Software Design Document

PowerEnJoy

Authors:
Patrizia Porati, Tommaso Sardelli

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

This document describes the architecture underlying PowerEnJoy, starting from the specifications and requirements described in the RASD document. We are going to describe our choices about software and hardware. Software components, will be presented showing relations and interactions between them, as well as architectural styles and patterns chosen. Hardware architecture will be presented showing how the software will be deployed on it. We will use UML diagrams in increasing detail as a standard language to explain concepts.

1.2 Scope

The intended audience of this specification include project managers, but above all software developers that are going to implement our application in the future. Within the document we are going to examine architectural choices and technologies employed with different levels of details, but we opted to omit implementations details that looked too specific to us. Examples of those details are programming languages or software to use across the system. We believe that the eventual implementer should (and should be able to) implement the devised architecture using the tools and languages he prefers or knows best.

1.3 Definitions, acronyms and abbreviations

The following are used throughout the document:

RASD Requirements Analysis and Specification Document.

DD Design Document.

REST REpresentational State Transfer.

IaaS Infrastructure as a Service.

RDBMS Relational Database Management System.

Reverse Proxy Proxy server that retrieves resources on behalf of a client from one or more servers.

Load Balancer Element handling the distribution of workloads across multiple computing resources

1.4 Reference documents

This document provides additional and more detailed information extending what has been presented in the RAS Document. They both concur to provide a general overview and understanding of the purposes and the architectural details of our platform. As for the organization of the chapters and the paragraphs, we followed the structure and the subdivision into paragraphs of the template for the design document made available by our professor. Finally, we also helped ourselves consulting the IEEE Standards for the Design and the Architecture Descriptions.

1.5 Document structure

Our Design Document is divided into eight main parts:

Section 1: Introduction this first section gives an introduction to the Design Document, specifying the purpose and the scope of the document, the glossary (containing all the terms, definitions, acronyms or abbreviations used) and the documents we referred to in order to develop this paper.

Section 2: Architectural Design this section is the core of the document. It gives information about the design and the architecture of our application, defining all the software and hardware components characterizing the system and how they can interact between each other.

Section 3: Algorithm Design this section contains the description of the main algorithms used in our system.

Section 4: User Interface Design this section contains a detailed representation of the users interfaces. Through specific and detailed mockups.

Section 5: Requirements Traceability this section contains the correspondences between the requirements we have listed in the RASD document and the components we have identified during the design and architectural analysis.

Section 6: Used Tools in this section, all the tools we have used in order to develop this document are listed.

Section 7: Working Hours this sections contains the result of our effort, quantified in the number of hours we have needed in order to develop this document.

Section 8: References a list of all the references we took into account when writing this document.

2 Architectural Design

2.1 Overview

In this chapter we are going to describe the architectural choices that we made for our system. Starting from a general description of the high level components and their interactions, we will describe through a deeper analysis all the components of our system. The subject of the analysis will be the backend of our platform that will be used by our mobile and web applications, as well as by the devices installed on the cars.

2.2 High level components

This first section is dedicated to the description of our architecture from a high level point of view. We based our decisions upon the requirements described in the RAS Document and we choose three principles as the foundations of our structure:

- **Client-server** model.
- **RESTful** API.
- **Microservices** approach.

2.2.1 Client-Server

The client-server aspect of our application can be seen in Figure 1. There we focus on the "physical" representation of our application. By saying physical we mean that every component depicted there will reside on separate machines (or is an external service). To be precise, we should have added another component between clients and application server which is the reverse proxy/web server. The reason why we preferred to omit it is that, at this stage, we want to address only the aspects that are relevant to the application development and that layer can be considered transparent with respect to the application. The application does not have to deal with it explicitly. Of course this will not be the same in a context of deployment. The interaction between client and server will be the usual one when dealing with web/mobile applications. Our client will make the initial request and the server will send back the information. In out case, such informations will be received as an XML or JSON document.

2.2.2 RESTful API

For the design of the API we decided to rely upon the REST principles. The reasons that moved this decision are diverse. To begin with, we decided that a generic API was better fitting in our context that is composed of different

clients (web application, and different mobile clients). This way, we can serve a single format of document as respond containing all the requested data and then the client can build the user interface using the most suitable technologies according to the specific case. Secondly, REST principles, when followed correctly, concur to gain desirable non-functional features, such as performance, scalability, simplicity and modifiability. Finally, messages exchanged in REST APIs are often self-descriptive and this is beneficial for the ease of development of web and mobile applications.

2.2.3 Microservices

Figure 2 is a high level overview of the application internals stripped from all the external components. We are going to describe its architecture further in this document but we wanted nonetheless to give an initial idea. The internal structure of the application is going to be a service oriented architecture. Upon receiving a new request, the router (our entry point) will produce a new message, that will be dispatched to all the required services. The response to this message will be rendered and sent back to the client. The reason why we choose microservices is that they allow us to separate the code in atomic components, each of them dealing with a single responsibility. Furthermore, using queues to dispatch messages, ensure that our internal communications are asynchronous and non blocking.

