

Power EnJoy Design Document

Niccolo' Raspa, Matteo Marinelli

December 8, 2016



Software Engineering 2 Course Project

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Scope	3
1.3	Definitions, Acronyms, Abbreviation	3
1.4	Reference Documents	4
1.5	Document Structure	4
2	Architectural Design	5
2.1	Overview	5
2.2	High Level Components	5
2.3	Component View	8
2.3.1	Application Server	9
2.3.2	Database	11
2.3.3	External Services	12
2.3.4	Client	12
2.4	Deployment View	12
2.5	Component Interfaces	12
2.6	Runtime View	13
2.7	Selected Architectural Styles and Pattern	20
3	Algorithm Design	20
4	UI Design	20
4.1	Mobile Application	20
4.2	Web Page	23
5	Requirements Traceability	24
6	Effort Spent	24

1 Introduction

1.1 Purpose

This is the Design Document for the Power Enjoy Service. It's aim is to provide a functional description of the main architectural components, their interfaces and their interactions, together with the algorithms to implement and the User Interface Design. Using UML standards, this document will show the structure of the system and the relationships between the modules. This document is written for project managers, developers, testers and Quality Assurance. It can be used for a structural overview to help maintenance and further development.

1.2 Scope

PowerEnjoy is a digital management system for a car-sharing service that exclusively employs electric cars. It allows registered clients (Power Users) to use a vehicle paying only on the basis of the actual use during each individual rental. For a more detail description of the domain and the requirements please refer to the Requirement and Specification Document.

The software system is divided into four layers, which will be presented in the document. The architecture has to be easily extensible and maintainable in order to provide new functionalities. Every component must be conveniently thin and must encapsulate a single functionality (high cohesion). The dependency between components has to be unidirectional and coupling must be avoided in order to increase the reusability of the modules.

Futhermore, to increase cohesion and decoupling as much individual components must not include too many unrelated functionalities and reduce inter-dependencies.

1.3 Definitions, Acronyms, Abbreviation

RASD: Requirements Analysis and Specification Document.

DD: Design Document.

DBMS: Relational Data Base Management System.

DB: Database layer,

UI: User Interface.

Backend: Term used to identify the Application server.

Frontend: The components which use the application server services (web front-end and the mobile applications).

SOA: Service Oriented Architecture.

JDBC: Java DataBase Connectivity.

JPA: Java Persistence API.
EJB: Enterprise JavaBean.
ACID: Atomicity, Consistency, Integrity and Durability.

1.4 Reference Documents

This document refers to the following documents:

- Project rules of the Software Engineering 2 project
- Requirement Analysis and Specification Document (from the previous delivery)

1.5 Document Structure

This document is structured in five parts:

Chapter 1: Introduction. This section provides general information about the DD document and the system to be developed.

Chapter 2: Architectural Design. This section shows the main components of the systems with their subcomponents and their relationships, along with their static and dynamic design. This section will also focus on design choices, styles, patterns and paradigms.

Chapter 3: Algorithm Design. This section will present and discuss the main algorithms for the core functions of the system, independently from their concrete implementation.

Chapter 4: User Interface Design. This section shows how the user interface will look like and behave, by means of concept graphics and UX modeling.

Chapter 5: Requirements Traceability This section shows how the requirements in the RASD are satisfied by the design choices, and which components will implement them.

2 Architectural Design

2.1 Overview

This chapter provides a comprehensive view over the system components, both at a physical and at a logical level. This description will follow a top-down approach, starting with the description of the high-level components and their relations and interactions. We will then reason and describe the single components and the functionalities they must implement. We will especially focus on the components that implement the core logic of our application and using sequence diagrams we will describe the runtime behaviour of the system.

We will also include deployment diagrams to show the physical implementation of the system.

2.2 High Level Components

Before describing the actual system architecture we introduce the high level components of our application. Following the requirements and the specification listed on the RASD, we identify what components are needed in order to implement them and only after we have detected them, we will focus on the architecture and explain our architectural decision and the technologies chosen. It's important to follow this process to generalize our design as much as possible and abstract from implementation details, in this way we are able to describe only the essence of our system.

The starting point to detect the main components is the Class Diagram described in the RASD.

