POLITECNICO
MILANO 1863

Politecnico di Milano
A.A. 2016/2017
Software Engineering 2: "*PowerEnJoy*" v. 1.0

Design **D**ocument

Matteo Bresich (mat. 774366)

December 11, 2016

Table of Content

1 Introduction

1.1 Purpose

This is the Design Document for the *PowerEnJoy* application. The target audience of this document are project managers, developers, testers and Quality Assurance. Its aim is to provide a functional description of the main architectural components. The document can be used for help maintenance and further development.

1.2 Scope

This document focuses on the non functional requirements of *PowerEnJoy* . It provides guidance and template material which is intended to assist the development phase of the project.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Thin client: low power calculation client.
- Fat server: high computing power server able to handle many parallel requests.
- User session: the entire period of time in which the user is logged in.

1.3.2 Acronyms

- JEE: Java Enterprise Edition
- RASD: Requirements Analysis and Specification Document.
- CSGestion: Car Sharing Gestion.
- ERP: Enterprise Resource Planning.
- UI: User Interface.
- API: Application Program Interface.
- MOM: Message Oriented Middleware.
- EIS: Enterprise Information Systems.

1.4 Reference Document

- Specification document
- RASD for *PowerEnJoy*
- IEEE Standard on Design Description

1.5 Document Structure

This document presents the system architecture using different levels of detail. Every design decision is justified by a description of the reasons. The design is develop in a top-down approach to ensure a good structure for the system.

1.5.1 Main Topics

1. Introduction:
   synopsis of the document.

2. Architectural design:
   are presented all the components of the system and the interaction between them.

3. Algorithm design:
   flowchart and descriptions of the fundamental algorithms of *PowerEnJoy* .

4. User interface design:
   mock-ups of the UI.

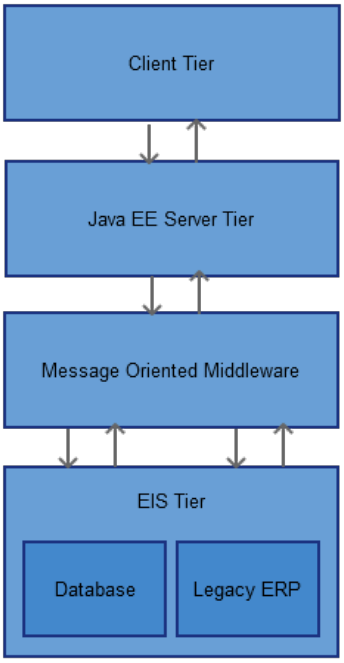
5. Requirements traceability:
   mapping between the requirements and the components.

6. References

2 Architectural Design

2.1 Overview

This section of the design document provides a general description of the design of the system and its processes. The *PowerEnJoy* system will follows the "event-driven" and "client-server" architectural style, there will be a thin client and a fat server and the interaction between the components takes place through asynchronous events. The *PowerEnJoy* system will be implemented using JEE following a multi-tier architecture as shown in the picture below. In according to the philosophy of divide and conquer, the choice of a multi-tier architecture is motivated by the flexibility and the reusability of the application. In fact developers, by segregating an application into tiers, acquire the option of modifying or adding a specific layer, instead of reworking the entire application. Furthermore the simplicity of the design and implementation of this architecture allows an easy integration with the old custom ERP.



1. *Client Tier*: contains Application Clients and Web Browsers and it is the layer designed to interact with the users. In our case it includes the browser and the mobile application.

2. *Java EE Server Tier*: contains the Servlets and Dynamic Web Pages that need to be elaborated and the Java Beans, which contain the business logic of the application.

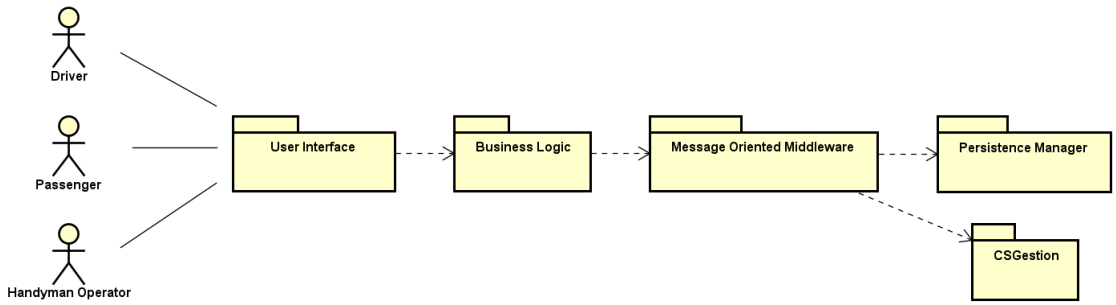
3. *MOM Tier*: contains a Message Oriented Middleware based on Java Message Service that allows an easier integration of the legacy ERP and also permit to have loosely coupled components that can be replaced with alternative implementations that provide the same services.

4. *EIS Tier*: contains the data source. In our case, there is a database and the legacy ERP that both are allowed to manipulate and to retrieve relevant data.

2.2 General Component view and interactions

2.2.1 General Package Design

Considering the chosen architecture, it can be identified the following packages:



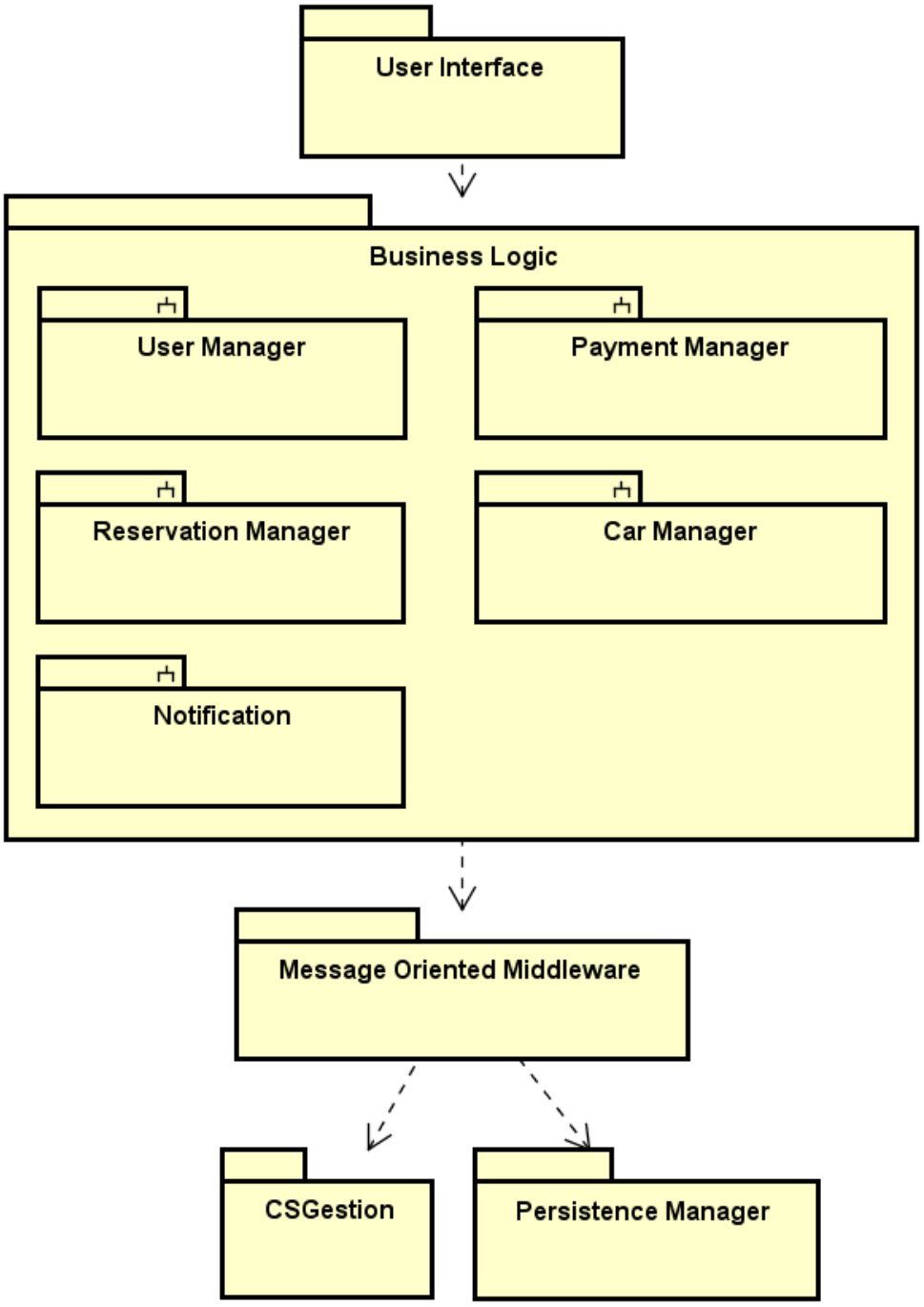
- User Interface
- Business Logic
- Message Oriented Middleware
- Persistence Manager
- CSGestion

All users, as shown in the picture above, can not directly access all packages except for the ui.

