Politecnico di Milano, A.Y. 2016/2017
M.Sc. Degree Programme in Computer Science and Engineering
Software Engineering 2 Project

Design Document

Philippe Scorsolini,
Lorenzo Semeria,
Gabriele Vanoni

4th January 2017

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to give the detailed structure of PowerEnJoy software system.
So we try to give developers a clear representation of:

- The high level architecture of the system.
- The design patterns applied in order to achieve our specific necessities.
- The main components and the interfaces they provide.
- The Runtime behaviour.

1.2 Scope

PowerEnJoy project aims to provide users and operators with means to use the system services they need and they are supposed to have access to. It also provides an API for external services to access the system with “operator” rights, in order to allow call center operators to interface with the system.
The system allows:

- Users to manage their personal data through both a web and a mobile app and Active Users to manage reservations.
- Callcenter’s operators to manage assistance tickets via the API. The tickets will then be managed internally by PowerEnJoy.
- PowerEnjoy’s operators to manage the open assistance tickets, take them in charge and update car’s and user’s data accordingly.

The system architecture shall guarantee future proof scalability and allow subsequent improvements and general reliability.

1.3 Abbreviations, Definitions and Acronyms

1.3.1 Abbreviations:

- Gn: the n-th Goal
- An: the n-th Assumption
- Rn: the n-th Requirement

1.3.2 Acronyms

- CC: Credit Card
- DL: Driving Licence
- AU: Active User

1.3.3 Definitions

- Visitor: person that may not be registered to the system or not logged in.
- User: a registered and logged in Visitor, that may be still waiting for his information to be verified.
- Active User: a User whose data (CC, DL) have been verified. (Shares all User's characteristics)
- Safe Zone: predefined zones where parking is allowed, parking is forbidden in any other zone.
- Park: park the car in the safe zone and terminate the rental.

2 Architectural Design

2.1 Design Process

In this subsection we'll try to give an overview of the process that brought to the final design chosen for the system, underlining the reasons that motivated our main choices in a mixed top-down/bottom-up approach. The more detailed description of the system will be given in the remainder of the document.
Initially, given the goals and the requirements we have depicted in the RASD, we have decided to adopt a three tier client-server architecture to be able to manage different clients and to decouple the three layers, this choice was trivial given the nature and the needs of our system, not so trivial was the choice of adopting a thin client in order to not have any performance constraint on the client's hardware to maximize the possibility of adoption by the user of our service.
Then we have started aggregating the different requirements in much granular components, trying to follow a single responsibility principle, as can be seen in section 5. So we have depicted the following components as part of the second tier on the server side of our system:

- Client Handler: orchestrates the needed services for the clients, translating client's requests to the server and returning the needed data.
- Authentication Manager: ensures that only authorised users are able to perform "restricted" actions as well as checking login credentials.
- Registration Manager: manages the registration of new users and the update of registered users' data if needed.
- Administration Module: grants PowerEnJoy's operators access to some specific features needed to manage the assistance tickets they have to take care of.
- Model Manager: grants access to the data to the other components, abstracting any kind of technology specific detail to the other services.
- Car Manager: manages the physical cars providing other services with the needed functionalities and informations.
- Power Station Manager: manages the physical power stations' data and functionalities.
- Assistance Ticket Manager: grants the assistance callcenter, the operators and the other services the APIs to manage the assistance tickets.
- Reservation Manager: manages the whole reservation process for an Active User from the reservation intention to the effective payment, interacting with the external payment handler through his APIs.

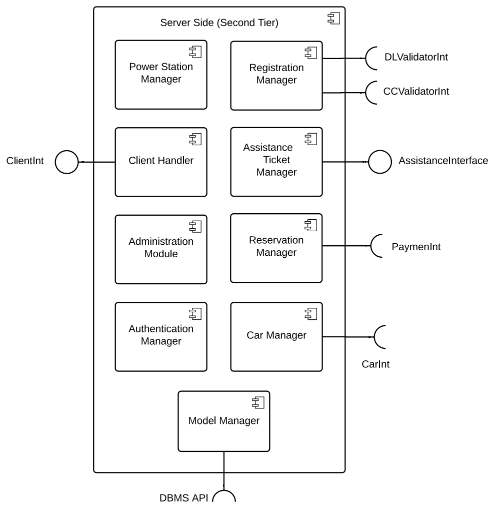


Figure 1: Component view derived from the Requirements analysis

Then we needed to find where our components could be deployed and how we would have managed the communication between them, we could have adopted a monolithic approach or take advantage of the well established application server architecture, but that wouldn't have granted the scalability and the independent deployment of the components we were looking for, so we decided to go for a microservice oriented architecture. This choice allowed us to take advantage of the growing number of Platform-as-a-Service providers, so it seemed natural given the previous choices to deploy our whole server side in the cloud, giving us the ability to scale our application on need and abstracting us from any hardware specific consideration in the future.
To fully take advantage of this architectural style we have introduced some components that will manage the life cycle of the deployed components:

- Discovery Service: gives to other services the references that allow them to communicate with each other.
- Binding Service: manages the queue binding for the services.i

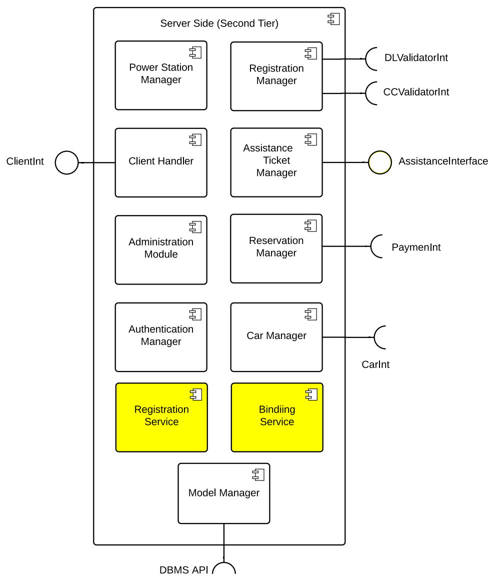


Figure 2: Component view derived from the Requirements analysis and the decision of microservice oriented architecture

Although we had decided to transform our three-tiered application in a three-tiered cloud application, the problem of the communication between our microservices still needed a solution, two options arose as the most valuable of attention: a point-to-point communication approach or a more centralized messaging/bus oriented one. These two options where not linked to the choice of a cloud application, it could have also be adopted in a server. The obvious advantages of the first one with respect to the second where the absence of a clear single point of failure and with that the absence of a possible bottleneck in the central message bus, but the disadvantages such as the difficulty of managing the whole system without a central coordinator and the numerous APIs that such approach would have needed, but given the previewed load of our system the advantage didn't seem really much of an advantage and the disadvantage would have burdened the development of the system.
Checked some numbers of the actual load the of the third party message broker could handle and the advantages of adopting one of them, we have decided to adopt a centralized message broker that with a publish subscribe pattern would manage the communication between the components giving us as secondary effect the possibility to implement some advanced load balancing techniques in the near future to better take advantage of the pay-per-use approach the cloud provides.

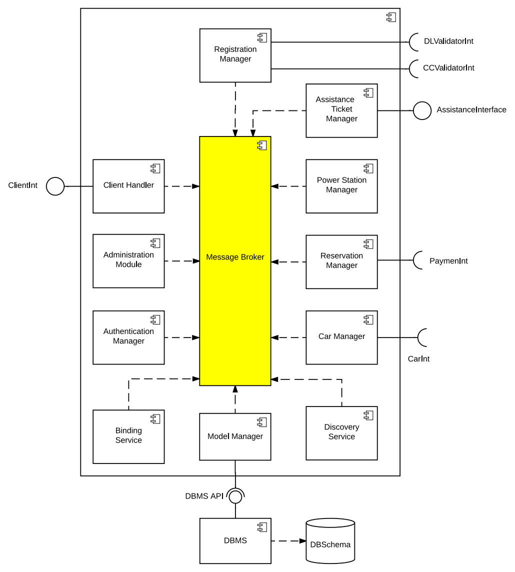


Figure 3: Component view derived from the message broker decision

After deciding the structure of the server side, we have focused on the remainder part of the deployment diagram and depicted the counterpart of the previous components outside the server, such as the software that will be deployed inside the cars and the one for the powerstations and the two clients' software accordingly to the RASD specifications.
Then we've tried to give a possible, but not mandatory, implementation of the system, especially concerning the communication protocols that could be used between the different parts of the system and the technologies that could be used to implement them, in order to check the feasibility of our choices. Java Enterprise Edition as implementation language seemed the most adopted solution for this kind of systems and microservice-oriented framework on top of JEE such as Spring Boot could ease the development fitting most of our needs out of the box, even if our design choice could give the development team of each service the freedom to choose the most appropriate framework/language as long as the communication protocol chosen is supported.

2.2 Overview

The system adopts a three tier architecture with a thin client represented by web and mobile app, which allows users and operators to access through a GUI the different functionalities of the system accordingly to the type of client used. These clients are both managed by a specific server-side ClientHandler offering the same interface to other server-side services and handling appropriately the communication with the two clients adopting in both cases an asynchronous implementation of the RESTful APIs over HTTPS in JSON format in order to achieve complete freedom of development-specific choices on both sides.
The second tier adopts a microservice oriented architecture with shared database that, taking into account the unknown but supposedly not massive load of the system in the near future, reduces the need of synchronization between services. This allows to keep the structure as simple as possible, allowing better performing solutions such as "database per service" or "schema per service" to be implemented when needed. This tier will also

manage the communication with the cars and the power station around the city through OpenVPN and MQTT protocols.
The third tier provides the previous one the necessary abstraction on the storage technology chosen and will manage the concurrent access to the database. In case a PaaS hosting service is chosen, this tier will be managed by the provider.
The second and third tier are designed to be deployed on the cloud in order to take advantage of the “scale on need” possibility it gives. For this reason it has been designed as a micro service architecture.