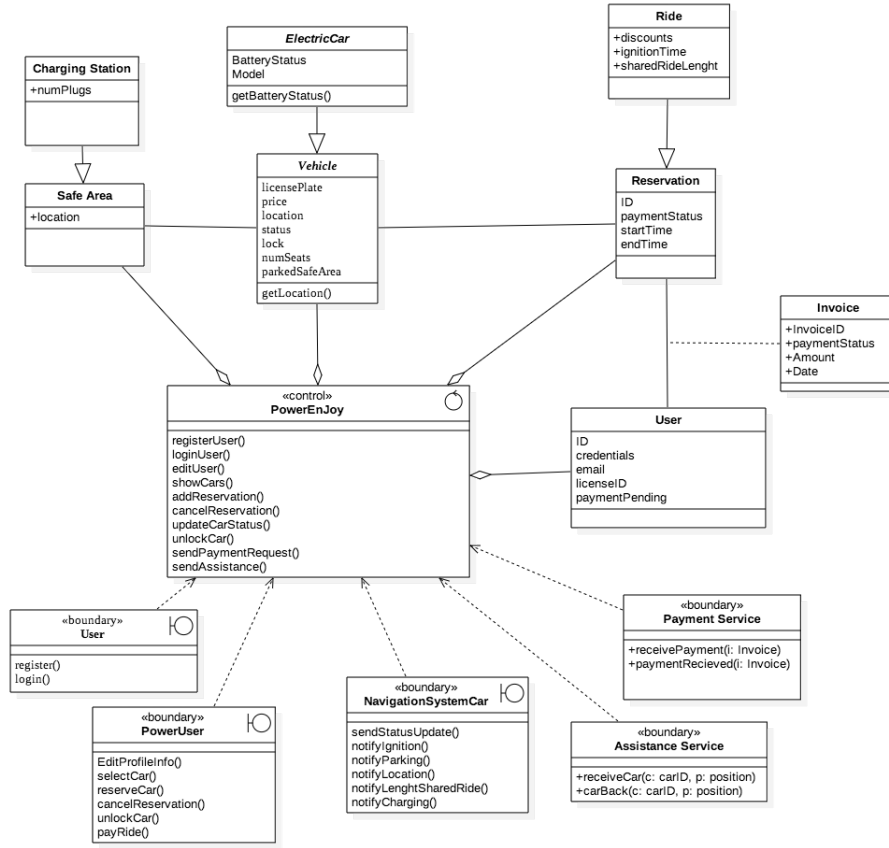


Figure 1: Class Diagram

The structure of the Diagram Suggests at least a three tiers architecture client-server based. The lower part of the Class Diagram shows the actors who are client of the PowerEnjoy section in the middle, the server in this case. Clients will be divided in proper Clients that are User and Power User, and NavigatorSystemCar, Assistaance Service and Payment Service as External Services. The central section, PowerEnjoy, is the core of the Power Enjoy services and, as just sad, is the server for Client layer and External Services Layer and Client of the last layer, the persistence layer. PowerEnjoy's layer will be the Application server. Charging Station, Safe Areas, Vehicle, Electric Car, Reservation, Ride, Invoice and User (the one on the right) are all in the persistence layer and obviously stored in a database. This layer is the server of the application server, who is the client this time.

The main high level components of the system are the following:

Mobile Application: Power Enjoy is a car sharing service therefore it must be implemented with mobility in mind. Since the majority of the mobile

devices have a GPS module and we need to have access to the user position for our application, it makes sense to require that the main user has our mobile application installed.

Application Server: This component contains all the logic for the system application. It will implement all the required functionalities and communicate both with the mobile application and the external services.

Web Server: This component does not contain any application logic, it's used to provide a web interface to the user. It helps to separate presentation from logic.

Web Browser: Using a web browser the user is able to communicate with the Web Server to obtain the required web pages.

Database: This component is responsible for data storage and retrieval which is crucial for our application. It will not implement any logic but it stores all the information needed for the correct functioning of our service. It must guarantee ACID properties and be accessible from the Application Server.

To give a complete overview of the system, we list also the external services which are not part of our system but with which the system depends to implement some functionalities (please refer to the RASD for a more detailed description).

High level components which are not part of our system:

Assistance Service: Power Enjoy is in charge of the management of the car-sharing system. All the secondary functionalities (recharging vehicles on-site, bringing cars back from unsafe areas and fixing malfunctions) are handled by an assistance service. This component provides an API to handle all the assistance request.

Payment Service: Every Power Enjoy user, in order to use our service, is required to have an account registered to a third party payment service, with a valid payment method. This component provides an API to handle all the payments functionalities.

Car On Board System: Every Power Enjoy vehicle comes with a pre-installed on board system which registers and notifies all the car activity. This component provides an API to monitor and remotely control every vehicle.

In the figure below we represent all the components listed above in a layered fashion, highlighting the relations among the different parts:

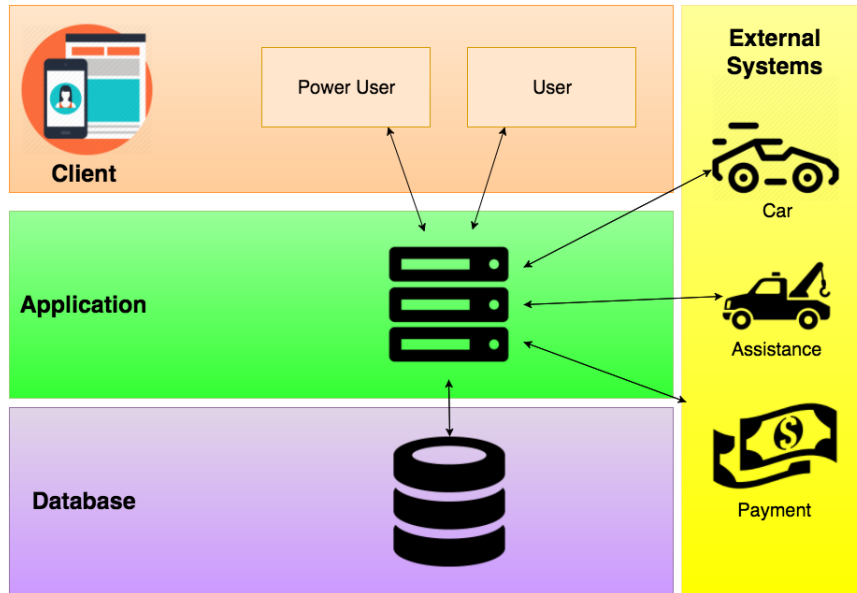


Figure 1: High Level Components

From this diagrams it's important to point out:

- This layered representation suggests a four tier architecture
- Separating Application Server and the Web Server improves scalability. We expect Power Enjoy usage to grow in different cities and in this way we're able to separate the tasks and optimize each layer individually to support increasing loads.
- The Application Server is the bottleneck of our system. Every other component is in relation with it, therefore its performance is strictly related to the performance of our system. But since every component expects different functionalities from the Application Server we can parallelize using threads and split the work load among different modules.

2.3 Component View

In this subsection we will look inside every single component and describe all the internal subcomponents. It's important to identify the relevant modules without increasing granularity too much. This will allow to have an efficient load balance in the present and it will be easy to integrate new functionalities in the future.

In a divide-and-conquer fashion for every component we will specify the implementation chosen and at the end we will show how to connect the single components.

2.3.1 Application Server

This components implements the logic of the Power Enjoy Application, it's the core of our business and in this part of the document we'll explore the subcomponents inside. To provide a natural continuation from the RASD, we will start from the Class Diagram, and we will focus on the control object. We will look at the singles control functions, logically group them in cohesive groups and map them in modules of our system.

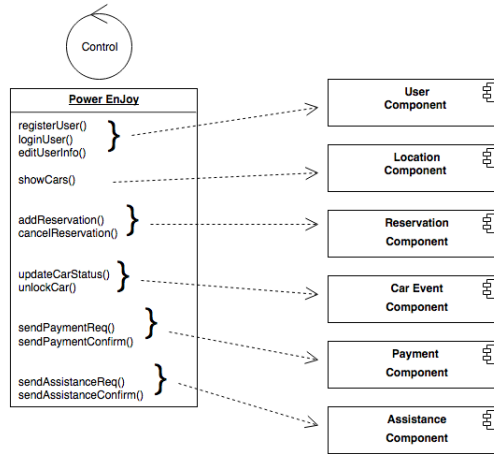


Figure 2: Mapping of control function to subcomponents of our system

A brief description of the functionalities expected from each module:

User Component This module provides the logic for a User and a Power User regarding all the user management features, namely: user login, user registration, user deletion, user profile editing. It will perform all the validation of the credentials received before inserting them into the system.

