Politecnico di Milano

Software Engineering II project

PowerEnJoy

Design Document

Authors:
Davide Piantella
Mario Scrocca
Moreno R. Vendra

Professor:
Luca Mottola

January 22, 2017

version 1.4

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose of this document

In this document we are going to describe software design and architecture of the PowerEnJoy system.
The software architecture of a system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.[1]

1.2 Scope

PowerEnJoy is a car-sharing service that exclusively employs electric cars; we are going to develop a web-based software system that will provide the functionalities normally provided by car-sharing services, such as allowing the user to register to the system in order to access it, showing the cars available near a given location and allowing a user to reserve a car before picking it up. A screen located inside the car will show in real time the ride amount of money to the user. When the user reaches a predefined safe area and exits the car, the system will stop charging the user and will lock the car. The system will provide information about charging station location where the car can be plugged after the ride and incentivize virtuous behaviours of the users with discounts[2]

1.3 Glossary

The *PowerEnJoy: Requirements Analysis and Specification Document*[2] should be referenced for terms not defined in this section.

1.3.1 Definitions

Base cost: cost before any discount or fee application, obtained from rent time and time-based cost

1.3.2 Acronyms

RASD: Requirements Analysis and Specification Document

DD: Design Document

API: Application Programming Interface

GPS: Global Position System

DB: DataBase

DBMS: DataBase Management System

GIS: Geographic Information System

ER: Entity Relationship Model

XML: eXtensible Markup Language

REST API: REpresentational State Transfer API

JAX-RS: JAVA API for REST Web Services

ISP: Internet Service Provider

ARP: Address Resolution Protocol

1.3.3 Abbreviations

m: meters (with multiples and submultiples)

w.r.t.: with respect to

i.d.: id est

i.f.f.: if and only if

e.g.: exempli gratia

etc.: et cetera

1.4 Reference documents

Context, domain assumptions, goals, requirements and system interfaces are all described in the *PowerEnJoy: Requirements Analysis and Specification Document.*[2] Others references are:

- IEEE Std 1016-2009 Standard for Information Technology, Systems Design, Software Design Descriptions

1.5 Document overview

This document is structured as

1. **Introduction**: it provides an overview of the entire document
2. **Architectural design**: it describes different views of components and their interactions
3. **Algorithm design**: it focuses on the definition of the most relevant algorithmic part
4. **User interface design**: it provides an overview on how the user interfaces of our system will look like
5. **Requirements traceability**: it explains how the requirements we have defined in the RASD map to the design elements that we have defined in this document.
6. **Appendices**: it contains references, software and tools used and hours of work per each team member

2 Architectural design

2.1 Overview

2.1.1 Context viewpoint

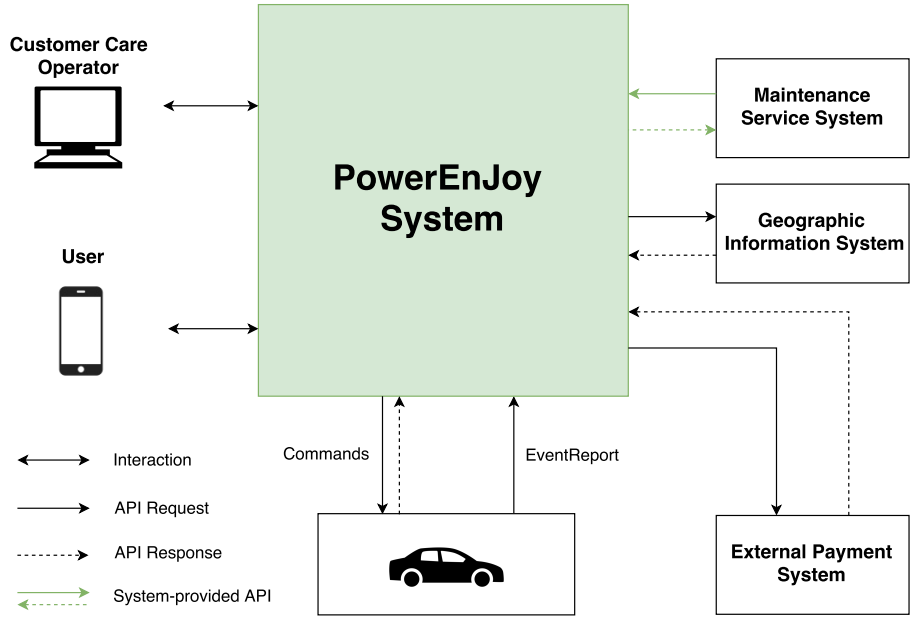

Figure 1: Context viewpoint

We need to design a system which allows communications with many agents such as cars, users, external systems, etc. Moreover we recognize that in most of the interactions the system is providing a service to agents so, after taking in consideration different alternatives, we decided to use a client-server architectural approach.

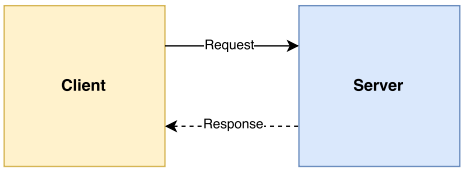

Figure 2: Client Server architecture

Cars offer to the system a set of primitives which allow it to interact with them: in this case it is clear that cars are providing the system services, so they can be identified as servers while the system acts as a client; on the other end the notification functionality offered by cars clearly yields to an event-based approach due to the asynchronous nature of such interactions, this led us to use a publish-subscribe paradigm for these specific interactions.

2.1.2 Composition viewpoint

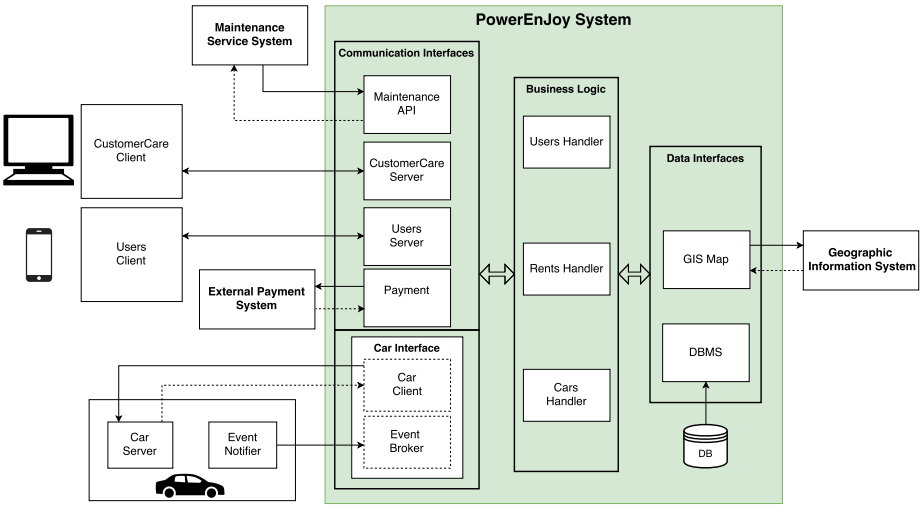

Figure 3: Composition viewpoint

Going deeper in the analysis of our system composition, we are able to identify some of the modules that will be required in order to provide the functionalities specified in the Requirement Analysis and Specification Document.

