

Design Document



POLITECNICO
MILANO 1863

Version 1.0

Luca Santini	808710
Riccardo Remigio	874939

Index

1. INTRODUCTION	3
1.1 PURPOSE	3
1.2 SCOPE	3
1.3 DEFINITION, ACRONYMS, ABBREVIATIONS	4
1.4 REFERENCE DOCUMENTS	4
1.5 DOCUMENT STRUCTURE	5
2. ARCHITECTURAL DESIGN.....	6
2.1 OVERVIEW	6
2.2 HIGH-LEVEL COMPONENT VIEW	7
2.3 COMPONENT VIEW	8
2.4 DEPLOYMENT VIEW	10
2.5 RUNTIME VIEW.....	11
2.6 SELECTED ARCHITECTURAL STYLES AND PATTERNS	19
3. ALGORITHM DESIGN	21
4. USER INTERFACE DESIGN	21
4.1 MOCKUPS.....	21
4.2 UX DIAGRAM	27
4.3 BCE DIAGRAM	28
5. REQUIREMENTS TRACEABILITY	29
6. REFERENCES	30
6.1 USED TOOLS	30
HOURS OF WORK	31
CHANGELOG.....	31

1.Introduction

1.1 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.

1.2 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.

1.3 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

1.4 Reference Documents

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

1.5 Document Structure

- **Introduction:** this section introduces the design document. It contains a justification of his utility and indications on which parts are covered in this document that are not covered by RASD
- **Architecture Design:** this section is divided into different parts:
 1. Overview: this sections explains the division in tiers of our application
 2. High level component view: this sections gives a global view of the components of the application and how they communicate
 3. Component view: this sections gives a more detailed view of the components of the applications
 4. Deployment view: this section shows the components that must be deployed to have the application running correctly
 5. Runtime view: sequence diagrams are represented in this section to show the course of the different tasks of our application
 6. Selected architectural styles and patterns: this section explains the architectural choices taken during the creation of the application
- **Algorithms Design:** this section describes the most critical parts via some algorithms.
- **User Interface Design:** this section presents mockups and user experience explained via UX and BCE diagrams.
- **Requirements Traceability:** this section aims to explain how the decisions taken in the RASD are linked to design elements.

2. Architectural Design

2.1 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.

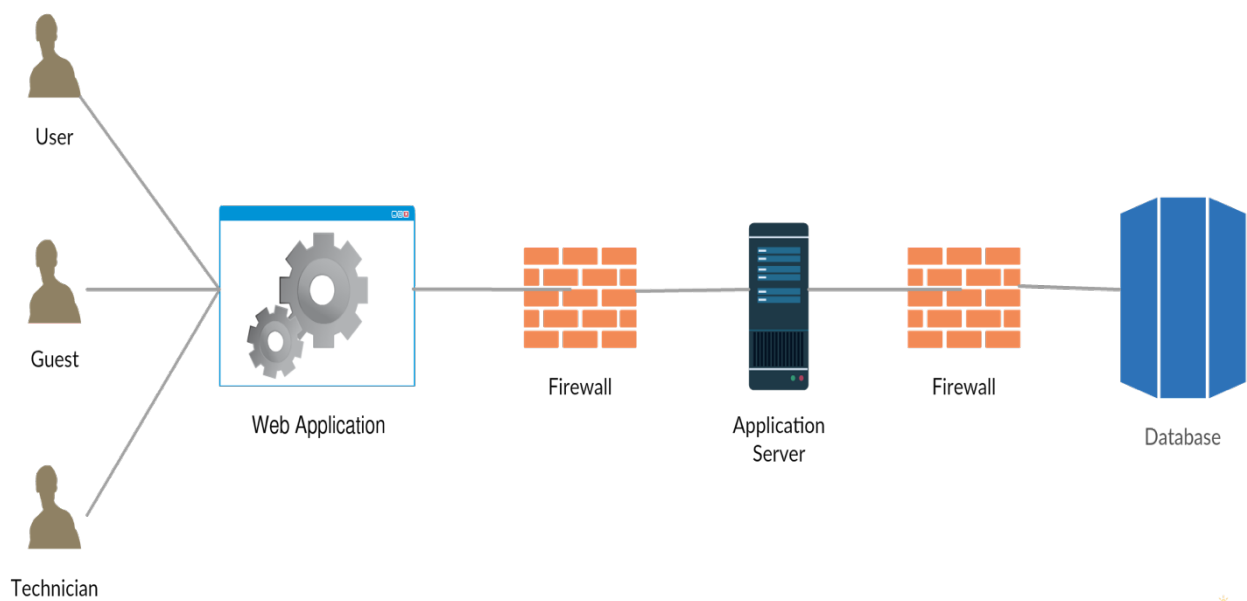


Figure 1. 3-Tier architecture

Users have access to a web site 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

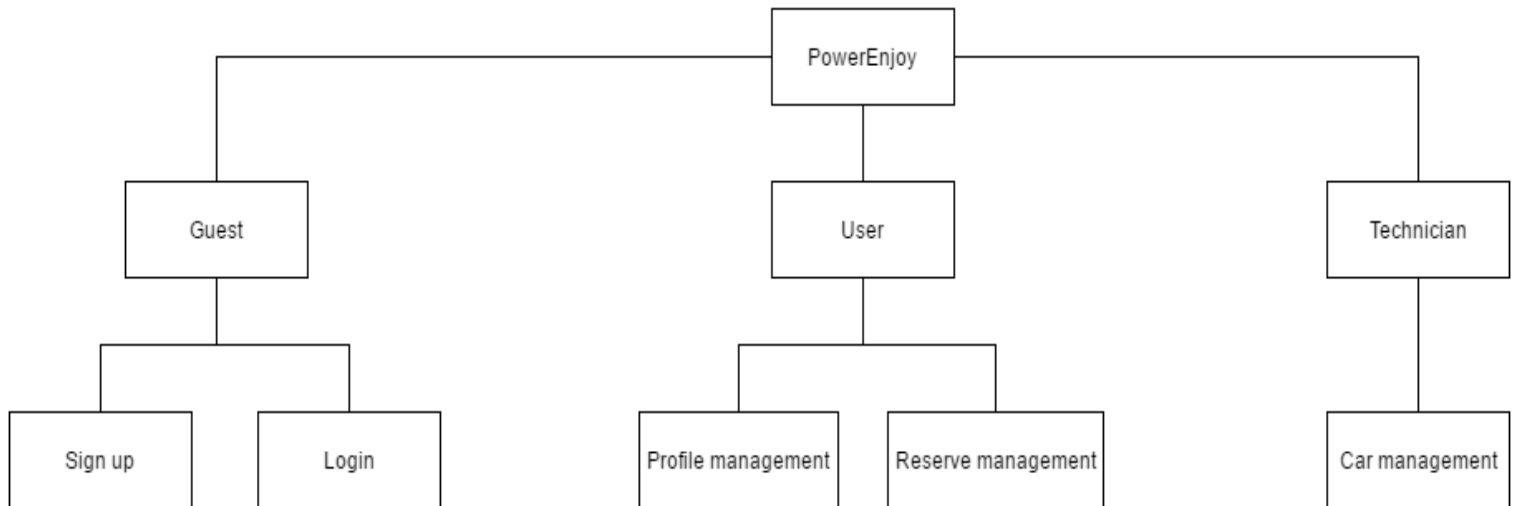


Figure 2. High-level functional system view

2.2 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 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.

