Politecnico di Milano A.A 2016/2017

Software Engineering 2 project:

PowerEnJoy

Design Document (DD)

Alessandro Perini, Federico Saini, Ali Merd Türkçapar

Version: 2.0 Release date: 10/03/2017

Contents

Chapter 1

Introduction

1.1 Purpose

This is the Design Document for the PowerEnJoy management system.
The purpose of this document is to show the architectural, design and functional characteristic of the software that will be part of the PowerEnJoy system. The goal is to give high level descriptions of the components which are part of the system. All the decisions described in this document are taken based on the specifications listed in the related Requirement Analysis and Specification Document [2].
This document is intended to give a clear idea on how the system is composed.

1.2 Scope

The whole PowerEnJoy system is composed of different parts: cars, power plugs, a centralized software, web and mobile apps and a maintenance service. All these parts must be linked together and work jointly. So the goal of this document is to define how to develop the hardware architecture, the software and how they are related.

1.3 Definitions, Acronyms, Abbreviation

All the old definitions remain valid. Here only the new ones are listed.

RASD: Requirement Analysis and Specification Document.

DD: Design Document.

DBMS: Database Management System.

LAN: Local Area Network.

WAN: Wide Area Network.

BackOffice: is the software component that handle all the processes needed to manage the entire system.

1.4 Reference Documents

[1] PowerEnJoy specification document: *"ASSIGNMENTS AA 2016-2017.PDF"*.

[2] The related RASD: *"PowerEnJoy RASD"*.

[3] *IEEE Standard for Information Technology - Systems Design - Software Design Descriptions.*

[4] *IEEE Standard Systems and Software Engineering - Architecture Description.*

1.5 Document Versions

Version	Date	Description
1.0	11/11/2016	Document release. First version
2.0	10/03/2017	➔ BackOffice Diagram fixed. ➔ 2.2 Component View - Class description added ➔ Contents Updated ➔ Added a new section 2.5.5 Car Interfaces separately from Shared Interfaces now 2.5.6 ➔ App server API moved to App Server (2.5.1 to 2.5.2) section of 2.5 Component Interfaces ➔ Changes in component description ➔ A new component diagram added to 2.2.6 Car ➔ Changes in 2.2.2 Interface-Server, new diagram Communication Manager removed. ➔ 2.7.5 Registration Process description added.

1.5 Document Structure

The document is structured as follows:

- **Chapter 1 (Introduction)** gives an overall description of the document. Here purpose, scope, definition, references and the structure of the document are described.

- **Chapter 2 (Architectural Design)** provides representations of the fundamental structure of the system. It will describe both software and hardware components and the relation between them. So it will give different level descriptions of the entire system. It will also show the behaviour of the system in some use cases through the run-time views.

- **Chapter 3 (Algorithm Design)** describes, through the pseudo code, how the main algorithms of the system should be written.

- **Chapter 4 (User Interface Design)** gives a representation of the user interface code.

- **Chapter 5 (Requirements Traceability)** describes how the design decisions match the requirements listed in the RASD document.

- **Chapter 6 (Hours of Work)** takes track of the hour of work spent to write the document.

- **Chapter 7 (References)** is a list of the related document on which the DD is based

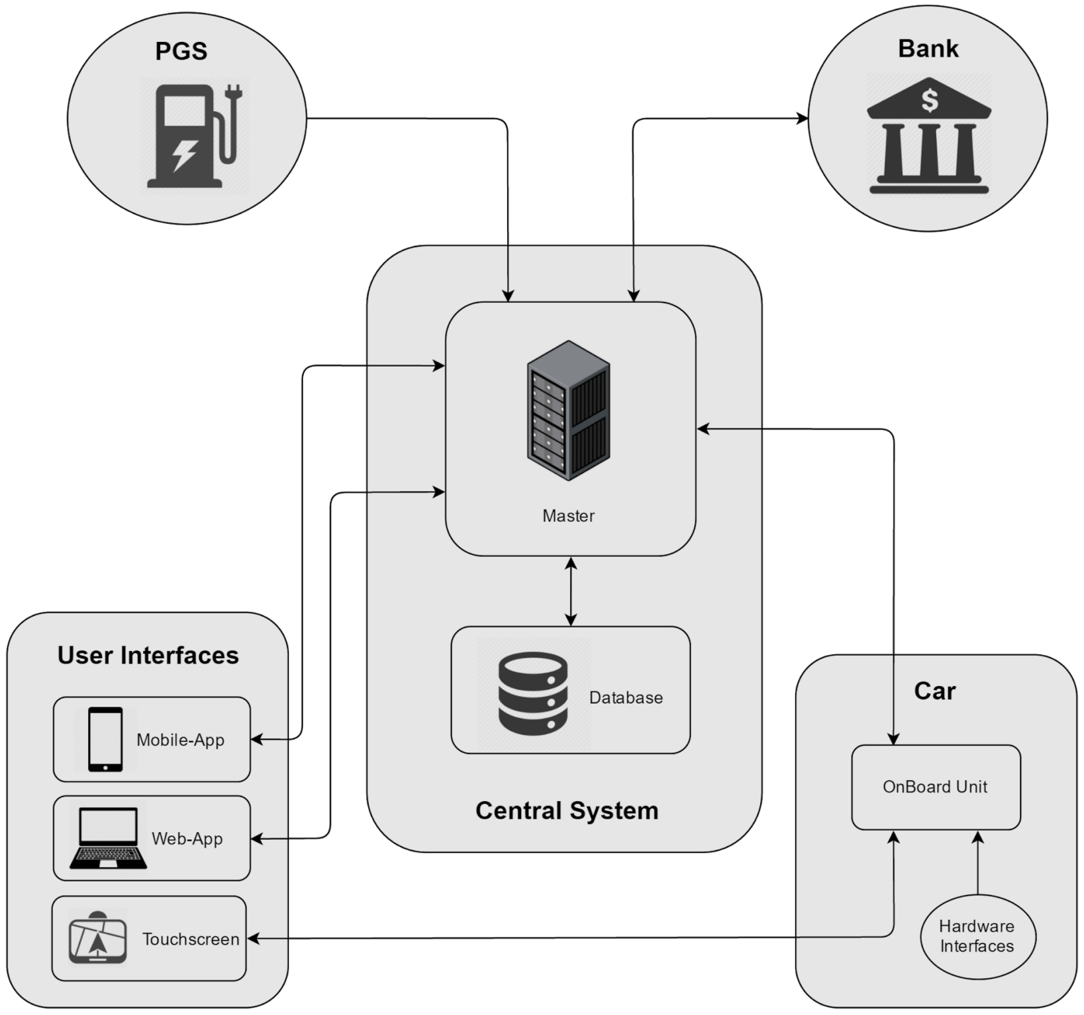
Chapter 2

Architectural Design

2.1 High Level Components and Their Interaction

2.1.1 Entire System

The following diagram gives a representation of the whole system. Arrows represent the data flow.



2.1.2 Tier View

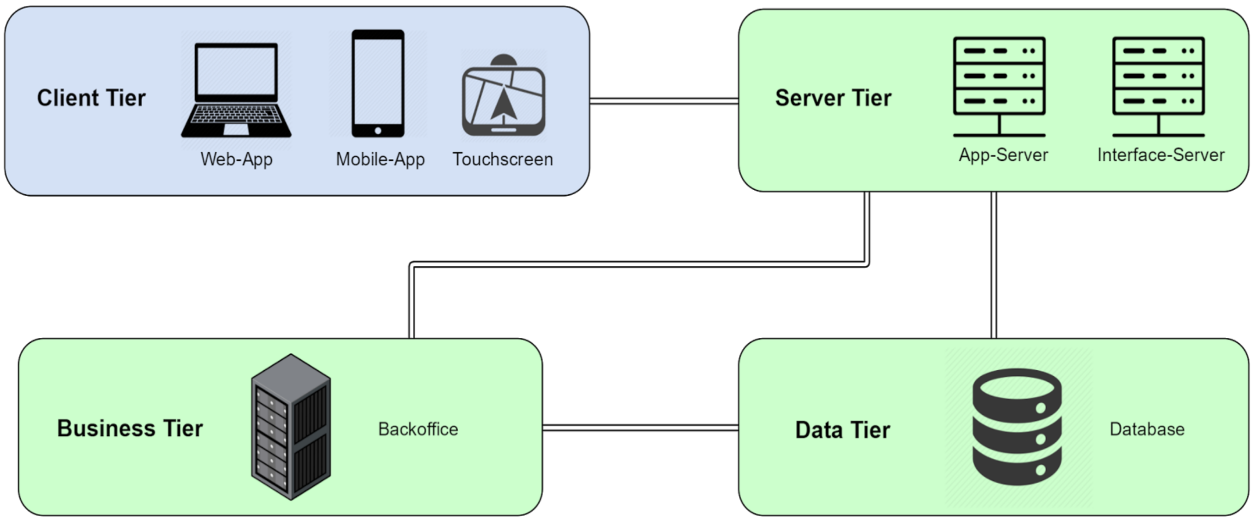
Our system is composed of a multi-tier architecture with 4 tiers:

Client Tier
The client tier is the level where user interacts with the system. This interaction is provided by the three user interfaces: the Web-App, the Mobile-App and the touchscreen integrated in the OnBoard Unit inside the car.

Server Tier
This tier is part of the central system and allows the user interfaces to communicate with it. The tier is composed of two different servers. The first is the App-Server that receives requests from the two applications. The second is the Interface-Server that receives data from the OnBoard Unit and the Power Grid Stations.

Business Tier
The business tier is composed of the BackOffice software. It manages all the functionalities related to the race and performs payments.

Data Tier
This tier is composed of a database with its DBMS. Database is accessible from the two servers and from the BackOffice software. Components communicate with it via sql queries.

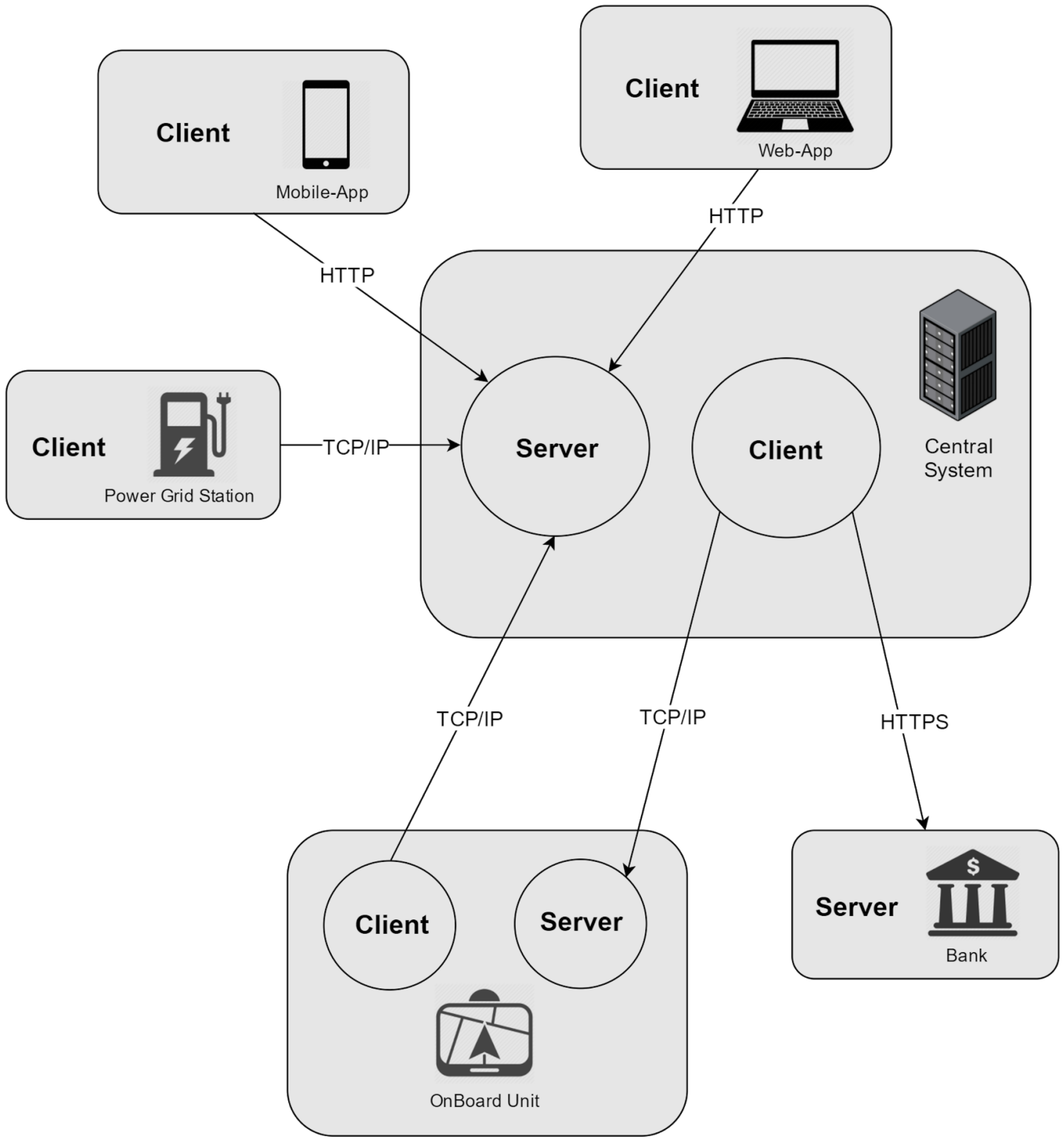


This tier representation is only a logical way to divide the system. In fact, app-server, interface-server and BackOffice can run on a single physical server but logically divided. Even the database can be part of the unique physical server or it can be separate and seen as an external service.

