**Design Document**



Version 1.0

Luca Santini 808710

Riccardo Remigio 874939

1. Introduction
   1. Purpose

The purpose of this document is to specify details about the architecture and design of the software that will be developed to meet the demands of PowerEnJoy.

This document will be useful to coordinate the work of all developers who are going to write the code, this will be their reference to follow.

* 1. Scope

The PowerEnJoy system will be developed to mainly satisfy two types of users:

* The clients of the service
* The technicians

What we're going to develop is a web application which can be accessed from PC or smartphone or other device that can connect to a browser. The user once logged in will be recognized as a customer or as a technician and as a result will have their own functionalities.

A customer can: visualize a map with the vehicles available, reserve a car and use it. During the rental, users can see the current charge. A user with good behavior will benefit from discounts.

The technicians can visualize all the vehicles and change manually their states. Then the functionalities for technicians will help them to work more efficiently and in a less stressful way.

* 1. Definition, Acronyms, Abbreviations

RASD: Requirements Analysis and Specifications Document

DD: Design Document

GPS: Global Positioning System

PC: Personal Computer

App: Software application

UX: User Experience

BCE: Boundary-Control-Entity

DB: Database

DBMS: Database Management System

API: Application Programming Interface, is a common way to communicate with other external system.

MVC: Model View Controller, it is an architectural software design pattern for implementing user interface

GUI: Graphical User Interface

* 1. Reference Documents
* RASD of PowerEnJoy produced before
* Assignments Document AA 2016-2017.pdf
* The IEEE Standard for Information Technology – System Design – Software Design Description
  1. Document Structure

1. Architectural Design
   1. Overview

We will adopt a top down approach for the description of the architectural design of our system.

PowerEnJoy will be developed with a 3-tier architecture,

using a Client-Server architectural style.

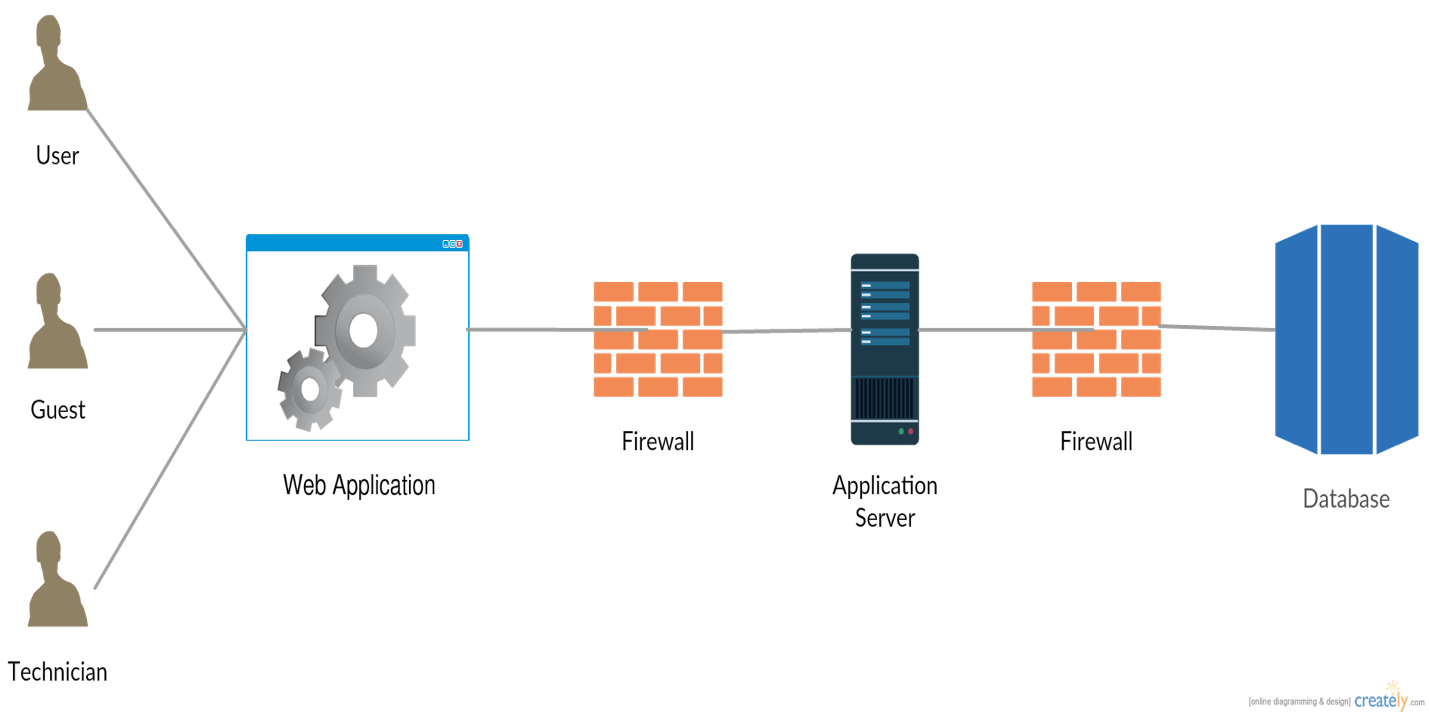


Figure 1. 3-Tier architecture

Users have access to a GUI to interface with the server and take advantage of the functionalities dedicated to them. The server will receive requests from users and will have access to the database, so it can use it to save or read data if necessary.  
To have a clearer outline of the high-level system we can see the functionalities divided into subsystems as shown in the figure below



Figure 2. High-level functional system view

* 1. High-Level Component view

There are fundamentally four components in our system. The main component is the server, it is the engine of the system. Customers and technicians send their requests to the server which elaborates a response and sends it to them.

The server, to response to the requests, must be able to communicate with external systems and with the database.

There are three external systems to help the server: there is a payment system that handles all requests regarding user fees. The server may request the external payment system to verify that the user data provided during registration are correct. At time of payment the server will send to the payment system the charging request to the user and this will provide to do so, in the case where the payment system is not able to execute the charging, it will report it to the server and the user will be moved on the black list.

The vehicles are provided of an external service that monitors and communicates to the server all the necessary information that it needs, such as the location of vehicles and the battery info.

The last external service used by the server is a mapping service.