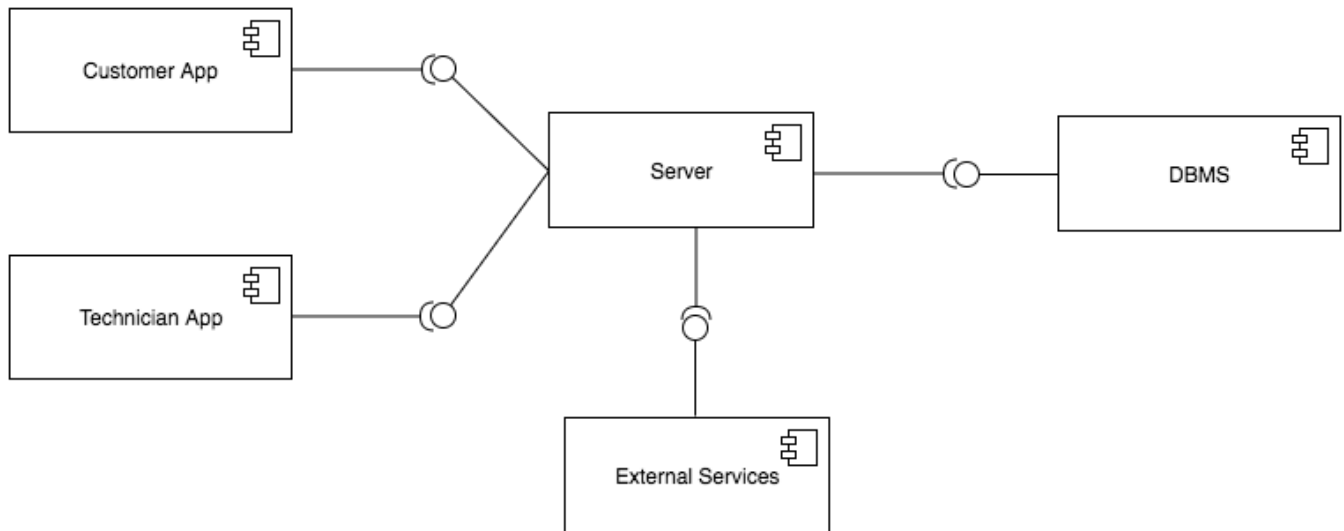


Figure 3. High-Level Component Diagram

2.3 Component View

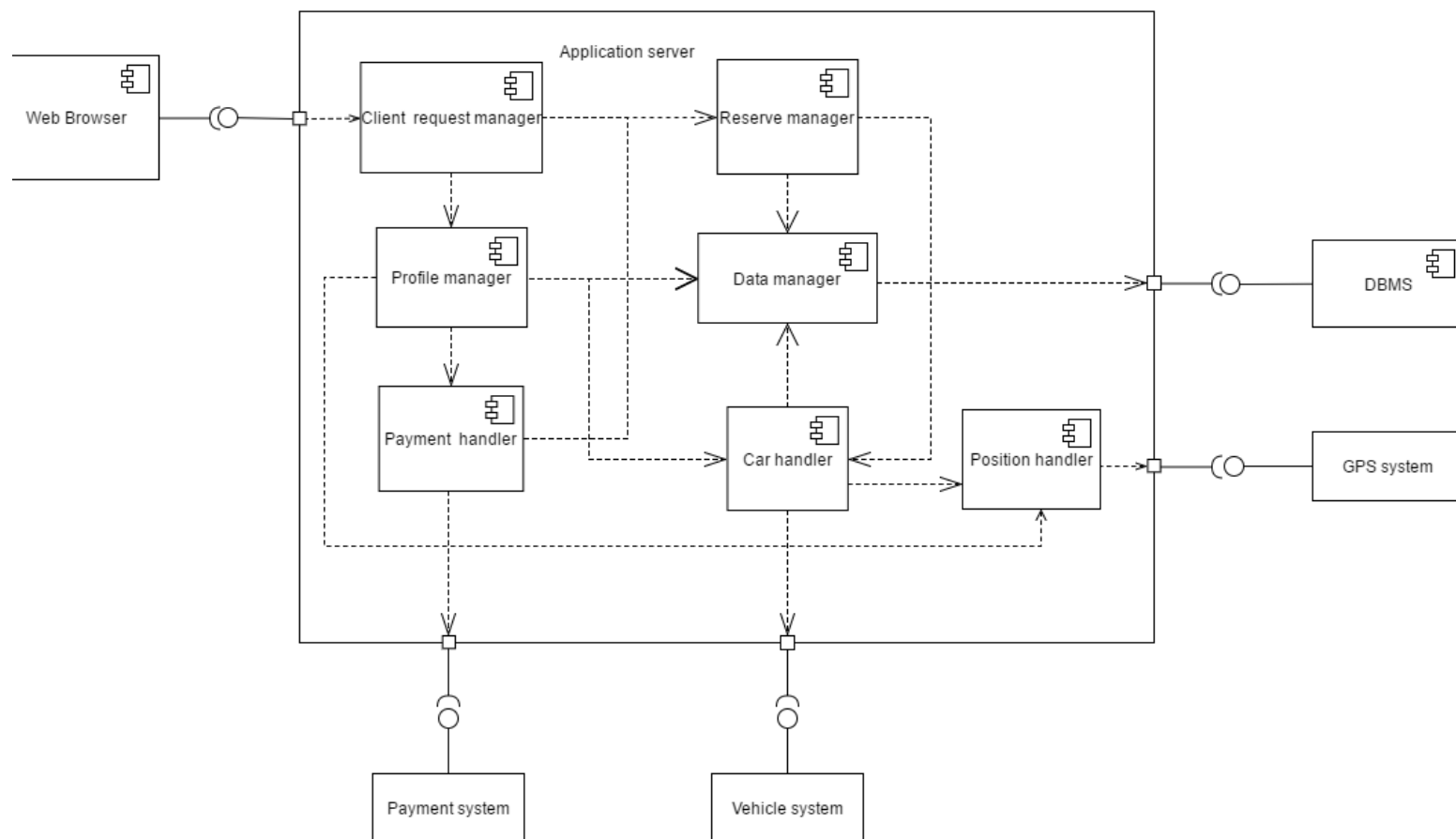


Figure 4. Component Diagram

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.

