DD
Design Document

Luca Marzi
Valeria Mazzola
Federico Nigro

January 15, 2017

Contents

1 Introduction

1.1 Hours dedicated to the project

- Luca Marzi: 20 h
- Valeria Mazzola: 20 h
- Federico Nigro: 20 h

1.2 Purpose

The purpose of this document is to define the design of the PowerEnJoy application up for developers. Next paragraphs will be focused on an expansion of the different technical components integrated in the application and their functionalities, starting from what the RASD has introduced. The analysis will comprise the following elements:

- high level architecture
- design patterns
- main components and interfaces and their interactions
- runtime behaviours

1.3 Scope

The System (described in the RASD as the entity containing the main aspects of the application) is now divided in five complementary parts. The first of them consists of the Mobile App, which interacts with the User; the second is the Database Server, containing a protected copy of User's and Car's datas, the third is the Application Server, which is a Server Java; the fourth part is the Administration App, from which the administrators control the entire service process and the fifth part is the Car's System (On Board System), whose role is to guaranteed the communication between the Application Server and the User. The Server Java, has the important role to correctly and syncronously manage the communications among the different entities and actors, participating to the different horizontal and vertical instances of the service. We already show the main actors in the RASD and now we can say that the Client is everyone who aims to drive one of the vehicles offered by the Service, whilst different connections and interfaces exist between the Application Server and the Car's System (which surely has a proper application inside) and between the System and the Payment System. The aspect of the registration by the Client is important, because the System has finally all the User's data when it is needed and this is a simplifiation on the various requests among actors too. Hence we can assume that errors due to incorrect data inserted by the User only come up during the registration phase. Other facilities are considered regarding the payment (characteristic of the Payment System) and the primitives of interaction with the database. In addiction to the main fucntionalities, the System can offer some extra functionalities: a FAQ service, a communication with an operator at necessity, a forum service and a news service are also guaranteed for the User. We finally conclude this sub paragraph saying that features of applications will be discussed in this document, keeping track of what the RASD already drew up about the System, with some more details and modifications when needed. We can also state that the System itself will be built to be reliable and efficient, according to its purpose and integration in the World and in what we addressed as Environment in the RASD.

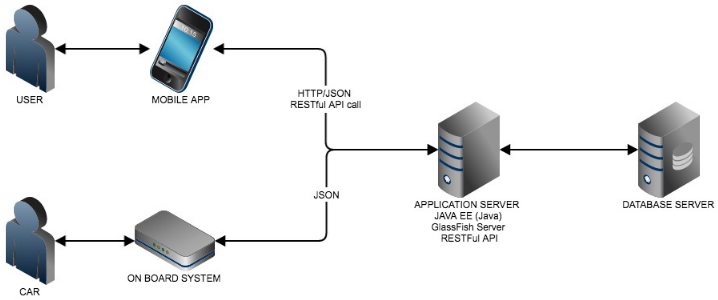
1.4 Definitions, acronyms and abbreviations

- **Mobile App:** the particular component of the System which aims to interacts with the User friendly and giving all its needed to correctly offer the service by the Application Server.
- **Application Server:** the particular component of the System which aims to efficiently offer the service to the User, giving all the required informations and responding to User inputs on the Mobile App.
- **Server Java:** it is the way the Application Server is built. Designing the Server through this kind of technology means using a Java environment, services and facilities.
- **Database Server:** it is where all the informations relative to the service's utilization are stored, permanently or temporarily. It has to be designed to have the important properties of databases: Atomicity and Consistency of operations and datas, for all the time they are in the database; Isolation, because the operations and datas of different Users and different instances must be detached from each others and Durability, because the database must ensure that datas won't be lost during their permanence in static memory.
- **Horizontal instances:** in the lecture above, it refers to multiple connections opened with different Users in parallel way. Users can communicate with the Server at the same time and without having any overlap during the whole service.
- **Vertical instances:** in the lecture above, it refers to the chains of actions taken to reach the scope of all the service relatives to a single User. This term embraces the actions of the User and of the Server too.
- **Clients:** it is the User and her/his Mobile App which generally interact with the Server.
- **Service:** it is generally what the System longs to offer to Users and, in particular, the topic, currently covered by the RASD and the DD too.

2 Architectural Design

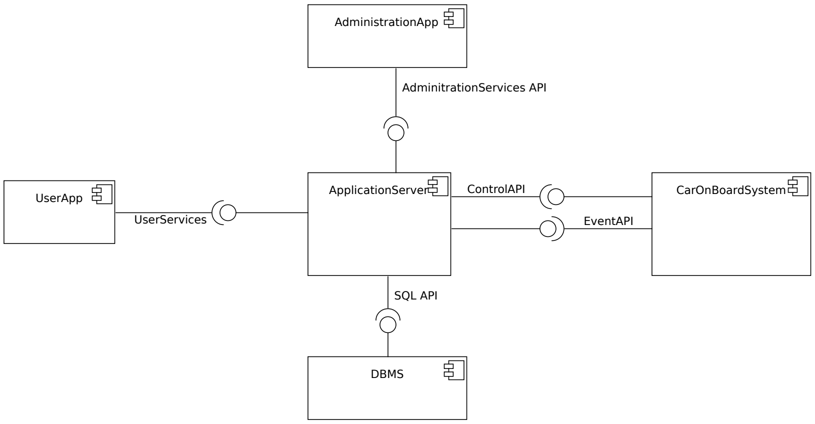
2.1 Overview

As mentioned before, the System is divided in different, indipendent, software and hardware components. The principal component is the Server Java. Every other component interacts with the Server using different protocols, designed for the particular purpose. The communication with the Mobile App is due to the Http/Json protocols and the meaning of the RESTful API architecture. This architecture shows some foundamental principles up: between Client and Server there are some web resources, manipulates for correctly allowing the interaction between the two entities; hyperlinks are useful for the transactions and some interfaces are displayed continuosly to the User. This kind of interfaces have some bonded and well-defined operations; some content, which can be changed on demand and by the Server; a particular protocol for regulating the trasmission. The protocol is a client-server and stateless protocol and establishes also how the cache is used and the tiers in which the System is generally divided. Going on the System division in tiers, we have one layer, which is the client; one another, which represents the On Board System (it contains its indipendent system for the correct utilization of the features inside the Car, but this component is strictly correlated to our System) and the Server, which is responsible of all the controls on inputs from the two entities. The logic of the application, in particular, is all up to the Server. Finally the Server stores and retrieves informations from the Database using MySql.



2.2 High level components and their interactions

A simple high level representation of the system is shown in the following diagram.



The architecture defines five main components that are:

1. ApplicationServer: it's the component that handles the business logic of the system and takes care of all the requests made by the User through the mobile application. The ApplicationServer also interacts with the cars and handles their states.

