

BOOK_MANAGER

Responsability This component will take care of managing the prenotations.

BOOK/Book

Responsability the functionality allow the user to book one of the available car.

Input The ID of the car and the ID of the user.

Output The car is booked and the ID of the prenotation is provided.

BOOK/Unbook

Responsability the functionality allow the user to remove a prenotation.

Input The ID of the user and the ID of the prenotation.

Output The prenotation is cancelled.

CAR

Responsability This component will manage all the iteraction between the users, the system and the actuall car.

CAR/Unlock

Responsability the functionality allow to unlock the car.

Input The ID of the car and the ID of the user requiring the unlock.

Output The car is unlocked.

CAR/Lock

Responsability the functionality lock the car.

Input The ID of the car.

Output The car is locked.

CAR/TurnOff

Responsability the functionality turn off the engines of the car.

Input The ID of the car.

Output The car is turned off.

CAR/Telemetry

Responsability the functionality allow to retrieve all the information associated with the car.

Input The ID of the car.

Output All the puntual information about the car.

POSITION

Responsability This component will locate persons, cars and areas.

POSITION/Car

Responsability the functionality allow to know the position of a car.

Input The ID of the car.

Output The coordinates of the car.

POSITION/User

Responsability the functionality allow to know the position of an user.

Input The ID of the user.

Output The coordinates of the user.

POSITION/Areas

Responsability the functionality allow to know the position of an area of interest.

Input The ID of the area.

Output The coordinates of the area.

BILLING_SYSTEM

Responsability This component will manage all the fees.

BILL/Calculate

Responsability the functionality calculate how much is a riding fee.

Input The ID of the ride.

Output The total cost.

BILL/Pay

Responsability the functionality ask the user to pay a specific bill.

Input The ID of the user and the ID of the ride at which the bill refer to.

Output The request of payment is done.

BILL/Collect

Responsability After the user confirm the payment this component will actually collect the money.

Input Credit car information of the user and the total fee.

Output The money are collected.

ISSUE_MANAGER

Responsability This component will manage the issues at the cars.

ISSUE/New

Responsability the functionality rise a new issue.

Input A reference to a car, the ID of the user raising the issue and a description of the issue.

Output The ID of the issue.

ISSUE/Modify

Responsability the functionality let the operators modify the status of some issues, the operator may have fixed the issue, may decide that he is not capable to fix the issues or it may decide that the issues is un-fixable.

Input The ID of the issue, the ID of the operator and the new status.

Output The status of a issue is fixed.

ISSUE/TakeCare

Responsability the functionality let the operator take charge of a particular issues.

Input The ID of the issue, the ID of the operator.

Output The operator is now responsable for the issue.

2.4 Deploying

Now that we have understand and named our componet we can start logically deploying them into our architecture.

As we have already anticipated in the RASD we are going to adopt a Client-Server between our main services and the users.

Then we have identify that is extremely practical to use a Pub-Sub mechanism for the communication between our fleet and the main server.

Moreover we decide to completely avoid all the digital communication between the cars and the final user.

The system is so divided into three parts, the fleet, composed by cars, the user applications and the main server.

Clearly most functionality are invoke in one part of the system and execute in another part of the system following standard comunication mechanism.

2.5 Logical Deploying

USER_MANAGER The user manager is logically deployed in the main server, while its functionality are invoke only by the users.

LOCATION The location service is logically deployed in the main server, its functionality are invoked by the users (LOCATION/AvailableCar), by the fleet (LOCATION/Areas) and by the staff (LOCATION/IssuesCar).

BOOK_MANAGER The book manager is deployed on the main server while its functionality are invoked by the users.

CAR This component is logically deployed in both the main server and the fleet. The functionality CAR/Unlock, invoked by the users, involve both the main server, that guarantee that the request is legitimate, and the cars, that actuall unlock the door. The functionalities CAR/Lock and CAR/TurnOff are both invoked by the server and actuatted by the cars. Finally the functionality CAR/Telemetry is logically invoked by the server and actuatted by the cars.

POSITION This component is deployed on the cars, on the users app and on 3rd part system. POSITION/Car is deployed on the car itself and invoked, indirectly, by the user. POSITION/User is deployed on the user application and invoked by the main server and, indirectly by the user itself (when the user ask to unlock a car he must be nearby the car itself). POSITION/Areas is deployed on 3rd part system and on the main server, its functionality are invoked by the car.

BILLING_SYSTEM The functionality of this component are logically deployed on the main server, the users' app and in a 3rd part system. BILL/Calculate is deployed and invoked by the server. BILL/Pay is invoked by the server but executed on the users application; while BILL/Collect is invoked in the server but actually executed in a 3rd part system.

ISSUE_MANAGER This manager is deployed in the main server and it functionality are invoked by the staff and by the users.

2.6 Deploy

At this point we have understood, logically, where each component should be deployed and what part of the system invoke what functionality.

Now we are going to physically deploy all the functionality in the correct part of the system and we are going to define a communication mechanism between those functionality.

The interface between the several functionality provided by the components are already defined, from this point on we are going to pick a particular technology only for the sake of simplicity, all the possible communication technology may have been choosen. Obviously some choices are more apt than others with the respect of latency, throughput, elegance of the design and other factors; however the whole design will remain intact whichever technology we choose.

Reason Technology Choice

The reason for the technology choice we made are expressed in this section.