Location Component This module handles the position of each car and user of the system. It's used to show cars on the map and to perform proximity checks to unlock vehicles. It also stores location of Safe Areas and Charging Stations.

Reservation Component This module is responsible to manage all the current reservation and to accept new reservations from Power Users. and for avoiding undesirable behaviours (reservation of an unavailable car, double-booking and multiple reservations). It's strongly connected to the Car Event Component. This module keeps track of all the ride informations.

Car Event Component This modules interfaces with the API of the car on board system. It's a "low level" component that collects all the car data for other components to use and can be used to remotely control the vehicle.

It will signal to the interested component all the events of the car (car locking/unlocking, motor ignition, malfunctioning).

Payment Component This module interfaces with the API of the Payment Service to request payments and receive confirmations. It will not perform the fee calculation but it will receive the final price and check for price variations via other components. It will also flag/unflag users as banned in case of Pending Payment.

Assistance Component This module interfaces with the API of the Assistance Service to request assistance (recharge on site, fix malfunctions, bring car back from unsafe to safe areas) and receive confirmations once the assistance is provided. Once a car is fixed it will update the car information (e.g. new position, new battery level).

Time Component This is an utility component, it will be responsible for timing features such as expiration of the reservation and unsafe park timing. This helps to avoid the necessity of introducing stateful components.

Implementation Choice

This component will be implemented using:

- Java Enterprise Edition 7 (JEE7) - the platform incorporates a design based largely on modular components running on an application server which is a natural consequence from the description above. It also provides support for large-scale, multi-tiered, scalable, reliable, and secure network applications. This modularity helps to handle such complex system and it makes easy to insert the functionalities in the future.
- Enterprise JavaBeans (EJB) to encapsulate all the business logic of the modules described above.
- GlassFish as the Application Server - the server provides services such as security, transaction support, load balancing and supports the JEE7 platform.

The next figure will show the Application Component implemented as session beans logically grouped in EJBContainers.

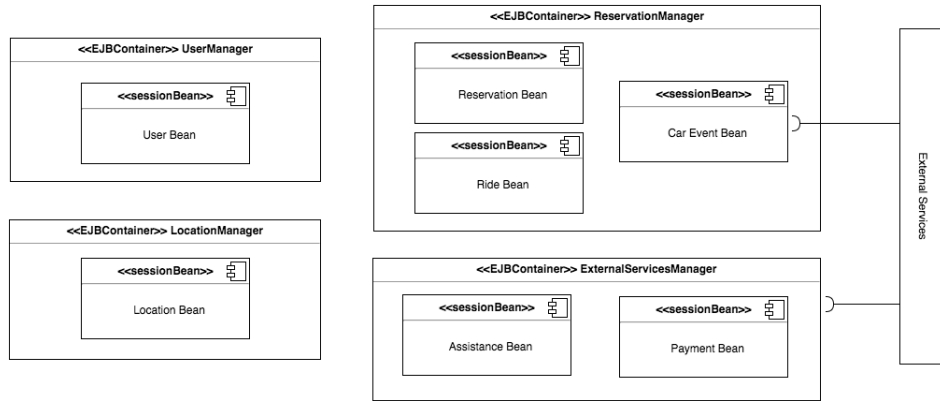


Figure 3: Application as Java Entity Beans

For the sake of clarity some information have been hidden from the diagram above but can be found in the next sections.

In Section 2.5 we will provide more details on the interfaces of each module.

In Section 2.6 using UML Sequence Diagrams we highlight the relationship between each component during a runtime analysis of our system.

2.3.2 Database

- JDBC/JPA JDBC (Java Database Connectivity) was chosen as connector between Database and the application server. Java database connectivity interface (JDBC), infact, is a software component that allows Java applications to interact with databases. To enhance the connection, JDBC requires drivers for each database. These drivers connect to the database and implement the protocol to transfer query and respective results between the client and database. JDBC was chosen because it is available for any DBMS also thanks to the ODBC bridge. ODBC infact allows programmers to make SQL requests that will access data from DB distinct without having to know the proprietary interfaces of each DB. In this way it is easier to change the database without altering the application layer. The Java Persistence API (JPA) is a Java application programming interface specification that describes the management of relational data in applications using Java Platform, Standard Edition and Java Platform, Enterprise Edition.
- The access to the DBMS is not implemented with direct SQL queries: instead, it is completely wrapped by the Java Persistence API (JPA). The object-relation mapping is done by entity beans. The Entity Beans representing the database entities (Figure 2.5) are strictly related to the entities of the ER diagram (Figure 2.4).

2.3.3 External Services

Qualche parola sugli external services

Car On Board System

aaaaaaa

Payment System

aaaaaaa

Assistance System

bbbbbbb

2.3.4 Client

- Spiegazione del perche' mobile-first.
- Spiegazione del perche' aggiungere un webserver
- Diagramma della mobile app (con gps ecc...)
- JAX-RS to implement proper RESTful APIs to interface with clients and the Web Server;
- • To interface with external systems, existing RESTful APIs defined by the partner (payment handlers, maintenance system) will be used.

2.4 Deployment View

In this subsection we'll move on the physical side of our application, describing his deployment with the support of UML diagrams.

