POLITECNICO
MILANO 1863

Design Document

Joan Ficapal Vila (876805), Nicolò Vendramin (879113)

December 11, 2016
v1.0

Revision History

Revision	Date	Author(s)	Description
0.1	19.11.16	N and J	First draft of the introduction
0.2	21.11.16	N and J	Started to design the high level architecture
0.3	23.11.16	N and J	Defined the high level architecture and Component view
0.4	26.11.16	N and J	Completed the section related to architecture
0.5	27.11.16	N	Algorythm design, and UX diagram
0.6	2.12.16	J	Designed some mockups
0.7	3.12.16	N and J	Definition of some runtime views
0.8	4.12.16	N and J	Runtime views concluded
0.9	5.12.16	N and J	Concluded the mockups
0.10	6.12.16	N and J	Added the mockups to the document
0.11	7.12.16	J	Definition of the Traceability Matrix
0.12	7.12.16	N	Document checking and formatting
0.13	9.12.16	J	Correction of some errors in the schemes
0.14	10.12.16	N and J	Inclusion of the Data paragraph

Hours of work

- Joan Ficapal Vila : 19 hours
- Nicolò Vendramin : 17 hours

Contents

1 Introduction

1.1 Purpose

The design document describes both the architectural and technical details of PowerEnjoy. This information is built in order to give to the software development teams a better understanding of how the system works, including an high level definition of the components and their interactions and interfaces. This document assume as a reference the Requirements Analysis and Specifications Document which has been produced and released.

1.2 Scope

The software system to be developed it's a service that allows clients to use environmental-friendly shared cars, paying proportionally to their usage of the car. The system is provided to the customer through a mobile application available for iOS and Android, which will be from now on referenced as the "Customer application", and its web server below called Application Server. In this document the relation between them and other related component will be defined in order to accomplish the system's goals. In this document in order to make clear the interaction between components are mentioned also two external parts that are already implemented: the billing system and the existing system. During the development must be kept in mind the fact that the goal of the application is to provide an optimized, cheap and simple car sharing service based on electric car. For this reason, and in particular to ensure the economicity of the service is important to guarantee an optimal data exchange between components. Semplicity is a very important feature as well and imposes the development team the task of coping with an user experience as simple and clear as possible. In addition the system must be designed taking into account its maintainability and scalability.

1.3 Definitions, Acronyms and Abbreviations

1.3.1 Acronyms

- **API:** Access point of Interface. This term is used to indicate the interfaces exposed to access a software service from another software service.
- **JEE:** Java Enterprise Edition. The version of the java language specifically designed for professional software development for companies.
- **DBMS:** Data Base Management System. It is the software used to query the database.
- **JDBC:** Java Data Base Connectivity. It is the driver component for databases which allows access and persistance of data for every programme written in the Java Language.
- **MVC:** Model View Controller. Typical design paradigm which splits the responsabilities of the components in 3 different layers.
- **DD:** Design Document. This document.
- **RASD:** Requirements Analysis and Specifications Document. The refernce document for the specifications of the system to be developed.
- **HTTPS:** Secure Hyper Text Transfer Protocol. It is a protocol for secure communication over a computer network which is widely used on the Internet.

- **XML:** eXtendible Markup Language. It is a language designed to store and transport data.
- **UX:** User eXperience. User experience is the interaction between the user and the system seen from the perspective of the user.
- **E.E.:** Execution Environment. The environment in which an application runs.
- **OS:** Operating System.
- **LAMJ:** Linux Apache MySql Java. This Acronym is used to indicate a particular configuration of a web server with an execution environment composed by an Apache + MySql + Java packet running on a Linux machine.

1.4 Reference Documents

- PowerEnjoy RASD Document.
- Course Slides.
- Templates provided at lesson.

1.5 Document Structure

The structure of the document is organized in 5 parts.
Following this section (Introduction) there is Section 2 that is the part of the document in which the design of the architecture is described, detailing the components form the system, their interfaces and explaining the reasons of the architectural choices.
Section 3 deals with the design of the difficult parts of software composing the application.
In section 4 some mock ups of the desired interface are presented as well as a simple schematic UX diagram used to describe more in detail the presentation layer of the application.
Section 5 is dedicated to the definition of the requirement traceability matrix, in which is performed the mapping between any requirement and the component that fullfills it.

2 Architectural Design

2.1 Overview

This section includes a description of the elements that compose the architecture of the software to be developed, from an high level perspective. In order to make clear the interfaces of the components, the interactions between them at runtime we are going to attach later in this document a Component View, a Deployment View and some detailed Sequence diagrams specifying the most complex interactions. In the description of the solution to be adopted there is also an indication for the developing teams about the design patterns that they are asked to follow during the implementation of the product.

2.2 High Level Components and their Interactions

The software must be designed to follow an MVC pattern to be placed on a three tier architecture. The three layer defined by the MVC splitting are mapped on the three tiers in a one to one relationship. In the following subsections the components at each level of the architecture are going to be introduced.

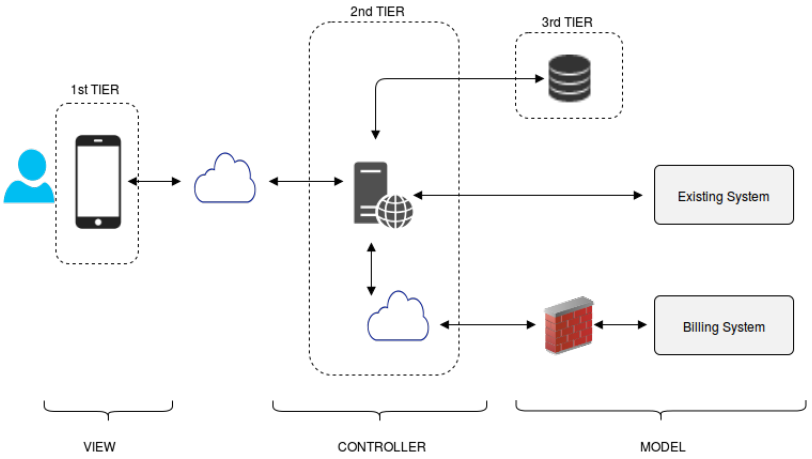

Figure 1: High Level Architectural Representation

2.2.1 Model Components

I **DBMS :** This component manages the private data of the service, the one related to the credentials of the users, in addition obviously to the data about bookings and cancellations. It interacts with the control layer only.

II **Existing System :** The exising system is not object of the design, since it is already implemented, but must be considered as part of the model from the point of view of our application. As a matter of fact it is the major source of information for the system: it provides a view on his private database with which the controller can access all the information that he needs about the status of the cars of the service. It interacts only with the controller receiving queries on datas and/or updates to be done on their status.

