Politecnico di Milano
A.A. 2016-2017
Prof. Elisabetta Di Nitto

Students:
Andrea Facchini - 874891
Andrea Milanta - 878403
Antonio Gianola - 877235

PowerEnJoy

Electric Car-sharing Service

SOFTWARE DESIGN DOCUMENT

Index

1. Introduction

1.1 Purpose

The purpose of this document is to identify the best suited software design for our system. This documents aims to give the developers a complete view of the software by describing all its main components and their interactions.

1.2 Scope

This document describes the software in all its functions.
Those aspects related to traffic size and memory requirement will not be directly addressed here. Instead all considerations are made considering a negligible traffic, but focused on scalability.
Description of external software interfaces towards online payment services and different transport administrations will also be limited to a standard interface, which will then be developed ad hoc for each external system.

1.3 Glossary

In this document the following acronyms and definitions have been used

- **PowerEnJoy Management System (PEMS):** It's the software system to be developed.
- **User:** The user of the software. He can register as a new user or access to the car sharing service via login and becoming an Authenticated User.
- **Authenticated User (AU):** The User after effectuating a successful login. The AU has access to the service and can reserve a car.
- **Suspended User:** User who has an account but cannot reserve a car because his driving license or payment information are found not be valid.
- **Geographical Region (GA):** Users can only reserve cars parked in this area.
- **Safe parking area (SPA):** Areas where the user is allowed to park.
- **Recharging area (RA):** Areas where the user can put the car in charge.
- **Money Saving Option (MSO):** Option for the AU to ask for a reduction. The system will suggest the user a SPA near the destination and suitable for a discount, according to the CT.
- **Call Center:** Telephone line managed by external
- **Onroad Service (OS):** People, provided by an external system but at the disposal of the PEMS, which provide autonomous support for exceptional situations directly on site. The OS also provides an emergency call center where the user may directly communicate with an operator.
- **Payment System:** Any generic external system through which a payment to PowerEnJoy happens.
- **National Transport System (NTS):** AU's country state agency that stores all driving licenses information to be validated by external users.

1.4 Reference Documents

- Assignments AA 2016-2017.pdf
- PowerEnJoy Requirement Analysis and Specifications Document – Version 2
- Runtime Diagrams.pdf
- Paper on the green move project.pdf
- Second paper on the green move project.pdf

1.5 Overview

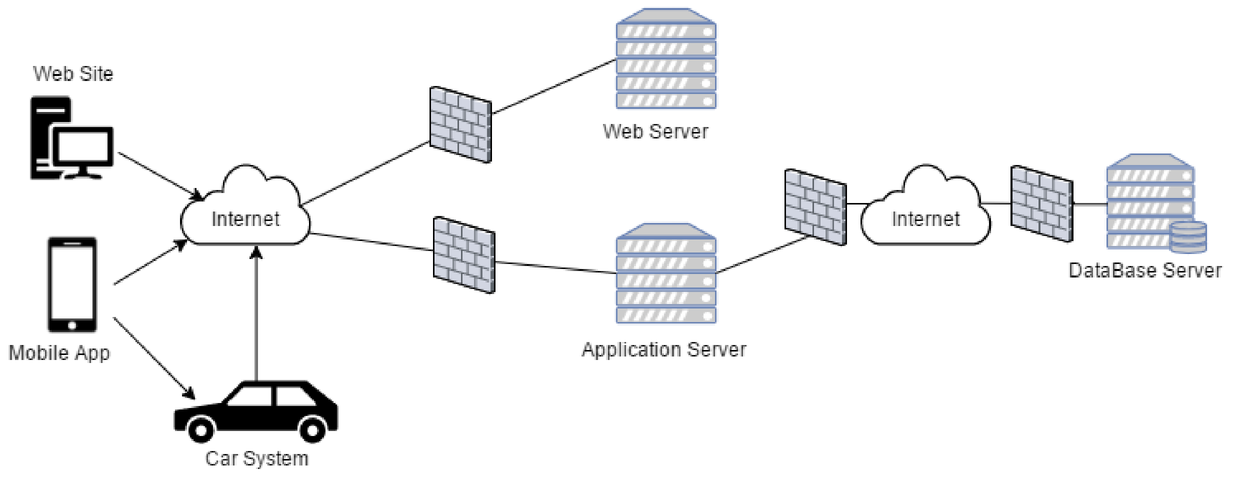
The present document contents the requirements for the PEMS project. This identifies, among others, the domain assumptions, the use cases, the users (stakeholders) and the requirements.
The document is organized in the following sections:

1. **Introduction**
   This section introduces the purpose, the scope, clarify the terminology and the acronyms and explain the general aspects of the project.
2. **Architectural Design**
   This section explains, using different diagrams, the architecture of the application. This include the high level, component, the deployment and the runtime view.
3. **Algorithm design**
   This section describes the most important algorithms. Pseudo code is used in order to hide unnecessary implementation details.
4. **User Interface Design**
   This section describes the UX and BCE diagrams, needed for describing the user interface and its interaction with the application. This section also includes an additional mockup.
5. **Requirements traceability**
   This section aims to explain how the decisions taken in the RASD are linked to design elements.

2. Architectural Design

2.1 Overview

The PEMS will follow a four-tier architecture.

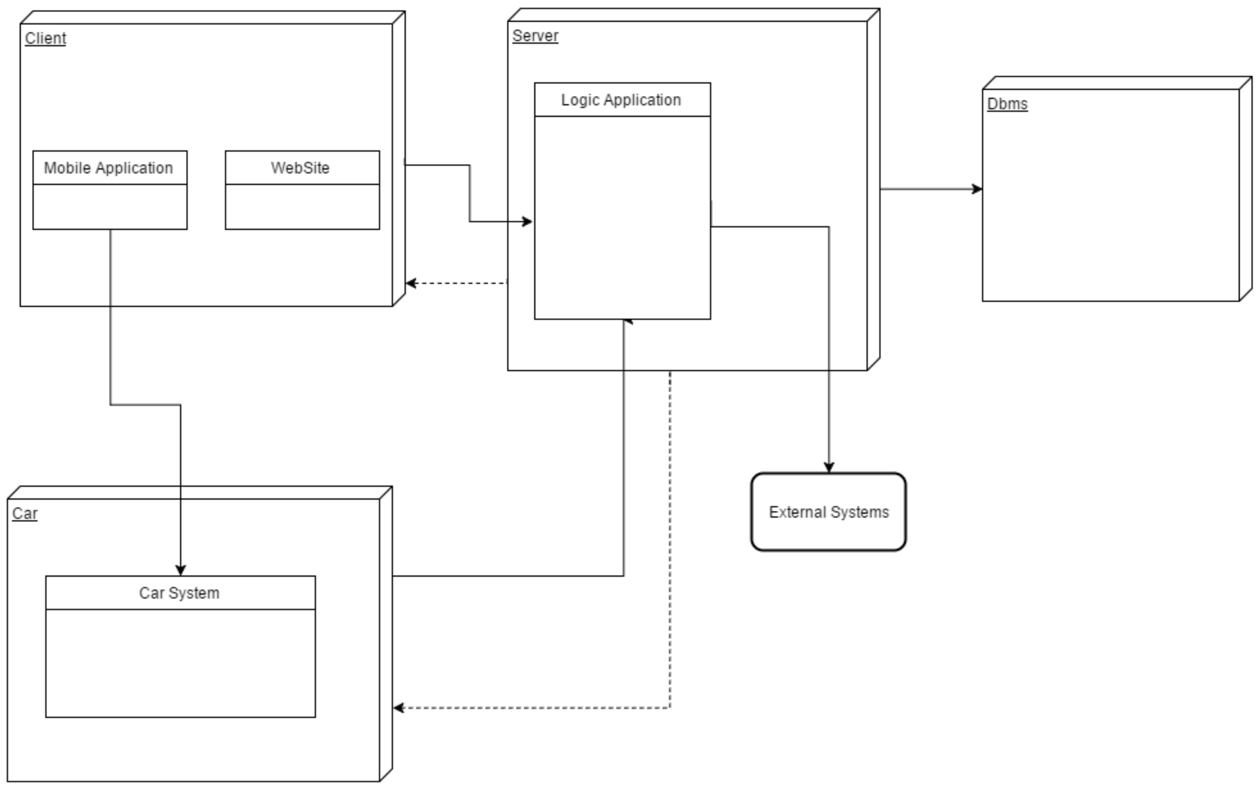


Architecture Overview Diagram

This architecture has been chosen because it allows for easy scalability and upgrading.
The “independent” GUI was considered mandatory to easily keep up with the new user experience technologies that will most likely be integrated in the future.
The separate application and database layers were chosen because they are easily scalable.

2.2 High Level Components

The system is made up of three main elements plus the database.
The core of the application is the server. This unique element hosts most of the application logic as it manages the users and the reservations.
Following the simple client-server logic architecture, the server receives requests from the clients, processes the data, possibly by reading and writing to the database, and gives out a response.
It has interfaces with third-party applications, in particular the NTSs and the payment systems, since it is also in charge of identification controls and payments.
The Car component manages the car. It provides the onboard user interface and is able to communicate with the car proprietary system to manage door and ignition locking, as well as to receive information about the car status.
This component contains some logic to allow the user access to the car via NFC and provide navigation.
The Client component on the opposite is only focused on the user interface.
The little logic contained is needed only to transform the user inputs from app or web to standard requests for the server and vice versa.

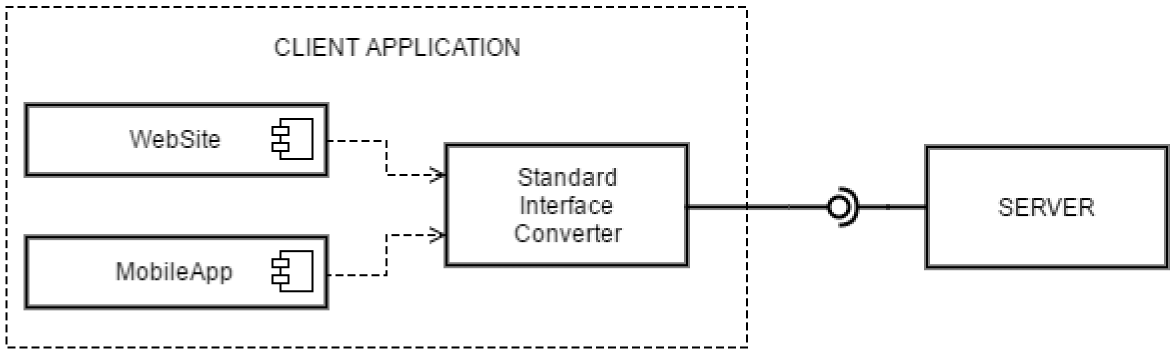


High Level Components Diagram

2.3 Components View

The final system will be made up of the following components:

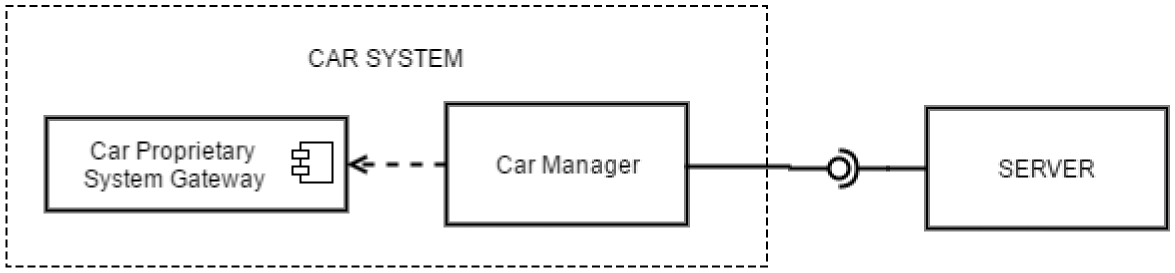
Client Application



Client Component Diagram

The Application client consists of a single module that only has to display the data received from the server and forward to the server the inputs of the user.
There are two components available for the client: a website or a mobile application.
Communication happens via a module that standardizes the interface for the server.