We decided to separate the server tier into two different servers to avoid possible complete interruptions of the service. In case of a failure of the app-server, for example due to a great number of user connection, the system can still manage the connections with all the car in use.

2.1.3 Client-Server View

The system is structured as a client server architecture where the main server is composed of interface, business and data tier, and then there are several clients. The arrows indicate which part of the system sends the request. OnBoard Unit has both client and server because it sends data about the car status and receives command, from the central system, like the unlock order when the user requests for it.



2.2 Component View

Here components of the system and their connections are described. Each component will be coded as a single class or as a package containing several classes, depending on the functionality that each component performs. Taken in example the BackOffice in paragraph 2.2.3, the Payment Manager, that is only in charge to perform payments, could be coded as a single class, differently, the Car Manager, that has multiple responsibility, could be divided in several classes. So, every component can represent one or multiple concrete classes depending on the programmers decisions.

Central System

The central system is the core of the system, where all the decisions are taken. It contains servers that allow the other part of the system to communicate with it and the database that stores all the relevant information. It is composed of two different servers that separate interactions between user interfaces and other interfaces. It has also a BackOffice component that provides functionality for internal operations that are not triggered by a client request.

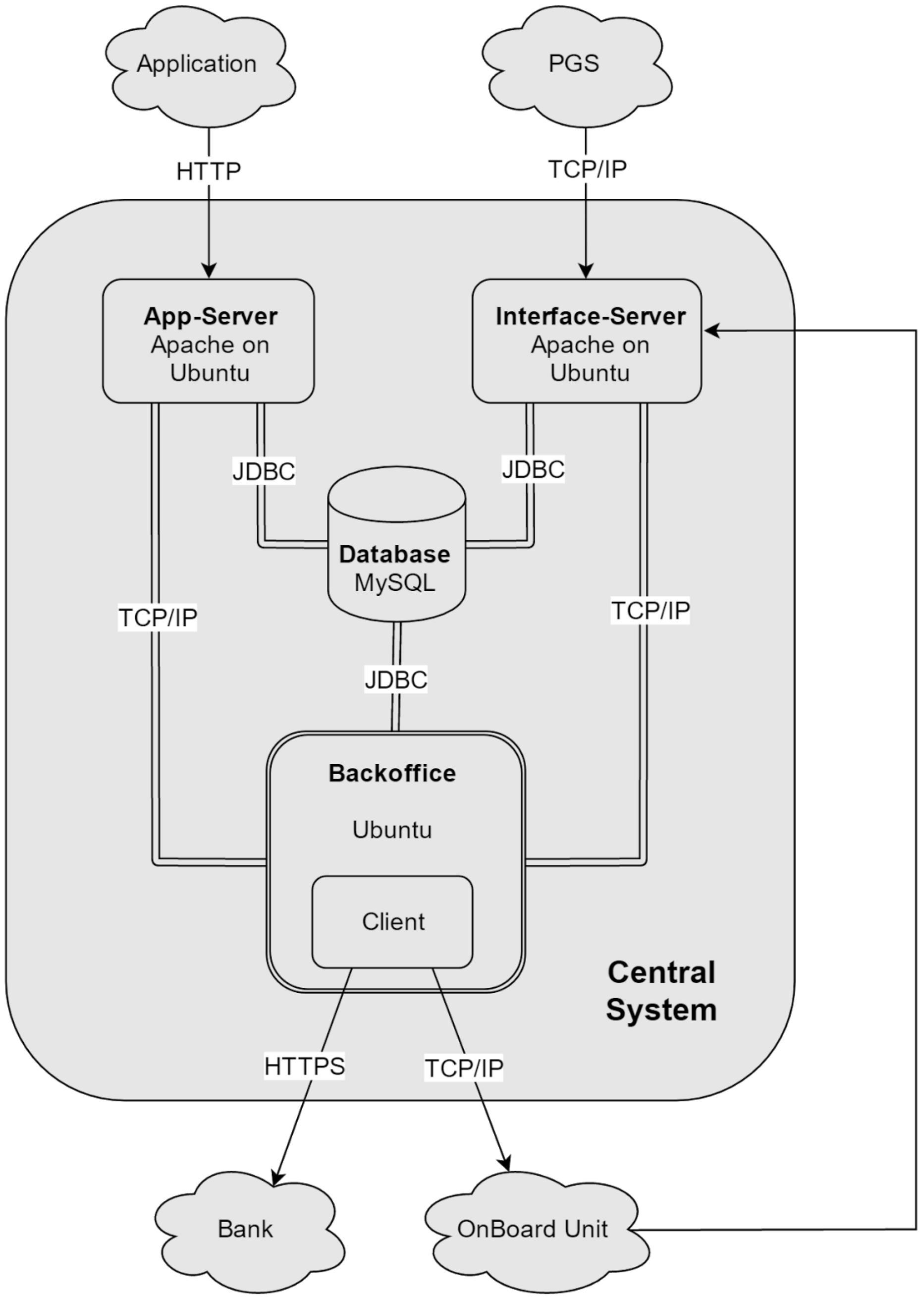
We decided to use a LAMP architecture because is a valid open source system.

Linux is the lowest-level layer and provides the operating system. Linux actually runs each of the other components. The next layer is Apache, the Web server. Apache, together with PHP provides the mechanics to create dynamic pages in the client applications. MySQL provides the data-storage side of the LAMP system.

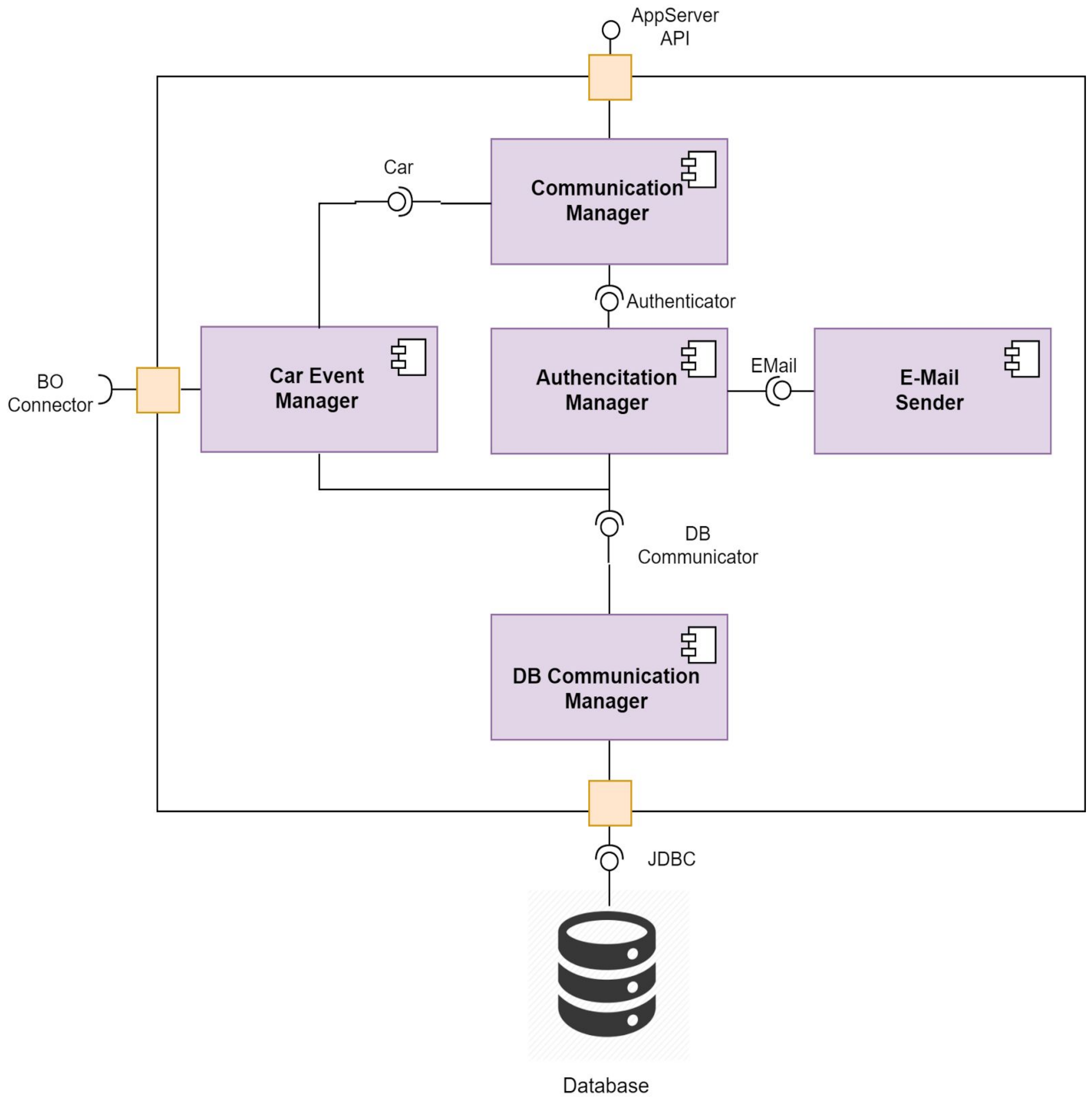
The components of the central system are connected with a LAN. BackOffice communicates with servers through a low level protocol (TCP/IP) because no high level communication is needed. All the components can query the database using the JDBC API which defines how a client can access to the database. We assume that the BackOffice, the main program that handle all the process needed to manage the entire system, is coded in Java. This Java program is running on the Ubuntu operating system.

User Interfaces

User interfaces are the applications and the touchscreen. To extend the range of users, we will develop two mobile applications, one coded in Swift for the iOS devices and one in JavaME for the Android operating system devices. The touchscreen interface will be coded in Java as the OnBoard Unit and all the parts of the central system.



2.2.1 App-Server



- **Communication Manager:** is the module that communicates with the applications. It receives requests and sends back responses. It is the main process of the server and it handles all the user sessions.