2.2.2 Controller Components

I Application Server : The controller is made up of one big component inside which several small components provide the different functionalities implemented by the control layer. The controller component is in the middle between the view and the model the controller interacts with all the other components : DBMS, Existing System and Customer Application (through the Request Manager), and also with the external components used for the billing procedure which is the so called Billing System. Here it follows a more detailed explanation of the sub-components that are included in this layer:

II Request Manager : The request manager is a component that realizes the interfaces the customer application with the controller by prompting the right sub-components of the Application Server in order to complete the requestes received from the client.

- **i** Registration/Login Manager : this component is meant to accept requests for registration or credential validation during the login phase.
- **ii** Pricing and Discount Manager : this component must be designed in order to be able to apply to each ride the correct fee, including also possible discounts or extra charges.
- **iii** Service Manager : this component is meant to provide all the functionalities related to the very fruition of the service, so handling the reservations , the active rides and all the request related to those functionalities.
- **iv** Resource Manager : the resource manager is the one that is queried in order to get informations about the resources of the system in terms of handling the data about them (e.g: cars, users, safe areas ...).

2.2.3 View Components

I Customer Application : The mobile application with which the customer access the service is the only component that constitutes the presentation layer. The application that here, from an high level perspective is considered only as a view, being a sub-system itself is built following an MVC pattern. In the following list we just briefly summarize how the pattern will be implemented inside this component:

Model The application chaces some data in order to be able to continue operating even if the connection drops temporarily. In particular all the data displayed by the last "refresh" of each of the pages of the application is stored in order to be rapidly reloaded.

View The view part of the application is made up of the pages used by the application to display the content and needed to provide access to the service.

Controller The controller part in the application is very slim and it's just the logic that handles the data to be displayed on the view, fetching informations from the server if connection is available or restoring the previous state from the cached data in case of connection unavailability, and other similar operations.

2.3 Components' View

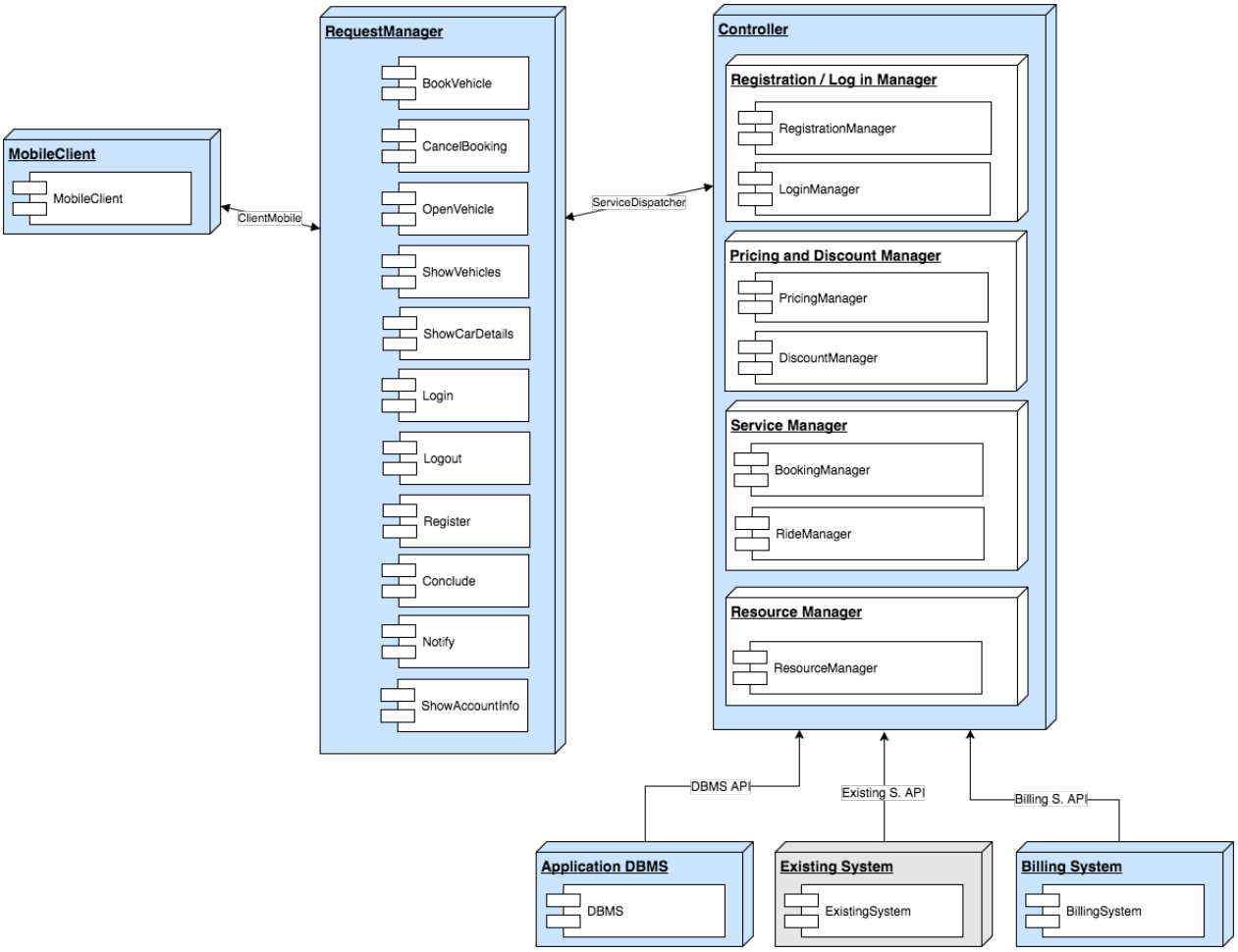

Figure 2: Components' View

2.4 Data

2.4.1 Wrapping Data Strucures

Here it follows a simplified version of the class diagram in which relations between the employed data structures are highlighted. In this scheme the focus is on the data structures used internally in the controller to wrap and manipulate the objects.

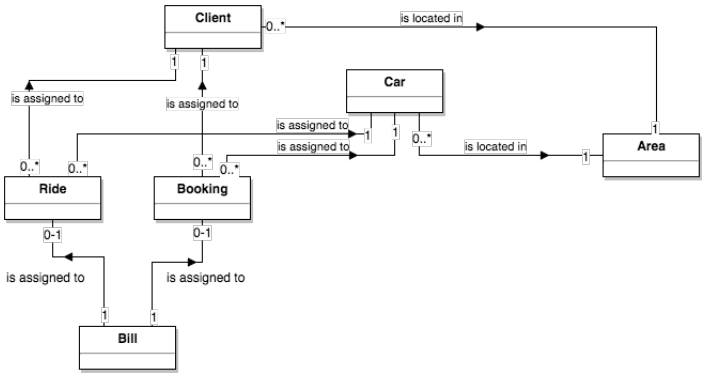

Figure 3: Deployment View

2.4.2 The role of the private Database

In order to clarify the functions of the private database of the application we dedicate this paragraph to its description. The private database is used by the application to store some temporary data cached during the usage. informations about the cars are cached in the database every time an update is done by the application. The cached data are used when retrieving information during the non critical operations. In case of booking, opening the vehicle and other critical operations the existing system is queried for the actualized data in order to avoid inconsinstencies in those acivities.