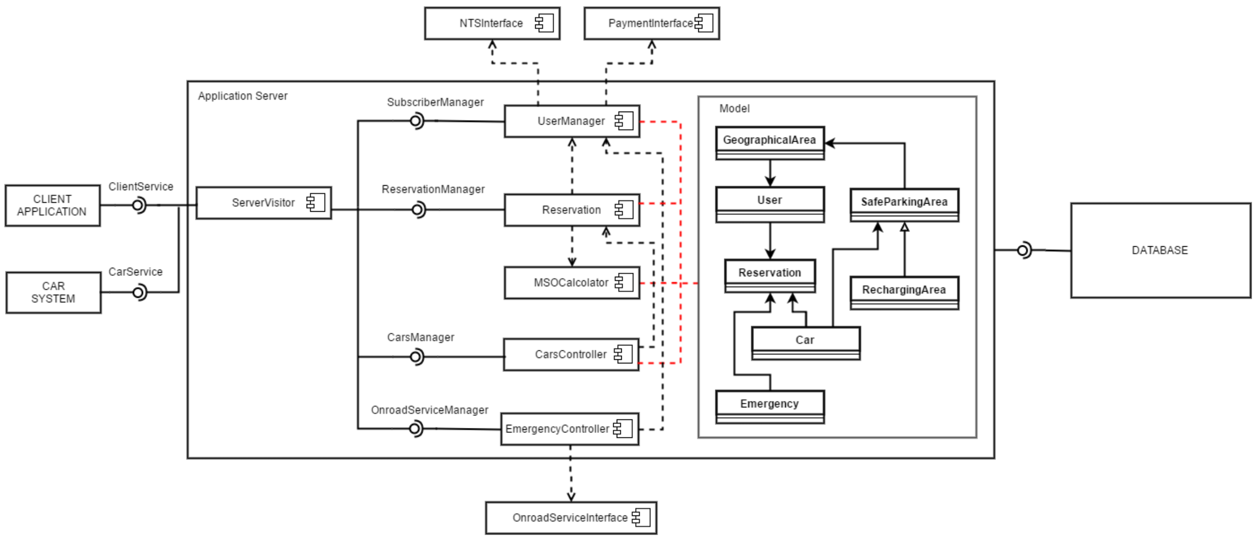
Car System



Car Component Diagram

The Car System has one component that manages all the functions related to the car. In particular, it manages the onboard display and communicates with the car proprietary system via a proper gateway.

Server



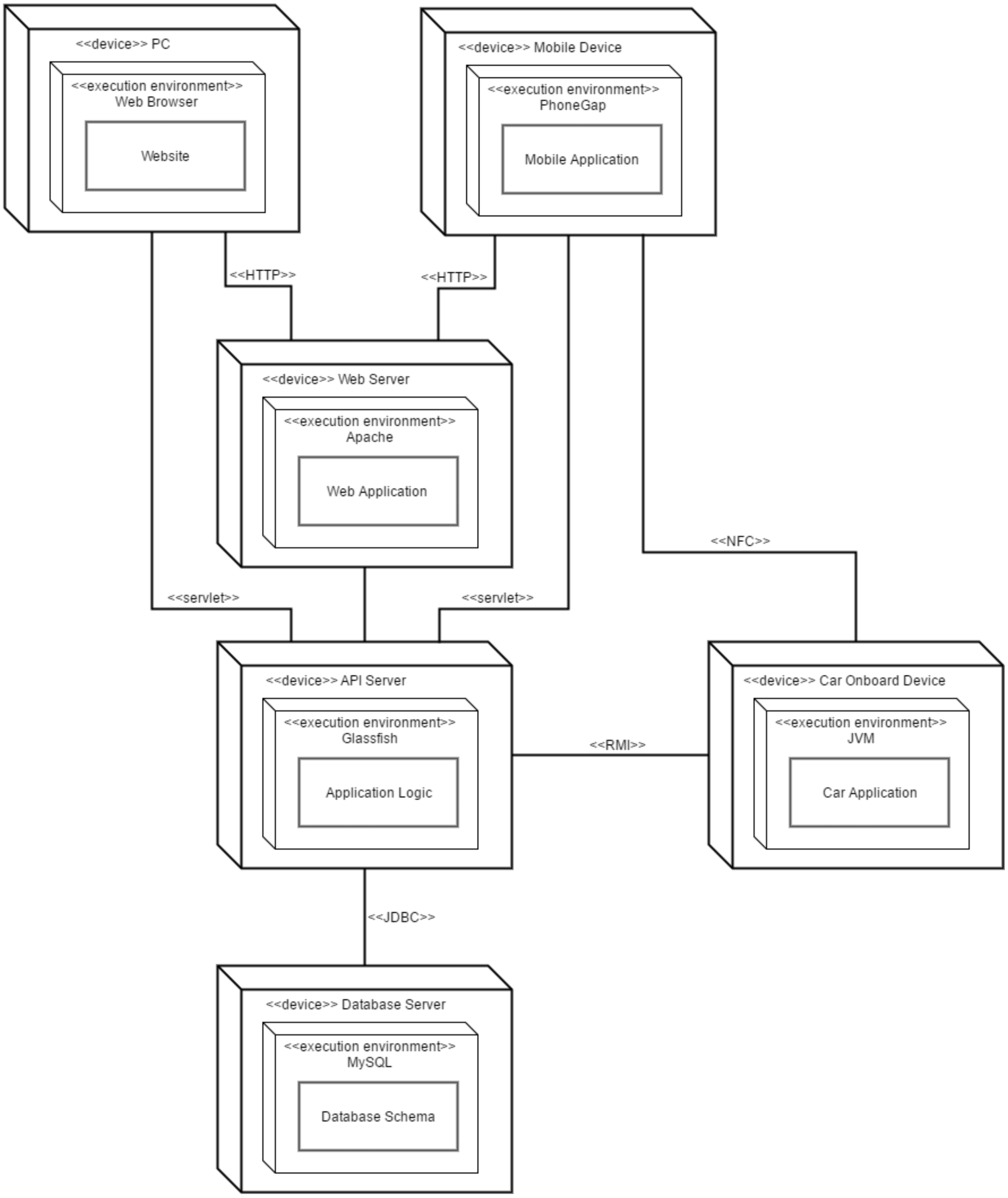
Server Component Scheme

The Server, where most of the logic is contained, is made up of many components.
Two main function groups can be identified though. One manages the user profile, registration and login, together with car positions and reservations.
The other takes care of all that is related to the ride, such as car status changes, MSO, etc.
More in detail the following components are defined:

- **User Manager:** Manages all users' information, including status, payment information and driving license.
- **CarFinder:** Takes care of finding the cars available for reservation and sorts them according to the distance from the user's position or inserted address.
- **Reservation:** Manages reservations. It contains the involved user and car, as well as the status of the reservation.
- **MSOCalculator:** Calculates the most suited destinations when a user sets the MSO. The algorithm is shown in the Algorithm Design (section 3.2)
- **EmergencyController:** Manages Emergencies, notifying the Onroad Service and charging the user when an extra fee is due.
- **ServerVisitor:** Dispatch incoming messages to the proper component.
- **XXXInterface:** Interfaces to third-party applications.

2.4 Deployment View

The system will work through six main logical devices.
The user interface is achieved by either a web browser or a mobile app.
The mobile app will be developed with PhoneGap to guarantee independency from the mobile platform.
Both the mobile app and the website will receive the main features to be displayed from a web server, while specific data and events will be shared with the application server using appropriate servlets.
The car interface will be a standalone Java executable, able to communicate with the application server via RMI, and equipped with an NFC interface to directly communicate with the mobile app.
The application server runs most of the logic of the system. Although it is represented as a single unit, it will most likely run on more than one machines.
The database communicates with the application server via a proper JDBC.
Database and application server are kept separated mainly for scalability purposes.
Actual implementation will happen using an Elastic Compute Cloud service. This allows for easy scalability and does not require any hardware maintenance.



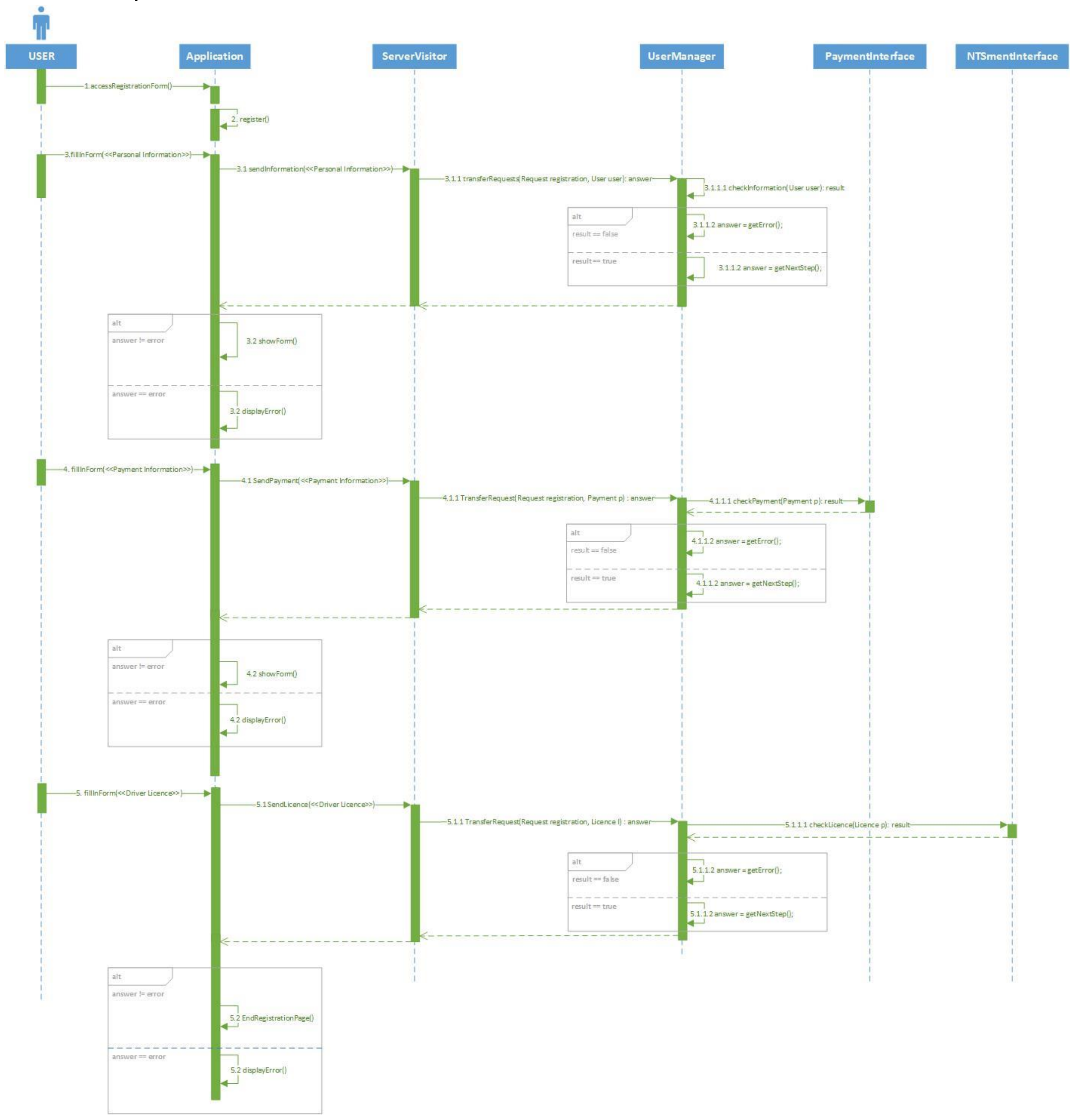
<<device>> PC
<<execution environment>> Web Browser
Website
<<device>> Mobile Device
<<execution environment>> PhoneGap
Mobile Application
<<HTTP>>
<<HTTP>>
<<device>> Web Server
<<execution environment>> Apache
Web Application
<<NFC>>
<<servlet>>
<<servlet>>
<<device>> API Server
<<execution environment>> Glassfish
Application Logic
<<RMI>>
<<device>> Car Onboard Device
<<execution environment>> JVM
Car Application
<<JDBC>>
<<device>> Database Server
<<execution environment>> MySQL
Database Schema