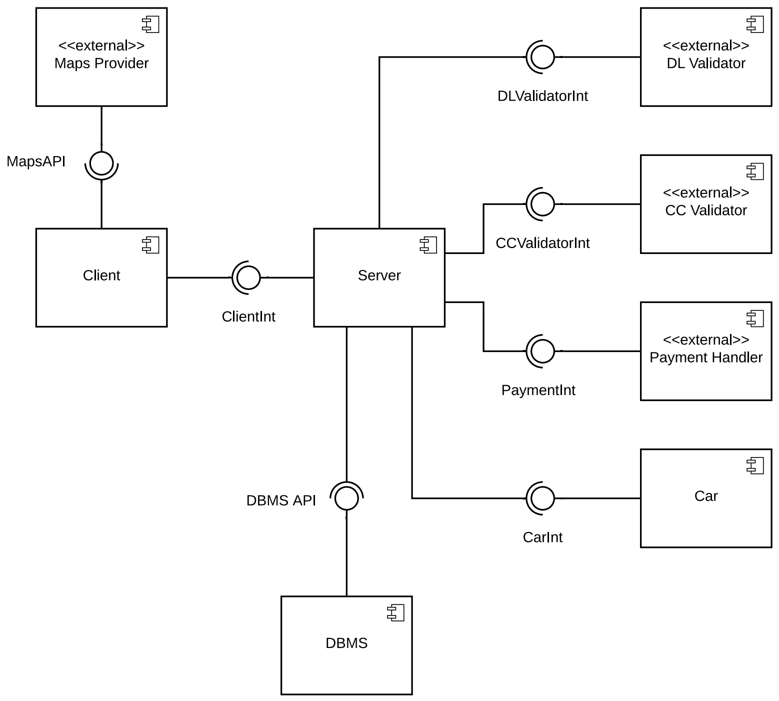


Figure 4: Component View overview

2.3 Component View

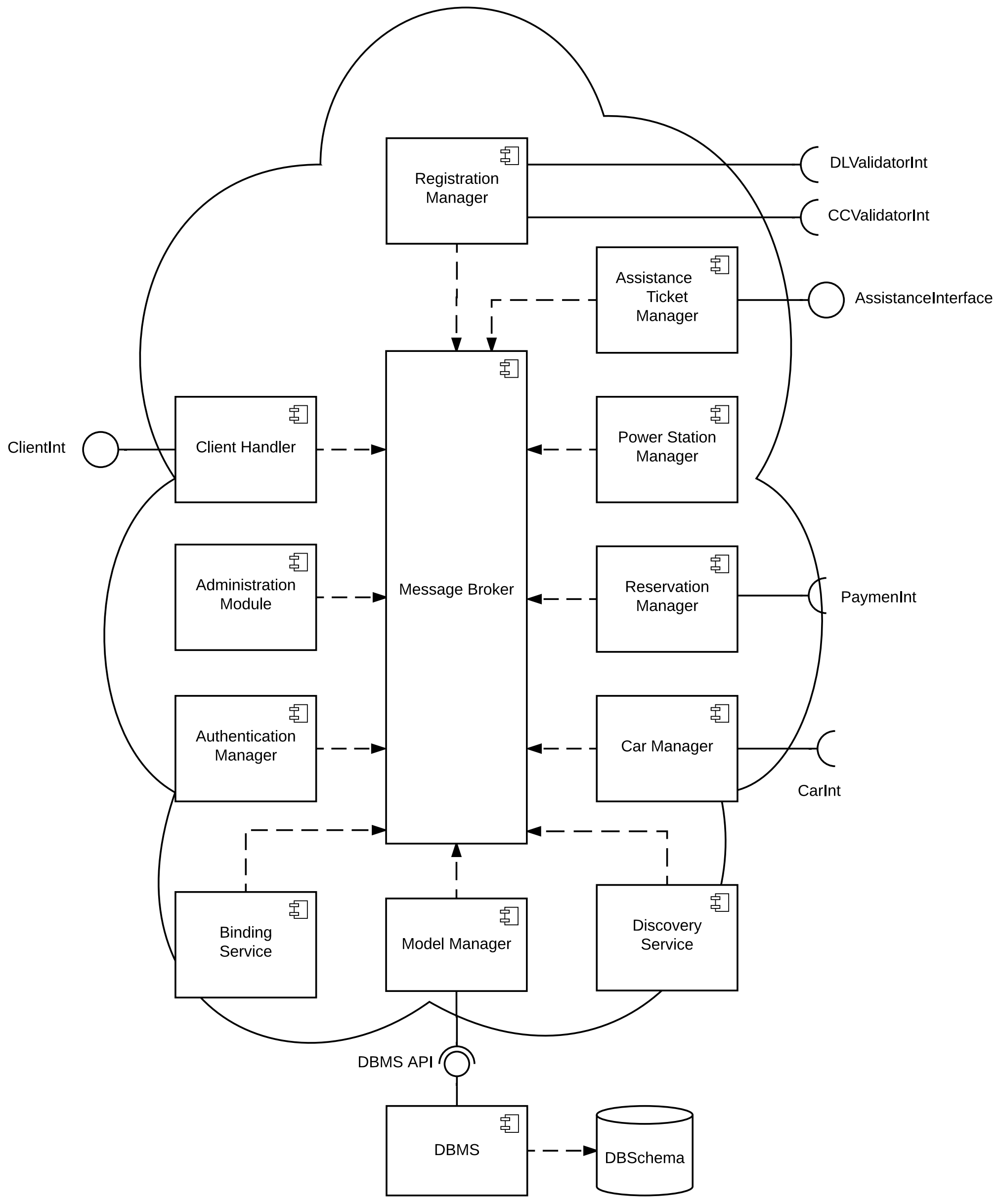


Figure 5: Component View of the server side single services

As previously said the system has been designed in order to respect the principles of microservice architecture. Therefore every component will be stateless, independently deployable, able to deal with a specific aspect of our business domain and to hide to the other services any kind of implementation detail.
The communication between these microservices will use AMQP (Advanced Messaging Queue Protocol) and will be managed by a centralized message broker (such as RabbitMQ or ActiveMQ). When a new instance of a service is created the message broker will be automatically connected to the new service and will publish its existence to the other services through the “discovery service”, then the “binding service” will take charge of setting up the correct queue binding rules with the message broker. That will allow

some advanced traffic routing features including per-version weighting and elastic load balancing on the different instances of the same service.
The hereunder specified components could be in the future divided in much more granular services in order to decouple even more their functionalities:

- Client Handler: orchestrates the needed services for the clients, translating REST HTTP requests into AMQP-compliant messages and sending them to the different microservices as needed.
- Discovery Service: gives to other services the references that allow them to communicate with each other.
- Binding Service: manages the queue binding for the services.
- Authentication Manager: ensures that only authorised users are able to perform "restricted" actions as well as checking login credentials.
- Registration Manager: manages the registration of new users and the update of registered users' data if needed.
- Administration Module: grants PowerEnJoy's operators access to some specific features needed to manage the assistance tickets they have to take care of.
- Model Manager: grants access to the data to the other components, abstracting any kind of technology specific detail to the other services.
- Car Manager: manages the physical cars providing other services with the needed functionalities and informations.
- Power Station Manager: manages the physical power stations' data and functionalities.
- Assistance Ticket Manager: grants the assistance callcenter, the operators and the other services the APIs to manage the assistance tickets.
- Reservation Manager: manages the whole reservation process for an Active User from the reservation intention to the effective payment, interacting with the external payment handler through his APIs.

All the communications between the different components will be asynchronous in order to minimise unneeded resource consumption that would result in an increase of the costs for the commissioner.
Each of these services can be instantiated and deallocated as many times as needed to handle the momentary load of the system, taking advantage of the queuing system offered by the message broker. This, coupled with a dynamic cloud infrastructure that allows quasi-immediate upscale of computing and networking capability, will allow the system to handle load spikes along with mitigating the risk of a downtime due to the excessive load.

2.4 Deployment View

To reach our goals we have depicted 4 main components to be deployed separately from the numerous services we previously indicated in section 2.2:

- Onboard Car Management system: it allows the server and the car to communicate, and more specifically makes it possible for the server to retrieve important data from the car and to remotely access car functions such as locking and unlocking the doors.
- PowerStation Data System: it will be deployed in each Power Station and will manage the communication with the central system ensuring that every service is able to access relevant data.
- Mobile App: it will give both operators and Active Users access to the specific features they have permission to access, only after they sign in. There are many possible implementations for the app, while the communication with the Client Handler (on the server) will use RESTful APIs. Maps will be downloaded from an external probider in order to have a user-friendly visualization of cars' positions.
- Web Page: it will allow Visitors to check the map only, while letting Users reserve cars and change their account's data.

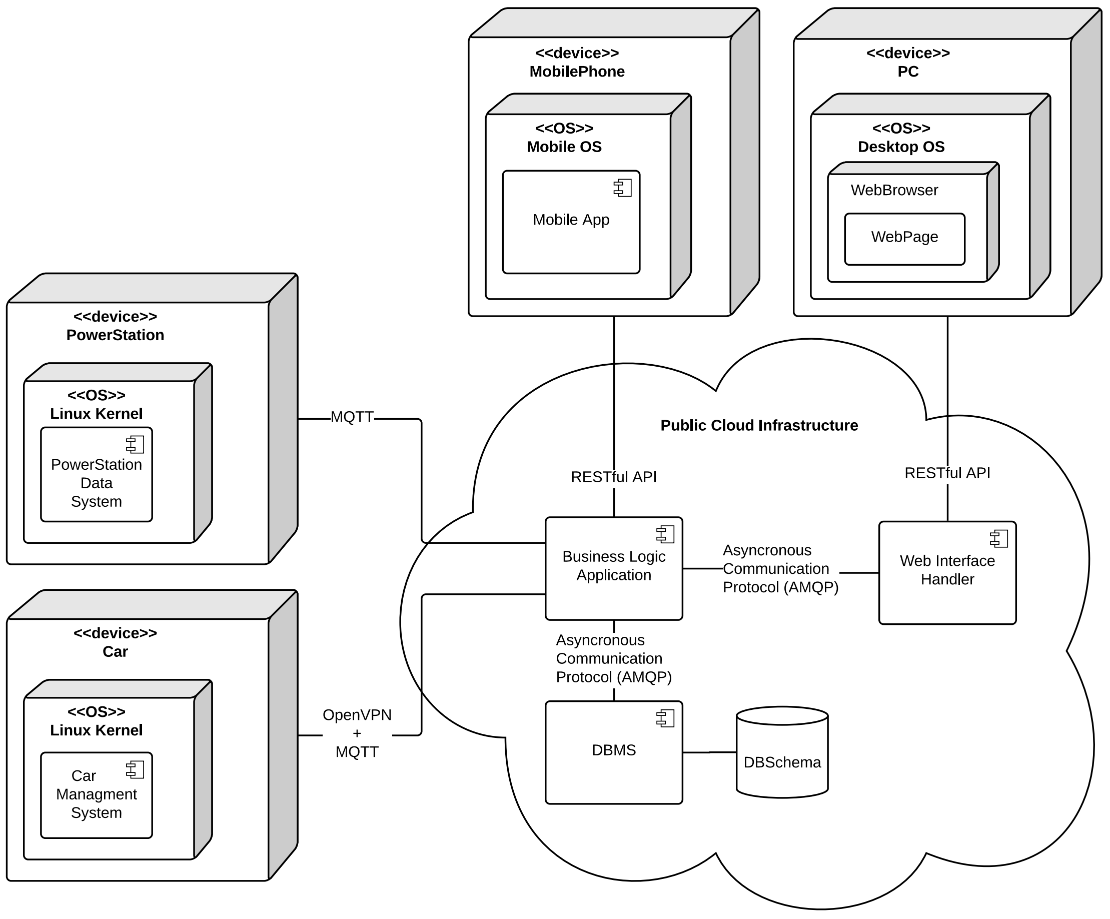
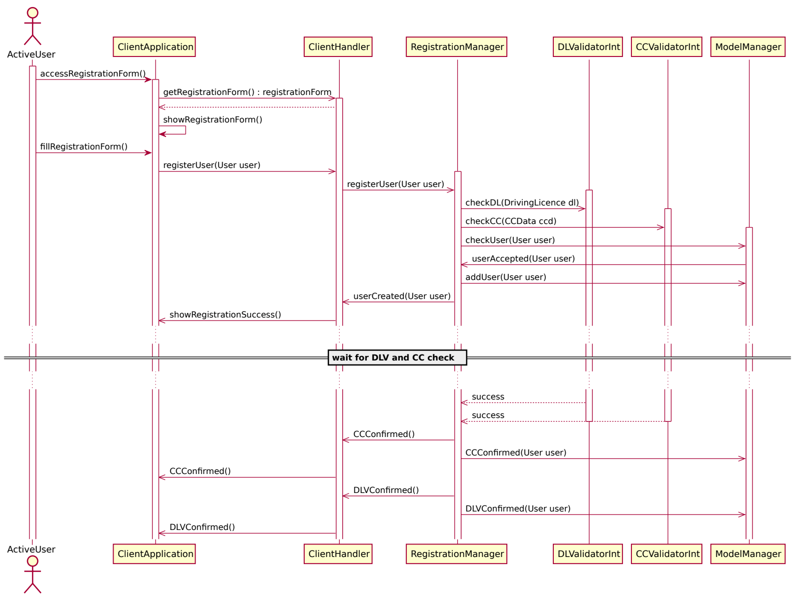


Figure 6: Deployment View of the system

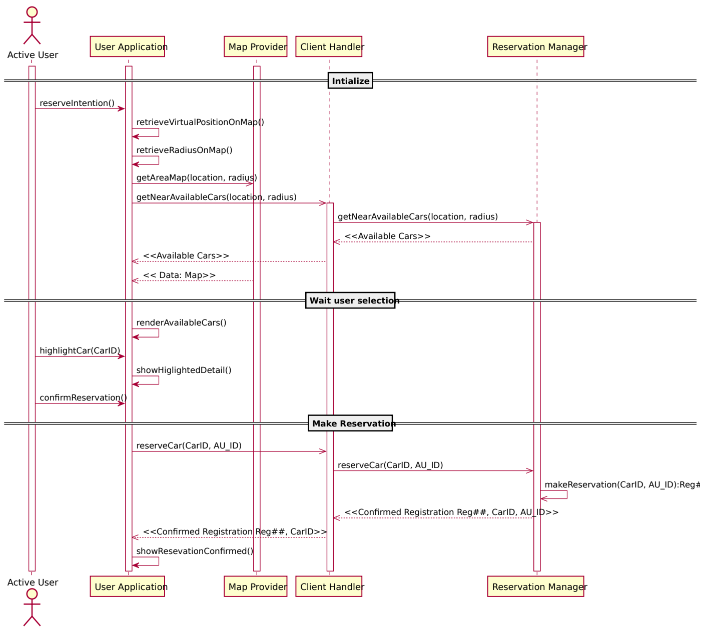
2.5 Runtime View

The following sequence diagrams will only show the way the system will act while performing certain tasks. They are not meant to be an exact representation of the software implementation of the procedures. For sake of simplicity we have preferred to show the intention of the calls to other services, omitting all the technical details concerning the mere technological choice of a specific message broker.

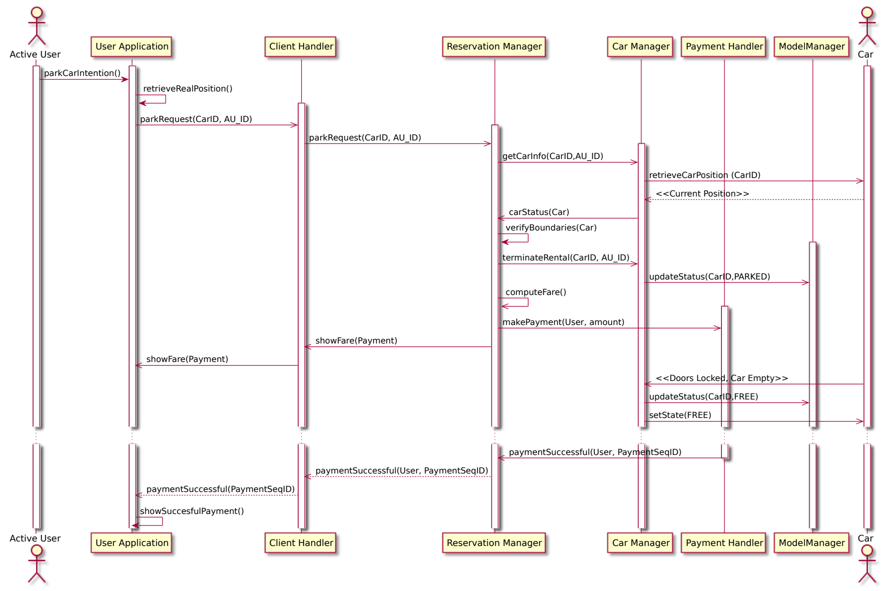
Sign up sequence diagram:



Car reservation sequence diagram:



End of a car rental:



2.6 Component Interfaces

Seen that we have adopted a message based communication between our services, the only type of interface with the outside world they need are the ones defined by the message broker in order to be able to receive and pull messages from the queues.
The client Handler will offer the necessary REST APIs to the clients.

