POLITECNICO
MILANO 1863

DESIGN DOCUMENT

v. 1.1 - Software Engineering 2 AA 2016-2017

Botta Daniel Enrico
806545

Bruschi Agnese
810762

Cano Adriana
812233

11/12/2016

Summary

1. Introduction

1.1 Purpose

The purpose of this document is to introduce and explain in detail the decisions taken regarding the Design of the PowerEnJoy project. Each decision made is associated with the reasons behind it. We will start first with the structure of our system, the layers we have chosen to organize it and then we will describe each one of them with all the components that they contain as well as how these components interact with one another.

1.2 Scope

The scope of the Design Document is to document the process of decision making regarding the design of the PowerEnJoy application. To be able to work on the application we need to establish beforehand the **architecture of the system** and the components that it contains. Each component is chosen with the intention to provide the functionalities that the system plans to offer to the users. Communication with external software is also very important, so first we will list the external components the system needs to deal with and then we will describe how the system will interact with them. Another key element to the system is the field operator, so will introduce their role in the system as well as the components of the system that handle said role.

1.3 Definitions, Acronyms, Abbreviations

- ✓ **UI**: User Interface
- ✓ **DD**: Design Document
- ✓ **RASD**: Requirement Analysis and Specification Document
- ✓ **OI**: Field Operator Interface
- ✓ **FO**: Field Operator
- ✓ **RDBMS**: Relational Database Management System
- ✓ **BCE**: Boundary Control Entity diagrams
- ✓ **UX:** User Experience diagram
- ✓ **ER**: Entity Relationship diagram
- ✓ **SD**: Sequence Diagram
- ✓ **API**: Application Performance Interface
- ✓ **Cross-platform application**: applications that run on different operating systems
- ✓ **OS**: Operative System
- ✓ **CSS:** Cascading Style Sheets
- ✓ **Cordova**: a framework where cross-platform applications can be written with
- ✓ **MVC**: Model View Controller
- ✓ **REST**: Representational state transfer
- ✓ **HTML**: HyperText Markup Language
- ✓ **ODBC**: Open DataBase Connectivity
- ✓ **GeoLocation:** identification of the real-world geographic location of an object

- ✓ **BatteryPriority:** points that will be assigned proportionally to the estimated battery life left. A low battery has a high priority.
- ✓ **NumberPriority:** points given proportionally with respect to the average number of PowerEnJoy cars in safe areas. A low number of cars corresponds to a high priority.
- ✓ **DistancePriority:** points assigned proportionally with respect to the distance between the safe area and the user destination. The smaller the distance, the higher the priority.
- ✓ **Priority:** the priority level is indicated by the value of the points of the three properties just listed. High priority equals to a high score.

1.4 Reference Documents

RASD version 2.0
AssignmentsAA 2016-2017.pdf
ITPN Version 1.0

1.5 Document Structure

Introduction: this section gives a brief introduction of the Design Document and explains what to expect during the reading of the document.

Architecture Design: this section contains many parts, ordered as follows:

1. ***Overview***: this subsection gives a textual description of the system and describes the main choices made.
2. ***High level components and their interaction***: this subsection introduces the components of the system through diagrams and textual description.
3. ***Component view***: this subsection goes into detail describing the components of the system, their subcomponents, what they do and how they interact with one another. In this subsection we use diagrams as well as textual description to present the view of the components.
4. ***Deployment view***: this subsection shows the components that needs to be deployed in order for the application to function.
5. ***Runtime view***: gives a view of the system at runtime.
6. ***Component interface***: shows the interfaces of the system and the components that use them.
7. ***Selected architectural styles and patterns***: in this subsection we list the implementation decisions made regarding the application, the architecture chosen, the patterns that we use and the reasons behind the decisions made.

Algorithm Design: in this section a description of the most important details of the system is introduced, showing how the system will handle them once implemented.

Management of persistent data: in this section we describe in detail how the database of the system is created, the thoughts behind it, and how it fits with the overall architecture. The main entities which will be recorded and that are essential for the well-functioning of the system are presented.

User Interface Design: in this section the mockups of the application are shown, enriching the set of mockups already presented in RASD. Both the mockups for the user and the FO are presented.

Requirements Traceability: in this section we will explain how the components introduced in the DD will be able to cover the functional requirements introduced in RASD.

Effort Spent: in this section we list the hours spent by each member of the group working on the project as well as what each one of the member worked on.

2. Architectural Design

Our system has a multi-tier architecture, composed by exactly three tiers: the presentation-tier, the business-tier and the data-tier. The **first tier** is the one representing the client and the part of our application that involves him/her as well as the part of the application regarding the field operator. It consists of what the client sees when he/she looks at the application and what the field operator sees. The second tier, the **business-tier**, is the one that holds the business logic of our system. It contains the logic behind the application, how things are managed, how requests coming from clients and operators are elaborated and also the communication with the database in order to respond to these requests. The third tier, the **data-tier**, is the one that is in charge of the database management. The application-tier communicates with this tier to get the information it needs for the requests coming from clients. It also handles the communication with the external services: Payment Gateway.

2.1 Overview: *High level components and their interaction*

In the following section we will describe in detail the components of the system. We have our three main components, the **User**, the **Application** and the **Database** component. For each one of them we will go into detail, introduce and describe the subcomponents that they contain to define thoroughly the architecture of the system.

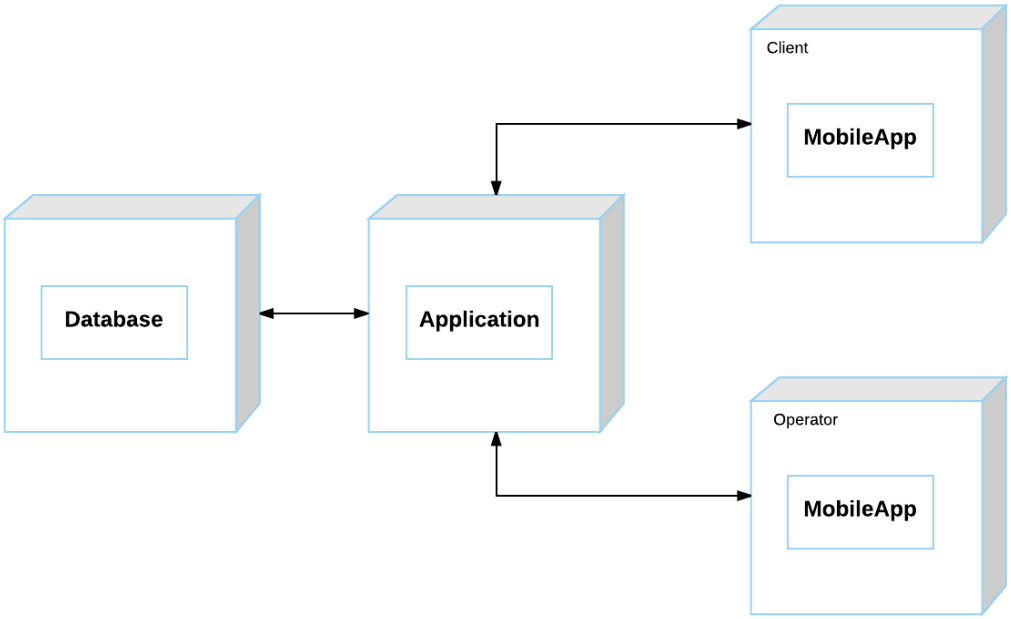


Figure 1: High level view components

In this simple diagram we can see the interaction of the components of the system with each other on a high level. We have the mobile client and operator that interact with the application, able to profit from the services offered by PowerEnJoy or to do the required work, in the case of the operator.

The operator is an employee of PowerEnJoy, responsible to handle the issues that might arise with the cars. He/she will also have access to the application, having a slightly different interface from the client. The central component in the diagram is the application, which is a mobile application, or

to better say it, the central part is the business logic behind the application. Apart from communicating with the clients and the operators, the application is also responsible for the communication with the database: it retrieves or inserts the desired information. The application does also communicate with external services.