Communication Interfaces Since our system interacts with many external agents, it needs to have different *Communication Interfaces* in order to communicate with them.

- An API is needed to provide *Maintenance Service System* the information it needs to work with us
- A software module is needed in order to provide users functionalities of the system
- A software module is needed to provide the *Customer Care* the functionalities it needs
- An internal payment software module will deal with the communication with the *External Payment System*
- A set of modules will manage the communications between the cars and the system: a module to call primitives on cars through the provided API and another one to observe events triggered by the car. When the trigger method is called on *CarClient*, it enables the requested triggers on the car passed as a parameter. This allows the cars to notify to the system the events related to the aforementioned triggers; these events will be received and dispatched by the *EventBroker* to the subscribed components.

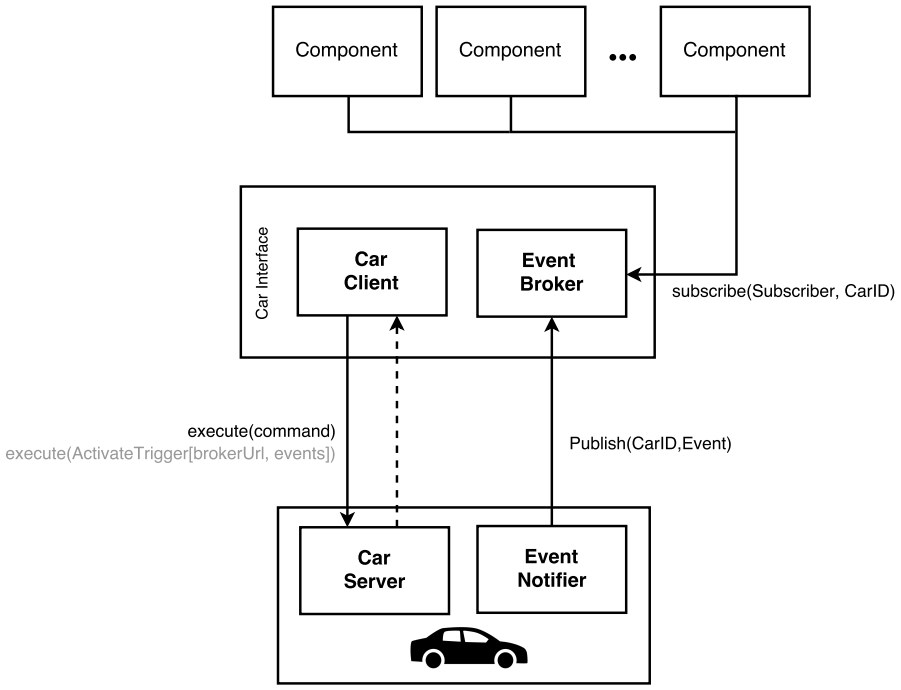

Figure 4: Car communication Interface

Business Logic The actual application logic of our system needs to manage the users information, the rents and the cars information; for each of these purposes several software modules are necessary; they will use communication interfaces to communicate with the agents and they will be able to retrieve data from the data interfaces.

Data Interface Our system needs a way to access and store the data it produces or retrieves from external resources, that is why *Data Interface* modules are needed. These modules allows interaction between the *Business Logic* modules and the System Databases; moreover they provide an interface to communicate with the GIS in order to allow the *Business Logic* modules to access its functionalities.

2.2 Component view

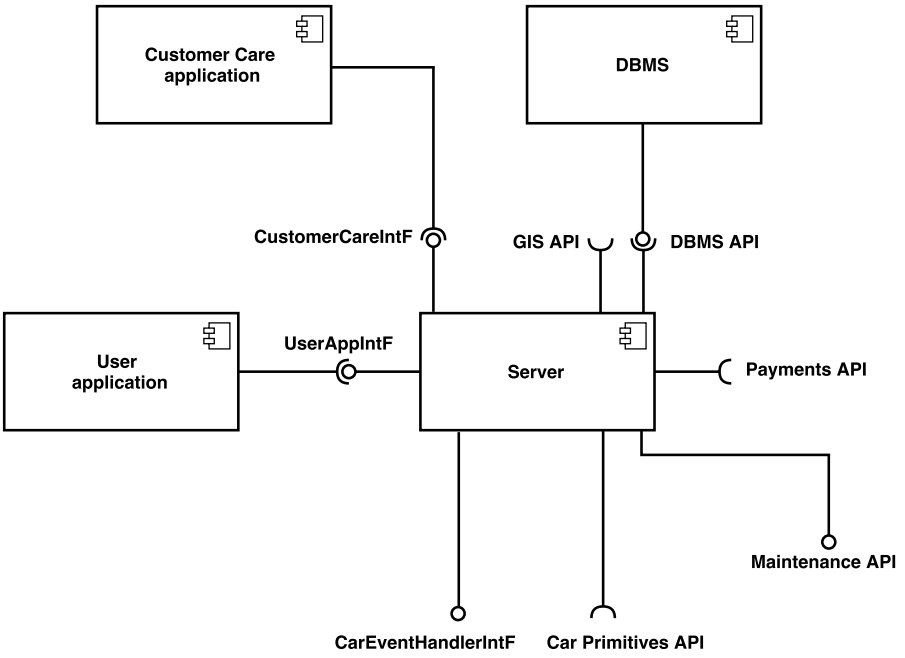

Figure 5: High-level components

2.3 Deployment view

2.3.1 Four tier architecture

Taking into account that:

- the *Composition viewpoint* diagram shows the need of database decoupling from the actual system
- in the *Server component view* we can clearly distinguish modules who take care of presentation and communication with the client
- in the *Server component view* we can clearly distinguish modules who take care of the specific application logic

we decided to design the system on a four tier architecture pattern (see also

Mapping of Server components on architecture: to clarify at a finer level how the Server components are mapped in the four tier layered architecture the following diagram represents the Server components highlighted in the same color of the tier in the previous diagram:

- Client: components used by users in order to access the functionalities offered by the system
  - UserApplication

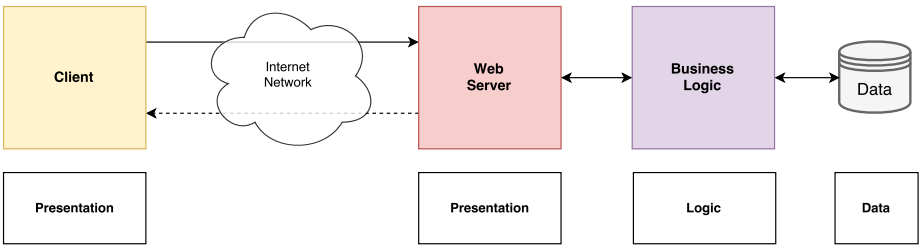

Figure 6: Four tier architecture with internet layer

  - CustomerCareApplication

- Web Server: components which provides interfaces to clients in order to allow them to use functionalities offered by the system
  - UserAppServer
  - CustomerCareServer
- Business Logic: components which realizes the functionalities offered by the system
  - RentManager
  - AccessManager
  - UserInformationManager
  - MaintenanceManager
  - EventBroker
  - CarHandler
  - DataProvider
- Data: components which store and manage the access to the data produced and needed by the Business Logic
  - DBMS

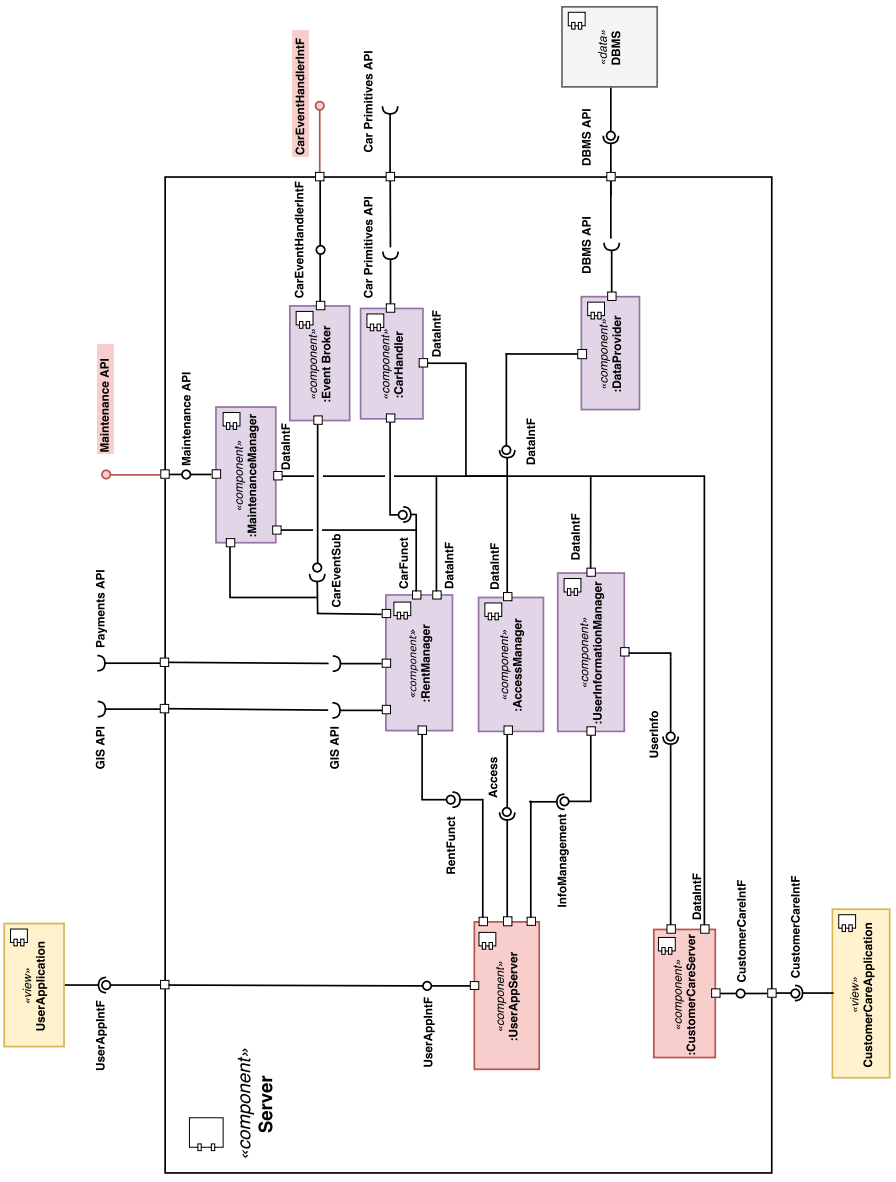

Figure 7: Mapping server component on architecture

2.3.2 Deployment diagram

The diagram in Figure 8 represents the mapping of the software components depicted in Figure 7 and Figure 5 on the devices that will run them. Many design concerns were considered while developing this solution.

Security: security is ensured in different points of the architecture, in particular the *UserAppServer* uses HTTPS as communication protocol to communicate with the users; the devices running the *CustomerCareApplication* component are in a VPN with the device running the *CustomerCareServer*.

Scalability: this model of deployment is scalable in the sense that the system administrators will be able to add more devices and deploy more instances of the needed components when and where performance issues will arise, in order to maintain a minimum level of performance even with loads increase.

Decoupling: decoupling in this architecture is present at different levels; in the deployment diagram it is clear that the each of the four tier runs on different devices, moreover the *UserAppServer* component runs on a different device than the *CustomerCareServer*, the Maintenance API and the *CarEventHandlerIntF* interface.

Redundancy: in this iteration of the architecture no redundancy of components or devices is present, but it is allowed in prevision of future expansions of the system's infrastructure.

Fault Tolerance: deploying different components on different machines allows the system to be easier to recover in case of a problem on one of the machines; for an example the Web Server running the *UserAppServer* component goes down, it can be replaced with another machine and in the mean time the *ApplicationServer* would still be up and running and still be able to provide the *CustomerCareApplication* with its functionalities.