2.7 Selected Architectural Styles And Patterns

As previously stated our system will adopt a 3-tier, cloud-based architecture. This will allow to decouple the presentation logic from the business logic as well as to have a shared database management that will be provided by a PaaS and will be accessed through a single model manager service, allowing the other services to be stateless.
The overall serverside system will be composed of microservices. The webserver will take advantage of a façade pattern offered by the ClientHandler that will work as an API gateway for both the web server and the client (either a mobile application or a web page). This pattern choice will allow access to our application without too strict hardware constraints on user's devices. If needed the system could be made elastic by adding a service as an "elastic component" managing the other services instantiation. The elastic load balancing will be managed by the API gateway and the message broker in order to minimize the use of physical resources and costs.The system will apply a server-side discovery pattern thanks to the discovery and binding services that will grant the necessary connection between the various components in a sort of publish-subscribe pattern where the services are subscribed by the binding service to the appropriate messaging queue and can publish to other. The communication between server and client will adopt a REST architectural style that will allow the decoupling between the two sides and will give them a common language that allows the developers to choose the most suitable technology on both sides.

2.8 Data management

Although we have used a microservice approach for the business logic tier, it doesn't make much sense to split data between the different modules. In fact our data model is very small and interconnected and we don't need to use different approaches (eg. SQL and noSQL at the same time). Of course in this way we couple the different modules but it is inevitable since they run for the same macro-functionality. A unique database grants us the possibility to ensure in a simple way ACID properties and to use the standardized SQL language for queries. We provide an Entity-Relationship model for our application in figure 4.

2.9 Other Design Decisions

Having taken into account the decisions explained in the previous sections we decided to implement the overall logic of the system in Java. We will use Java EE with Frameworks such as Spring Boot as well as open source services such as RabbitMQ as Message Broker and other to cover the discovery and binding services, the ClientHandler and the Authentication Manager. The web server part could be managed by Nginx in order to reach the needed scalability and reliability and manage the web interface for the clients.

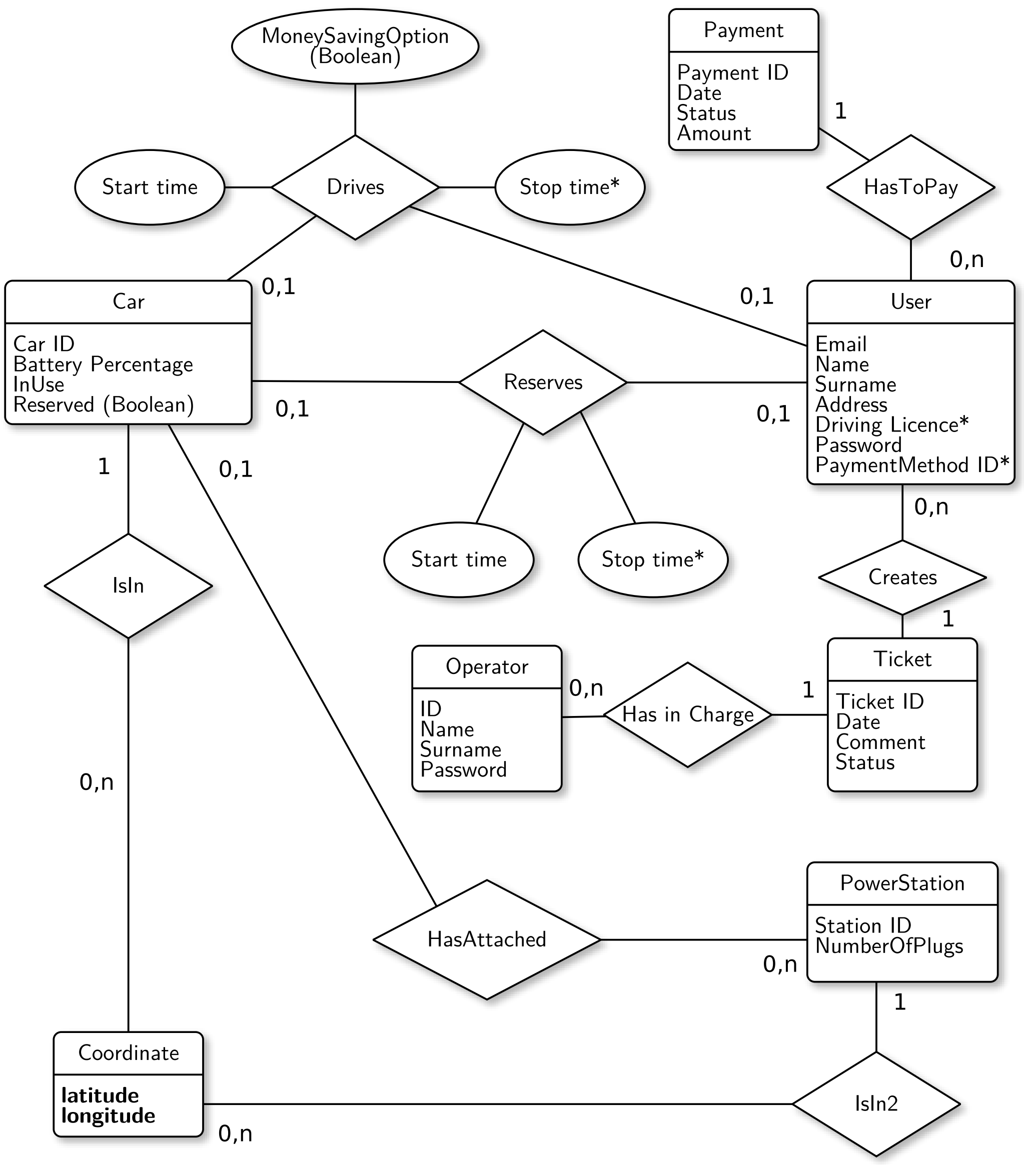


Figure 7: ER data model

3 Algorithm Design

All algorithms needed in the project are trivial but the one dealing with uniform repartition of cars in the city.
Cars are picked up by by users in one location dropped off in another one. Of course the distribution of picking up and dropping off locations is not uniform. So some operators are needed in order to perform some relocations, in particular during the night, so that in the morning cars are located where there is actual necessity. The money saving option tries to drop the load of relocations, stimulating users to park where there is a deficiency, with a discount. So the global situation have to be monitored in real time, taking into account, the static situation, the reservations and and the current rides.
This problem has been studied a lot and there are in literature various algorithms that solve it. They are mainly based on mixed integer linear programming techniques and in particular [1] presented a complete model. In [2] is presented a greedy algorithm that achieves almost the same result. In [3] a more sofisticated approach is used taking into account a three dimensional objective function and exploiting genetic algorithms and local search methods. [4] offers a sort of classification of the strategies proposed in the past years.
For our purpose the approach described in [2] is the best since it minimizes the number of the operators needed to relocate cars and so the costs.

Riferimenti

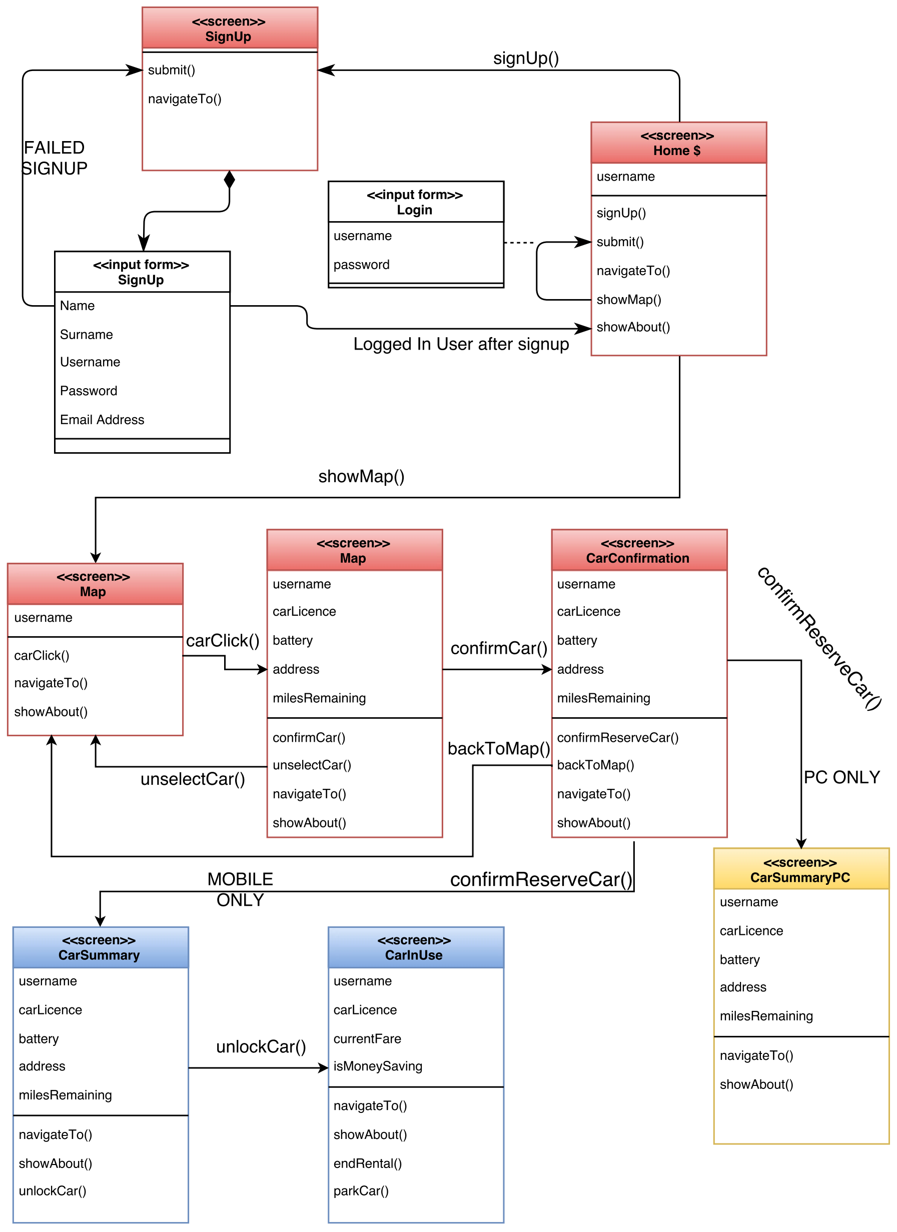
[1] A. G. Kek, R. L. Cheu, Q. Meng, and C. H. Fung, “A decision support system for vehicle relocation operations in carsharing systems”, Transportation Research Part E: Logistics and Transportation Review, vol. 45, no. 1, pp. 149–158, 2009.
[2] R. Zakaria, L. Moalic, A. Caminada, M. Dib, “A Greedy Algorithm for relocation problem in one-way carsharing”, 10th International Conference on Modeling, Optimization and Simulation - MOSIM’14 – November 5-7-2014- Nancy – France “Toward circular Economy”.
[3] Moalic, L., Lamrous, S., & Caminada, A. (2013). A Multiobjective Memetic Algorithm for Solving the Carsharing Problem. Proceedings Of The 2013 International Conference On Artificial IntelligenceIcai 2013, Vol. 1, pp. 877-883.
[4] S. Weikl, K. Bogenberger, “Relocation Strategies and Algorithms for free-floating Car Sharing Systems”, 15th International IEEE Conference on Intelligent Transportation Systems Anchorage, Alaska, USA, September 16-19, 2012.

4 User Interface Design

Below are some mockups to show how users will interact with the service. Since PowerEnJoy can be used both from a computer (except for unlocking the car), both Mobile and Web mockups are provided. Moreover, since both Users and Operators have access to the service via browser and app, interfaces for both types of users have been created.

4.1 User Experience Diagram

This diagram briefly sums up the core interactions that a User can have with the system, highlighting how each "Page" (called Screen) is accessible in a normal browsing behaviour. A Screen represents both a Web Page and a Mobile App page since the interactions themselves do not change.

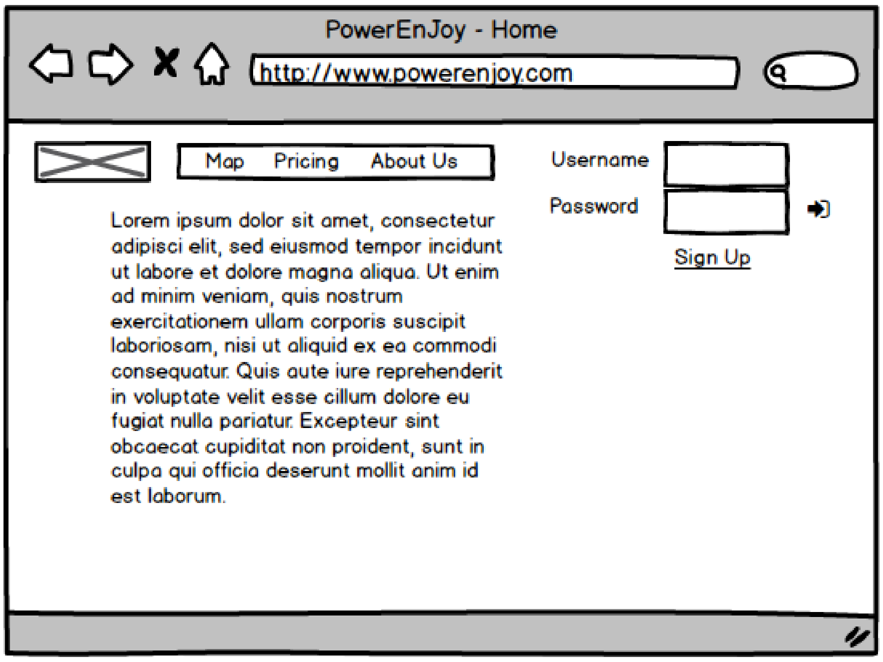


4.2 User Interfaces

As anticipated, in this section all User mockups are analyzed. These mockups show how all actions that can be performed by our users. This section is further split between Web interfaces, imagined for standard browsers, and Mobile interfaces, designed having a smartphone App in mind.