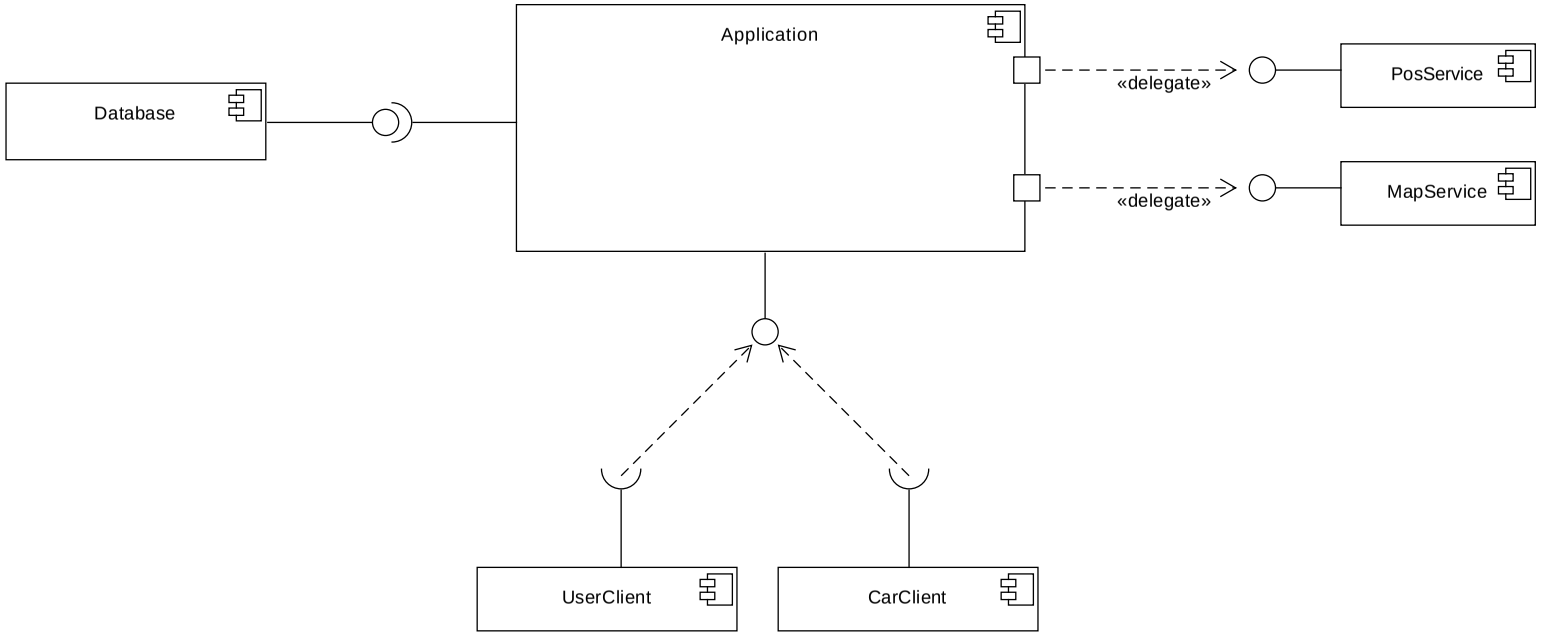


Figure 1: Component view: High Level Architecture

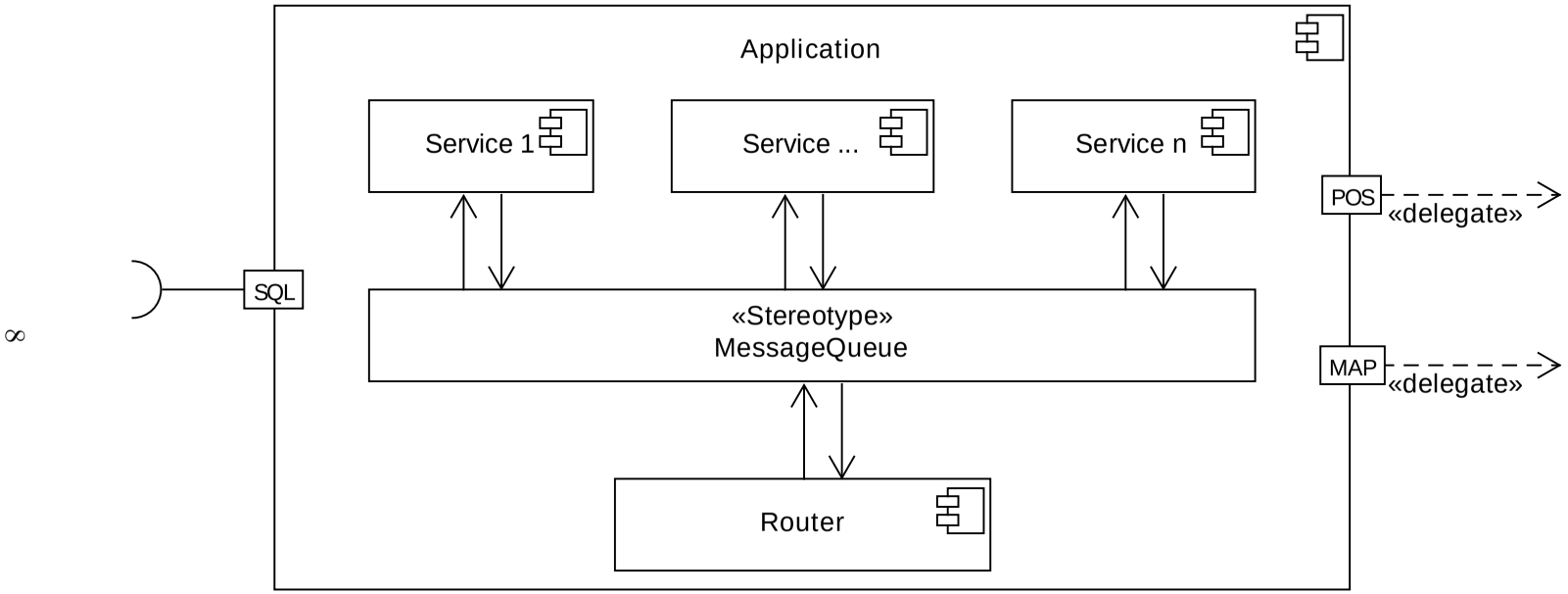


Figure 2: Component view: Internal High Level Architecture

2.3 Low level components

In this section we are going to describe further the internal configuration of our server application. After giving a general overview, we will analyze each component separately. As we anticipated in the previous section, the application is going to follow the microservices architecture internally with some functions delegated to external services or components. We will start by describing the latter and then we will move on to the internal ones. To do so we will use Figure 3 as a reference. In that diagram you can see a more detailed example of what we have already seen in Figure 2.

2.3.1 Map API

Starting from the latter, we assume that all functions related to the map (path finding, distance calculation, map representation, etc) will be delegated to an external service. All the classes that are going to use that API will provide an additional interface if they need to expose some functionalities to the other components.

2.3.2 POS API

The other aspect that we are not going to develop internally is the Payment Service (POS). In this case we suggest to use services like Paypal or Stripe that have been chosen for its ease of use and powerful API.

2.3.3 Database

Finally, the last component that will not be part of the application in a strict way is the database. It won't be an entirely external service since we are going to host and mange it in our infrastructure, but it's still something our application will user through its dedicated API. Regarding the database we will use the solid and well tested RDBMS system.

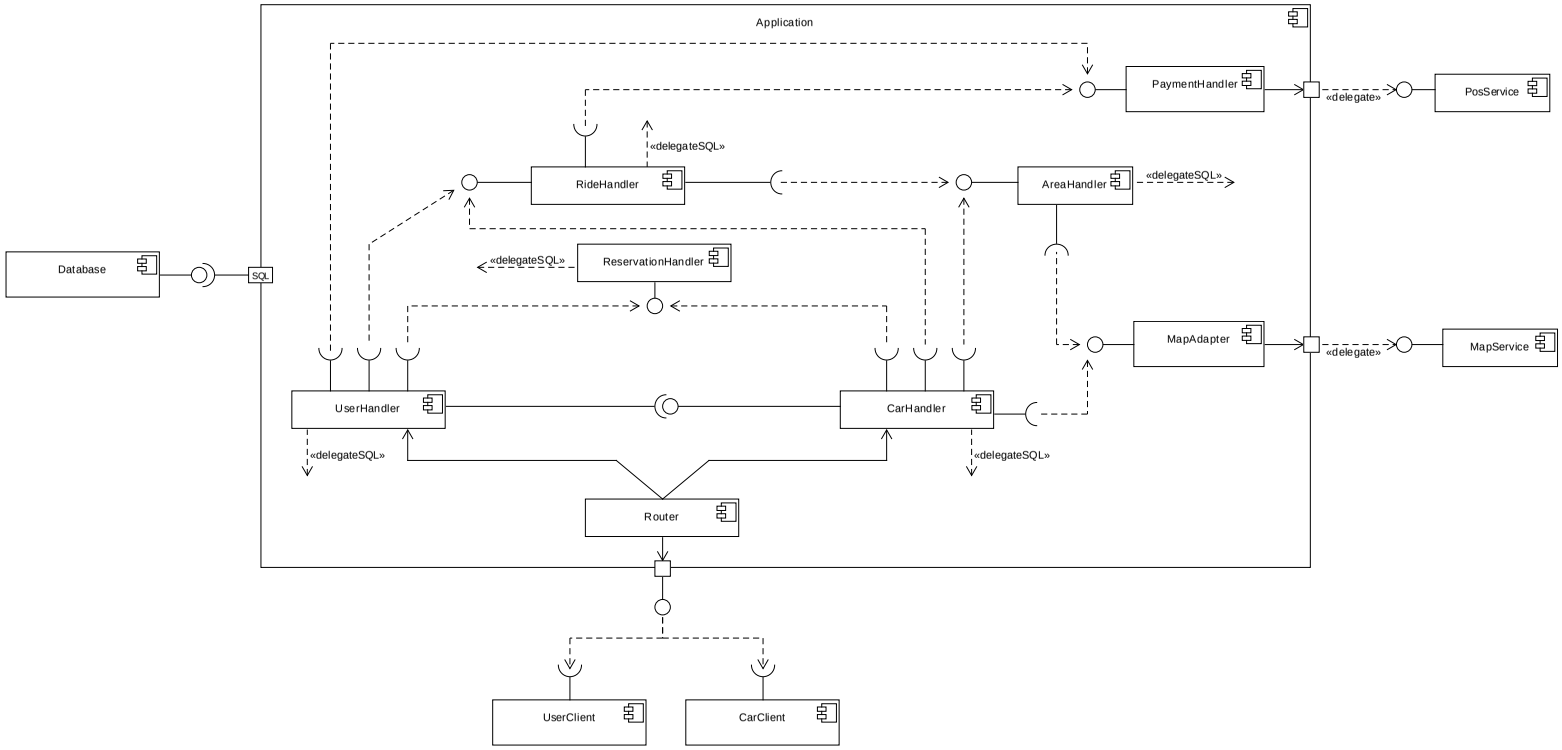


Figure 3: Component view: Internal Low Level Architecture

2.3.4 Microservices

Router The Router is the first of our internal component. The idea behind is that this is the component delegated to do the initial parsing of the request coming from the client. It will analyze the URI and the HTTP method following the REST principles and based on that informations, it will produce a message to be enqueued. Once it will get back a reply from the service corresponding to the initial request, it will send back that response to the client.

UserHandler This service deals with everything that concerns users. It's invoked to perform user creation and to deal with requests coming from users devices. All the informations involving users will be passed from the router to this component. In order to gather the required information it may invokes other services as well.

CarHandler This is the car counterpart of what we have already discussed for users. This component will register new cars when they are added to the system and provide informations about cars position, charge status or availability.

ReservationHandler ReservationHandler provides informations related to reservations. It is likely that it will respond to UserHandler messages when the list of reservations for a single user is requested. Alternatively, CarHandler can retrieve a list of all reserved cars.

RideHandler This is the component that handles all rides information. It can be queried in order to have a list of all rides, of the active ones, rides belonging to a specific user (they may be active or not) and is the link to payment and discount informations. All the informations displayed on the car monitor during the ride are gathered from this service.

PaymentHandler The PaymentHandler has two main purposes. The first one is to manage the operations dealing with the external Payment service. The second one is to save information about users payments after it gets a reply from the external service. This is done in order to provide a history of all payments or a list of pending payments.

AreaHandler This component is in charge of operations dealing with all the safe areas belonging to the company. Safe areas are stored to know exactly their position and the type of the safe area, being it a normal area or a power grid station.

2.4 Deployment

This section will be dedicated to the deployment strategy adopted for our application. Figure 4 shows the physical structure of the system and we are going to describe it in greater detail.

IaaS The infrastructure for our deployments will not be based on bare metal servers but we preferred to adopt an approach based on IaaS. In this way we can benefit from the flexibility provided by this kind of services that will allow us to start small and scale up, when our user base will grow.

Servers We choose to use a three-tier architecture with three different servers:

- Reverse Proxy: This our first tier and has the main role to redirect users request to the application server. Other jobs executed on this server are: caching, load balancing and serving static files. Load balancing is a function that maybe is not going to be exploited in the beginning, but we expect it to become useful, when the load of out application server will increase.

- Application Server: This is the server containing our application and is our second tier. All requests will be redirected here in order to provide a response. Looking at Figure 4, this component is represented by a solid rectangle with additional dashed rectangles around it. This kind of notation represent our idea that in the beginning there will be only one application server but, when the load will increase, this is the first component that will be affected by it. For this reason we want to be able to add other instances of our application seamlessly. This is made possible thanks to our previous decisions to use IaaS and a load balancer, while data consistency will be granted by a stateless architecture and persistence delegated to the database.

- Database Server: Our third layer is the database. In this case we adopted a traditional approach and used a relational database. Since RDBMS are usually well optimized to work even under heavy load, we don't expect to need any kind of replication right now.