2.1.1 System technologies

In order to build the different components of our 3-tier architecture we need different tools. In this section we will specify the technologies we have chosen to use for our system.

The application we are going to create is a mobile application and we intend to create a cross-platform application, for iPhone users as well as Android and Windows. To be able to do that we are going to use the framework called Cordova. In choosing the framework we were mainly influenced by the fact that the application will be a cross-platform application and Cordova allows us to achieve this goal. In addition, Cordova provides a set of JavaScript API's that connect with the device's native functionalities such as the GeoLocation, which is crucial for our application, as well as the camera, compass and contacts, which we think will be useful for future releases. Therefore, to implement the code needed in order to create the application we will use languages such as HTML5, JavaScript and CSS, which are all supported by Cordova. To handle our database, which is a RDBMS, we plan on using MySQL as a tool to deal with the organization of the database of our system.

The protocols we used for the communication between the tiers are:

- RESTful API for the communication between the presentation tier and the business tier.
- Authentication and authorization are done through HTTP.
  It was chosen for the fact that it leads the system to a fast performance, reliability, and the ability to grow, by using reused components that can be managed and updated without affecting the system as a whole, even while it is running
- ODBC: this protocol is used to make more secure and easy the communication between the business tier and the data tier.

2.1.2 Overall design

In our three-tier architecture we want to cover all the aspects that the application intends to offer to the client. In this section we go into detail with respect to what each one of them does.

Presentation-tier: this tier is responsible for the interface with the client, being that the user of PowerEnJoy or a FO. It is responsible for collecting the requests sent from the client and sending them to the business tier as well as for displaying the resulting messages that follow such requests.

Business-tier: this tier is responsible for the management of all the actors of the system as well as all their interaction with one another. It handles the requests made by the clients, elaborates them and it also communicates with the database of the system through the Data tier. It is also responsible for the communication with the physical cars and the external system of E-mail gateway.

Data-tier: this tier is in charge of the database. It's the only component of the system with direct access to the database of the system, with the rights to store and retrieve information. It is also responsible for the communication with the external system of Payment Gateway.

So all in all we have three tiers:

- Presentation tier (Thin Client layer)
- Business-tier (Application layer)
- Data-tier (Data Access layer)

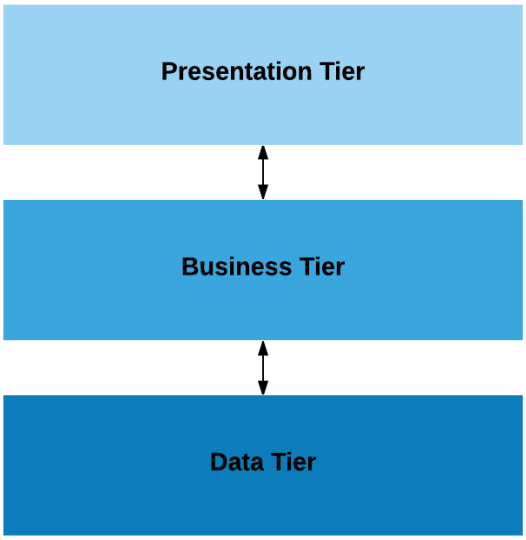


Figure 2: View of the architecture

2.1.3 Assumptions

Assumption	Motivation
An Android tablet is given to each field operator at the beginning of a shift.	Operators need to have internet access to be able to control the cars of the systems and the possible issues presented so that they can go and fix them.
At the time of the shift the profile of the operator whose shift is at the moment is activated once the operator has inserted his employee-id and password.	The tablets that PowerEnJoy provides to its employees can be reused by all of them during their shifts. The activation of the profile only happens if the employee-id inserted is the one of the employee whose shift is scheduled at the time.

The cars have a software installed at production time that the system can communicate with.	Such system makes possible the communication of the Business tier with the car, to be more specific the Car Controller, with the physical cars in order to get the information it needs, such as location, seating sensors, locking state.

2.1.4 General constraints

Element	Requirement
User	It needs to have an OS on his/her phone that is either iOS, Android or Windows. Must also have at least 100MB of free storage space on his/her mobile phone or tablet to be able to support the application.
Database server	MySQL
Network	Internet access, HTTP protocol

2.2 Component View

Each tier consists of components that we will describe in detail in the following section.

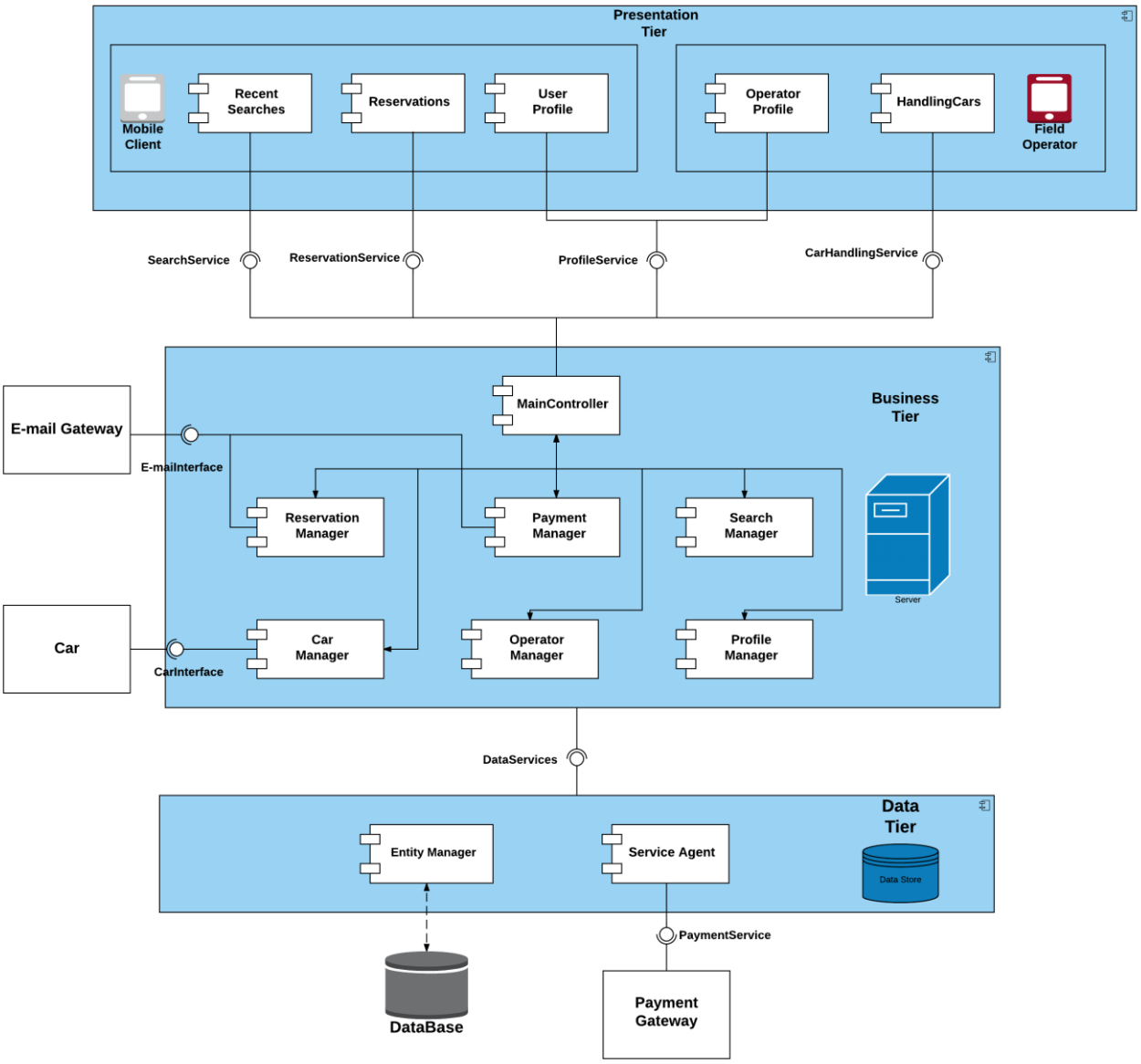


Figure 3: Component view

As it can be seen in the diagram, we have a **Main Controller** that handles the request coming from the clients and the operators once they are logged in the system. For every incoming request, the controller elaborates it and sends it to the right controller to handle it. Once the corresponding controller has generated a response, it sends it to the Main Controller and it's the Main Controller that sends the response back to the user.
For each layer we have:

The presentation tier components:

- User Profile: users have a profile where they can see their personal information inserted upon registration and where they have the possibility to change some of this information.
- Reservations: the user can see all reservation history as well as the ongoing reservation if there is one. Each one contains the date and timed of the reservation as well as the final fee paid (in the case of the current reservation, the fee is first 0 and will later on be updated).
- Recent Searches: the application gives the possibility to save the recent searches made, this way the user can simply click on them if desires to make the same search again.
- Operator profile: this component is related to the field operators and the profile each operator has in the system.
- HandlingCars: this component is the one that contains all the cars that the field operator is handling during his/her shift.