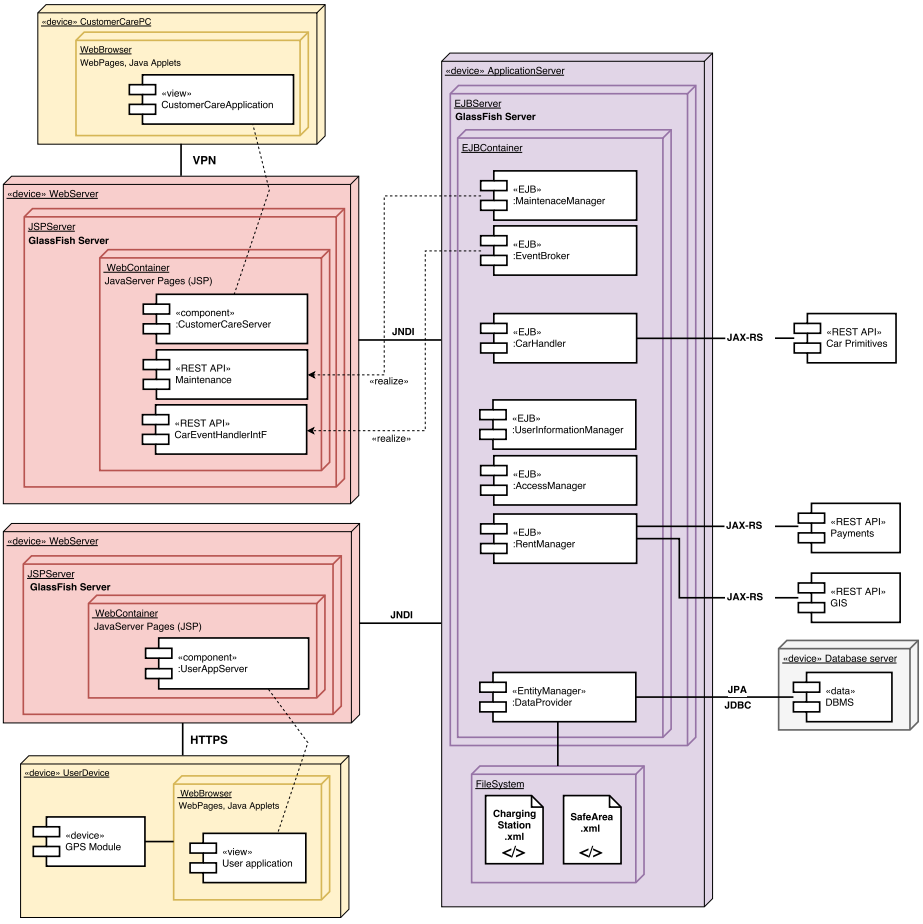

Figure 8: Deployment diagram

2.4 Runtime view

In this section we represent some runtime views of the supposed interactions between the components of the *PowerEnJoy* system.

Notes to read the diagram When an attribute is modified in a JPA object the changes are reflected into the *DataProviderComponent* in order to keep the database updated.

2.5 Component interfaces

3 User interface design

In this section are presented some mockups of the main features and related user interfaces the system is supposed to offer to the user and to the customer care through the proper web-based application.

The presented mockups have been designed based on both the *PowerEnJoy: Requirements Analysis and Specification Document*[2] and on the architectural design decisions and component interactions presented on this document.
In particular mockups show how the user interface is supposed to offer to the user the possibility to interact and make request to the system (obviously such user's interactions will result in a client-server communication of the user/customer care app view with the related server component based on the protocol chosen for that communication).

The main goal of our mockups design process is to build an interface that clearly distinguish functionalities offered by the system taking into account the architectural decoupling offered by the taken design choices.

3.1 User app

The user app must have a charming and intuitive user interface in order to provide a easy-to-use experience to the user. The user app is supposed to be a web-based application, so the interface must be optimized for mobile devices even if the application is accessible and must be usable from every web browser on different size devices.

3.1.1 Login page

Simple initial page for the application to allow the user to authenticate to the system through username and password. A new user can access the registration process through the *New User* button.
As specified in the requirements document, this page also allows a guest user or a *banned* registered user to access to customer care contact information through the *Contact us* button.
If a user is not recognized or is banned an error alert is shown when credential are submitted.

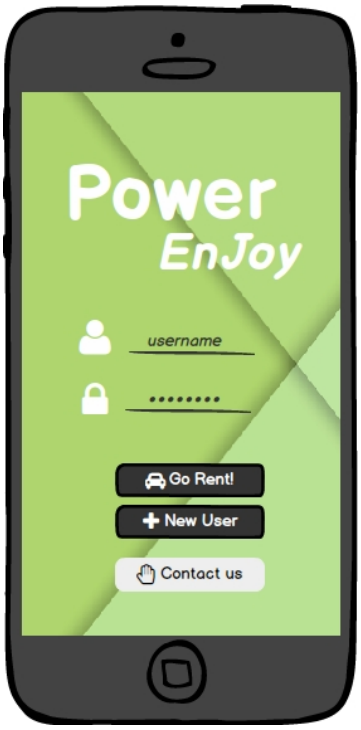

Figure 9: *Login page* mockup

3.1.2 Home page

The home page shows to a logged user all the possible functionalities provided from the app:

- See available cars and reserve one of them (eventually with the *money saving option*)
- Unlock a car (disabled button in the mockups, it would be active only if there is an active reservation for the registered user logged in)
- See user's information and edit them
- See user's rent history
- See user's payment history
- Show customer care contact information

This page shows also the user's name to ensure to the user he has been correctly recognized by the system and to make the interface more customized.

The icon in the right corner, from this page, brings the user to the functionality of see available cars; on all other pages (without the orange position icon) it brings the user to this page: the home page.

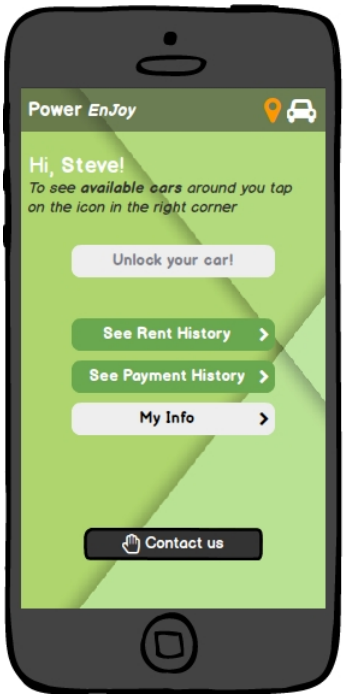

Figure 10: *Home page* mockup

3.1.3 See available cars

Accessing to the see available cars functionality, as described in the requirements document, the user app asks to the user if he wants to search car nearby his actual position (using GPS position of user's device) or he wants to insert a different position from which starting the research.