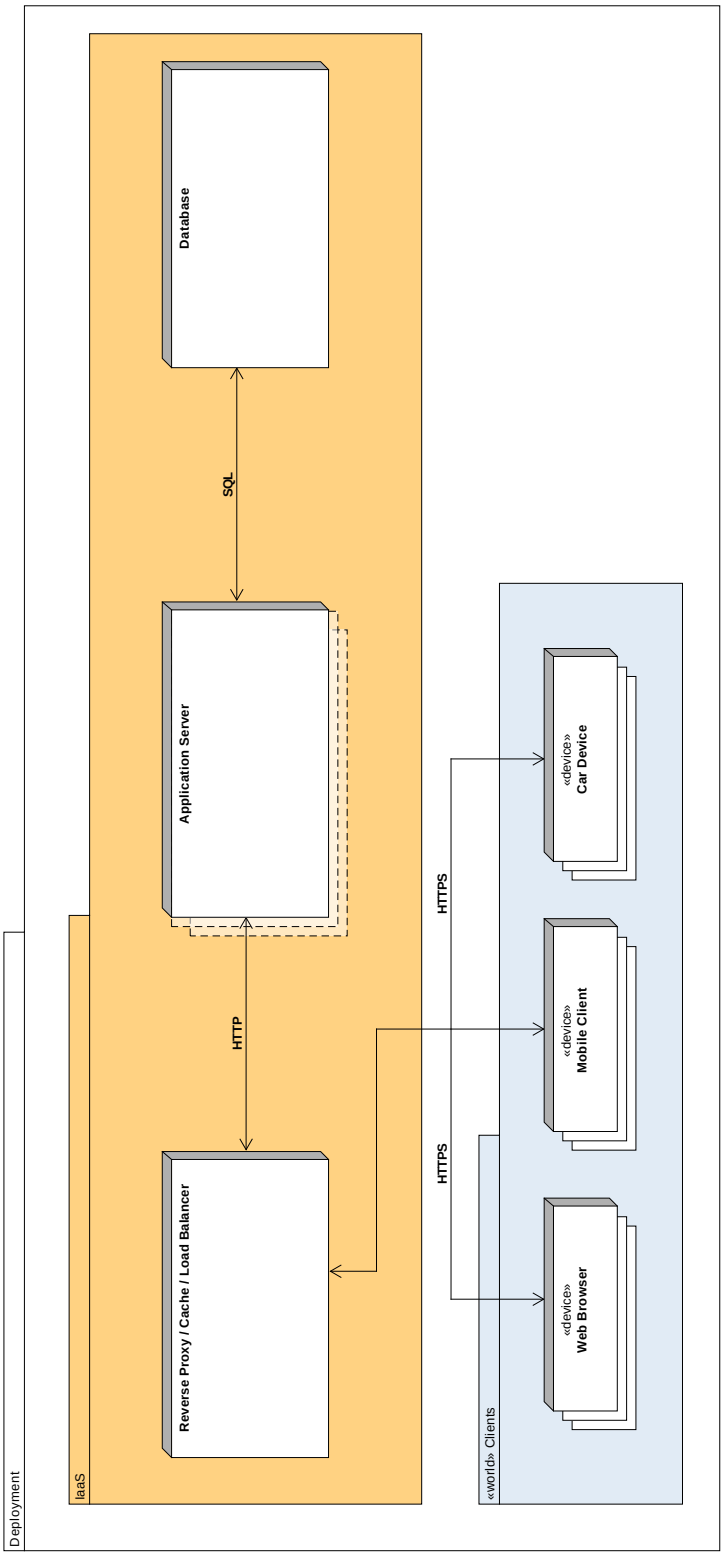


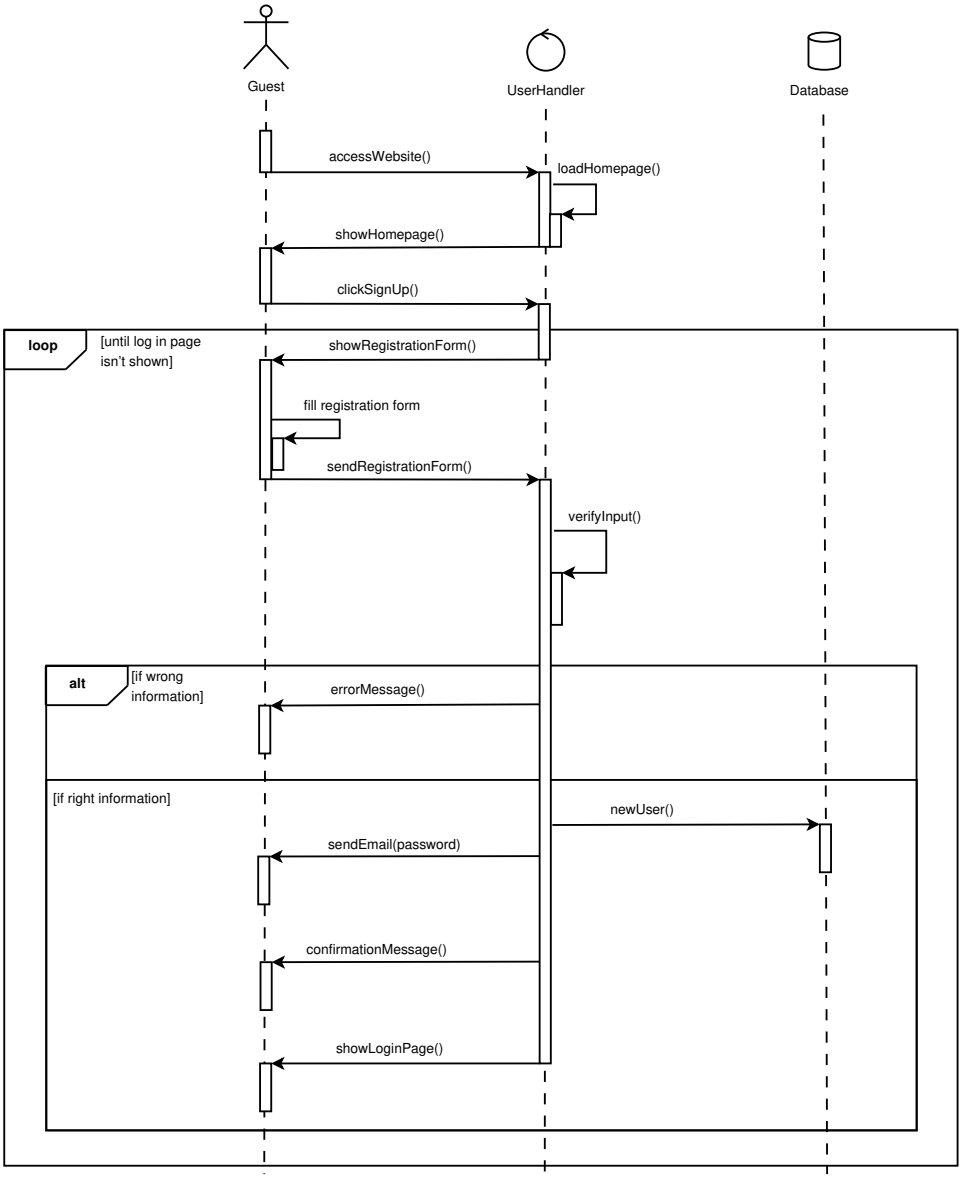
Figure 4: Deployment view

2.5 Runtime view

In this paragraph we are going to show some sequence diagrams related to the use cases identified in the RASD document, in order to explain how the components of the system interacts with each other and with the actors.

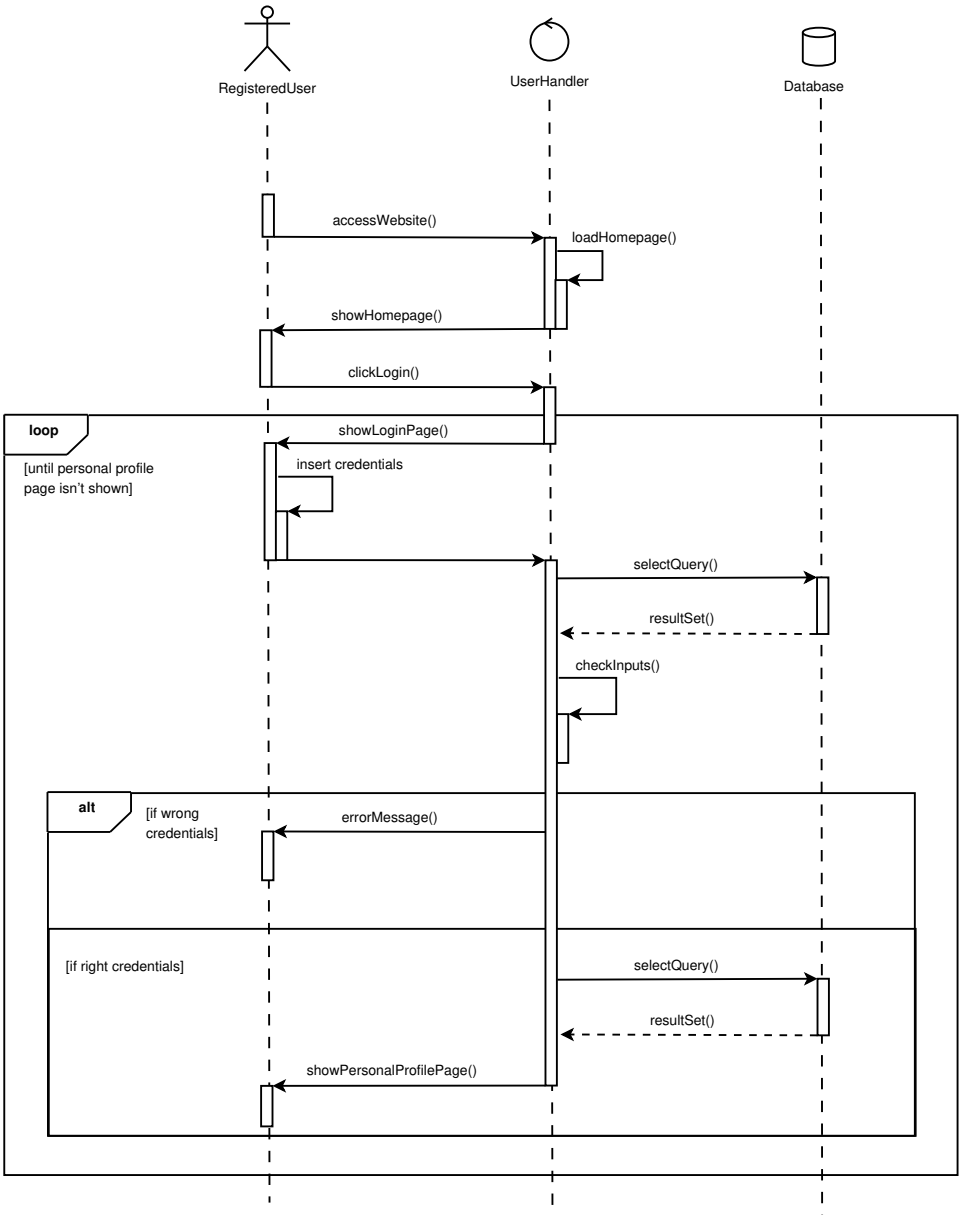
In order to enhance understanding, in these sequence diagrams we decided to omit the router component because its only task is to switch the request to the user handler or the car handler according to the request type.