2.5 Runtime View

In order to better detail the functioning of the system at runtime, we detail the most important operations in terms of interactions between components.

2.5.1 Register

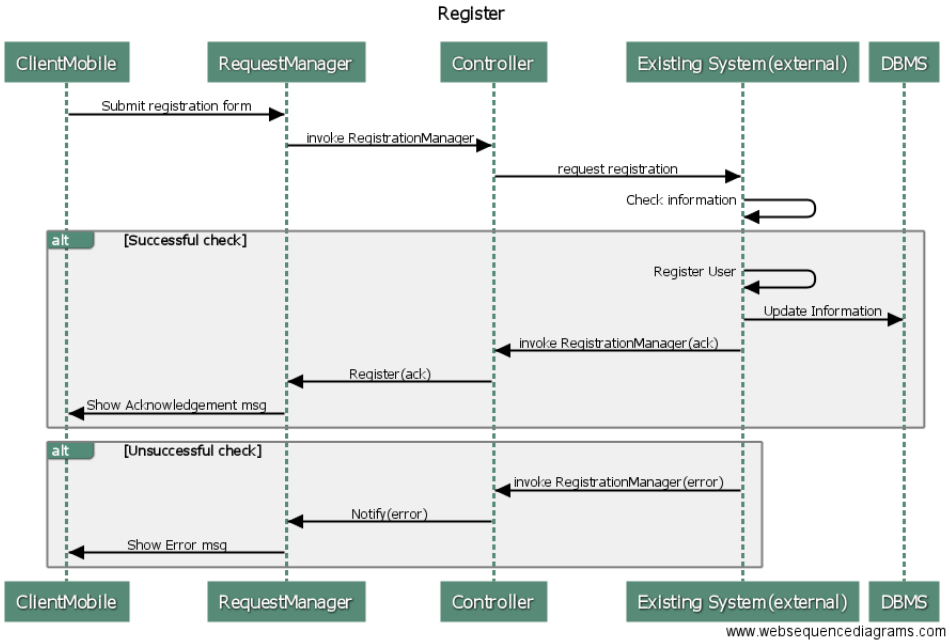

Figure 4: Registration detailed sequence diagram

2.5.2 Log in

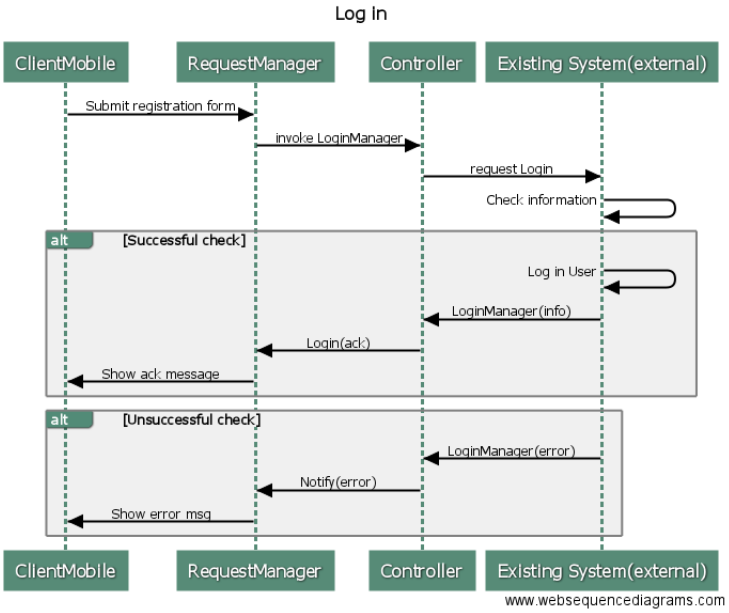

Figure 5: Log in detailed sequence diagram

2.5.3 Show Account Details

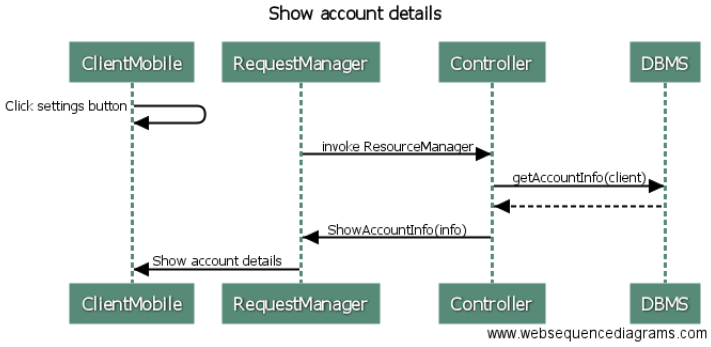

Figure 6: Show Account Details detailed sequence diagram

2.5.4 Find Cars Next to a Specified Position

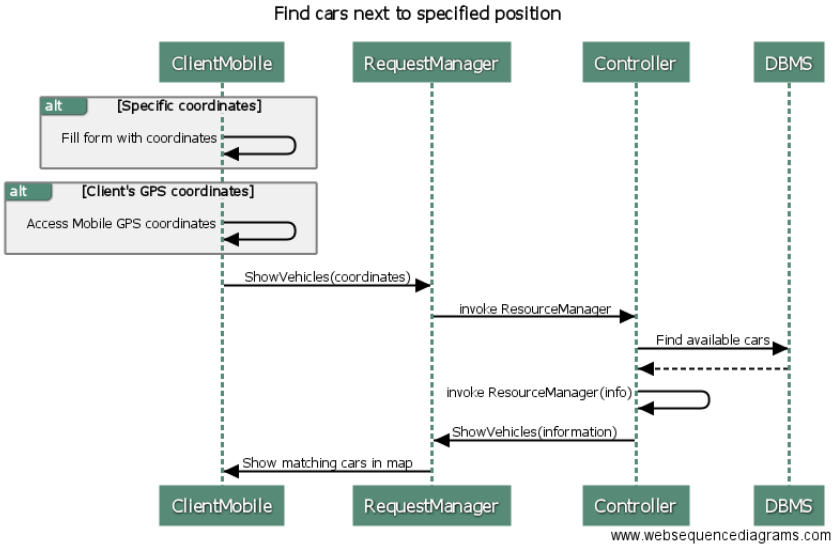

Figure 7: Find Cars Next to a Specified Position detailed sequence diagram

2.5.5 Show Details

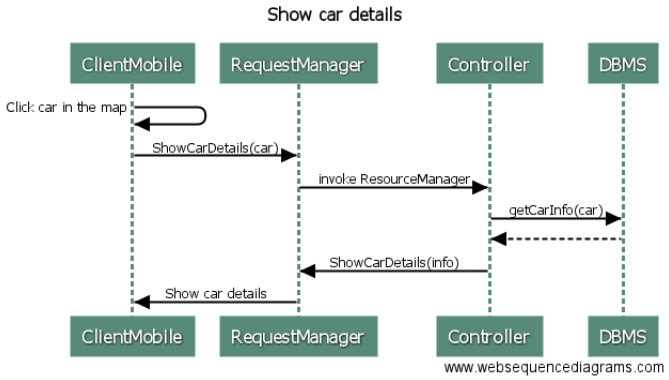

Figure 8: Show Details detailed sequence diagram

2.5.6 Book a Vehicle

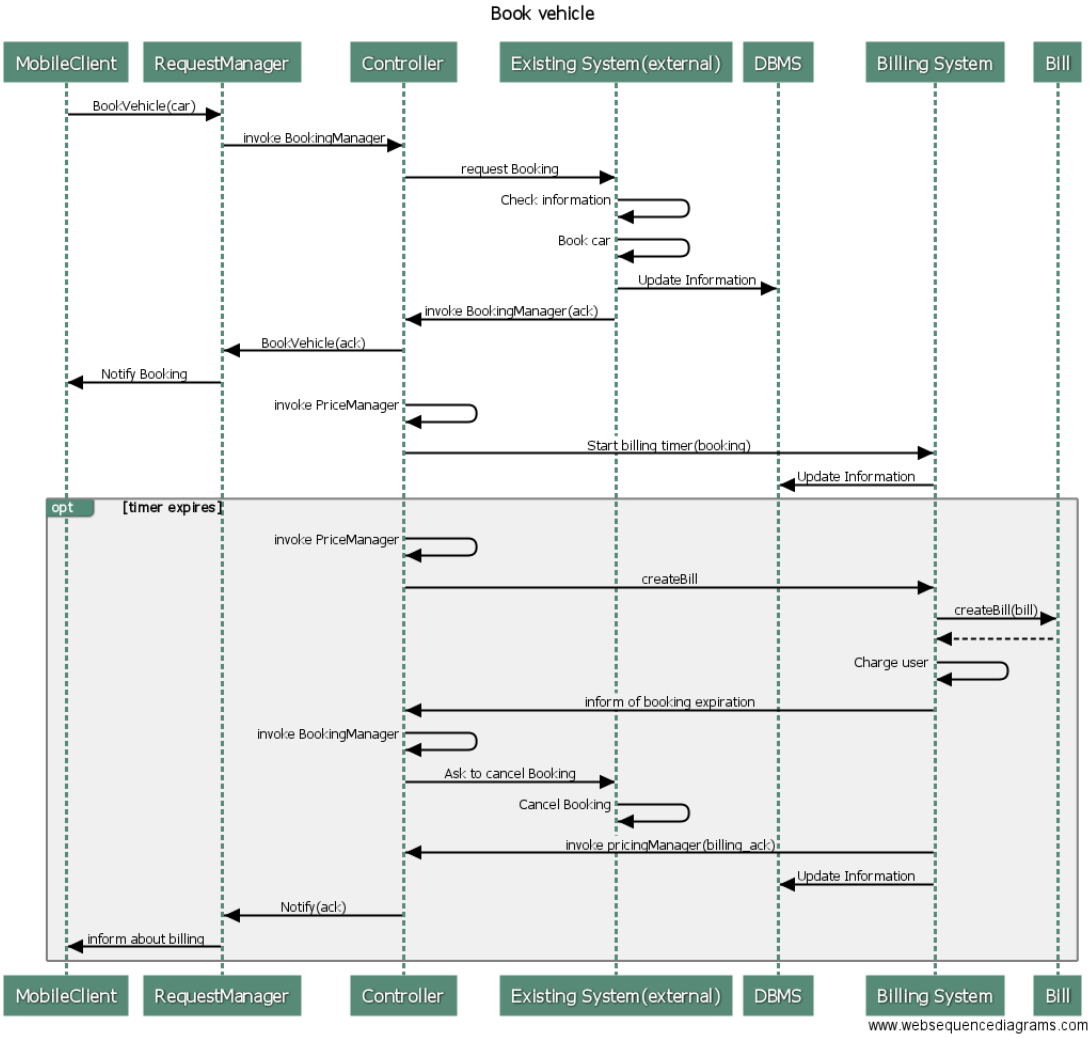

Figure 9: Book Vehicle detailed sequence diagram

2.5.7 Cancel Booking

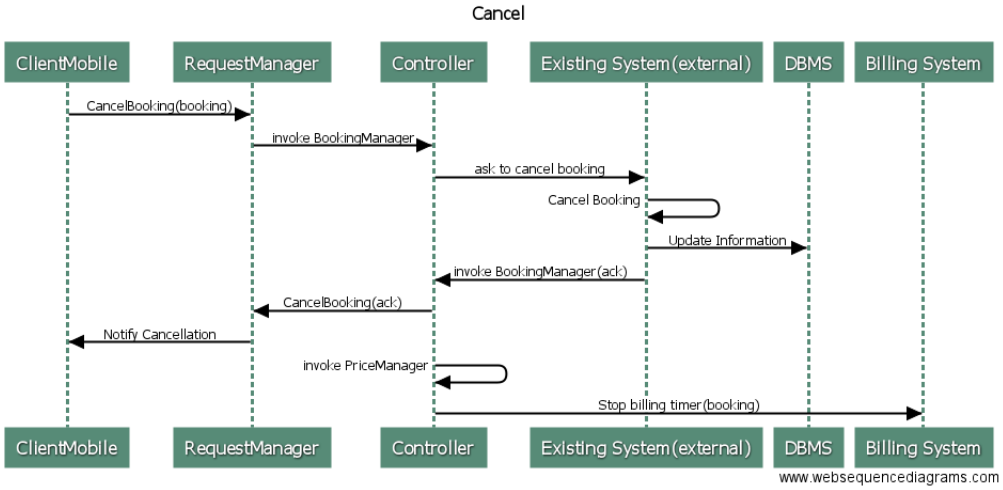

Figure 10: Cancel Booking detailed sequence diagram

2.5.8 Open the Vehicle

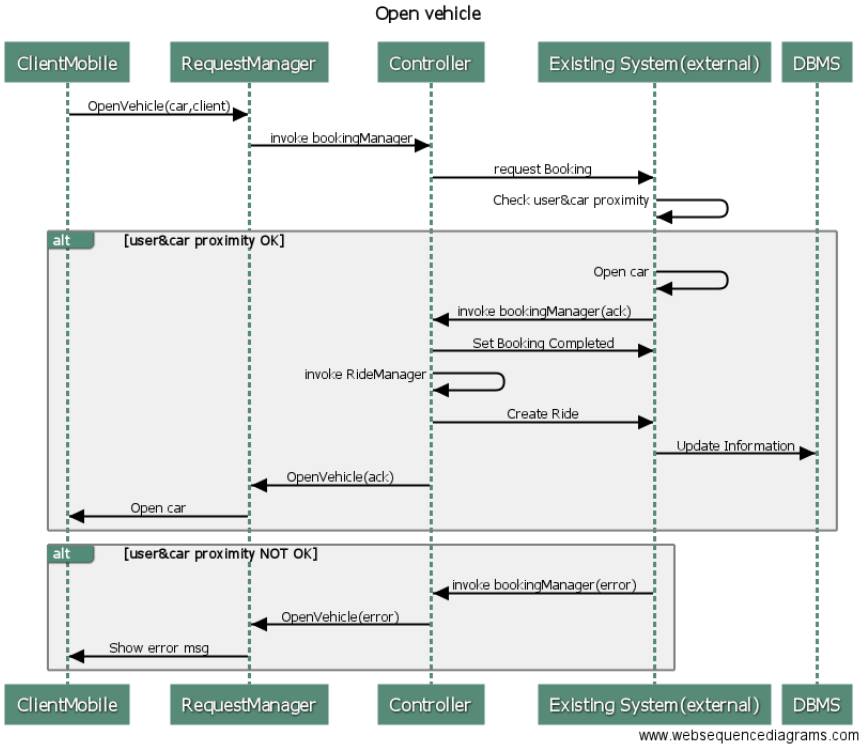

Figure 11: Open the Vehicle detailed sequence diagram

2.5.9 Conclude the Rent

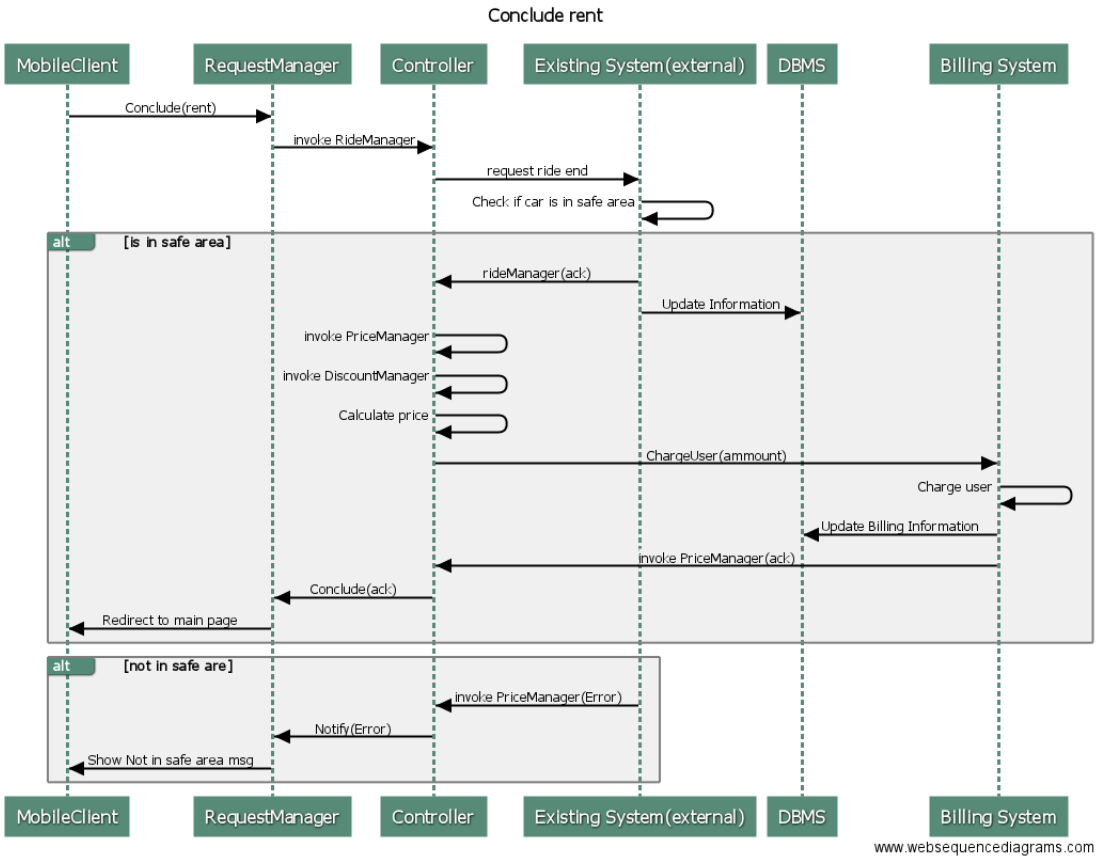