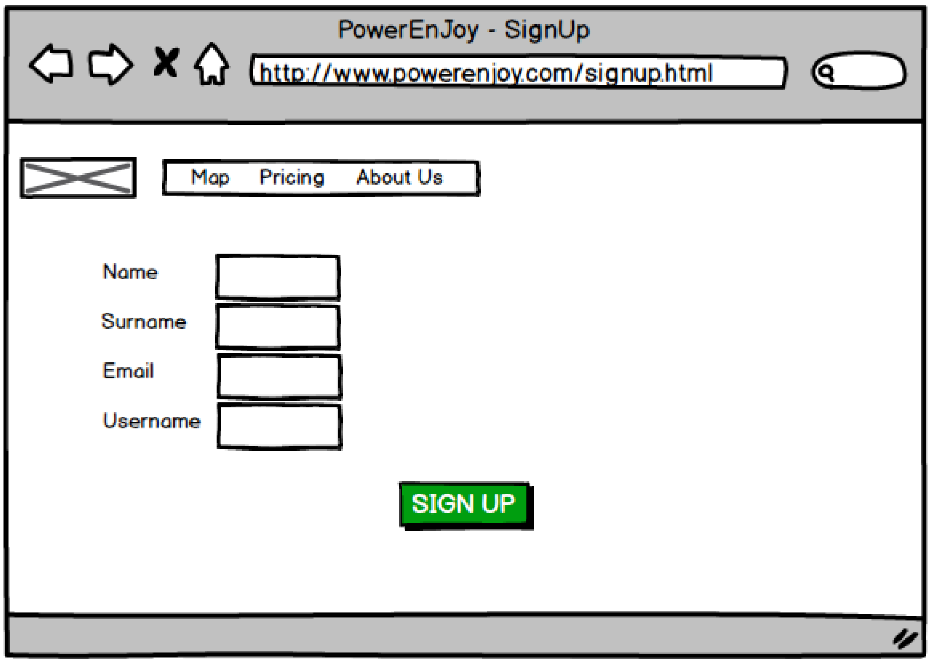
4.2.1 Web Interfaces

4.2.1.1 Home Page (Web) From the home page any user can try to login inserting username and password or they can choose to sign up and go to the registration page. This page will likely show a description of the service as well as providing links to other important part of the website (Map, Pricing, About Us).

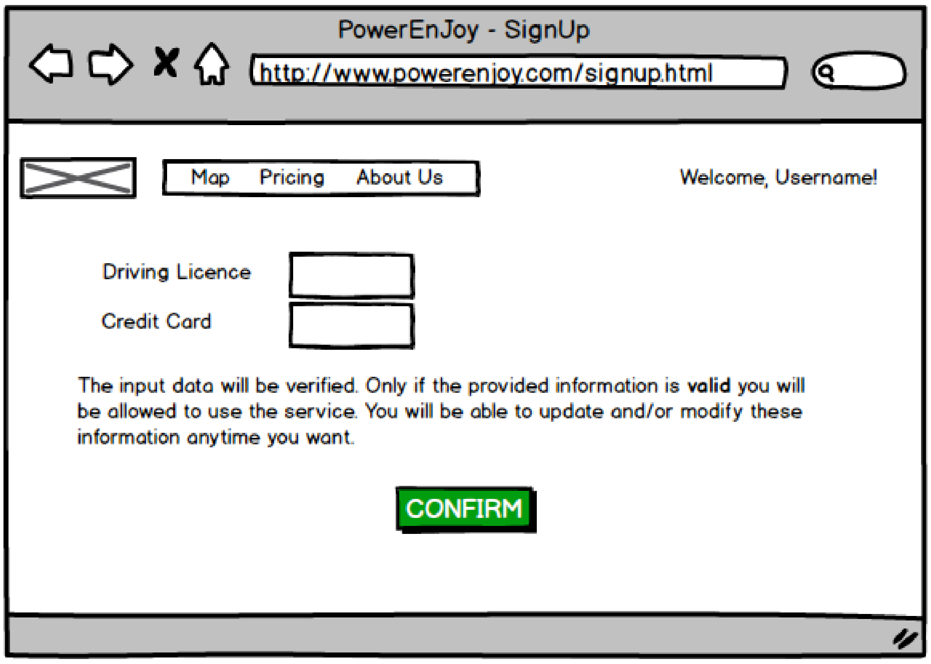


Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

4.2.1.2 Registration Page (Web) In this page users must input the core informations to register online: name, surname, email address and username.

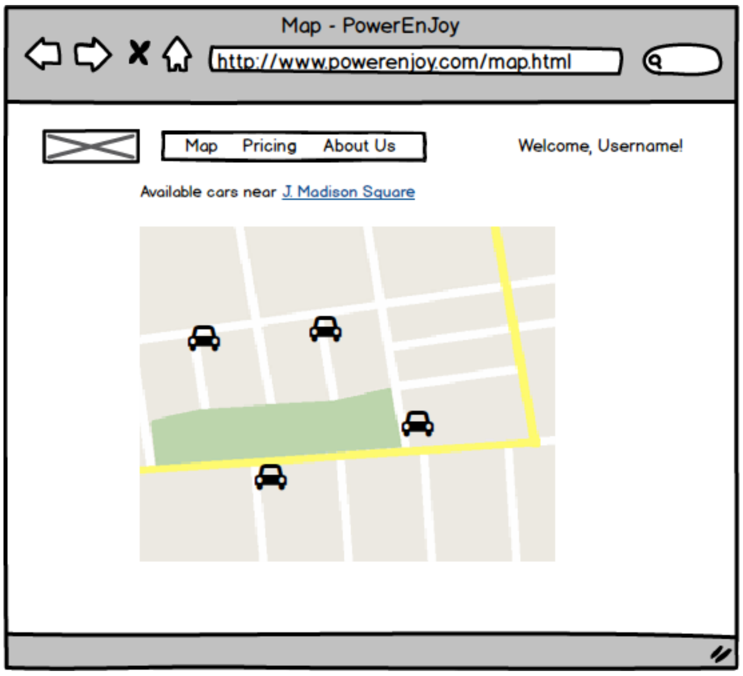


4.2.1.3 Further Information (Web) After having registered and logged in, users must input their Licence and Credit Card in order to use the service if they haven't already.

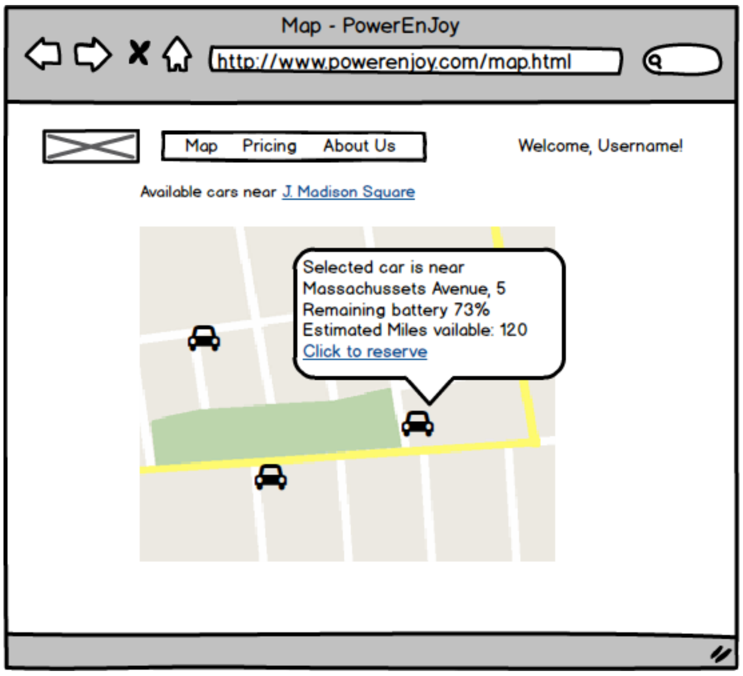


The input data will be verified. Only if the provided information is **valid** you will be allowed to use the service. You will be able to update and/or modify these information anytime you want.

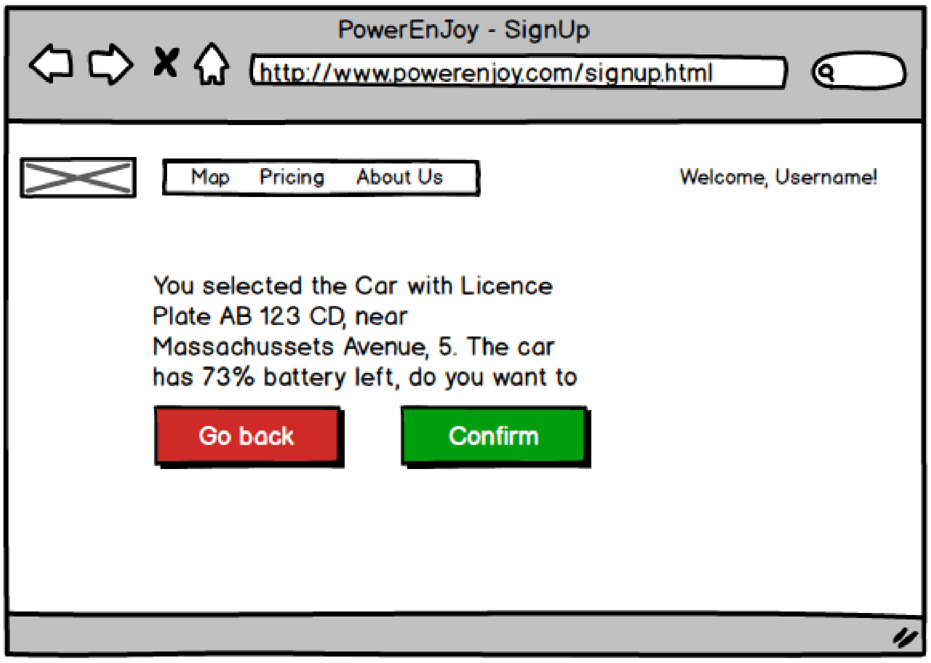
4.2.1.4 Map (Web) All users can view available cars near their position and choose one if they want more info (see next mockup).


Map - PowerEnJoy
http://www.powerenjoy.com/map.html
Map
Pricing
About Us
Welcome, Username!
Available cars near J. Madison Square

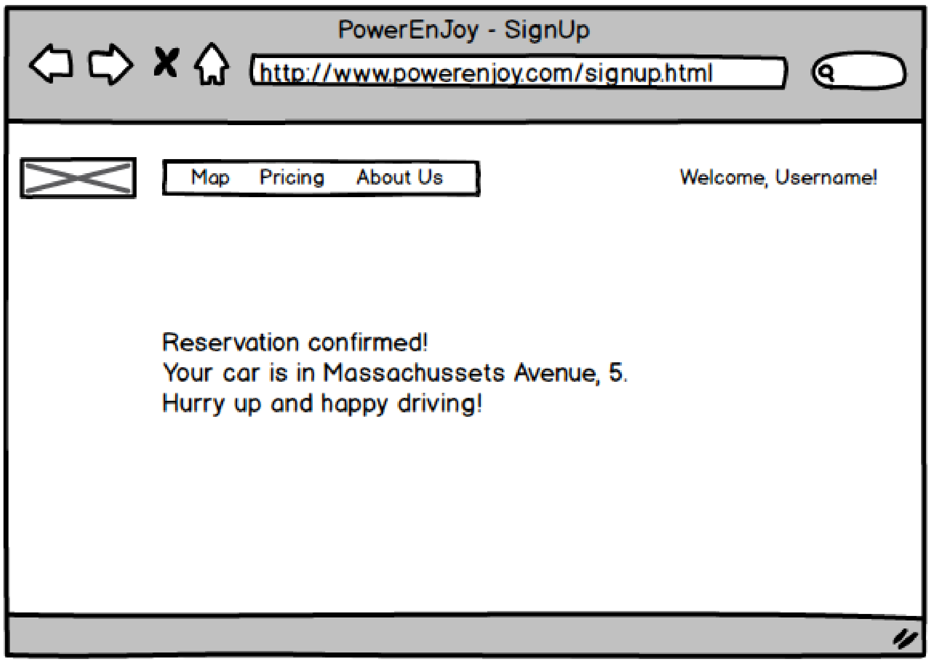
4.2.1.5 Map - Selected Car (Web) Selecting a car provides relevant information about that specific car, in order to give to the user the chance to pick a car that can suit his needs.



4.2.1.6 Selected Car Confirmation (Web) After having selected a car and having pressed "Click to reserve", users are asked to confirm their choice one last time, to avoid errors in case of mistakenly pressed links or misread information.