Deployment View Diagram

2.5 Runtime View

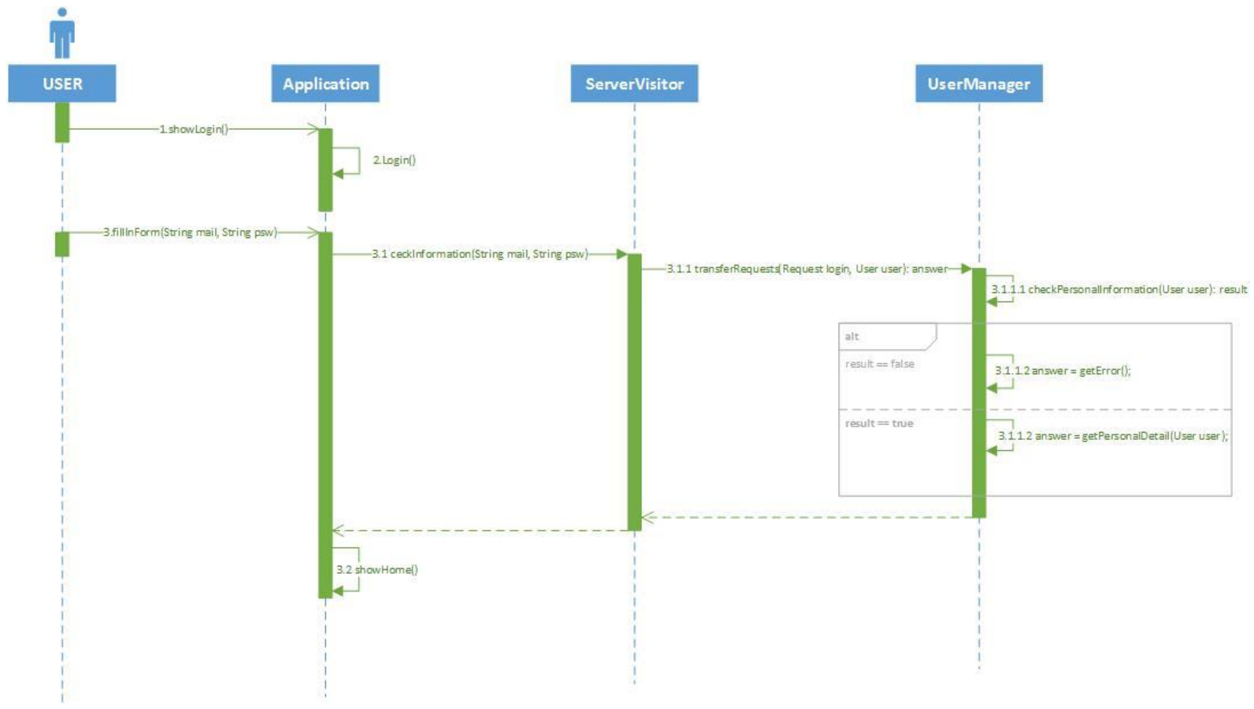
1. Registration

Registration happens in three phases.
In the first one personal information of the user are inserted and forwarded to the User Manager. The only controls are on birthdate (before today) and email (not already in user).
In the second one the payment information is inserted. Data correctness is achieved by executing a 1€ payment and checking the result. In the last phase the driving license data are inserted. This are controlled by checking on the NTS of the country of issue. Any failed control is reported to the user.


USER
Application
ServerVisitor
UserManager
PaymentInterface
NTSmentInterface
1.accessRegistrationForm()
2. register()
3.fillInForm(<<Personal Information>>)
3.1 sendInformation(<<Personal Information>>)
3.1.1 transferRequests(Request registration, User user): answer
3.1.1.1 checkInformation(User user): result
alt
result == false
3.1.1.2 answer = getError();
result== true
3.1.1.2 answer = getNextStep();
alt
answer != error
3.2 showForm()
answer == error
3.2 displayError()
4. fillInForm(<<Payment Information>>)
4.1 SendPayment(<<Payment Information>>)
4.1.1 TransferRequest(Request registration, Payment p) : answer
4.1.1.1 checkPayment(Payment p): result
alt
result == false
4.1.1.2 answer = getError();
result == true
4.1.1.2 answer = getNextStep();
alt
answer != error
4.2 showForm()
answer == error
4.2 displayError()
5. fillInForm(<<Driver Licence>>)
5.1 SendLicence(<<Driver Licence>>)
5.1.1 TransferRequest(Request registration, Licence l) : answer
5.1.1.1 checkLicence(Licence p): result
alt
result == false
5.1.1.2 answer = getError();
result == true
5.1.1.2 answer = getNextStep();
alt
answer != error
5.2 EndRegistrationPage()
answer == error
5.2 displayError()

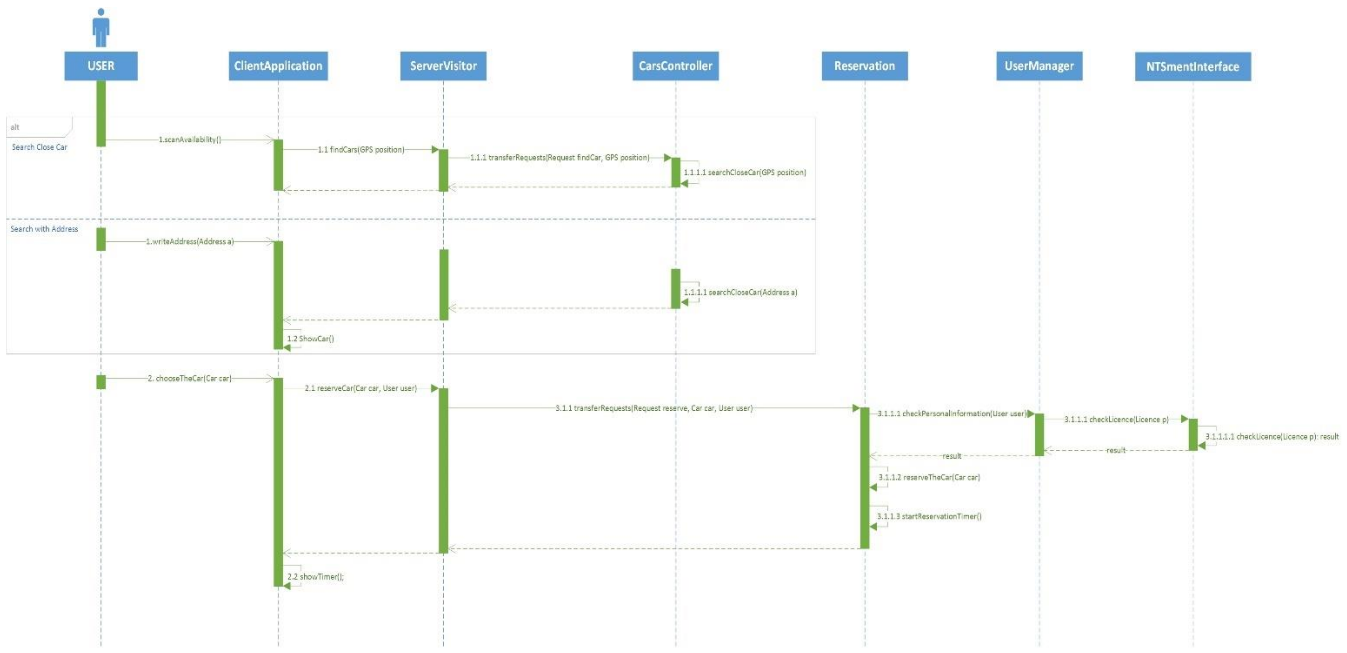
2. Login

The login follows the standard logic for such cases. The inserted credentials are forwarded to the user manages that checks their validity and sends back a feedback.