The business tier components:

- Main Controller: this is the controller responsible for routing the requests coming from clients to the right controller in the business tier. Once received the request from the client, it directs it to the controller responsible for handling it. After the controller elaborates it and generates a response, the main controller sends back the response to the client.
- Profile Manager: controls the profiles of all the users of the system. It is responsible for the process of checking the correctness of the information inserted upon registration or changed at some point later on. It is the one that access the user's information in the database through the data tier when the user signs-in (to check if the couple password-email inserted exists) and when he/she asks to see his/her own profile info.
- Reservation Manager: controls the reservations made through the application. It handles the changes of the states of the reservation and communicates with the Car Manager for the updates of the car. It also communicates with the Payment Manger once the user resigns the car after being done using it to finalize the fee that the user needs to pay. The Reservation Manager handles the communication with the E-mail gateway to communicate via e-mail with the client, sending information once the reservation is made.
- Search Manager: controls the searches made by the users. It is responsible for showing to each user that makes the search the available cars in the desired location. Communicates with the Car Manager to get the information about the cars.
- Car Manager: it handles the cars that PowerEnJoy offers to its clients. It does the communication with the physical cars and retrieves the information it needs or its asked by other components of the system. It is responsible for the updates of the states of the car as reservations happen.
- Operator Manager: it is in control of the management of the operators that the system has hired to deal with the various car issues that might arise. It communicates with the Car Manager to show to the operator the cars that might need fixing so that the operator can proceed to take care of the problems.

- **Payment manager:** it is in control of the payment after a reservation is done. It retrieves all the necessary information from the other components and it then communicates a third party to finalize the payment, the Payment Gateway.

The data tier components:

- **Entity manager:** it is in control of the entities stored in the database. It handles the insertion and removal of data from the database as well as information retrieval when asked by other components. Grants access to the cars data only to the Car Manager.
- **Service agent:** this component is the one responsible for the communication with the external services (in our case, the Payment gateway, which is the external service that makes possible the payment of the fee through the credit card information the user has provided upon registration).
- **E-mail Gateway, Car** and **Payment Gateway**: this three elements are not components of the system, but they play a crucial role in making the system function as it should. The first one makes possible the **e-mail service** to the system so that e-mails can be sent to the users for reservations and payments (once the payment is done, a receipt is sent to the user via mail). The **car** is one of the most crucial elements of PowerEnJoy so it is going to play a big role in the architecture as well. It is the Car Manager that handles all the possible interactions with the physical cars. The Payment Gateway is a necessary element for the system to be able to **payments** to happen.

2.3 Deployment View

The following diagram describes the deployment view of the PowerEnjoy software.

The deployment diagram specifies a set of constructs that can be used to define the **execution architecture** of systems that represent the assignment of software **artifacts** to **nodes**. Nodes are connected through communication paths to create network systems of arbitrary complexity. Nodes are typically defined in a nested manner, and represent either hardware devices or software execution environments. Artifacts represent concrete elements in the physical world that are the result of a development process.

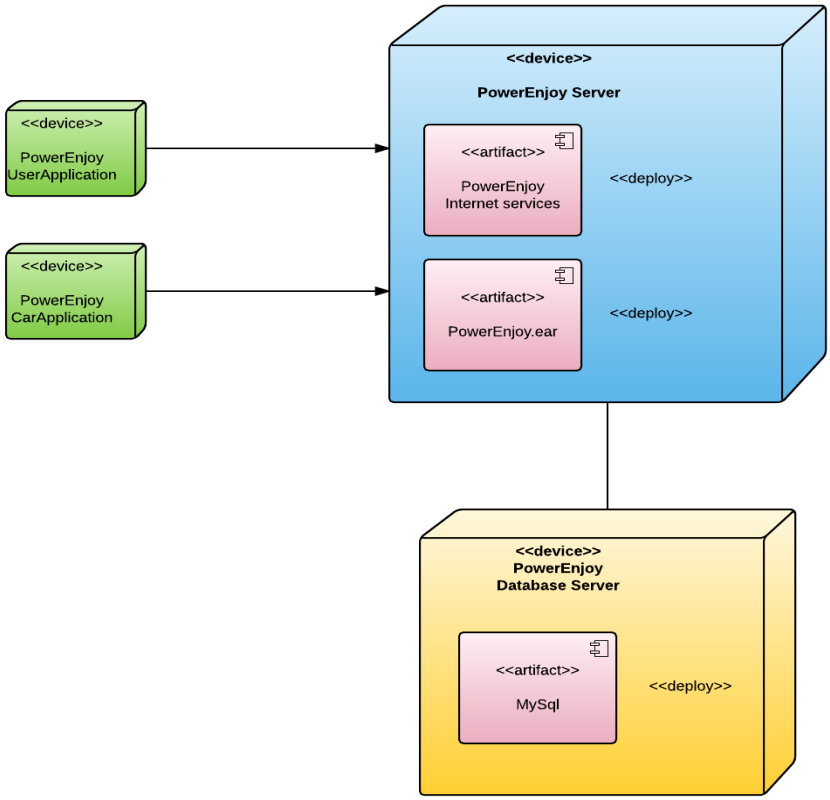


Figure 4: Deployment view

2.4 Runtime View

2.4.1 Runtime View diagram

The following diagram represents the runtime view of our system. The EAR file is a standard JAR file. It represents the models of the application. Within we can find all of the functionalities of the PowerEnjoy server. It makes the deployment of the various models onto the server happen simultaneously and coherently.

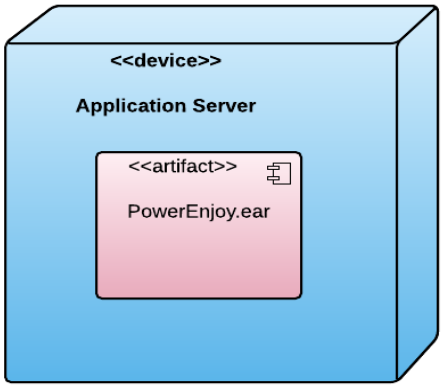


Figure 5: Runtime view

2.4.2 Sequence diagrams

To complete the representation of the system, we have also included Sequence Diagrams to better describe how the system elaborates request coming from the users.