2. UserApp: it's the component that allows the User to access the services of the System through a graphical interface.

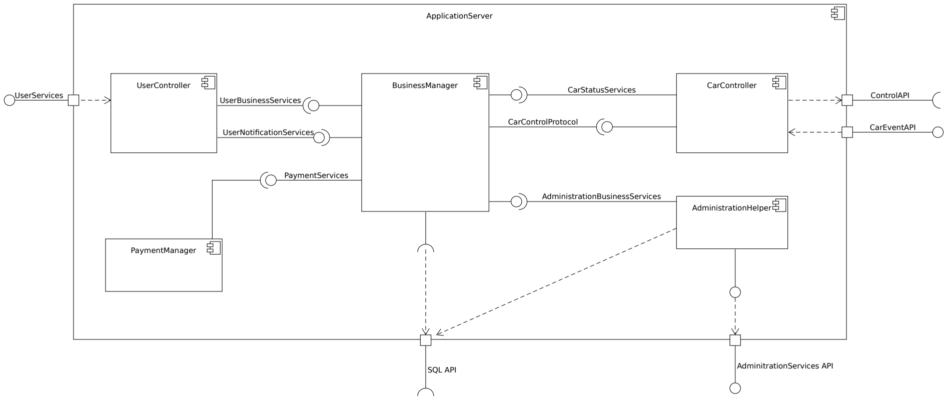
3. AdrministrationApp: it's a special app that is intended to be a desktop appication by which some administration operations can be performed.

4. CarOnBoardSystem: it's the component that it's installed into the physical cars and interacts with their operating system. The CarOnBoardSystem takes care of all the events of interests that happen on the Car and all the commands that the ApplicationServer wants to perform over the Car.

5. DMBS: it's the component that provides a standard SQL API to all the components that requires persistent data storage.

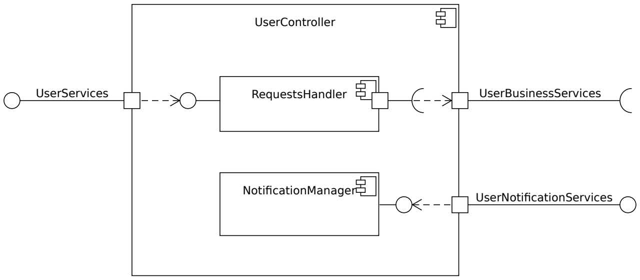
2.3 Components of the application server

This section describes the components of the application server and defines their role in the architecture. The basic structure of the application is made up of a Business Manager, handling application logic, and multiple controllers handling communication with different actors, as shown in the following diagram.



2.3.1 User Controller

The User Controller module handles communication with the user's mobile application.



The User Controller traces requirement [R1.2].

2.3.2 Car Controller

The Car Controller implements the communication protocol between the on-board system of the cars, previously identified as actors, and the core of the application to be developed.

Therefore, this module will provide an interface to the rest of the application allowing the exchange of the following types of messages:

- ordinary telemetry such as GPS position, battery status and other sensor information;
- commands from the application to the car;
- events from the car to the application.

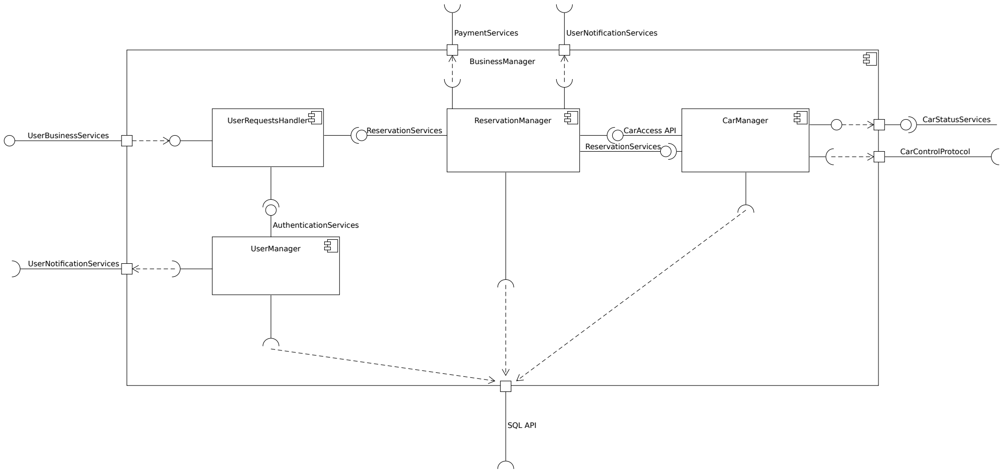
The Car Controller is a thin module and has no knowledge of the application's logic, or the cars' state. It is to be implemented as a thread pool, each thread in charge of keeping the communication active with a single car.

2.3.3 Business Manager

The Business Manager implements the business logic of the application. It is composed of the following sub-components:

- User Request Handler
- User Manager
- Reservation Manager
- Car Manager

They are described more in detail in section 2.4.



The Business Manager is the core of the application. As such, it will access the database to issue all the needed database operations.

2.3.4 Administration Helper

This module provides the API necessary for administration purposes.

2.3.5 Payment Manager

The Payment Manager implements the interface between the application and the external payment system used for monetary transactions.

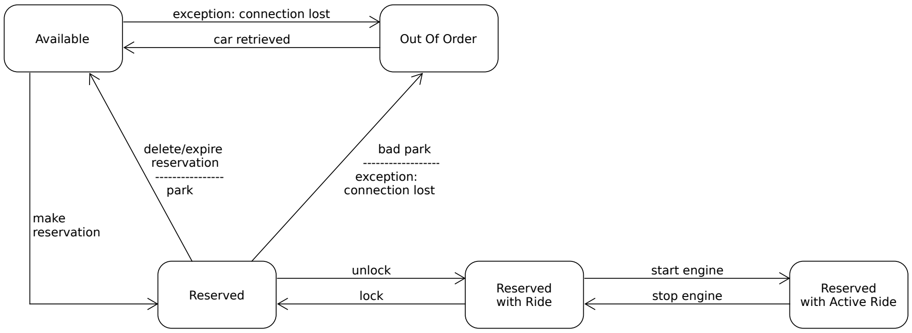
2.4 Sub-components of the Business Manager

2.4.1 Car Manager

The Car Manager encapsulates the handling of car states logic, as well as the car pool itself, meaning it must provide the following functionality:

- car state handling;
- authentication of cars into the system;
- subscription of new cars to the pool and removal of old ones.

The car states are described in the following state diagram:



In order to encapsulate the car states, the Car Manager will interact with the Car Controller by issuing commands and receiving events. The Car Manager must be able to respond to information queries such as:

- get the state of a car;
- get a set of cars by position and state.