The *Use GPS* allows the user to choose the first option skipping other interactions on this page.

If the user chooses the second option, he is supposed to insert an address location (e.g. 34 Maria Victoria Lane, London) before pushing the *Search* button; the system will resolve that address as a GPS location displaying an error message if it could not done it.

The *Cancel* button allows the user to return to the home page.

The system searches for available cars (nearby the position given by the user or retrieved by the GPS) and displays a map with available cars on their actual position, charging station positions (green spot) and safe areas (black areas). Blue circled cars are actually plugged on a charging station, all others car are green circled.

The map is interactive and the user can move around the actual position.

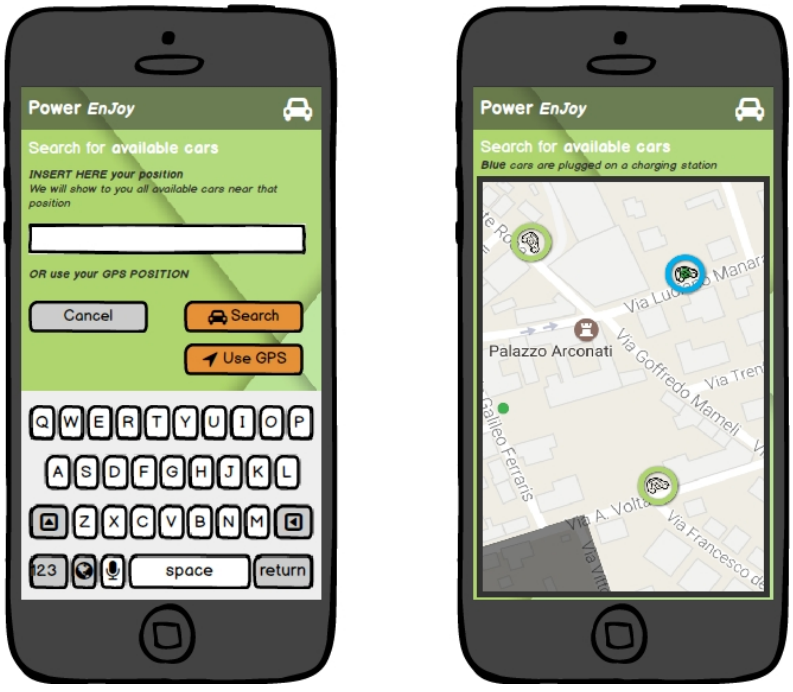

Figure 11: *See available cars* mockup

3.1.4 Reserve a car

From the page showing available cars on the map, a user tapping on a car could access information about it, in particular:

- Model of the car
- License number of the car
- The battery percentage level of the car

When the user taps on a car the circle around it becomes orange, to give a feedback on the tap to the user, and a box appears on the screen. Through it the user can access the aforementioned info about the car and reserve the selected car.

Before clicking the *Reserve it!* button the user has the possibility to choice if he wants or not to enable the *money saving option* through an on/off toggle.

If a user has already an active reservation or the selected car has been reserved while the user navigates on the map an error message is displayed.

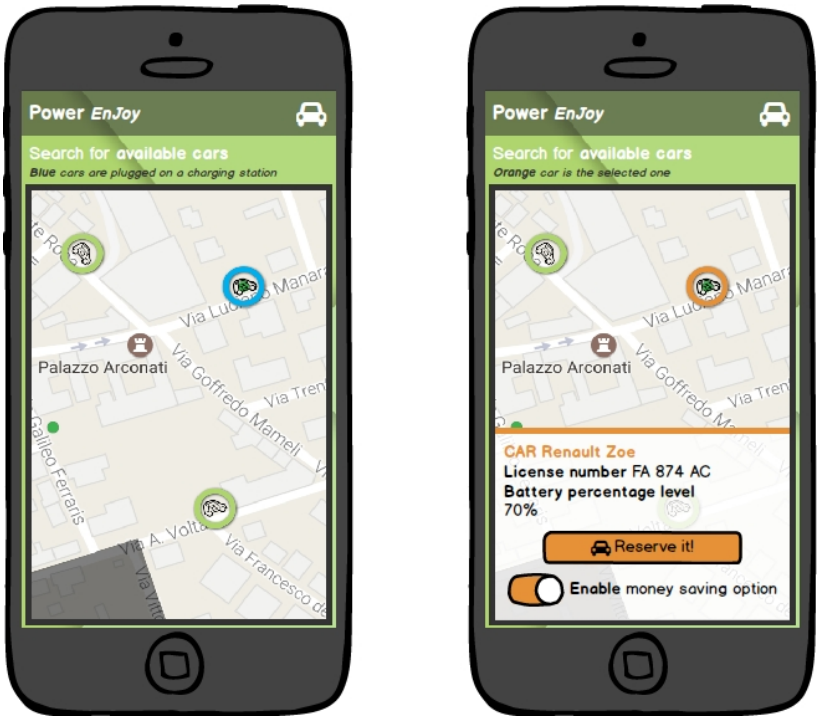

Figure 12: *Reserve a car* mockup

3.1.5 Money saving option

If an user enables the *money saving option* while reserving a car, the user app shows him a dedicated page to accomplish the reservation with the option. On this page the user must insert the planned destination of his rent in order to give to the system the possibility to calculate the charging station the user must leave the car plugged in to get the discount.

A brief description of the *money saving option* is offered to the user to clarify why the user app is asking him his planned destination.

The user is supposed to insert an address location (e.g. 34 Maria Victoria Lane, London) before pushing the *Confirm* button; the system will resolve that address as a GPS location displaying an error message if it could not done it.

If the address inserted by the user is correctly processed the system notifies the user with the charging station (number and address) the user must leave the car plugged in to get the discount.

The *Cancel* button allows the user to go back on the map, for example to make the reservation without enabling the *money saving option.*

Note Note that if the user chooses to enable the *money saving option* the car reservation request is sent to the server only when the user inserts also the destination.