2.5 Component Interfaces

Application Server to Database

JPI - DOA volendo

Application Server to Front-Ends

REST-API

Application Server

Dei bean

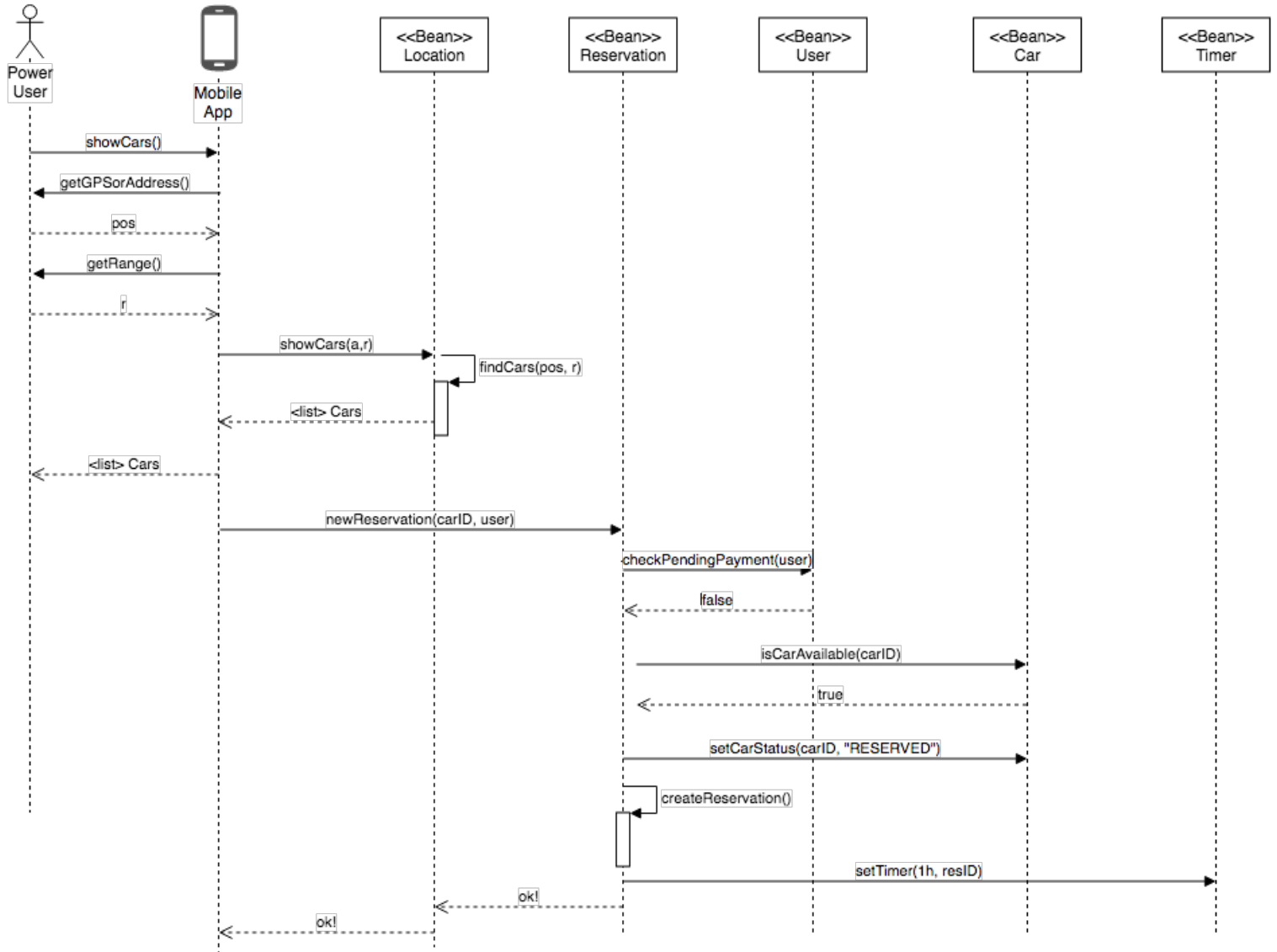
Application Server to External-System

REST-API

2.6 Runtime View

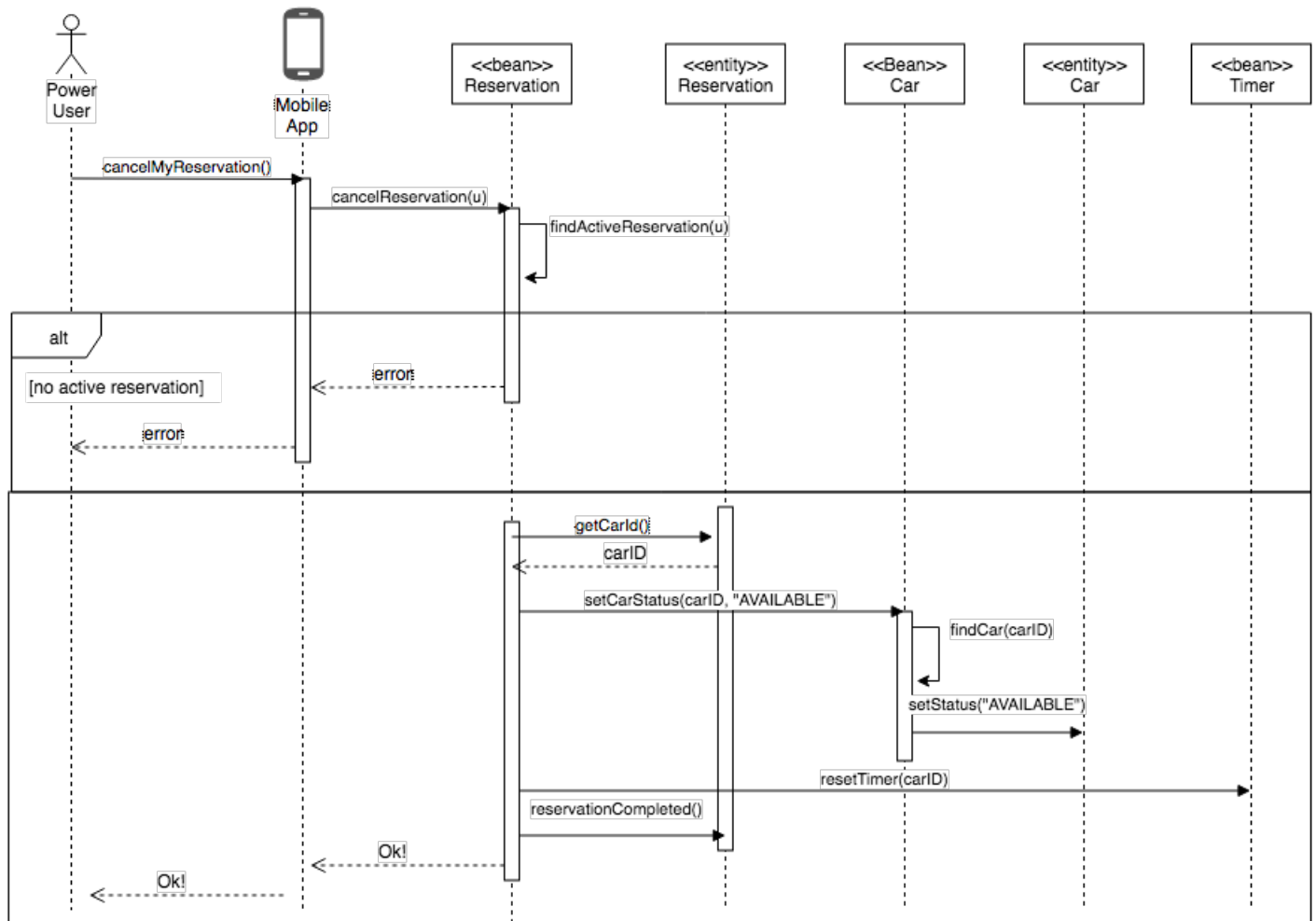
In this section we will describe the dynamic behaviour of the system. In particular, it will be shown how the software and logical components defined in section 2.3 interact one with another, using sequence diagrams for the more meaningful functionalities of the system. We decided not to represent the database in the sequence diagram, because the interaction with the database is totally abstracted by the entities via the Java Persistence API.