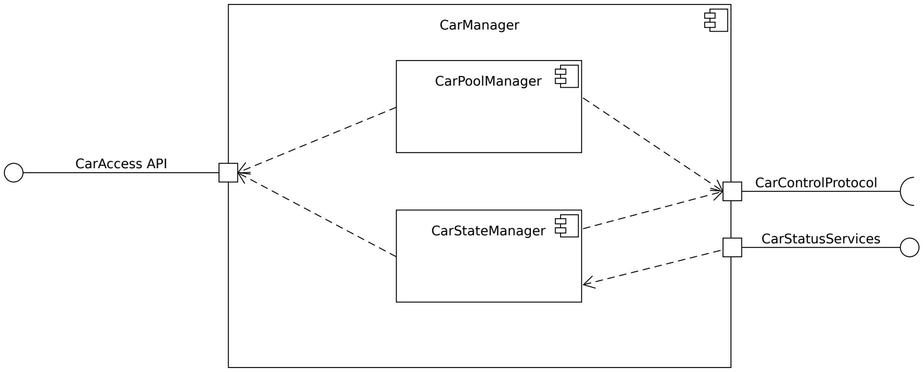
The Car Manager also has knowledge of the parking areas and is in charge of deciding if a car has been parked in an acceptable location. This information is then forwarded to the Reservation Manager in order for it to account for fees or discount when necessary.

The functionalities implemented by this module will trace requirements [R2.2], [R2.3], [R4.4].

The Car Manager will expose an interface to the Reservation Manager with the following types of methods:

- observers of car states and car sets;
- observer of car position;
- modifiers of car states.

The internal structure of the Car Manager is composed of a Car Pool Manager, handling authentications and subscriptions of cars to the pool, and a Car State Manager, handling the car states.



2.4.2 Reservation Manager

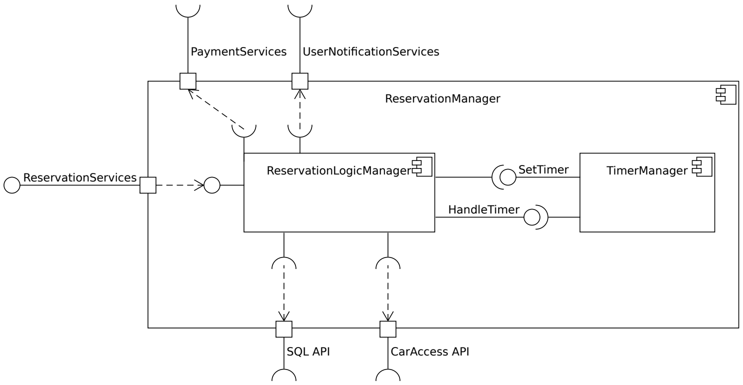
The Reservation Manager is in charge of all reservation operations, including the computation of a reservation's cost.

It will interact with the other modules like so:

- interaction with the Car Manager:
  - issue queries as described in section 2.4.1;
  - use the Car Manager's methods to change the state of a car as a result of operations such as creation, deletion or expiration of a reservation;
  - receive car events such as start/end of rides, activations/deactivations of the car's engine;
  - receive special information, such as a badPark event, needed to account for fees to the user;
- interaction with the User Request Handler:
  - provides services for creating and handling reservations that are visible by the User through the User Request Handler.

It will also interact with the external payment system in order for the transactions to happen.

As timings are a core part of the reservation logic, a dedicated sub-module is designed to handle the timers:



It is responsibility of the Reservation Manager to ensure the correct handling of reservation logic and the fulfilling of related requirements [R2.4], [R3.2], [R4.1], [R4.2], [R4.3], [R4.5].

2.4.3 User Manager

The User Manager provides all the services about User Athentication. Among these services there are:

- registration of a new User;
- authentication of an existing User;
- retrieving/updating of User information.

This module traces requirements [R1.1] and [R1.3].

2.4.4 User Request Handler

The User Request Handler simply receives the requests of services coming from the User and forwards them to the correct business component, therefore tracing requirement [R2.1].

2.5 Component interfaces

This section describes the semantics of interfaces exposed by the modules presented in section 2.3 and section 2.4.

UserServices Functionalities exposed by the application server to the mobile application.

UserBusinessServices Interface exposed by the Business Manager for business logic functionalities regarding the User.

UserNotificationServices Interface exposed by the User Controller to the Business Manager in order to allow notification services on the mobile app as well as email communication to the user.

PaymentServices Interface exposed by the Business Manager in order to allow the Payment Manager, which acts as a controller, to handle all payment functions.

CarStatusServices Interface exposed by the Car Manager, used by the Car Controller to change the car states when necessary.

CarControlProtocol Interface exposed by the Car Controller, used by the Business Manager to forward commands to the car units.

CarControlAPI Control communication protocol of the car that the Car Controller will implement. As this protocol may change between car vendors, the Car Controller will implement all necessary alternatives.

CarEventAPI Communication protocol used by the cars to notify the application of car events. This protocol is also vendor specific.

ReservationServices Interface exposed by the Reservation Manager, used by the User Request Handler to perform reservation related operations.

CarAccessAPI Interface exposed by the Car Manager, used by the Reservation Manager for operations such as changes of car states.

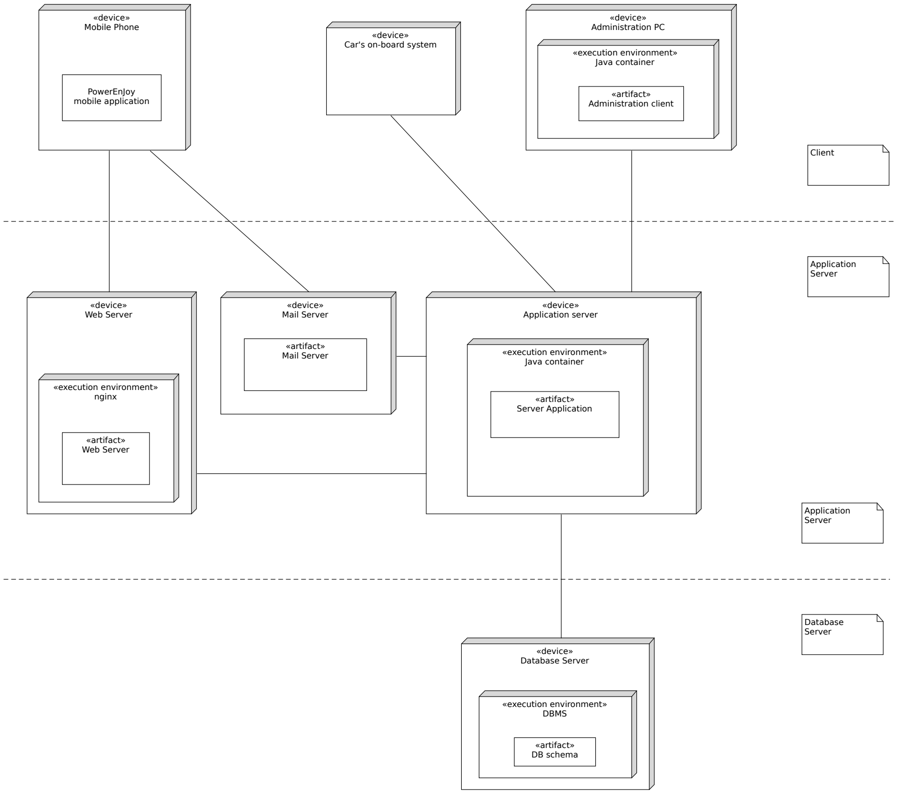
AuthenticationServices Functionalities exposed by the User Manager, used by the User Request Handler to perform authentication tasks as well as the creation and deletion of user accounts.