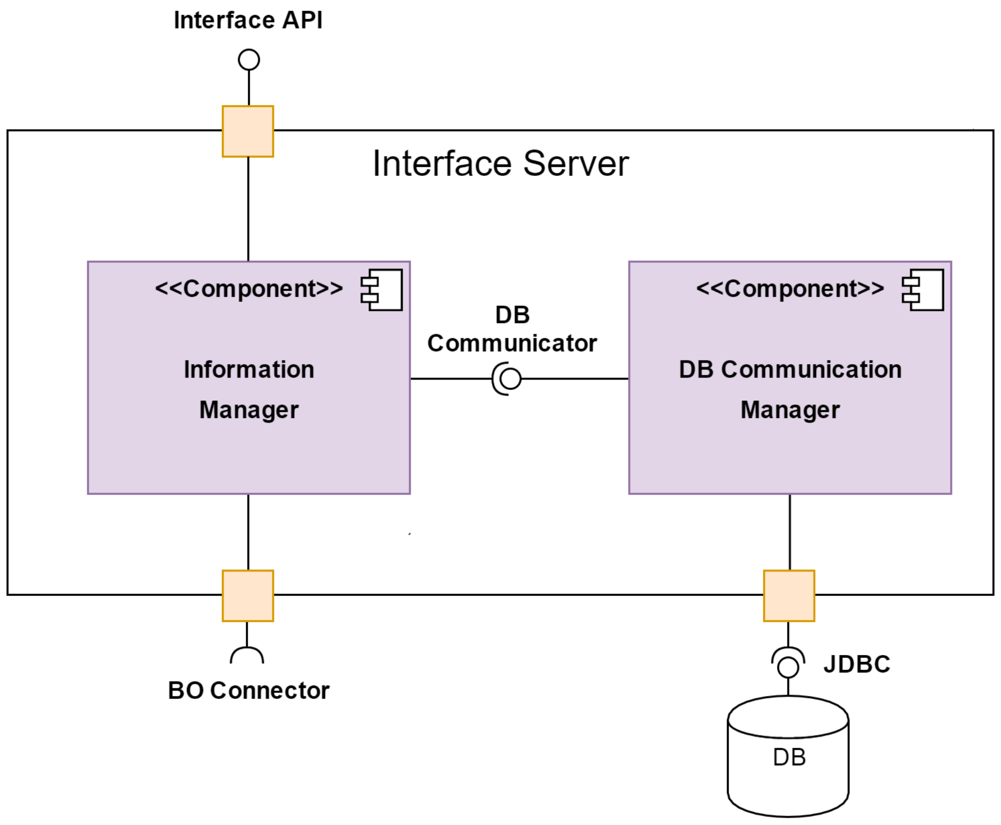
- **Authentication Manager:** it performs the registration and login procedures. It communicates with the database to check credentials and the validity of data during the registration. It also communicates with the email sender module to send the email with the generated password.

- **Email Sender:** it generates the first password for the registration process, build the email and then send it.

- **Car Event Manager:** manages all the event related to a car that come from applications. For example, car reservations, car unlocking by user or technician, issues solved by technician, etc. It can communicate with the BackOffice passing the operations related to a race.

- **DB Communication Manager:** is the only component that can directly contact the database. It receives requests from other components and then it submits the related queries.

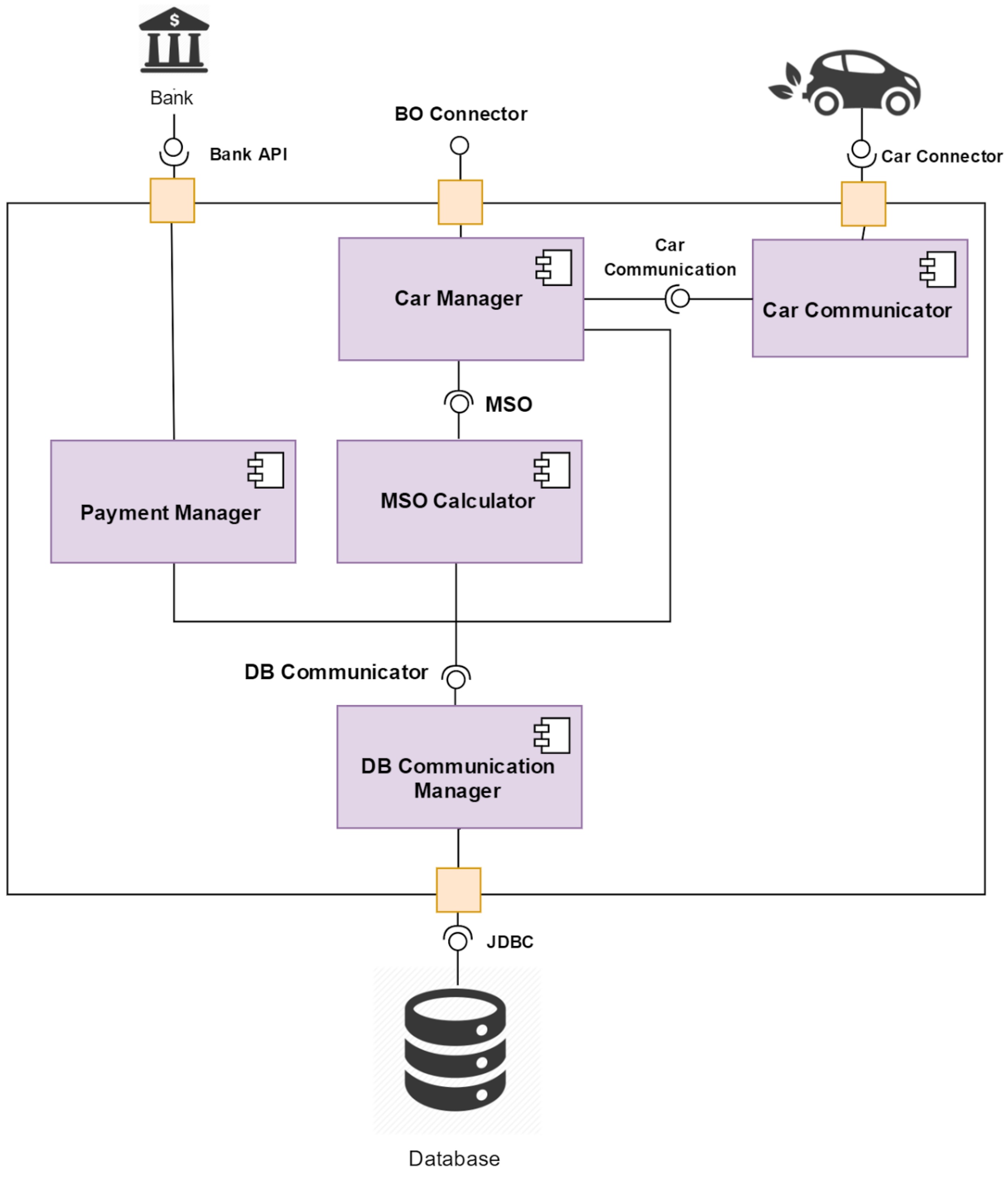
2.2.2 Interface-Server



- **Information Manager:** receives and interprets data from cars and power grid stations and then stores it in the database using the DB Communication Manager. It can also send information directly (using BO Connector) to the Car Manager without passing through the database. For example, destination address during the MSO request or the number of passenger during the ending procedure.

- **DB Communication Manager:** receives requests from car and PGS managers and then it submits the related queries.

2.2.3 BackOffice



- **Payment Manager:** is the component that is in charge to perform payments and check the validity of the user payment information.When a race ends, the Race Manager creates a new non completed transaction in the transactions table of the database with all the related information. The Payment Manager periodically checks into the transactions table all the non completed transactions and performs the payments contacting the bank. It also check the validity of payment information during the user registration phase.

- **Car manager:** is the component that handles all the operations related to a car. It receives requests from the App-Server Car Event Manager and creates a process related to every use of a car. When a user reserves a car, the process is created. Every process remains active until the ending procedure ends. When a user asks to unlock a car, the Car Manager will check the permission and possibly will unlock the car using the Car Communicator module.
  It also manages all the activities related to a race coming from the App-Server (like an issue reported from the application or an operation by a technician) and from the Interface Server (all the information inserted in the touchscreen by the user).

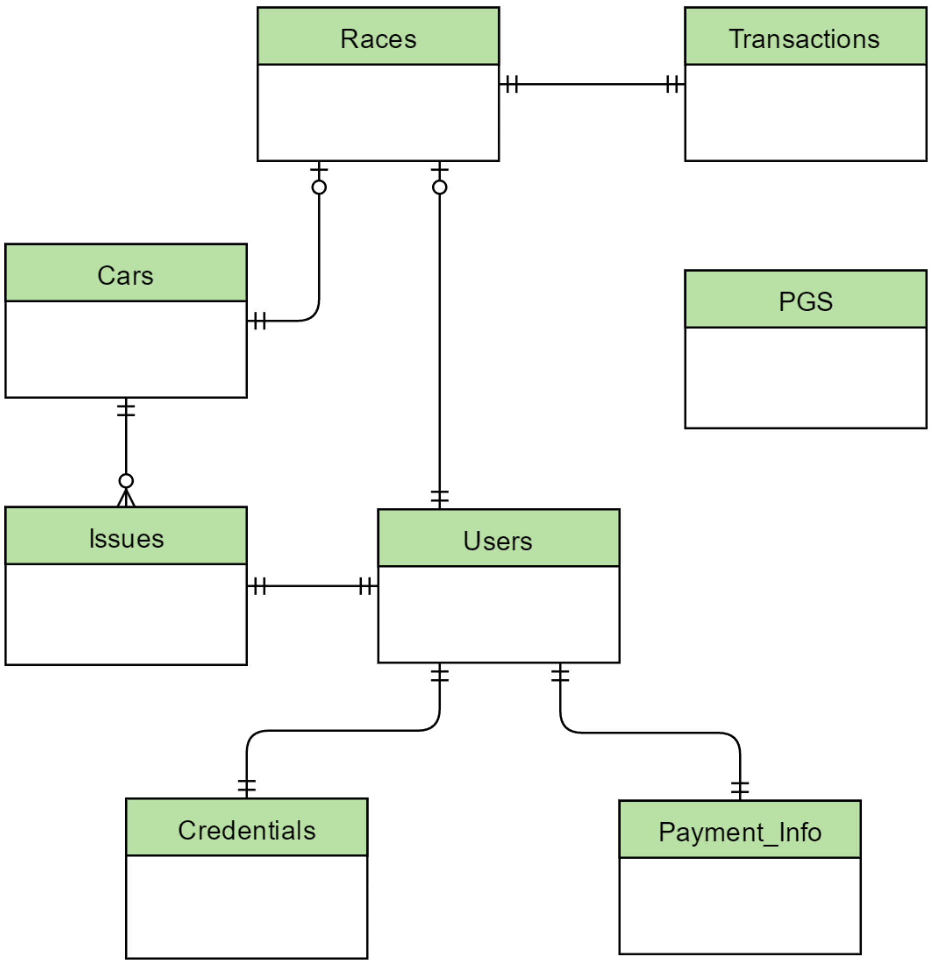
- **Car Communicator:** it is a client that sends all the requests that come from the central system. An example is the unlocking communication.

- **MSO Calculator:** is the module in charge of calculate the best station to park. To make that, it can check the database for retrieve the PGS information. The destination address is passed by the Race Manager that receives it from the OnBoard Unit during the starting of the drive.