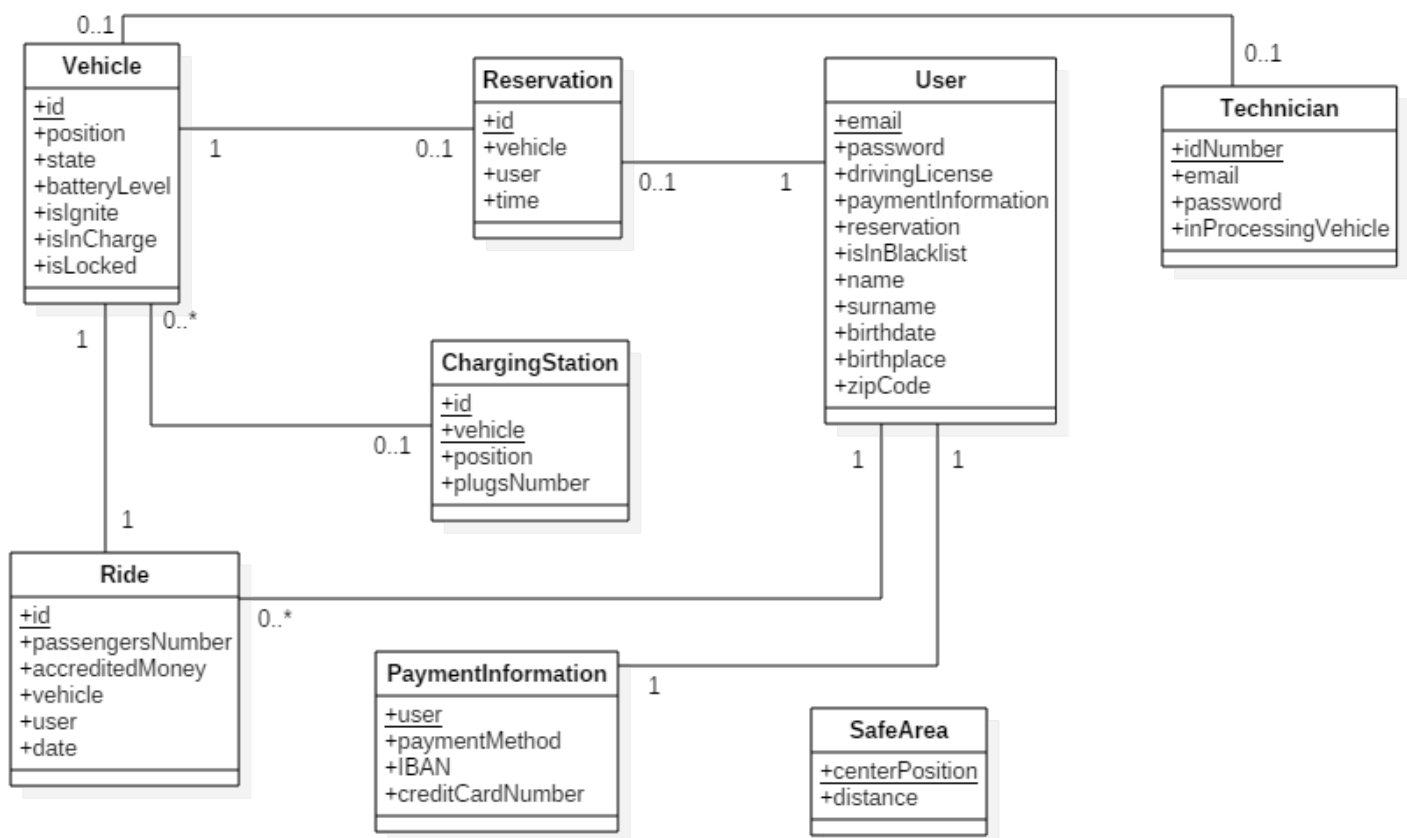


Figure 5. Class diagram representing database logic

2.4 Deployment View

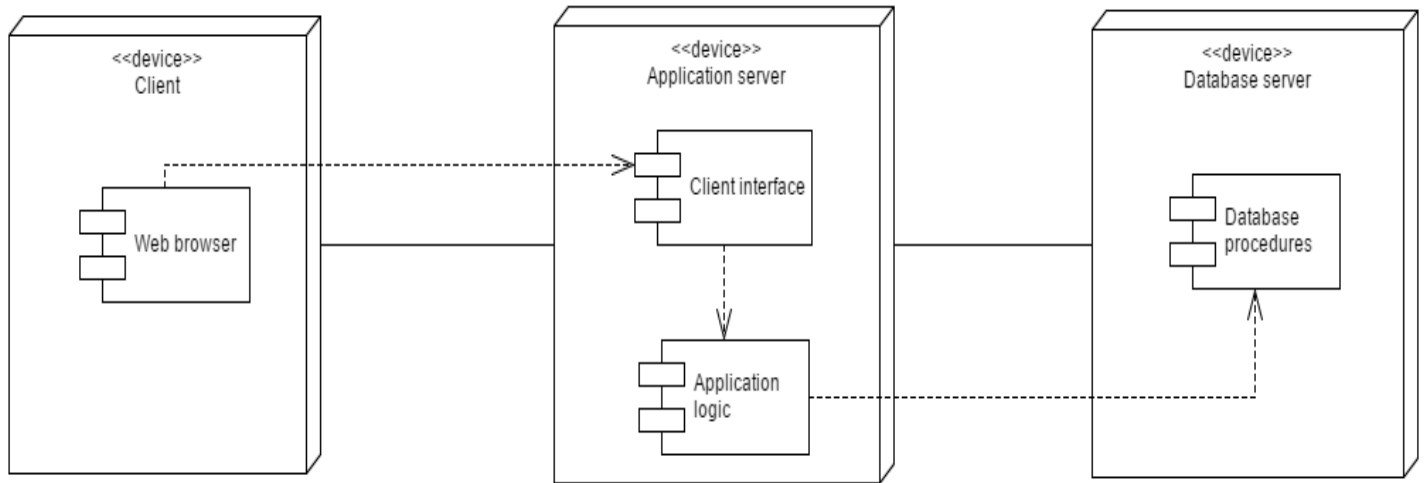
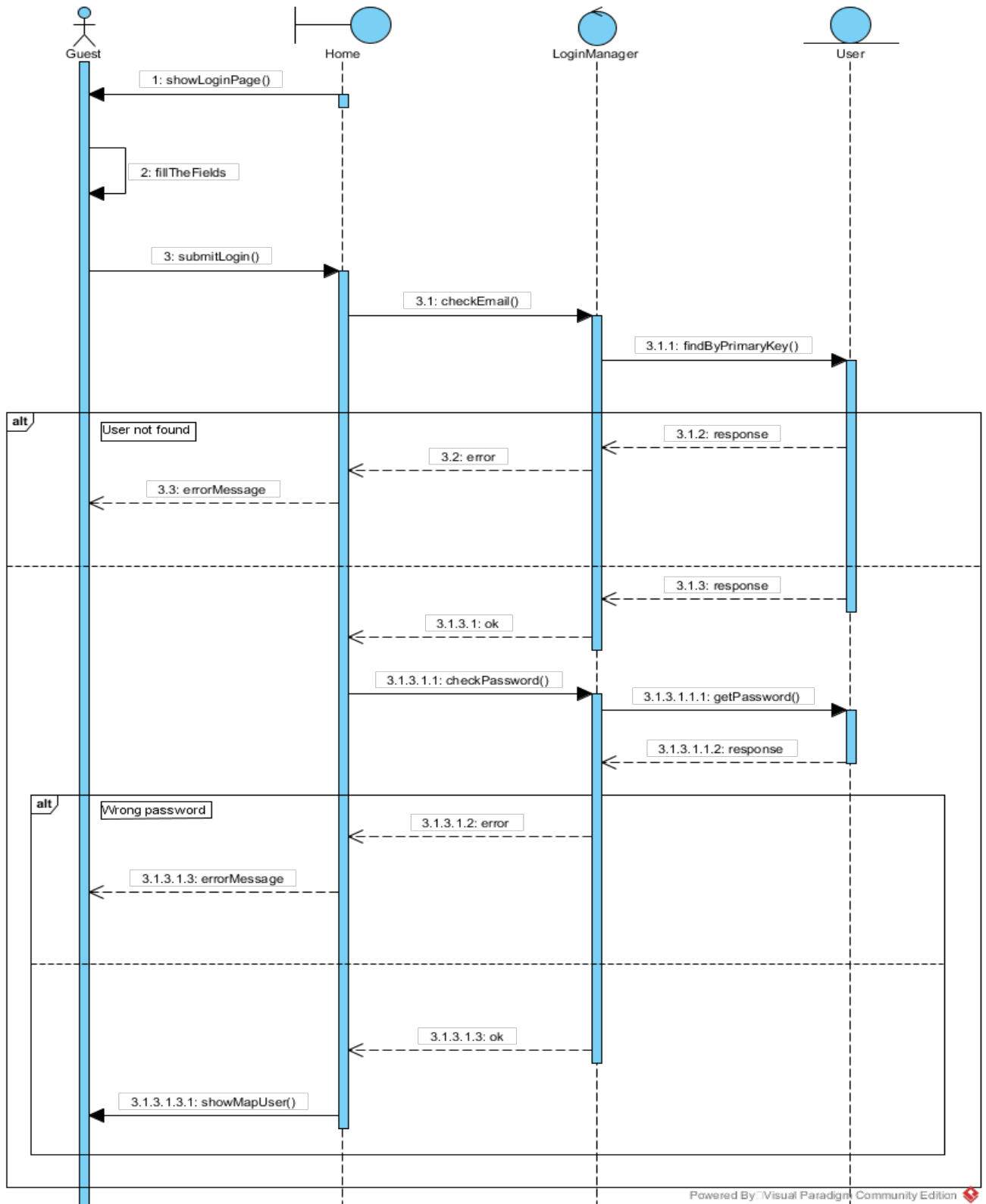


Figure 6. Deployment diagram of the system

2.5 Runtime View

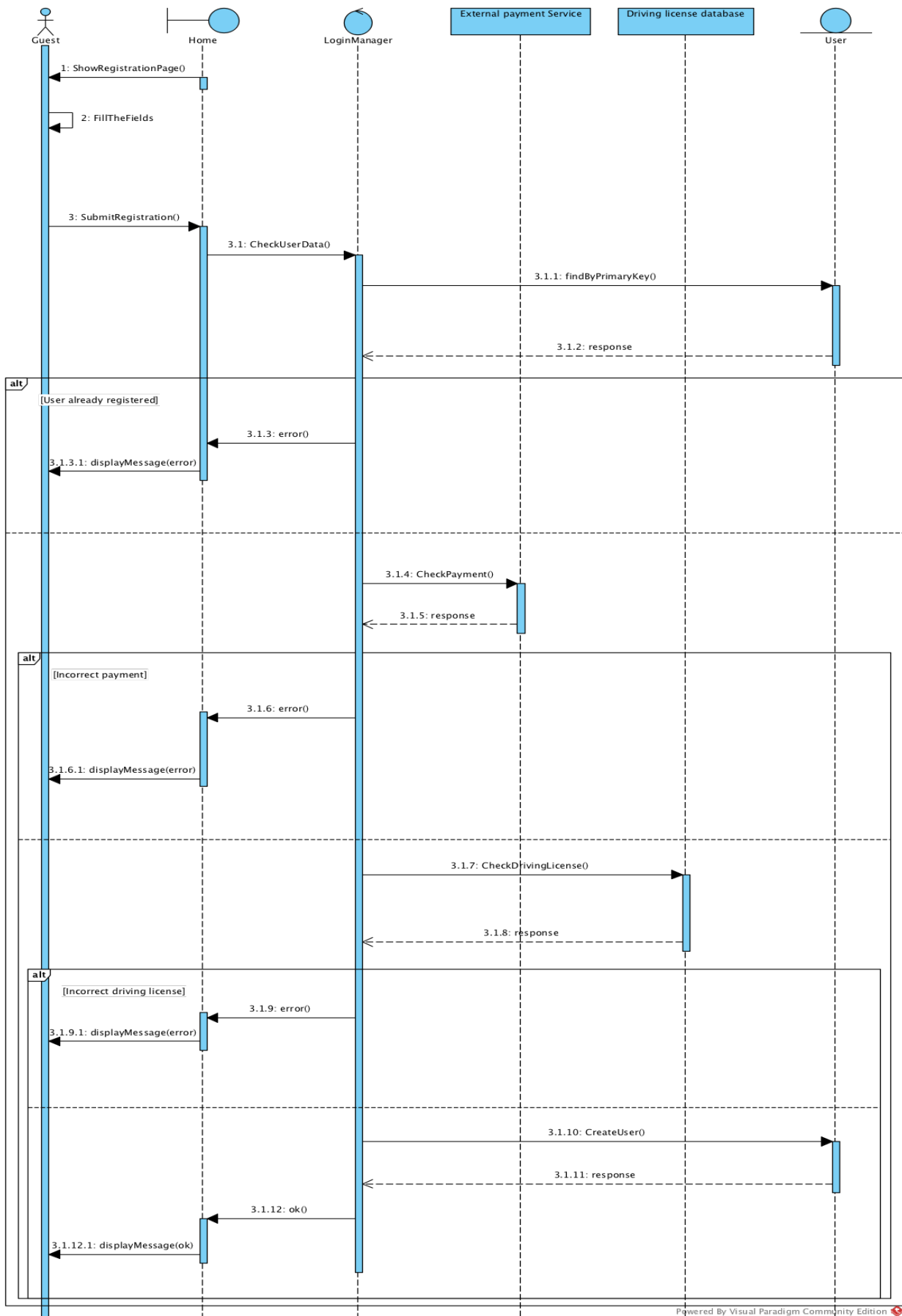
Now we describe the dynamic behavior of the system in most relevant case.

Login sequence diagram



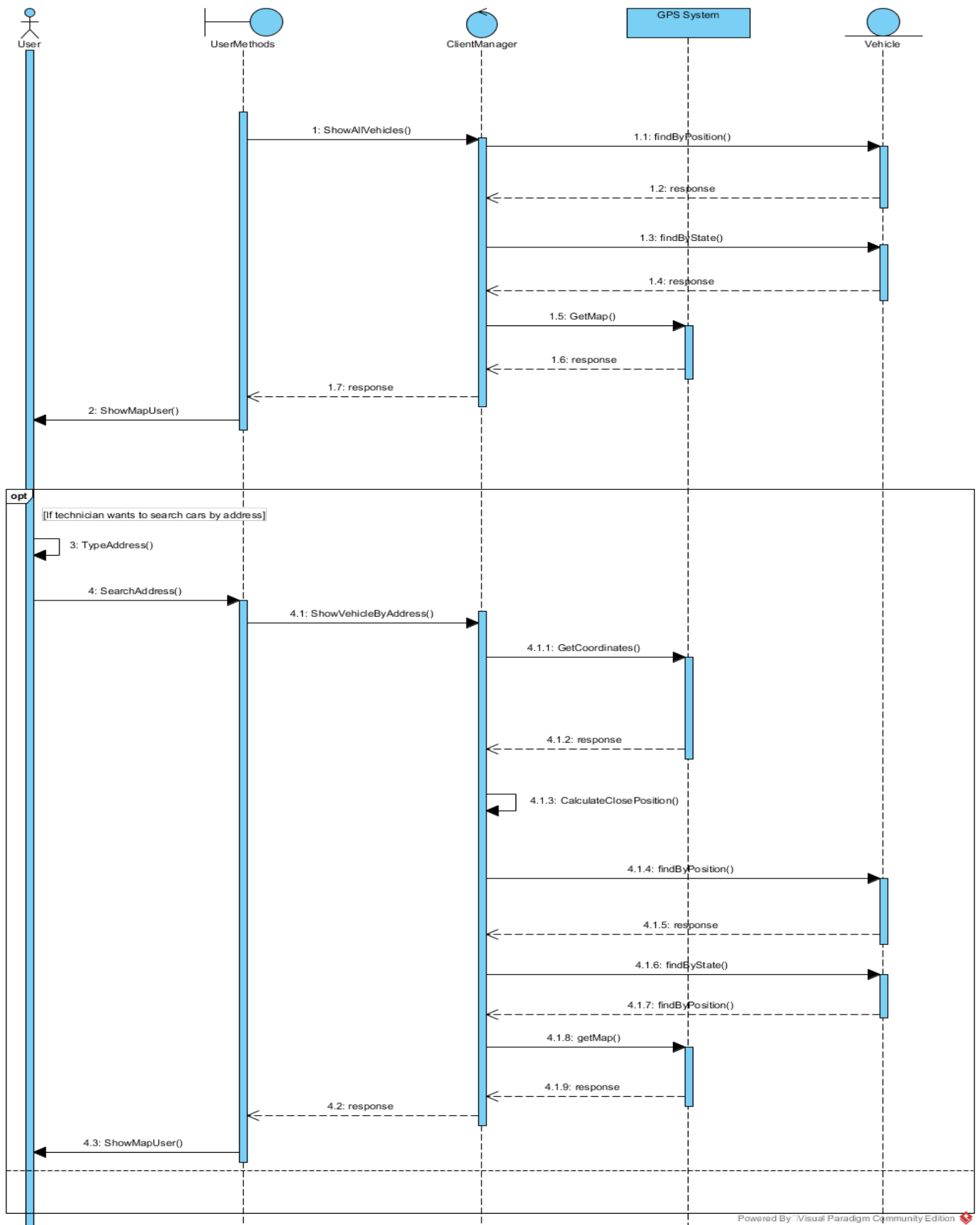
Powered By: Visual Paradigm Community Edition