Create Reservation

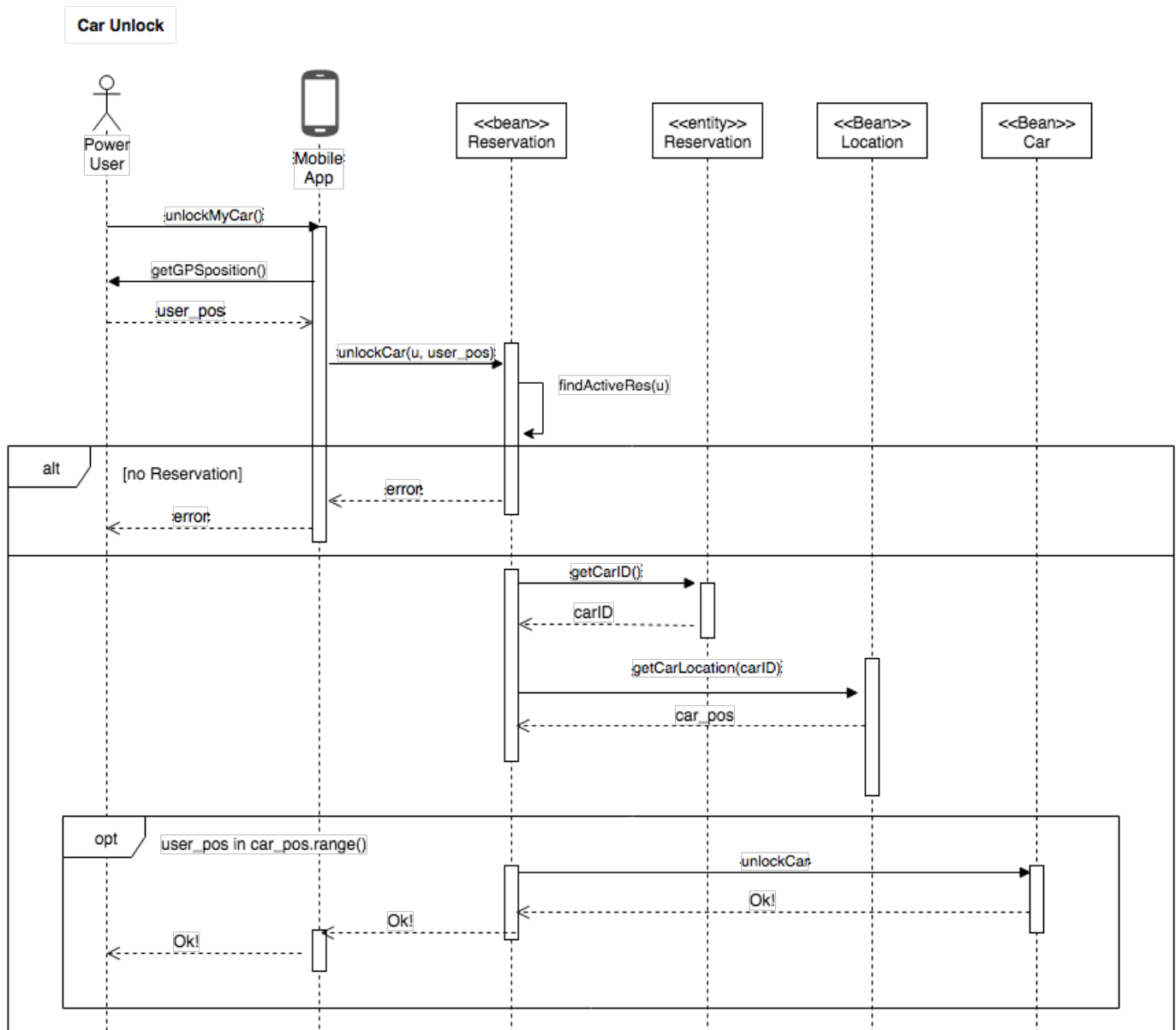


Cancel Reservation

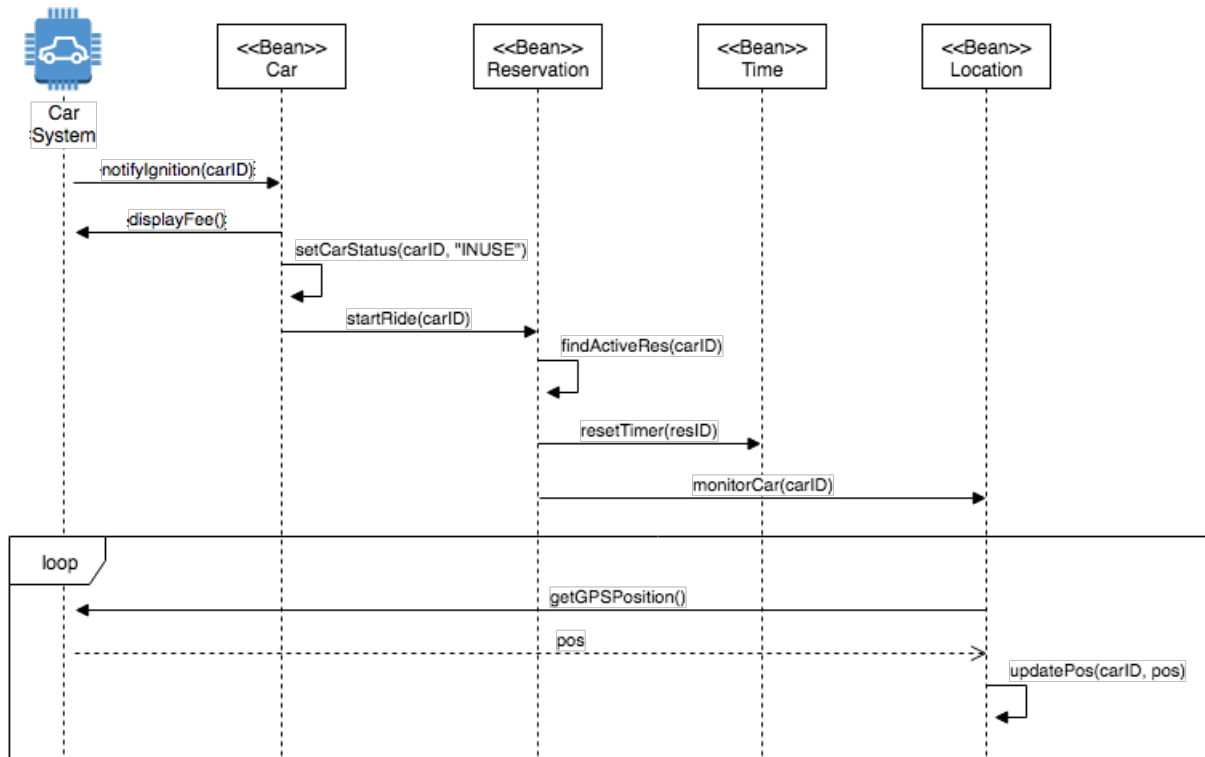
Cancel Reservation



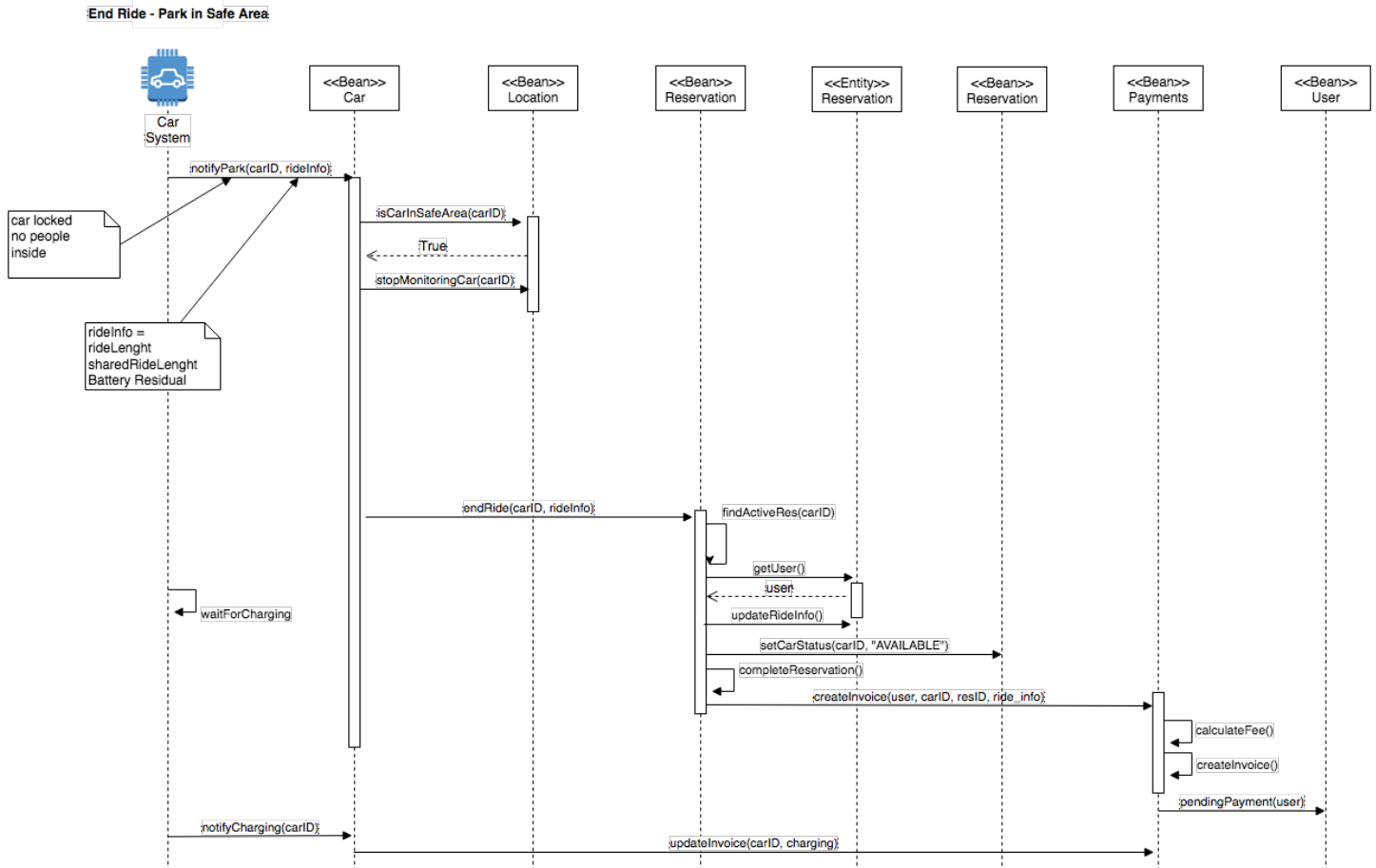
Unlock Car



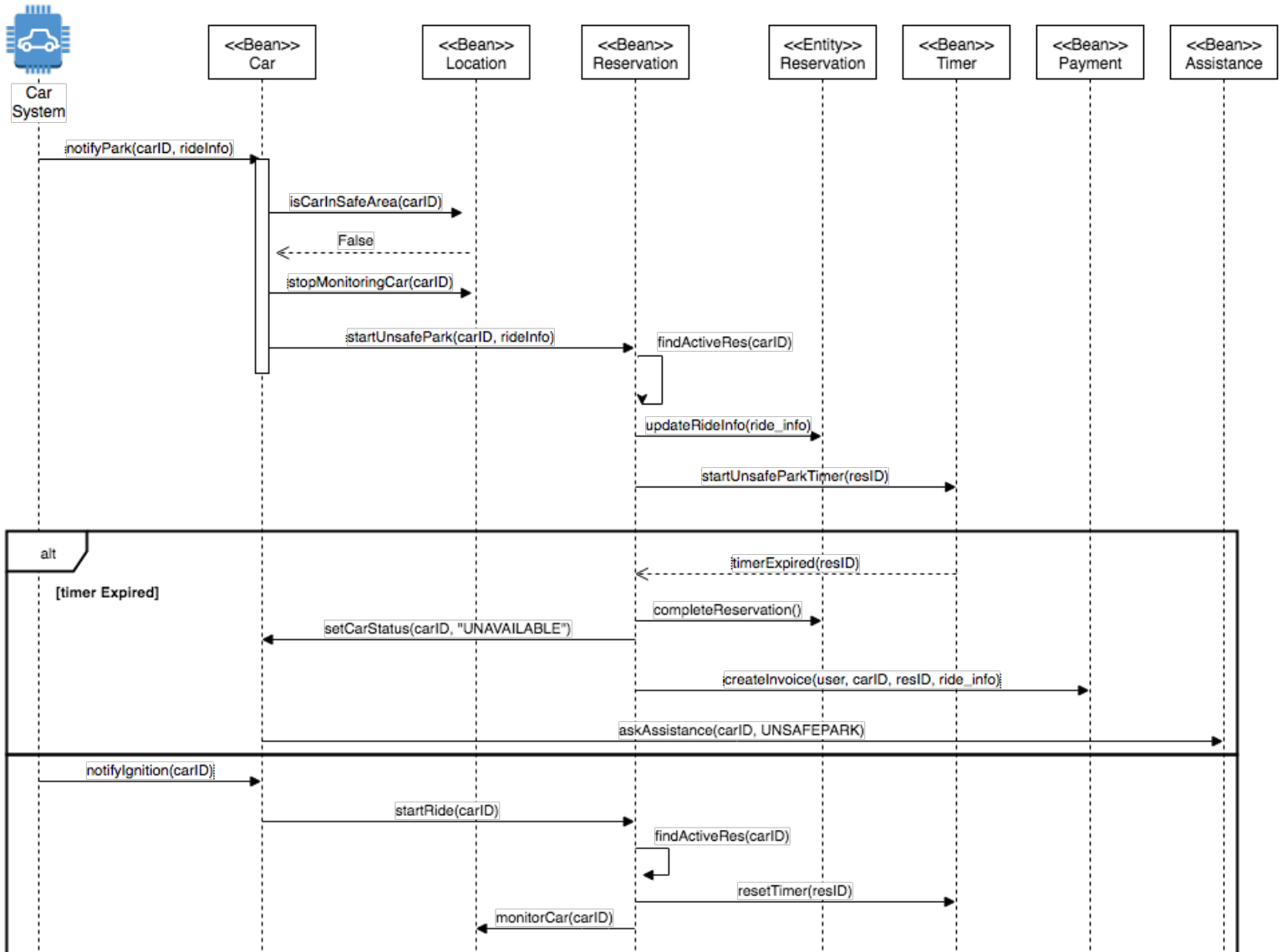
Start Ride



End Ride (Safe Park)



End Ride (Unsafe Park)



2.7 Selected Architectural Styles and Pattern

Explain styles/patterns used and why/how

3 Algorithm Design

Focus of the most relevant algorithmic part: FIND_ZONE()

4 UI Design