2.2.2 Detailed Package Design

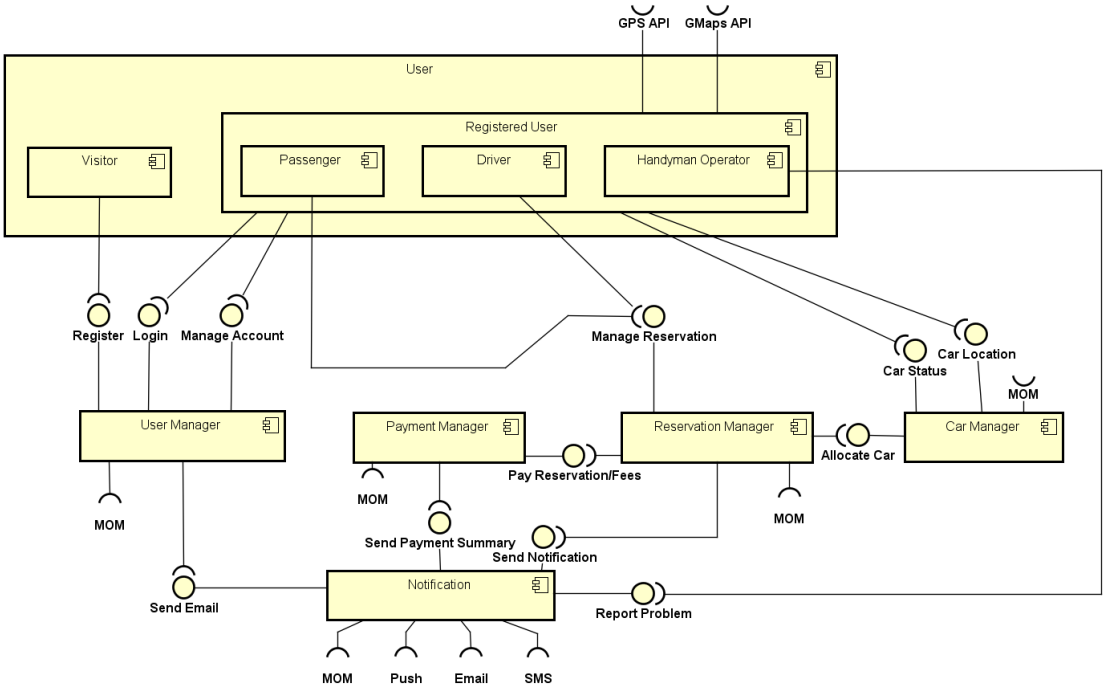
The inner packages are described as follows:

- User Interface: set of sub-packages responsible of user's interactions and of forwarding information requests to the Business Logic sub-packages.
- Business Logic: set of sub-packages responsible for handling requests from the User Interface package, process requests and send back a result. These packages may access the Message Oriented Middleware package.
- Message Oriented Middleware: set of sub-packages prepared to be an intermediary between Persistence Manager, Legacy ERP and the Business Logic package.
- Persistence Manager: set of sub-packages contains the data model for the system. It accepts requests from the Message Oriented Middleware package.
- CSGestion: set of sub-packages that represent the legacy ERP of the company (for more information see CSGestion documentation).


User Interface
Business Logic
User Manager
Payment Manager
Reservation Manager
Car Manager
Notification
Message Oriented Middleware
CSGestion
Persistence Manager

2.2.3 General Component View

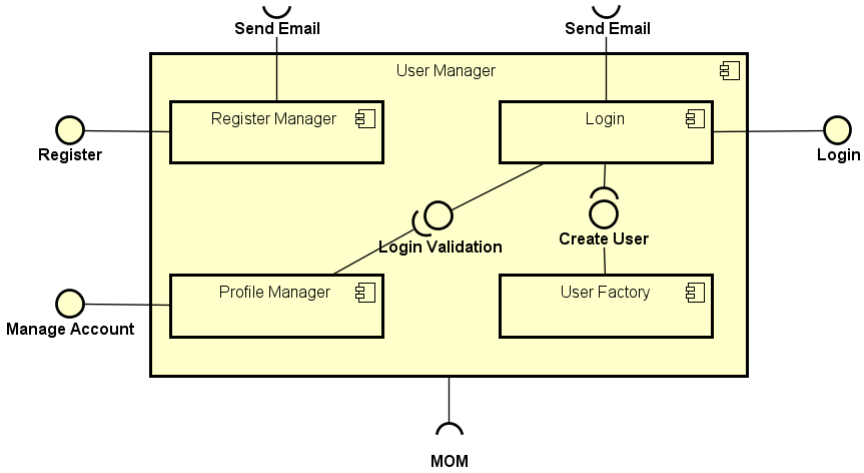
The picture below represents the main components and interfaces of *PowerEnJoy* .



2.3 Component view and interfaces

Follows a more detailed description for each component with its interfaces.

2.3.1 User Manager



Register Manager

Definition	Component that controls visitors' registrations.
Responsibilities	This component allows visitors to sign up into *PowerEnJoy* and become registered users. It connects to the MOM to store and retrieve user's credentials and Send Email interface to verify the sign up procedure through a confirmation email.
Interaction	With the visitors, the MOM and Notification.
Interfaces offered	• Register for Visitor
Interfaces required	• MOM • Send Email
Implementation	Static class

Login

Definition	Component that controls users' login.
Responsibilities	This component allows users to login into *PowerEnJoy* . It is connected to the MOM to verify the credentials and grants access to the Profile Manager and it sends email and to recover the user's password.
Interaction	With all users and the MOM.
Interfaces offered	• Login for User • Login Validation for Profile Manager
Interfaces required	• MOM • Send Email • Create User
Implementation	Static class

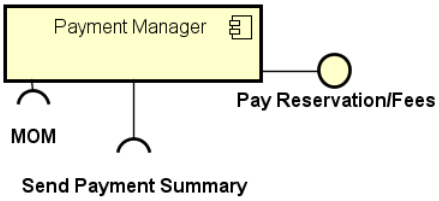
Profile Manager

Definition	Component that controls users' profile.
Responsibilities	This component allows users to edit their profile. For instance to change the password, payment method, phone number and so on.
Interaction	With all the Registered User of the system, with the MOM and with the Login component.
Interfaces offered	• Manage Account for Registered User
Interfaces required	• MOM • Login Validation for Login
Implementation	Multi instance: one for each user session.

User Factory

Definition	Component that instantiates registered users.
Responsibilities	This component is used for the creation of an instance for every logged in user.
Interaction	With Login component and with the MOM.
Interfaces offered	• Create User for Login
Interfaces required	• MOM
Implementation	Factory design pattern.

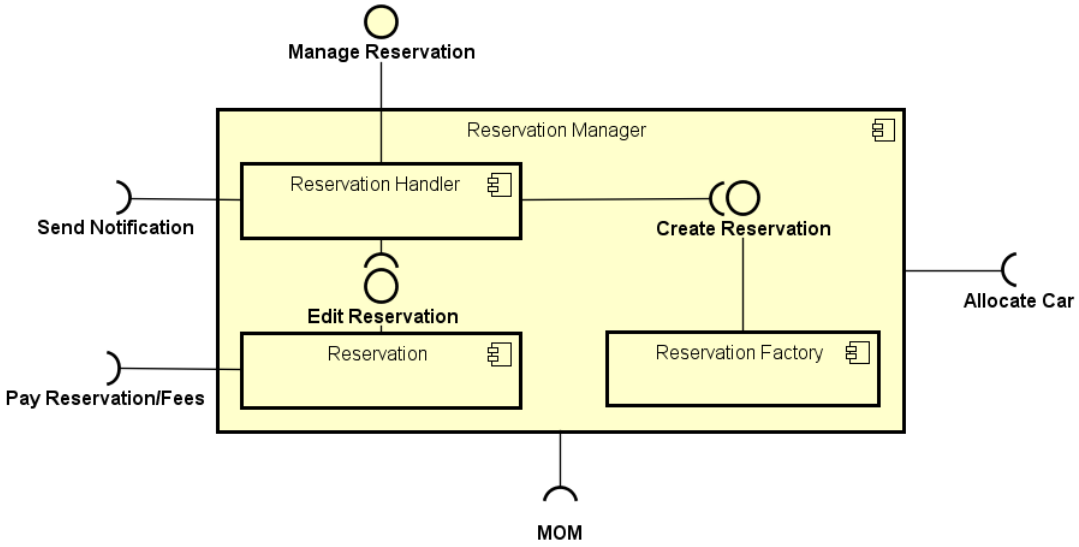
2.3.2 Payment Manager



Payment Manager

Definition	Component that controls the payment process.
Responsibilities	This component is able to computing the amount of money that the user have to pay. It uses an interface to the MOM to interact with the payment service of the legacy ERP.
Interaction	With the Reservation Manager to receive information about the reservation. With Notification to send the payment summary. With MOM to interact with the ERP's payment service.
Interfaces offered	• Payment Reservation/Fees for Reservation Manager
Interfaces required	• MOM • Send Payment Summary
Implementation	Static class

2.3.3 Reservation Manager



Reservation Handler

Definition	Component that receives the users' requests.
Responsibilities	This component is able to add or remove a car or a seat reservation made by Drivers or Passengers. It sends notification to Drivers and Passengers in case of a cancellation of a booking. This component also has the function of expire a reservation.
Interaction	With the Reservation Factory to create a reservation. With the Reservation to edit reservation data.
Interfaces offered	• Manage Reservation for Driver and Passenger
Interfaces required	• MOM • Create Reservation for Reservation Factory • Edit Reservation for Reservation • Send Notification for Notification
Implementation	Multi instance: one for each Driver or Passenger.

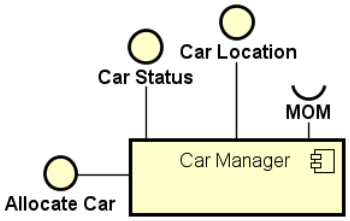
Reservation Factory

Definition	Component that controls the creation of a reservation.
Responsibilities	This component is able to create a car reservation. It uses an interface to the Car Manager to select a car and book it.
Interaction	With the Reservation Handler and with the Car Manager.
Interfaces offered	• Create Reservation for Reservation Handler
Interfaces required	• Allocate Car for Car Manager
Implementation	Factory design pattern.