You selected the Car with Licence Plate AB 123 CD, near Massachussets Avenue, 5. The car has 73% battery left, do you want to

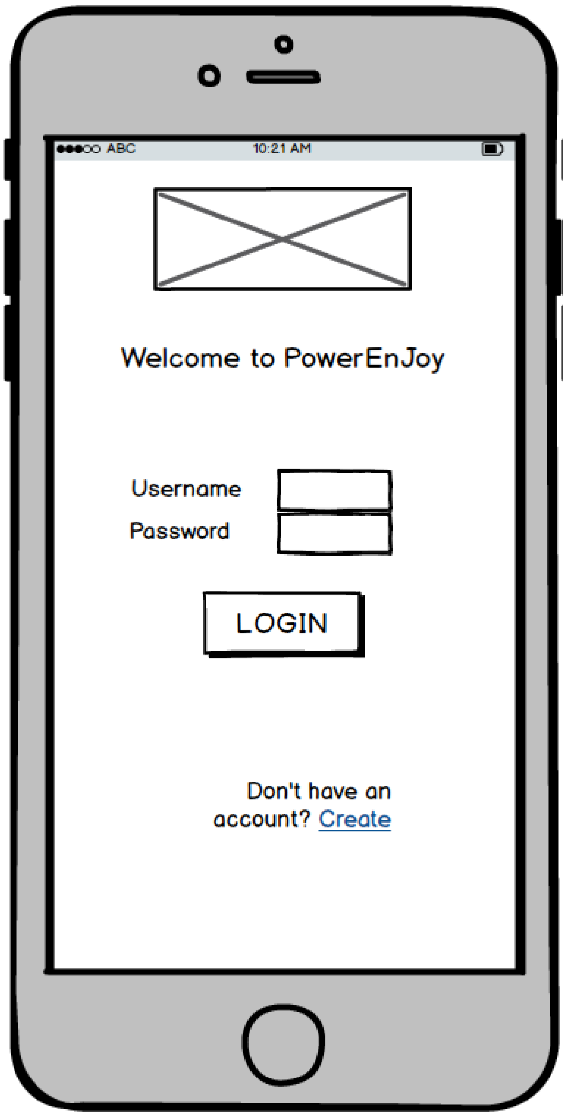
4.2.1.7 Reservation Confirmed (Web) Confirming a reservation shows a brief summary containing the address at which the car is parked.

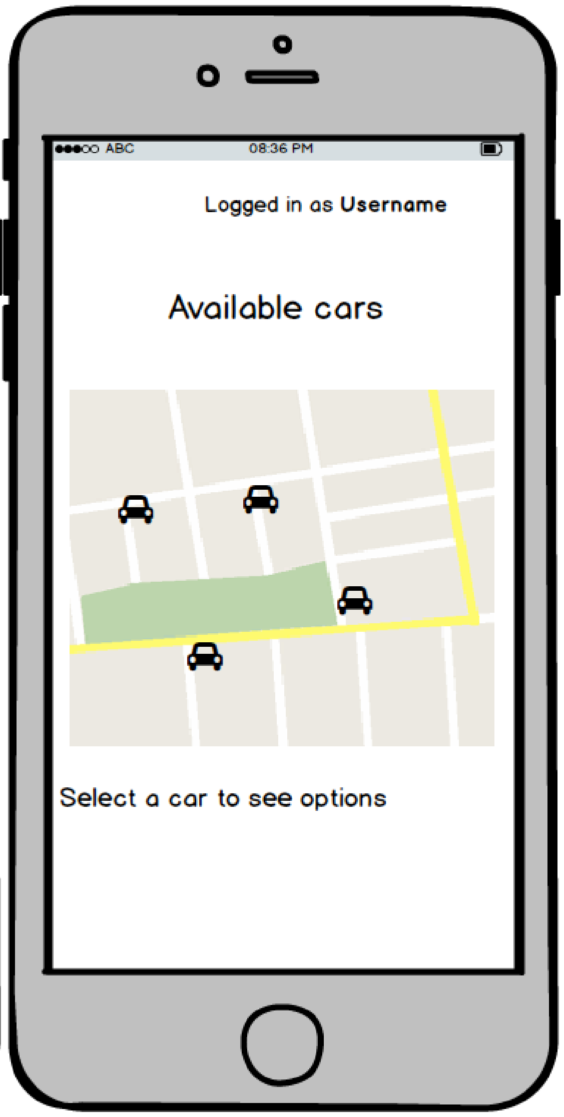

Reservation confirmed!
Your car is in Massachussets Avenue, 5.
Hurry up and happy driving!

4.2.2 Mobile Interfaces

4.2.2.1 Home (Mobile) From the App's home page users can either login or create an account. Login is handled inside the app, while to create an account the user is redirected to the website. (On the left)

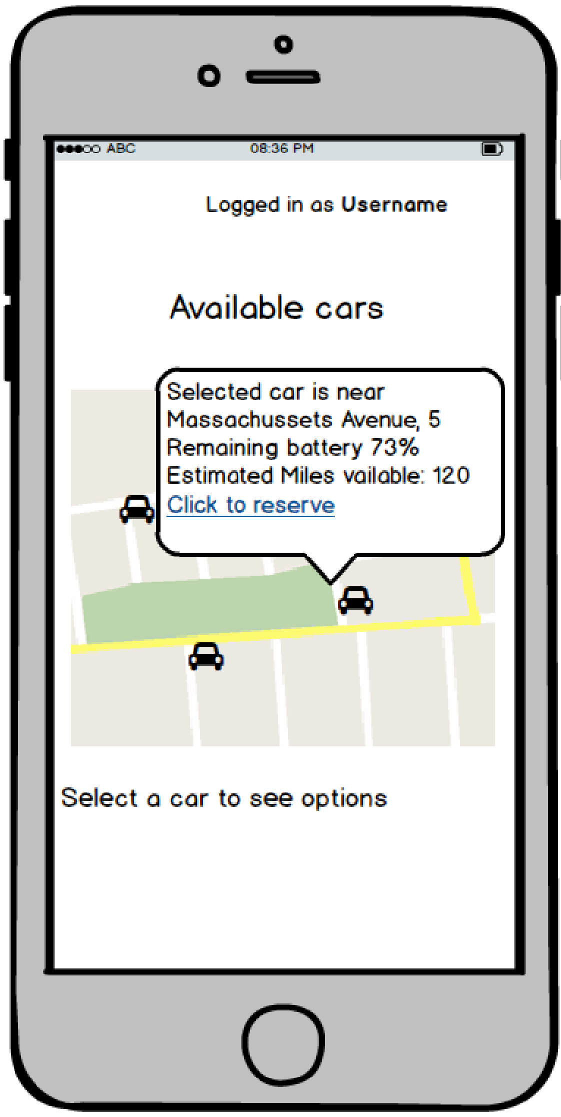
4.2.2.2 Map (Mobile) The user is shown a map displaying all cars that are nearby. His position can be calculated using the built in GPS receiver or can be manually input. (On the right)





4.2.2.3 Selected Car (Mobile) Selecting a car on the map shows relevant information, like it happens on the website. (On the left)

4.2.2.4 Car Confirmation (Mobile) A user can confirm a reservation or go back if he chose that car by mistake. (On the right)

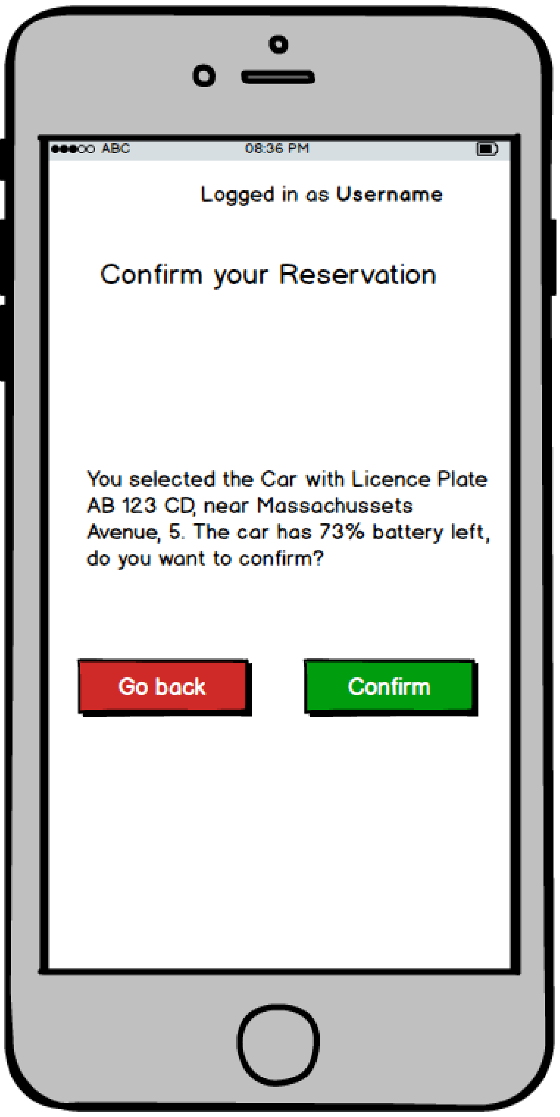


Logged in as **Username**

Available cars

Selected car is near Massachussets Avenue, 5
Remaining battery 73%
Estimated Miles vailable: 120
Click to reserve

Select a car to see options



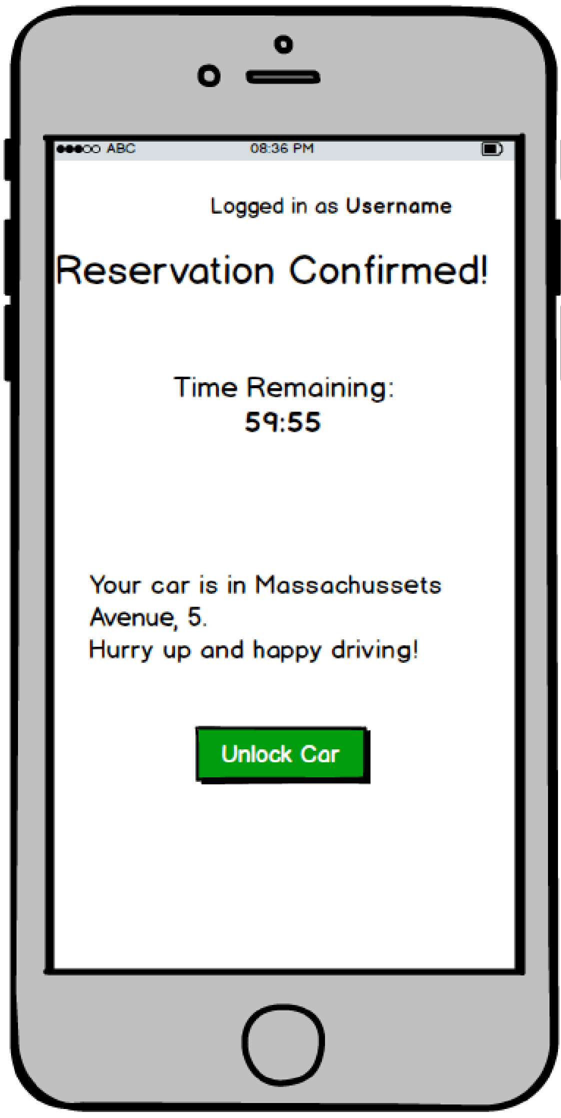
Logged in as **Username**

Confirm your Reservation

You selected the Car with Licence Plate AB 123 CD, near Massachussets Avenue, 5. The car has 73% battery left, do you want to confirm?

4.2.2.5 Reservation Confirmed (Mobile) When the reservation is confirmed, the smartphone shows a countdown as well as the car's address. (On the left)

4.2.2.6 Car In Use (Mobile) While using the car, Users can decide to park it — which signals the system that the car will be picked up by the same user and that the rental is not to be terminated — or to end the rental. (On the right)

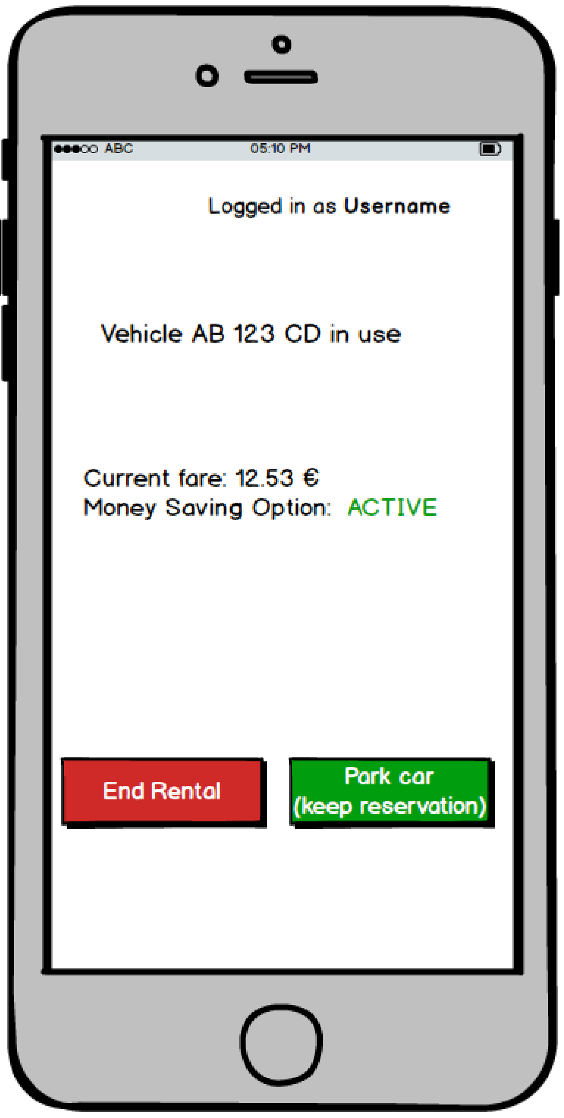


Logged in as **Username**

Reservation Confirmed!

Time Remaining:
59:55

Your car is in Massachussets Avenue, 5.
Hurry up and happy driving!