2.5.1 Registration



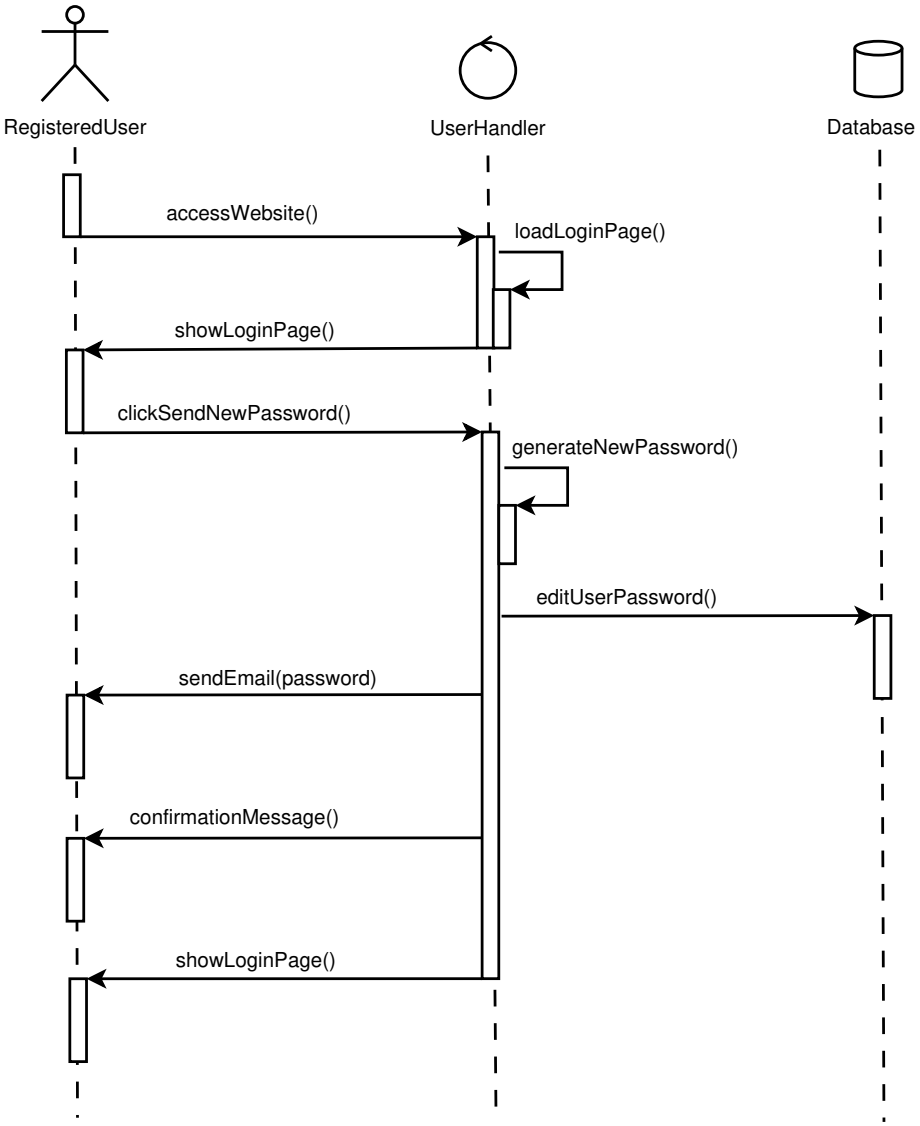
Sequence Diagram 1: The registration

2.5.2 Log in



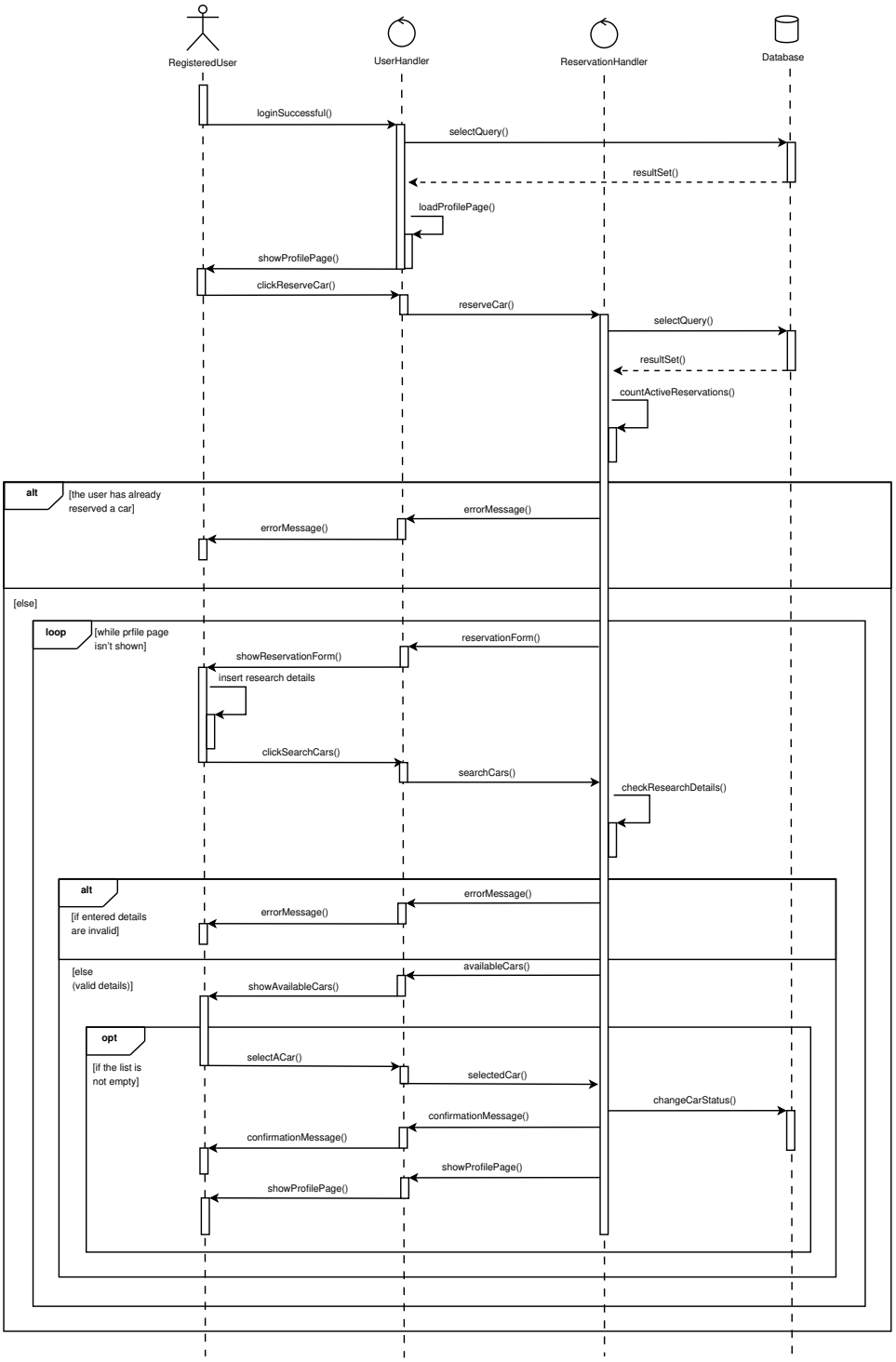
Sequence Diagram 2: The log in

2.5.3 Recover the password



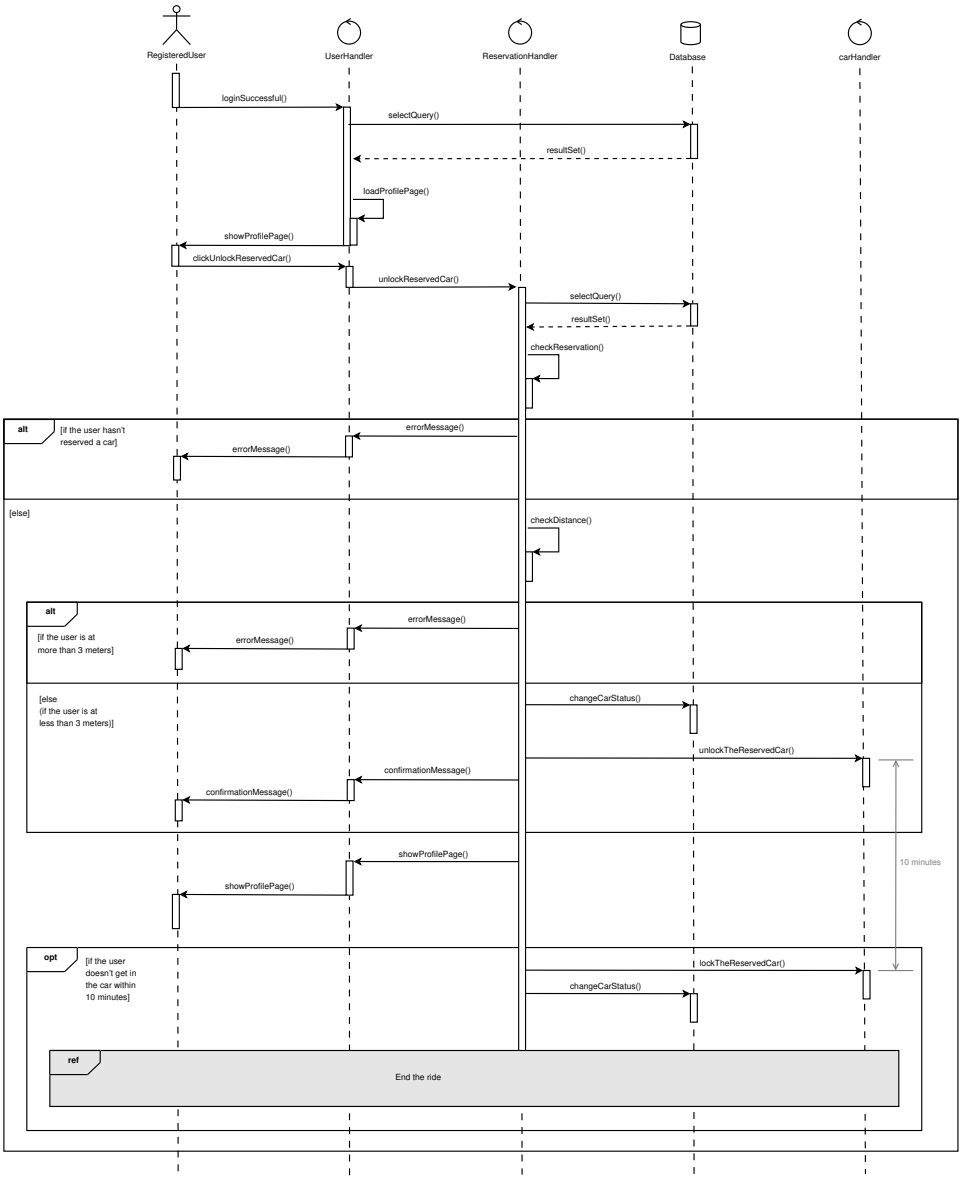