Reservation

Definition	Component that controls the reservation.
Responsibilities	This component is able to edit reservation informations. It uses an interface to the MOM to save into the DB reservation data that is not managed by the legacy ERP. It uses an interface to the Car Manager to release the booking of a car when the reservation expires or ends. This component also have to create the reservation payments.
Interaction	With the Payment Manager to create payments. With the MOM to retrieve and modify reservation data.
Interfaces offered	• Edit Reservation for Reservation Handler
Interfaces required	• MOM • Payment Reservation/Fees for Payment Manager • Allocate Car for Car Manager
Implementation	Multi instance: one for each Driver or Passenger reservation.

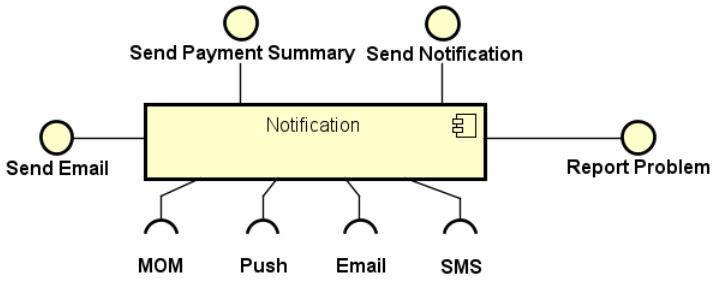
2.3.4 Car Manager



Car Manager

Definition	Component that controls the cars informations.
Responsibilities	This component provides the car status, the car location and if a car is booked or not. It uses an interface to the MOM to retrieve and update cars' informations.
Interaction	With the Reservation Manager and with the Registered User.
Interfaces offered	• Allocate Car for Reservation Manager • Car Status for Registered User • Car Location for Registered User
Interfaces required	• MOM
Implementation	Singleton.

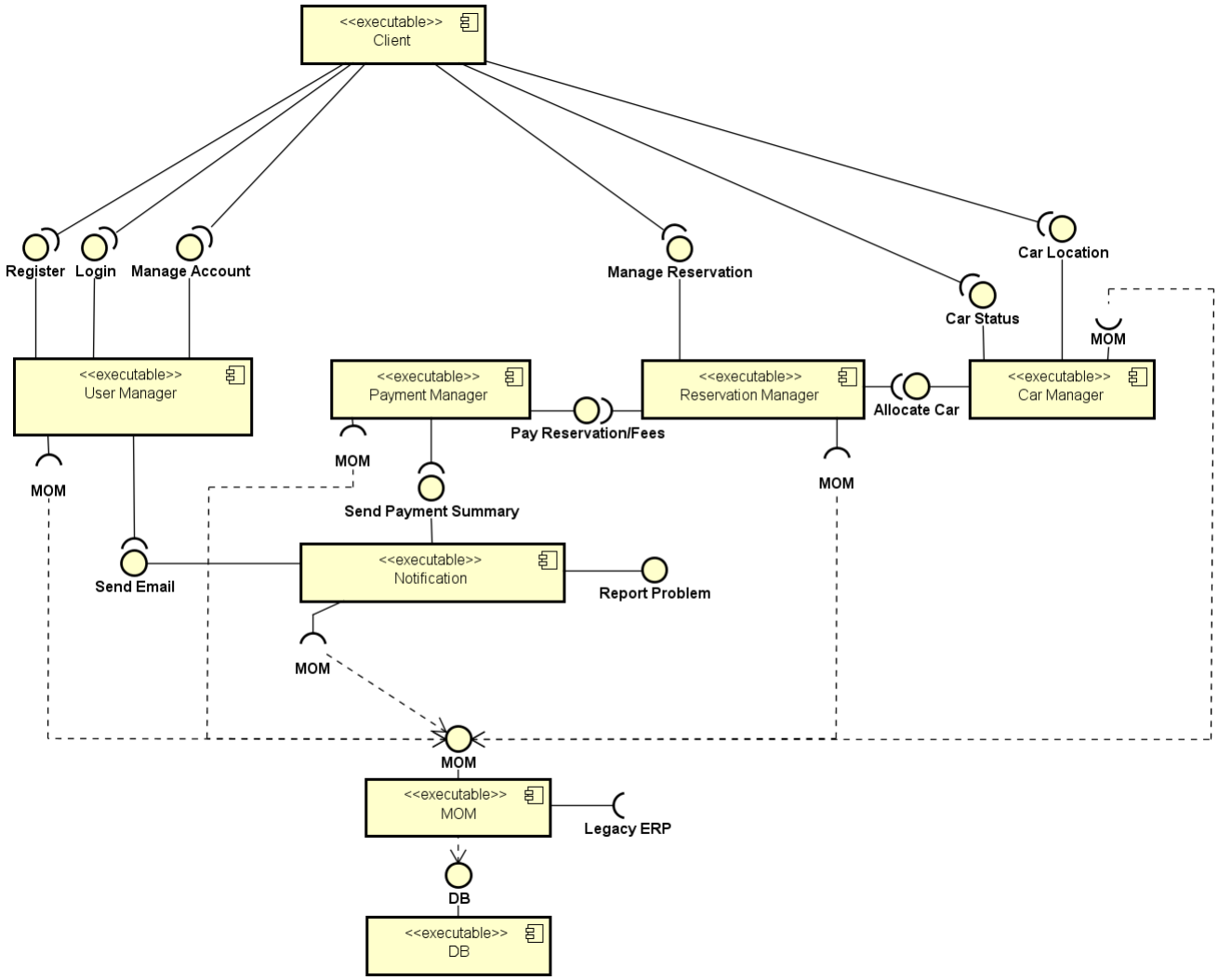
2.3.5 Notification



Notification

Definition	Component that controls notifications sent to the users.
Responsibilities	This component is able to send messages to an user using SMS, Emails and in-app push notifications and it is also able to send an internal company notification through the MOM.
Interaction	With User Manager to send Emails for verify the sign up procedure and to recover user's passwords. With Payment Manager to send payment summary. With Handyman Operator to send internal company notification. With the Reservation Manager to send notification to the user.
Interfaces offered	• Send Email for User Manager • Send Payment Summary for Payment Manager • Send Notification for Reservation Manager • Report Problem for Handyman Operator
Interfaces required	• MOM • Push Notification service • Email service • SMS service
Implementation	Static class

2.4 Deployment view

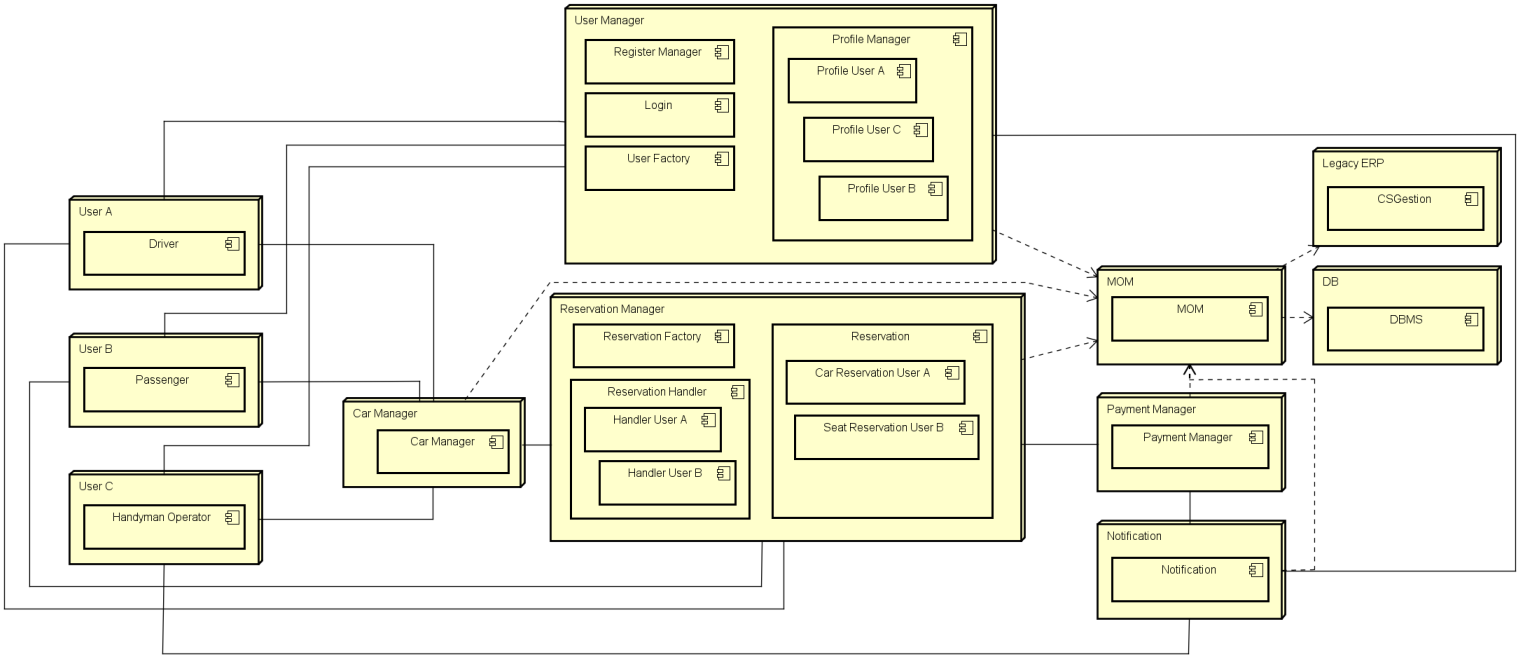


According to the previous schema the physical deployment of the system will composed by the web server ("Client" in the diagram), by the application server (all the managers, "MOM" and the "Notification" node in the diagram), by the database server ("DB") used for the necessary persistence not provided by the Legacy ERP.