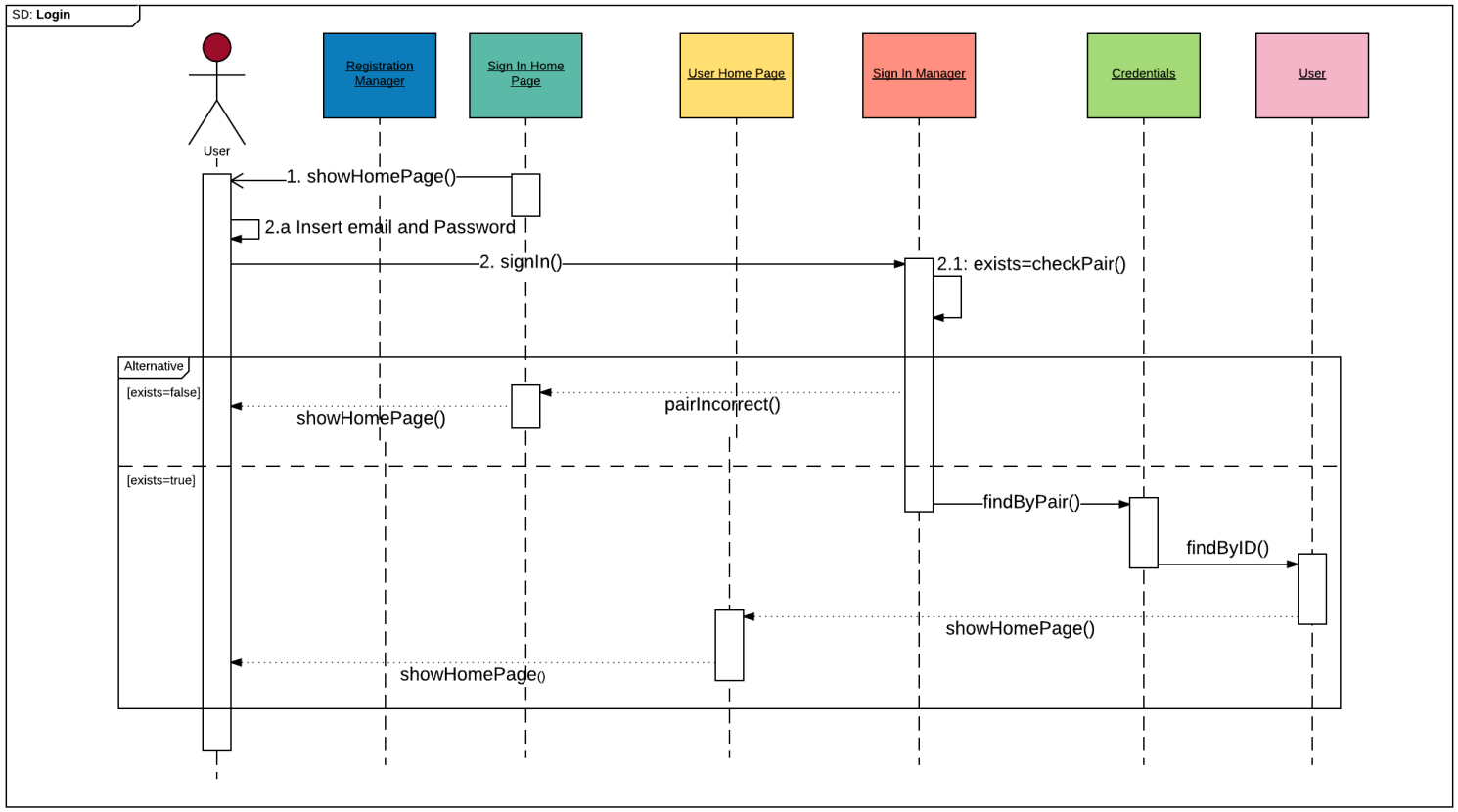


Figure 6: Sequence Diagram-Login

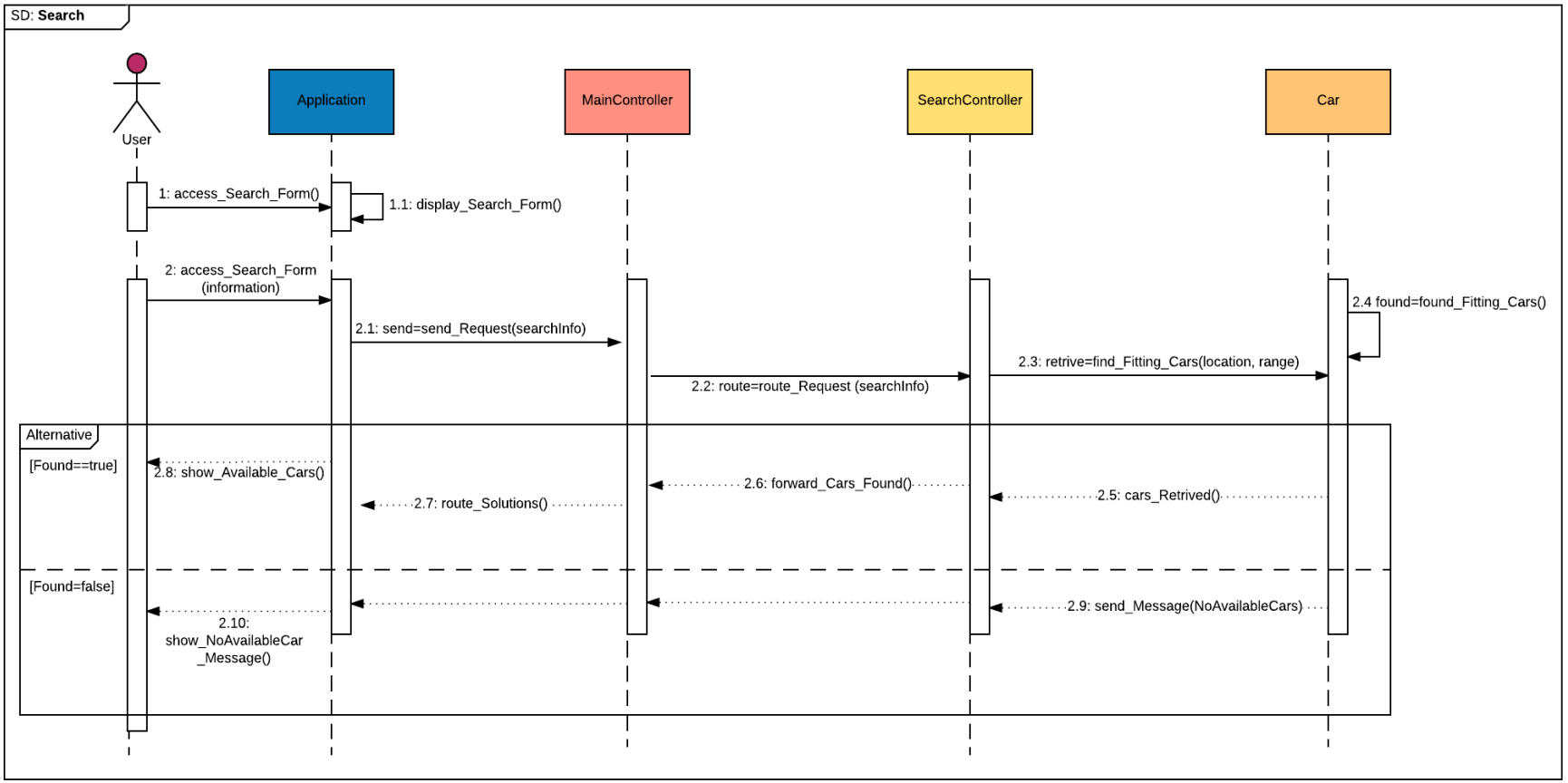


Figure 7: Sequence Diagram-Search

2.5 Component Interfaces

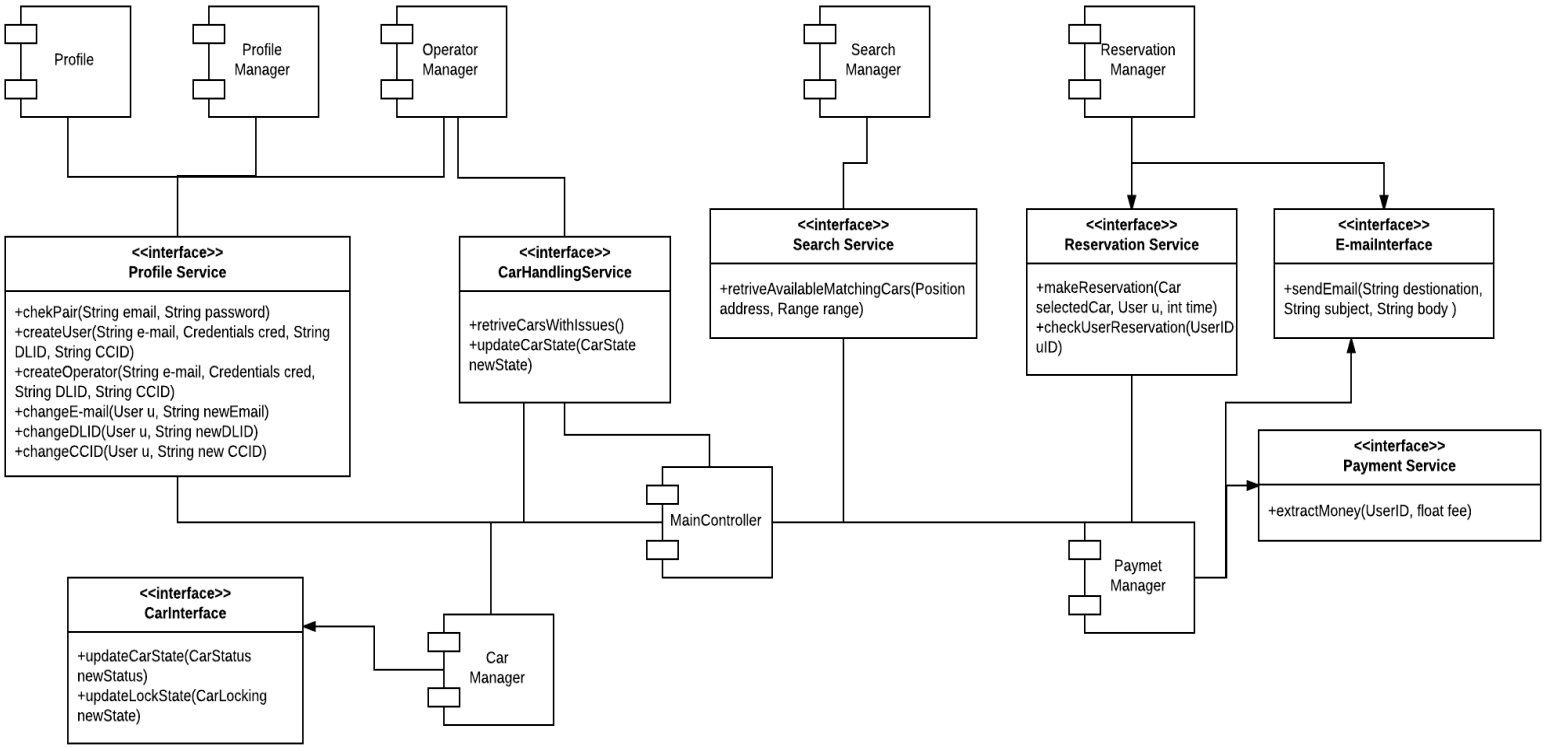


Figure 8: Interfaces of the system

2.6 Selected architectural styles and patterns

- MVC: we chose this design pattern to simplify the division between the components of the system in Model, View and Controller as the pattern indicates. It is also a pattern supported by the Cordova framework.

- Chain of responsibility: this pattern is used to make the work of the main controller and the other controllers as effective as possible.

- Façade: this pattern is chosen to handle the unification of the results of the different components in the business tier to one single message that will be later on sent to the client.

- Transactions: used in the data tier to handle the transactions of the data. By using this pattern, we prevent conflicts from happening when transactions make changes to the data in the Database.

3. Algorithm Design

A high level description of some algorithms will be made, to give an idea of how the system will implement features which we reckoned to be interesting. The algorithms we will describe are those of the most critical parts of the system.

We will describe the following algorithms:

- Search for a car
- Money Saving Option

3.1.1 Search for a Car: *Description*

The search for a car option must retrieve *a list of available cars within a certain range*. The user inserts in input the location of the origin and the radius of the circumference from where to search cars. There are 3 possible radiuses. The system then elaborates the input and retrieves the list of cars and informs the user.

3.1.2 Search for a Car: *Algorithm*

1. The user inserts the **origin** and **radius** of the circumference. (selects one of the 3 possible ranges).
2. The system gathers the input and starts the search in the area.
3. Every car the system finds is analyzed.
4. The system checks if the "Car_Status" of the Car is "AVAILABLE" and adds it to a list. If it's not available, it is ignored.
5. Once the list is complete it is sent to the user. He will be able to visualize it through the PowerEnJoy application.

3.2.1 Money Saving Option: *Description*

The money saving option has been decided to be implemented so that everyone has a better chance to enjoy to the fullest the services of PowerEnJoy. Depending on the final destination given by the user in input, the system will retrieve a station location of where to leave the car to get a discount. This station is determined to ensure a uniform distribution of cars in the city and depends both on the destination of the user and on the availability of power plugs at the selected station.

3.2.2 Money Saving Option: *Algorithm*

1. System receives in input the location where the user wants to park.
2. The Car will send the information on its current battery level.
3. The system elaborates the info, and retrieves a list of safe areas.
4. The list of safe areas that are further than 10 min walk from the destination are discarded.
5. An estimate of the battery life after the trip is calculated.

6. BatteryPriority points will be assigned proportionally to the estimated battery life quantity. For a low battery quantity, we have a high priority.
7. From the list of safe areas, an average of number of cars safe area is calculated.
8. Each safe area will get NumberPriority points proportional to the average number of PowerEnjoy cars. For a low number of cars, we have a high priority.