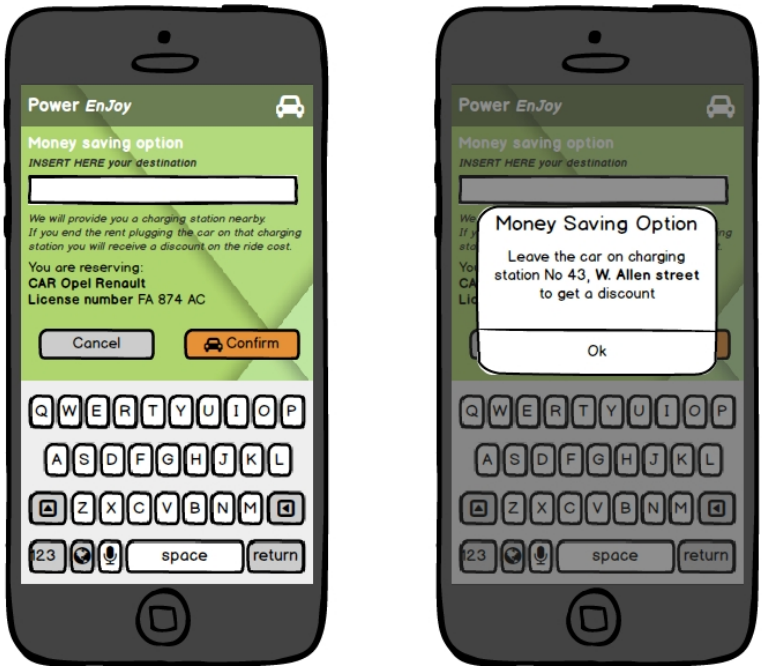

Figure 13: *Money saving option* mockup

3.1.6 Unlock a car

From the home page, if the user has an active reservation he can access to the *Unlock car* functionality through the dedicated button.

On this page the user can see data about his reservation:

- model and license number of the car he has reserved
- time until the reservation expires
- (*Optional*) charging station related to money saving option
- current car position (clicking on *see it on the map* link the user app looks for an installed program on the user device to open the GPS coordinates, if it can not find any program it only shows the GPS coordinates)

The user could ask the system to unlock the car through the *Unlock your car* button. If the user GPS position is not 5m away from the car an error message is displayed to ask the user to reach the car before trying to unlock the car.

If the system manages to process the request to unlock the car a message is displayed to notify the user the car has been unlocked and he could start the rent turning on the engine.

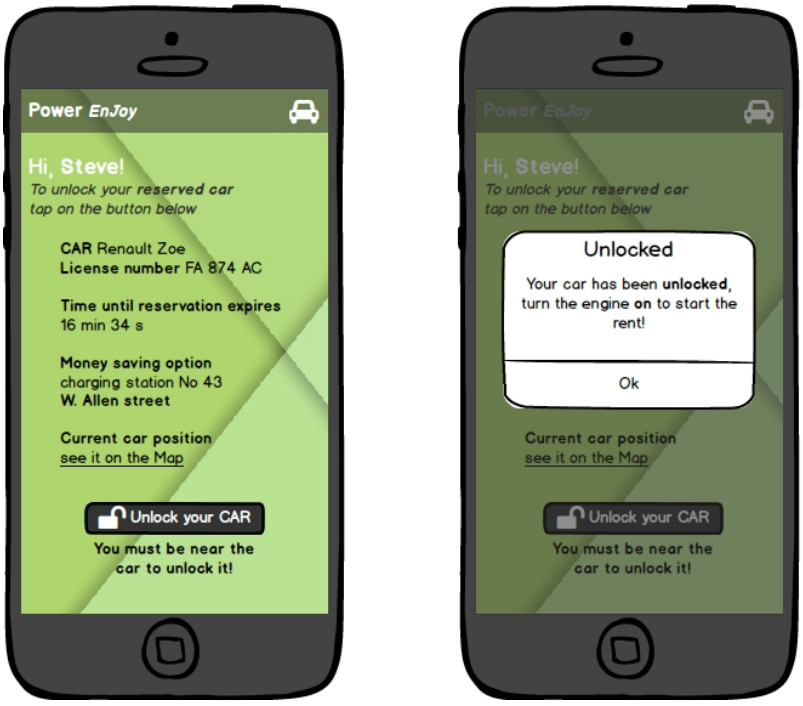

Figure 14: *Unlock a car* mockup

3.1.7 Payment history

From the home page, the user can access his own payment history through the *See Payment History* button.

On this page the user app shows to the user all made payments in chronological order, from the more recent to the latest ones as shown in Figure 15.

Payment not related to a rent, for example fee related to an expired reservation, are shown with a different color to clearly distinguish it.

Through this page a user can only see if a payment is related or not to a rent and date and hour of each payment. The user could access all payments details clicking on one of the rows shown, or can rapidly access to customer care contact information if for example it finds out some payment he is not aware of.

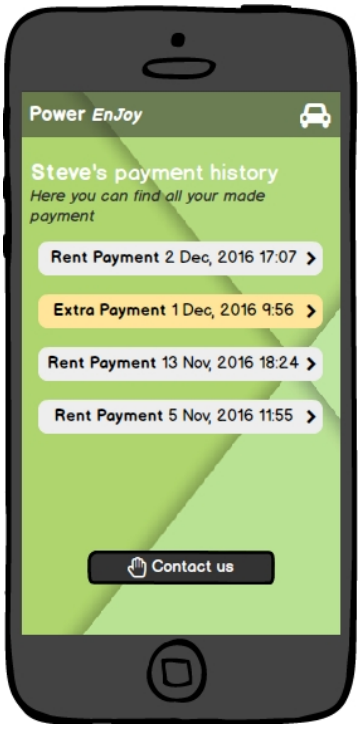

Figure 15: *Payment history* mockup

In Figure 16 are shown two examples of payments history's single record. The one on the right shows a payment related to a rent, instead the one on the left shows a payment related to a reservation expired fee.

For each payment record the user can see information about:

- payment ID
- time of the payment
- base cost (in case of rent it's calculated as *rent time* x *time based rate*)
- discount amount applied on the rent
- fee amount applied on the rent
- total paid amount
- payment used method (for security reason only the last four numbers are shown)
- (*optional*) rent associated with the payment

If the payment is associated with a rent, the user can access the rent record through the link in the *rent* section.