2.5 Runtime view

2.5.1 Runtime Unit

The schema below describes the behavior and interaction of the system's building blocks as runtime elements.



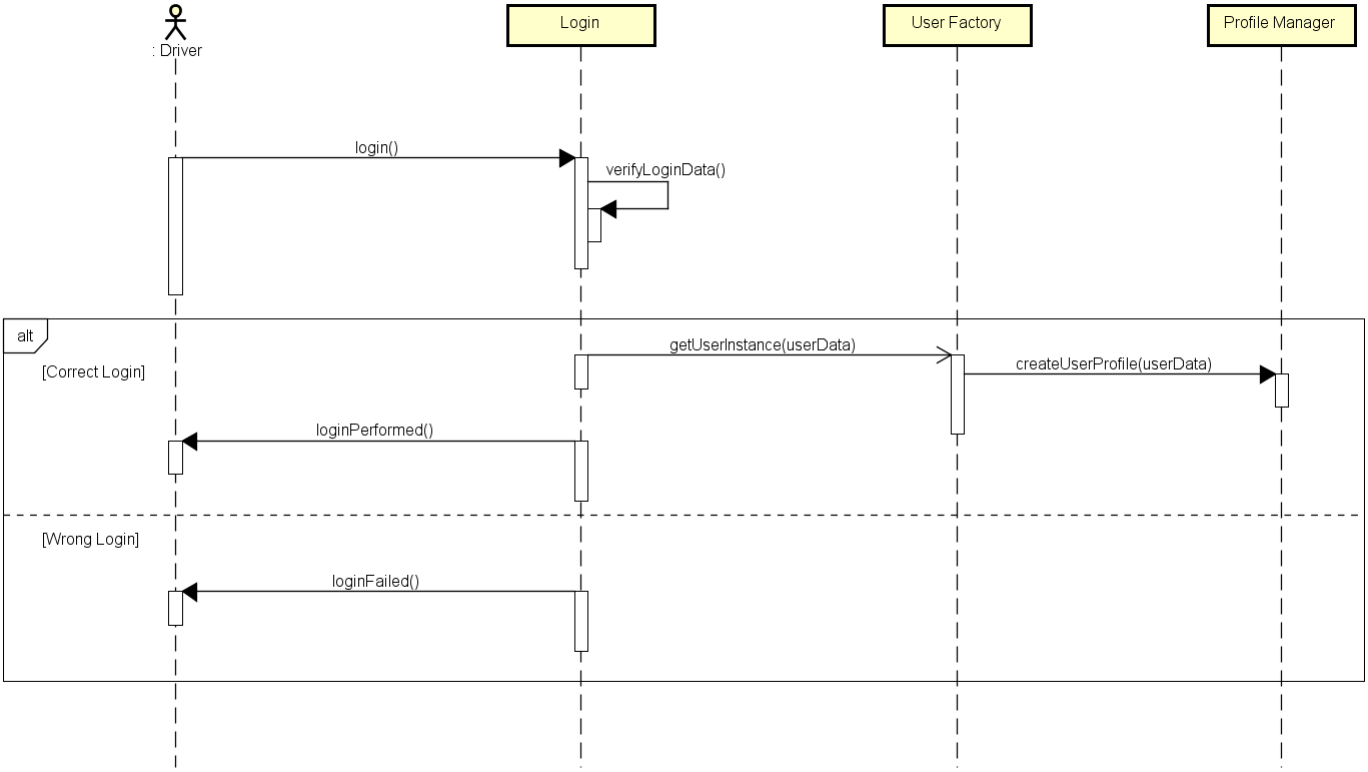
It clarify which elements have more than one instance at runtime and which not. In our case Profile Manager and Reservation Handler are multi-instance elements.

2.5.2 Sequence Diagram

Follows some meaningful sequence diagrams that show the interaction between components at runtime.

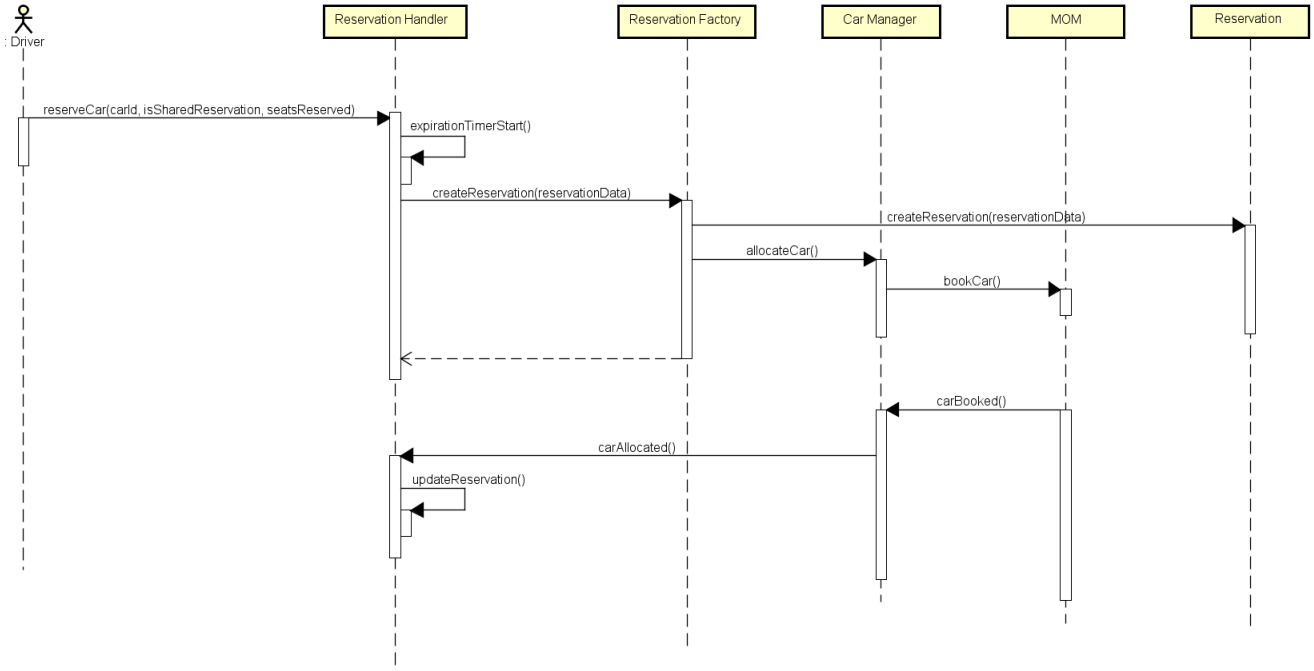
2.5.2.1 Login

The sequence diagram below represents what happens when a registered user logins to the system.



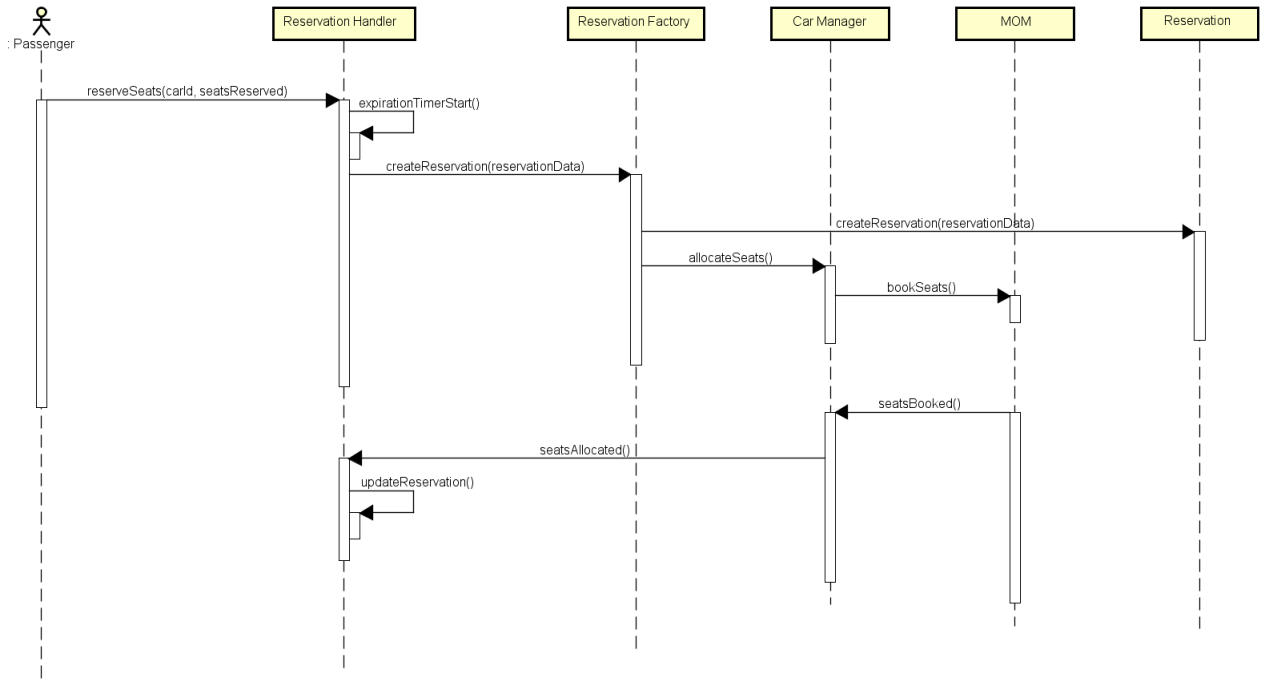
2.5.2.2 Car Reservation Request

The sequence diagram below represents what happens when a Driver requests a car.