Sequence Diagram 3: The recovery of the password

2.5.4 Reserve a car



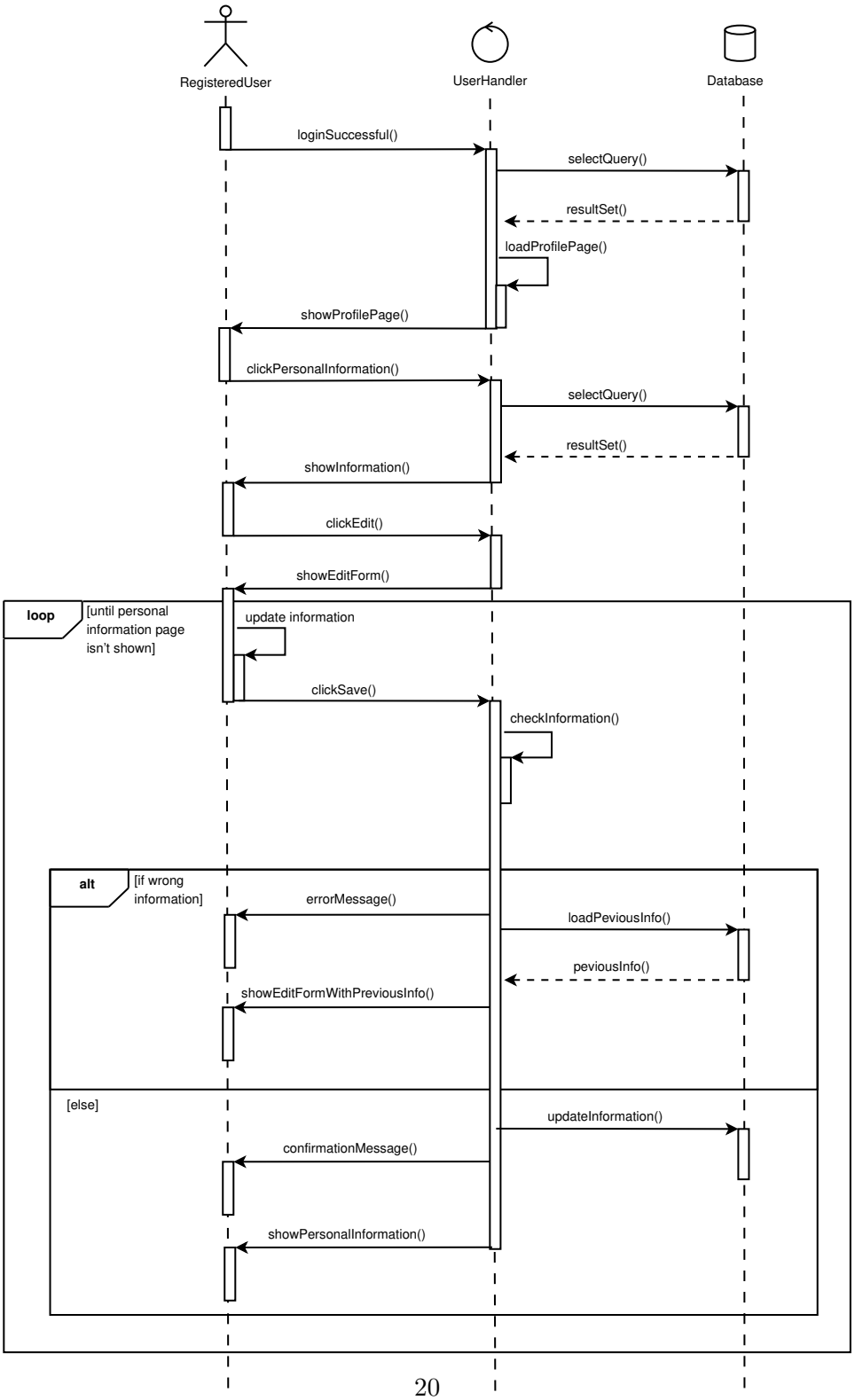
Sequence Diagram 4: Reservation of a car

2.5.5 Unlock the reserved car



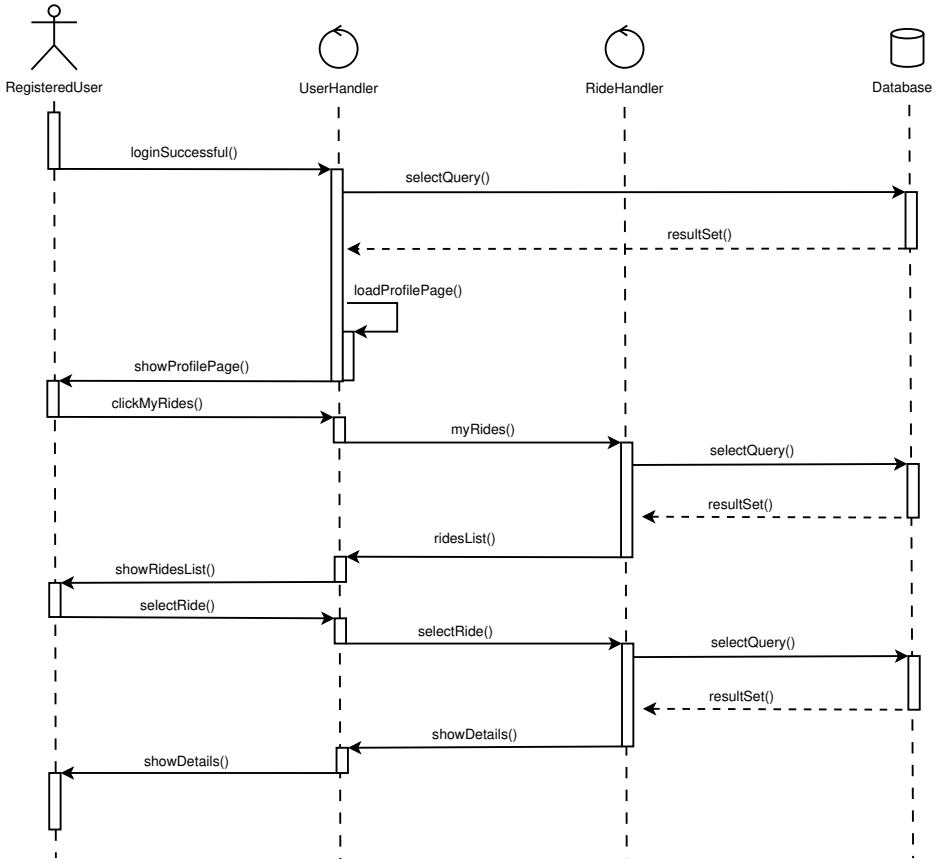
Sequence Diagram 5: Unlock the reserved car