The main server will expose its API in the most conventional possible way, using the classical HTTP/TCP/IP stack. In particular we focus ourselves in provide RESTfull interfaces. Model everything as an entity will provide enough capabilities to actuall implement the whole system while remaining constrainted to only the basic REST verb will help in keeping the whole API simple.

The users app will consume the REST interface provide by the server, however to implement the communication between the server and the app we will use the long polling strategy.

We believe that the car will need to communicate very often a lot of valuable information to the main server and in order to achieve high troughput and low latency we believe that the PubSub protocol is the most apt. Moreover, the PubSub protocol is pretty natural in this scenario, having the car publish messages about it own status and having the the server subscribe to those messages. Also most of the communication between the server and the car will happens using the PubSub protocol. Between all the implementation of the PubSub protocol we chose to use MQTT for its low overhead, its QoS and because it is widely used in the industry.

2.7 The rest

The high lever architecture of the system is made up of three main elements:

- The main server
- The mobile apps (user apps and staff apps alike)
- The car's onboard system

On top of these elements, there is a number of external services the servers interacts with in order to provide functionalities to the apps or to the cars' system.

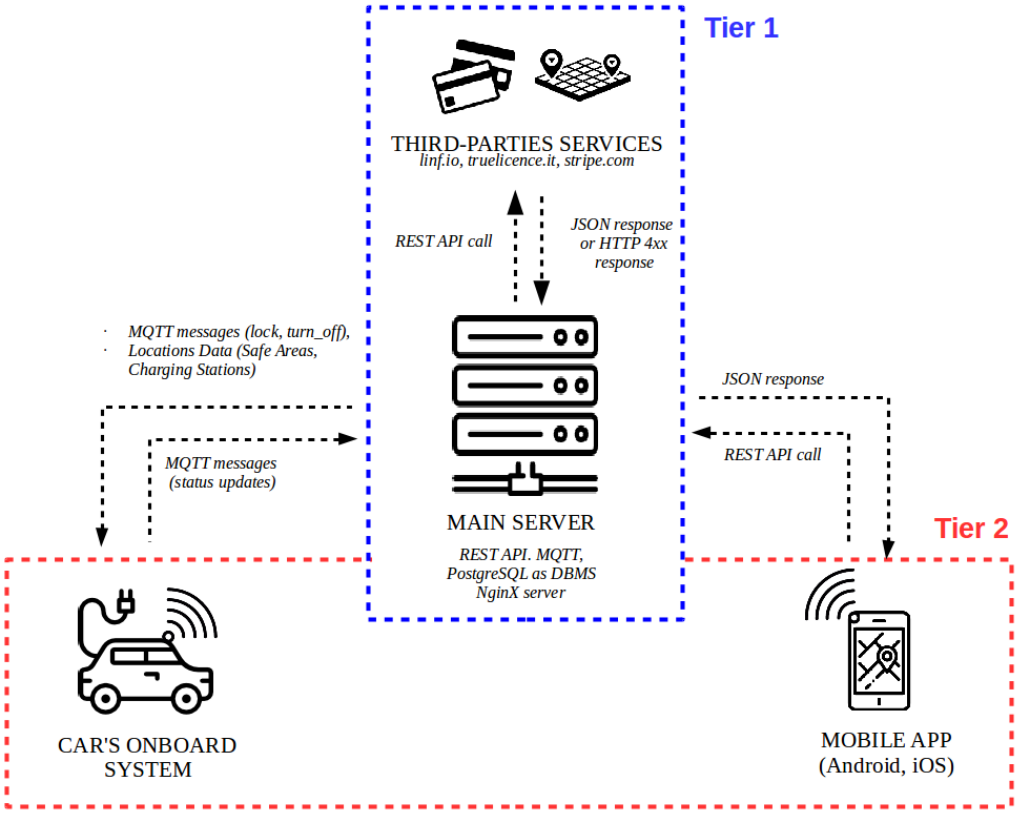

Figure 1: High level overview of the architecture

At this stage, the system's architecture is clearly two-tier:

- Tier 1, the main server, handles the application logic and data management.
- Tier 2, comprising mobile apps and cars, hosts the User Interface.

The system blends three different interaction models for each of the three pair of components interacting. In detail, we designed:

- a pure **Client Server** approach when the main server interacts with the apps (customer's apps and staff's apps alike)
- a **Service Oriented** communication model between the main server interacting with external services like linf.io, truelicence.it, stripe.com etc.
- a **Publisher Subscriber** model between the main server and the cars' systems.

In subsequent sections we will provide more details about these components.

2.8 Component View

2.9 Deployement View

2.10 Runtime View

[Includes sequence diagrams to show how components interact to accomplish specific use cases]

2.11 Component Interfaces

2.12 Architectural Styles and Patterns

[Explain patterns used above]

2.13 Other Design Decisions

3 Algorithm Design

[Definition of critical sections of code]

4 User Interface Design

[UX Diagrams and mockups, possibly extended]

5 Requirements Traceability

[Map requirements with components]

6 Conclusions

6.1 Tools used

During the development of this document we used the following tools:

- **Github** to version control the project
- **LaTeX** on TeXworks to redact this document
- **www.draw.io** to draw UML graphs
- **Gimp v.2.8** to mockup the application

6.2 Hours of work

- SZ: overall 1 hours
- SM: overall X hours