USER
Application
ServerVisitor
UserManager
1.showLogin()
2.Login()
3.fillInForm(String mail, String psw)
3.1 ceckInformation(String mail, String psw)
3.1.1 transferRequests(Request login, User user): answer
3.1.1.1 checkPersonalInformation(User user): result
alt
result == false
3.1.1.2 answer = getError();
result == true
3.1.1.2 answer = getPersonalDetail(User user);
3.2 showHome()

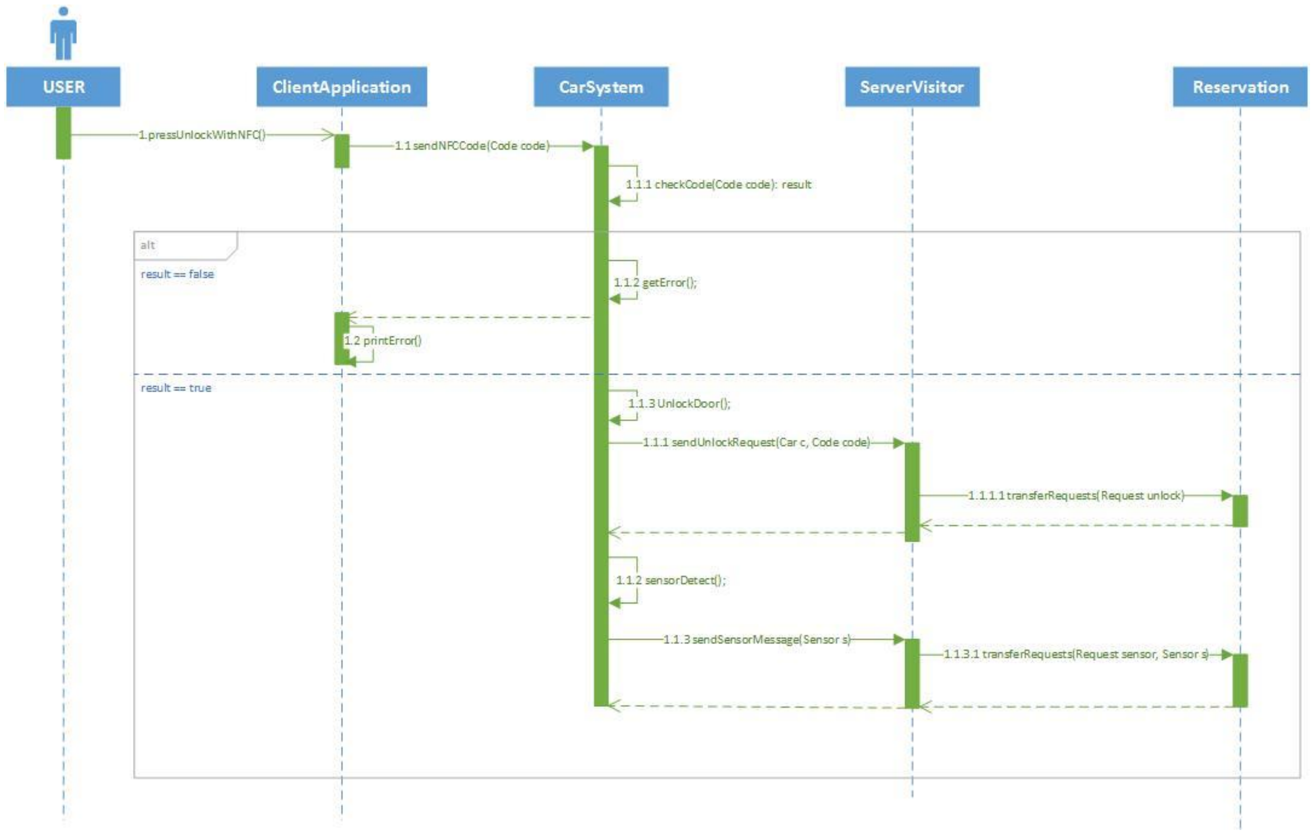
3. Reserving a car

Reservation can be achieved by either choosing a car on map, or by writing an address and choosing between the returned car list sorted by distance. The list is created and sorted by the server.
Once a car is chosen, the server checks the user's driving license validity and then proceeds in creating the reservation.
The car receives information about the reservation, in particular the reserving user NFC code.
The user is then displayed a timer, representing the time he has left to reach the car.


USER
ClientApplication
ServerVisitor
CarsController
Reservation
UserManager
NTSmentInterface
alt
Search Close Car
1.scanAvailability()
1.1 findCars(GPS position)
1.1.1 transferRequests(Request findCar, GPS position)
1.1.1.1 searchCloseCar(GPS position)
Search with Address
1.writeAddress(Address a)
1.1.1.1 searchCloseCar(Address a)
1.2 ShowCar()
2. chooseTheCar(Car car)
2.1 reserveCar(Car car, User user)
3.1.1 transferRequests(Request reserve, Car car, User user)
3.1.1.1 checkPersonalInformation(User user)
3.1.1.1 checkLicence(Licence p)
3.1.1.1.1 checkLicence(Licence p): result
result
result
3.1.1.2 reserveTheCar(Car car)
3.1.1.3 startReservationTimer()
2.2 showTimer();

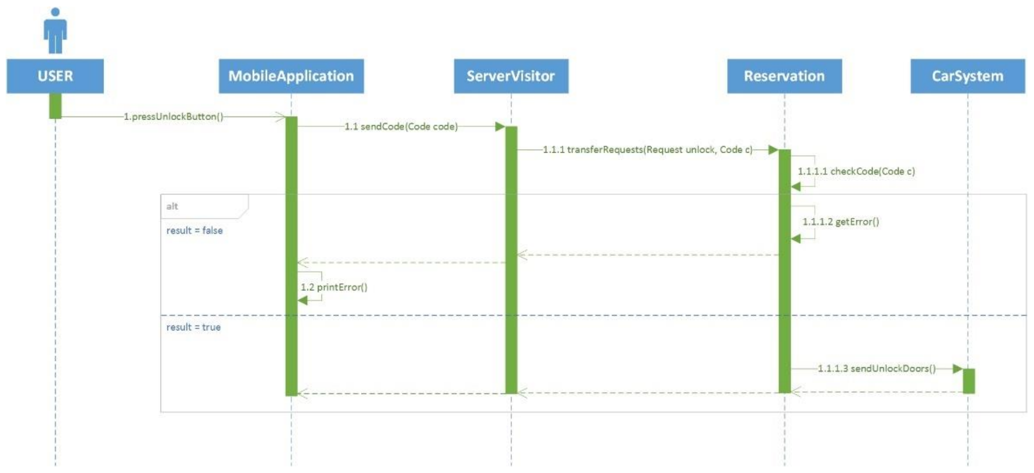
4. Unlock

The unlock process made directly via NFC does not require a server validation.
The user starts an unlock request which is received via NFC by the car. The car then checks whether the requesting user matches the one of its reservation, and if so unlocks the door. The server is notified after unlock and after the user has actually entered the car (sensor reading)


USER
ClientApplication
CarSystem
ServerVisitor
Reservation
1.pressUnlockWithNFC()
1.1 sendNFCCode(Code code)
1.1.1 checkCode(Code code): result
alt
result == false
1.1.2 getError();
1.2 printError()
result == true
1.1.3 UnlockDoor();
1.1.1 sendUnlockRequest(Car c, Code code)
1.1.1.1 transferRequests(Request unlock)
1.1.2 sensorDetect();
1.1.3 sendSensorMessage(Sensor s)
1.1.3.1 transferRequests(Request sensor, Sensor s)

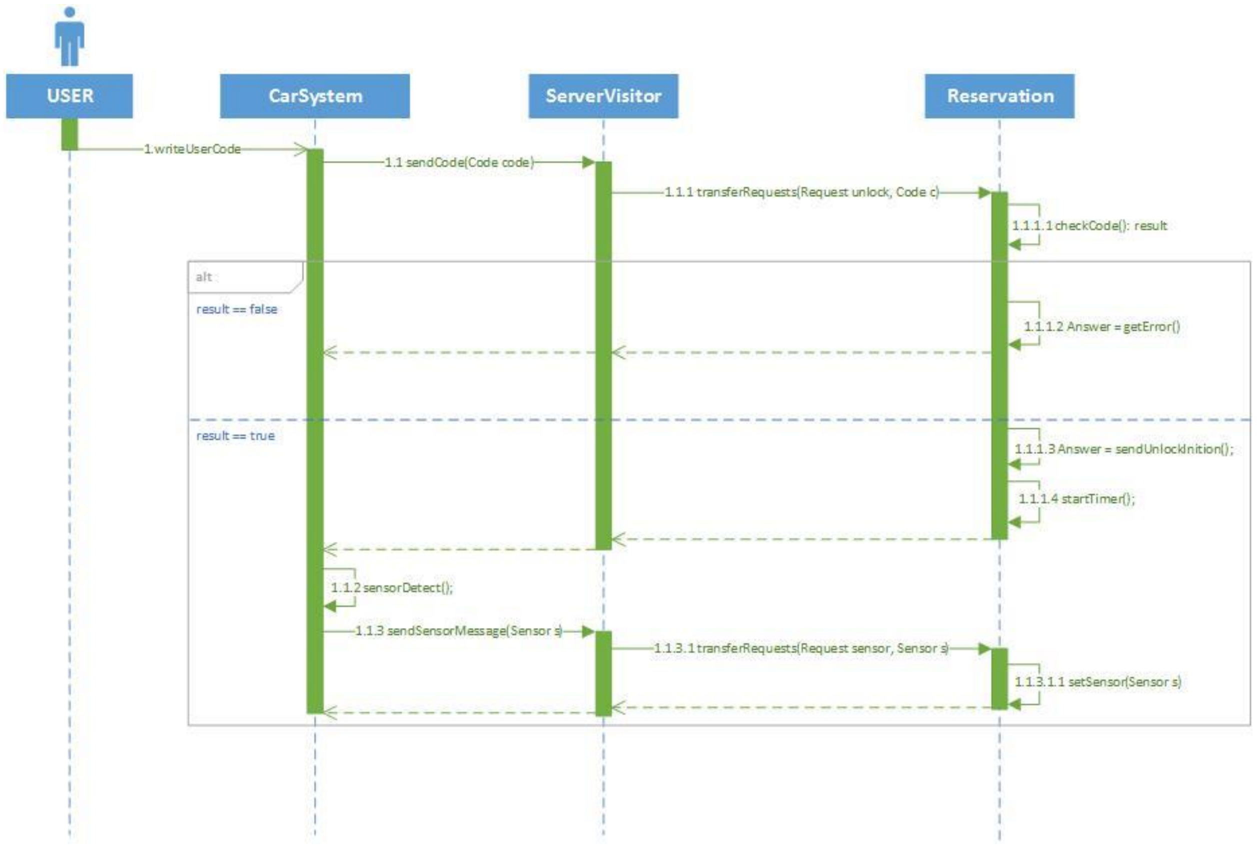
5. Unlock Mobile

The unlock from mobile works the same way as NFC but the user control happens on the server.