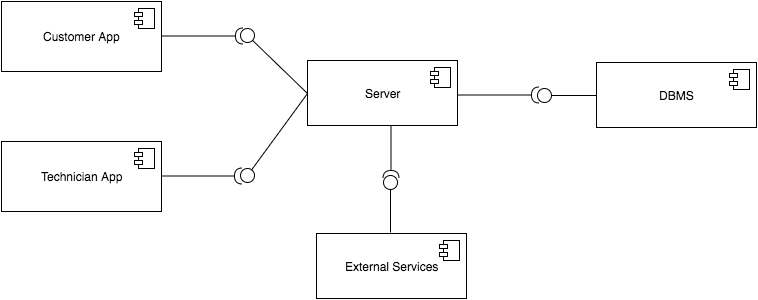


Figure 3. High-Level Component Diagram

* 1. Component View



Figure 4. Component Diagram

The Component diagram summarizes in a more detailed way what we have already explained. It is also made explicit the internal division in the server of the application modules.

Through this class diagram we explain how it will be organized the database to which the server will have access. In the database will be stored all the information of that we will need to keep track.

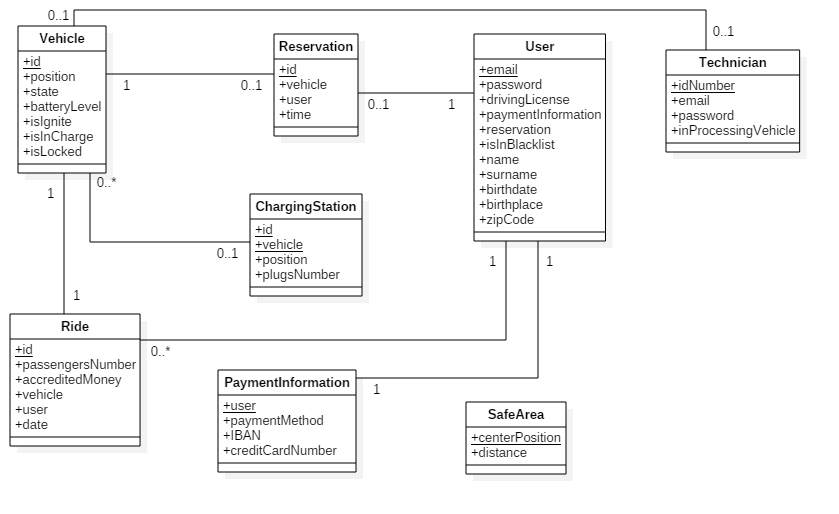


Figure 5. Class diagram representing database logic

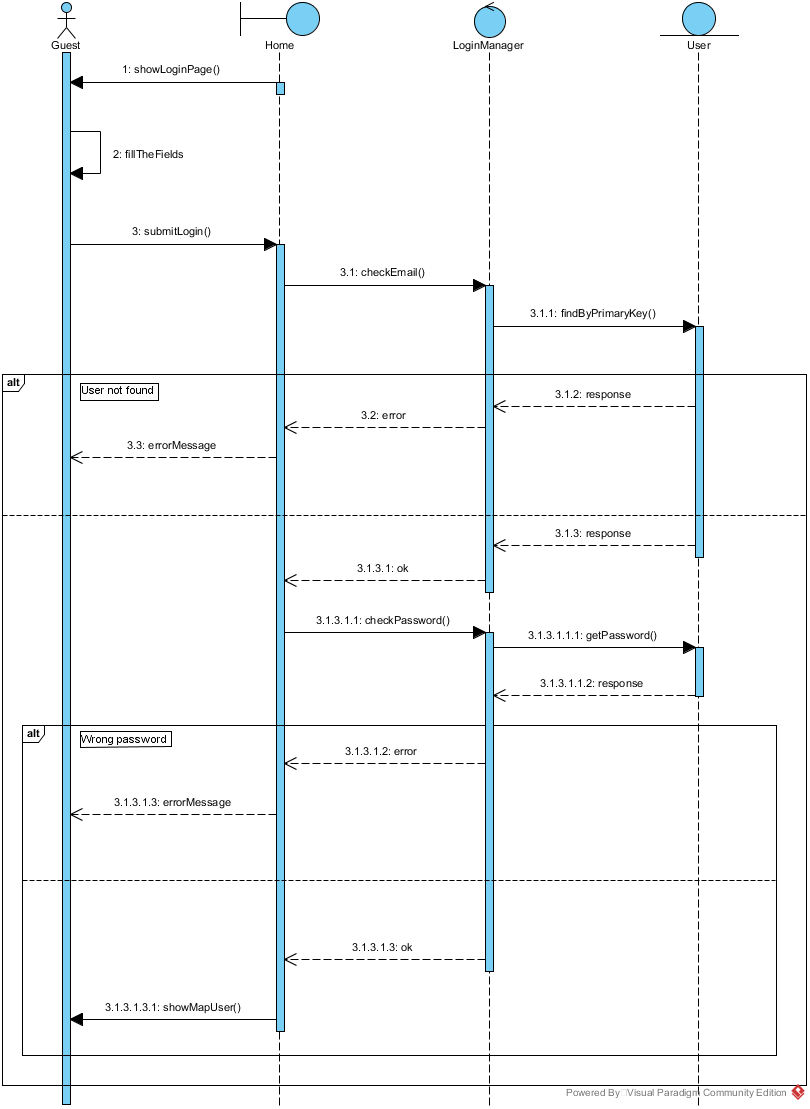
2.4 Deployment View

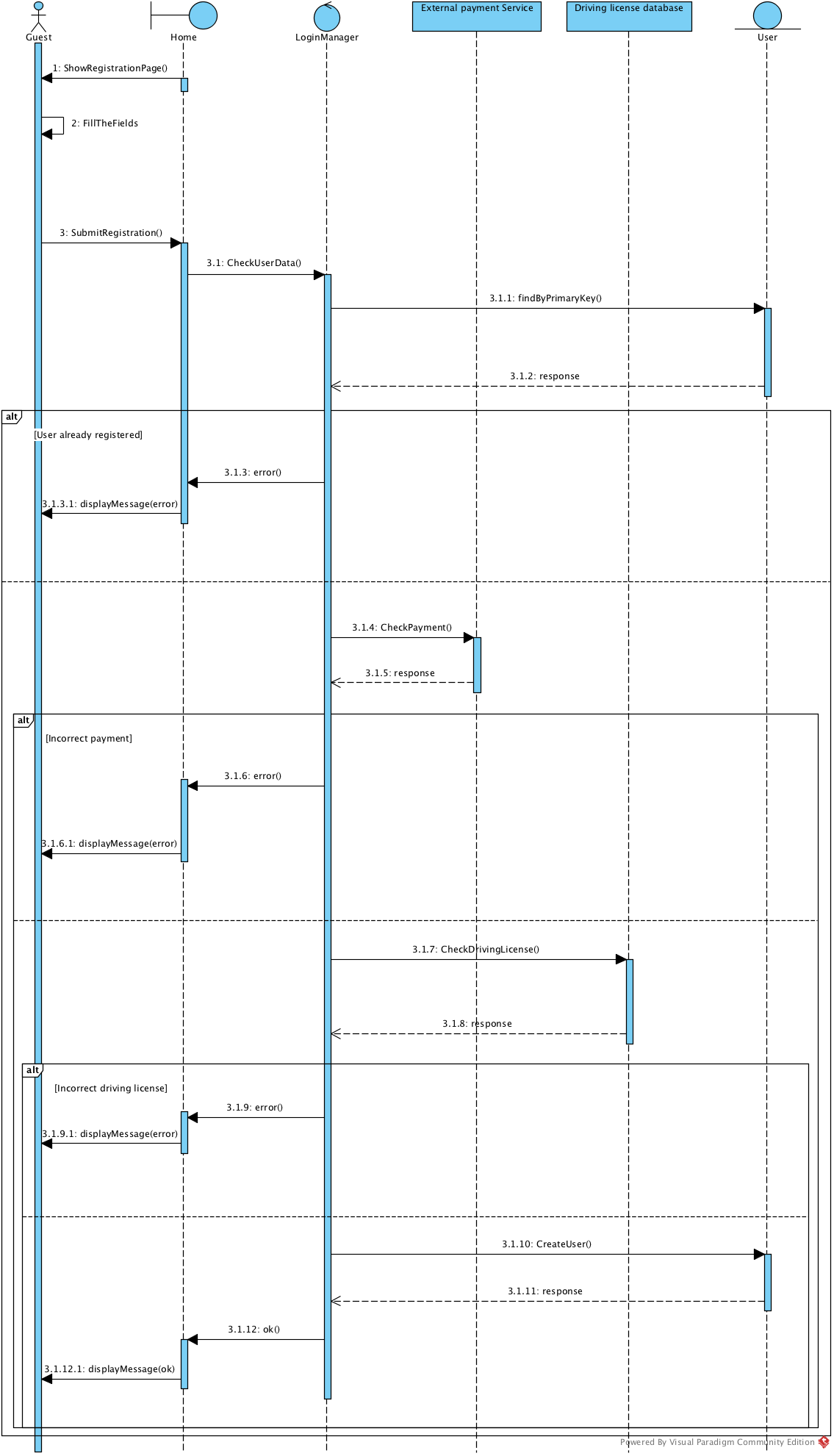


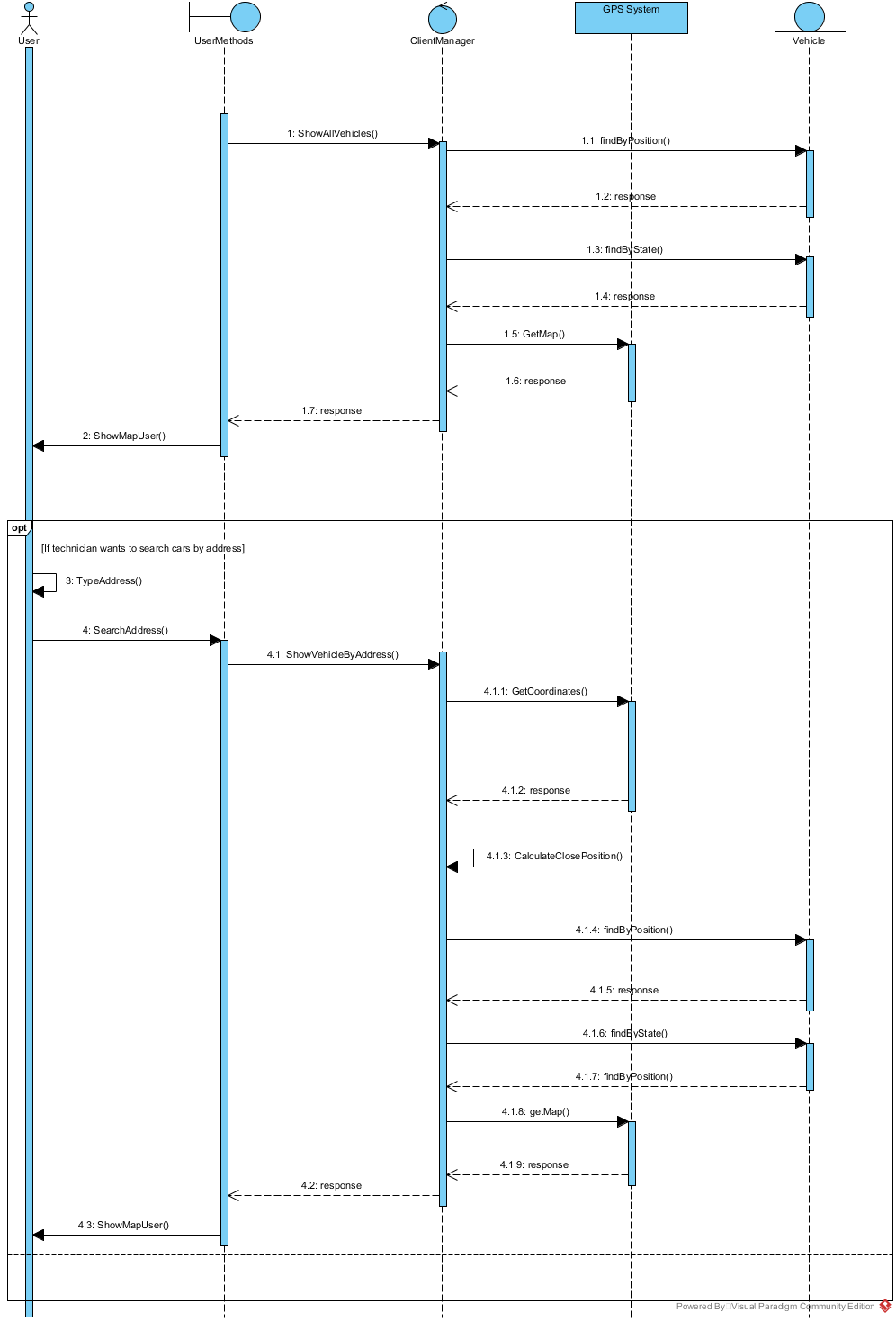
Figure 6. Deployment diagram of the system