Registration sequence diagram

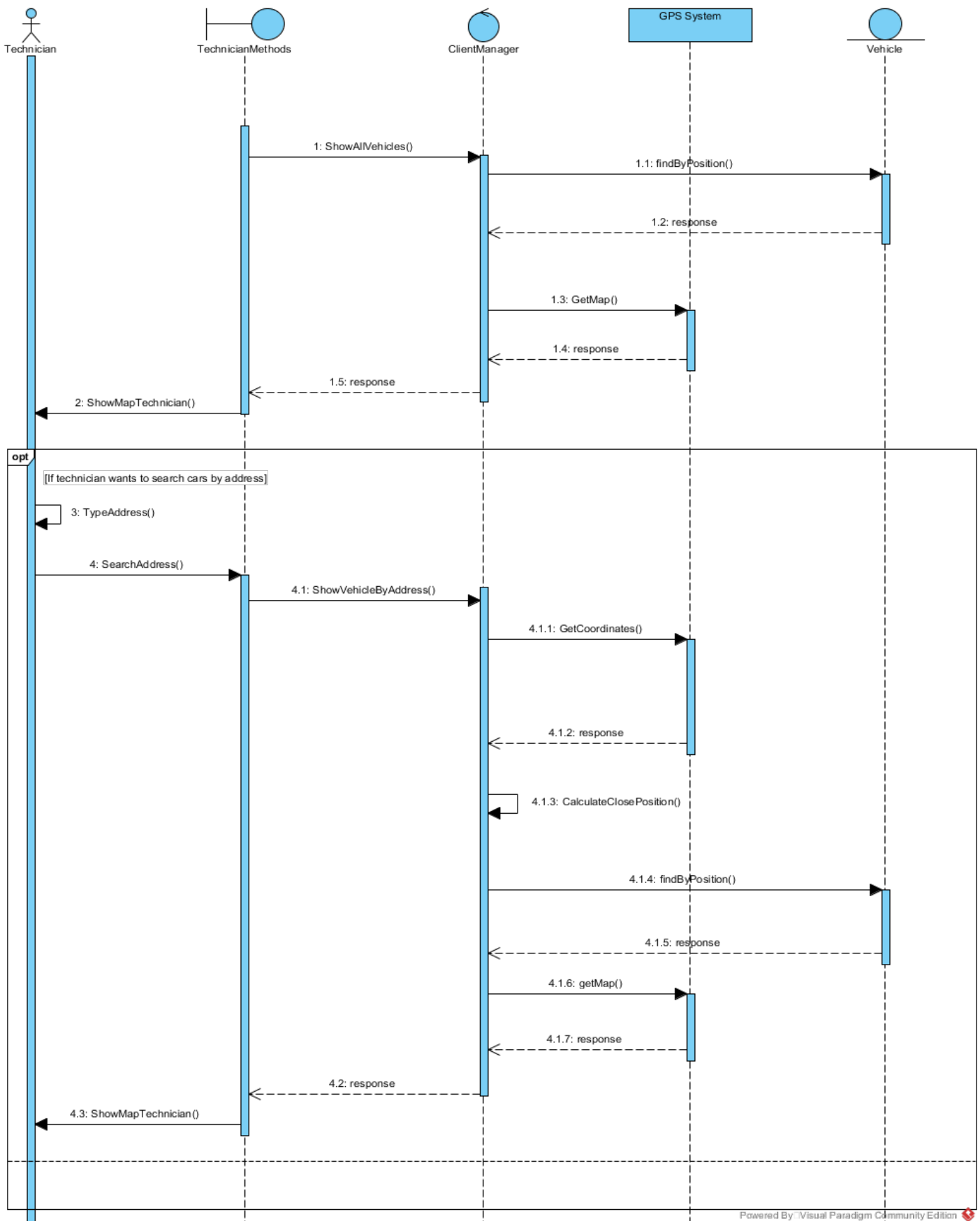


Powered By Visual Paradigm Community Edition

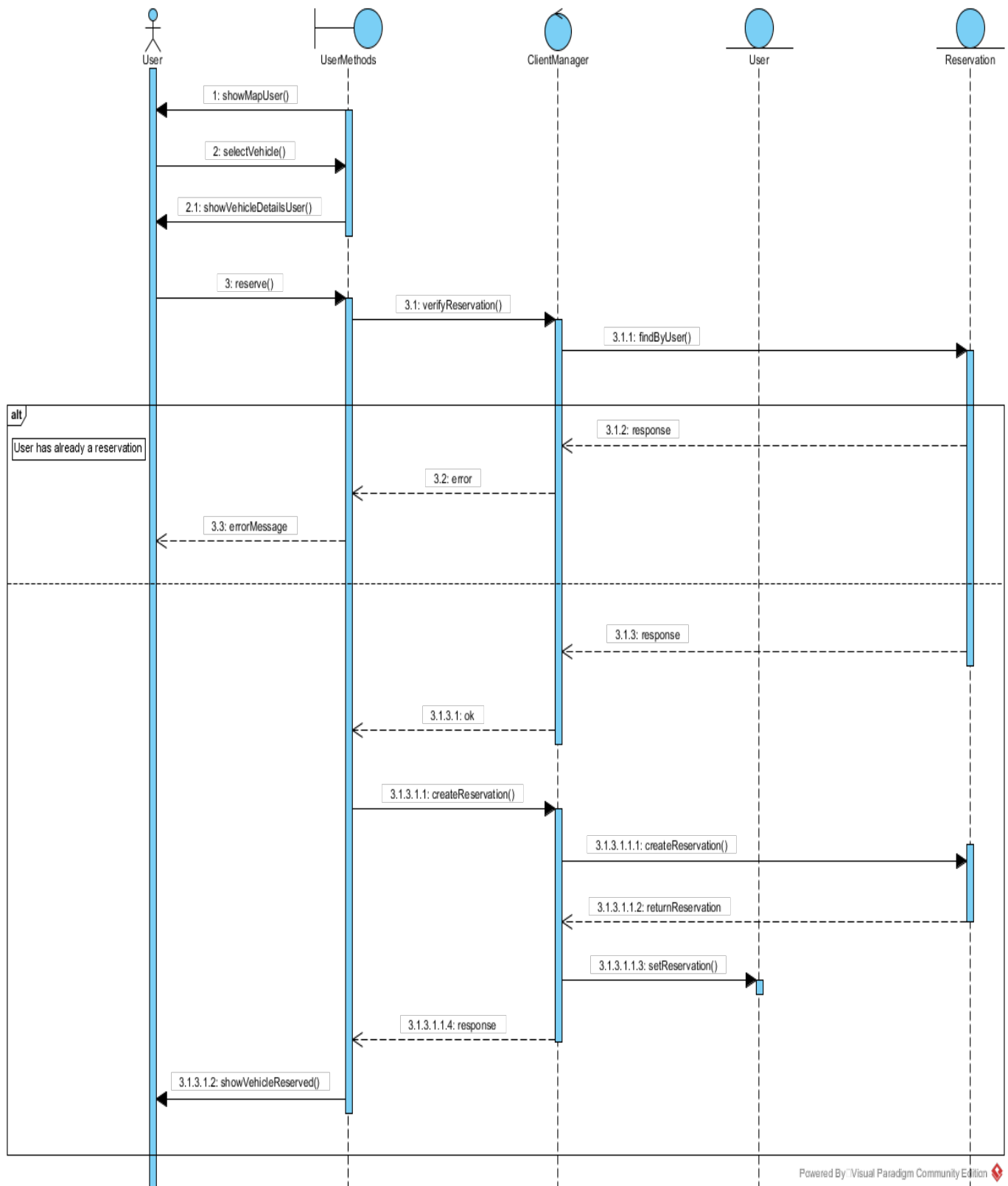
Show vehicles positions(user) sequence diagram