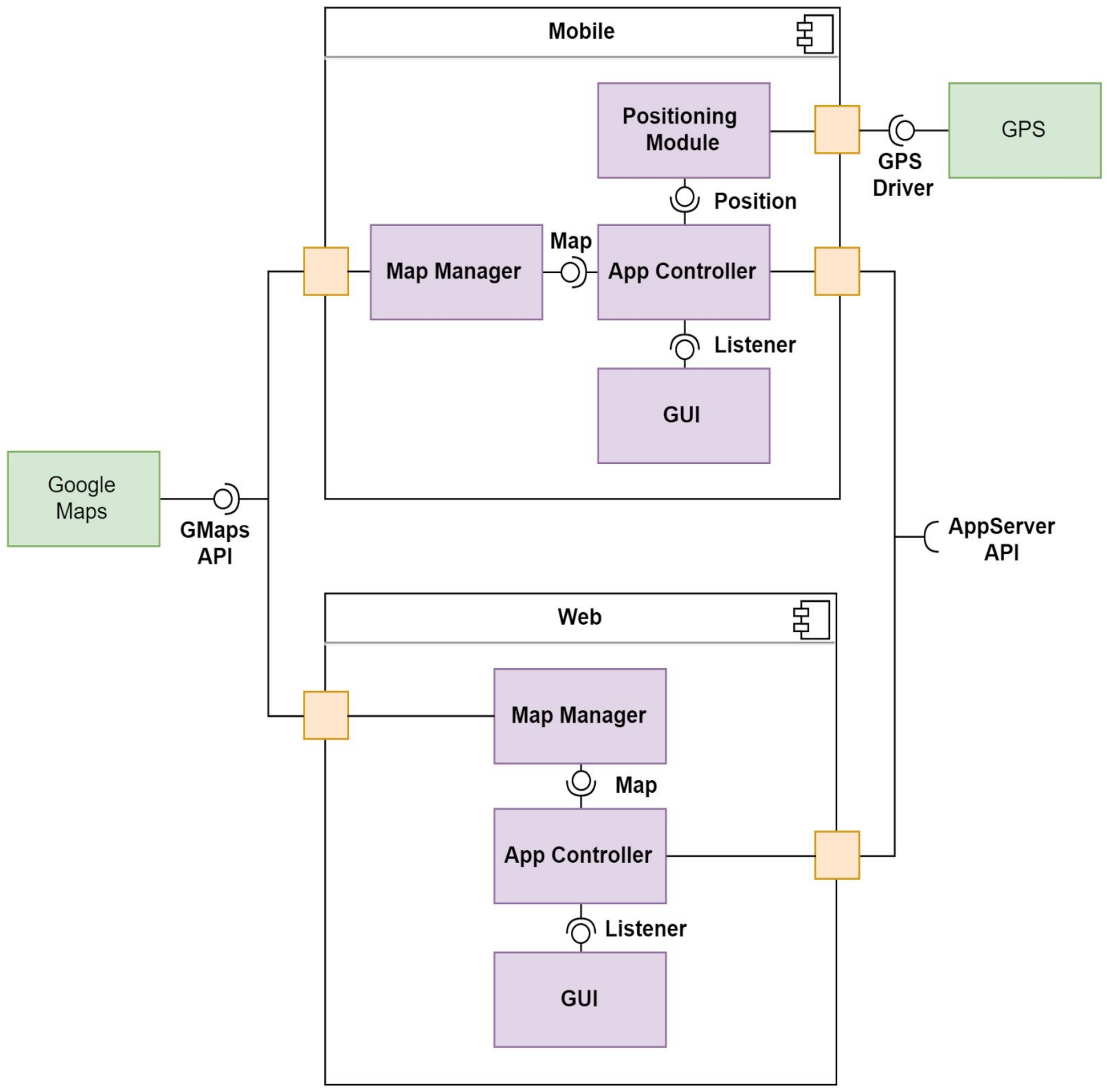
- **DB Communicator Manager:** is the only component that can directly contact the database. It receives requests from other components and then it submits the related queries.

2.2.4 Database

As said before, the chosen DBMS is MySQL. The structure of the database is very simple and is represented in the following diagram.



2.2.5 User Applications



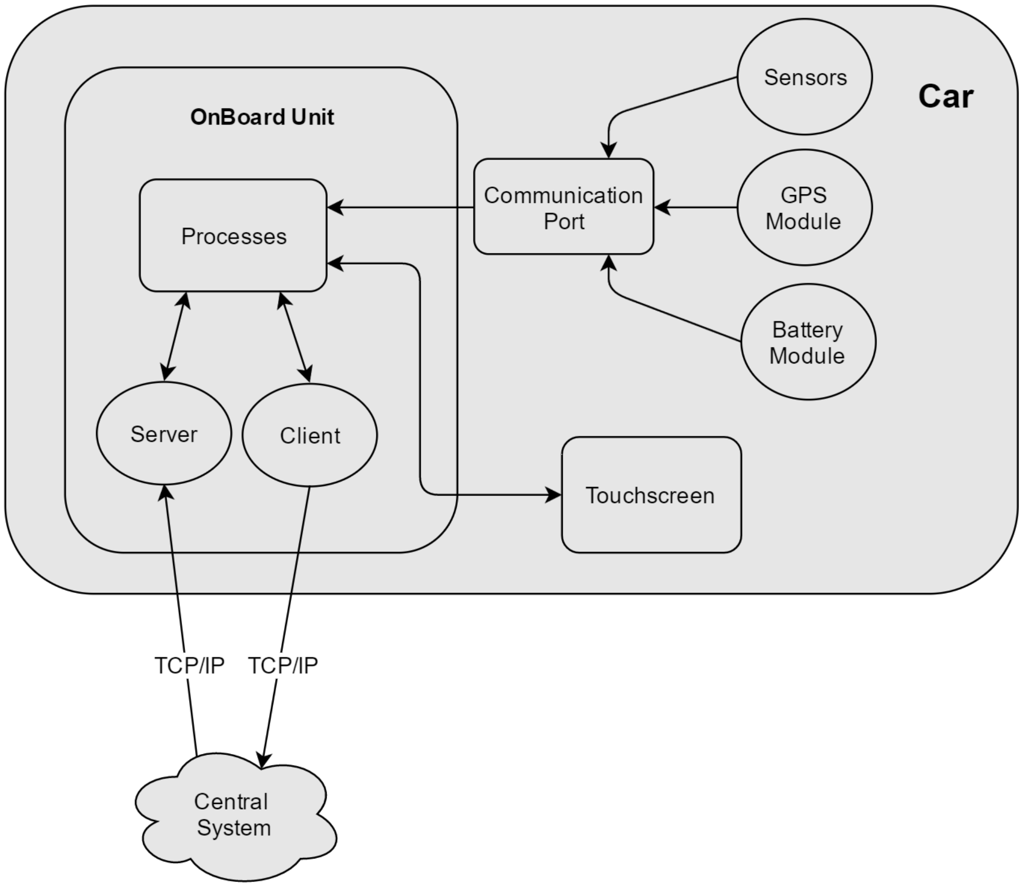
- **Application Controller:** it manages the interactions between component. It represents the business layer of the application.

- **Map Manager:** it interacts with Google Maps, using the related interface, for all the operation that will display a map on the screen. In the mobile application is also needed during the navigation of the user towards the car.

- **GUI:** represents the application graphical user interface.

- **Positioning Module:** it acquires the user position using the mobile GPS module.

2.2.6 Car



The core system of the car is an embedded software called OnBoard Unit. It is composed of 3 subsystems:

- **Processes**: these are the background activities that handle all the signals coming from the communication port and manage the information exchanged with the touchscreen. These activities are independently executed from the inputs coming from the Central System, but they can communicate with it. For example, when the process dedicated to control the battery level detects a change, it opens a client connection to the Central System in order to send the new status of the car.

- **Client***:* is a software charged to open a TCP connection with the central system.

- **Server***:* is a software always running and listening on a port able to handle TCP connections coming from the Central System.

Beside the OnBoard Unit, we have some hardware components. All these components must have a dedicated process always running in background.

- **Touchscreen**: is the user interface responsible for taking the inputs from the user and to display the result of the computation.

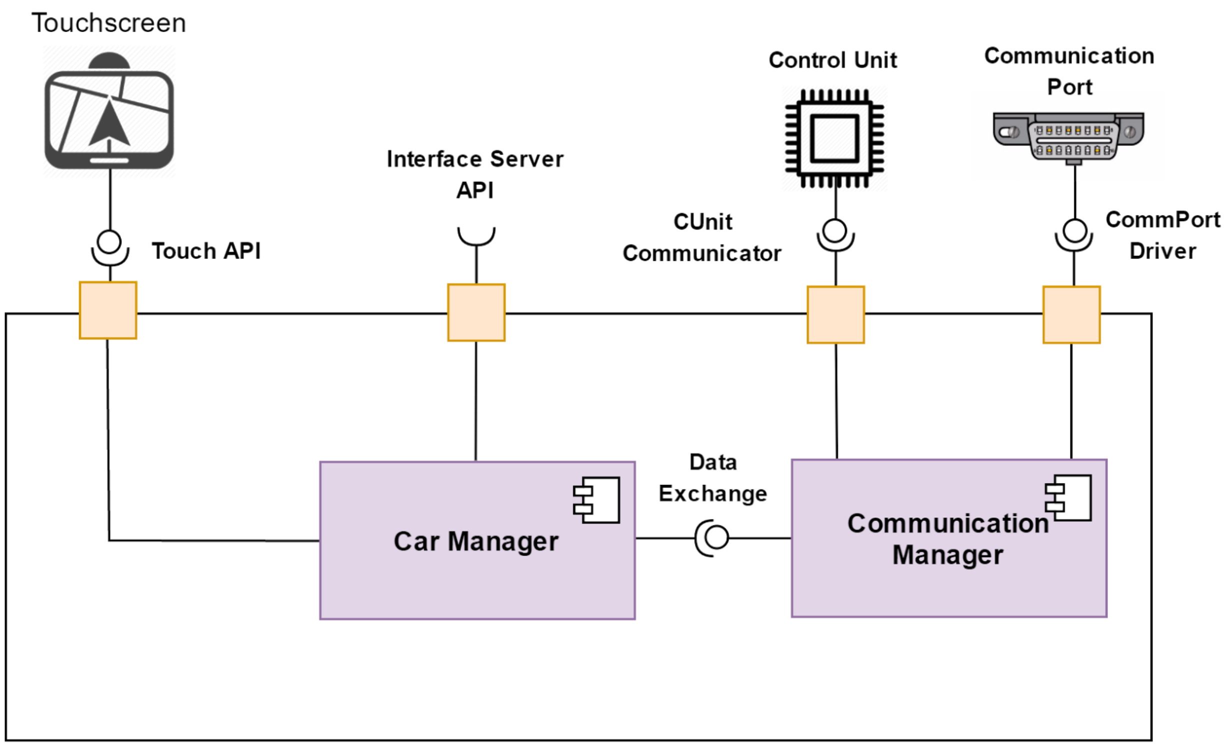
- **Battery Module**: is an interface that reports the current available battery level.

- **GPS Module**: is an interface that indicates in real time the coordinates of the car position.

- **Sensors**: send a binary signal in real time. The first four sensors indicate if the corresponding door is opened or closed. The last one indicates if the car is plugged into a Power Plug.

We also assume that the OnBoard Unit is able to interact with the mechanical part of the car as the engine and the lock of the door. It must be able to allow or forbid the starting of the engine, to turn it off and also to allow or forbid the opening of the doors.

Component Diagram



- **Car Manager**: manages all the car processes and the communications with the central system.

- **Communication Manager**: is the component that is in charge to communicate with the car control unit and that receives all the signals coming from the sensors using the communication port. It collects this data and it sends it to the Car Manager.