9. Each safe area from the list will receive DistancePriority points proportional to the distance between the safe area and the user destination. The smaller the distance the higher the priority.
10. The safe area with more priority points is chosen. The application first checks the BatteryPriority. If the values are similar, NumberPriority points are checked. If the values are similar, DistancePriority points are checked. If we have similar values in all three cases a random safe area will be selected.
11. The system informs the user through the PowerEnjoy application where this chosen safe area is at.

4. Management of persistent data

4.1 Database

The software chosen to administer the database is MySQL.

4.2 Security

The database used for our system will ensure security of the stored information:

- *integrity of data*: performing periodical logical backups.
- *verifiability*: record of all the access to the database and the transactions occurred.
- *privacy*: access control by strict authentication, using a password.
- *availability*: all the data stores in the database are essential to the good behaviour of the system, so all of them will be reachable by accessing from the main server of the application.

4.3 Conceptual design

The main actors and objects that is essential to register to keep track of all the operations performed in the system are stored as entities.

The conceptual design allows us to decide afterwards, in the logical design, which elements and fields are indispensable to well describe and recognise the transactions and the operations performed in and by the system.

4.3.1 Entities

The entities stored in the database must allow the application to ensure all the services it claims to offer. The most important entities to be regarded in this system are listed below:

1. Customer

2. Field Operator

3. Car

Since both Customer and Field Operator are users that share some properties, which can be identified as keys to ensure the uniqueness of the entity, a new entity has been created, that carries those data.

4. Credentials

Moreover, we thought that, to better manage the information regarding the main properties of the objects of our system, it was necessary to add two more entities:

5. Coordinate

6. Area

4.3.1.1 Customer:

The customer is our final client. A new customer is created each time a new registration is made, and deleted when the final client deletes his/her account. The field of this entity are:

- ID: main key to identify the user. It's unique.
- Payment Data: employed to detain the payment information. It's a complex data type, in which are contained: credit card number, name and surname of the credit card owner, CCV and expiring date of the credit card.
- Driving Licence: complex attribute that contains all the main properties of the customer's driving licence.
- Coordinates: the final user can allow the system to detect his/her position by giving access to his/her GPS. (it's mandatory to search a car around the user position, otherwise it's not a required field and it's Null).

4.3.1.2 Field Operator:

The FO is an employee of the system. He/she is not only defined by his/her credentials, but also from his ID, which is unique.

He/she is registered at the time he/she is employed. (When the FO stops working for the society the database does not discard his/her information completely, but maintains a record of former employees).

A FO can be operative or not at the time.

- ID
- Operative (Boolean): the FO can be operative or not, and this field is used to detect the FOs to which is possible to submit cars.