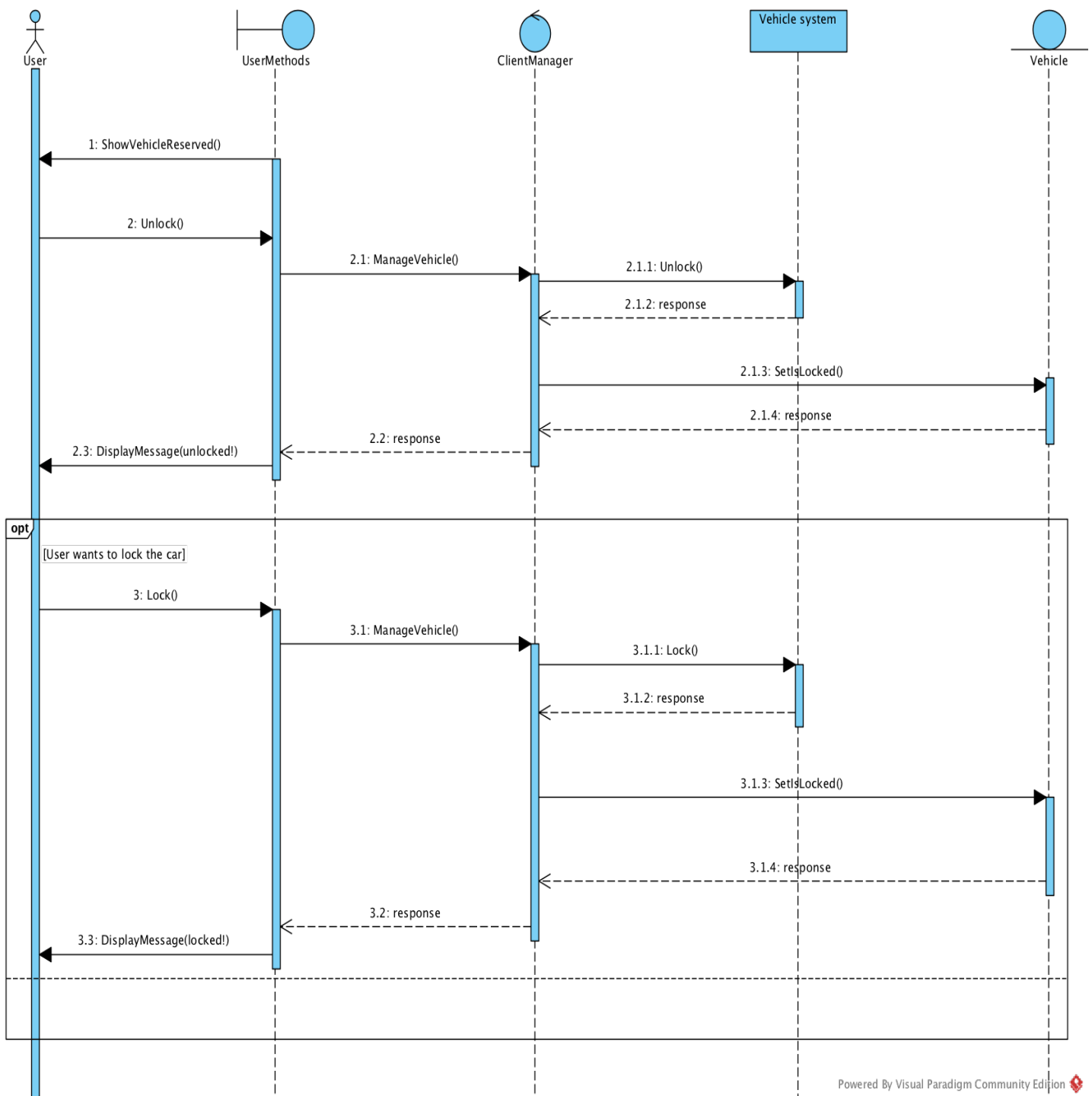
Show vehicles positions(technician) sequence diagram



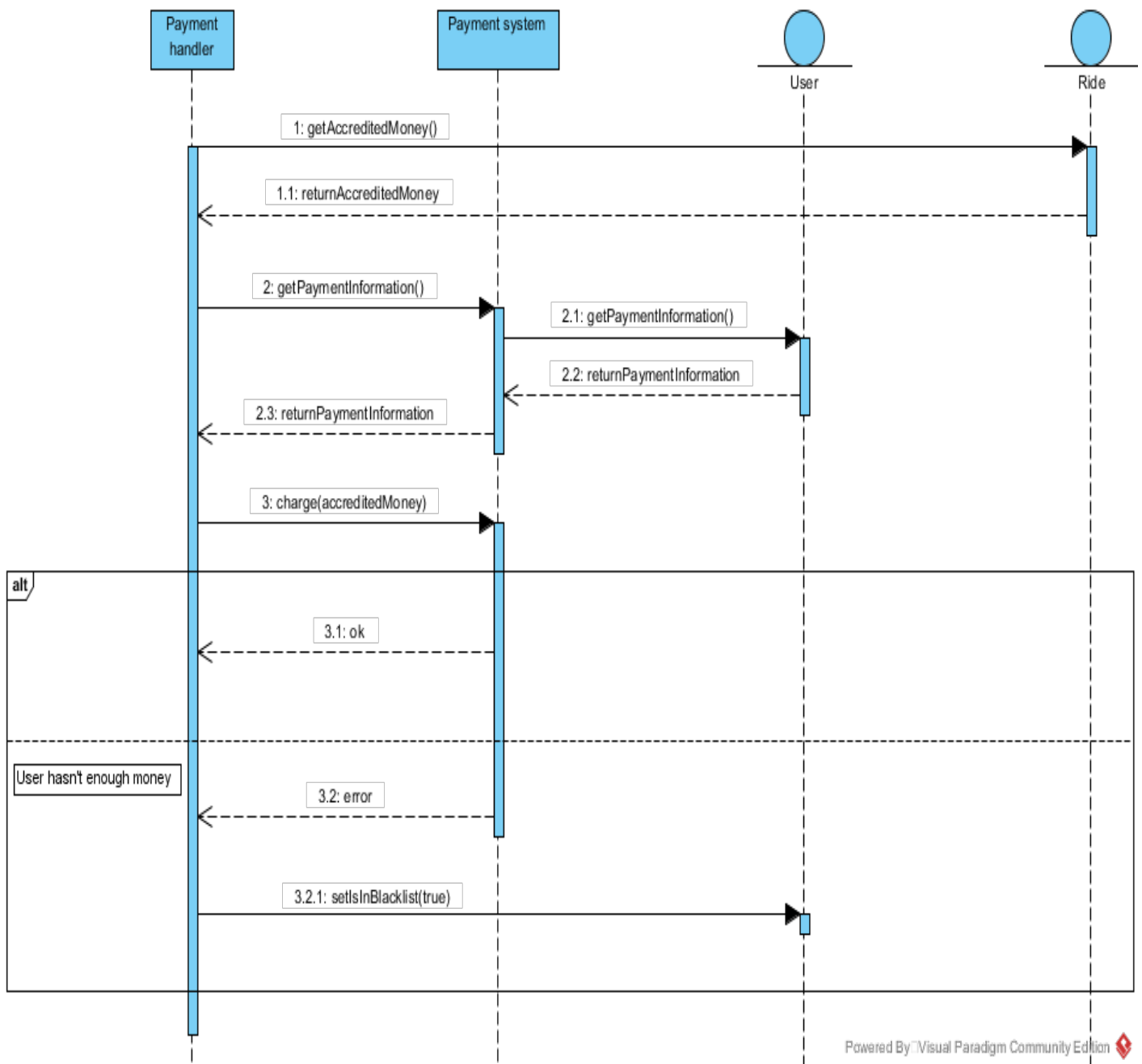
Reservation sequence diagram



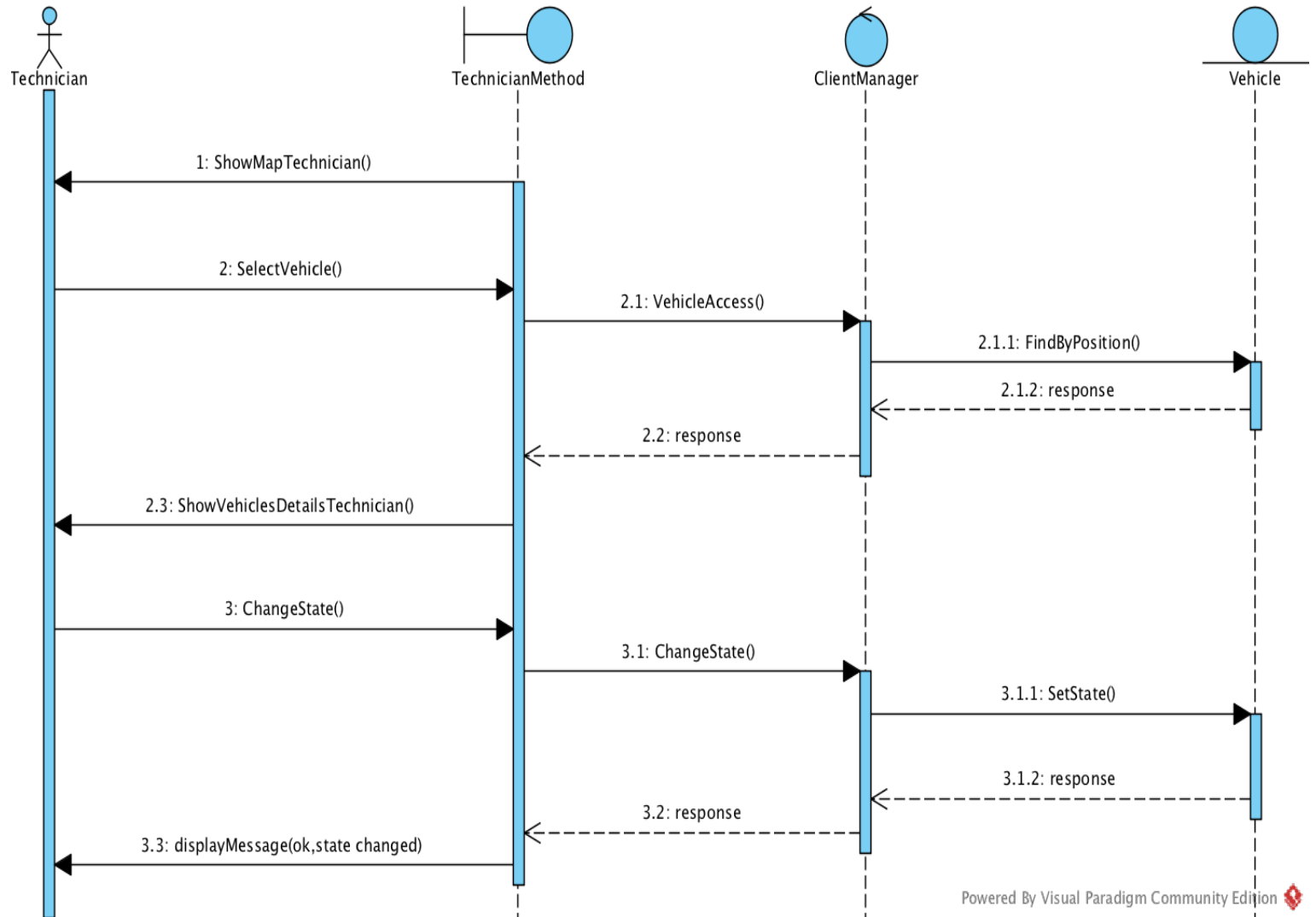
Use the vehicle sequence diagram



Payment sequence diagram



Change state (technician) sequence diagram



2.6 Selected architectural styles and patterns

For the developing of our application we have done some choices about patterns and architectures used.