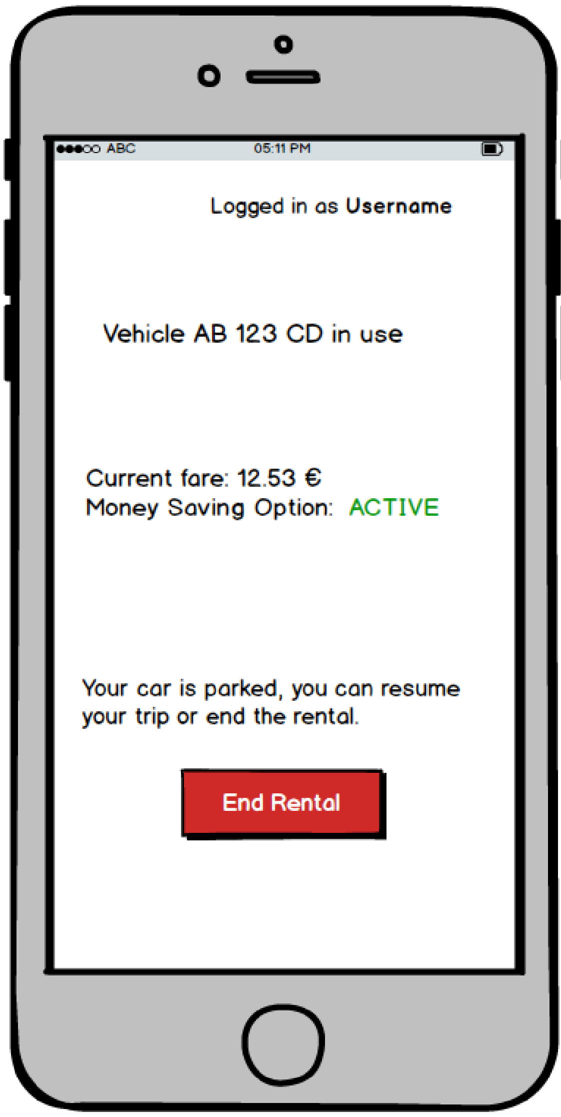
Logged in as **Username**

Vehicle AB 123 CD in use

Current fare: 12.53 €
Money Saving Option: ACTIVE

4.2.2.7 Car Parked (Mobile) If the car is already parked users can decide to end the rental using the app. In case they want to continue their journey, they only have to jump back on the car. (On the left)

4.2.2.8 Rental Ended (Mobile) Ending a rental shows the total as well as whether the Money Saving Option was active for the trip. (On the right)

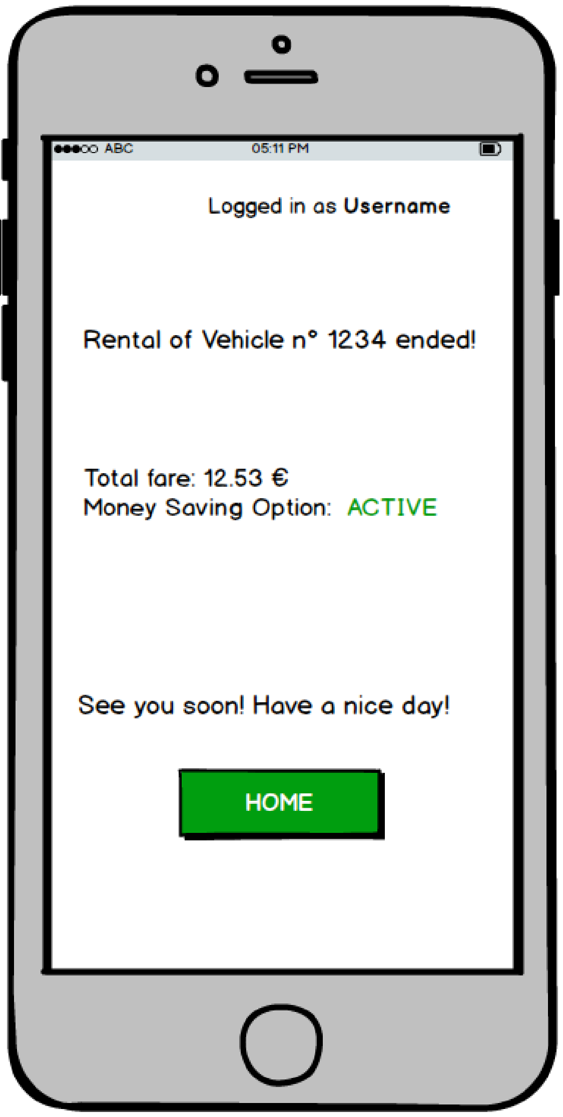


Logged in as **Username**

Vehicle AB 123 CD in use

Current fare: 12.53 €
Money Saving Option: ACTIVE

Your car is parked, you can resume your trip or end the rental.



Logged in as **Username**

Rental of Vehicle n° 1234 ended!

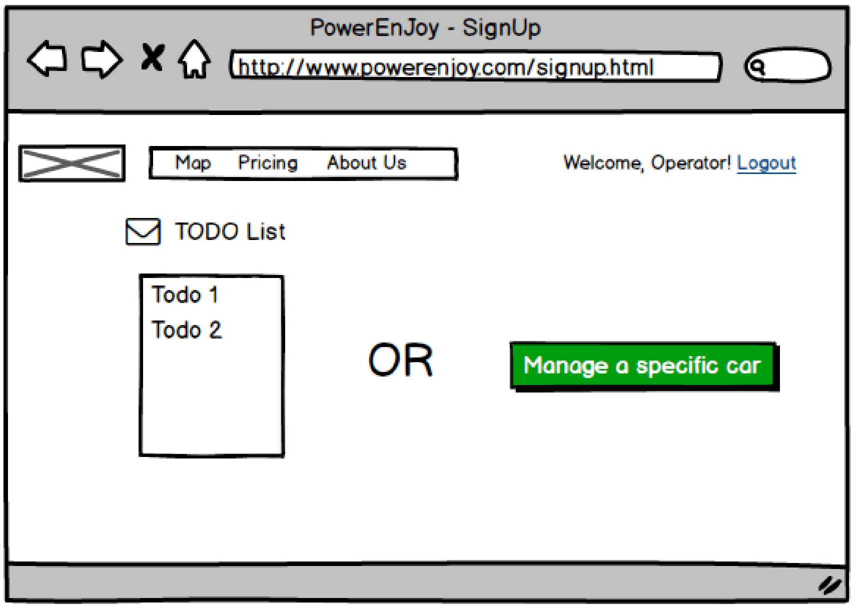
Total fare: 12.53 €
Money Saving Option: ACTIVE

See you soon! Have a nice day!

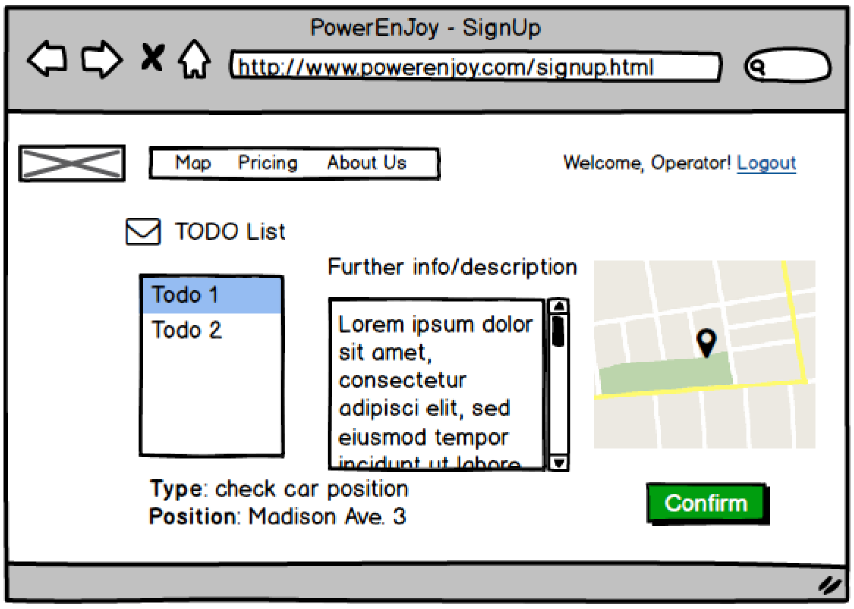
4.3 Operator Interfaces

4.3.1 Web Interfaces

4.3.1.1 Operator Main Page (Web) Operators can choose between performing a pending task (a "Todo") or managing a specific car.

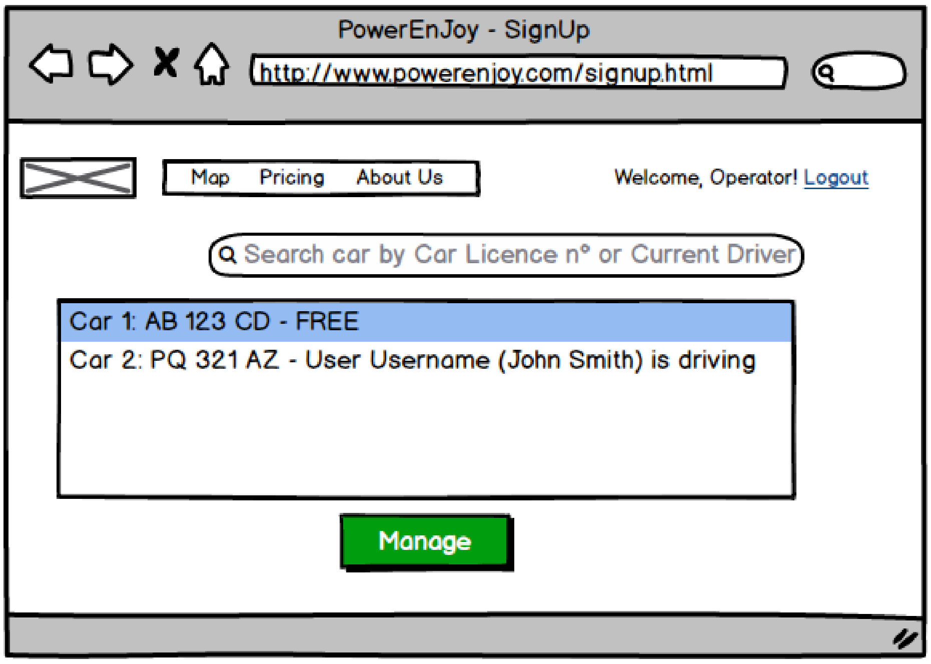


4.3.1.2 Operator chose TODO (Web) Choosing a pending operation brings up relevant information about it: its type, the position (if relevant), a short summary and a map. The operator can confirm the task if he will take care of it or go back to the home page(the logo links to the home page).


PowerEnJoy - SignUp
http://www.powerenjoy.com/signup.html
Map
Pricing
About Us
Welcome, Operator! Logout
TODO List
Todo 1
Todo 2
Further info/description
Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore
Type: check car position
Position: Madison Ave. 3
Confirm

4.3.1.3 Operator Searched Car (Web)

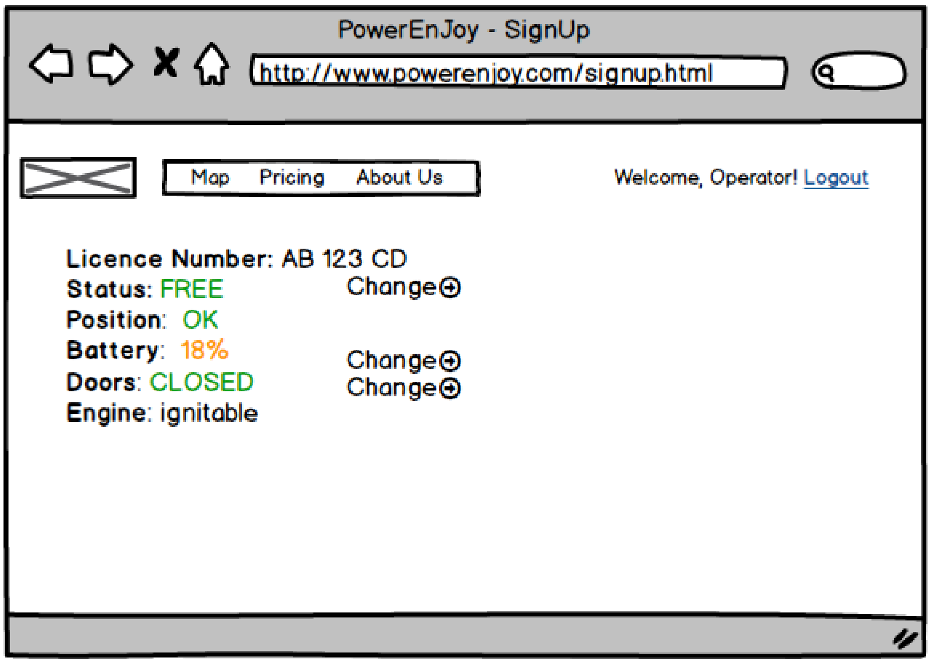
Choosing to manage a specific car allows the operator to search among all cars, by car Licence Plate or by Current Driver (User) if in use.