4.3.1.3 Car:

A car is an important entity of the system. Not only it's an object that has to be kept under surveillance and the main service offered, but also it's able to send information to the system, which are important for the good behaviour of the application. The fields that describe the car are listed below:

- Car plate
- Car_status: it identifies the status of the car in each moment. As stated in the RASD, a car can be:
  - AVAILABLE
  - RESERVED
  - IN USE
  - MAINTENANCE
  - PENDING
- Lock_status: A car can also be locked or unlocked. In particular, our system provides three types of lock_status:

  - LOCKED
  - UNLOCKED
  - TEMPLOCKED: this kind of lock is used when the car is in the PENDING status.
- Charge: the charge status is described as an integer number going from 0 to 100.
- Psg_number: this field represents the number of passengers in the car at a given moment. In particular, this information is essential for the customer to gain discount after a ride. It is an integer value that goes from 0 to 5 (maximum number of spots in a car according to the assumptions).

4.3.1.4 Credentials: Credentials is an entity created to store information that describes the main identification point of a user of the system (both customer and field operator). So these data have been stored in a separate entity, which will be employed by both of the other two. The main fields of this entity are:

- Name
- Surname
- Password
- E-mail
- ID: Through the ID (which can be user_ID or FO_ID) it's possible to have access to the other data regarding the user, given Password and email (inserted when signing in).

4.3.1.5 Coordinates: Coordinates is a simple entity which is used in multiple other entities: this is why we considered it important enough to become independent.

- latitude
- longitude

4.3.1.6 Area: This entity is defined by two keys:

– position

– radius

Position identifies the centre of the area while radius defines how much space the area actually covers, starting from that point. Radius is a number, expressed in meters.

Moreover, there is a peculiar kind of area which is more interesting for our system: a safe area, which is the one in which the final customer (driver) can park the rented car.

– Safe: Boolean

We added a Boolean value that states if the area is safe or not. Stakeholders may decide to only store safe areas. However, we implemented the Safe field because it may happen that an area which was safe changes to not safe, (temporarily or permanently).

4.3.2 Relations

Relations tie entities together. Most of them, in the case of this system, are complex relations which needs, to be specified, to be described as entities themselves.

We recognised three fundamental relations:

1. Search: a user can do a non-determined numbers of searches.

2. Reservation: one customer can reserve multiple cars, but only one car at a time.

3. DealWith: this relation ties the FO and the cars he deals with (hence the name).

The following is a relational diagram representing the interactions between the database entities.

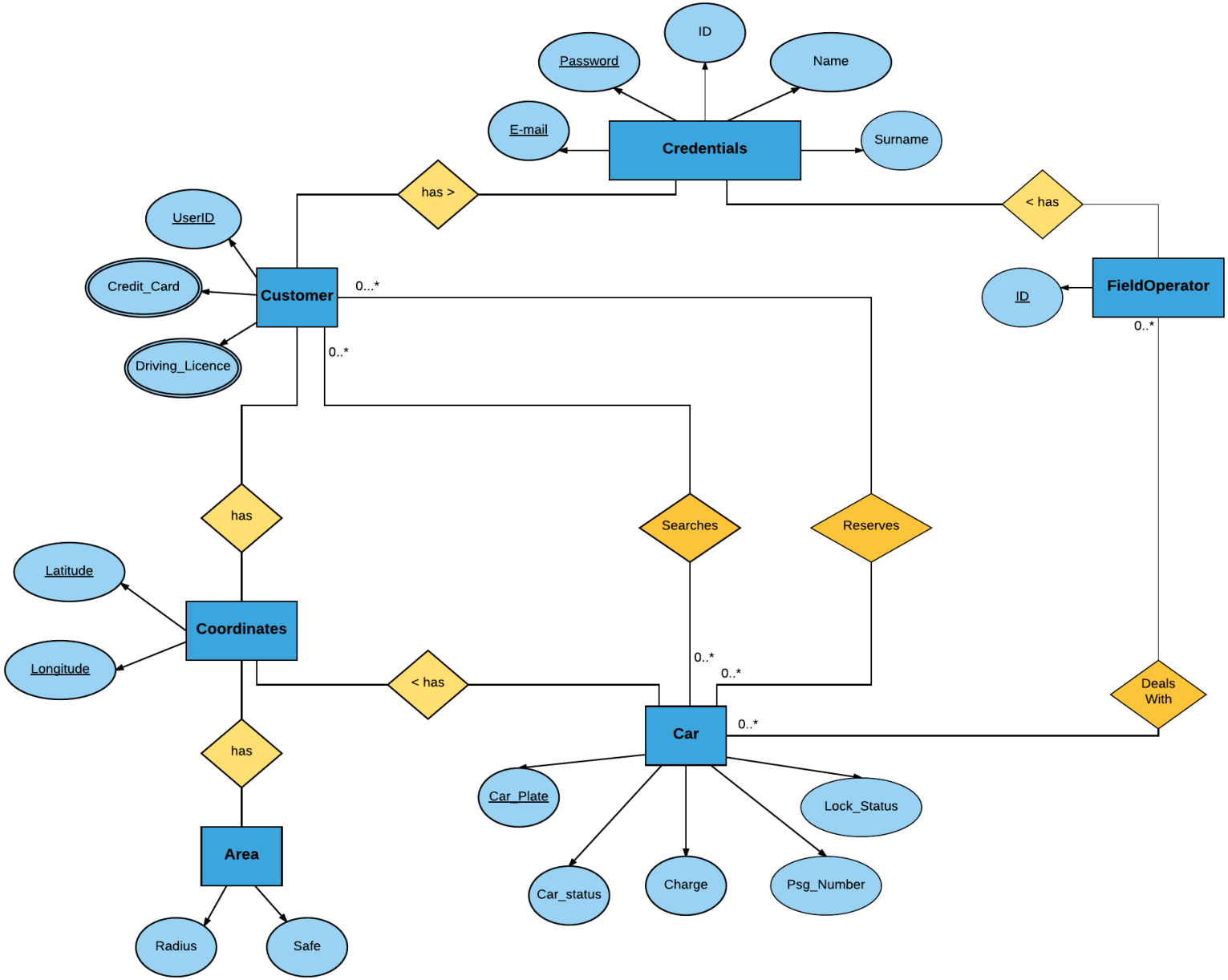


Figure 9: ER diagram

4.4 Logical design

N:N relations and complex attributes must be translated into entities.

4.4.1 Search

Search is uniquely identified by an ID. It also refers to the final user that made the search (user_ID).

We considered that the main purpose of memorizing the search is to be able to show to the users their last choices for a search and not their results: therefore, we will not save into the persistent memory the result of the search (also because it will likely be changed the next time the user chooses the same search).

The information stored in the search entity:

– Search ID

– User_ID: The identification of the user that makes the search.

– Coordinates: the position that is inserted (either by typing an address or by GPS). The coordinates to be inserted in the field are calculated by the system.

4.4.2 Reservation

The reservation is one of the most important entities: it keeps track of all the service process.

- Reservation_ID
- User_ID: the user that reserved the car.
- Car_Plate: the identifier of the reserved car.
- Time_Reserv: time in which the car should be picked up according to the reservation.
- Time_Start: the actual time in which the car was switched on
- Time_end: if the reservation is completed, this field will present the time in which the final user has left the car and has been charged with the bill. Otherwise it's Null.
- Status_Reservation: The reservation can be:
    - VALID
    - CONFIRMED
    - FROZEN
    - COMPLETED
    - CANCELED
- Search_ID: The reservation is always linked to a previous-done search. It may be useful to keep track of the creation of the reservation to have a direct link to the search.

4.4.3 DealWith

Relationship that links the FO and the Car: when the car needs to be charged, or has any kind of mechanical/physical/technical problem, the FO is informed and needs to take care of it.

The DealWith relation registers all the data necessary to keep track of the work of the FOs and the costs of the interventions and so on.