2.6 Requirement tracebility

All requirements not explicitly traced in section 2.3 will result from the combined action of the previously identified application modules. This is true for all [R5.*] requirements as well as [R6.1] and [R7.1].

2.7 Deployment View

PowerEnJoy will be deployed as a three-tier architecture system as described by the following diagram:



2.8 Runtime View

2.8.1 Registration

In this phase the NonRegisteredUser obtains a valid registration to the service. She/He

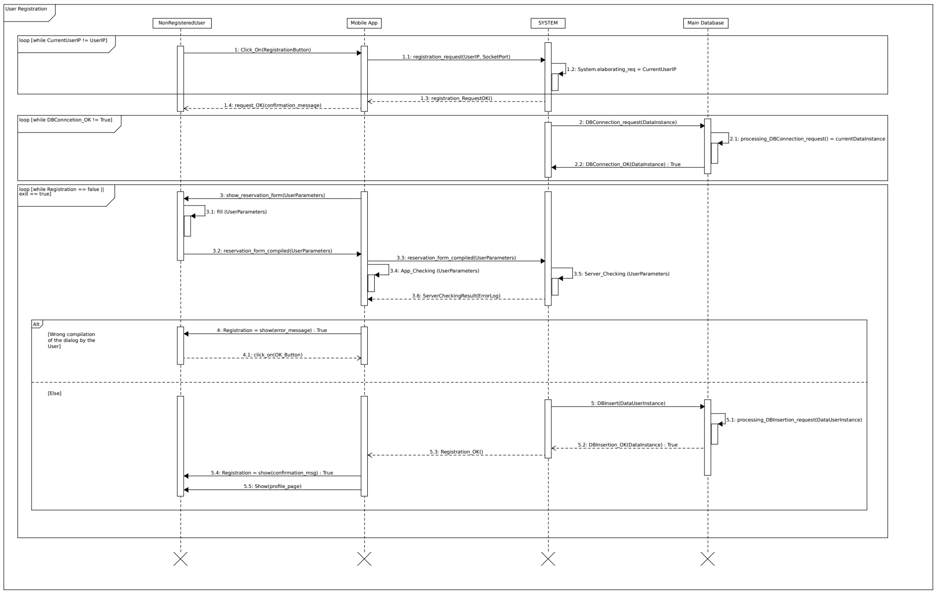
1. expresses the will to make a registration by clicking on the registration Button in the Mobile App. The Mobile App sends a new request to the Server Java which

2. opens a new connection with the DBMS for the future storage of datas from the whole process. At this point the Mobile App

3. displays to the User the form with fields to be filled for the Registration. The User compiles the form and confirms it to the Mobile App. The Mobile App and the Server Java check if the compilation is maden correctly. If it is not the Mobile App

4. alerts the User of the lackness, else

5. datas are finally stored in the DBMS from the Server and the User receives the confirmation of the correct registration.


User Registration
NonRegisteredUser
Mobile App
SYSTEM
Main Database
loop [while CurrentUserIP != UserIP]
1: Click_On(RegistrationButton)
1.1: registration_request(UserIP, SocketPort)
1.2: System.elaborating_req = CurrentUserIP
1.3: registration_RequestOK()
1.4: request_OK(confirmation_message)
loop [while DBConncetion_OK != True]
2: DBConnection_request(DataInstance)
2.1: processing_DBConnection_request() = currentDataInstance
2.2: DBConnection_OK(DataInstance) : True
loop [while Registration == false || exit == true]
3: show_reservation_form(UserParameters)
3.1: fill (UserParameters)
3.2: reservation_form_compiled(UserParameters)
3.3: reservation_form_compiled(UserParameters)
3.4: App_Checking (UserParameters)
3.5: Server_Checking (UserParameters)
3.6: ServerCheckingResult(ErrorLog)
Alt
[Wrong compilation of the dialog by the User]
4: Registration = show(error_message) : True
4.1: click_on(OK_Button)
[Else]
5: DBInsert(DataUserInstance)
5.1: processing_DBInsertion_request(DataUserInstance)
5.2: DBInsertion_OK(DataInstance) : True
5.3: Registration_OK()
5.4: Registration = show(confirmation_msg) : True
5.5: Show(profile_page)

2.8.2 Authentication

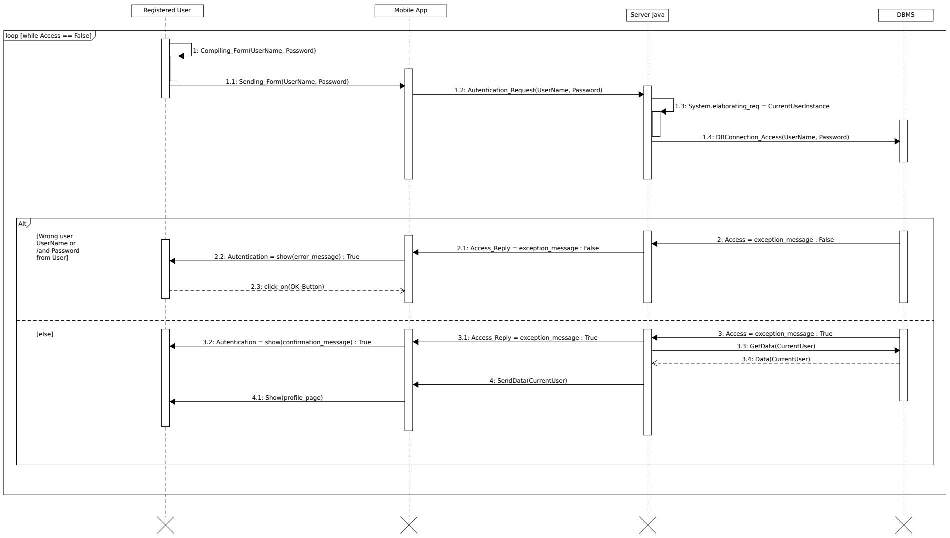
In this phase the User

1. compiles the form with her/his User Name and her/his password, the Mobile App sends the request for accessing to the Server Java, which questions the DBMS. The database makes a research and

2. informs the Server that there isn't any correspondence between the datas it has received and any of the stored instances, or

3. the correspondence exists. In every of these cases the Server warns the Mobile App of the current event and the User is so alerted by the App. If the authentication process ends without errors the Server retrieves correct informations from the DBMS and