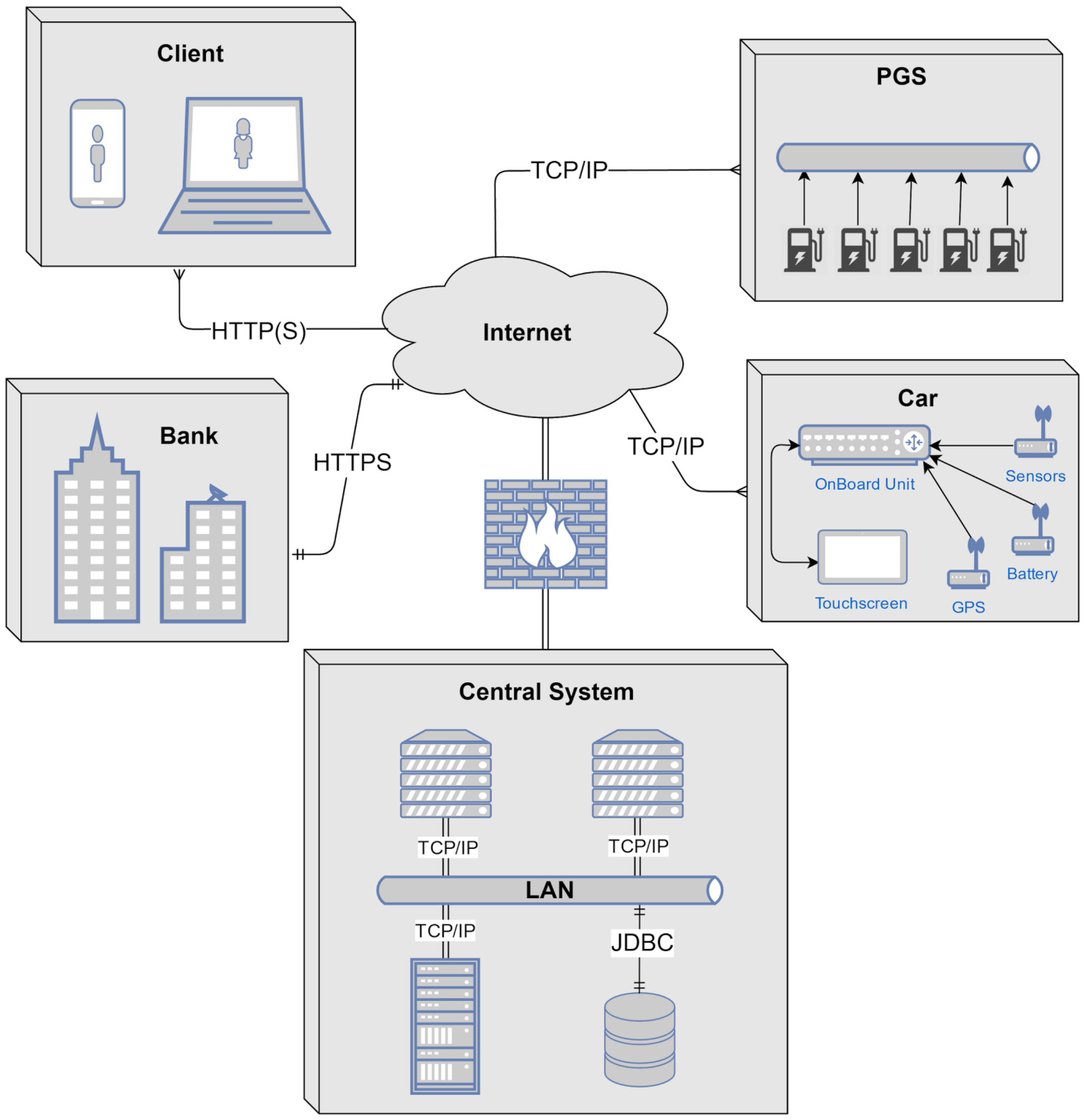
2.2.7 Power Grid Station

Power grid stations are managed by a simple software that continuously check the availability of all the power plugs of the station. When this number change, due to a new car connection, the software sends the information to the central system.
We decided to model the PGS like clients that sends requests to the dedicated server in the central system. In this way we have a single server, that is the more complex part, and several simple clients instead of having an opposite communication with one client, on the central system, and several servers on the PGS.
We assume that the whole PGS (even with the software module) is already built and is seen by the central system as an external interface.

2.2.8 Bank

For the central system, the bank is a server that provides payments services. BackOffice software acts like a client sending payment request to the bank.

2.3 Deployment View

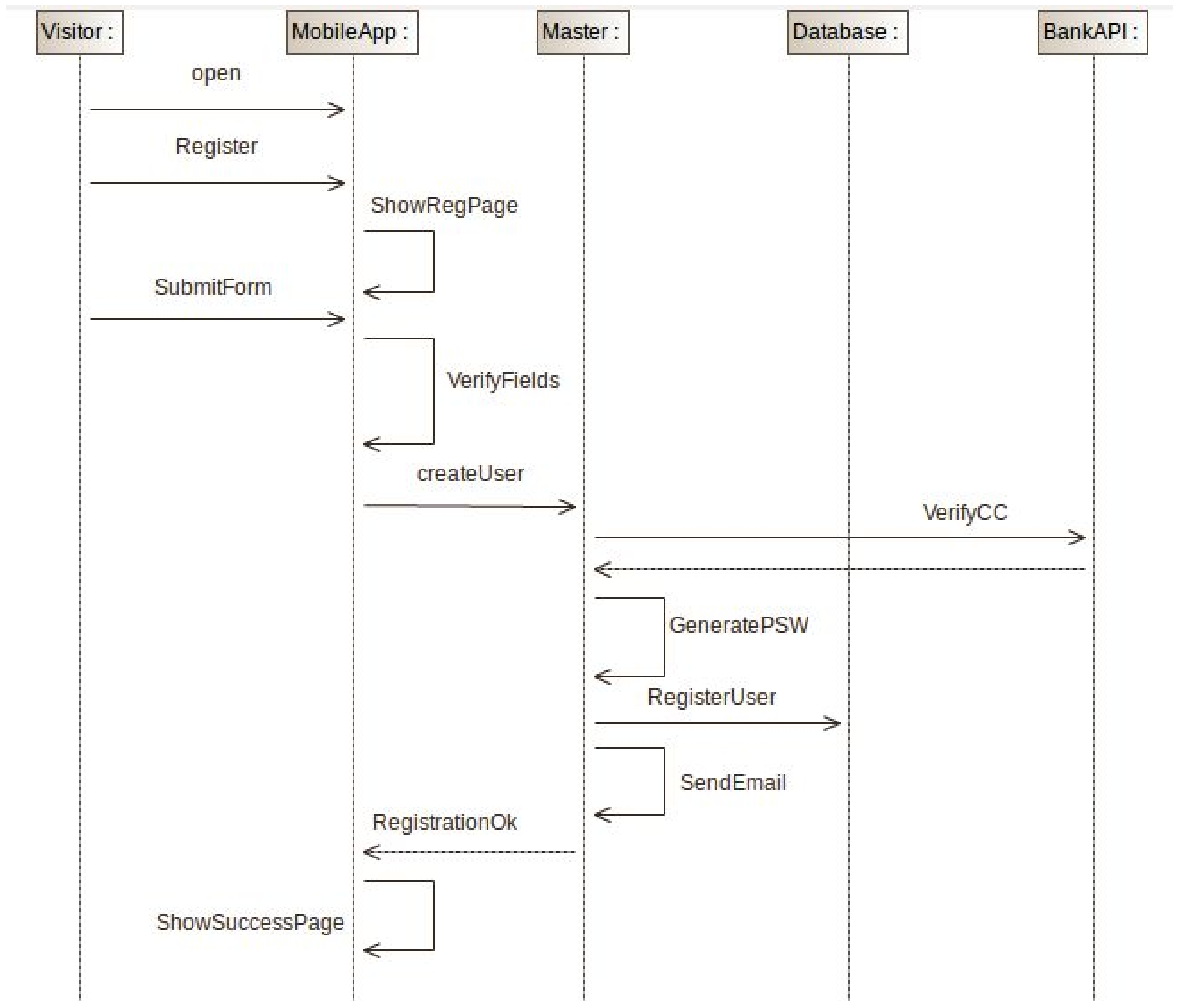


We decided to use a firewall to separate the Central System (LAN) from the internet (WAN) to ensure the security of the system.

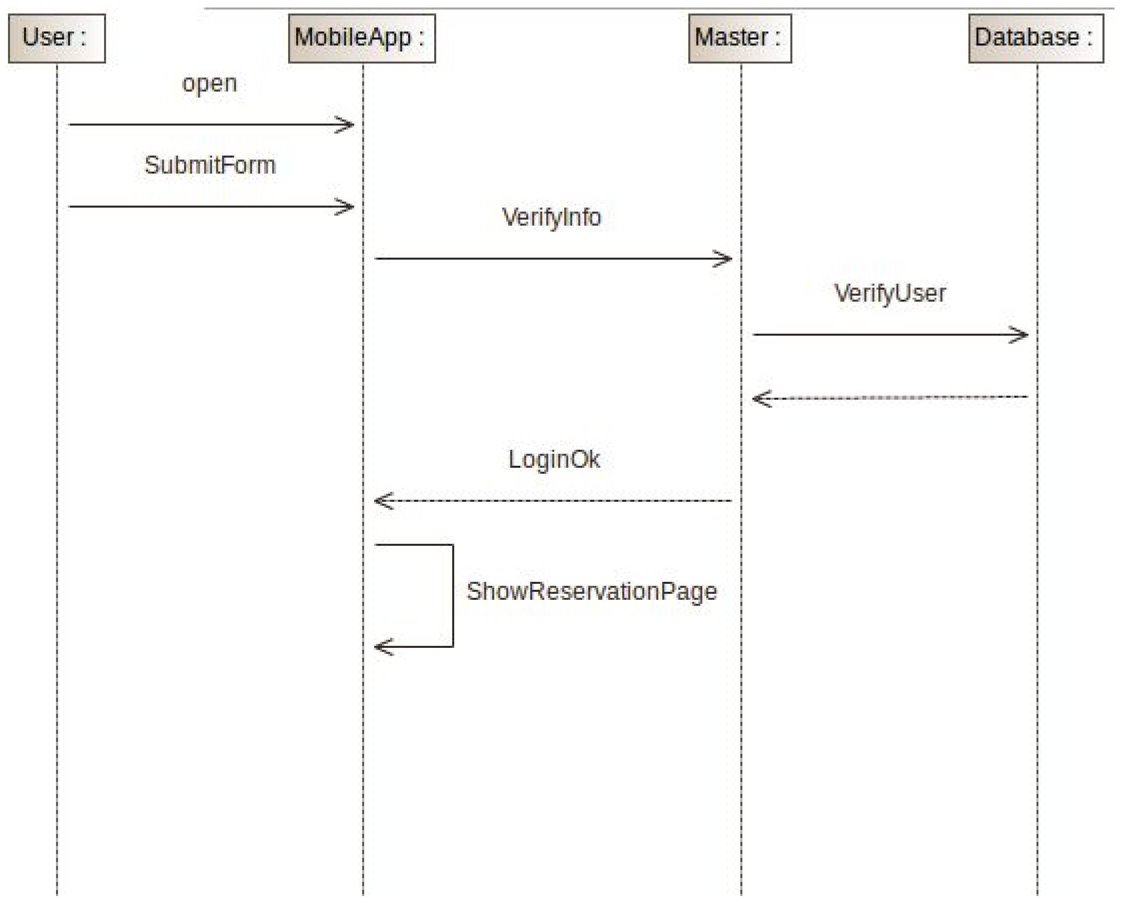
2.4 Runtime View

This section is intended to extend the use cases described in the RASD document.
The sequence diagrams below describe how the different components of the systems, viewed as the first level of abstraction, reacts to the input of the actors.