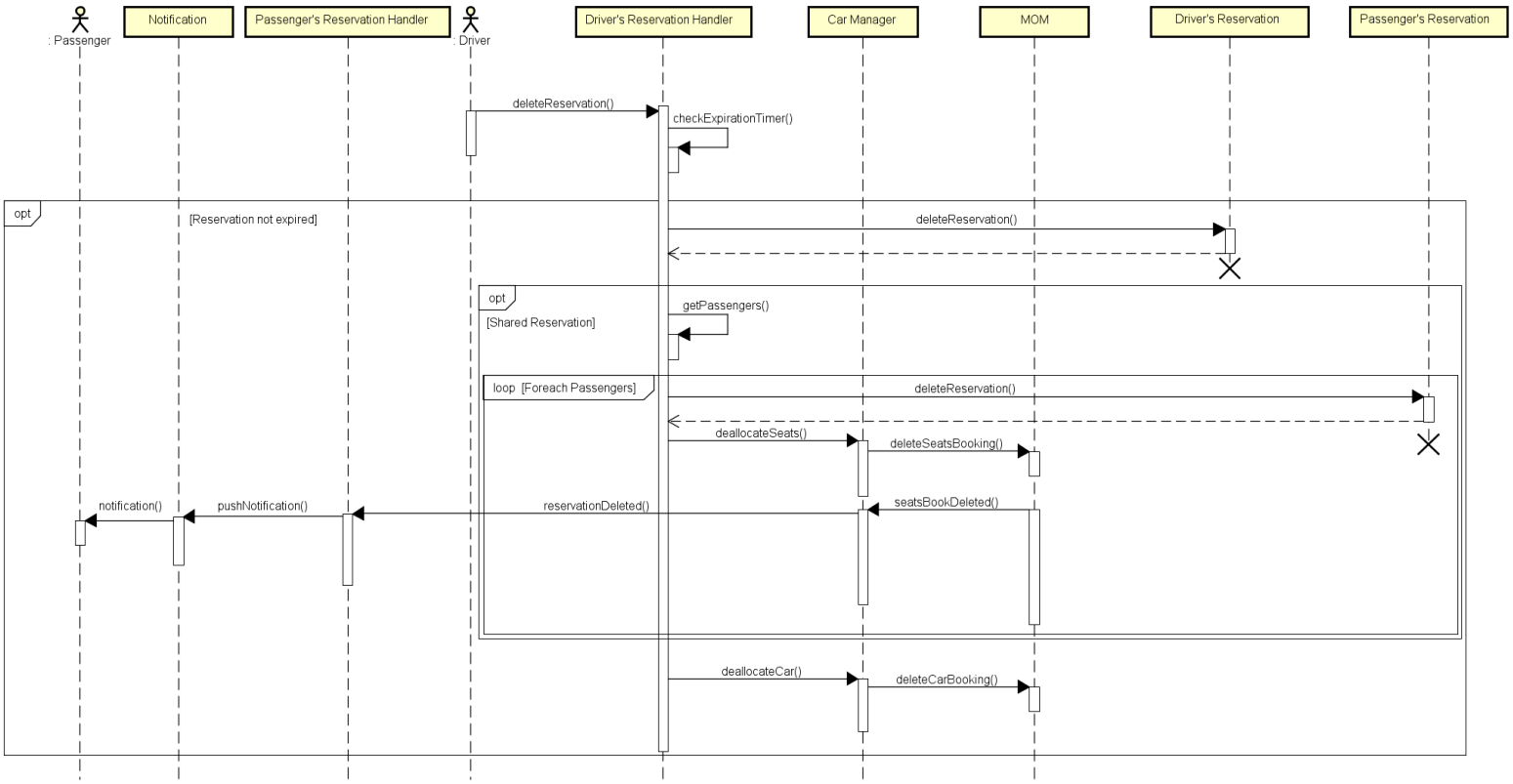
2.5.2.3 Seat Reservation Request

The sequence diagram below represents what happens when a Passenger requests a seats.



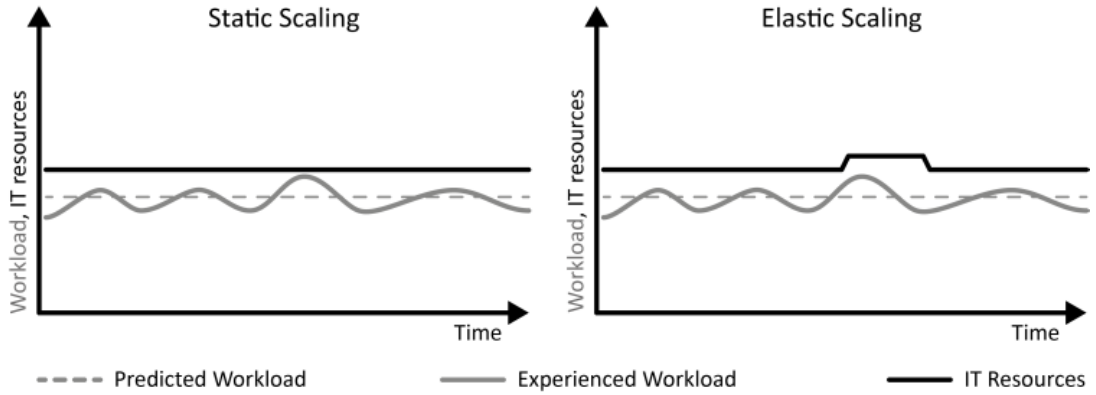
2.5.2.4 Delete Car Reservation

The sequence diagram below represents what happens when a Driver deletes a car reservation.



2.6 Selected architectural styles, patterns and other design decisions

The decision of using the client-server architecture is motivated by cost savings. In fact the company wants to use its server that already have. However, the *PowerEnJoy* application is also designed to be implemented in a cloud architecture. In the future cloud architecture may be the best solution. In fact though the expected workload is static in case of a heavy load increase the cloud architecture offers an elastic scaling (see the figure below).



Some software design patterns are used in the design of the system, for instance the Factory method pattern is used for the creation of instances for Reservations and the Profiles, while the Singleton pattern is used for Car Manager.
The system have to be always fast and responsive, also in case of a high number of requests and accesses to the service at the same time, for this reason parallelization is highly used;

3 Algorithm Design

Follows some distinctive algorithms of *PowerEnJoy* . All algorithms are presented with pseudocode and flowchart diagrams.

3.1 Payment

Follows the payment algorithm. It represents how the system calculate and deducts money from the payments method chosen by the user. This algorithm is part of Payment Manager component. It gets all the reservation's informations, calculate the payment with the function cost (see below for more details) and sends the payment summary. It interacts with the Reservation Manager, with the MOM and the Notification component.

3.1.1 Cost Function

There are different functions cost, one for each case:

- Non Shared Ride
- Shared Ride
- Fees

3.1.1.1 Non Shared Ride This function is chosen in case of a non shared ride.

- t: movement time
- N: number of seats occupied during the ride (for at least 50% of the movement time)
- k: hourly cost float constant [€/hour]

$$f_{cost}(t) = \begin{cases} kt, & N < 3 \\ kt(0.9), & N \geq 3 \end{cases} \tag{1}$$

3.1.1.2 Shared Ride This function is chosen in case of a shared ride. It is calculated for each i-th registered user that take part to the ride.

- t: movement time
- N: number of seats occupied during the ride (for at least 50% of the movement time)
- n: number of seats occupied by registered user
- k: hourly cost float constant [€/hour]
- S: scale factor

$$S_i = \frac{n}{N} \tag{2}$$

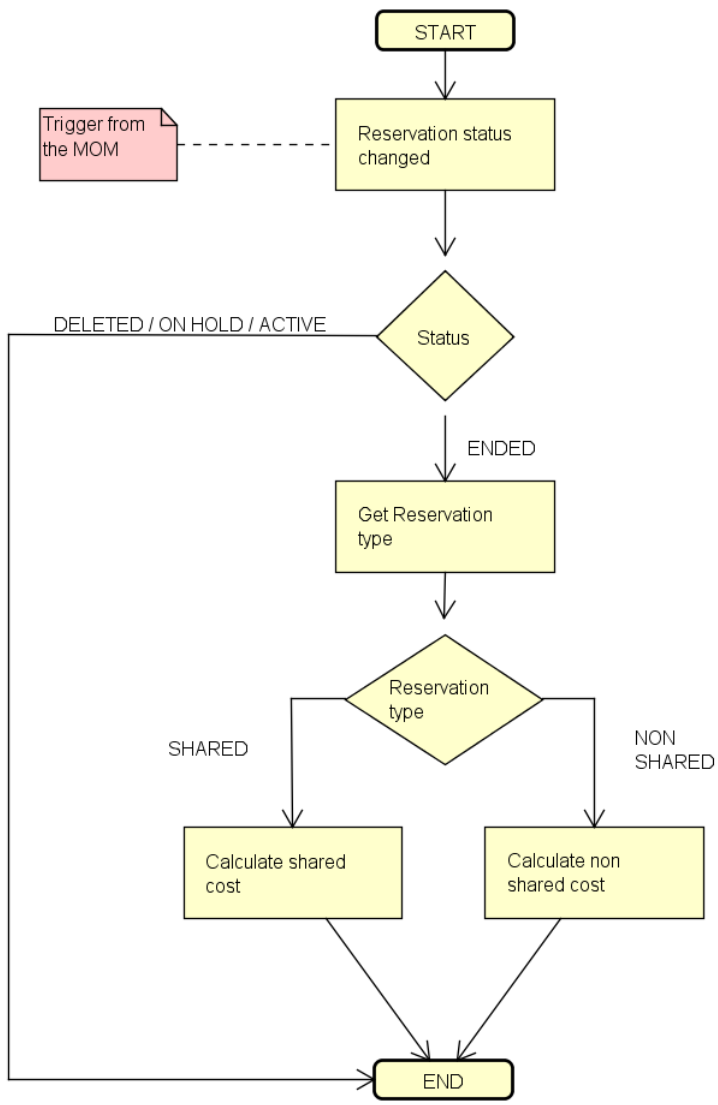
$$f_{cost_i}(t) = \begin{cases} S_i kt, & N < 3 \\ S_i kt(0.9), & N \geq 3 \end{cases} \tag{3}$$