PowerEnJoy - SignUp
http://www.powerenjoy.com/signup.html
Map
Pricing
About Us
Welcome, Operator! Logout
Search car by Car Licence n° or Current Driver
Manage

Car 1: AB 123 CD - FREE
Car 2: PQ 321 AZ - User Username (John Smith) is driving

4.3.1.4 Car Details (Web)

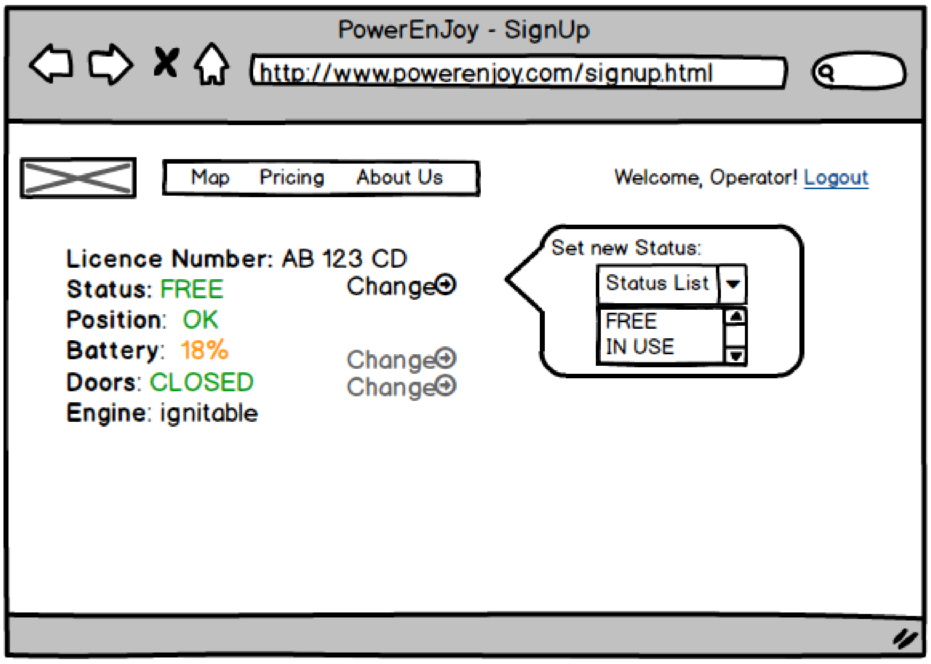
The Car Details page shows all informations available for the chosen car as well as providing buttons to change all editable parameters (for instance the car's status).


PowerEnJoy - SignUp
http://www.powerenjoy.com/signup.html
Map
Pricing
About Us
Welcome, Operator! Logout

Licence Number: AB 123 CD
Status: FREE Change
Position: OK
Battery: 18% Change
Doors: CLOSED Change
Engine: ignitable

4.3.1.5 Changing a Car parameter - Sample (Web)

Choosing to edit a parameter brings up a “pop-up” providing the needed options to edit.

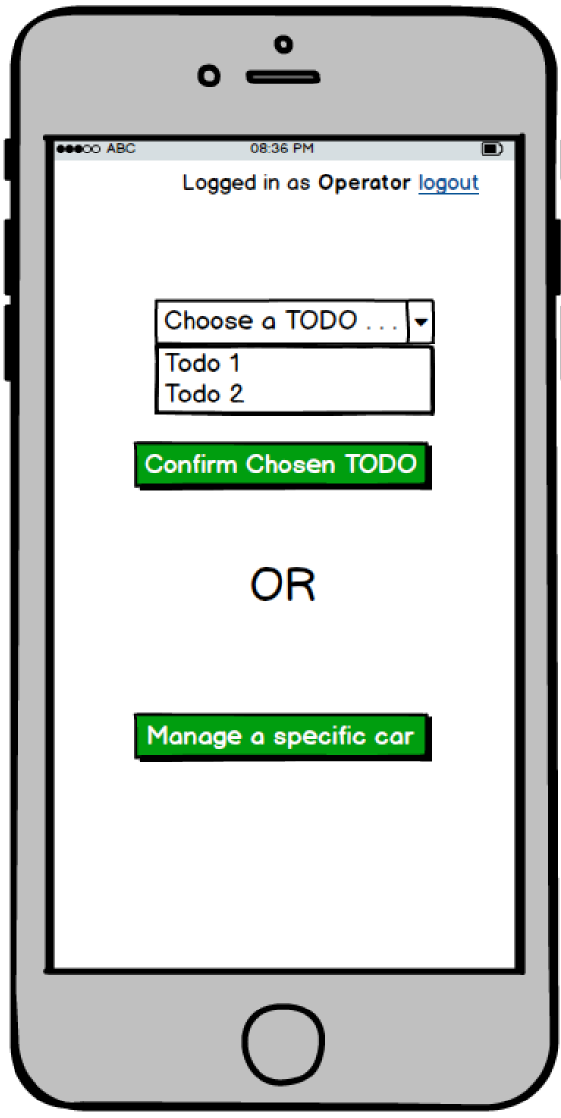


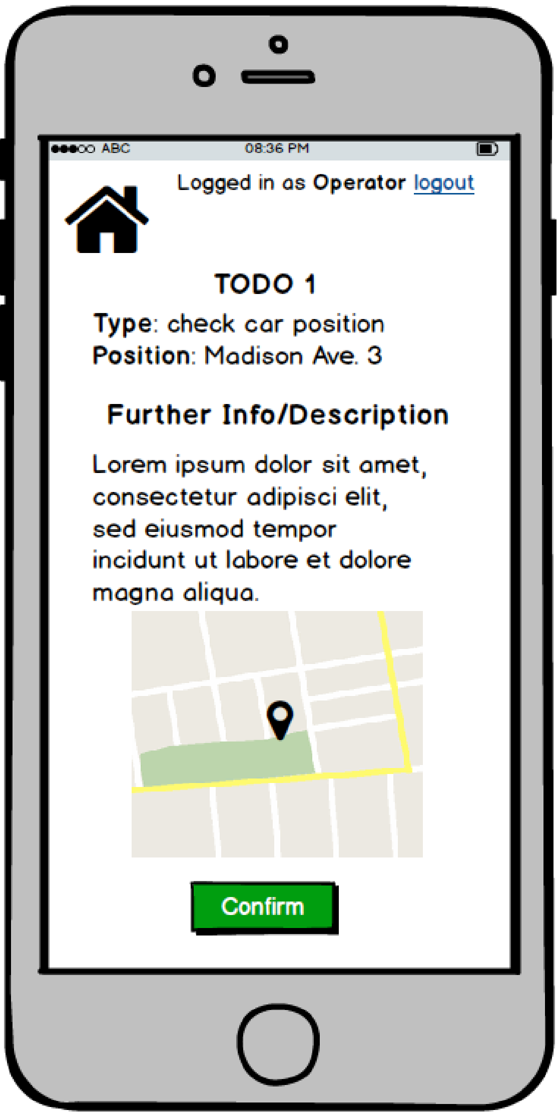
Licence Number: AB 123 CD
Status: FREE
Position: OK
Battery: 18%
Doors: CLOSED
Engine: ignitable

4.3.2 Mobile Interfaces

4.3.2.1 Main Page (Mobile) The mobile Main page for operators offers the same options of the web one: they caan either choose a pensing task or opt to manage a car. (On the left)

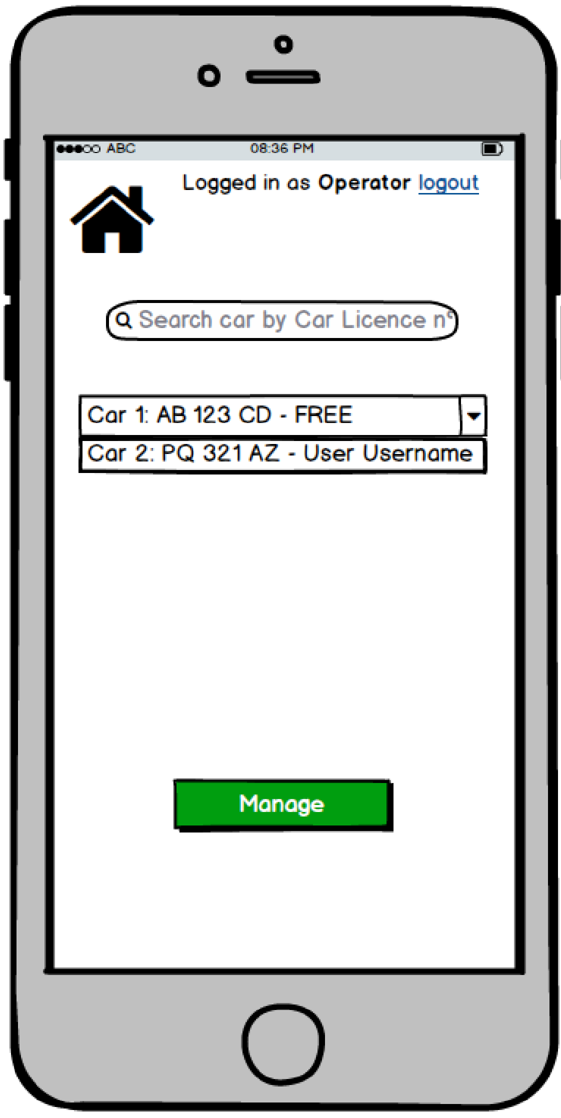
4.3.2.2 Operator chose TODO (Mobile) (On the right)




●●●○○ ABC
08:36 PM
Logged in as **Operator** logout
TODO 1
Type: check car position
Position: Madison Ave. 3
Further Info/Description
Lorem ipsum dolor sit amet, consectetur adipisci elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua.
Confirm

4.3.2.3 Operator Searched Car (Mobile) (On the left)

4.3.2.4 Operator Car Details (Mobile) (On the right)



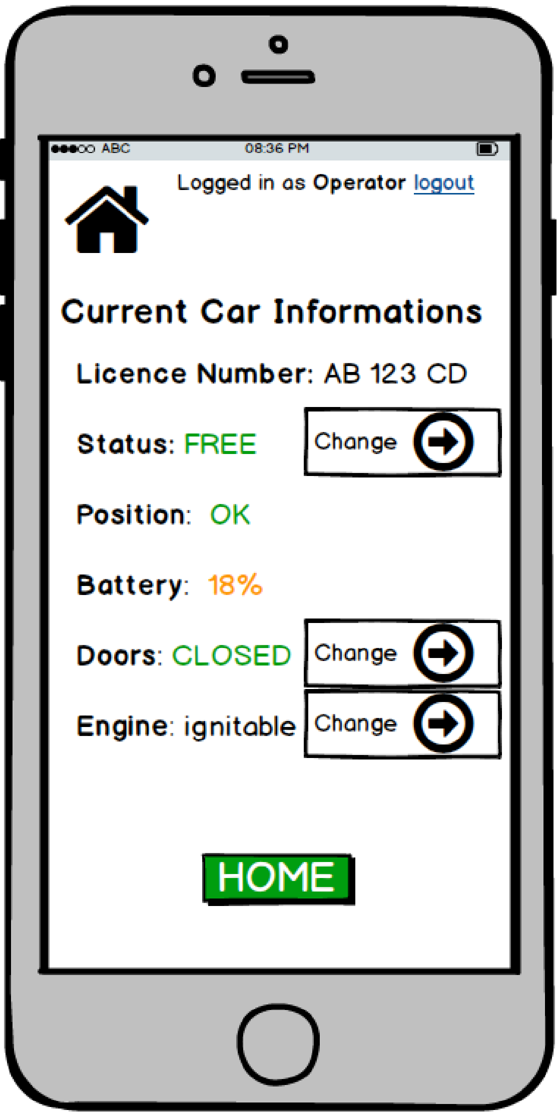
●●●○○ ABC 08:36 PM

Logged in as **Operator** logout

Search car by Car Licence n°

Car 1: AB 123 CD - FREE

Car 2: PQ 321 AZ - User Username



●●●○○ ABC 08:36 PM

Logged in as **Operator** logout

Current Car Informations

Licence Number: AB 123 CD

Status: FREE

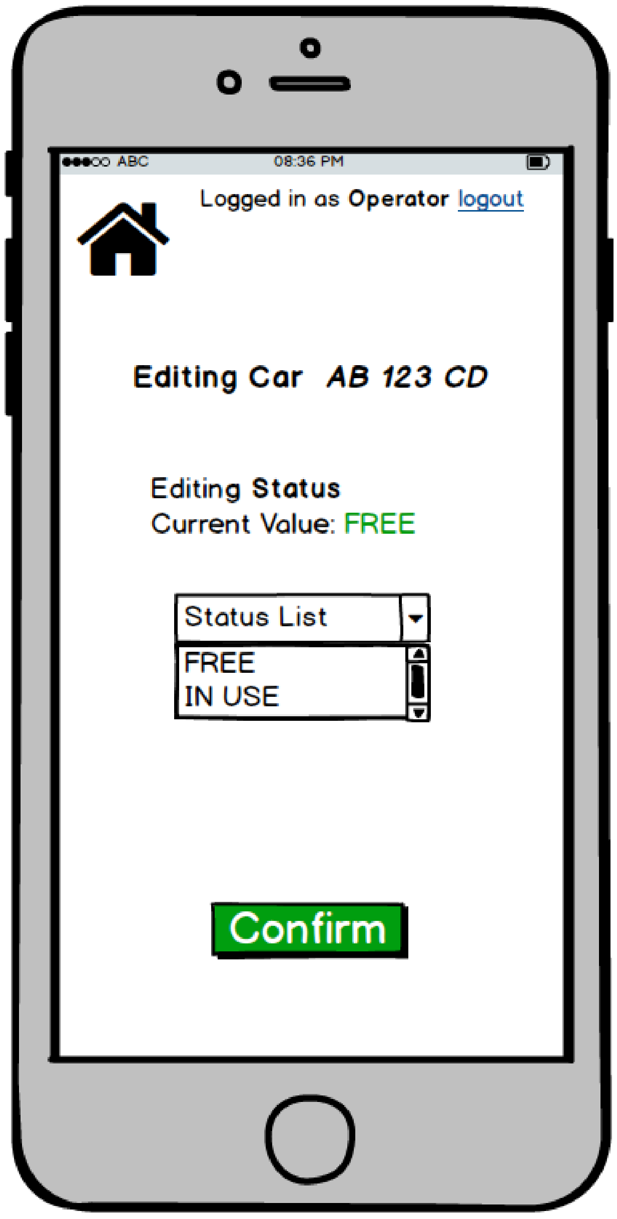
Position: OK

Battery: 18%

Doors: CLOSED

Engine: ignitable

4.3.2.5 Changing a Car parameter - Sample (Mobile)



Logged in as **Operator** logout

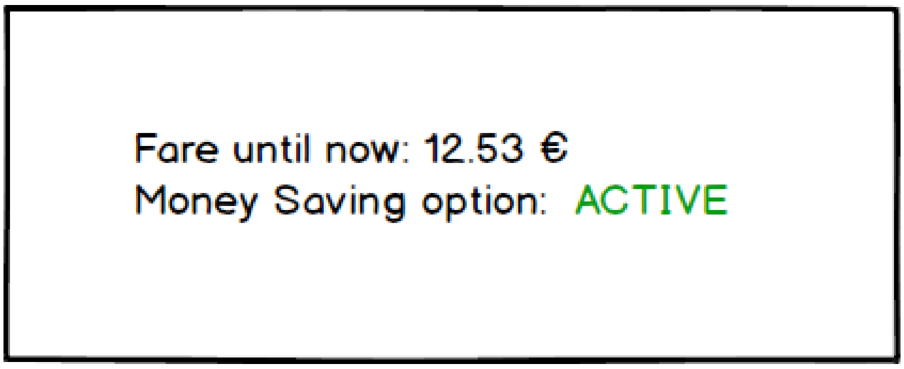
Editing Car *AB 123 CD*

Editing **Status**
Current Value: FREE

4.4 Car Interface

4.4.1 Car Screen

The car has a screen that shows the fare in real time as well as showing whether the Money Saving Option is active or not.