2.5.6 Update personal information



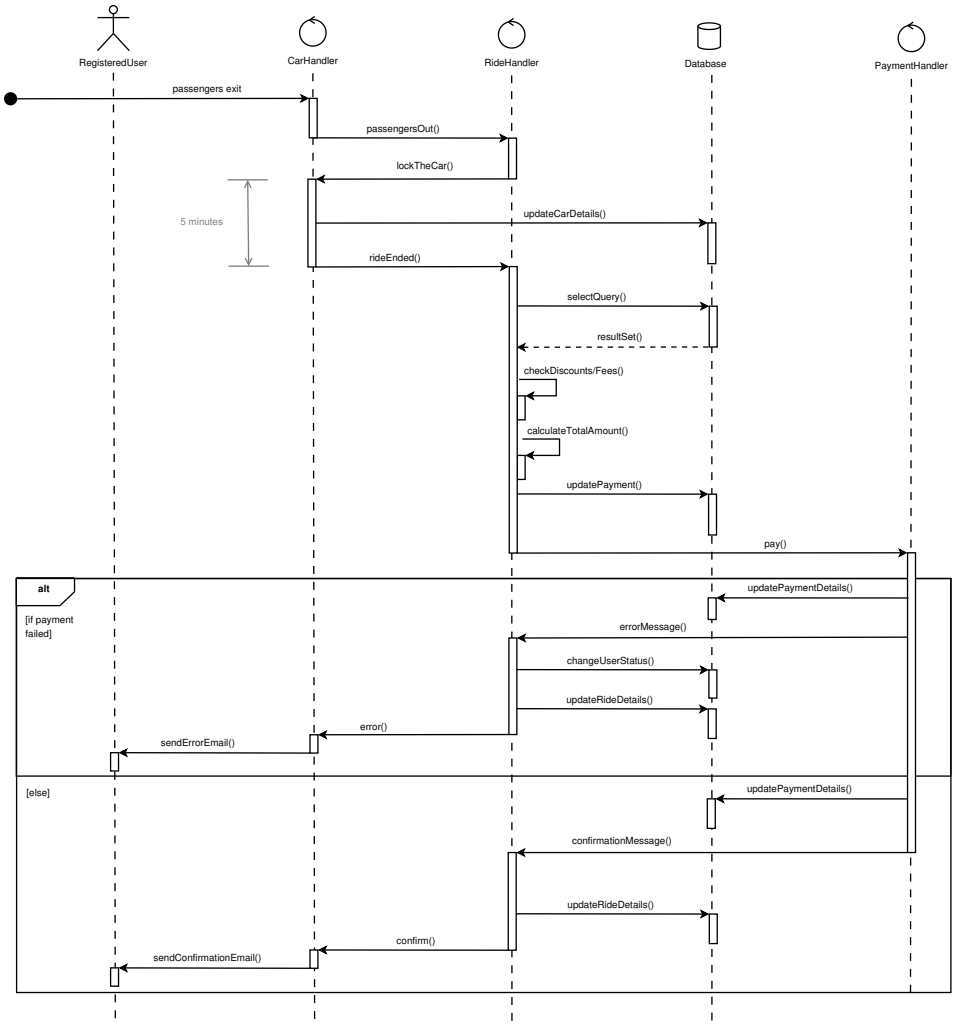
Sequence Diagram 6: Update personal information

2.5.7 Show the details of the ride



Sequence Diagram 7: Show the details of the ride

2.5.8 End of the ride



Sequence Diagram 8: End of the ride

3 Algorithm Design

Although our application holds many point of complexity we don't think any of them is relevant with respect to algorithm design. Almost all operations requires trivial algorithms, also thanks to the fact that we delegate the distances or routes computing to an external system. The real complexity of our system resides in how to handle all the different components, or how to ensure that important data are made persistent but again, we didn't find anything worth noting from the point of view of algorithms.

4 User Interface Design

In the following pages there are some mockups representing how the application and the car screen interfaces will look like.
In the RASD document we have already provided some mockups in order to show the general and essential structure of the most important pages. We now want to show how our application will really be.

We represented the mobile application pages of the main opeations and the car screen, but we don't represent the interface of the website pages because they're similar to the mobile application pages.

4.1 Homepage

Mockup 1: The home page.

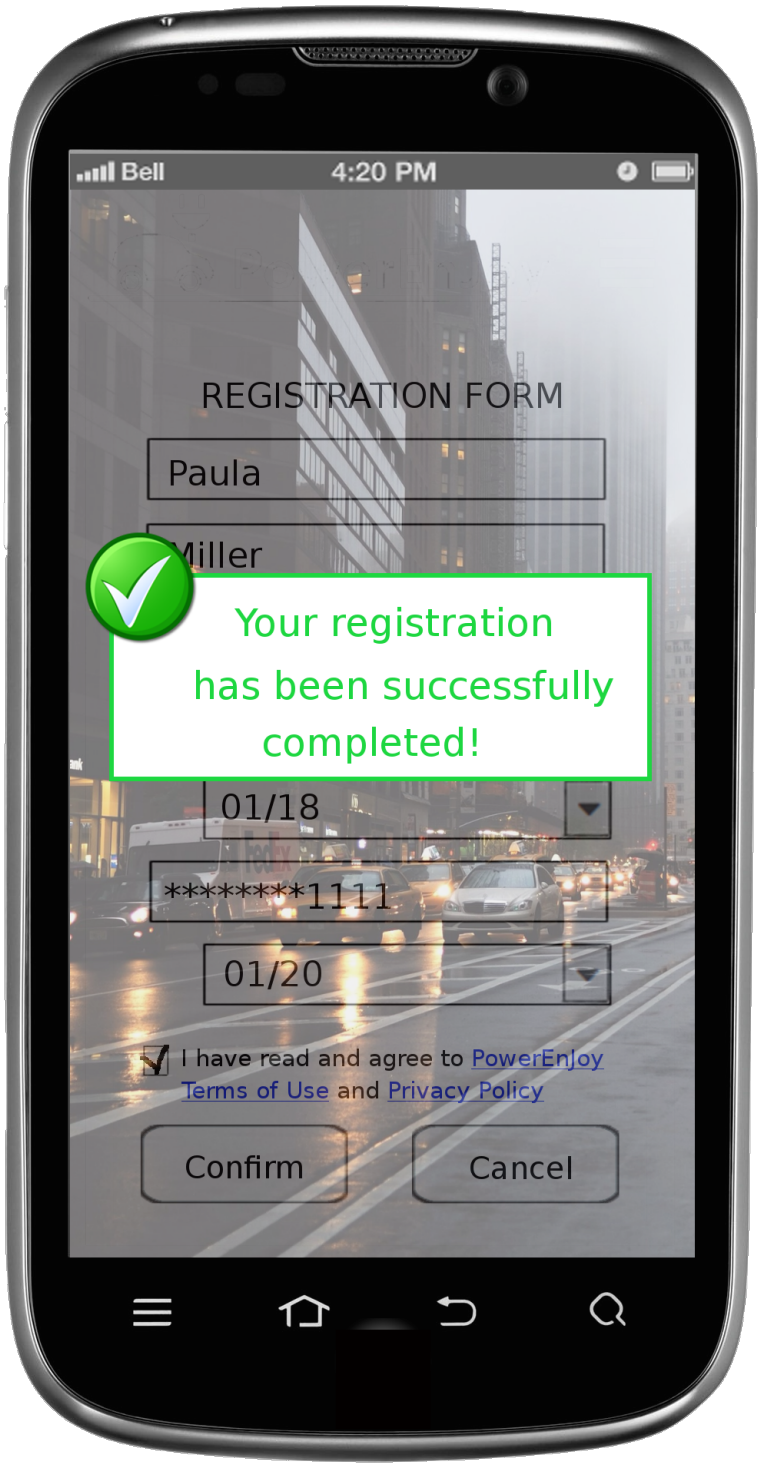
4.2 Registration form

Mockup 2: The page with the registration form.

4.3 Error message

Mockup 3: An example of error message.