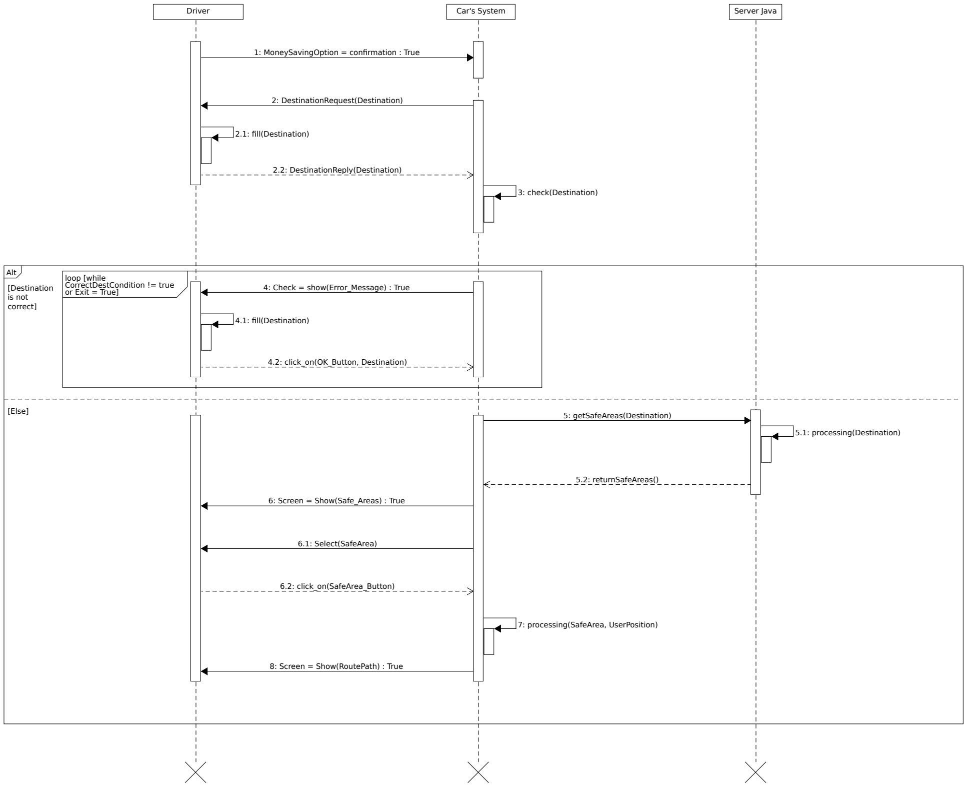
4. as a final result the User can see her/his profile displayed by the Mobile App.


Registered User
Mobile App
Server Java
DBMS
loop [while Access == False]
1: Compiling_Form(UserName, Password)
1.1: Sending_Form(UserName, Password)
1.2: Autentication_Request(UserName, Password)
1.3: System.elaborating_req = CurrentUserInstance
1.4: DBConnection_Access(UserName, Password)
Alt
[Wrong user UserName or /and Password from User]
2: Access = exception_message : False
2.1: Access_Reply = exception_message : False
2.2: Autentication = show(error_message) : True
2.3: click_on(OK_Button)
[else]
3: Access = exception_message : True
3.1: Access_Reply = exception_message : True
3.2: Autentication = show(confirmation_message) : True
3.3: GetData(CurrentUser)
3.4: Data(CurrentUser)
4: SendData(CurrentUser)
4.1: Show(profile_page)

2.8.3 Activating the Money Saving Option

In this phase, the User who gets on the Car,

1. types on the Screen of the Car the will to activates the Money Saving Option. The Server sideCar's System asks the User which is the final destination of her/his journey. The User types the Destination and
2. the Car's System checks the validity of the just inserted coordinates through the GPS service. If the control reports any error,
3. the User has to digit again the Destination (she/he can also come back to the previous context without Activating the Money Saving Option); else
4. the Car's System makes a request to the Server for obtaining the list of the available and nearest Safe Areas given the Position. The Server processes the request and effectively returns to the Car's System the requested list. At this point,
5. the list is showed to the User who selects one of the Areas. It is the Car's System which finally
6. computes the path to the Safe Area and
7. shows it to the User through the Car's Screen.


Driver
Car's System
Server Java
1: MoneySavingOption = confirmation : True
2: DestinationRequest(Destination)
2.1: fill(Destination)
2.2: DestinationReply(Destination)
3: check(Destination)
Alt
[Destination is not correct]
loop [while CorrectDestCondition != true or Exit = True]
4: Check = show(Error_Message) : True
4.1: fill(Destination)
4.2: click_on(OK_Button, Destination)
[Else]
5: getSafeAreas(Destination)
5.1: processing(Destination)
5.2: returnSafeAreas()
6: Screen = Show(Safe_Areas) : True
6.1: Select(SafeArea)
6.2: click_on(SafeArea_Button)
7: processing(SafeArea, UserPosition)
8: Screen = Show(RoutePath) : True

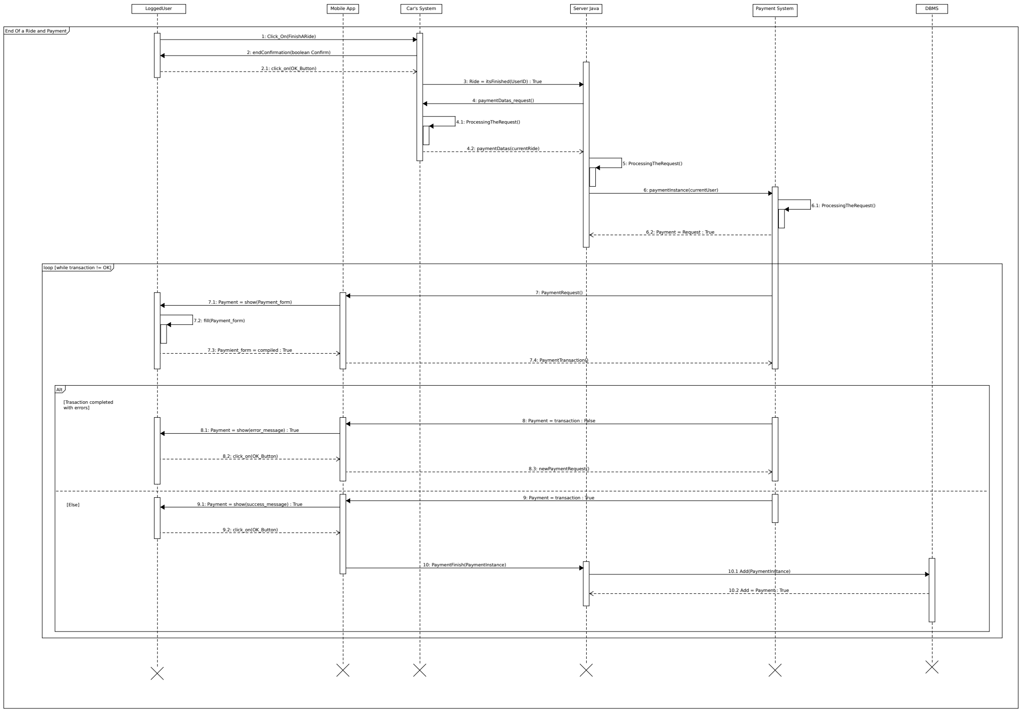
2.8.4 Finish a Ride

The User tells to the Car's System that

1. the ride ended andServer side
2. the Car's System asks for a confirmation. The Car's System
3. forwards the information to the Server which
4. wants to know from the Car's System all the details of the ride, for proceding to the payment phase. The system on board effevtively provides the required datas. The Server
5. processes the incoming informations and after
6. sends a request of payment to the Payment System. This last component confirms the payment to the Server and
7. explicitly asks for the payment to the User through the Mobile App. The User fills the form with her/his sensible datas about credit card and sends it again to the Payment System. The whole money transaction can

8. complete with errors and the User is alerted of this by the Mobile App; or

9. can complete correctly. In this last phase the User and the Server are alerted and the Server fills some fields in the DBMS with the history of current User's payment.