The fields present in the entity will be:

- Intervention_ID
- FO_ID
- Coordinates
- Time_Request: the time in which the request of intervention was sent to the FO.
- Status_Request: the status of the request. It may be:
    - HANGING
    - DEALING
    - TO_THIRD_PARTY
    - COMPLETE
- Time_End_Request: the time in which the status of the request was changed to COMPLETE. If it is yet to complete, this field is Null.
- Type: the request can be of different types. The ones that we provided at this time of the design study are:
    - CHARGE
    - SYSTEM_FAILURE

4.4.4 Physical Structure

This is a representation of the entities and relationships as tables of the database.

Legenda:

___: foreign key

___: primary key

1. Credentials (Name, Surname, Email, Password, ID)
2. Field_Operator (Field_Operator_ID)
3. Customer (User_ID, Coordinates, Credit_Card_Nb, Driving_Licence_Nb)
4. Coordinates (ID, Longitude, Latitude)
5. Area (Coordinates, Radius, Safe)
6. Car (Car_Plate, Charge, Psg_Numbers, Car_Status, Lock_Status)

7. Search (Search_ID, User_ID, Coordinates)

8. Reservation (Reservation_ID, Search_ID, User_ID, Car_Plate, Charge, Psg_number, Car_Status, Lock_Status)

9. DealWith (Intervention_ID, FO_ID)

10. Driving_Licence (ID_Number, Name, Surname, Starting_Date, Expiration_Date)

11. Credit_Card (ID_Number, Name, Surname, CVV, Expiration_Date)

ID, foreign key in the Credentials table, may be both User_ID or Field_Operator_ID.

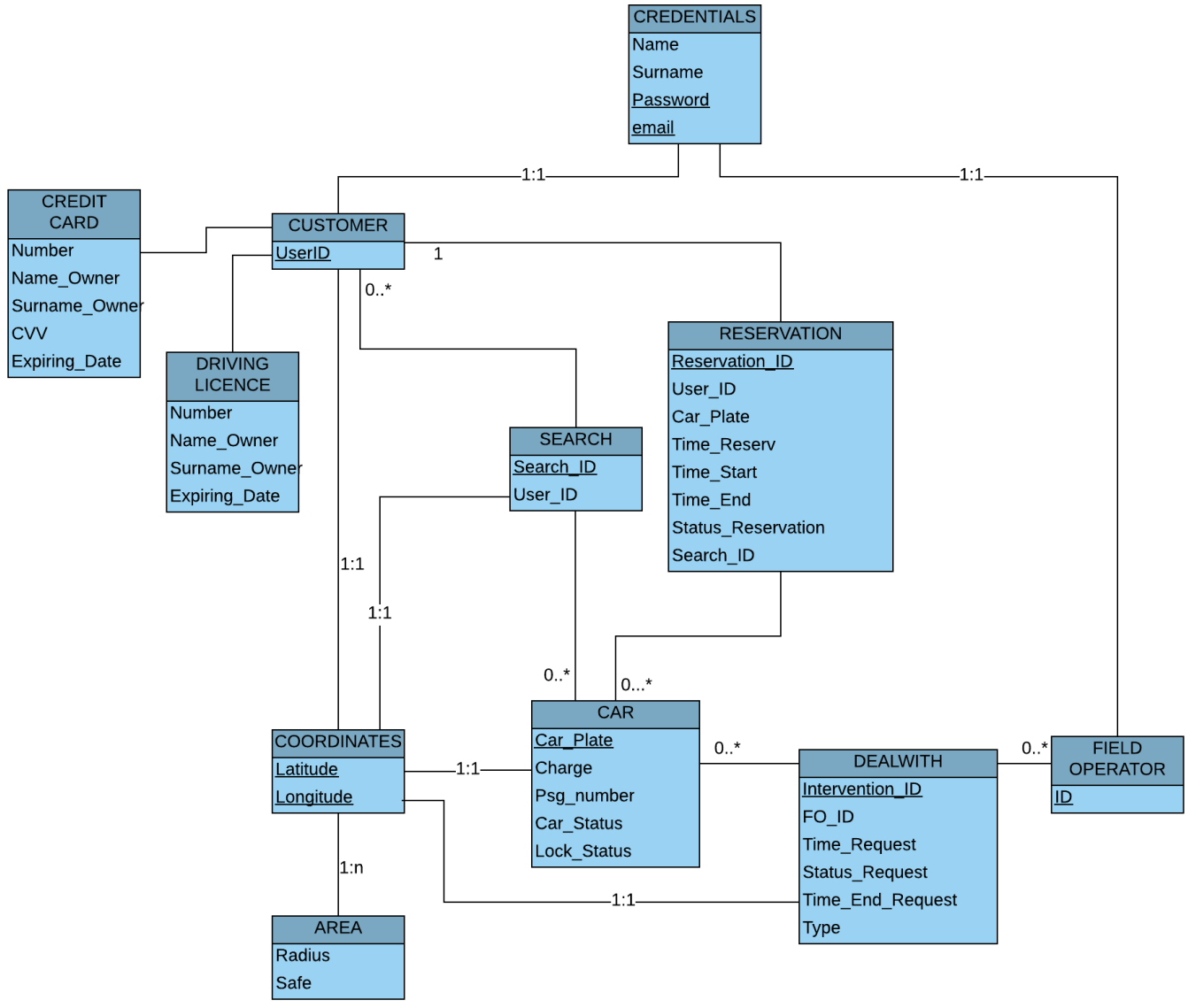


Figure 10: Database entities

5. User Interface

5.1 Mock-ups

An initial design of the user interface was proposed in the RASD. In this document we will reprise and expand the interface examples we proposed and add an essential interface: the field operator interface.

5.1.2 Field Operator Interface

All the FOs will be equipped with a tablet. This will allow them to have access to the application.

FOs can access to the application like every other client, but the system will recognize the password and email pair as belonging to a FO_ID.

Each FO has a profile, where he/she is advised from the system if a problem occurred. The e-mail associated to the profile is a work e-mail, so that for any necessity the FO can communicate with third parties through this contact.

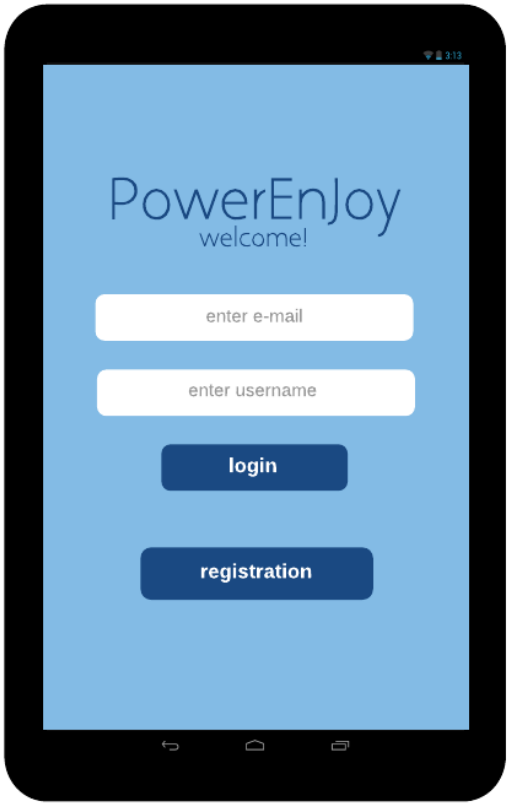


Figure 11: Operator's Sign In screen on a tablet

Each FO can operate on multiple cars: when a car needs charge, is damaged or assistance is required, the FO is informed.