Figure 12: Conclude the Rent detailed sequence diagram

2.6 Component's Interfaces

Here we list for each of the components placed at every single layer the interface used to communicate with the others components they are connected with.

2.6.1 Model Components

I **DBMS :** The DBMS which handles the private data of the application communicates with the controller using the JDBC interface which provides the controller methods to query and update the data.

II **Existing System :** The existing system is an external component that interfaces (and is interafaced by) the controller through specific APIs that are managed using a JEE dependency injection.

2.6.2 Controller Components

I **Application Server :** The application server communicates with the model using the above mentioned methods. The methods of the controller are invocated directly by method call by the request manager component. The application exploits the APIs provided by the billing service in order to interface with that service. In addition it interfaces the institutional service used for driving license validation using the HTTPS protocol.

II **Request Manager :** The request manager is basically an interface between the controller and the customer application. It is the counterpart on the server side of all the actions that can be done by the user. The request manager dispatches the request to the right component or components of the controller. It's interface with the controller is just a simple direct method call. This component communicates with the remote view using a XML format for data that travel using the HTTPS protocol over the standard internet infrastracture.

2.6.3 View Components

I **Customer Application :** The application only communicates with the server sending XML wrapped data, through the request manager, to which is connected by the standard internet infrastracture.

2.7 Deployment of the software architecture

The software components described above will be deployed on a three tier architecture. In order to have a cost efficient management of the web servers that host the controller components (Controller and Request Manager) we choose to place them in the cloud. The private database will be hosted in a dedicated unit controlled directly. The component above named "Mobile-CLient" will be deployed in the mobile phone of the user, through the app store. Below we illustrate this selected deployment scheme with a simple diagram.