We decided to use the MVC pattern to decouple the interfaces, the logic and the model each other. In this way we can develop each part separately and in case of modification or upgrade to the software the developers haven't to touch all the components. Furthermore, this pattern works well with the client server architecture.

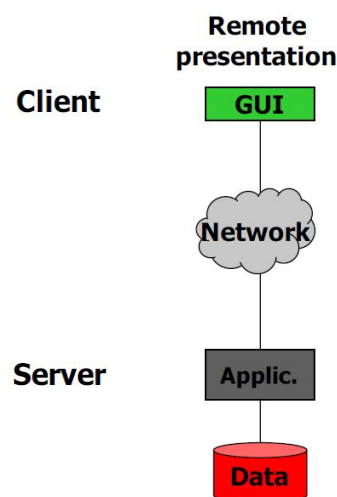
Regarding the architecture, we used a client-server architecture with three tier:

-**Client:** the user must have a web browser on his device in order to connect to the web site and access to the functionalities offered by the server.

-**Server:** in our server there are both the web server which interfaces with the browser of the client, and the application server in which there are the logic of our system and the methods to interface with the database.

-**Database:** in the database there are the persistent data of our system like vehicles, users and so on.

So our client-server architecture has a thin client, indeed the client has to manage only the presentation side, while the server manages all the logic.



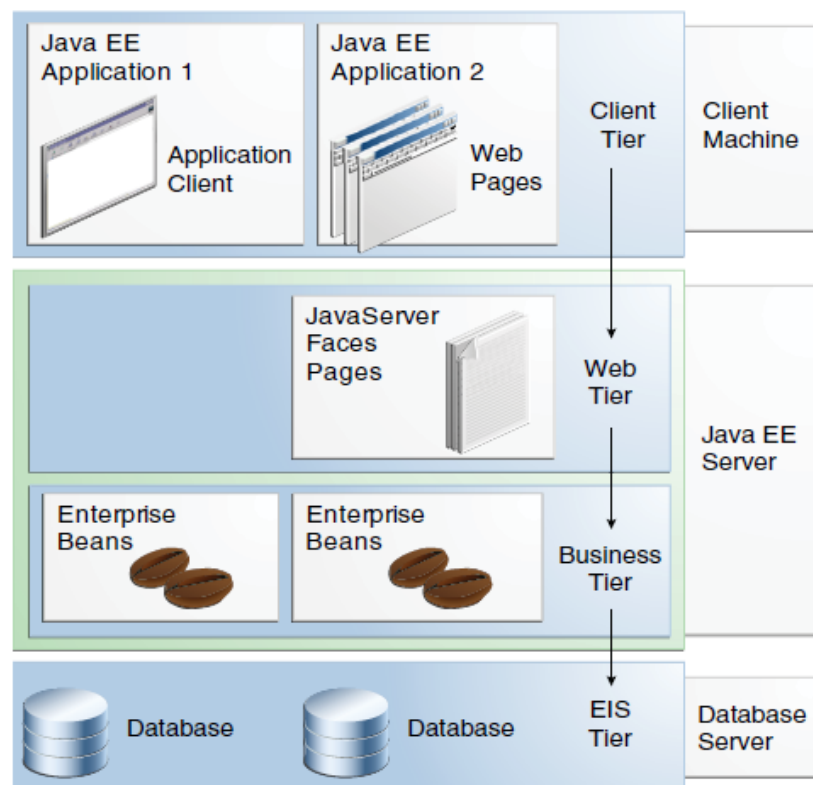
To develop our system we will use the APIs and containers given by Java EE.

JSP: Java Server Pages for the web server, to create dynamic web pages to send to the client.

Enterprise beans: we will use session beans to manage the communication between the web tier e the business tier in order to increase the scalability and the security.

JPA: Java Persistence API for the management of the database. With this API we can guarantee the persistence of the data, with the protection to failures and so on.

We followed the java EE architecture:



3. Algorithm design

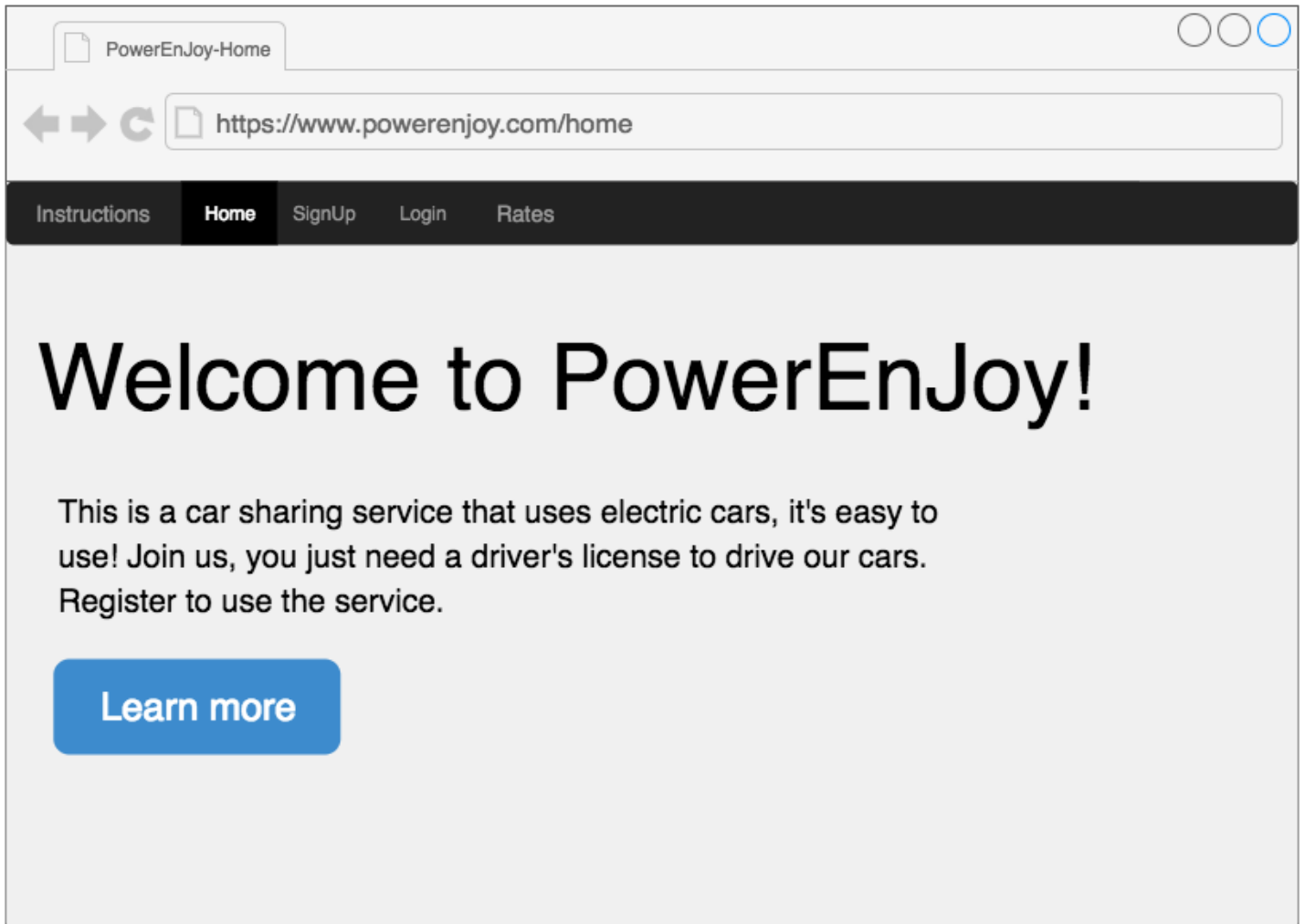
To develop this software, there are no particularly complex algorithms, and there are no particular constraints for developers. The important thing is that the code works and meets the functional requirements described in RASD. The algorithm that calculates the charge for the user must, of course, comply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