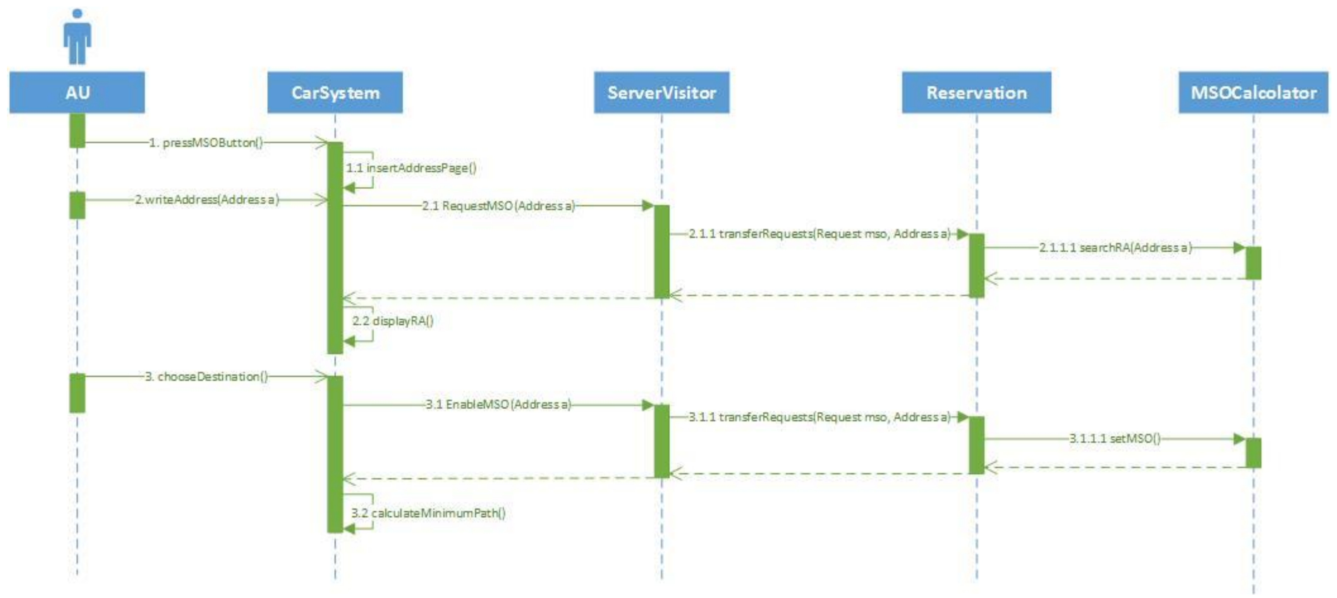
6. Starting the car

Once the user is in the car in has to write is personal code on the onboard display. This code is then forwarded to the server where it is verified. After successful verification the car is notified and unlocks the ignition. Afterwards the car sends to the server information about the number of passengers onboard (sensor reading)


USER
CarSystem
ServerVisitor
Reservation
1.writeUserCode
1.1 sendCode(Code code)
1.1.1 transferRequests(Request unlock, Code c)
1.1.1.1 checkCode(): result
alt
result == false
1.1.1.2 Answer = getError()
result == true
1.1.1.3 Answer = sendUnlockInition();
1.1.1.4 startTimer();
1.1.2 sensorDetect();
1.1.3 sendSensorMessage(Sensor s)
1.1.3.1 transferRequests(Request sensor, Sensor s)
1.1.3.1.1 setSensor(Sensor s)

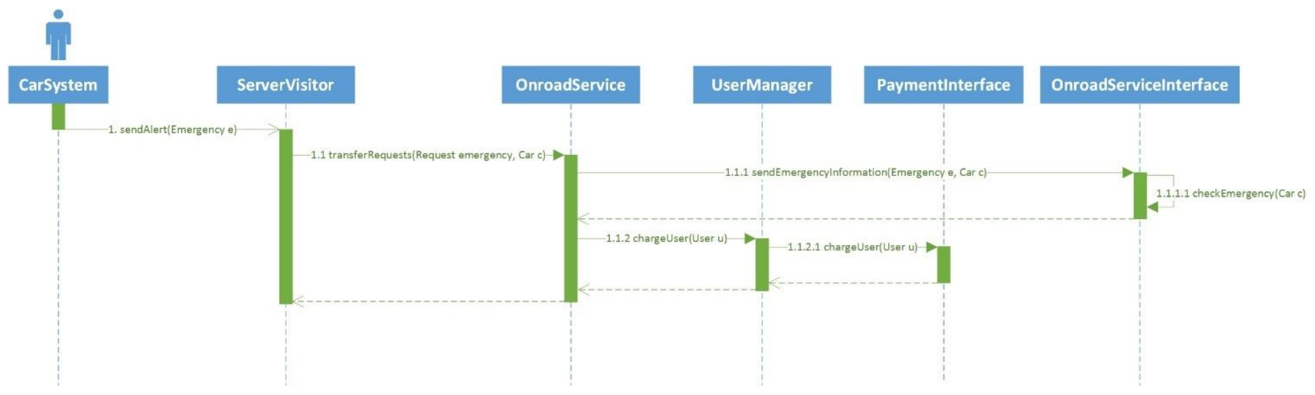
7. Money Saving Option

The money saving option starts when the user presses the MSO button.
The car then requires the user to insert the destination address, and forwards it to the server. In the server the MSOCalculator creates a list of available RAs and forwards it to the car. The user chooses the preferred RA, and is then displayed navigation information. At the same time the server is notified.


AU
CarSystem
ServerVisitor
Reservation
MSOCalcolator
1. pressMSOButton()
1.1 insertAddressPage()
2.writeAddress(Address a)
2.1 RequestMSO(Address a)
2.1.1 transferRequests(Request mso, Address a)
2.1.1.1 searchRA(Address a)
2.2 displayRA()
3. chooseDestination()
3.1 EnableMSO(Address a)
3.1.1 transferRequests(Request mso, Address a)
3.1.1.1 setMSO()
3.2 calculateMinimumPath()

8. Emergency

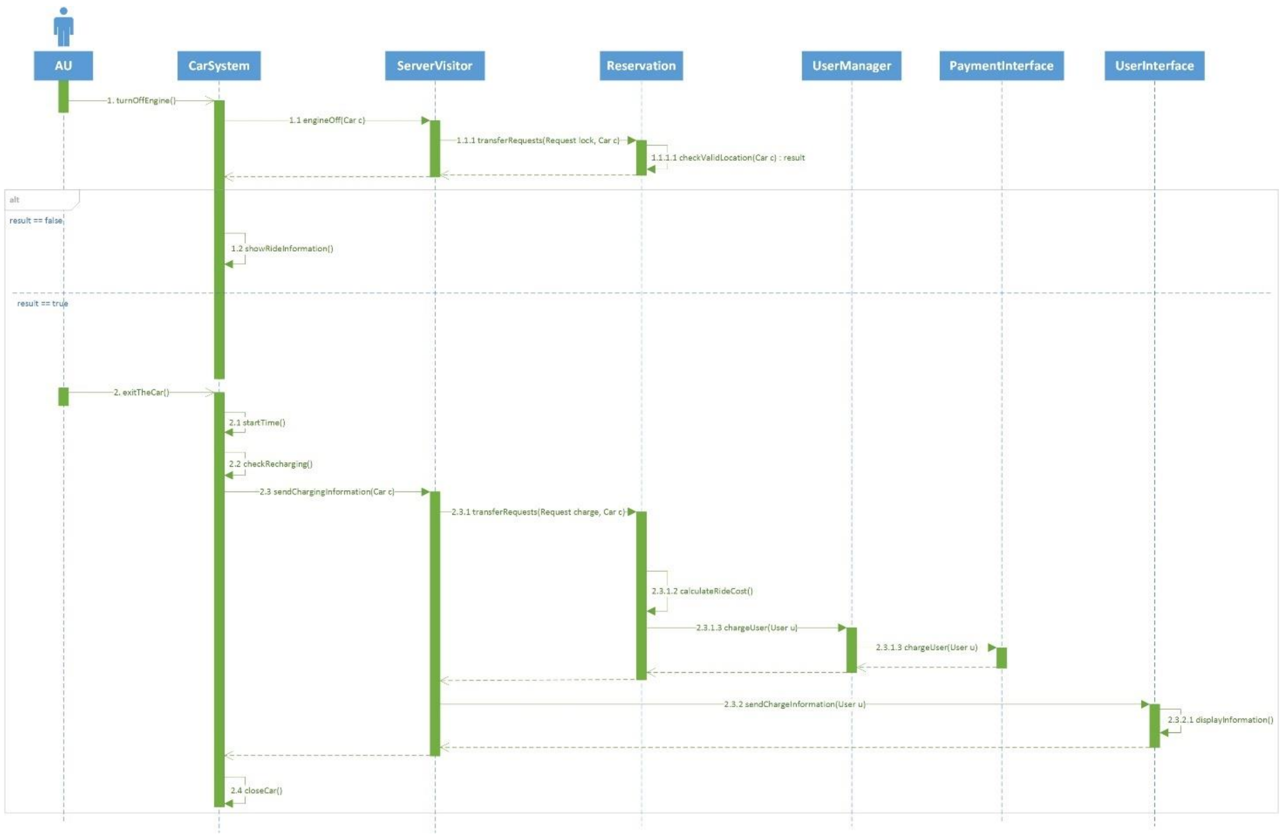
When an emergency happens the server is notified by the car itself
The onroad service is then notified. After checking the situation, the OS tells the user manager what sort of emergency happened, and the user is charged accordingly.


CarSystem
ServerVisitor
OnroadService
UserManager
PaymentInterface
OnroadServiceInterface
1. sendAlert(Emergency e)
1.1 transferRequests(Request emergency, Car c)
1.1.1 sendEmergencyInformation(Emergency e, Car c)
1.1.1.1 checkEmergency(Car c)
1.1.2 chargeUser(User u)
1.1.2.1 chargeUser(User u)

Andrea Facchini
Antonio Gianola
Andrea Milanta

9. Leaving the car

When the user turns off the engine the server is notified and checks if the car is in a valid spot. Once the user gets off. The car locks and notifies the server. The paying process is then executed, and the user notified.


AU
CarSystem
ServerVisitor
Reservation
UserManager
PaymentInterface
UserInterface
1. turnOffEngine()
1.1 engineOff(Car c)
1.1.1 transferRequests(Request lock, Car c)
1.1.1.1 checkValidLocation(Car c) : result
alt
result == false
1.2 showRideInformation()
result == true
2. exitTheCar()
2.1 startTime()
2.2 checkRecharging()
2.3 sendChargingInformation(Car c)
2.3.1 transferRequests(Request charge, Car c)
2.3.1.2 calculateRideCost()
2.3.1.3 chargeUser(User u)
2.3.1.3 chargeUser(User u)
2.3.2 sendChargeInformation(User u)
2.3.2.1 displayInformation()
2.4 closeCar()

Note: All diagrams can be found in higher resolution in the " Runtime Diagrams.pdf " file

2.6 Architectural Styles and Patterns

Architecture

The architecture follows a standard four-tier architecture composed of:

- **Thin Client**: Simple user interface implemented as a mobile application or a website.
- **Web Server**: Manages the presentation layer, actually building the HTML pages that will be provided to the client.
- **Application Server**: Hosts most of the logic of the application. Receives user inputs from clients and provides data output. It manages all the external interfaces.
- **Database Server**: Data storage and transactions execution (DBMS).

The car is implemented as a thick client which is in direct communication with Application Server. It hosts the interface to the car proprietary system.

Design Patterns

The software has been designed following some of the most popular patterns.

- **Client-Server**: Main design for communication between layers.
- **MVC** (Model-View-Controller): Main pattern followed in the design of the system, in particular for the Application server.
- **Adapter**: Widely used for both client and external interfaces with the server.
- **Visitor**: Used in the application server to manage incoming messages and forward them to the proper component.

3. Algorithm Design

In this section we analyze the main algorithms used in the software.
Pseudocode is provided to help better understand the algorithm. It may be incomplete.