4.1 Mobile Application

The mobile application, as usual, will have a recognizable icon that can be added on the desktop. When the application is opened, the display will show the login screen.

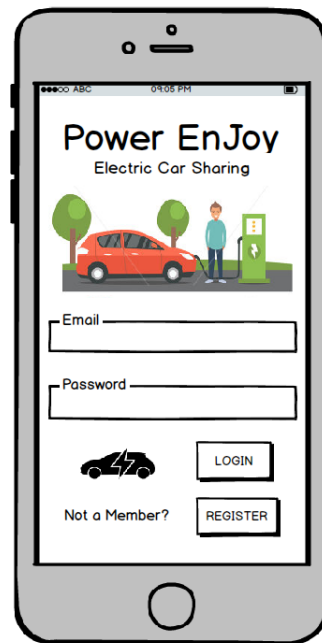


Figure 1: Mockup

E-mail and Password are required for the login. Insert wrong credential cause the refresh of the page with a notification explaining the problem, without giving information about the wrong field. In this screen is possible to register new Users by the button "REGISTER". Clicking leads to the registration screen, it is a form who must be filled entirely and correctly to have a successfull registration. The fields must be at least "name", "Surname", "Driving Licence", "E-Mail"

and "Payment System", add other fields could be helpful but not strictly necessary. If some data are not acceptable, a notification will be displayed and the form will be reuploaded. After the login, the display shows the Main Screen.