4. User Interface Design

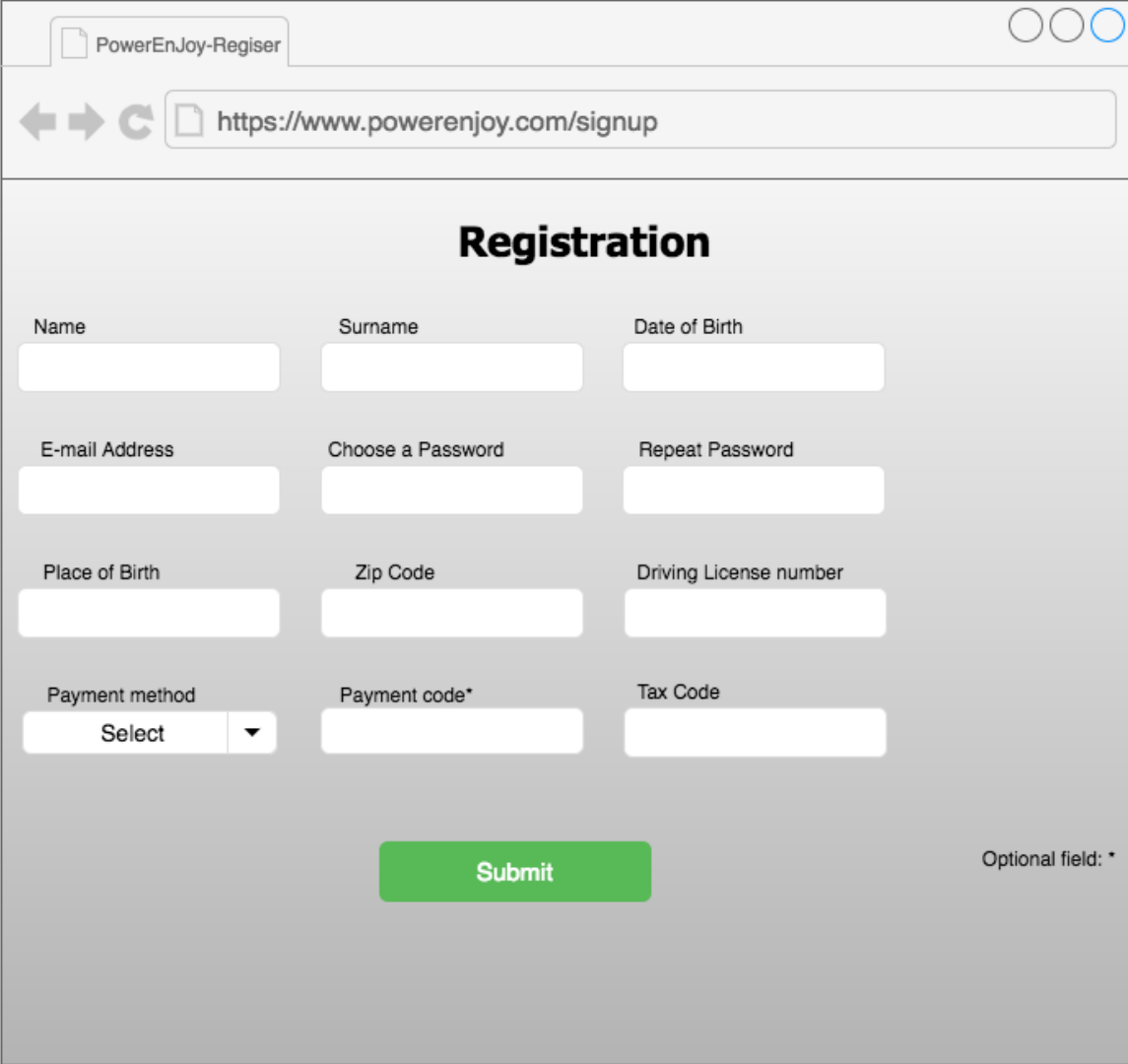
4.1 Mockups

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.

This is the PowerEnJoy homepage



This is the registration page

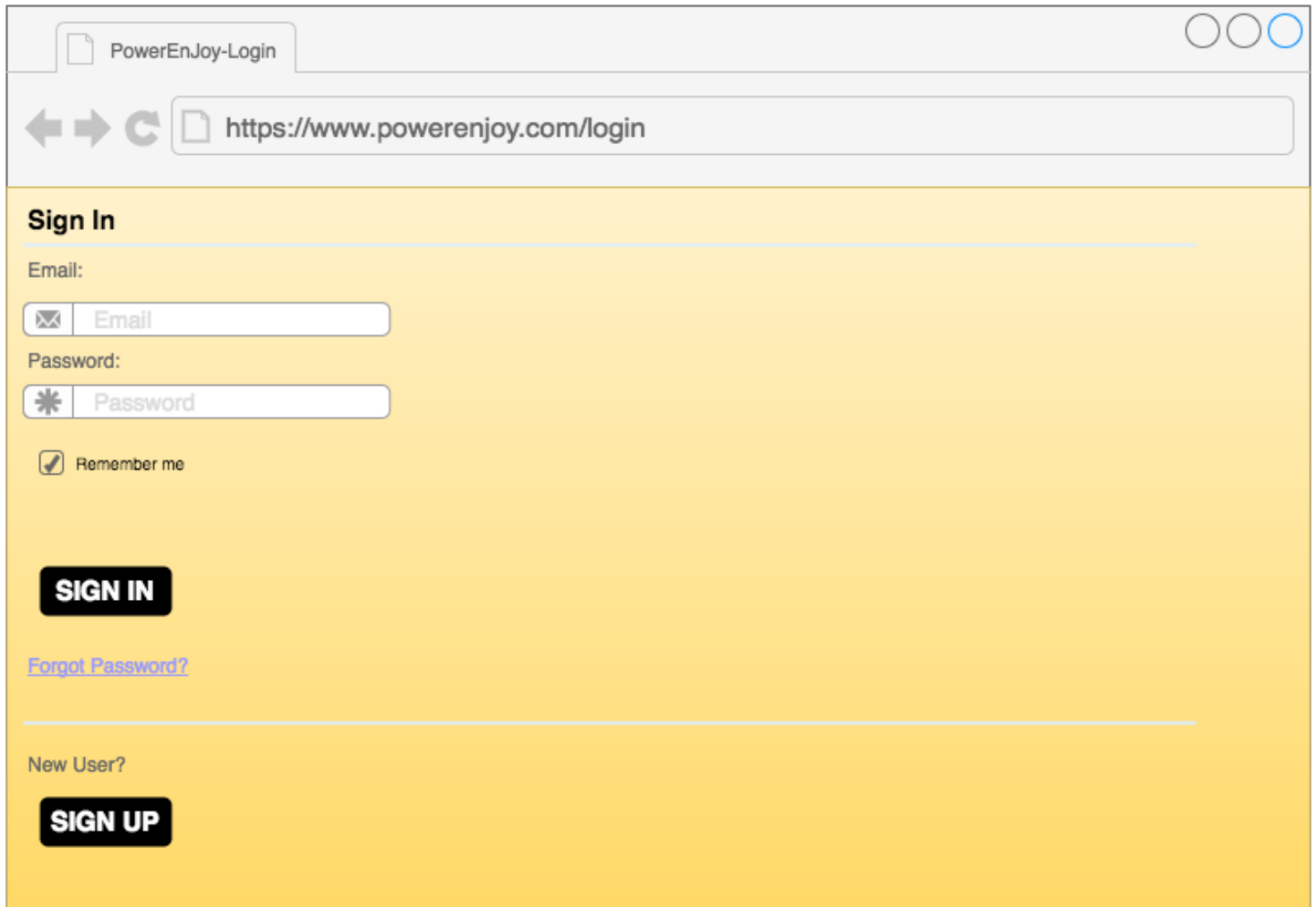


The image shows a web browser window with the title "PowerEnjoy-Regiser". The address bar displays the URL "https://www.powerenjoy.com/signup". The main content area is titled "Registration" and contains a form with the following fields:

Name	Surname	Date of Birth
<input type="text"/>	<input type="text"/>	<input type="text"/>
E-mail Address	Choose a Password	Repeat Password
<input type="text"/>	<input type="text"/>	<input type="text"/>
Place of Birth	Zip Code	Driving License number
<input type="text"/>	<input type="text"/>	<input type="text"/>
Payment method	Payment code*	Tax Code
<input type="text" value="Select"/> ▼	<input type="text"/>	<input type="text"/>

Below the form is a green "Submit" button. To the right of the button, the text "Optional field: *" is displayed.

This is the login page



The image shows a web browser window with a single tab titled "PowerEnJoy-Login". The address bar displays the URL "https://www.powerenjoy.com/login". The page content is on a yellow background and includes a "Sign In" section with input fields for "Email" and "Password", a "Remember me" checkbox, a "SIGN IN" button, a "Forgot Password?" link, a "New User?" section, and a "SIGN UP" button.

PowerEnJoy-Login

https://www.powerenjoy.com/login

Sign In

Email:

Password:

☒ Remember me

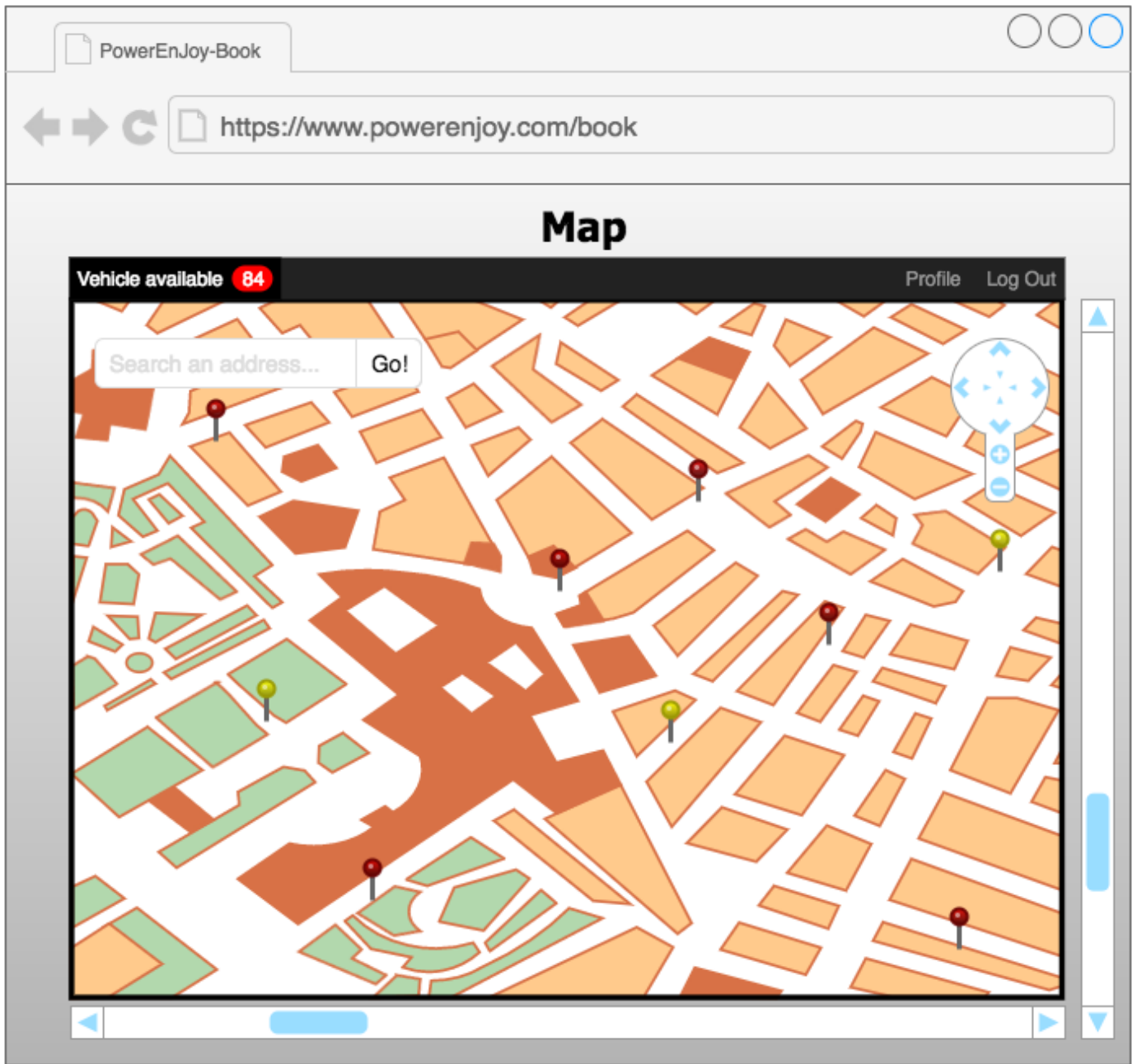
SIGN IN

[Forgot Password?](#)

New User?