LoggedUser
Mobile App
Car's System
Server Java
Payment System
DBMS
End Of a Ride and Payment
1: Click_On(FinishARide)
2: endConfirmation(boolean Confirm)
2.1: click_on(OK_Button)
3: Ride = itsFinished(UserID) : True
4: paymentDatas_request()
4.1: ProcessingTheRequest()
4.2: paymentDatas(currentRide)
5: ProcessingTheRequest()
6: paymentInstance(currentUser)
6.1: ProcessingTheRequest()
6.2: Payment = Request : True
loop [while transaction != OK]
7: PaymentRequest()
7.1: Payment = show(Payment_form)
7.2: fill(Payment_form)
7.3: Paymient_form = compiled : True
7.4: PaymentTransaction()
Alt
[Trasaction completed with errors]
8: Payment = transaction : False
8.1: Payment = show(error_message) : True
8.2: click_on(OK_Button)
8.3: newPaymentRequest()
[Else]
9: Payment = transaction : True
9.1: Payment = show(success_message) : True
9.2: click_on(OK_Button)
10: PaymentFinish(PaymentInstance)
10.1 Add(PaymentInstance)
10.2 Add = Payment : True

2.8.5 Search and Book a nearby Car

In this phase the Logged User makes the request

1. to see the available Cars on the Screen of her/his device. Hence the Mobile App asks to the Server for the list of all the available Cars and, once obtained it, shows it to the User. At this point the User

2. selects one of the Car on the Screen. The App forward the selection to the Server which controls if the Car is still available. If the Car is no available anymore,

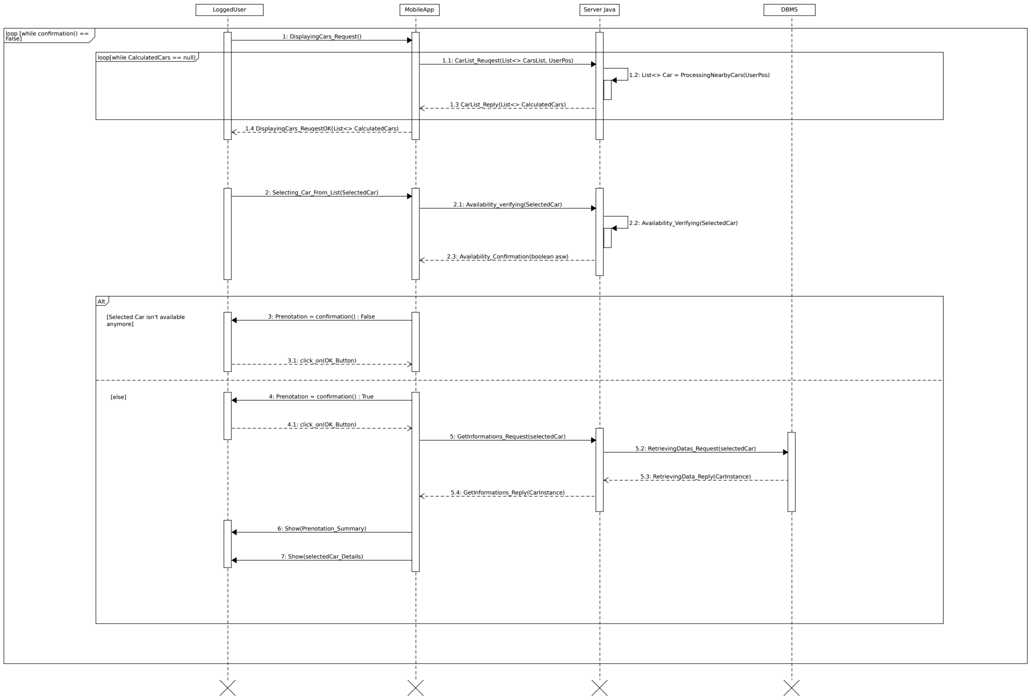
3. the User is informed by the App and there is the return to the previous context. Else

4. the prenotation is confirmed, the Mobile App

5. asks to the Server to retrieves the complete informations of the Car,

6. the details of the reservation are showed to the User and

7. she/he can see the details of the Car too.

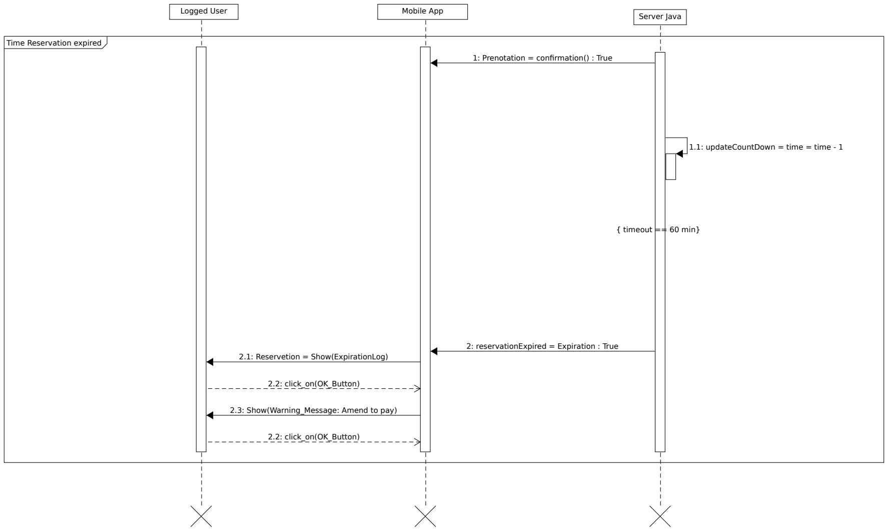

LoggedUser
MobileApp
Server Java
DBMS
loop [while confirmation() == False]
1: DisplayingCars_Request()
loop[while CalculatedCars == null]
1.1: CarList_Reuqest(List<> CarsList, UserPos)
1.2: List<> Car = ProcessingNearbyCars(UserPos)
1.3 CarList_Reply(List<> CalculatedCars)
1.4 DisplayingCars_ReugestOK(List<> CalculatedCars)
2: Selecting_Car_From_List(SelectedCar)
2.1: Availability_verifying(SelectedCar)
2.2: Availability_Verifying(SelectedCar)
2.3: Availability_Confirmation(boolean asw)
Alt
[Selected Car isn't available anymore]
3: Prenotation = confirmation() : False
3.1: click_on(OK_Button)
[else]
4: Prenotation = confirmation() : True
4.1: click_on(OK_Button)
5: GetInformations_Request(selectedCar)
5.2: RetrievingDatas_Request(selectedCar)
5.3: RetrievingData_Reply(CarInstance)
5.4: GetInformations_Reply(CarInstance)
6: Show(Prenotation_Summary)
7: Show(selectedCar_Details)

2.8.6 Time Expiration

Once

1. a prenotation is confirmed, a countdown associated to the current ride instance starts. Reached a given timeout,

2. the Server informs the Mobile App of the expiration and the User is warned about the event and the amend to pay. She/He will pay it afterwards through a redirection to the Payment System.