2.7.1 Deployment View

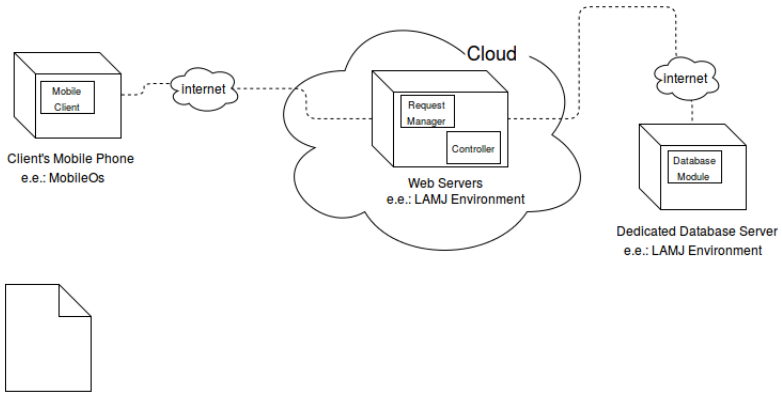

Figure 13: Deployment View

2.8 Selected Architectural Styles and Patterns

The system development is based on the use of an MVC pattern on a three-tier architecture, employing a cloud service to host the web server in order to make it able to scale with the demand of the service. In addition a proxy pattern is adapted having the Request Manager beign a proxy of the controller for the client, and an adapter pattern in the design of the class wrapping the APIs to access the existing system. In the following sub-sections we briefly explain the meaning of each of this design choices.

2.8.1 Three Tier Architecure

The choice of using a three-tier architecture is mainly based on the fact that most of the data are stored separately in another system (the existing system). That's why it seemed obvious to divide also the private data of the system from the control logic. The splitting of presentation and control layer was made mandatory by the fact that the access to the service (the presentation) is by design of the service, on the client's mobile phone which becomes the third tier of the architecture. Another reason of using a three tier architecture is that since the service is open to a wide public, it's better to avoid users to directly access the data of the application. In addition in order to minimize the usage of internet connection the client is meant to be thin and the computation will be done by the application server so that the only thing that must travel on the net is input and output of each operation required by the user. The replication of a chaced version of recent data on the device make sure that even in case of short connection problems it is possible for the user to partially use the application.

2.8.2 Cloud Hosting Platform

The application must be developed in order to be executed parallely on a remote web server hosted by a Cloud service provider. The cloud solution has been indicated as the most suitable one because of the possibility to add servers on demand and to resize the dimension of the server basing on the actual traffic on the system, reducing thus at the minimum the risk of overflow or server failures. In addition the cloud solution offers the hosting service at a cheap price that can reduce maintenance and upgrading costs.

2.8.3 Model View Controller Pattern

The model view controller design pattern is used in this case both to adapt to the standard given by that paradigm and because it's the most fitting design choice given the three-tier architecture and the structure of the application itself where view is physically separated by the server and part of the data is stored in another system. An additional reason for which this pattern has been followed is that a flexible structure was needed in order to strictly separate the view. As a matter of fact the view that interfaces the application must run on different operating systems and thus is absolutely necessary to be able to design different views without the need of reimplement the control logic.

2.8.4 Proxy Pattern

The proxy pattern is used to simplify the interface that the controller exposes to the view. The proxy then maps the request received to actual calls on the controller's components involved in the action.

2.8.5 Adapter Pattern

This pattern is used in order to make compatible the interfaces used by the controller to acess data with the APIs exposed by the Existing System as a mean to access the data contained in it.

3 Algorithm Design

Since there is nothing particulary difficult in terms of Algorithm design, freedom is left to the developing team for the implementation choices of every component with the only limitation of being consistent to what is defined here and in the Requirement Analysis and Specification Document.

4 User Experience

This section is dedicated to the specification of the user experience. We will focus on presenting a User Experience diagram (UX Uml diagram) and some mockups of the desired interface of the application.