3.1.1.3 Fees This function is calculated when the system have to discourage bad behavior (i.e. the driver do not pick-up the car within one hour from the reservation). It is time invariant.

$$f_{cost} = \text{const} \tag{4}$$

3.1.2 Pseudocode and Flowchart

```
for (every car reservation status changed)
        if (reservation status == ON HOLD)
                no action;
        else if (reservation status == ACTIVE)
                no action;
        else if (reservation status == DELETED)
                no action;
        else if (reservation status == EXPIRED)
                calculate(expire_fee);
        else
                get reservation type;
                if (reservation type == NON SHARED)
                        calculate non shared cost;
                else
                        calculate shared cost;
        payment process;
```

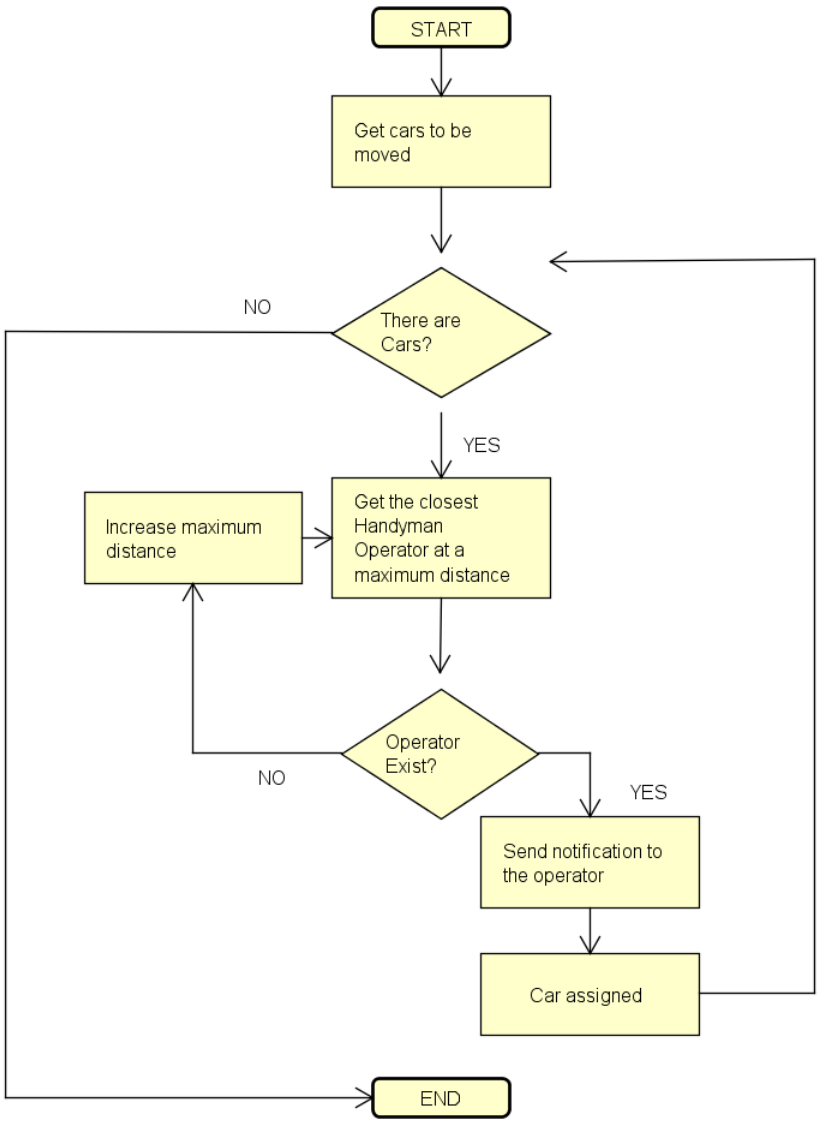


3.2 Good car distribution

Follows the car distribution algorithm. It represents how the system allocates to handymand operators cars to be moved. It interacts with the MOM to get cars that have to be moved, the initial and the final location for each auto.

3.2.1 Pseudocode and Flowchart

```
get cars to be moved;
for (every car to be moved)
        getTheClosestHandyman(car location, max distance);
        while (closest operator not exist)
                increase max distance;
                getTheClosestHandyman(car location, max distance);
        send notification to the Handyman Operator;
```

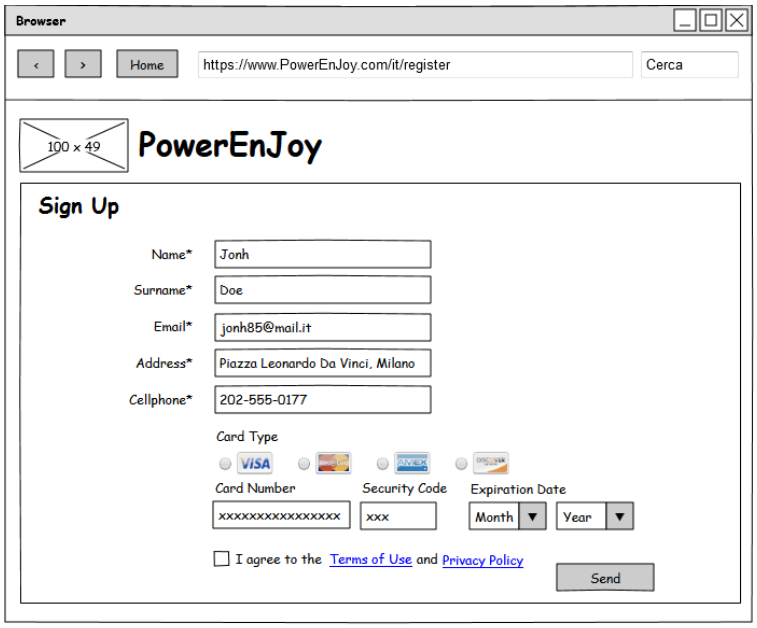


4 User Interface Design

This section is the same featured in the RASD (see User Interfaces). Follows some mock-ups that preview the user interface.

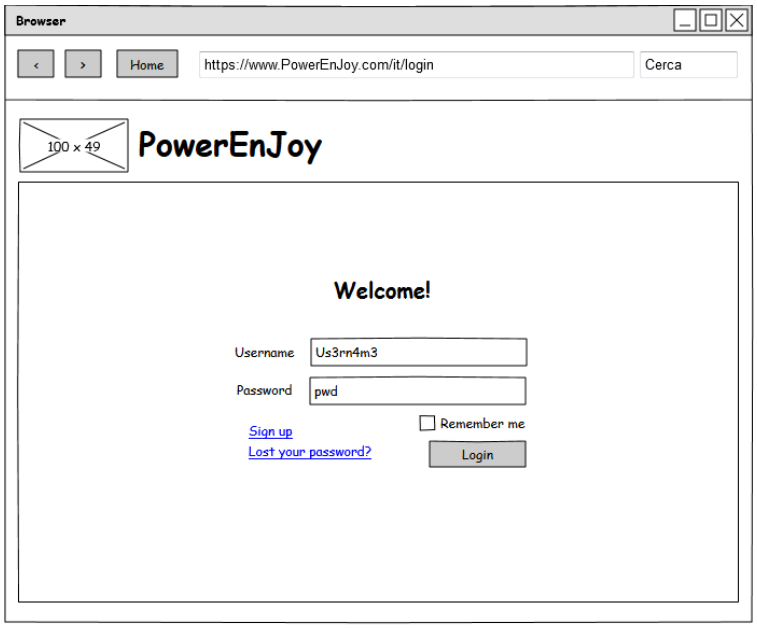
4.1 Registration

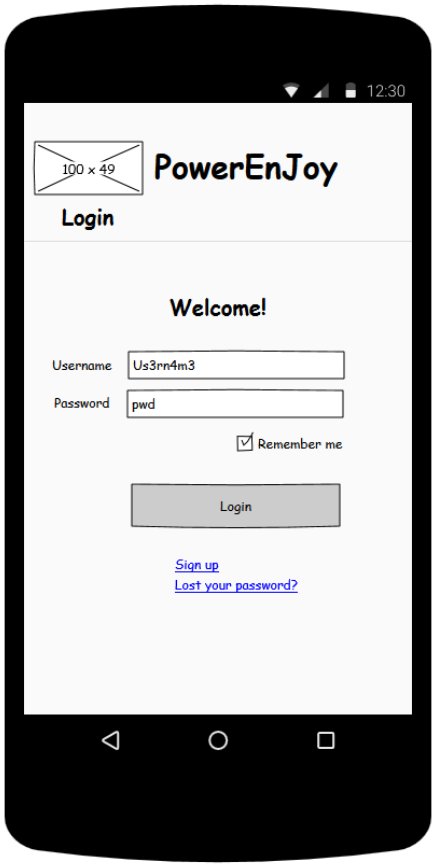
This page presents the registration form for the user.