3.1 Fee Calculation

F01 Eligibility Check

The F01 fee increment is assigned when a user leaves a car with less than 20% of residual battery and at more than 3km to the nearest RA.

```
Car = Get car from reservation
ResBatt = get battery level from car
InCharge = Get from car if the car is charging
if !inCharge and ResBatt < 20 %
      SafeArea = get Safe Parking Area of car
      if SafeArea contains a Recharge Area
            RecArea = Recharge Area contained in SafeArea
            if minimum distance between car and RecArea < 3km
                  return false
            end if
      end if
      GeoArea = get Geographical Area from SafeArea
      RecAreas[] = set of recharging areas in GeoArea
      while RecAreas[] is not empty
            if minimum distance between car and RecAreas[i] < 3km
                  return false
            end if
      end
      return true
end if
return false
```

D01 Eligibility Check

The D01 fee decrement is assigned when a user starts his journey with two or more passengers.

```
Car = get car from reservation
numOfPeople = get number of people onboard from the car
if numOfPeople > 3
      return true
end if
return false
```

D02 Eligibility Check

The D02 fee decrement is assigned when a user ends the reservation with more than 50% of residual battery.

```
Car = Get car from reservation
ResBattery = get battery level from car
if resBattery > 50 %
      return true
end if
return false
```

D03 Eligibility Check

The D03 fee decrement is assigned when a user leaves the car charging

```
Car = Get car from reservation
if Car is charging
      return true
end if
return false
```

Final fee calculation

Final fee is calculated by adding to the standard fee the increment (if eligible) and the biggest between the eligible decrements.

```
Delta = 0;
Car = Get car from reservation
start = get start time from reservation
stop = get end time from reservation
time = stop - start
stdfee = time * fee_min
if F01 Check
      Delta = stdfee * 3 / 10
end if
if D03 Check
      Delta = Delta - stdfee * 3 / 10
else if D02 Check
      Delta = Delta - stdfee * 2 / 10
else if D01 Check
      Delta = Delta - stdfee * 1 / 10
end if
fee = stdfee + Delta
```

3.2 MSO Option

Find Closer Recharge Areas

This algorithm calculates the RAs available for the MSO according to the user's destination.
The algorithm looks for all the RAs within 500 meters from the user location. If none are found it tries again with a 1 km radius. The available RAs are then sorted according to the number of free spots, and is returned to the caller.

```
#define LIMIT 500m
#define MAX_LIMIT 1km

if Dest in Geographical Area
    GeoArea = get geographical area of destination
else
    GeoSet[] = set of all GeoAreas
    GeoArea = GeoSet[0]
    forall GeoSet[]
        if mindist(Dest, closestGeo) < mindist(Dest, GeoSet[i])
            GeoArea = GeoSet[i]
        end if
    end
end if
AvailableRA[] = empty
RASet[] = set of all RAs in GeoArea
forall RASet[]
    if RASet[i].freeSpaces > 0 and mindist(Dest, RASet[i]) < LIMIT
        add RASet[i] to AvailableRA[]
    end if
end
if AvailableRA[] is empty
    forall RASet[]
        if RASet[i].freeSpaces >0
          and mindist(Dest, RASet[i]) < MAX_LIMIT
            add RASet[i] to AvailableRA[]
        end if
    end
end if
if AvailableRA[] is empty
    return null
end if
sort AvailableRA[] according to freeSpaces
return AvailableRA[]
```

Note: mindist(x, y) returns the minimum distance between x and y.

4. User Interface Design

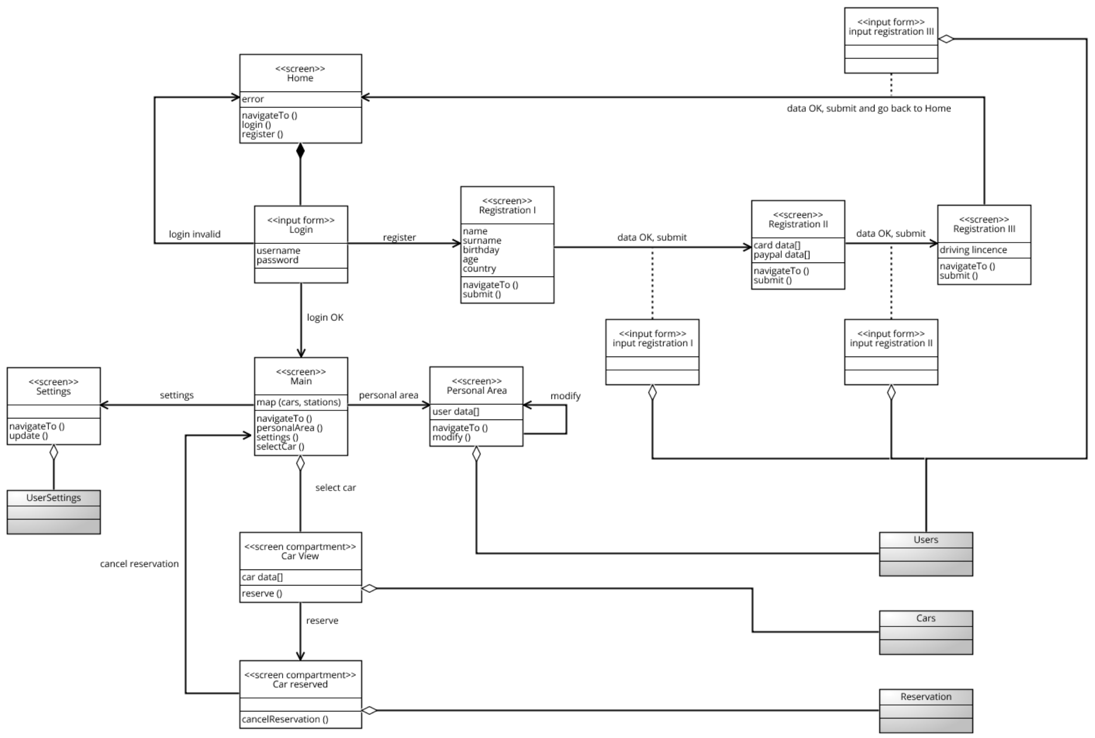
4.1 Overview

This section shows and explains in depth the user interface, providing detailed diagrams that depicts the logic behind the graphical interface of the application; This include UX and BCE diagrams. The second part features an additional mockup and a deep analysis of the graphical design, like colors and fonts used in the application.
Refer to chapter 4.1.1, page 40 of the RASD for the previous mockup.

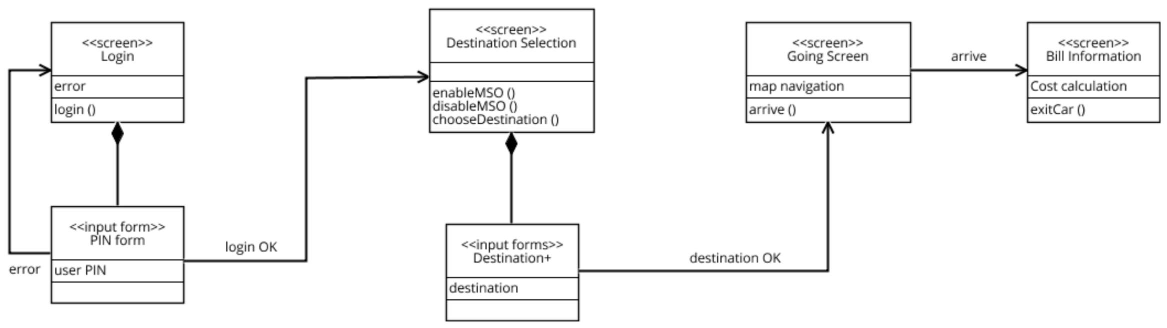
4.2 BCE and UX diagram

This chapter illustrates first the UX (User eXperience) diagram, that focus on showing the navigation of the user and the possible functions that each page provides. Then it shows the Boundary - UX diagram, that associates to each screen-type of the UX a boundary-type from the BCE diagram. The BCE is then shown as the last of the group.
Each diagram is shown twice: one time for the mobile/browser application and one time for the car screen. The 2 BCE diagrams contain some homonymous controller-type: in this case they refer to the same one.
Note that the BCE diagram's controllers doesn't match uniquely with the architecture's packages. Note also that the fields of the entity-type classes are omitted.