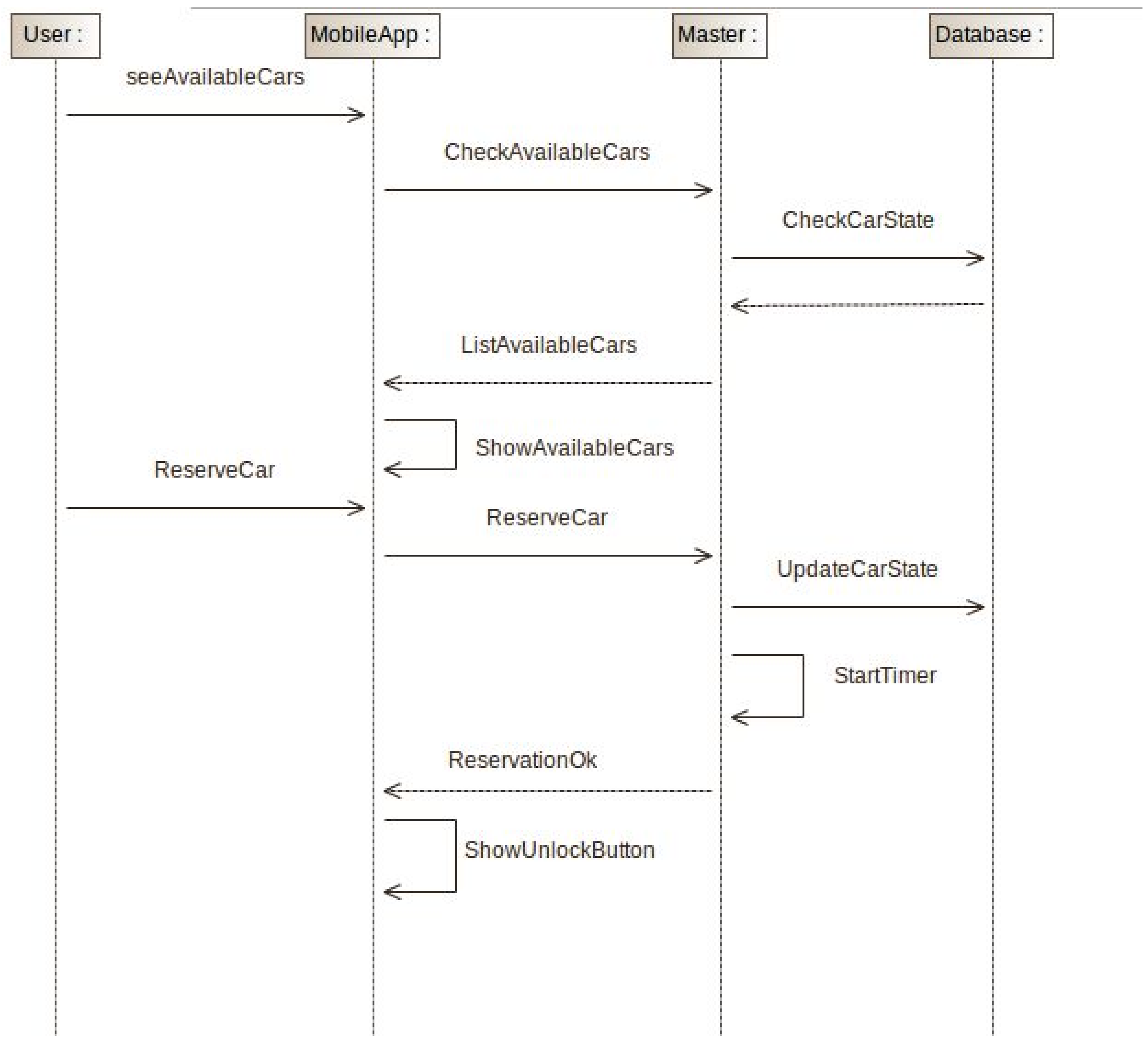
2.4.1 Registering to the system



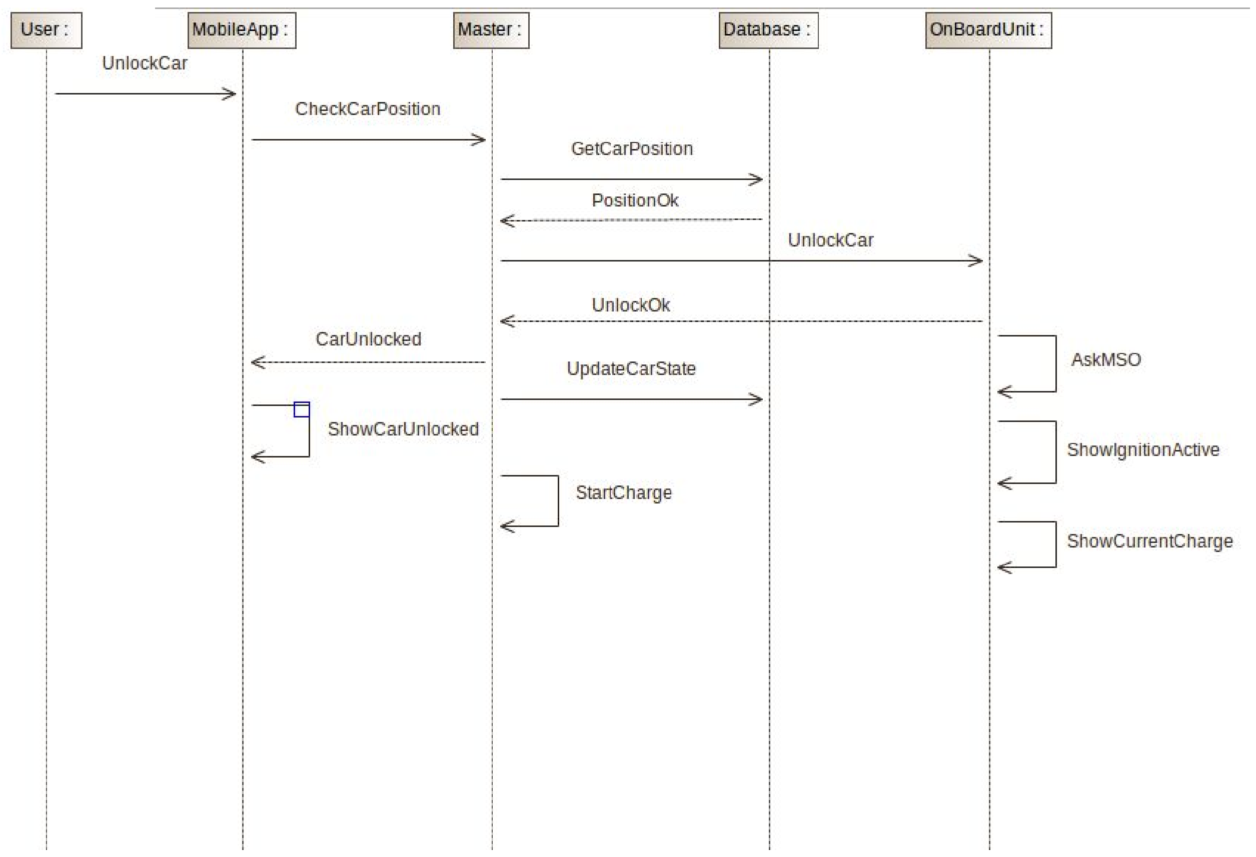
2.4.2 Login


User :
MobileApp :
Master :
Database :
open
SubmitForm
VerifyInfo
VerifyUser
LoginOk
ShowReservationPage