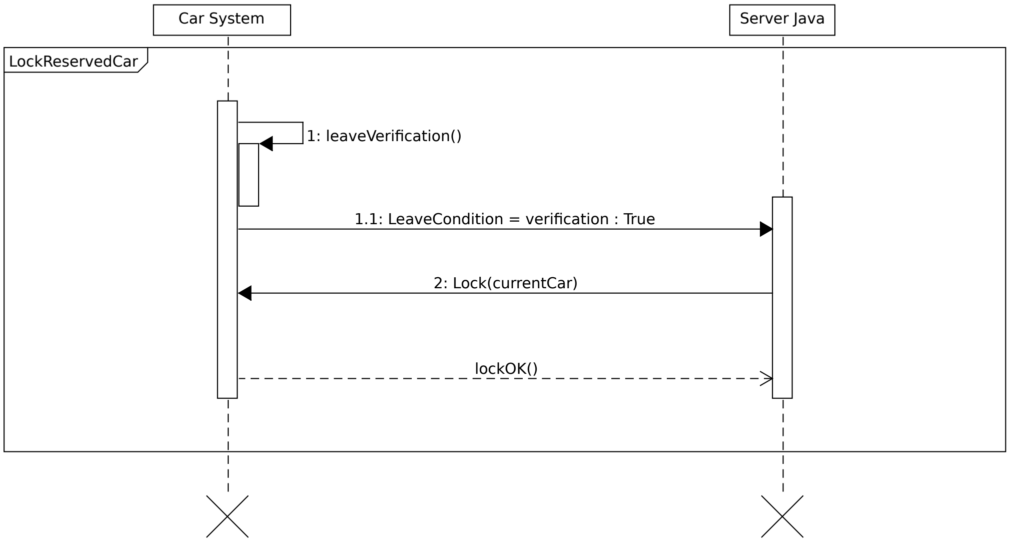

Logged User
Mobile App
Server Java
Time Reservation expired
1: Prenotation = confirmation() : True
1.1: updateCountDown = time = time - 1
{ timeout == 60 min}
2: reservationExpired = Expiration : True
2.1: Reservetion = Show(ExpirationLog)
2.2: click_on(OK_Button)
2.3: Show(Warning_Message: Amend to pay)
2.2: click_on(OK_Button)

2.8.7 Locking a Car

The Car's System

1. constantly verify a leave condition which consists to understand if all the passengers left the Car after a ride through the Car's Sensors. After the Server has received the leave condition,

2. it Locks the Car which confirms the event through an ack.



2.8.8 Unlocking a Car

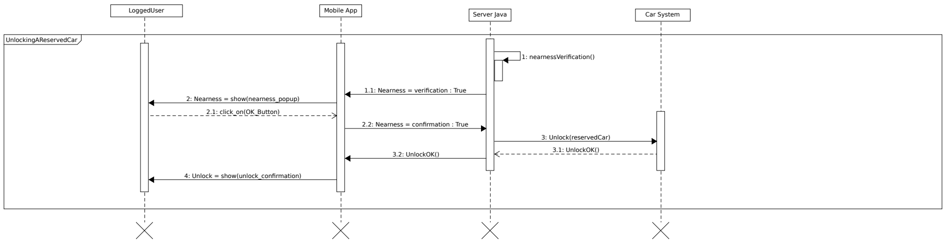
The Server

1. constantly monitors the User position and it informs the Mobile App of the nearness condition when he/she exits. The Mobile App

2. shows the User a button or a popup, giving her/him the information of nearness. The User confirms the will to unlock the near Car and

3. the Server sends the command to the Car System. It finally warns the Server of the Unlock and

4. the User is informed of the event.

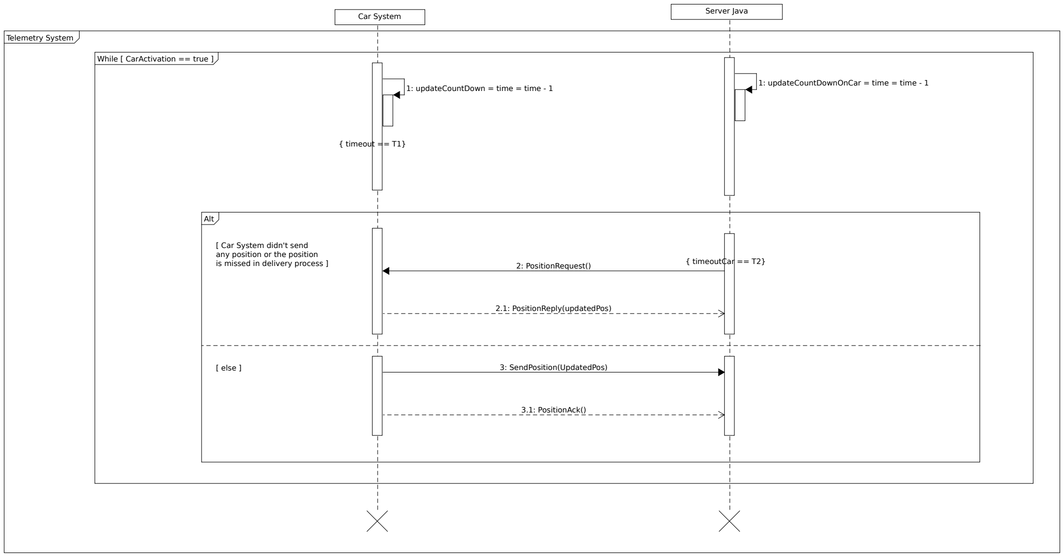

LoggedUser
Mobile App
Server Java
Car System
UnlockingAReservedCar
1: nearnessVerification()
1.1: Nearness = verification : True
2: Nearness = show(nearness_popup)
2.1: click_on(OK_Button)
2.2: Nearness = confirmation : True
3: Unlock(reservedCar)
3.1: UnlockOK()
3.2: UnlockOK()
4: Unlock = show(unlock_confirmation)

2.8.9 Telemetry

The Car constantly continues to communicate with the System during its lifetime. In this particular case there is an updating of a first countdown T1 by the Car's System. If the Car's System doesn't send anything after the countdown is reached,

1. a position request is made by the Server after the achievement of a second countdown in it. Else

2. the Car's System correctly sends its position to the Server Java, which replies with an Ack.