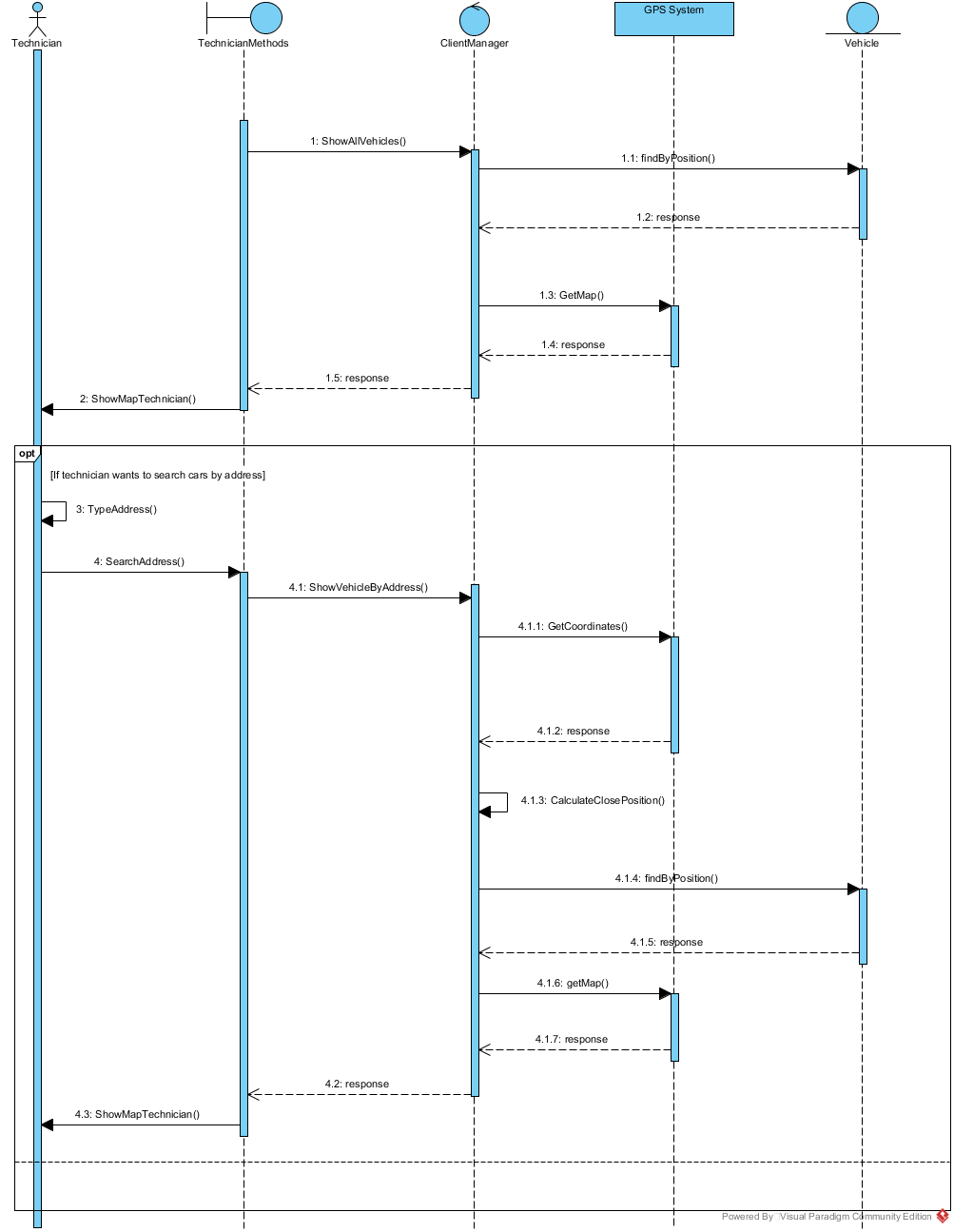
* 1. Runtime View

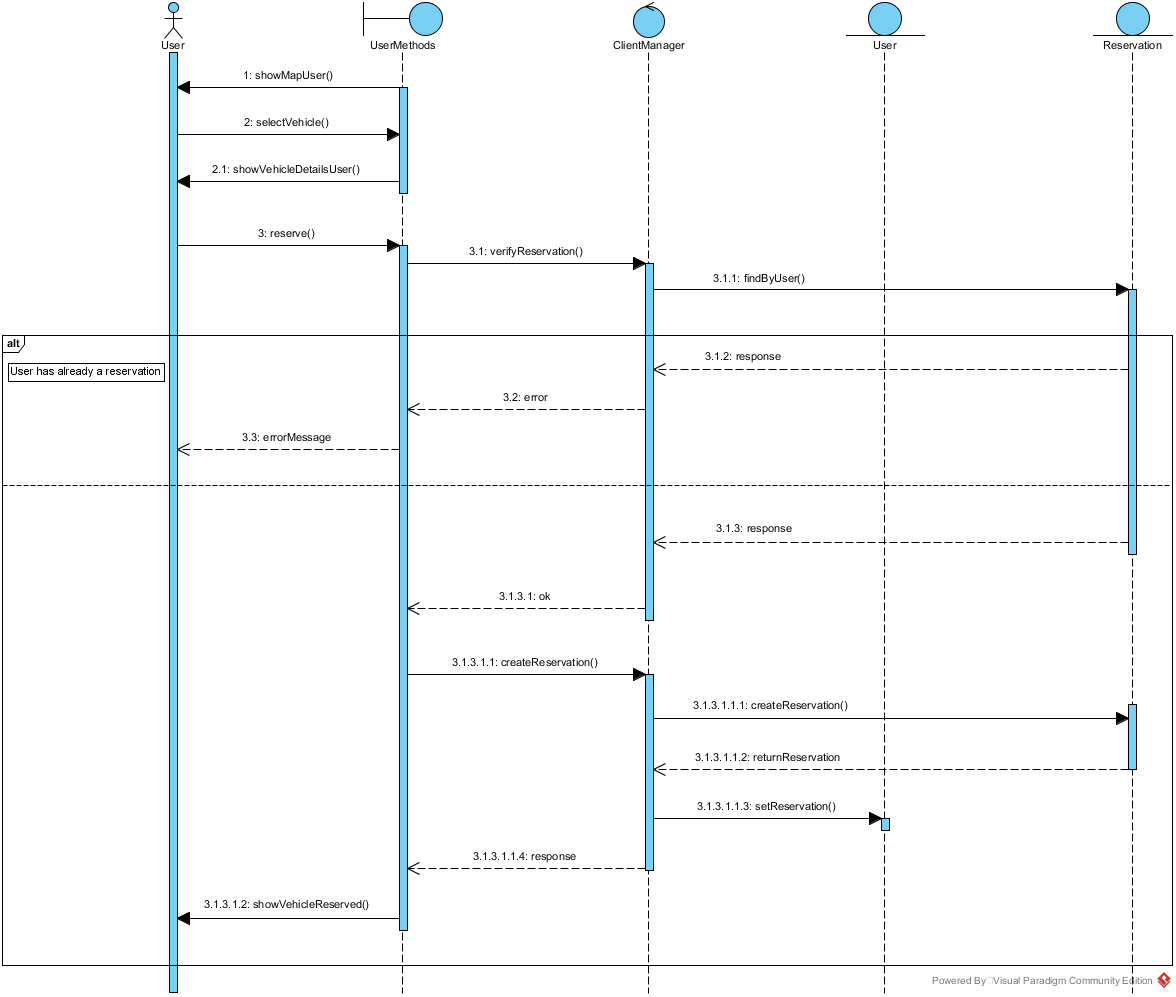
Now we describe the dynamic behavior of the system in most relevant case.