2.4.3 Reserving a Car

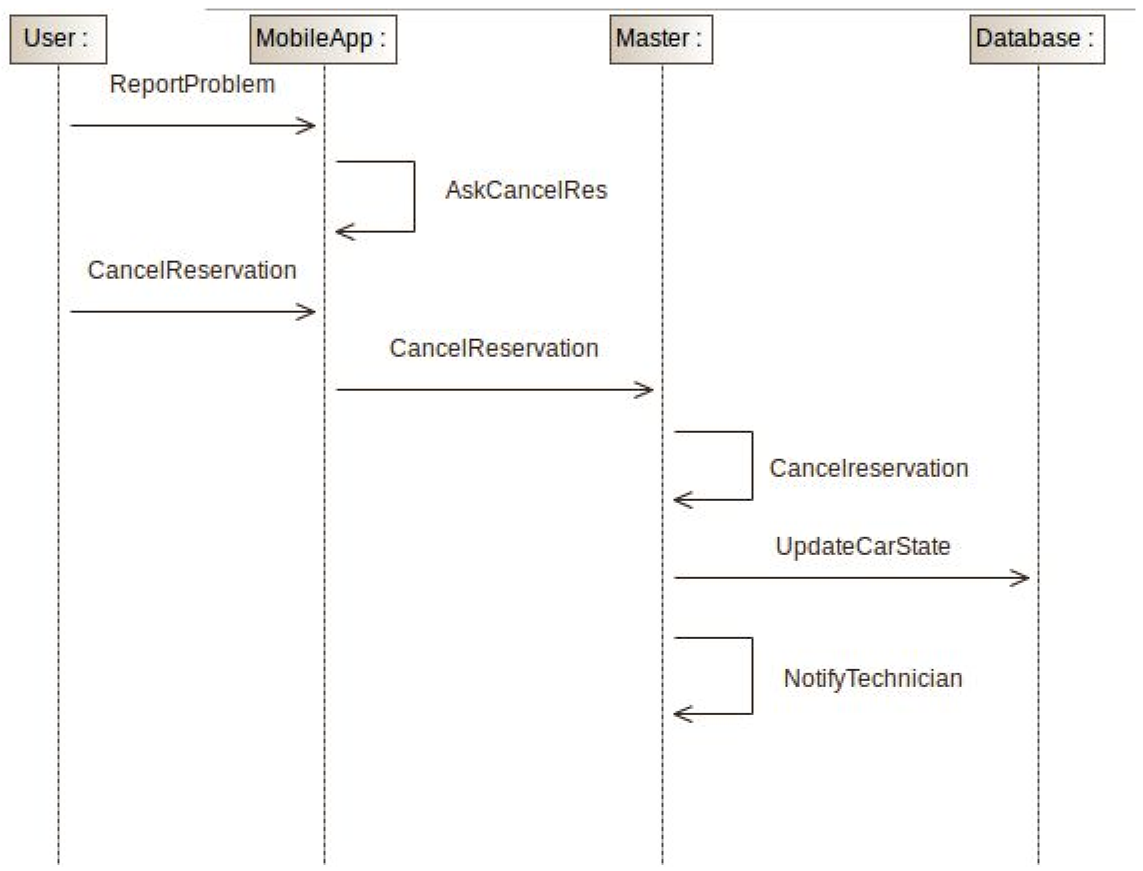


2.4.4 Starting a Race

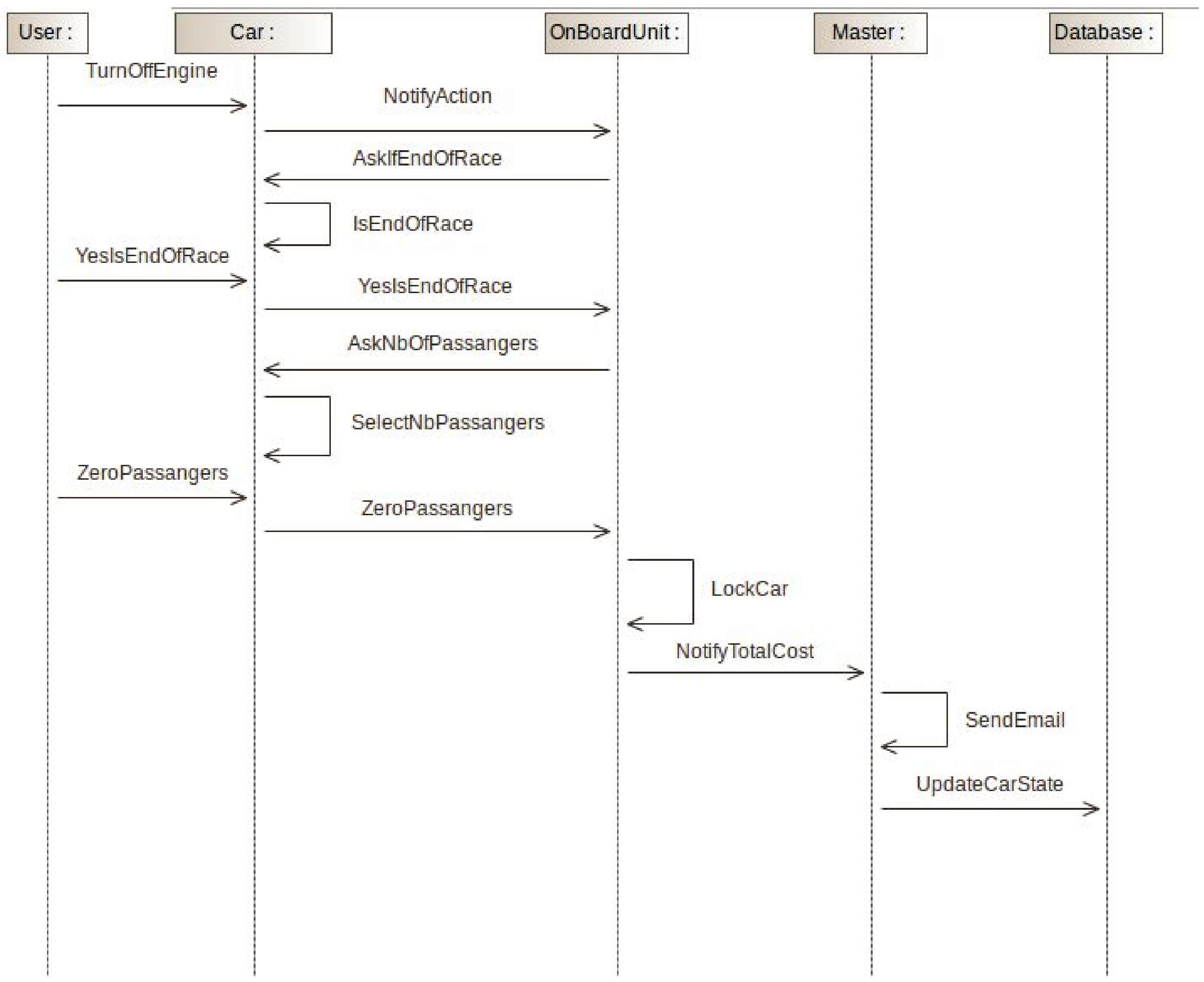


2.4.5 Reporting an Issue

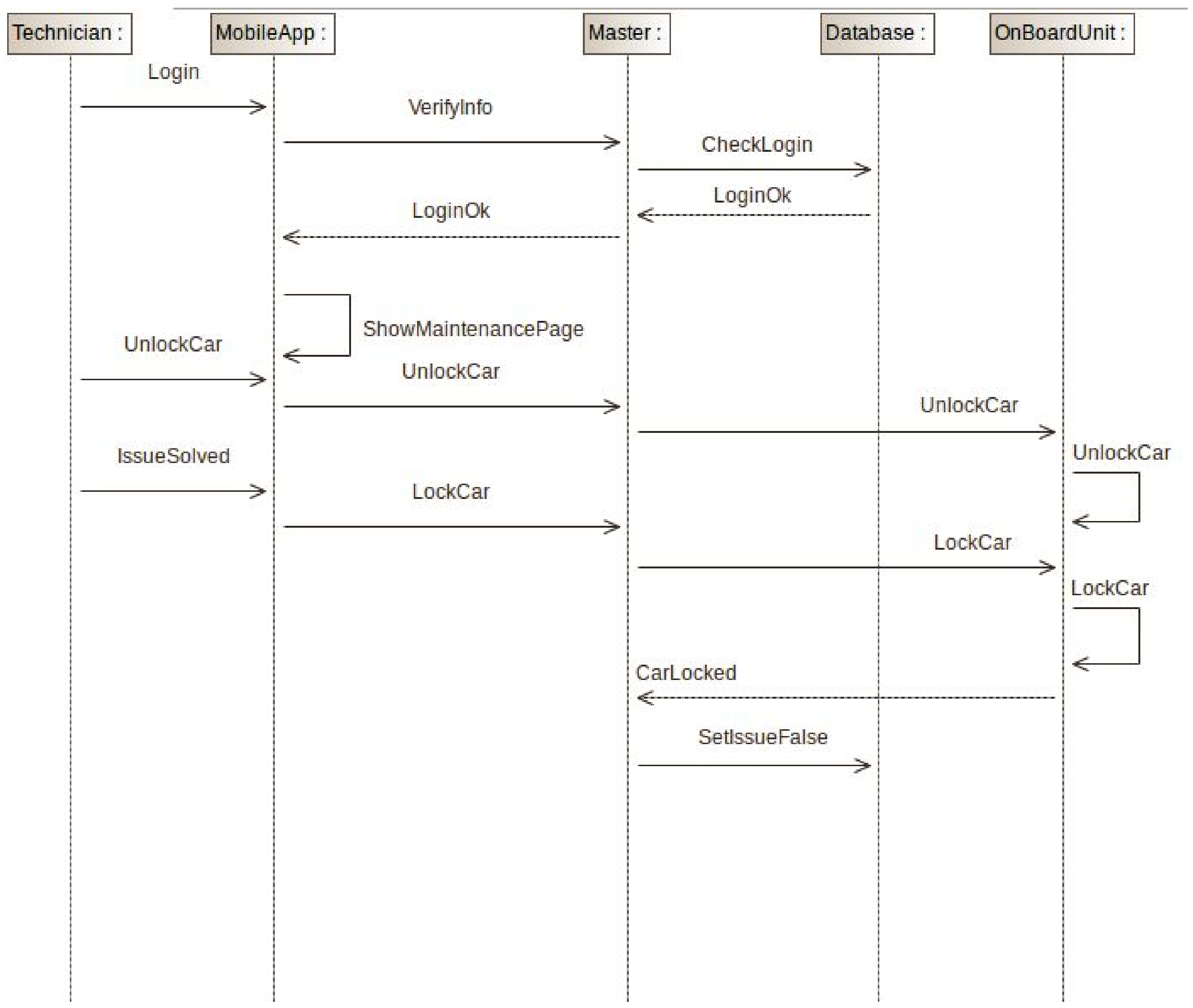
Before starting the race:



During the race:



2.4.6 Solve an Issue



2.5 Component Interfaces

2.5.1 Mobile/Web Application

- **Position**: provides the methods that allow the App Controller to know the current user's position.
- **Map**: provides the methods used by the App Controller to interact with the Map Manager.
- **Listener**: allows the App Controller to use the GUI.
- **GMap API**: is the API offered by Google that allows to use Google Maps services. There are different but similar APIs for Android, iOS and web service.
- **GPS Driver**: allows the Positioning Module to get position coordinates from the GPS.

2.5.2 App Server

- **Car**: allows the Communication Manager to interact with the Car Event Manager.
- **Authenticator**: provides a set of methods used by the Authentication Manager.
- **Email**: allows the Authentication Manager to send emails.
- **App Server API**: is the interface that the App Server offers to the applications in order to communicate with it.

2.5.3 BackOffice

- **Bank API**: provides the methods used from the central system to communicate with the bank. It is an API offered by the bank software that allows the system to ask for payments and to verify the information of a credit card.
- **Car Communication**: allows the Car Manager to communicate with Car Communicator in order to send information to the car.
- **MSO**: allows the car manager to access methods of the MSO Calculator.
- **BO Connector**: is the interface that the BackOffice offers to allow apps and interface servers to send information to it.

2.5.4 Interface Server

- **Information**: provides methods to the Communication Manager for access the Information Manager
- **Interface API**: is the interface that the Interface Server offers to allow external component (OnBoard Unit and PGS) to send information to it.

2.5.5 Car Interfaces

- **Car Connector:** is the interface that the Car offers to allow the Central System to send requests.

2.5.6 Shared Interfaces

- **DB communicator**: provides methods in order to communicate with the DB Communication Manager. It is present in App-server, Interface-server and BackOffice and it is used by all the components that have the permission to access the database.
- **JDBC**: is a driver used to query the database. This interface is independent from the choice of the DBMS that can be changed without any consequence for the rest of the components.

2.6 Selected Architectural Styles and Patterns

Some architectural styles and patterns we choose are:

2.6.1 Client-Server

The network connections of our application are based on a client-server model. The users act as clients when reserving a car, unlocking the car. These requests from the users are handled by the application server and forwarded to the back office to interact with the cars. In the payment system and car management, the system acts as a client and sends requests to the bank and car. This way the system is centralized within each tier making the system more secure. Data synchronization is easier and it is practical.

2.6.2 Four Tier Structure

Our application is divided into four tiers: Client, Server, Business, Data. Tiered architecture makes the design modular and easier to use. The data management can be done independently from the physical storage and visual updates can be done with ease. The system is also more secure since each tier is independent.

2.7 Other Design Decisions

2.7.1 Open Source Software

We have chosen to develop the system using open source software for several reasons. Firstly because is more flexible and customizable. It is very stable and free. This will minimize the costs for software. For all these reasons, choosing the software for the central system, we decided to use a LAMP architecture that is based on free software like Linux, Apache, MySQL and PHP.

2.7.2 Programming Languages

Java is the main programming language of the system, used for the OnBoard Unit, the Android Mobile-App and the BackOffice.
We chose it for some reasons:

- It is portable, so a module can be moved for future modification of the system.
- Many services offer a Java API to communicate with it (GMaps API in our case).
- It has performance we require and it is simple to use.

For the iOS application we must necessarily use Swift programming language.

PHP is used for the application server.

2.7.3 Servers Division

We decided to have two different servers to avoid the use of a unique server for both users and system interfaces. This is to avoid possible complete interruptions of the service.
For example, in case of a failure of the app-server, due to a great number of user connections through applications, the system can still manage the connections with all the car in use. In this way, user application accesses are blocked but drivers can still use cars and interact with the system.

2.7.4 Mobile-App Division

We decided to have one single mobile application for both user and technician. Technician application must have few functionality, so we decided to add it to the user mobile application exploiting the already existent component of it. In this way we don’t have to create a different application with similar component.
Technician can use the mobile-app logging in with dedicated credentials. The mobile-app detect if a user or a technician is logged and displays the proper interface.

2.7.5 Registration Process

Here is defined the complete procedure to perform the registration.

1. User inserts credentials and payments information.
2. The application controls the correctness of the format of the credit informations.
3. The user submits the request.
4. The App-Server writes the information on the database and send a request to the BackOffice for checking the validity of the credit card informations.
5. BackOffice checks the validity connecting to the bank.
6. When information are verified, the App-Server creates and sends the registration email.

Chapter 3

Algorithm Design

This chapter shows the pseudo-code implementation of the most relevant algorithm of the system that have been described in the RASD Document [2]. The first is the MSO Algorithm that find the best station to park the car when the user ask for it at the beginning of the race. The second is the algorithm that calculates the total cost applying the possible discounts and extra fees to the cost of the race.

3.1 MSO Algorithm

```
/**
 * This function determines the best station where a user can park the car
 * @param  userDestPos = destination address
 * @return the chosen PGS
 */

Function bestStation ( userDestPos ){

    proximityList = list()          // list of PGS sorted by distance from
the user’s destination
    nearList = list()          // list of PGS near 1500m from user’s
destination
    copyList = PWSList

    while ( copyList is not empty ) {
        min = copyList[0]
        for ( p in copyList ) {
             if   the distance between userDestPos is less than min and the
number of
            available plugs is greater than one
      then min = p
    }
        Add to proximityList min
        Delete min from copylist
    }

    for ( p in proximityList) {
        if the distance between p and userDestPos is < 1500
    then add p to nearList
    }




    if ( nearList is empty ) return the first PWS of proximityList
    else {
            min = number of available plugs on the first PWS of proximityList
            index = 0
            for ( p in nearList ) {
                        if ( p.numberAvailablePlugs < min) {
                        min = numberAvailablePlugs of p
                        index = the position of p in nearList
                        }
             }
      return the PWS at the position ‘index’ in proximityList
    }
}
```

3.2 Discount and Extra Fee Algorithm

```
/**
 * This function determines the total cost of the race
 * @param  cost = cost of the race before any discount or extra fee
 * @param  car = car used for the race
 * @return final cost with discount or extra fee applied
 */
```

```
Function totalCost( cost, car ) {

      if ( car.passengersNumber > 1)      discount = 0.1
      if ( car.batteryLevel > 50 )  discount = 0.2
      if ( car.isPlugged )          discount = 0.3

      if ( car.isOutsideSafeArea )  extraFee = - 0.3
      if ( car.batterylevel < 20 )  extraFee = - 0.3

      return cost * ( 1 + (discount + extraFee) )

}
```

Chapter 4

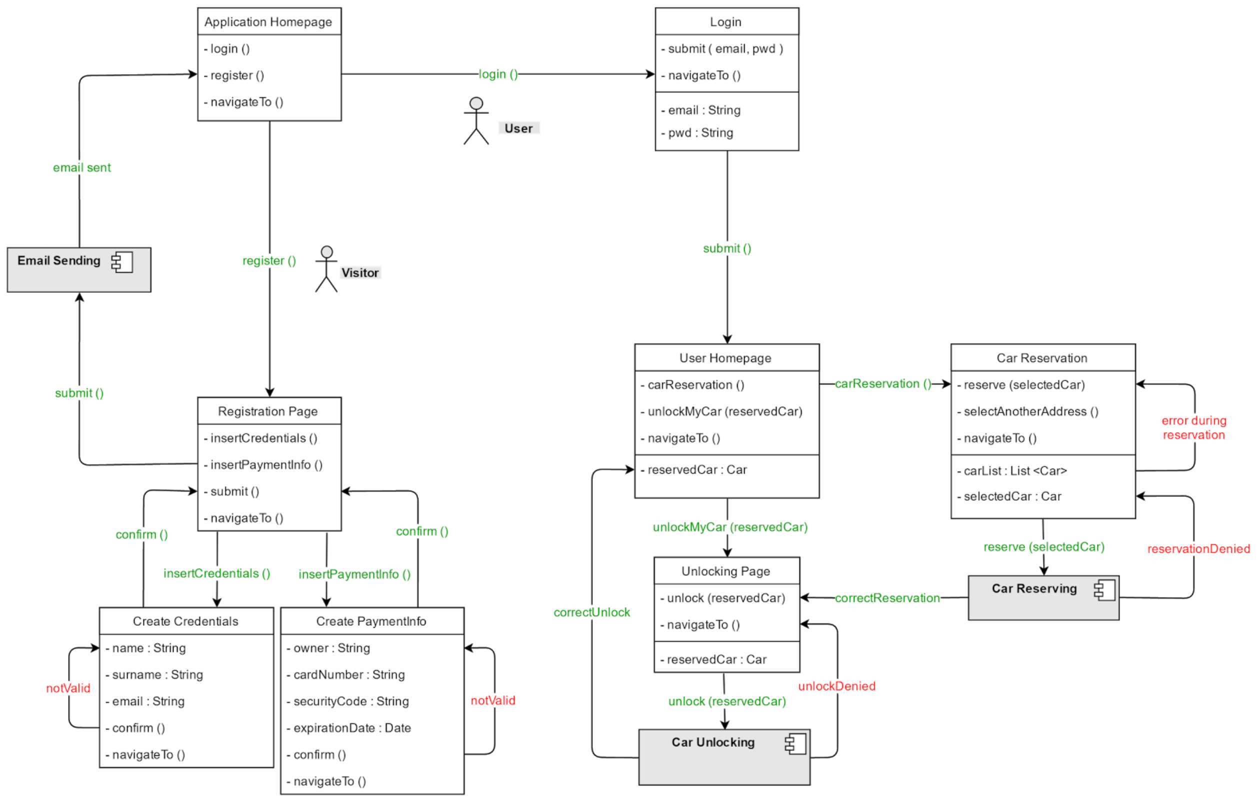
User Interface Design

Some mockups of the user interface are shown in the RASD Document [2].
Here is represented how the software part of the user interface must be structured.