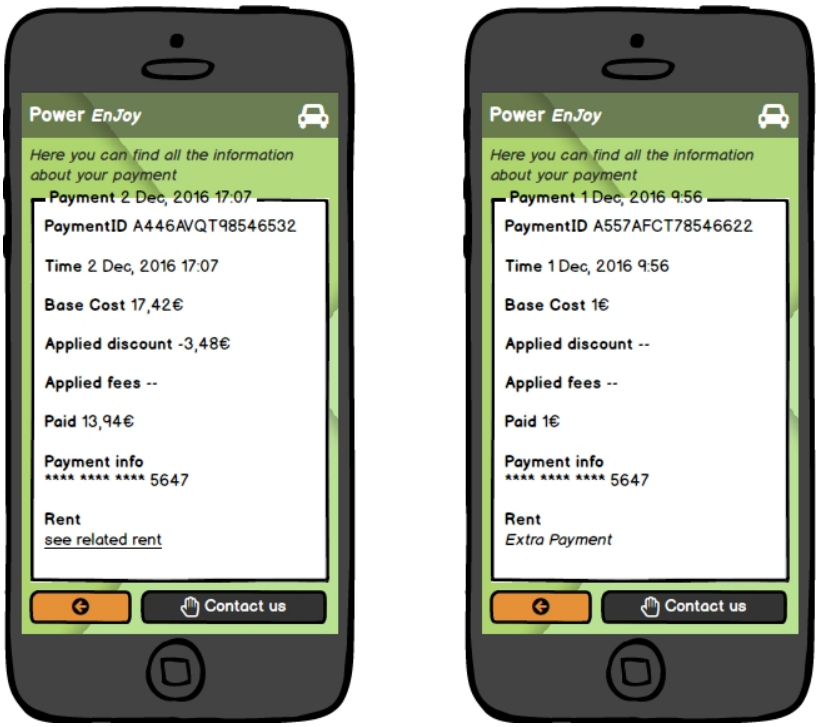

Figure 16: *Single payment records* mockup

3.1.8 Rent history

From the home page, the user can access his own rent history through the *See Rent History* button.

On this page the user app shows to the user all made rents in chronological order, from the more recent to the latest ones as shown in Figure 17. Through this page a user can only see date and hour of each rent. The user could access all rents details clicking on one of the rows shown.

For each rent record the user can see information about:

- rent ID
- start/end time and location of the rent
- all discount applicable to the rent
- all fee applicable to the rent
- payment related to the rent

The user can access the payment record related to the rent through the link in the *payment* section.

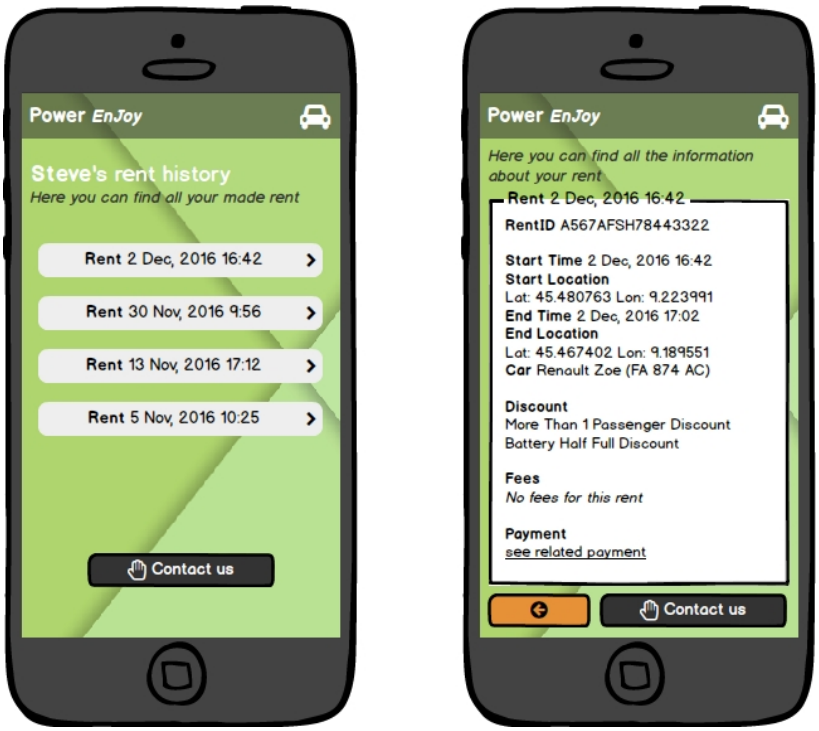

Figure 17: *Rent history* mockup

3.2 Customer care app

The customer care app has a simple interface in order to provide features to customer care operator in a clear and simple way. This interface must be optimized for a desktop monitor size.

3.2.1 Home page

All main features are accessible directly from the home page to provide a rapid and intuitive access to them.
From the home page the operator can:

- Access information about a user through the user's username or email address (see user's information section)
- Mark/Unmark user as banned through its username (if the username inserted is found by the system, in order to fulfil this task a new page is shown otherwise a new error message is displayed. The new page shows to the operator the current state of the user and if the operator wants to mark the user as *banned* asks the operator a brief description of the reasons)
- Retrieve basic data for each user (username, name, surname, email)
- Tag a car as *Not Available* (if the car license number inserted is found by the system, in order to fulfil this task a new page is shown otherwise a new error message is displayed. The new page asks the operator a brief description of the reasons why he wants to mark the car as Not Available)

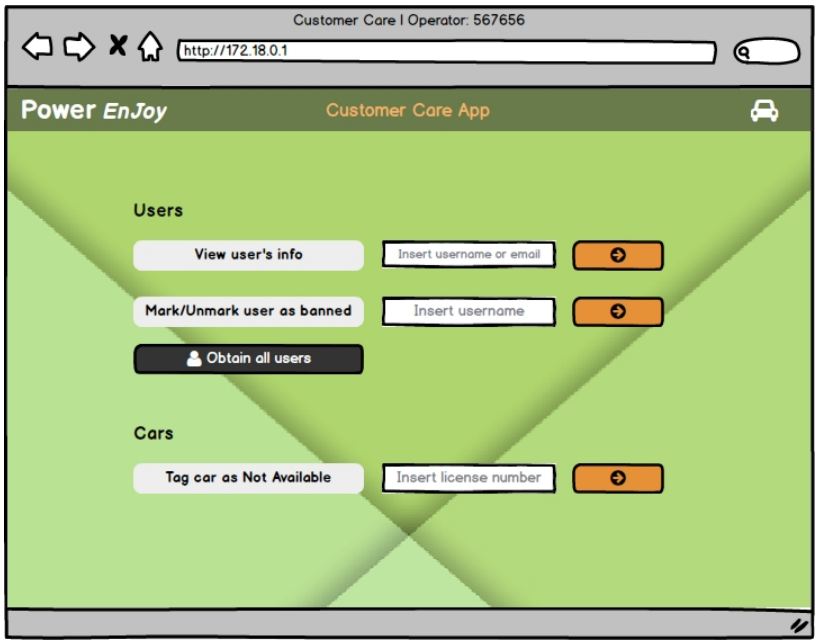

Figure 18: *Customer care home page* mockup

3.2.2 User's information

From the home page the operator could access information about a specific user. In this page the customer care app shows to the operator all user's information unless the sensitive ones (as password and credit car number).