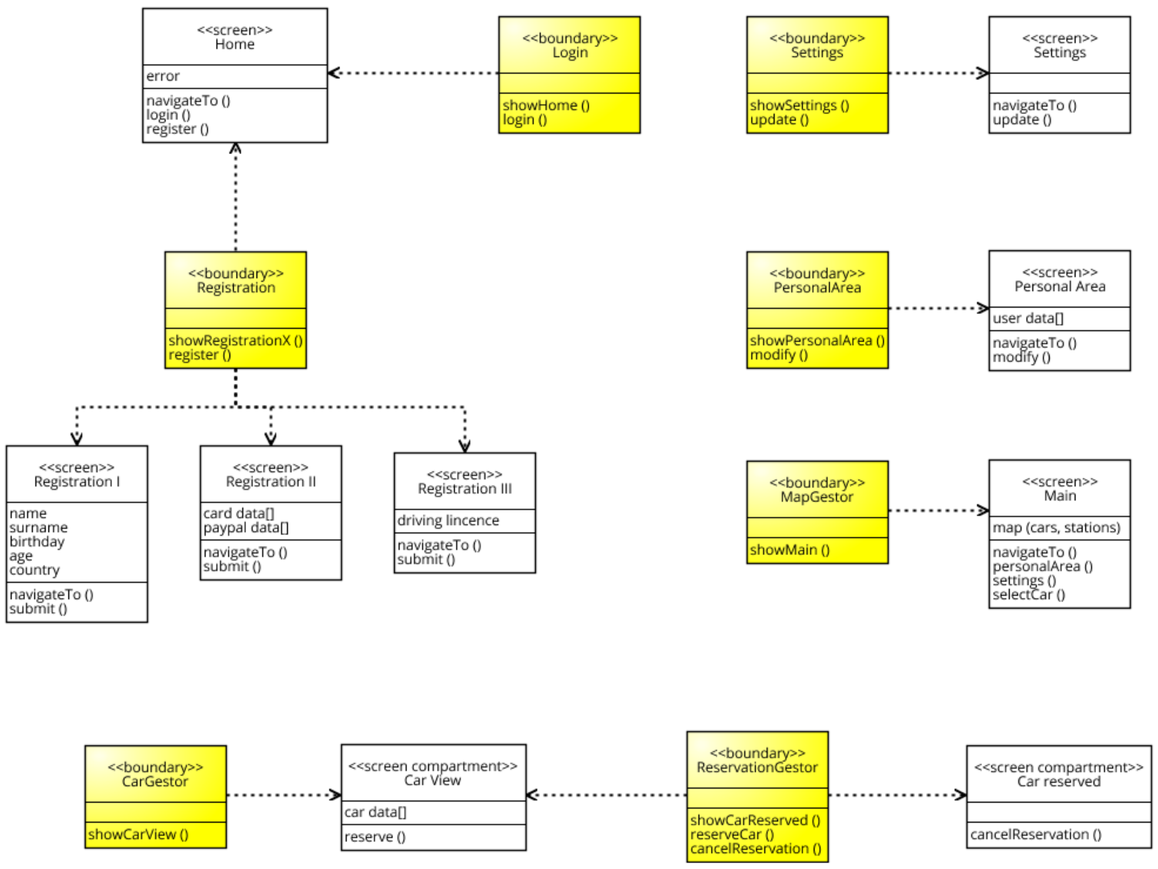
UX - Web Application



UX - Car screen application

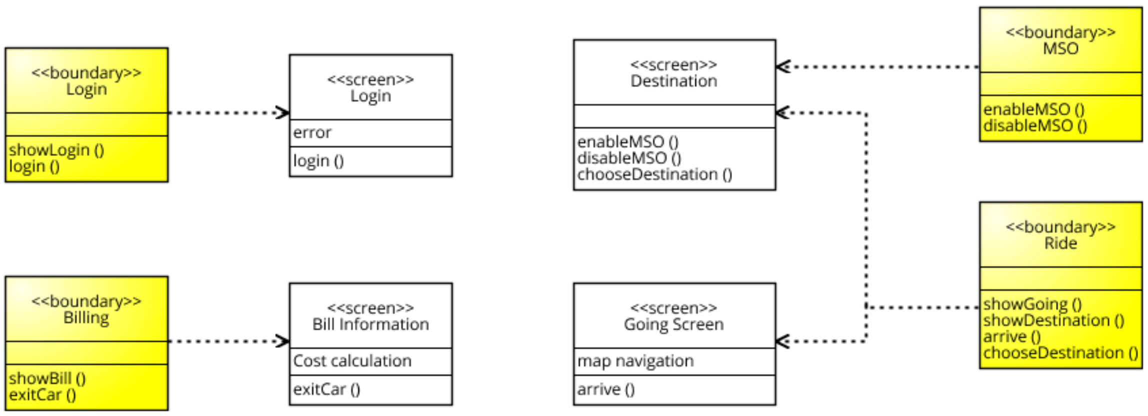


BCE-UX - Web Application

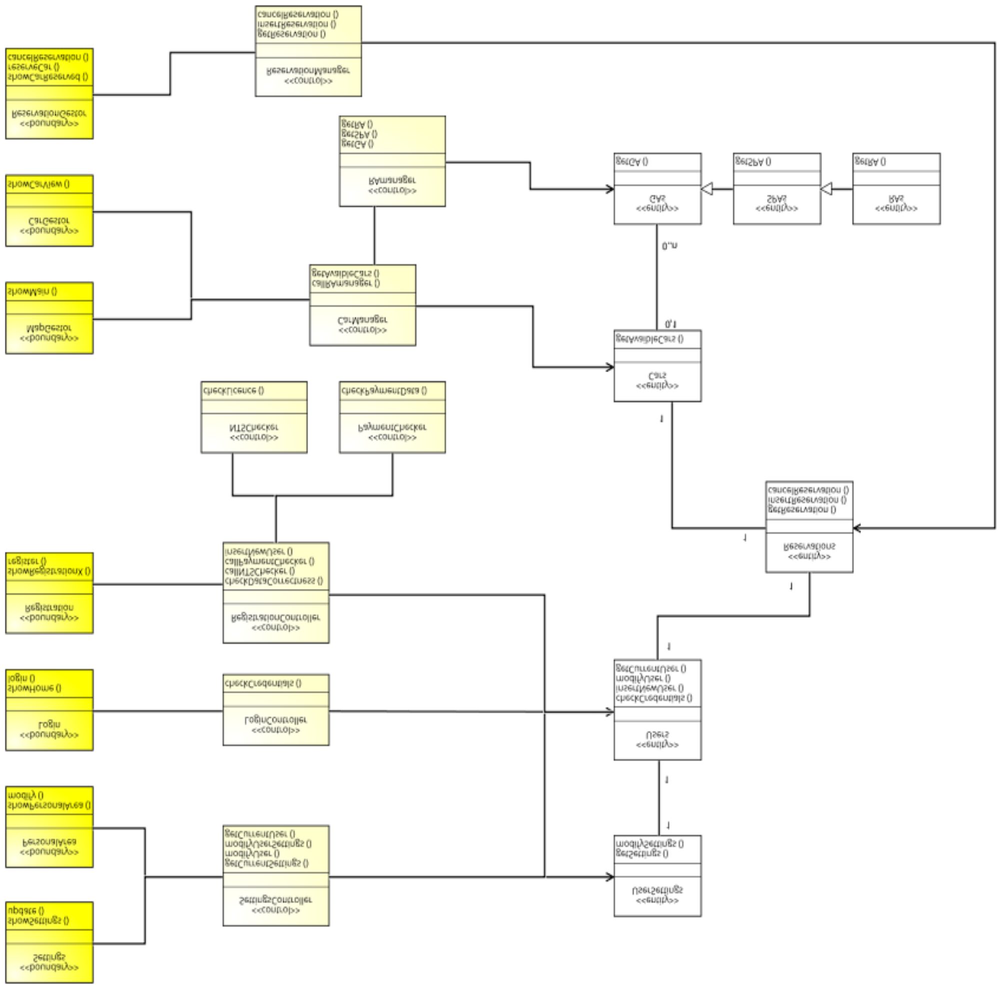


Andrea Facchini
Antonio Gianola
Andrea Milanta

BCE-UX - Car screen application

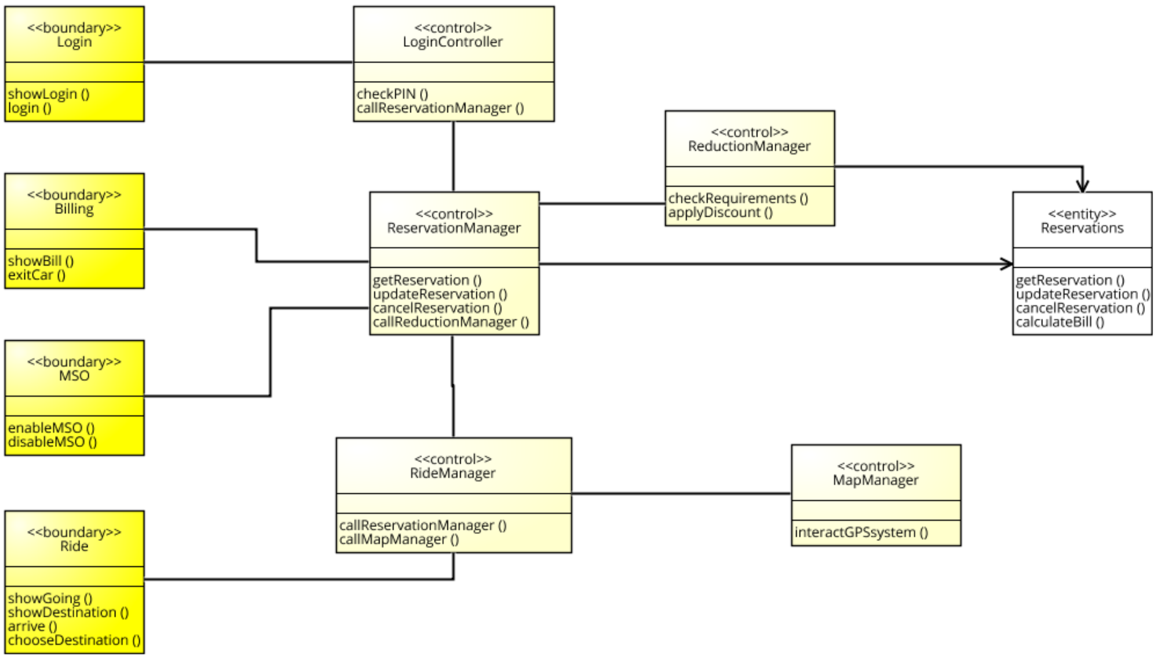


BCE - Web Application



Andrea Facchini
Antonio Gianola
Andrea Milanta

BCE - Car screen application



4.3 Additional Mockup

Registration

In addition to the previous document, the mockup of the registration page is following. The registration is composed by 3 phases: a main one, where the user inserts his personal data, one for the payment information and a last one for registering the driving license. The Pictures are shown sequentially