4.1 User Interface UML

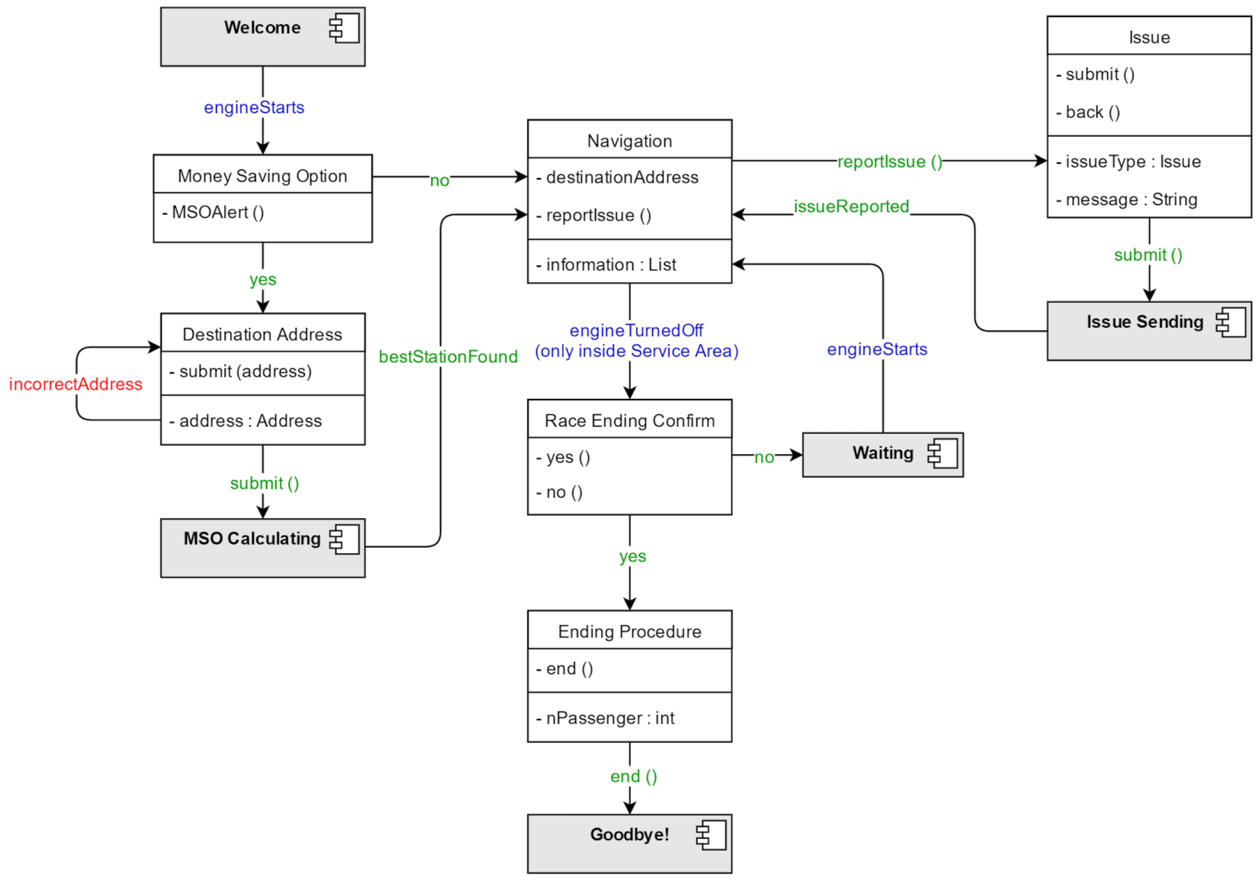
4.1.1 Application Design Interface

The following diagram represent the structure of the user interface of the mobile application. The browser application is similar, but without the possibility to unlock the reserved car.



4.1.2 Touchscreen Design Interface

Here is showed the touchscreen user interface.



Chapter 5

Requirements Traceability

In this section is shown which component of the system will satisfy all the goals and requirements listed in the RASD [2].

[G1] - Requirements R1, … , R4	**Components**
R1: A visitor, to register into the system, must provide valid credentials and payment information.	Authentication Manager
R2: The system must be able to check the validity of the visitor credentials and payment information.	Authentication Manager
R3: If the information received are correct, the system must reply to the visitor with an email containing a random generated password.	Authentication Manager Email Sender
R4: Registration is the only function that a visitor is allowed to use.	App Controller

[G2] - Requirements R5, … , R9	**Components**
R5: The system must always provide the login function to a not already logged user.	App Controller
R6: Only users can log into the system.	App Controller Authentication Manager
R7: A user can log into the system only if the email and password indicated are correct	Authentication Manager
R8: All the functionality of the system, except for registration and login, are provided only to logged users.	App Controller
R9: System must avoid multiple simultaneous login from a user.	Authentication Manager

[G3] - Requirements R10, R11	**Components**
R10: The system should allow the user to search for an available car sorted by the distance from his position.	App Controller Positioning Module
R11: The system should allow the user to search for an available car sorted by the distance from an indicated address.	App Controller

[G4] - Requirements R12, … , R18	**Components**
R12: The system should allow user to reserve only Available cars.	App Controller Car Event Manager
R13: A user can reserve only one car at a time.	App Controller Car Event Manager
R14: Only a user can reserve a car.	App Controller
R15: If the user doesn't unlock the car after one hour from the reservation, the car returns Available again.	Car Manager
R16: If the user doesn't unlock the car after one hour from the reservation, the system charges the user with 1€.	Car Manager Payment Manager
R17: The reservation must hold even if the battery level goes below 40%.	Car Manager
R18: User cannot cancel a reservation without reporting an issue.	Car Manager

[G5] - Requirements R19, … , R24	**Components**
R19: Reserved cars can be unlocked only by the user that made the reservation.	Car Manager
R20: A user can unlock only Reserved cars.	App Controller Car Manager

R21: The user should be able to indicate to the system that he reaches the reserved car and he wants to unlock it.	App Controller
R22: The system should unlock the reserved car only if the user asked for it.	App Controller Car Manager
R23: The system should unlock the reserved car only if the user is at most 20 meters far from it.	App Controller Car Manager
R24: After the user unlocks the car, the car state will change from Reserved to In-Use.	Car Manager

[G6] - Requirements R25, … , R27	**Components**
R25: As soon as the state of the car become In-Use, the counter starts.	Car Manager
R26: During the race, the system should notify to the user the cost of the race.	OnBoard Unit Touchscreen
R27: During the ending procedure, the system stops the counter as soon as the user exits the car, closes the door and all the other doors are closed.	OnBoard Unit Car Manager

[G7] - Requirements R28, … , R38	**Components**
R28: A user can start the ending procedure only if the car is located inside the service area.	OnBoard Unit
R29: Every time the engine is turned off inside the service area, the user must be able to start the ending procedure.	OnBoard Unit Touchscreen
R30: The ending procedure can only begin if the engine is turned off.	OnBoard Unit
R31: When the ending procedure is started, the engine can't be turned on again.	OnBoard Unit
R32: During a race, if the battery level becomes less or equal than 5%, the system automatically stops the engine and starts	OnBoard Unit

the ending procedure if it is not already begun.	
R33: The race ends as soon as the ending procedure ends.	OnBoard Unit Car Manager
R34: When the race ends, the total cost of the race is charged to the user and the state of the car is changed.	Car Manger Payment Manager
R35: Between the end of a race and a new use of the car, the doors can be opened only from inside the car.	OnBoard Unit
R36: Between the end of a race and a new reservation of the car, if any door will be opened, the user of the last race will be charged for the unpaid time between the end of the last ride and the opening of the door.	OnBoard Unit Car Manager Payment Manager
R37: When a race ends, the system changes the state of the car into Maintenance if the level of battery is less than 40% or the issue flag is equal to true.	Car Manager
R38: When a race ends, if the level of battery grows up to 40% and the issue flag is equal to false, the system changes the car state into Available.	Car Manager

[G8] - Requirements R39, R40	**Components**
R39: During the ending procedure the system must ask to the user the number of passenger.	OnBoard Unit Touchscreen
R40: If the number of passenger inserted by the user is greater than 1, the system must provide a 10% discount on the race.	Car Manager

[G9] - Requirements R41, … , R44	**Components**
R41: If the race ends and the charger is plugged, the system must provide a 30% discount on the race.	Car Manager
R42: If the car is not plugged into a power grid station and the level of the battery is greater than 50%, user gets a discount of 20% of the cost.	Car Manager
R43: If the car is not plugged into a power grid station the level	Car Manager

of the battery of the car is less than 20% user gets an extra fee of 30% of the cost.	
R44: If the car is left outside a safe area, user gets an extra fee of 30% of the cost of the race.	Car Manager

[G10] - Requirements R45, … , R49	Components
R45: The system must ask the user if he wishes to active the MSO or not before he turns on the engine for the first time.	OnBoard Unit Touchscreen
R46: If the MSO has been activated, the system should allow the user to select his destination through the touchscreen interface.	OnBoard Unit Touchscreen
R47: If the MSO has been activated, the system indicates to the user the best station to park selected by the MSO algorithm.	OnBoard Unit Touchscreen MSO Calculator
R48: The system applies the highest discount that user gets to the cost of the race.	Car Manager
R49: The system applies to the cost of the race at most one extra fee.	Car Manager

[G11] - Requirements R50, … , R53	Components
R50: The system allows the user to report an issue about the car he reserved or he is using.	App Controller Touchscreen
R51: If the user reports an issue when the car is reserved, he can choose to start the race or not and if he chooses not to start the race, the system changes the car state to maintenance and the user reservation expires without any costs.	App Controller Car Manager
R52: During the race, a user is able to report an issue and continue the ride.	Touchscreen OnBoard Unit
R53: A technician must have access to the cars on maintenance and must be able to solve the issues and set again the issue flag to false.	Technician App Car Event Manager

Chapter 6

Hours of work

Date	Description	Perini	Saini	Türkçapar
20/11/2016	Creation of the document structure	1h	1h	
23/11/2016	Team work	3h	3h	
24/11/2016	Diagrams / Algorithms	3h	1h	
26/11/2016	User Interface	1h		
28/11/2016	User Interface	3h		
01/12/16	Team Work	4h	4h	
05/12/16	Components	4h		
06/12/16	Components	2h	1h	
09/12/16	Team Work	6h	6h	6h
09/12/16	Component Diagrams			3h
10/12/16	Component Diagrams Fix			4h
11/12/16	Team Work	4h	4h	4h

Chapter 7

References

[1] PowerEnJoy specification document: *"ASSIGNMENTS AA 2016-2017.PDF"*.

[2] The related RASD: *"PowerEnJoy RASD"*.

[3] *IEEE Standard for Information Technology - Systems Design - Software Design Descriptions.*

[4] *IEEE Standard Systems and Software Engineering - Architecture Description.*