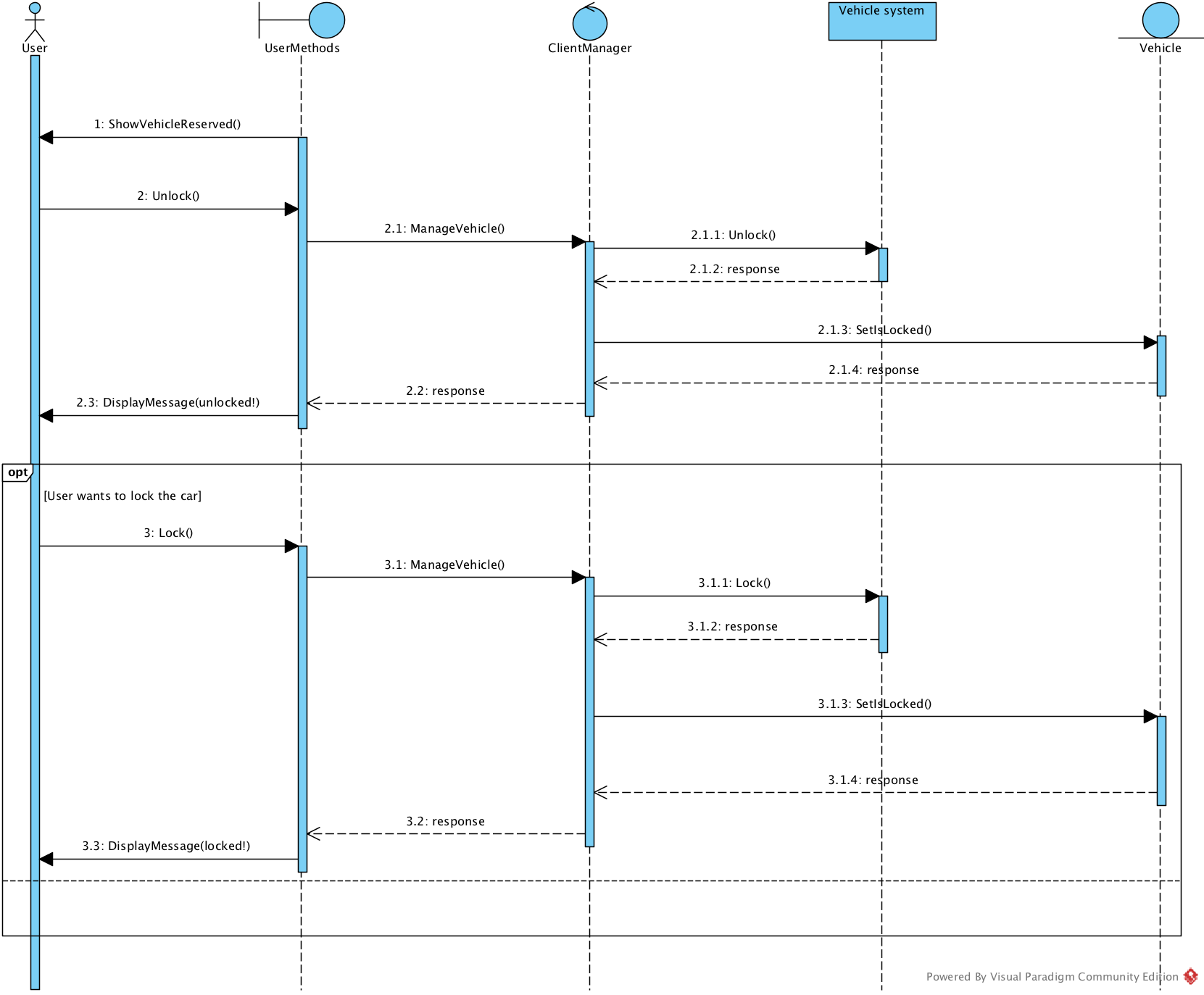


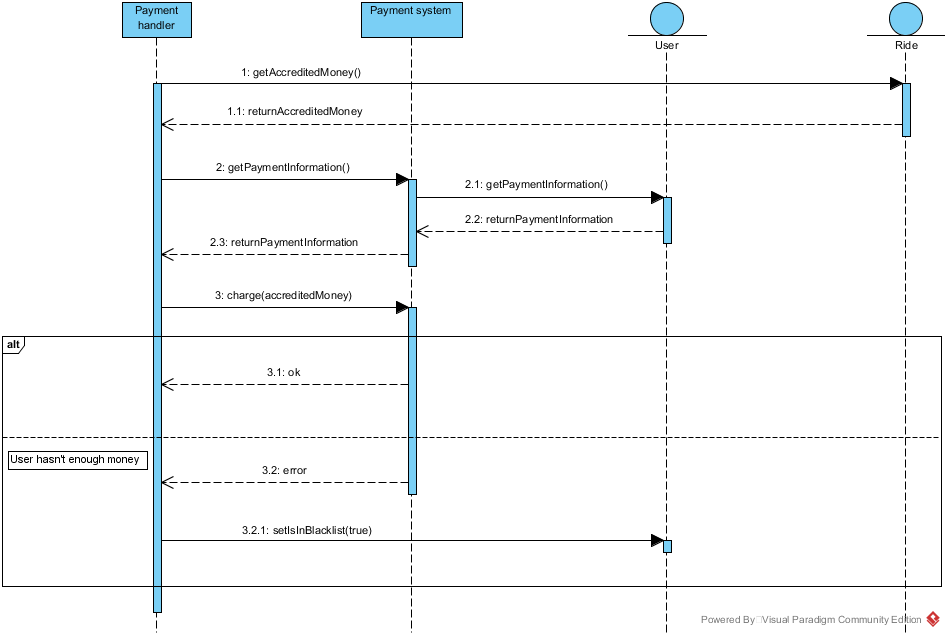


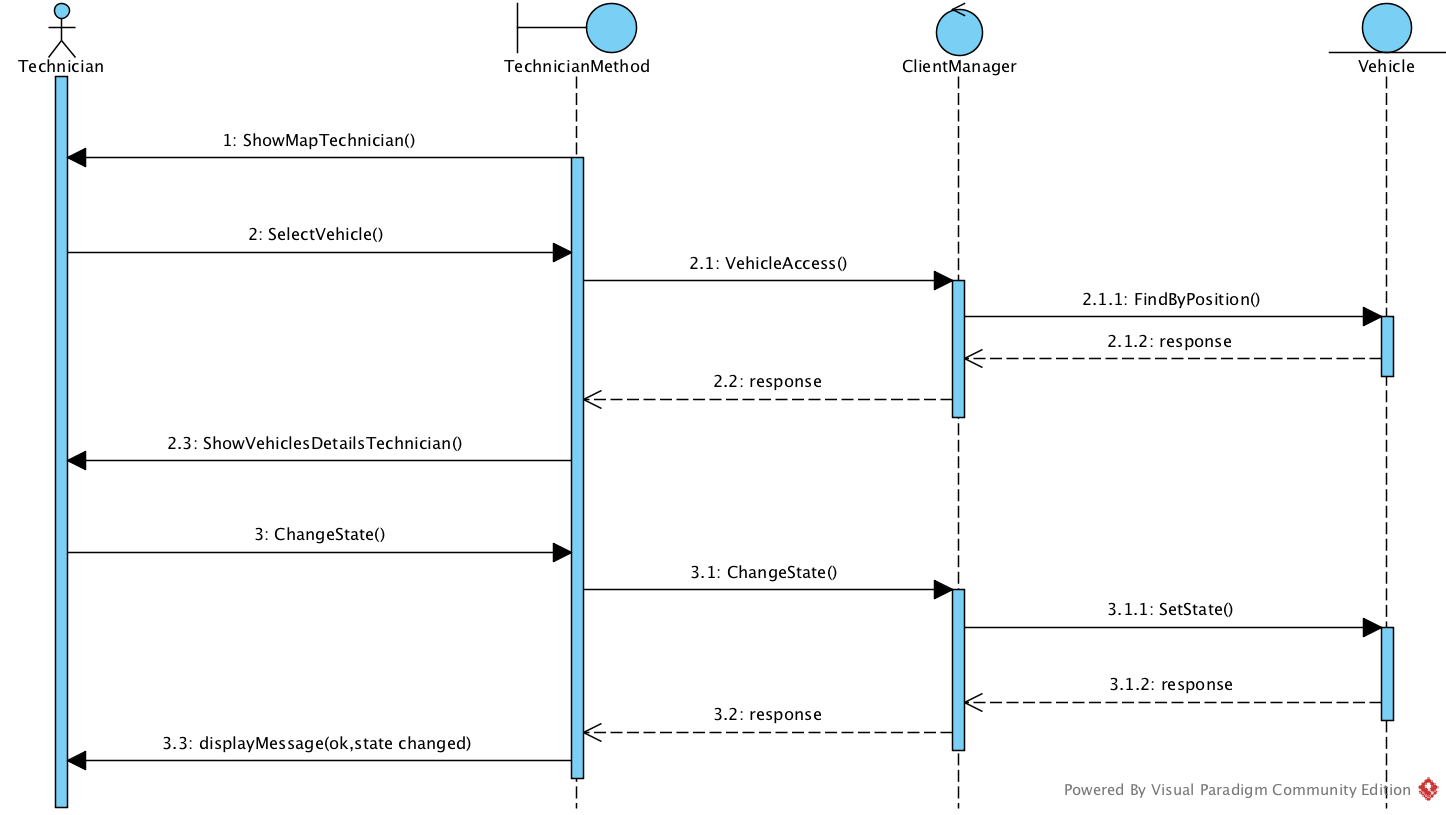












* 1. Selected architectural styles and patterns

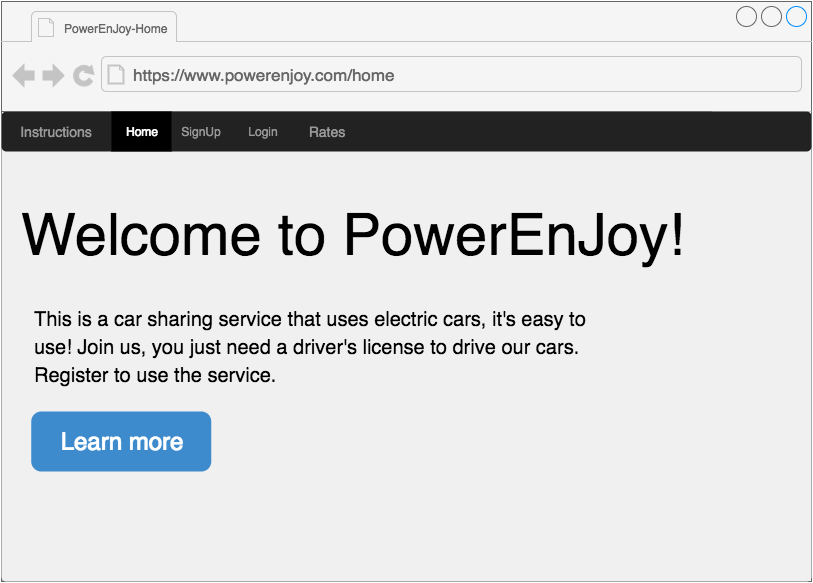
1. Algorithm design

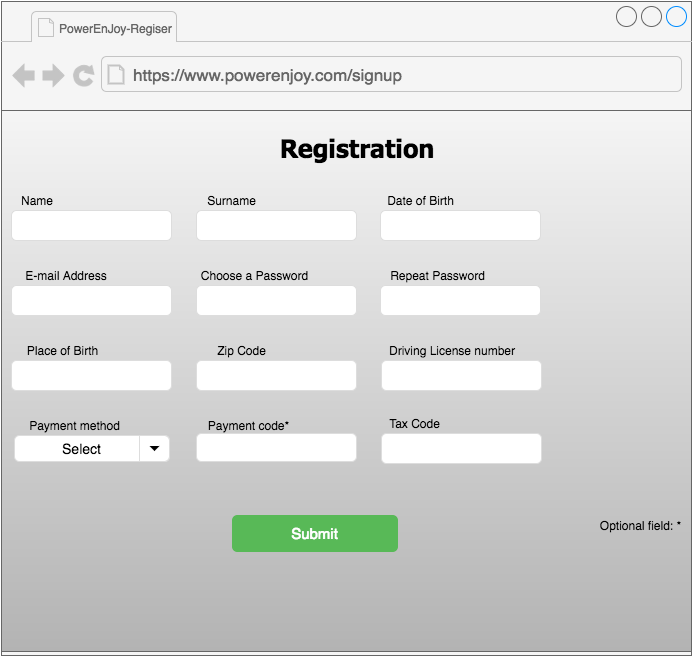