4.2 Login

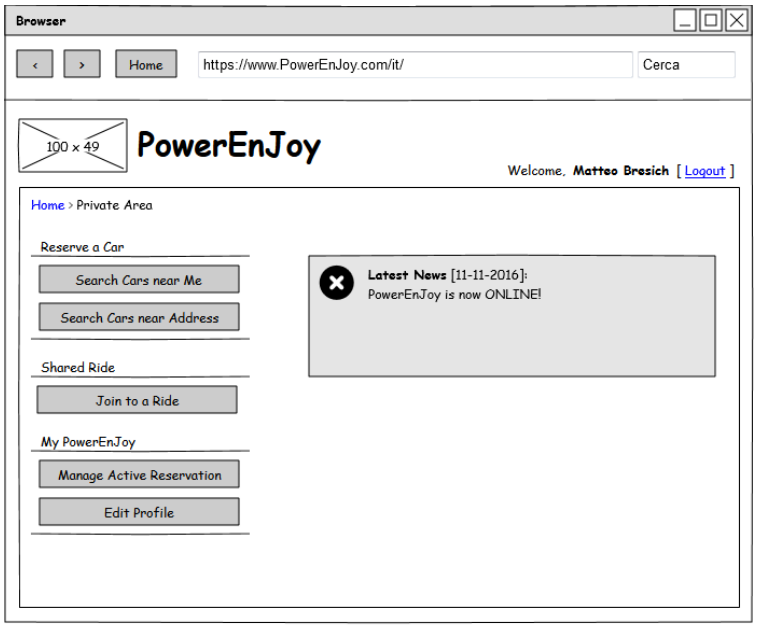
This page presents the login form for the user.

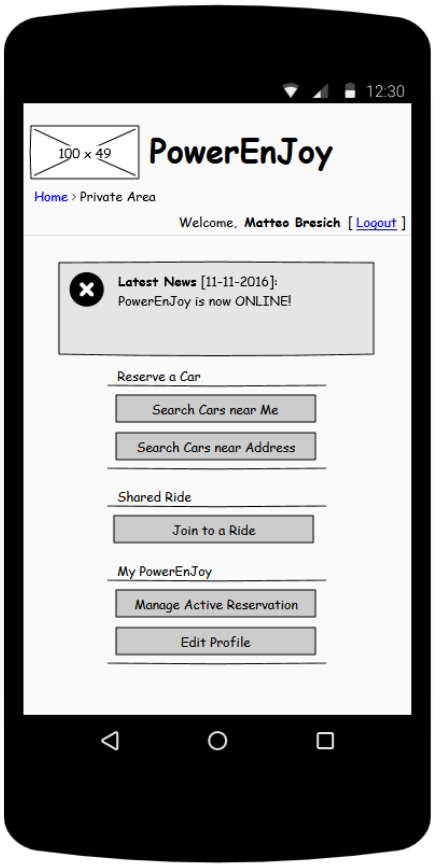




4.3 Private Area

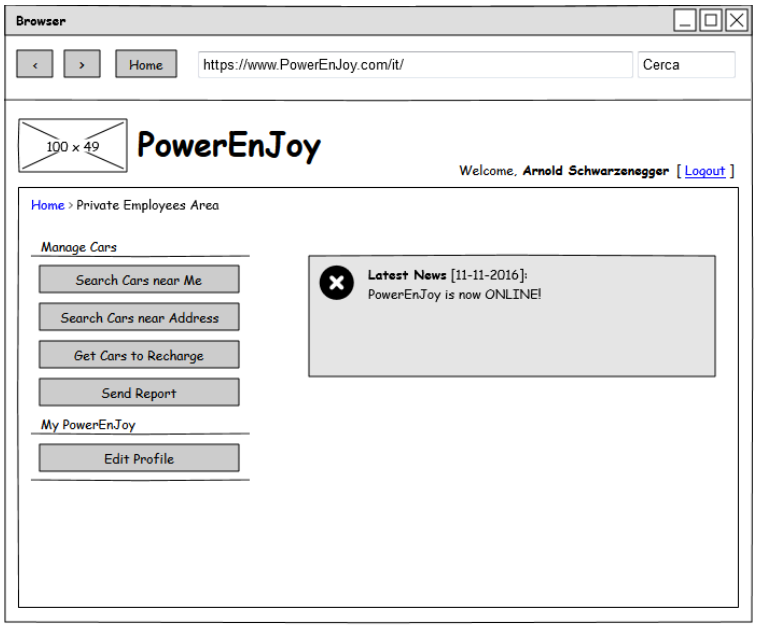
This page presents the private area for the registered user.

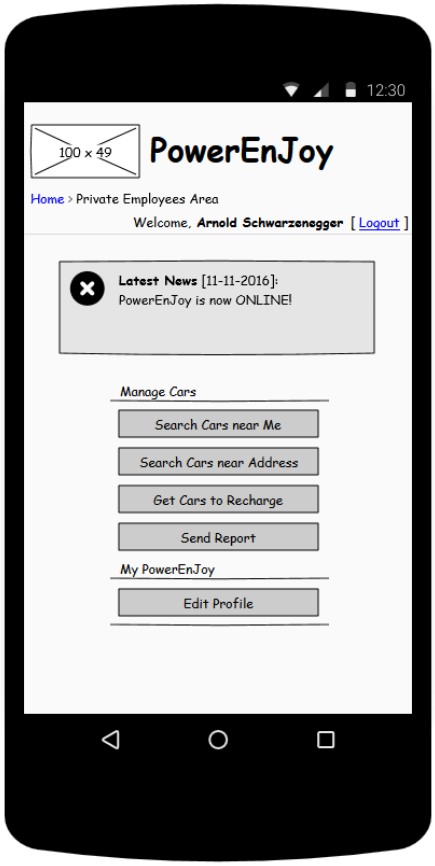




4.4 Private Employees Area

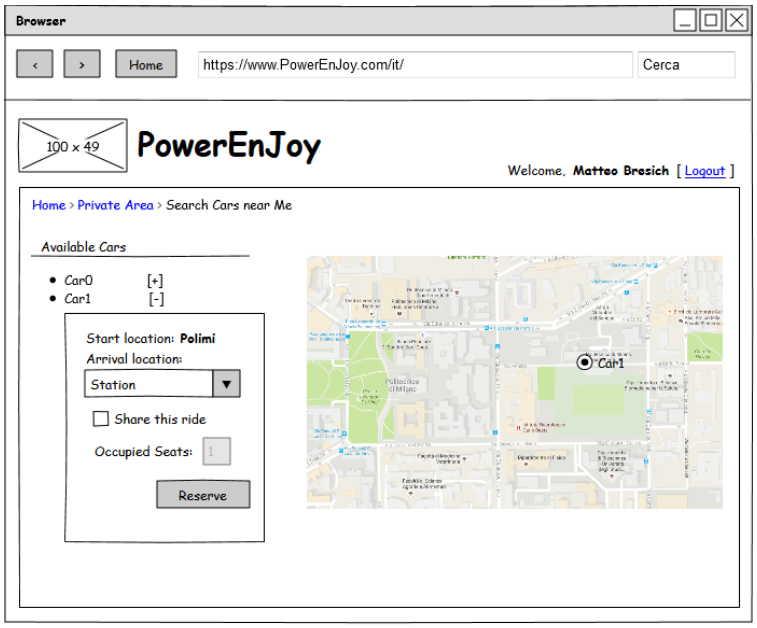
This page presents the private area for the Handyman Operator.

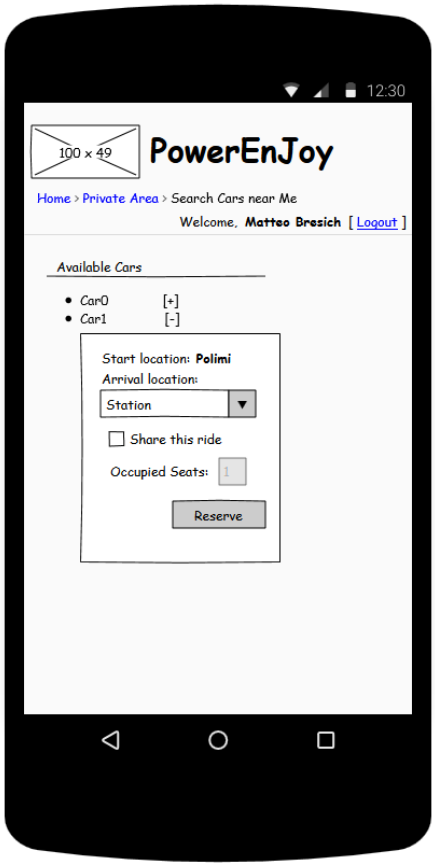




4.5 Car Reservation Page

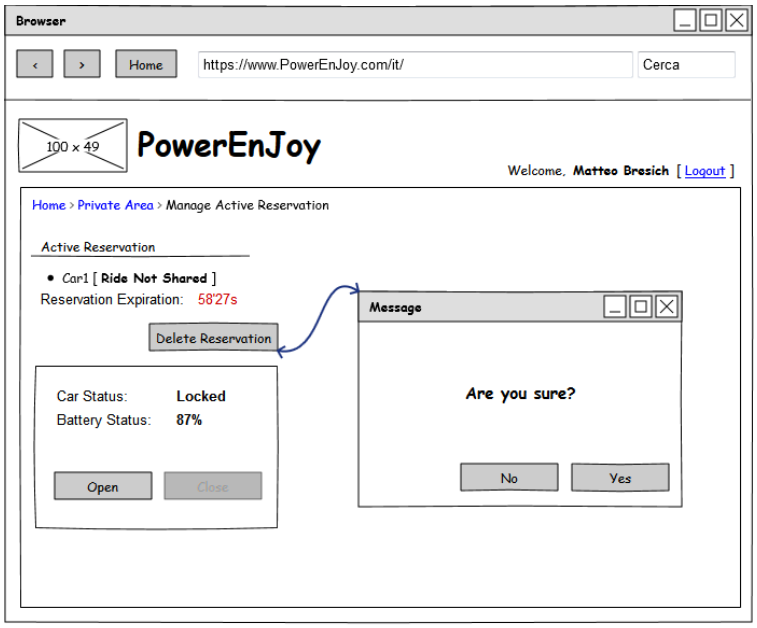
This page presents the car booking page for the registered user.