4.1 Mockups

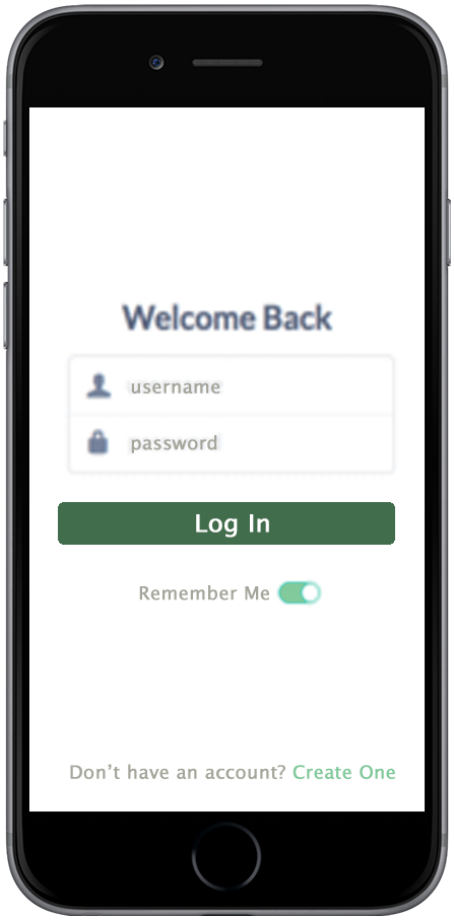

Figure 14: Login screen mockup

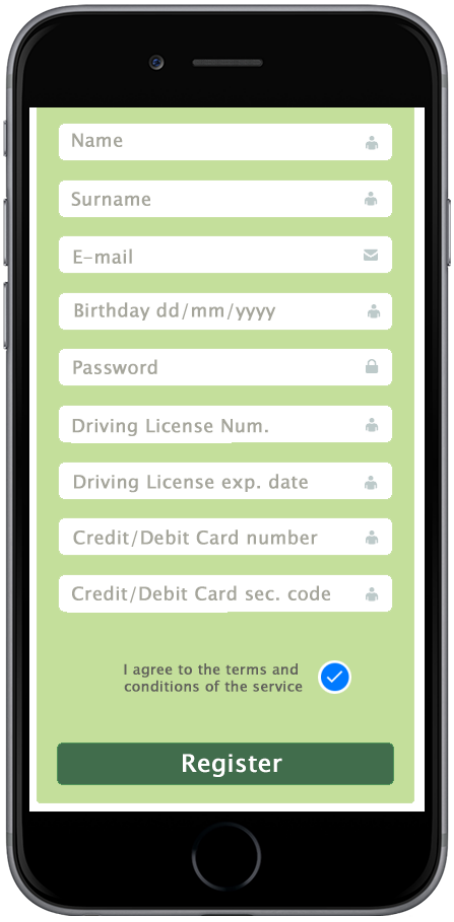

Figure 15: Sign up screen mockup

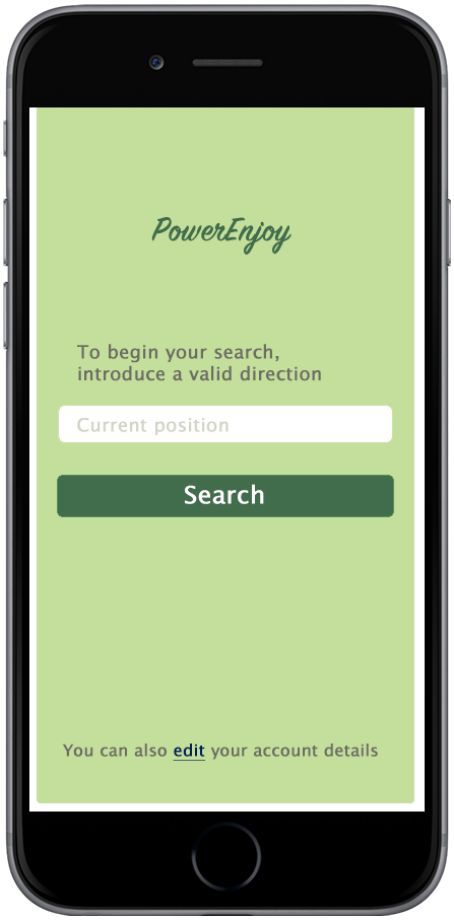

Figure 16: Home screen mockup

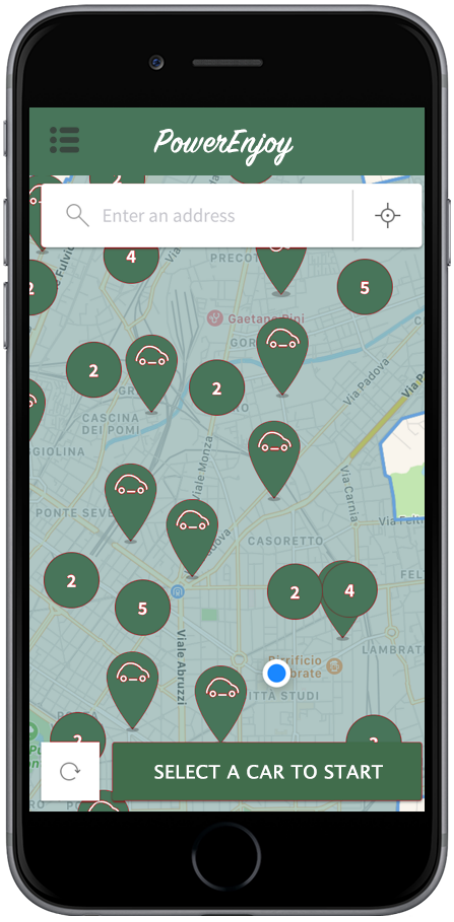

Figure 17: Results screen mockup

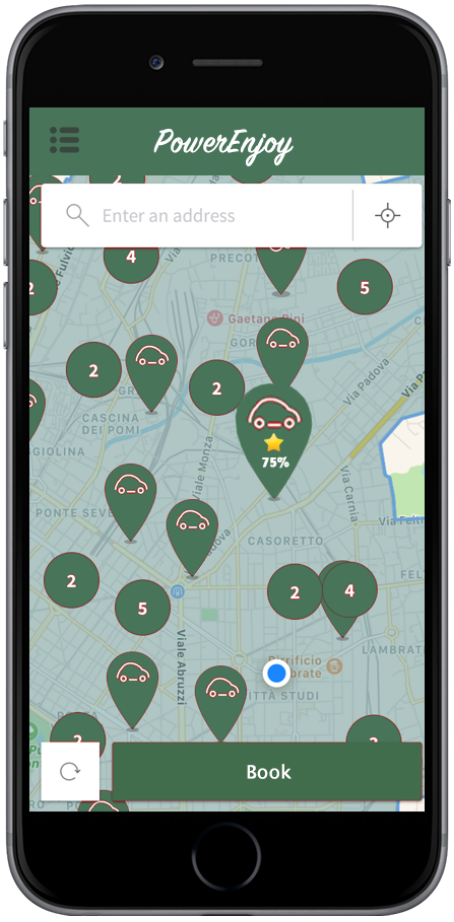

Figure 18: Inspect screen mockup

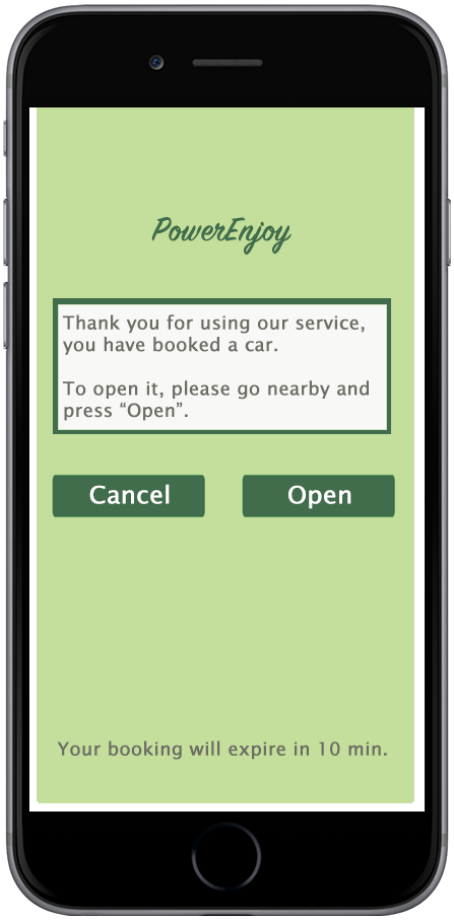

Figure 19: Booking screen mockup

Figure 20: RideActive screen mockup

4.2 User eXperience Diagram

The aim of the user experience diagram is to represent in a formal way the different screens that the user can navigate in the application and to highlight the possible ways to pass from one page to the other. In order to make the notation less heavy in every screen has been omitted the navigateTo() method which is just the default method that guarantees the reachability of the page. In our representation the reachability of every screen is defined by the incoming connections from other screens.