4.4 Confirmation message

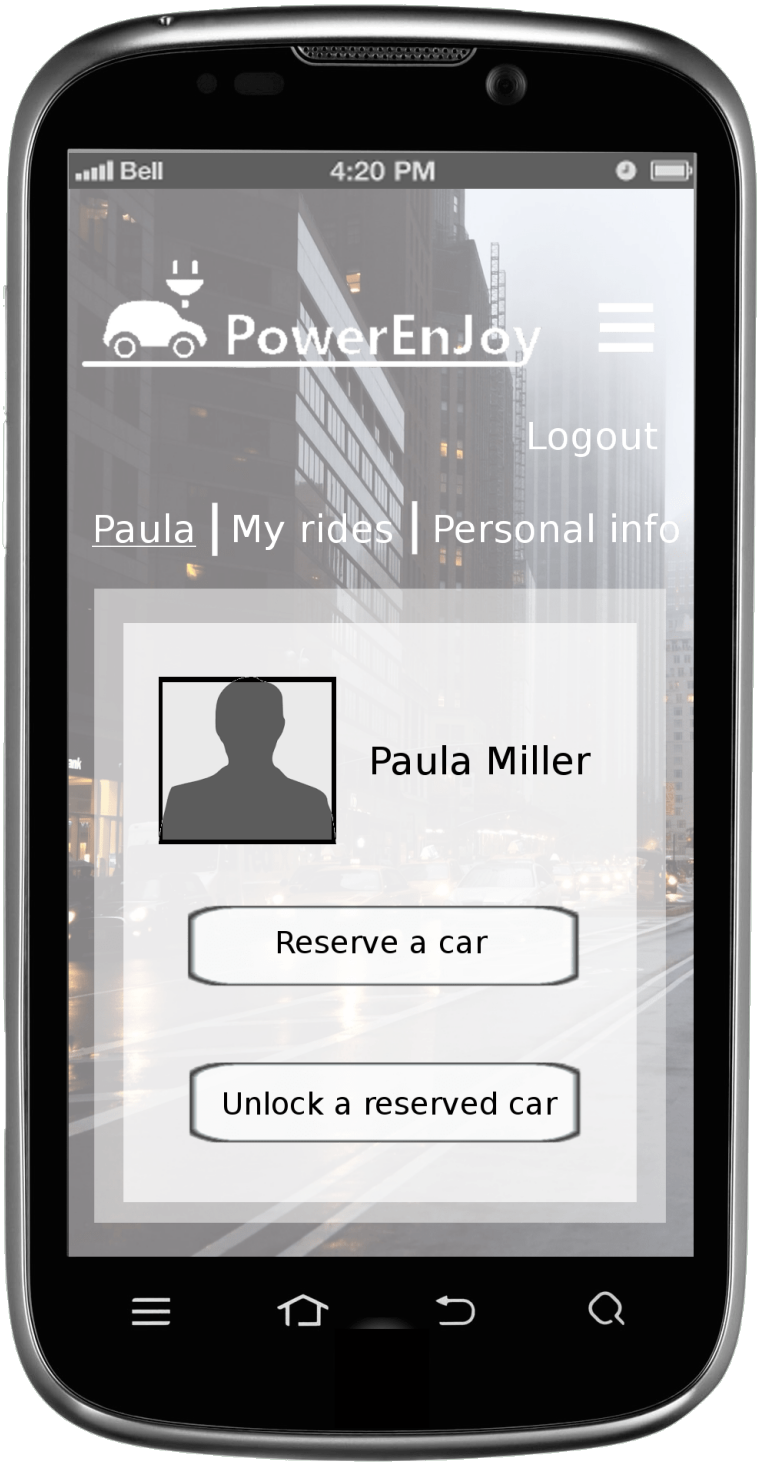


Mockup 4: An example of confirmation message.

4.5 Login

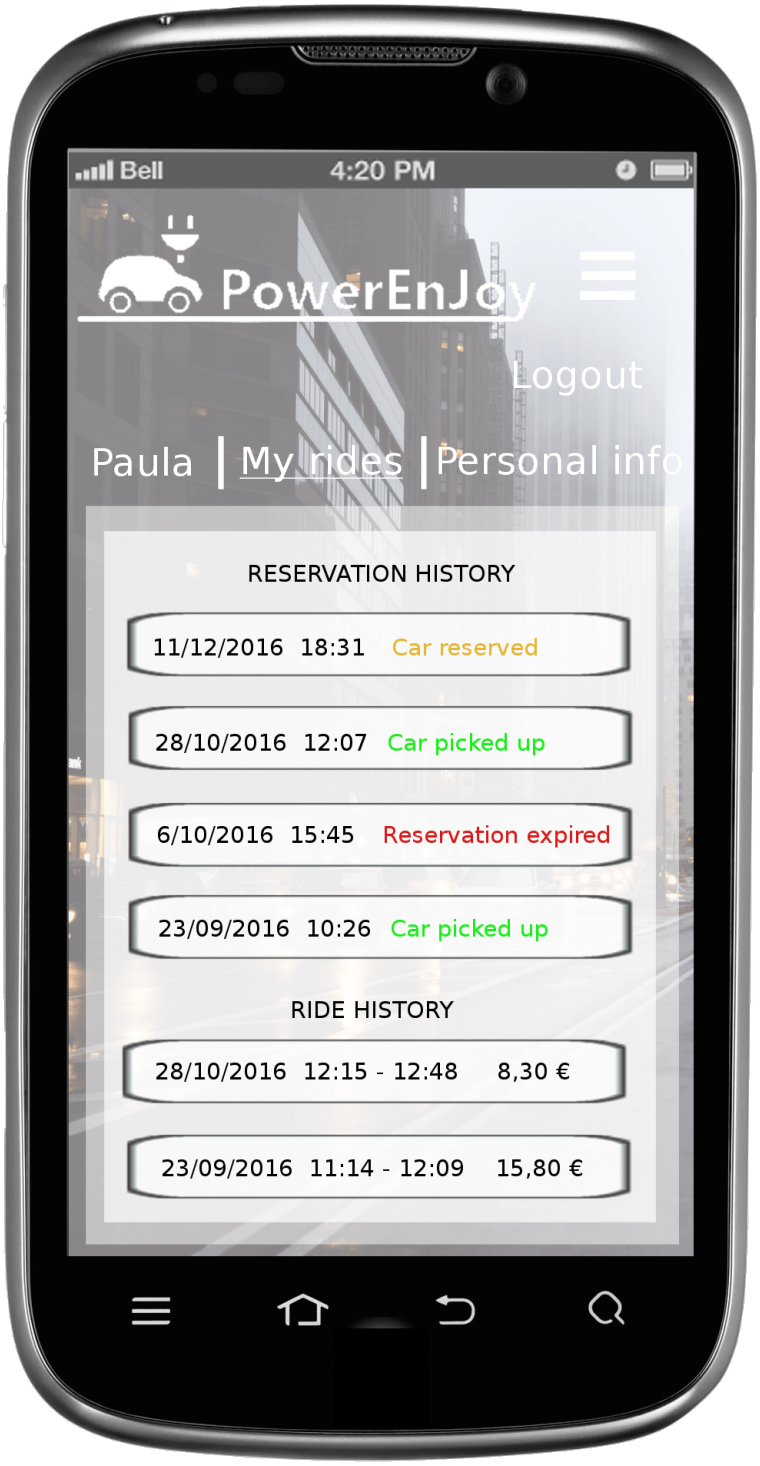
Mockup 5: The login page.

4.6 Personal profile



Mockup 6: The user's personal profile page.

4.7 My rides



Mockup 7: The list of rides and reservations.

4.8 Ride details

Mockup 8: The page with the details of a specific ride.

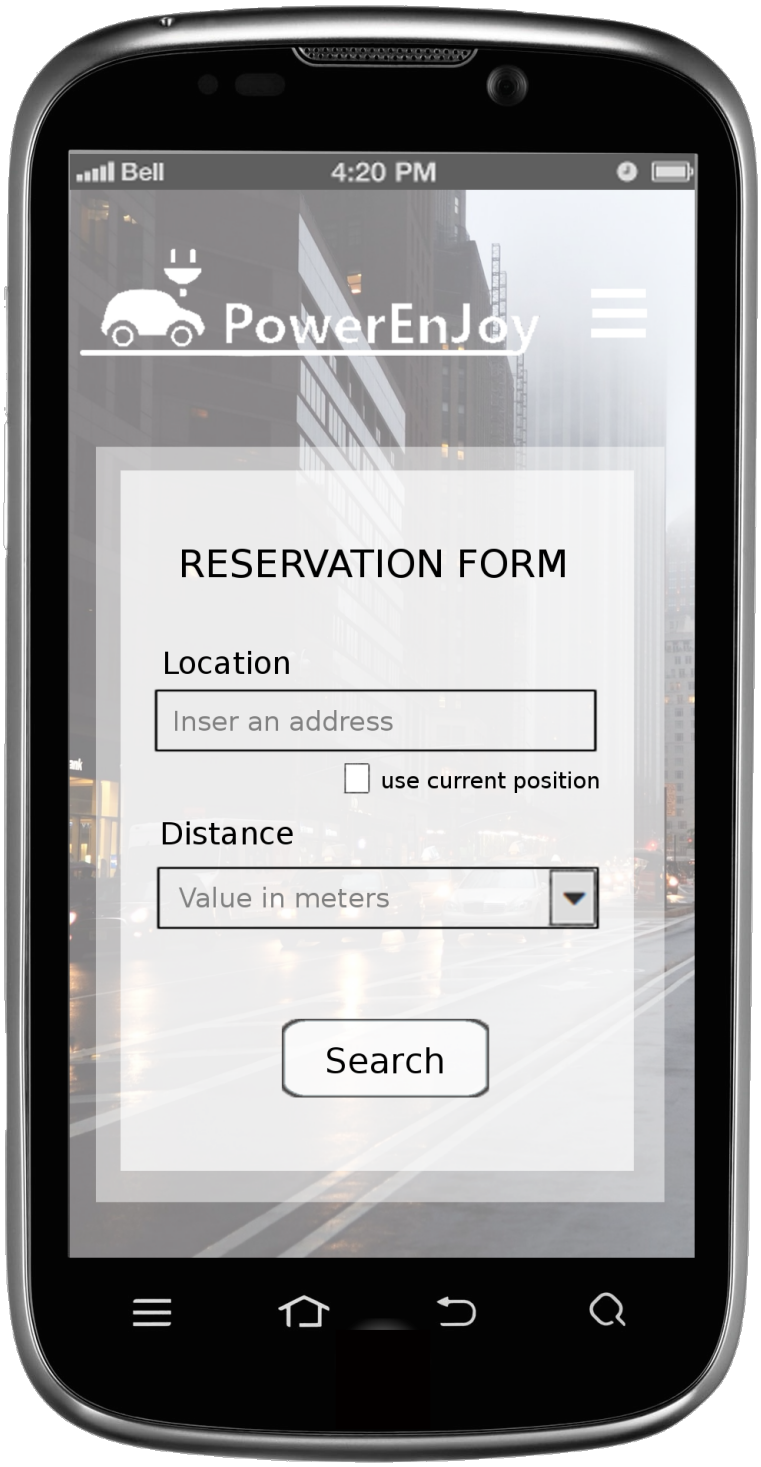
4.9 Personal information

Mockup 9: The page where user's personal information is shown.