Fare until now: 12.53 €
Money Saving option: ACTIVE

5 Requirements traceability

Below are listed all requirements defined in our RASD and their mapping on the architecture. The map is neither injective nor surjective. In fact some requirements need more than one module to be satisfied and modules achieve the satisfiability of many requirements.

#	**Requirement**	**Mapping on the architecture**
	G1: Allow visitors to sign up	
1	A valid email is required during the registration.	Registration manager
2	A valid DL is required.	Registration manager + DL validator
3	Invalid emails are all addresses that are either already in use or not well formed.	Registration manager
4	Email Address must be verified.	Registration manager
5	The DL is valid if all the fields match the user's information and the Licence Office confirms that the Licence Number, the Name and the Surname are correct.	Registration manager + DL validator
6	Foreign DL must be verified by an Operator.	DL validator + Assistance ticket manager + Administration module
7	If some of the provided information is incorrect, the registration is rejected and the user is prompted to fix the issue(s).	Registration manager
	G2: Allow visitors to log in	
8	A previously registered email is required.	Autentication manager
9	The password chosen during the registration of the submitted email has to be inserted in order to log in.	Autentication manager
10	Visitors submitting incorrect email and/or password shall not be allowed to log in.	Autentication manager
	G3: Allow Users to update or modify their profile's information	
11	Every registered User can modify his information.	Autentication manager + Registration manager
12	Changing email address is allowed but the new email must be confirmed for the account to be valid.	Autentication manager + Registration manager
13	Changing the CC information is allowed as long as the new CC is valid.	Registration manager + CC validator
14	Changing the DL is allowed but the Licence has to be verified.	Registration manager + DL validator
15	Changing name and/or surname is not allowed.	Registration manager

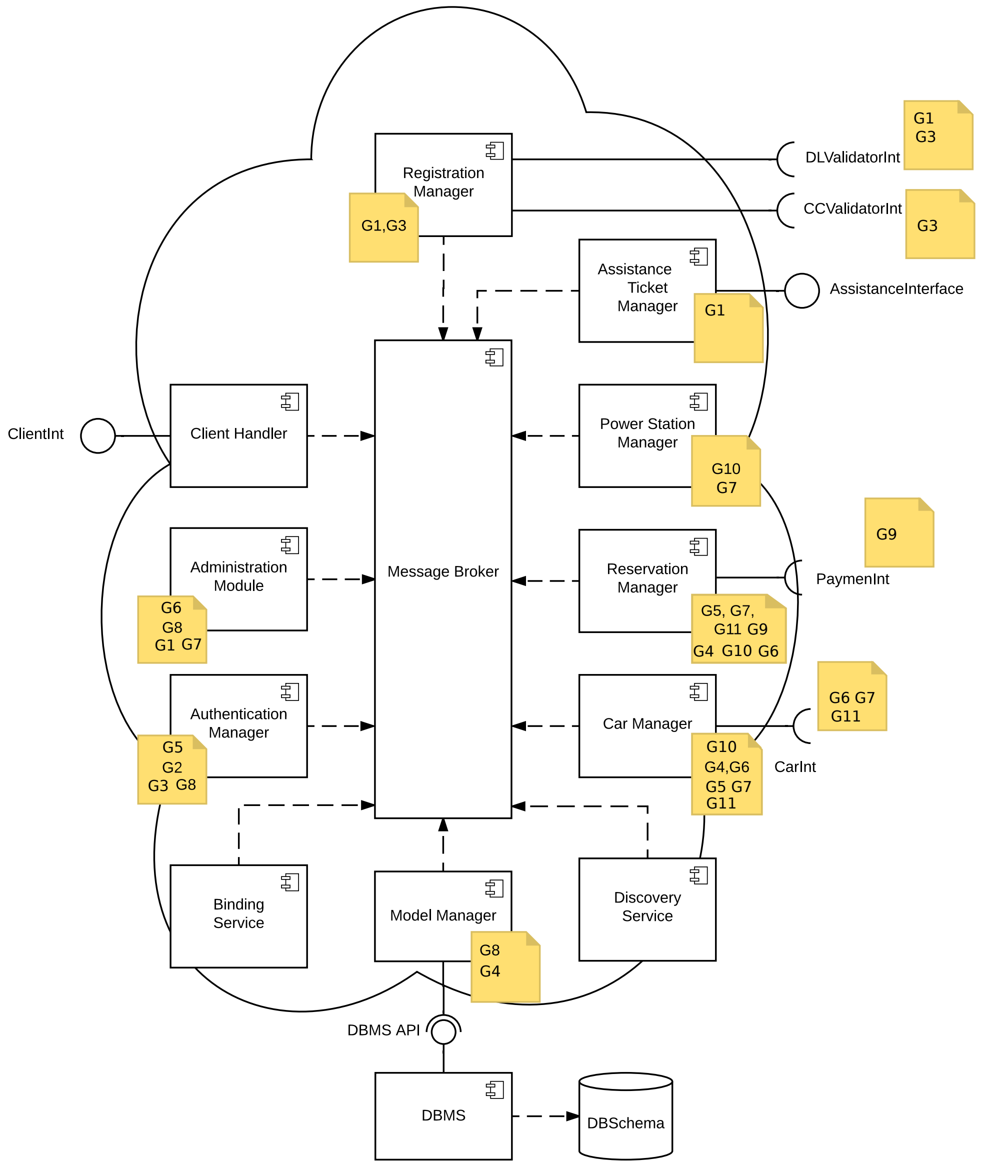
#	**Requirement**	**Mapping on the architecture**
	G4: Show updated information on available cars	
16	The list of available cars always includes only cars that are parked and not reserved and is shown on a map in the location where it is actually parked.	Car manager + Reservation manager
17	If a car is reserved is tagged on the map as reserved.	Car manager + Reservation manager
18	For every car is displayed the remaining percentage of the battery.	Car manager
19	Users should be able to apply filters to show only cars within a certain distance from a specified location or with a minimum percentage of battery left.	Car manager + Model manager
	G5: Allow Active Users to reserve a car	
20	Only Active Users can reserve cars.	Autentication manager + Reservation manager
21	Cars can be reserved by a specific user for up to one hour before being unlocked.	Car manager + Reservation manager
22	If a reserved car is not unlocked in one hour the reservation expires and the user pays a fine of 1€.	Reservation manager
23	Only available cars can be reserved.	Reservation manager
24	If an active user is already using a car he cannot reserve another one.	Reservation manager
25	The system shall provide the user the possibility to delete his pending reservation.	Reservation manager
26	The deletion of a reservation shall be allowed only before the reserved car has been unlocked.	Reservation manager + Car manager
	G6: Allow Active Users to unlock the car they reserved	
27	A car can be unlocked only by the user who has reserved it.	Reservation manager + Car manager
28	There exists a mechanism of acknowledgment between the car and the user.	Car management system + Car manager + Reservation manager
29	The car is unlocked only after the user is acknowledged.	Car management system + Car manager
30	If a user unlocks the car without igniting the engine, the systems starts charging the regular price after the pick up time (one hour from the reservation) expires.	Car management system + Car manager + Reservation manager

#	Requirement	Mapping on the architecture
31	If a user does not ignite the car within 15 minutes from the moment he unlocks it, the systems prompts the user to confirm he is fine. If no answer is received, an operator checks the car.	Car management system + Car manager + Reservation manager + Administration module
G7: Compute the fare		
32	The fare takes into account all price modifiers (discounts or extra fees).	Reservation manager
33	The computed fare is based on how many minutes have passed since the engine was ignited.	Reservation manager + Car management system
34	If the engine was never ignited the fare is calculated as explained in R30.	Car management system + Car manager + Reservation manager
35	The computed fare is shown real-time on the car's screen.	Car management system + Car manager + Reservation manager
36	The system shall be able to apply predetermined discounts to users following some predetermined behavior.	Reservation manager
37	The system shall be able to know the number of passengers into a specific rented car.	Car management system + Car manager
38	The system shall be able to know the battery percentage of a specific car in any moment.	Car management system + Car manager
39	The system shall be able to locate a car and know whether or not it is plugged at one of the predetermined parking areas in any moment.	Car management system + Car manager + Power station manager
40	The system shall be able to measure the distance of a specific car from the nearest power grid station.	Car management system + Car manager + Power station manager
41	The system shall be able to make an Operator charge a specific car.	Car management system + Car manager + Power station manager + Administration module
42	The system shall know if a user selected the money saving option.	Reservation maanager
G8: Allow System Administrator(s) to update information		
43	The System Administrator is granted the necessary permissions allowing him to access each cars' data, status and position.	Autentication manager
44	The system shall present an Interface for the System Administrator in order to access with the necessary permissions.	Autentication manager

#	Requirement	Mapping on the architecture
45	Only those accessing the systems with these permissions shall be able to access sensible data regarding the cars.	Autentication manager
46	The system shall be able to check the consistency of the information modified given a set of rules.	Model manager
G9: Ensure that the fare is paid		
47	As soon as the rental is ended, use the payment method provided by the AU to pay the fare.	Reservation manager + Payment Handler
48	If the provided method is invalid (e.g. expired or empty CC), notify the user.	Reservation manager + Payment Handler
49	Users with unsuccessful pending fare shall not be allowed to book cars.	Reservation manager
50	Allow users to specify the main payment method.	Registration manager
51	Periodically try to charge pending fare through the specified main payment method.	Reservation manager + Payment Handler
G10: Allow the driver to choose the money saving option		
52	The systems suggests a charging station near the user's destination if present (within a user-selectable radius).	Reservation manager
53	The suggested station must have at least a free plug to be suggested.	Reservation manager
54	The suggested station shall be determined to ensure a uniform distribution of cars.	Car manager
55	If the AU doesn't park in the suggested station the money saving option shall not be taken into account for future decisions.	Reservation manager
56	If the AU chooses this option, the system gives him the address and shows directions on the smartphone.	Reservation manager
G11: Allow the user to park the rented car in safe zone		
57	Parking shall be allowed only in safe zones.	Reservation manager
58	The system shall not charge the user after the car has been parked and he has exited the car.	Reservation manager + Car manager + Car management system
59	The system shall lock the car, if parked in a safe area and the user has exited the car.	Reservation manager + Car manager + Car management system

#	Requirement	Mapping on the architecture
60	The system shall be able to recognize whether or not there is a user inside the car.	Car manager + Car management system

The corrispondence is summarized in the following figure.



6 Effort spent

Component	Time spent (in hour)
Philippe Scorsolini	35
Lorenzo Semeria	22
Gabriele Vanoni	22

7 References

- https://www.rabbitmq.com/documentation.html
- http://microservices.io/patterns/apigateway.html
- http://microservices.io/patterns/microservices.html
- http://microservices.io/patterns/data/shared-database.html
- http://microservices.io/patterns/microservice-chassis.html
- http://projects.spring.io/spring-boot/
- https://www.adayinthelifeof.nl/2011/06/02/asynchronous-operations-in-rest/
- https://spring.io/blog/2015/07/14/microservices-with-spring
- https://sudo.hailoapp.com/services/2015/03/09/journey-into-a-microservice-world-part-2/
- https://www.nginx.com/blog/introduction-to-microservices/?utm_source=deploying-microservices&utm_medium=blog&utm_campaign=Microservices
- http://www.cloudcomputingpatterns.org/
- https://www.amqp.org/
- http://mqtt.org/
- http://brunorocha.org/python/microservices-with-python-rabbitmq-and-nameko.html
- http://www.slideshare.net/chris.e.richardson/developing-apps-with-a-microservice-architecture-svforum-microservices-meetup
- https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-amqp-overview
- http://projects.spring.io/spring-cloud/
- http://nordicapis.com/api-gateways-direct-microservices-architecture/
- https://www.nginx.com/blog/introducing-the-nginx-microservices-reference-architecture/
- https://www.nginx.com/blog/adopting-microservices-at-netflix-lessons-for-team-and-process-design/
- https://msdn.microsoft.com/en-us/library/dn568099.aspx
- https://sudo.hailoapp.com/web/2014/12/08/webapps-as-microservices/
- https://github.com/mfornos/awesome-microservices