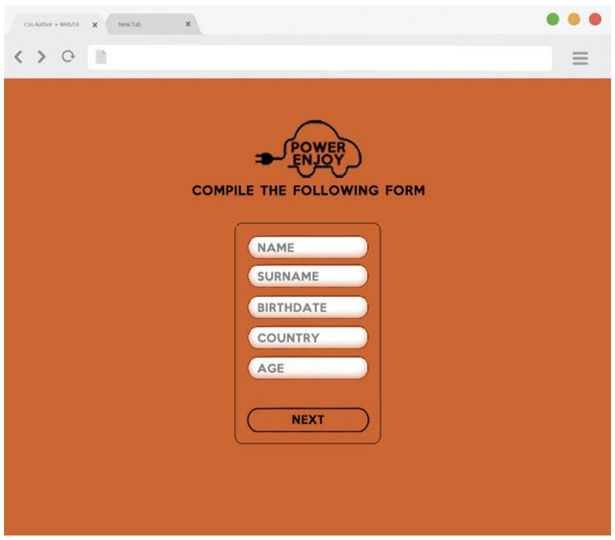


image 1.1

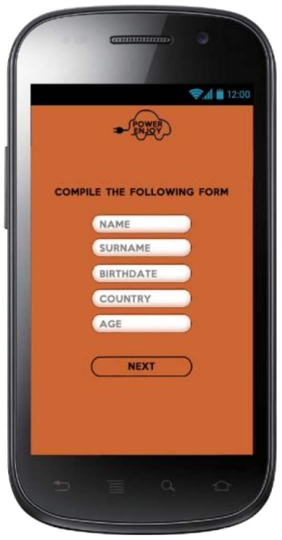


image 1.2

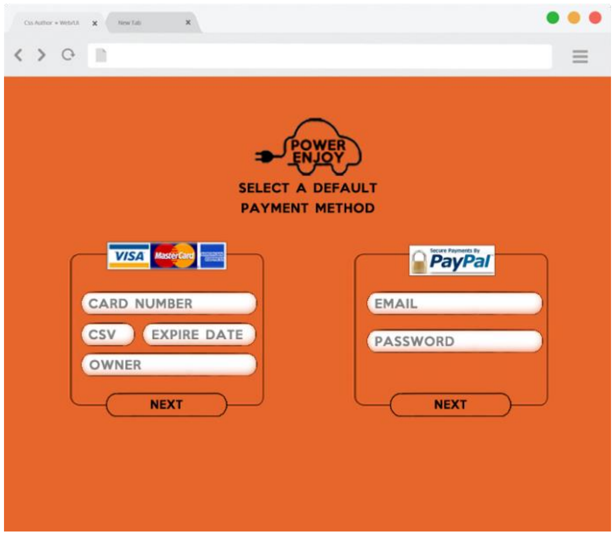


image 1.3

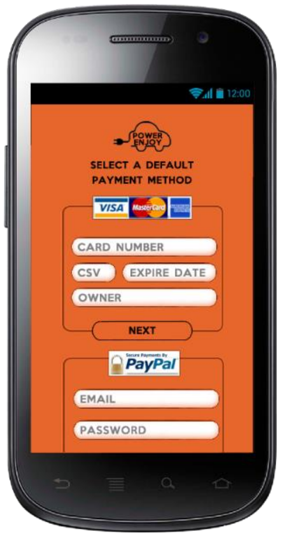


image 1.4

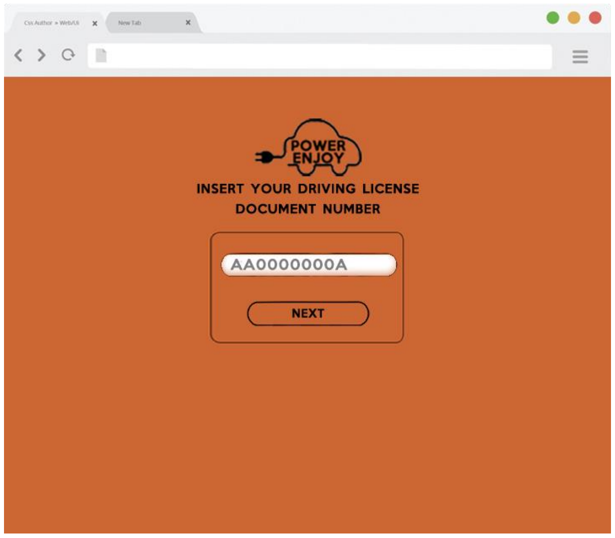


image 1.5

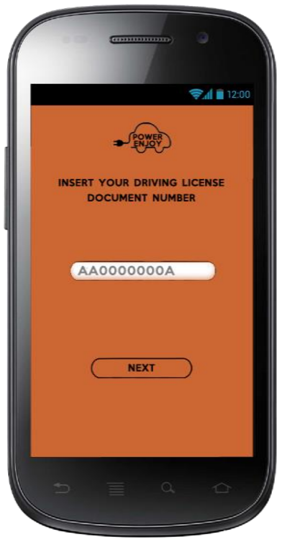
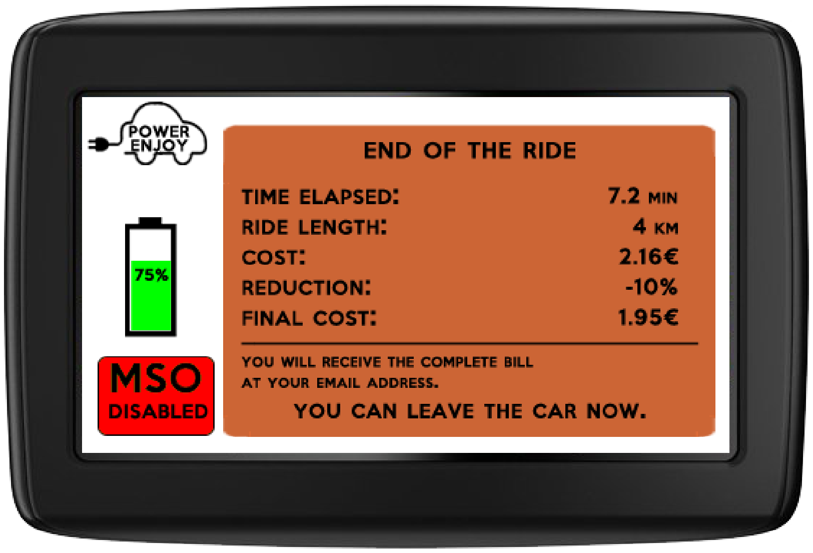


image 1.6

Payment Information

The second additional mockup is the Payment information page. This is shown at the end of the ride, when the user turns off the engine and is going to leave the car. It gives a fast recap on the last ride.



END OF THE RIDE

TIME ELAPSED:	7.2 MIN
RIDE LENGTH:	4 KM
COST:	2.16€
REDUCTION:	-10%
FINAL COST:	1.95€

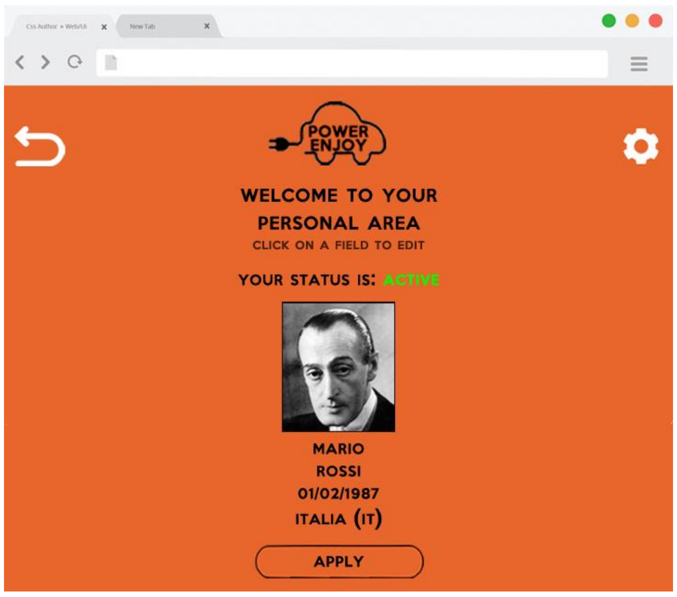
YOU WILL RECEIVE THE COMPLETE BILL AT YOUR EMAIL ADDRESS.

YOU CAN LEAVE THE CAR NOW.

Image 2.1

Personal Area

The third addiction consist in the personal area page and in the settings page. These are reachable from the corresponding icon in the main page.



WELCOME TO YOUR PERSONAL AREA

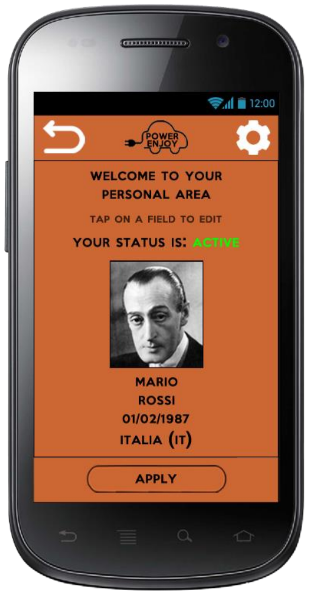
CLICK ON A FIELD TO EDIT

YOUR STATUS IS: ACTIVE

MARIO
ROSSI
01/02/1987
ITALIA (IT)

APPLY

image 3.1



WELCOME TO YOUR PERSONAL AREA

TAP ON A FIELD TO EDIT

YOUR STATUS IS: ACTIVE

MARIO
ROSSI
01/02/1987
ITALIA (IT)

APPLY

image 3.2

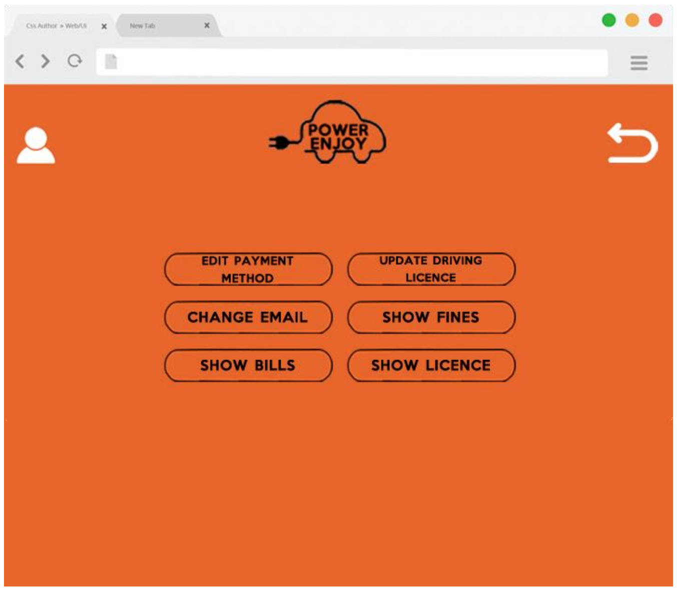


image 3.3

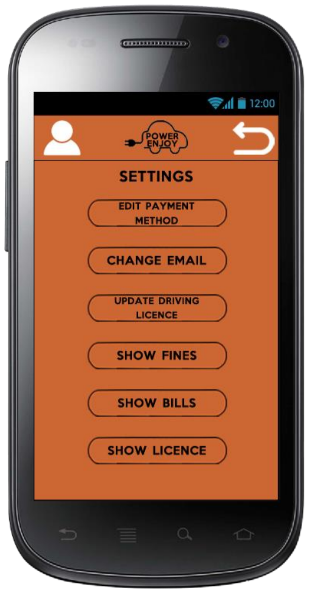


image 3.4

4.4 Detailed Component Description

Colors

The colors used in the mockup follow the provided table

Color Code (HEX)	Main Function	Example
#CC6633	Background, Car Indicator (Map)	🟧
#0099CC	RA Indicator (Map)	🟦
#333366	User Indicator (Map)	🟪
#663300	Keyboard (Onboard screen)	🟫

note: Black and White are omitted

Logo

The logo is in a solid black line containing the text “Power Enjoy”. The logo is the one provided and it’s never change color and proportions.

Font

The fonts used in the mockup follow the provided table:

Font Name	Main Function	Example
Keep Calm	Logo, Text	Keep Calm
Times New Roman	Map Text (street names, etc..)	Times New Roman

5. Requirements Traceability

The software was designed to fully satisfy the requirements and goals specified in the RASD. Below the set of requirements and goals are mapped to the designed component of the application which will assure its fulfillment.

- Managing the users: allow new users to subscribe to the service and already registered users to access the service.
  - The ServerVisitor
  - The UserManager
  - The NTS Interface and The PaymentInterface
- Allowing Authenticated Users to find a car close to their location and reserve it.
  - The ServerVisitor
  - The Reservation
  - The CarController
- Managing the reservations: Allow the user to reserve a car up to 1 hour before, and eventually charge him if he fails to reach the car within the hour.
  - The ServerVisitor
  - The Reservation
  - The NTS Interface
- Charging the user for the ride.
  - The UserManager
  - The PaymentInterface
- Managing charge reductions and increments based on the charging table.
  - The UserManager
  - The PaymentInterface
- Managing eventual exceptions such as accidents or traffic law fines.
  - The ServerVisitor
  - The OnroadService

6. Effort Spent

Work Division

section	Paragraph	author	revisioned by
Introduction			
	Purpose	Milanta	
	Scope	Milanta	
	Glossary	Milanta	
	reference documents	Milanta	
	Overview	Facchini	Milanta
Architectural design			
	Overview	Milanta	Gianola
	high level components	Gianola	Milanta
	components view	Gianola	Facchini
	deployment view	Milanta	Gianola
	runtime view	Gianola	Milanta
	Architectural Styles and Patterns	Milanta	Gianola
Algorithm design			
	fee calculation	Milanta	Facchini
	MSO option	Milanta	Facchini
User interface design			
	Overview	Facchini	Milanta
	BCE and UX diagrams	Facchini	Gianola
	additional mockup	Facchini	Gianola
	detailed component description	Facchini	Milanta
Requirements traceability			
		Gianola	Facchini
Effort Spent			
		Facchini	Milanta Gianola

Working Hours

	Week I							Week II							Week III							Week IV							tot
	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	1	2	3	4	5	6	7	8	9	10	11	12	
Antonio Gianola										1		2	1		2	2		2				2	2					**4**	**18**
Andrea Milanta			1					1				2			2	1				2		2					2	**4**	**17**
Andrea Facchini										2					2	2		2				2				2		**4**	**16**