To develop this software, there are no particularly complex algorithms, and there are no particular constraints for developers. The important thing is that the code works and meets the functional requirements described in RASD. The algorithm that calculates the charge for the user must, of course, comply with all the rules that have been explained in the RASD

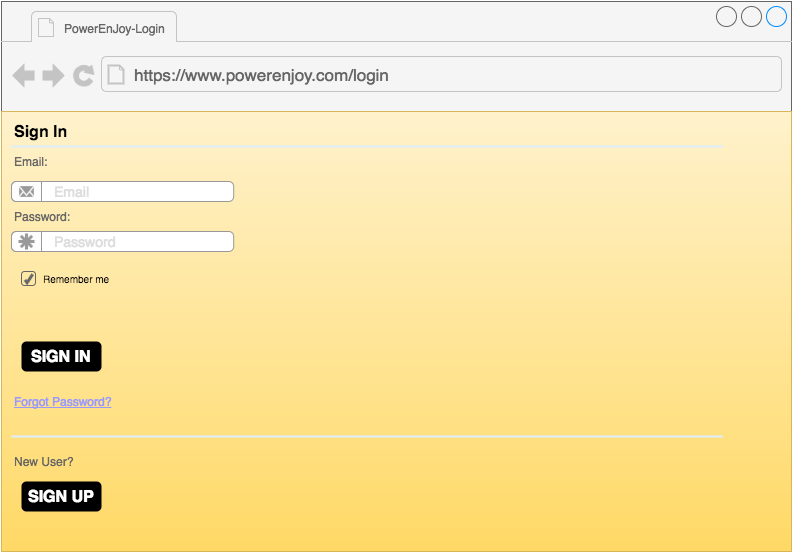