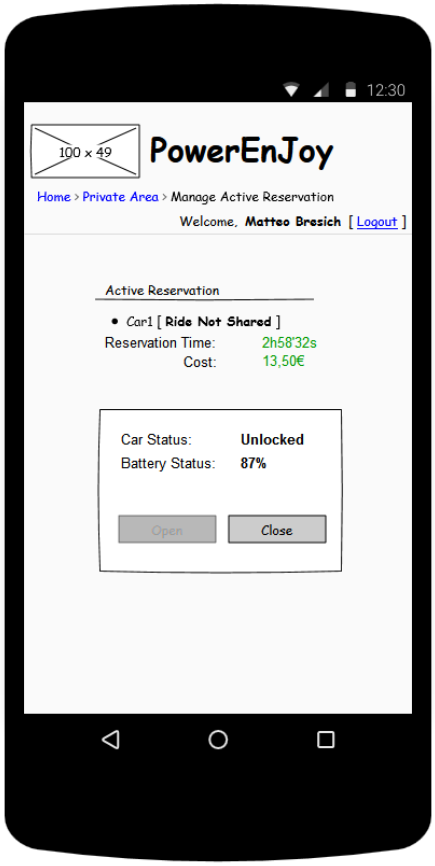




4.6 Reservation Management Page

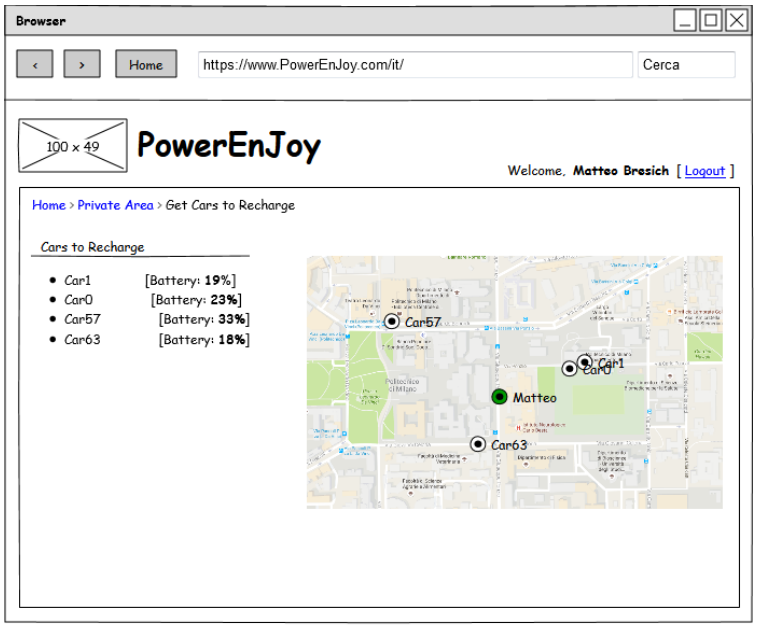
Follows the managment reservation page for the registered user.

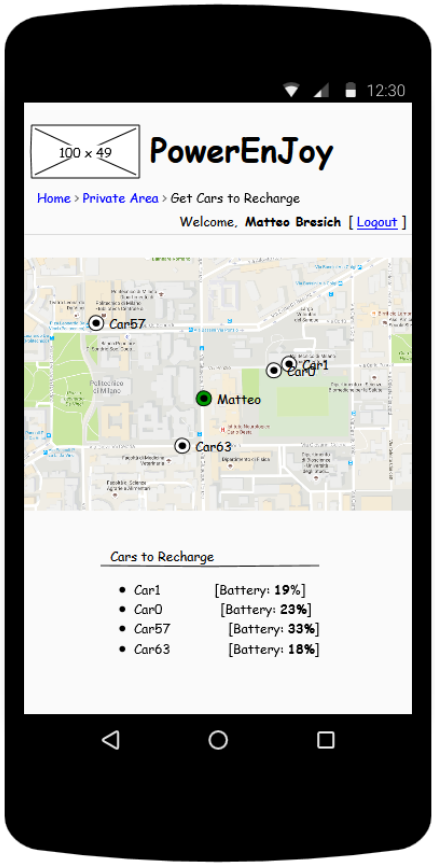




4.7 Get Cars to Recharge Page

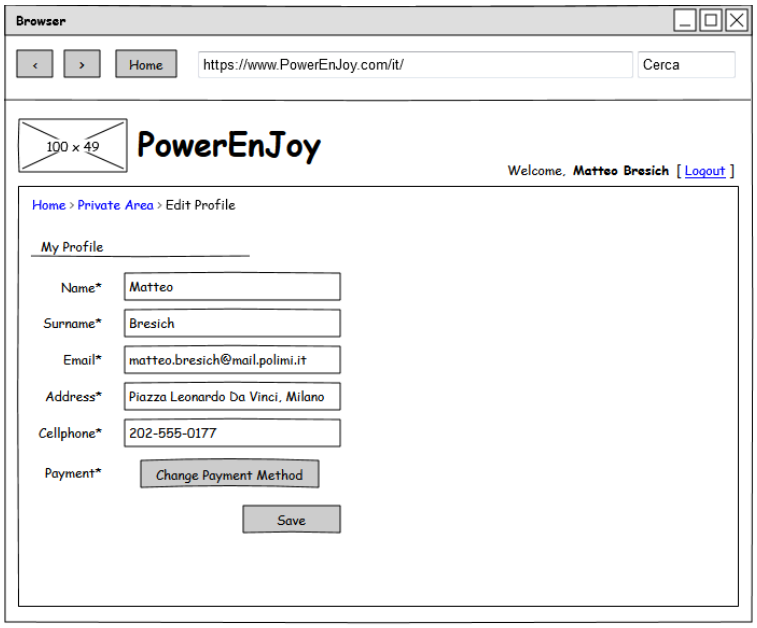
This page presents cars that have to be connect to the charging station for the Handyman Operator.





4.8 Edit Profile

Follows the edit profile page for the registered user.



5 Requirements Traceability

The following table shows the mapping between the required functionalities of the system and the components used to implement and satisfy them.

Goals	Assigned component
[G1] The System has to encourage the user to have a good behaviour using discounts.	Payment Manager handles the payment algorithm that includes discounts. (for more details see section 3.1).
[G2] The System has to discourage the user to have a bad behaviour using fees.	Payment Manager handles the payment algorithm that includes fees. (for more details see section 3.1).
[G3] The System has to ensure a good distribution of cars in the territory.	Car Manager handles the distribution of the cars in the territory. (for more details see section 3.2).
[G4] The System has to provide the ability to share a ride.	The Reservation Manager component handles cars reservations and allows shared rides.
[G5] The Visitor shall be able to sign-up via app.	The User Manager component allows a visitor to sign up to the service.
[G6] The Registered User shall be able to log into the service.	The User Manager component allows to log in the system.
[G7] The Registered User shall be able to reserve a car up to one hour before via app.	The Reservation Manager component handles cars reservations and allows to reserve a car.
[G8] The Registered User shall be able to remove a reservation before the reservation time runs out.	The Reservation Manager component handles cars reservations and allows to delete a reservation.
[G9] The Registered User shall be able to find the locations of available cars within a certain distance from him.	Car Manager component allows to find the locations of available cars within a certain distance from the user.
[G10] The Registered User shall be able to find the locations of available cars from a specified address.	Car Manager component allows to find the locations of available cars at a specified address.
[G11] The Registered User shall open/close a reserved car via app when he/she is close to it.	Car Manager component allows to lock and unlock a booked car.
[G12] The Passenger shall be able to use a car in shared mode.	The Reservation Manager component handles cars reservations and allows to use a car for a shared ride.
[G13] The Handyman Operator shall know the location of the car that has to be charged.	Car Manager component allows to know the location of the car that has to be charged.
[G14] The Handyman Operator shall report any problem with a car.	Notification component allows to report a problem.

6 Effort Spent

6.1 Hours of work

The time spent to redact this document:

- Bresich Matteo: 66 hours.

Days	Hours of work
28/11/16	3h
03/12/16	4h
05/12/16	10h
06/12/16	10h
07/12/16	10h
08/12/16	8h
09/12/16	8h
10/12/16	10h
11/12/16	3h

7 References

- TeXstudio v2.11.2 (http://www.texstudio.org/) to produce this document.
- Evolus Pencil v2.0.5 (http://pencil.evolus.vn/) to generate mockups.
- Astah Professional 7.1.0 (http://astah.net/) to create Use Cases Diagrams, Sequence Diagrams, Class Diagrams and State Machine Diagrams.