From this page the operator could also access user's rent and payment history through specific buttons.

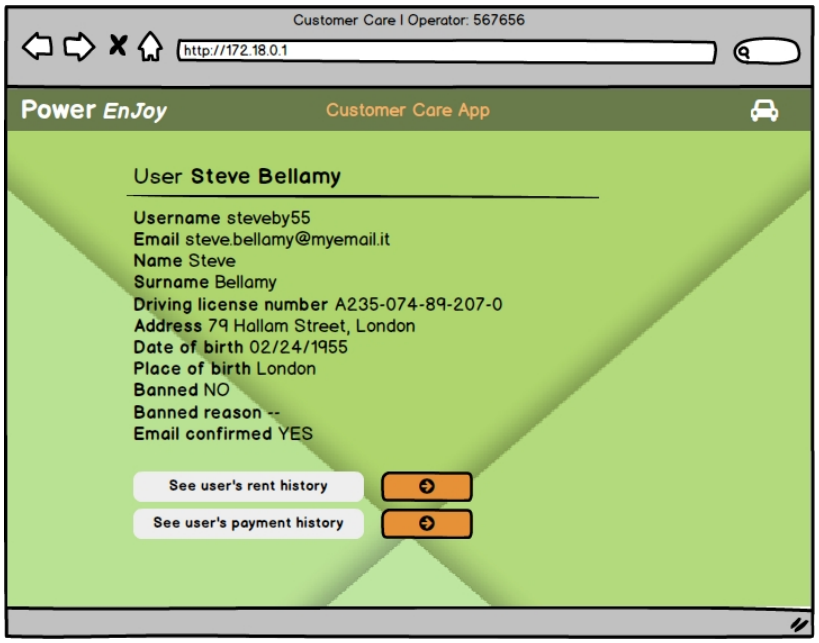

Figure 19: *Customer care user's information page* mockup

4 Requirements traceability

4.1 Functional requirements

In the Table 1 is presented a mapping correspondence between the requirements defined in the RASD related to each goal and the components identified in the server component diagram.

Requirements of goal	Components
G1 Allow guest users to register to the system	UserAppServer AccessManager DataProvider UserInformationManager
G2 Allow registered users to authenticate to the system	UserAppServer AccessManager DataProvider
G3 Provide logged users with the position of available cars	UserAppServer RentManager DataProvider
G4 Notify maintenance service with a list of not available cars	MaintenanceManager DataProvider EventBroker CarHandler
G5 Provide the maintenance service with a way to notify the system when a car is available again	MaintenanceManager DataProvider
G6 Provide users with a way to report a damaged car	UserAppServer CustomerCareServer
G7 Provide a way to show to each user his rents and payments history	UserAppServer UserInformationManager DataProvider
G8 Provide customer service with a way to ban registered users in order to prevent them from reserving or using other cars, and enable them to use the service again	CustomerCareServer UserInformationManager DataProvider
G9 Allow a logged user to reserve a car, if available, and hold that reservation for an hour	UserAppServer RentManager DataProvider
G10 Charge a user for 1€ in case he hasn't used the car he reserved after an hour from such reservation	UserAppServer RentManager DataProvider

G11 Allow a logged user to perform a complete rent: reserving a car, using it and leaving it terminating the rent, accomplishing the payment procedure related to aforementioned rent	UserAppServer RentManager EventBroker CarHandler DataProvider
G12 Calculate and charge the user for the correct amount of money he has to pay for his last ride, also considering the various discounts and fees applicable based on the ride	RentMaganer CarHandler DataProvider
G13 Allow a logged user to enable a money saving option which provides him with a charging station as destination of the ride to get a discount on the cost of the aforementioned ride	UserAppServer RentManager DataProvider

Table 1: Mapping goals on components

4.2 Non functional requirements

Performance requirements To guarantee a short response time we have tried to decouple components in order to enable an instance pooling management of components by the EJB container and so an as much as possible concurrent management of requests.
During all the design process we also have kept in mind the scalability of the software w.r.t. the number of cars trying to keep a linear complexity factor and to reduce car-dependent activities where it is possible.

Availability To ensure availability requirements server will be running 24 hours for day. The architecture is designed with the purpose of enabling a redundancy architecture if availability constraints would make it necessary.

Security Security protocols are used for transmission and storage of sensitive data. Maintenance API is only accessible through token mechanisms. Customer Care Server is accessed over a VPN to ensure security.

Portability Users access the system through a web-based application that enables the portability and a cross-operative system compatibility.

Appendices

A Software and tools used

For the development of this document we used

- LaTeX as document preparation system
- Git & GitHub as version control system
- Draw.io for graphs
- StarUML for sequence diagrams
- Balsamiq Mockups for user interface mockups

B Hours of work

This is the amount of time spent to redact this document:

- Davide Piantella: $\sim$ 48 hours
- Mario Scrocca: $\sim$ 52 hours
- Moreno R. Vendra: $\sim$ 51 hours

C Changelog

- **v1.0** December 11, 2016
- **v1.1** January 3, 2017
  - Add notes to the *User interface design* section clarifying reason of mockups design in relation with architectural design choices presented in the document
  - Add diagram and comments to the Car Communication Interface and the non represented interactions in the relative sequence diagrams.
  - Add an attribute in the DB to better specify how our system connects with cars (Car Server Interfaces)
- **v1.2** January 8, 2017
  - Add notes to clearify some sequence diagrams
  - Fix **??** attributes names about Failures
  - Add notes to *money saving option* mockup
- **v1.3** January 15, 2017
  - Update Map running view section, related mockups and GIS API specifications improving description about how the map is generated, referenced and shown to users

- **v1.4** January 22, 2017
  - Small changes to **??**
  - Fix missing link in **??**
  - Fix typos

References

[1] Bass, Clements, Kazman, *Software Architecture in Practice*, SEI Series in Software Engineering, Addison-Wesley, 2003

[2] D. Piantella, M. Scrocca, M.R. Vendra, *PowerEnJoy: Requirements Analysis and Specification Document*, Politecnico di Milano - Software Engineering II Project, 2016

[3] GPX open standard, The GPS Exchange Format, GPX website