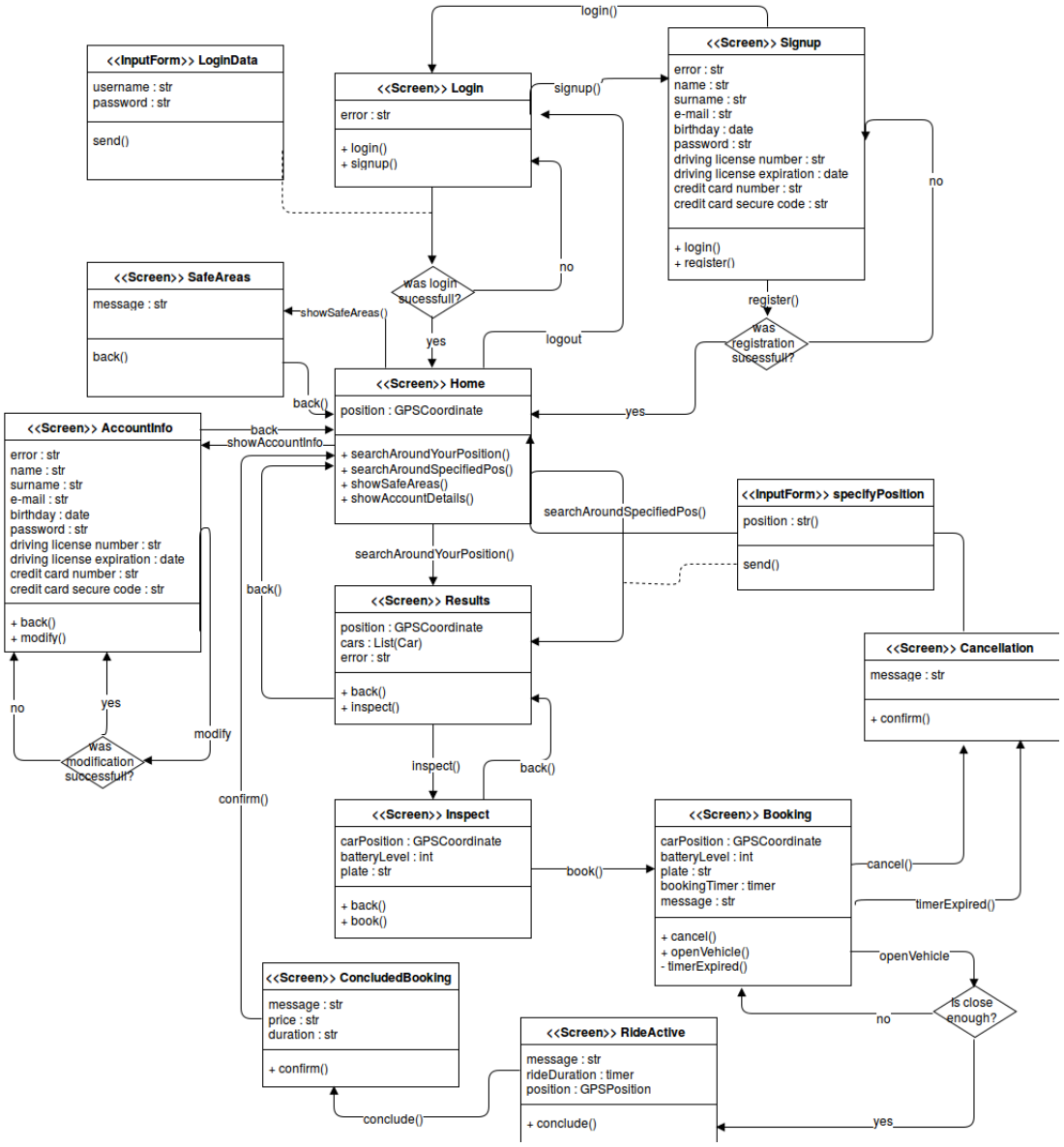

Figure 21: User eXperience Diagram

5 Requirement's Traceability

In order to make sure that every requirement is covered by one of the components of the system, and to clarify which function is assigned to which part of the software we attach the requirement's traceability matrix.

Code	Requirement	Component Name
R0	Recognise if an e-mail address has already been used.	RegistrationManager
R1	Store the data of the new subscriber.	RegistrationManager ResourceManager
R2	Validate the data relating the driving license.	RegistrationManager
R3	Provide access to the system for the new subscriber.	RegistrationManager ResourceManager
R4	Check the correctness of the password+username couple.	LoginManager
R5	Receive an address as an input string from the user.	MobileClient
R6	Select all the cars available within 3 km from the address.	ResourceManager
R7	Display the map of the zone with the available cars on it.	MobileClient
R8	Get the address of the user from the GPS of the mobile device.	MobileClient
R9	Selecting a vehicle from the map.	MobileClient
R10	Set the car as booked.	ResourceManager BookingManager
R11	Start the timer associated with the booking.	BookingManager
R12	In case of expiration intrust the billing service to charge the client with a fee of 1$	BookingManager PricingManager
R13	Checks the validity of the Driving License of the user	RegistrationManager ResourceManager
R14	Check if the client is close to the booked vehicle.	ResourceManager
R15	Check if the booking is still valid.	BookingManager
R16	Set the car as in rent.	BookingManager ResourceManager RideManager
R17	End the timer of the booking.	BookingManager

Code	Requirement	Component Name
R18	Delete the record about the reservation.	BookingManager
R19	Open the correct vehicle.	RideManager ResourceManager
R20	Create a record of the rent initializing the time at 0.	RideManager
R21	Display the credentials of the user.	MobileClient
R22	Allow the modification of any of the fields.	MobileClient
R23	Make the modifications permanent.	ResourceManager
R24	Check when the rent is over.	RideManager
R25	Assign to the billing service the task of charging the client.	RideManager Pricing and Discount Manager
R26	Notify the client for the conclusion of the ride.	MobileClient
R27	The billing system confirms the correct completion of the pay-ment.	PricingManager
R28	Display the details of the vehicle.	MobileClient ResourceManager
R29	Displaying a map of the city with highlighted the safe areas.	MobileClient ResourceManager
R30	Displaying the address for each of the safe areas.	MobileClient ResourceManager
R31	Display existing booking if present.	MobileClient
R32	Cancel the booking.	MobileClient BookingManager
R33	Apply discounts of 10% to the rides in which the user took atleast 2 passengers with him to incentivate car sharing.	DiscountManager PricingManager
R34	Apply discounts of 20% to the last ride if the car is left with at least 50% of the battery level	DiscountManager PricingManager
R35	Apply discounts of 30% to the last ride if the client leaves the car in with at least 50% of the battery level	DiscountManager PricingManager
R36	Apply an extra charge of 30% of the cost of the last ride to those users that leave the car at at least 3 Km from the nearest powergrid.	DiscountManager PricingManager
R37	Apply an extra charge of 30% of the cost of the last ride to those users that leave the car with at least 80% of the battery empty.	DiscountManager PricingManager