SIGN UP

This is the map page, where the users can localize the cars



This is the page for the technicians that shows car details.
There is also a page like this for users, that permits to reserve
the vehicle, but of course, is not possible to change the state
of the vehicle from that page.

PowerEnjoy-profile

← → ↺

https://www.powerenjoy.com/profile

Vehicle ID: P123EWAS34

Charge: 52 %


Current state: Available

Is ignited: No

Position ---> latitude: 45.465454 longitude: 9.186516

Change state:

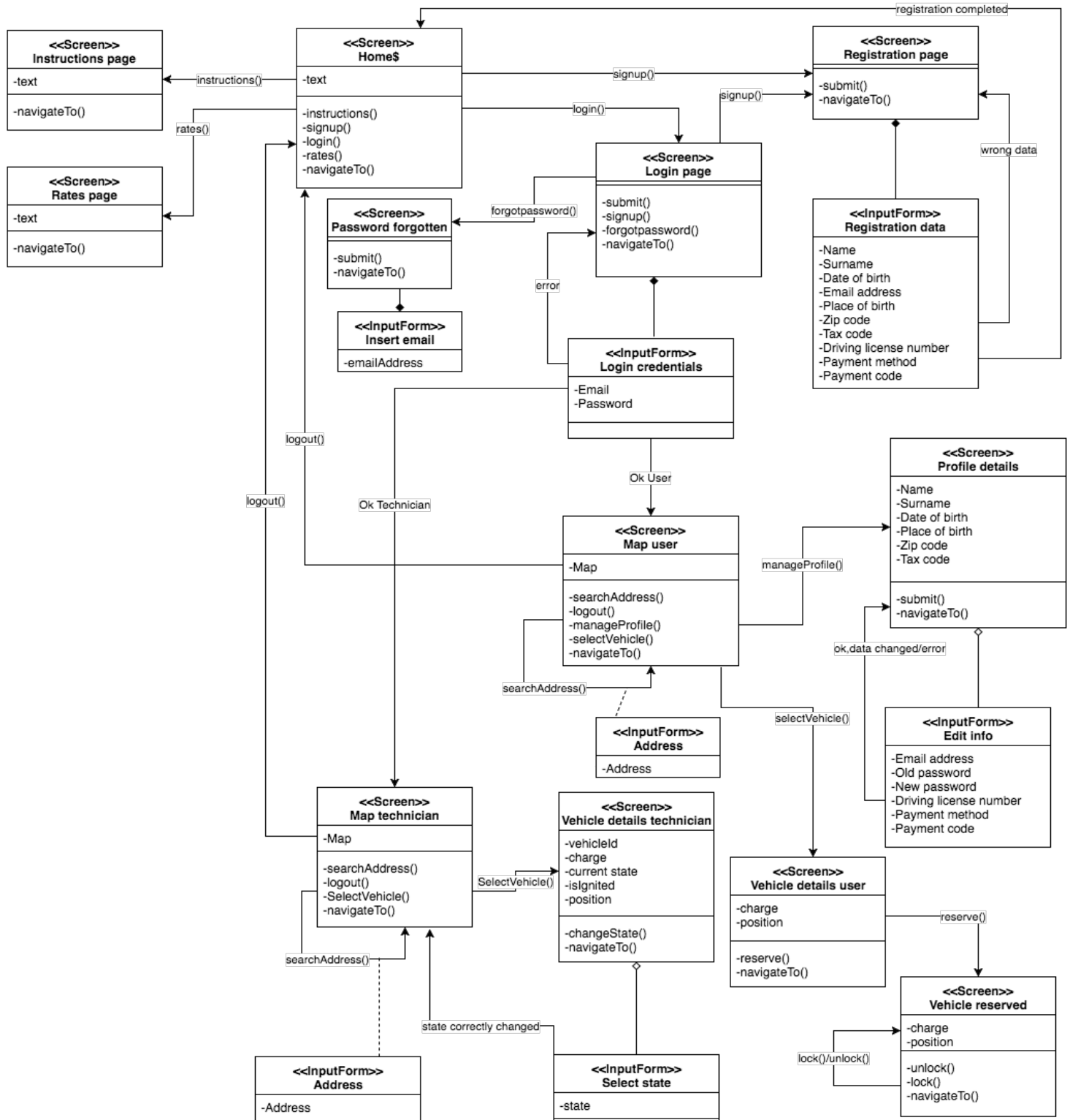
Select new state ▼



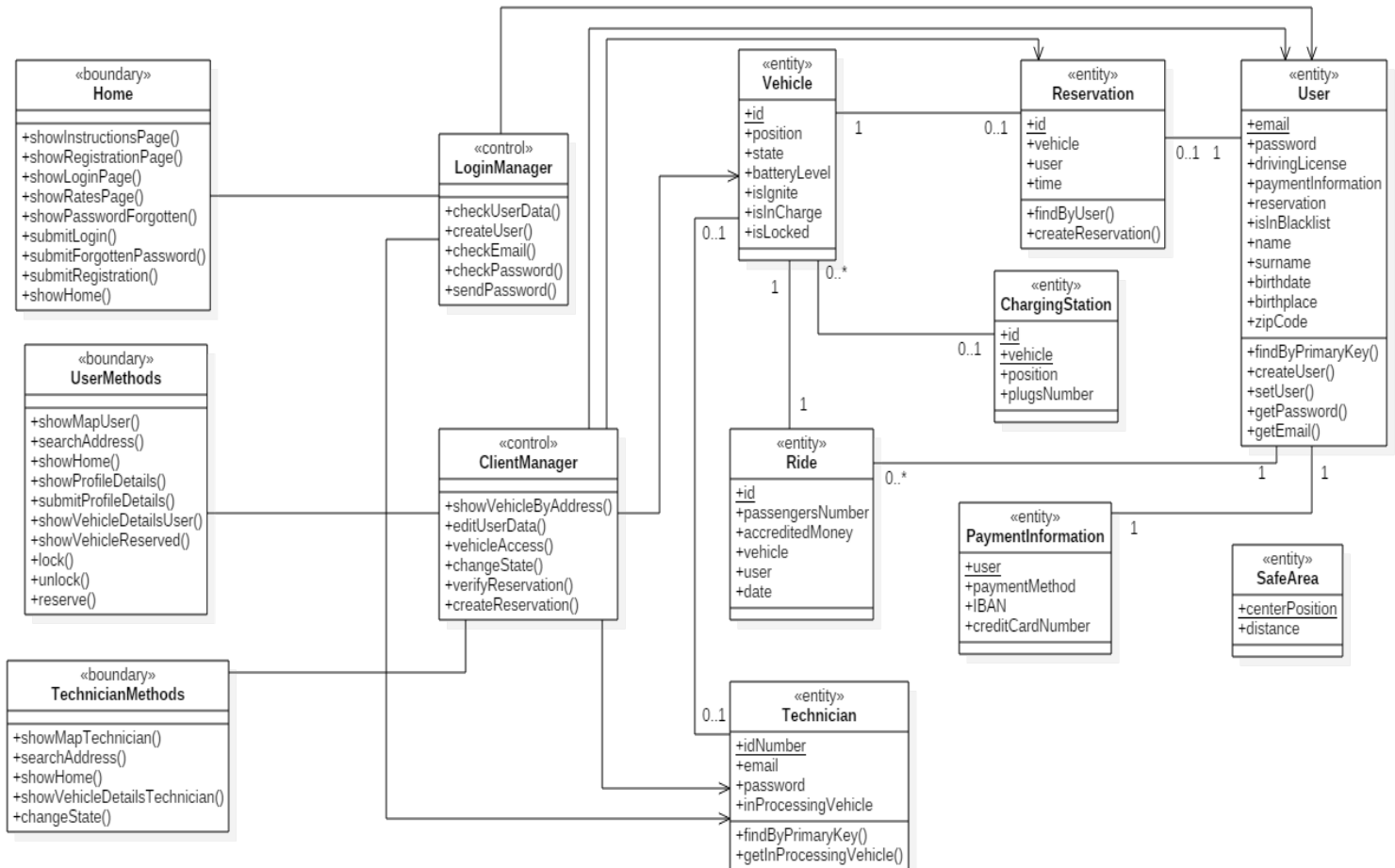
Submit

4.2 UX Diagram

From UX diagram you can understand more precisely what are the structures of the pages and what are the possible transitions between them.



4.3 BCE Diagram



5.Requirements traceability

G1. A person who has the right requirements must be able to register himself to the system.

- Client request manager
- Profile manager
- Data manager

G2. A registered person must be able to authenticate himself to the system.

- Client request manager
- Profile manager
- Data manager

G3. A user must be able to localize the positions of the available vehicles.

- Client request manager
- Profile manager
- Position handler
- Car handler

G4. A user must be able to reserve an available vehicle, for a limited time.

- Client request manager
- Profile manager
- Reserve manager
- Car handler
- Data manager

G5. A user who has reserved a vehicle, must be able to use it.

- Client request manager
- Car handler

G6. The system must properly charge for the user the cost of used services.

- Payment handler
- Car handler

G7. The system must properly manage the availability of vehicles.

- Car handler

G8. The system must simplify the organization of the technicians in their work.

- Client request manager
- Profile manager
- Data manager
- Car handler
- Position handler

6. References

6.1 Used Tools

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

Hours of work

These are approximatively the time we spent to write this document

Luca Santini: 30 hours

Riccardo Remigio: 30 hours

Changelog