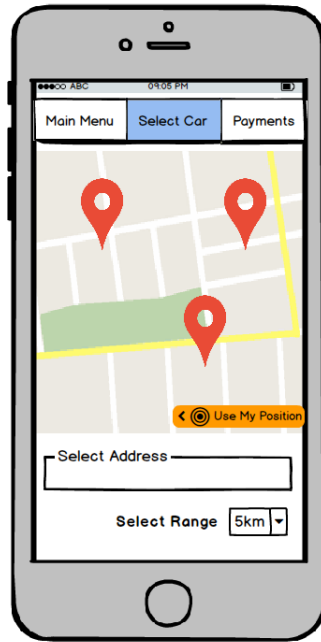


Figure 1: Mockup

On the screen is possible to see the map of the city, if is available the center will be the user position, if it is not available the center will be the center of the city. In the upper side there is a menu with the sections "Main Menu", "Select Car" and "Payments". Main Menu open a list of fields. It contains the field "Modify Profile" that lead to a screen equal to the Registration Screen previously described. In the Main Menu list is auspicious to put all functionalities not strictly related to the reservation and the payment, this will make more clean and simple the application. Payments allows to pay the unpaid fee, if there is. If there are no unpaid fee will be shown a message like "You payed the last fee, there is nothing to pay more". If there is a payment pending the message will be "There is a pending payment. You have to pay 5\$ (example)." and a button will start the payment procedure. Select car is the basic screen of the application. In this section is possible to set the center point of research and the research range, user should be guided to choose his position as center even if is possible to put any position as center of the research. When center and range are available for the server, it starts to search cars. Cars will be shown on the map and user can click on them to see the car informations.

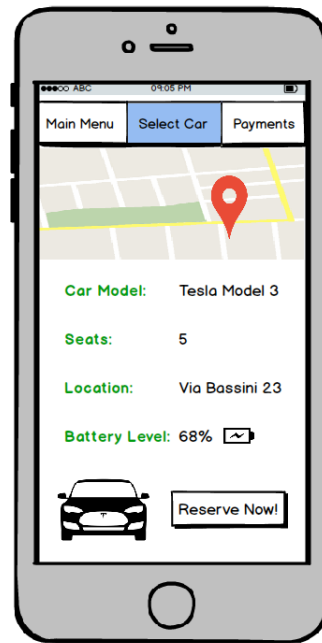


Figure 1: Mockup

Information must be about the battery and the location, could be useful add other information as model and number of sets. A button "Reserve Now" will start the reservation. When the reservation starts the display will show a one hour countdown, the timer indicates the time to the expiration of the reservation. An unlock button and a cancel button will be in the screen as well. If the unlock button is pushed not near enough, will be displayed a message that notify the problem. When the car is ignited the countdown stops and the application go in standby, in the logic the application switch from reservation to ride. Only the application is in standby, not the entire phone. Application will wake up when the car is turned off. If the car is in an unsafe area, "You left car in an unsafe area" will be displayed together with an unlock button and a one hour countdown. At the end of the countdown the ride will end. If the car is in a safe area the ride will end. When a reservation or a ride ends, a payment notification is shown. It should be like "Your reservation has ended. Your fee is 8€. Thanks for using Power Enjoy Service". A confirmation button allows to go back to the Main Screen. The payment message is not shown if the reservation is canceled. It is important to underline that on the payment notification will NEVER appear the word "ride": ride is useful for the logic but its existence is useless for the user, so is useful to avoid the use of two terms. Every time is not specified, the "go back" button of every mobile system will accomplish the "go back" function.

The application must be as simple as possible, the main idea is to make every

marginal utilities, like the main menu options, obscured by the main functionality. This reasoning increase usability thanks to the focus on the reservation.

The language to write the application obviously will change from a operating system to another. The application must run on Android and iOS. For Android the language will be Java and for iOS the language will be Swift.

4.2 Web Page

Web Page is a support of the application. The presence of a download link should be as constant as possible.

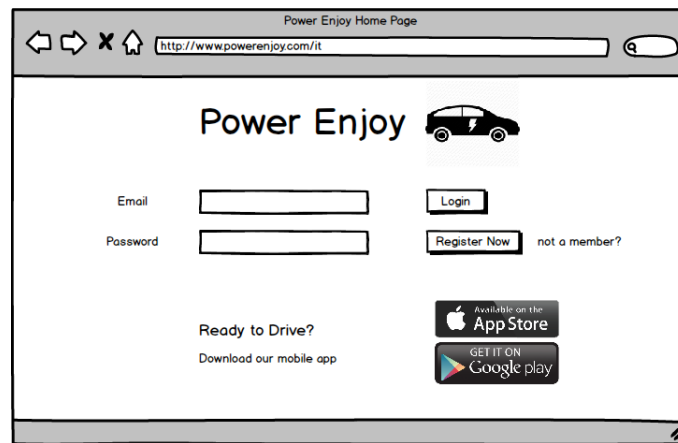


Figure 1: Mockup

The main purpose of the web page is to give the possibility to the user to have a more efficient way to manage side functions. The app will focus on the reservations, and the web page allow to have a better interface for the options of the main menu list. The main page is very similar to the access screen of the application and the consequences of the buttons are the very same.

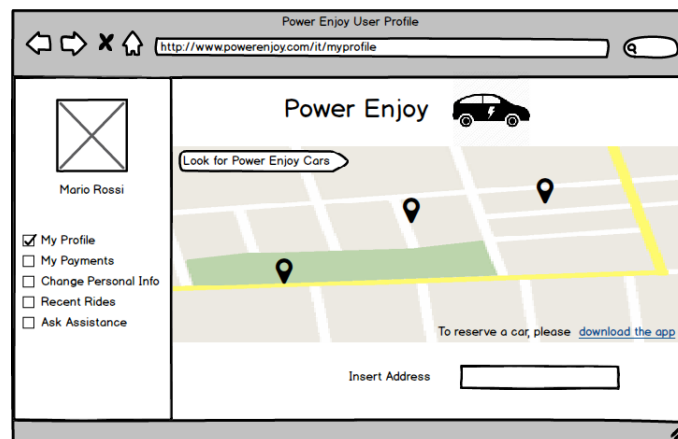


Figure 1: Mockup

On the main page there is a list on the left. From the lists fields is easy to manage the profile, from the personal information to the history of payments. On the main page is also possible to see Power Enjoy Cars distributed on the map. This possibility doesn't allow to reserve a car, this is possible only via application

The web page should be written in XML or JSON, to allow quick and efficient data transfer through textual data files over HTTPS.

5 Requirements Traceability

Explain how requirements defined in the RASD map to the design elements that you have defined in this document.

6 Effort Spent

Niccolo' 20 Ore