4.10 Edit personal information

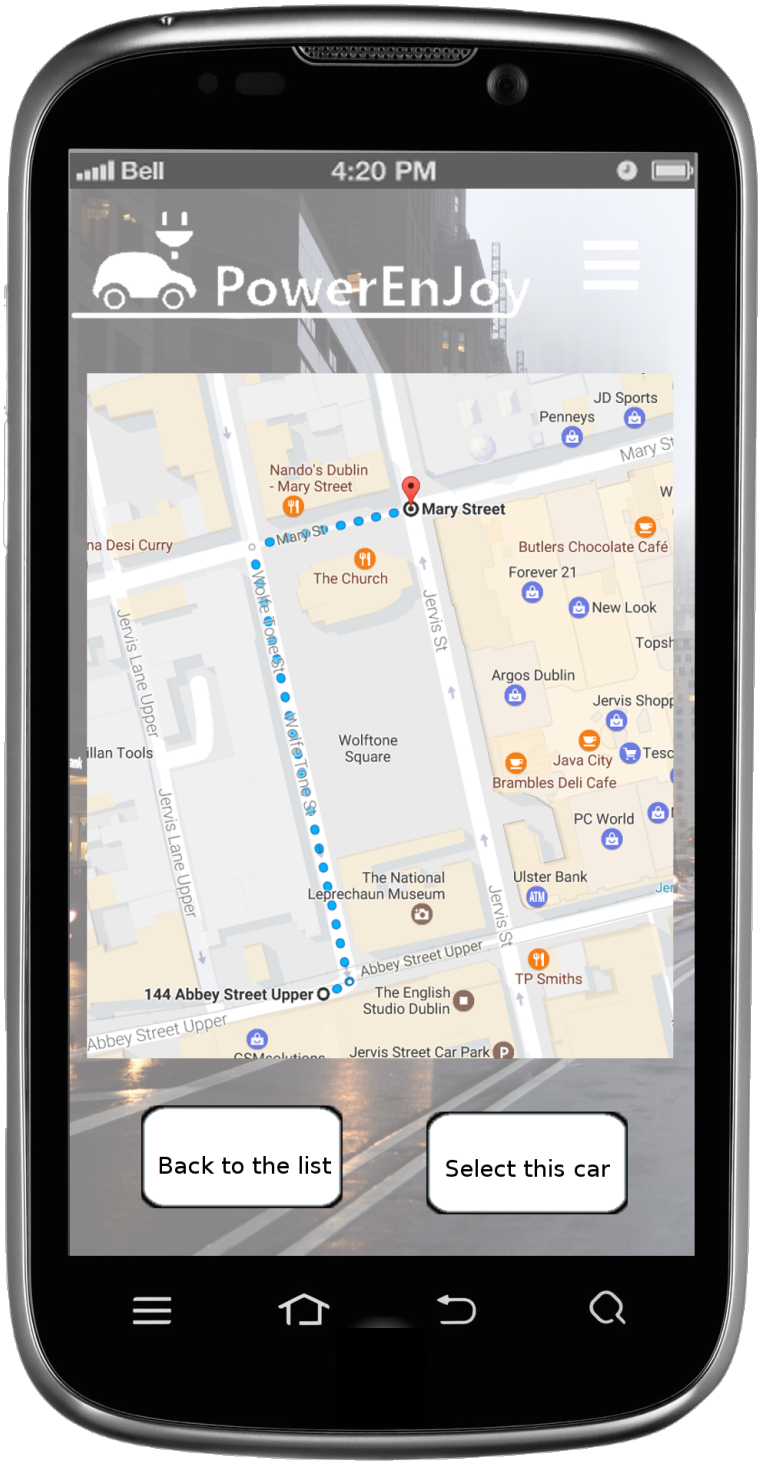
Mockup 10: The page where the user can edit his/her information.

4.11 Reserve a car



Mockup 11: The page where the user can insert research details.

Mockup 12: Here is the page where the list of the available cars is shown.



Mockup 13: The page where the car's location is shown on the map.

4.12 Car screen

Mockup 14: The interface of the car screen.

5 Requirements traceability

Here we are going to describe and explain how the goals and requirements defined in the RASD document map the components defined in this document.

- **Goal 1:** *Guests can register to the platform receiving back a login password.*
  The **UserHandler** should provide a 'sign up' functionality: it verifies the validity of user's inputs (driving license, credit card number) and sends a login password in less than five minutes.

- **Goal 2:** *Registered users must be able to search available cars.*
  The **UserHandler** provides the details chosen by the user for the car research (maximum radius and a specified address or the user's position). Then the **ReservationHandler** searches for available cars in the selected area according to the database, where there is information about the position of each car.

- **Goal 3:** *Registered user must be able to reserve a single car among all the available cars.*
  The **ReserveHandler** shall verify that the user is not reserving more than one car at a time, then it changes the car status from 'available' to 'reserved' and starts keeping track of the time elapsed.

- **Goal 4:** *Car reservation expires after one hour and a fee is charged to the user.*
  After one hour from the reservation, the **ReservationHandler** changes the status of the car from 'reserved' to 'available' and the **Payment Handler** charges the user 1 euro for the reservation expiration.

- **Goal 5:** *Unlock the car when the user who reserved it is closer than a defined distance.*
  When the user clicks on the 'unlock the reserved car' button, the **ReservationHandler** checks the distance between the car and the user, then changes the car status from 'reserved' to 'in use' while the **CarHandler** unlocks the car.

- **Goal 6:** *The user is charged on a per minute basis from the time when the ride begins.*
  When the car is unlocked, the **RideHandler** starts keeping track of the time elapsed and updates the total cost of the ride on a per minute basis, using the current elapsed time.

- **Goal 7:** *The user is notified of the current charges through the car display.*

The **CarHandler** sends the current cost of the ride to the car. The current cost is shown on the car display.

- **Goal 8:** *At the end of the ride the car is locked automatically and the user is charged.*
  As soon as all passergers got out of the car, the **CarHandler** locks the car, the **RideHandler** waits five minutes and then checks possible discounts or additional fees and calculates the total amount. The **PaymentHandler** charges the user the total amount. At the end of the payment, the **UserHandler** sends the user an email containing the details of the payment.

- **Goal 9:** *Suspend the account for insolvent users and redirect them to the customer service.*
  At the end of the ride, the **PaymentHanlder** charges the user and, if the payment fails, the **RideHandler** changes the user account status from 'active' to 'insolvent' and the **UserHandler** sends the user and email telling to get in touch with the customer service.

- **Goal 10:** *Discourage parking outside of safe areas by charging 80% more on the ride balance and if that happens, mark the car as unavailable.*
  At the end of the ride, the **CarHandler** updates the database with the car location and the **AreaHandler** checks if the car is in a SafeArea. If the car isn't in a SafeArea, the **RideHandler** adds a fee of 80% on the ride balance and changes the car status to 'unavailable'.

- **Goal 11:** *The car display provides information about the location of safe areas and power grid stations.*
  The **AreaHandler** searches for SafeAreas nearby the car location and the **CarHandler** displays them on the car screen map.

- **Goal 12:** *Cars with less than 20% of battery left are marked as unavailable.*
  At the end of the ride, the **CarHandler** updates the database with the battery level of the car and, if it's less than 20%, the **RideHandler** changes the car status to 'unavailable'.

- **Goal 13:** *Apply a 30% discount if there are more than 2 passengers.*
  At the end of the ride, the **CarHandler** updates the database with the number of passengers. If they are more than 2, the **RideHandler** applies a 10% discount on the cost of the ride.

- **Goal 14:** *Apply a 20% discount if the car is left with no more than 50% of battery empty.*
  At the end of the ride, the **CarHandler** updates the database with

the car battery level. If it's less than 50%, the **RideHandler** applies a 20% discount on the cost of the ride.

- **Goal 15:** *Apply a 30% discount if the car is plugged to the power grid at the end of the ride.*
  At the end of the ride, the **CarHandler** updates the database with the car location and specifying if the car is plugged to the powr grid, then the **AreaHandler** checks if the car is in a PowerGridStatiom. If the car is plugged in and is in a PowerGridStation, the **RideHandler** applies a 30% discount on the cost of the ride.

- **Goal 16:** *Charge an additional fee if the car is left more than 3km far from the nearest power grid station and with less then 20% of battery left.*
  At the end of the ride, the **CarHandler** updates the database with the car location and the level of battery, then the **AreaHandler** checks the distance between the car and the nearest PowerGridStation. If the car is at more than 3km far from the nearest PowerGridStation and with less than 20% of battery left, the **RideHandler** changes the car status to 'unavailable' and applies an additional a fee of 30% on the cost of the ride.

6 Effort spent

Section	Working hours	Author
Introduction	7	Tommaso Sardelli
Architectural design	25	Patrizia Porati, Tommaso Sardelli
Algotithm design	3	Tommaso Sardelli
User interface design	10	Patrizia Porati
Review	8	Patrizia Porati, Tommaso Sardelli
Graphic adjustments	3	Patrizia Porati, Tommaso Sardelli
RASD review	2	Patrizia Porati, Tommaso Sardelli

7 Used tools

Here is the list of the tools we used to create this document:

- *TeXstudio, Vim (with vimtex plugin) and TeX Live*: to write this doument
- *Dia and Umlet*: for UML models
- *Gimp*: to create mockups
- *Trello and Git*: for working coordination and cooperation

8 References

Here are some references used to redact this document:

[1] *PowerEnJoy - Requirement Analysis and Specification Document* - Patrizia Porati, Tommaso Sardelli

[2] *Onextrapixel - Web Design and Development: The Basics of Good Database Design in Web Development* - http://www.onextrapixel.com/2011/03/17/the-basics-of-good-database-design-in-web-development/

[3] *Web API Design: Crafting Interfaces that Developers Love* - https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf

[4] *Pattern: Database per service* - http://microservices.io/patterns/data/database-per-service.html

[5] *Microservice architecture patterns and best practices* - http://microservices.io/index.html

[6] *Capgemini - IS REST BEST IN A MICROSERVICES ARCHITECTURE?* - https://capgemini.github.io/architecture/is-rest-best-microservices/