1. User Interface Design
   1. Mockups

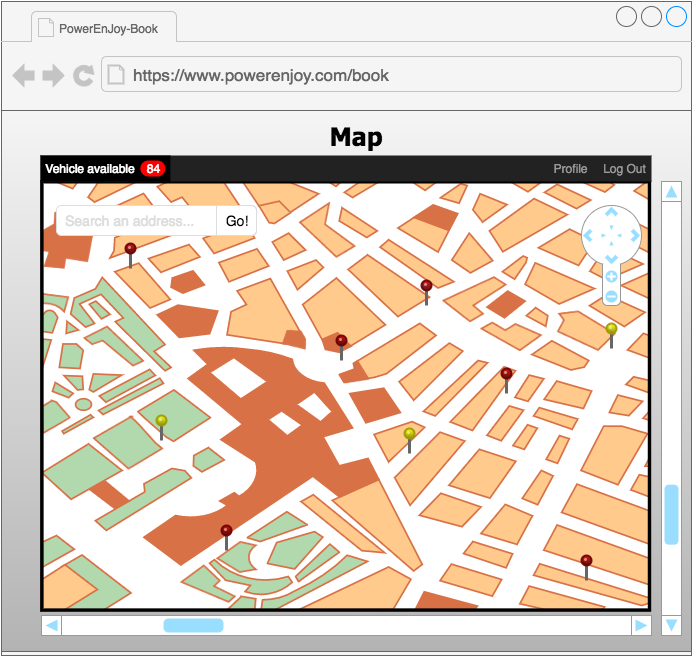
We provided some pages of our web app that are useful to understand how users will interact with the system.

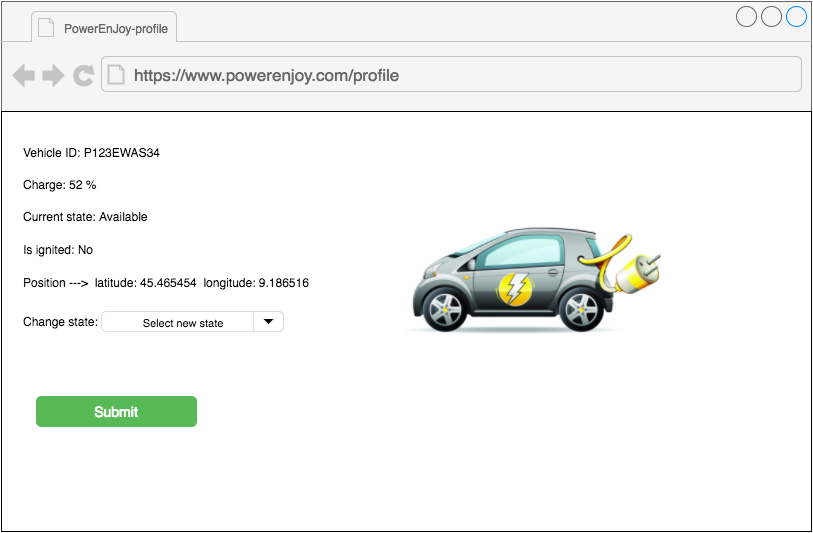
So these sketches may be improved by those who will design the graphical interface.