Car System
Server Java
Telemetry System
While [CarActivation == true]
1: updateCountDown = time = time - 1
1: updateCountDownOnCar = time = time - 1
{ timeout == T1}
Alt
[Car System didn't send
any position or the position
is missed in delivery process]
{ timeoutCar == T2}
2: PositionRequest()
2.1: PositionReply(updatedPos)
[else]
3: SendPosition(UpdatedPos)
3.1: PositionAck()

2.9 Selected Architectural Styles and Patterns

We have a selected a modified MVC-like architecture, with the Business Manager acting as a Model and multiple Controllers for distinct purposes. There will also be multiple distinct View modules, in the mobile application and in the car's on-board system, as well as a dedicated view for administration purposes.

Our design also depicts a three-tier architecture, comprised of a client layer, an application server layer and a database server layer.

2.10 Other decisions

2.10.1 Administration View

We have decided to delay the implementation of the administration view for this phase, while the development of the administration API and a view stub will be prioritary, in order to provide the stakeholders with a way to access the system as soon as possible.

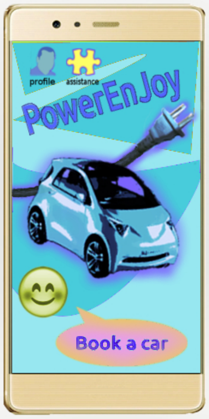
2.10.2 Encryption of car-to-server communication

For security reasons, we decided to design an authentication feature for cars managed by the system. The identity of the cars will be checked by the server with a TLS-like cryptographic handshake and all communication between car and server, including ordinary telemetry, will be encrypted. Validation and deprecation of the keys will be managed through the administration API and the keys physically inserted into the car's system by human operators.

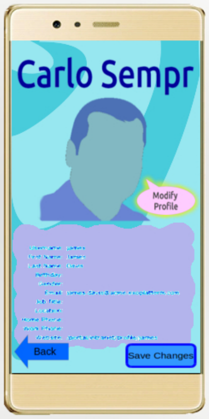
3 User Interface Design

This first part of the paragraph is strictly correlated to the “Nonfunctional Requirements” one, presented in the RASD document. The images below, in particular, show some examples of how the User friendly will interact with the Mobile App illustrated in the document. Specifics and descriptions of this interfaces can be find in the RASD document: the image **(1)** illustrates the Main Menu of the application; the image **(2)** refers to the User Profile, accessible from the Main Menu and in which the User can also decide to modify some of her/his datas; the image **(3)** shows how the User can reserve a Car seeing it displayed in green color on the Map; **(4)** once selected and after all the controls by the Server, the displayed Car becomes red and it appears also the Button to be pressed by the User once the nearness condition will be verified.

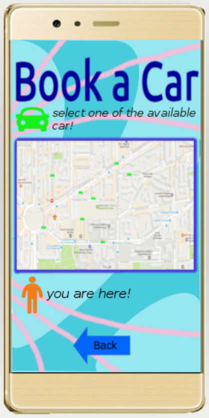
1)



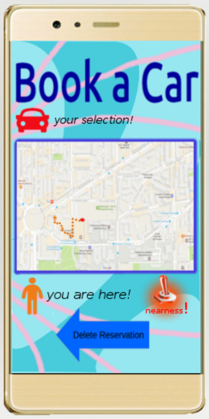
2)



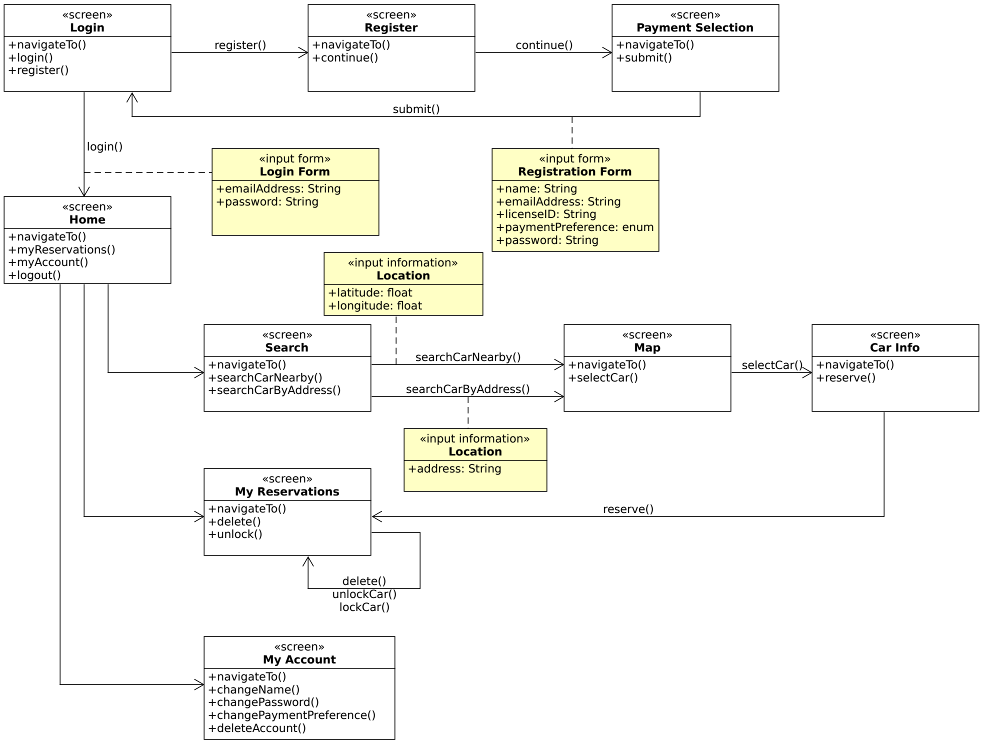
3)



4)



The user interface of the PowerEnJoy mobile app is described by the following UX diagram.


«screen»
Login
+navigateTo()
+login()
+register()
register()
«screen»
Register
+navigateTo()
+continue()
continue()
«screen»
Payment Selection
+navigateTo()
+submit()
submit()
login()
«input form»
Login Form
+emailAddress: String
+password: String
«input form»
Registration Form
+name: String
+emailAddress: String
+licenseID: String
+paymentPreference: enum
+password: String
«screen»
Home
+navigateTo()
+myReservations()
+myAccount()
+logout()
«input information»
Location
+latitude: float
+longitude: float
«screen»
Search
+navigateTo()
+searchCarNearby()
+searchCarByAddress()
searchCarNearby()
searchCarByAddress()
«screen»
Map
+navigateTo()
+selectCar()
selectCar()
«screen»
Car Info
+navigateTo()
+reserve()
«input information»
Location
+address: String
«screen»
My Reservations
+navigateTo()
+delete()
+unlock()
reserve()
delete()
unlockCar()
lockCar()
«screen»
My Account
+navigateTo()
+changeName()
+changePassword()
+changePaymentPreference()
+deleteAccount()