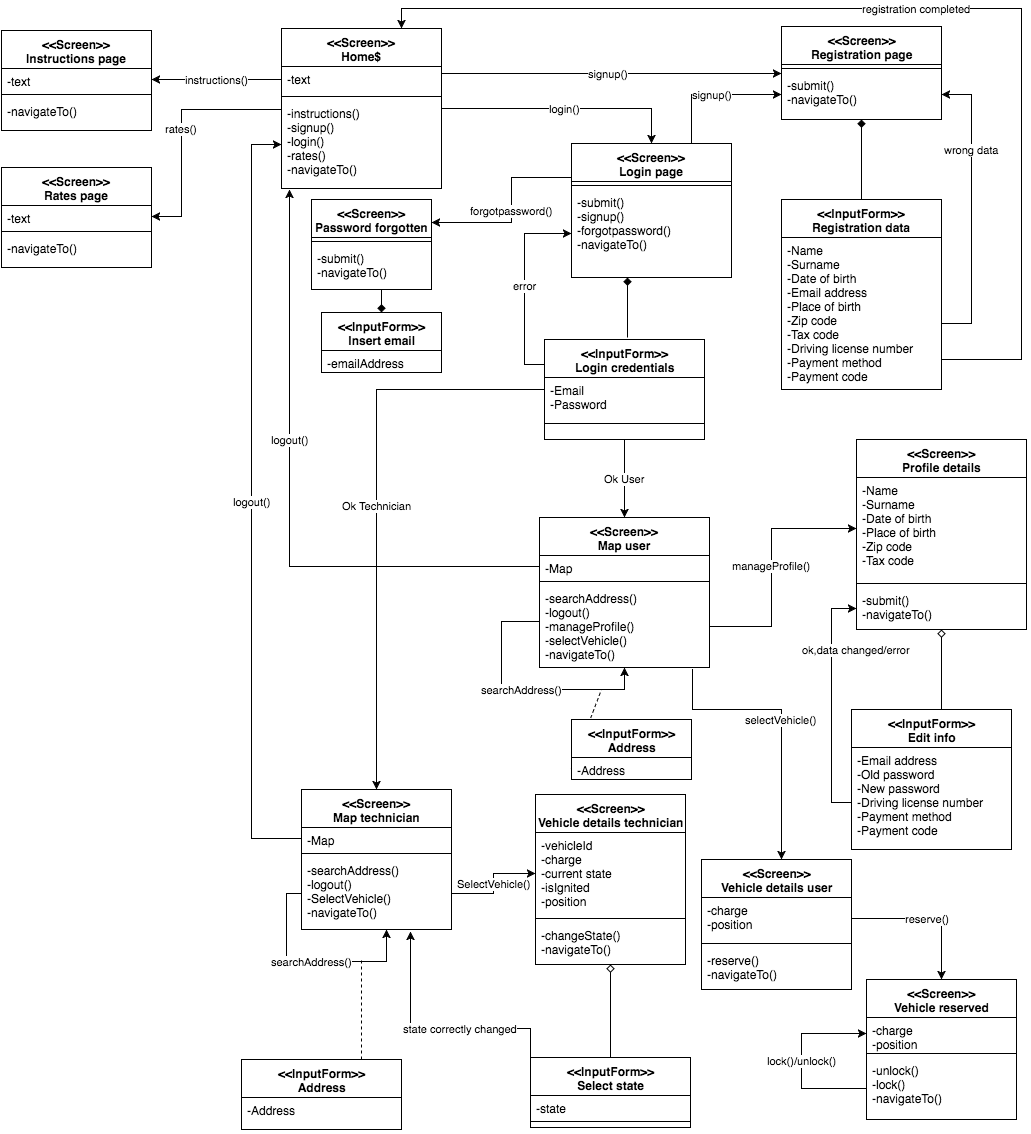




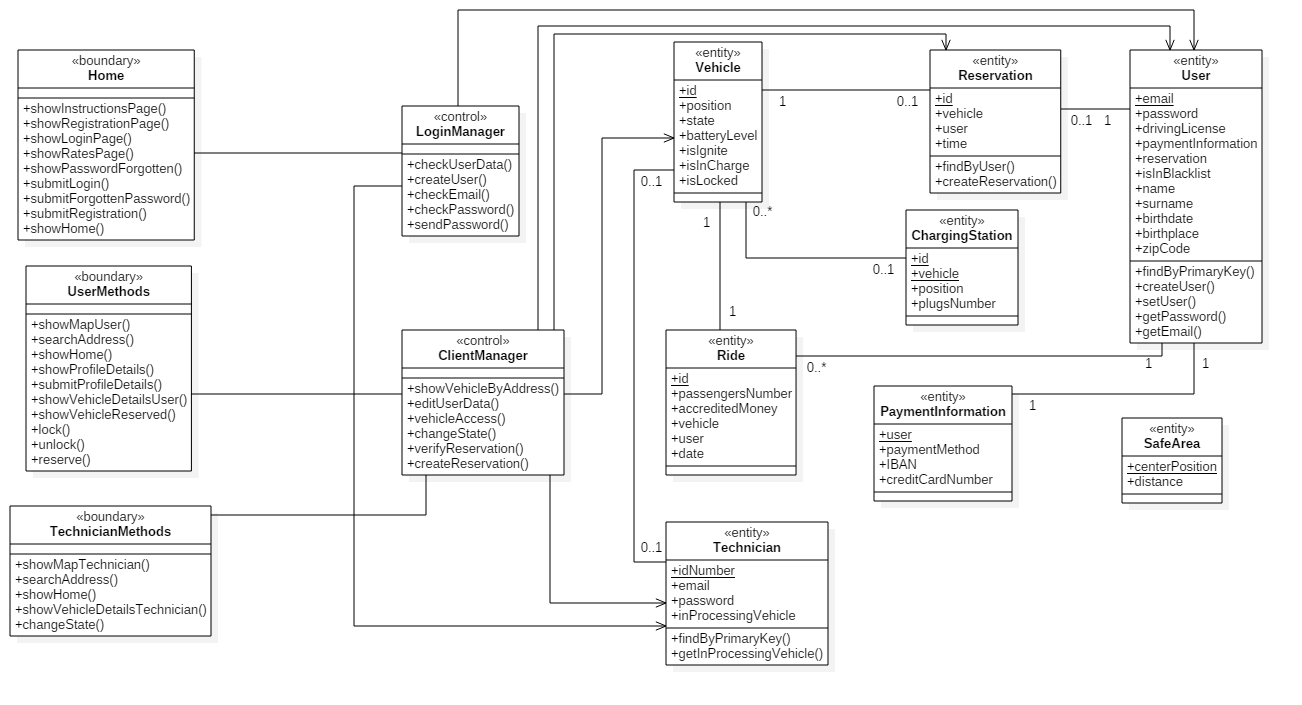




* 1. UX Diagram



* 1. BCE Diagram



1. Requirements traceability
2. References
   1. Used Tools

* GitHub
* Microsoft Word
* Draw.io
* Visual Paradigm community edition
* Star UML

1. Effort Spent