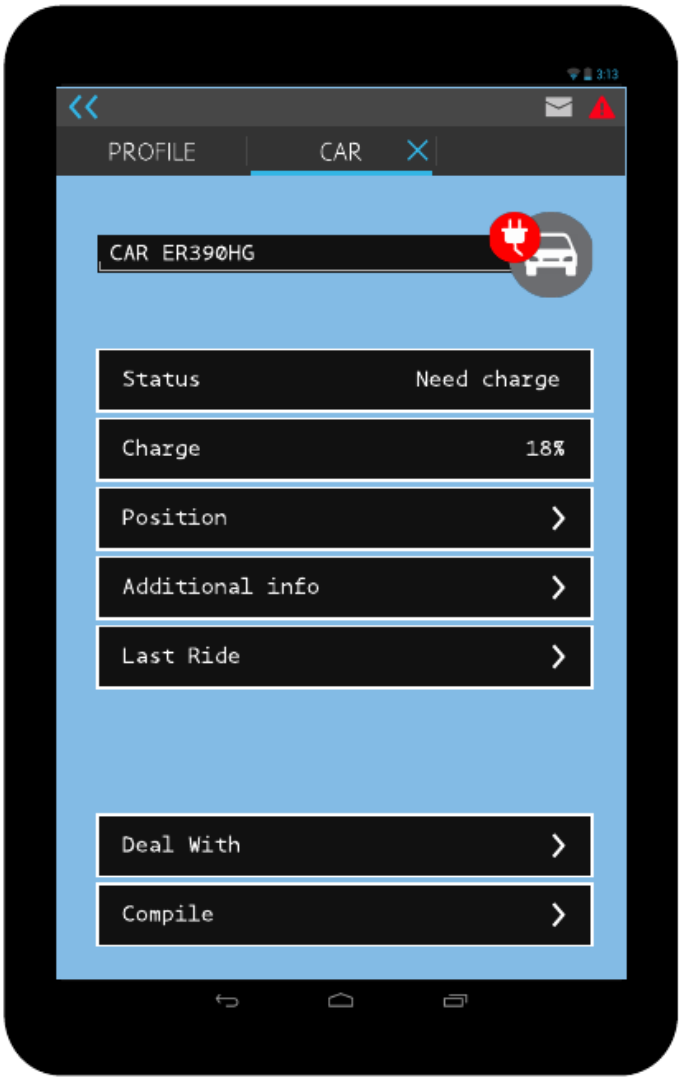
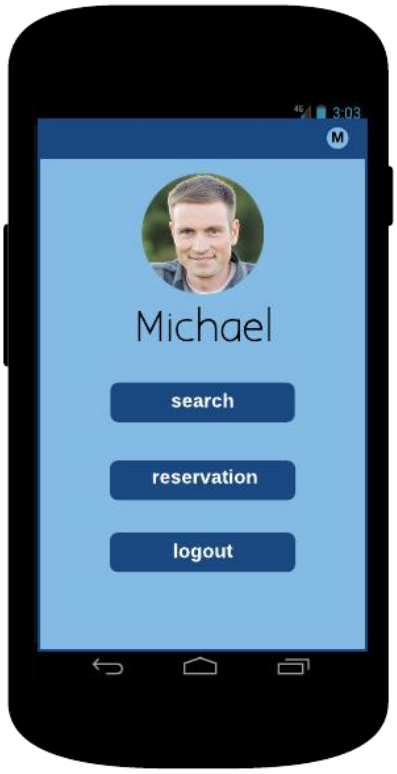


Figure 12: Operator's car profile

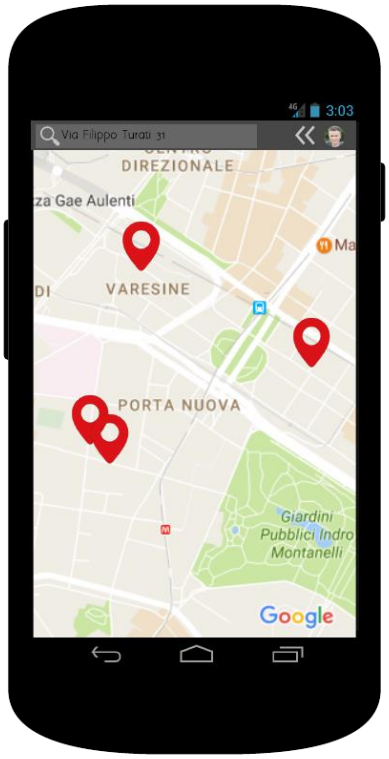
5.1.3 User Interface

The user interface is essentially the one we presented in RASD.



The user can sign in (or register) and is presented with the main page, from where he/she can log out, check out the current reservation, search for a car or look at his/her profile and personal information (blue dot on the right upper part of the screen)

Figure 8 A: User sign in *Figure 8 B: User profile*



If the user chooses to push the "search" button, he will be asked to insert an address (or his/her position through accessing to the GPS of the phone) and one of three ranges (which will search for available cars in walking distance from the search starting point).

Figure 9: Search

Figure 10 A: car info

Figure 10 B: set time of reservation

By clicking on a car, the user is shown its properties, and has the possibility to reserve it, by inserting the reservation time.

5.2 User Experience

5.2.1 User UX Diagram

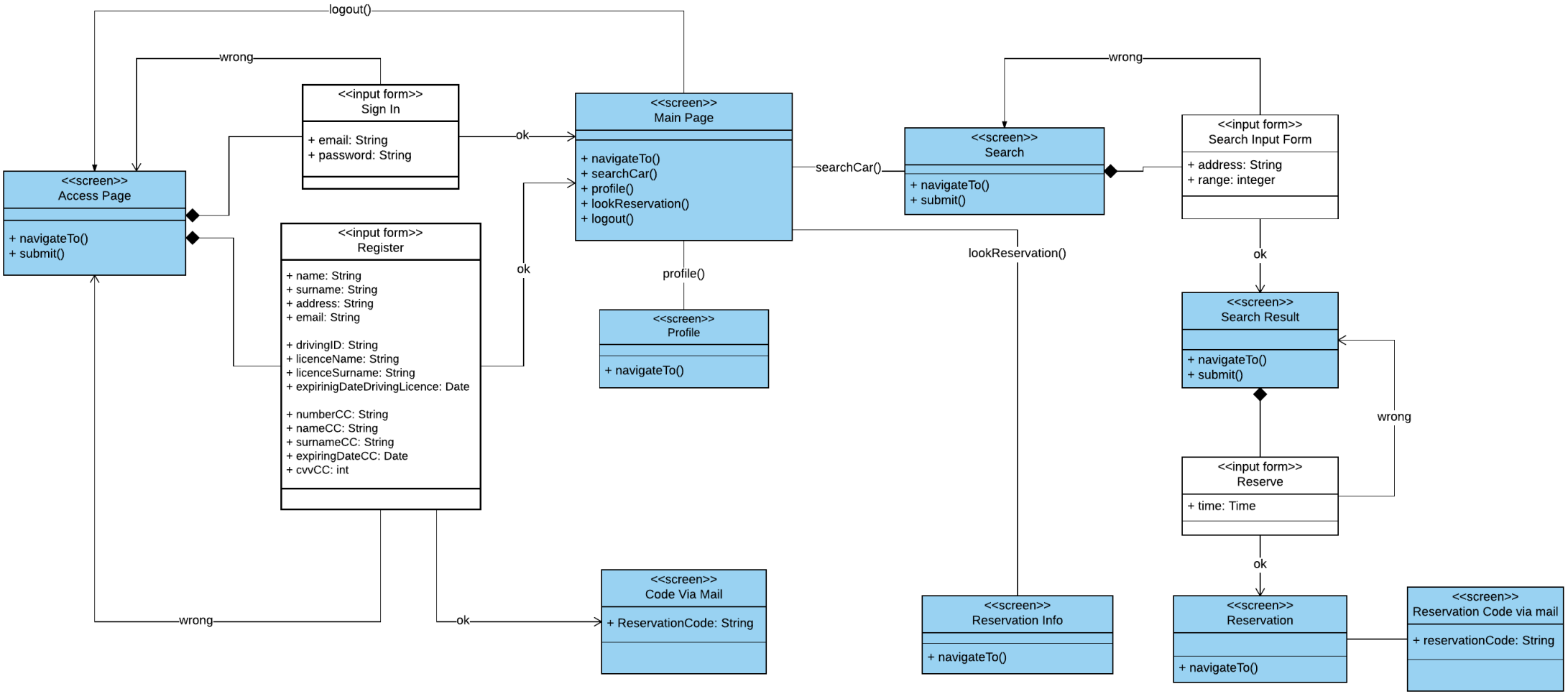


Figure 11: UX Diagram, user

5.2.2 FO UX Diagram

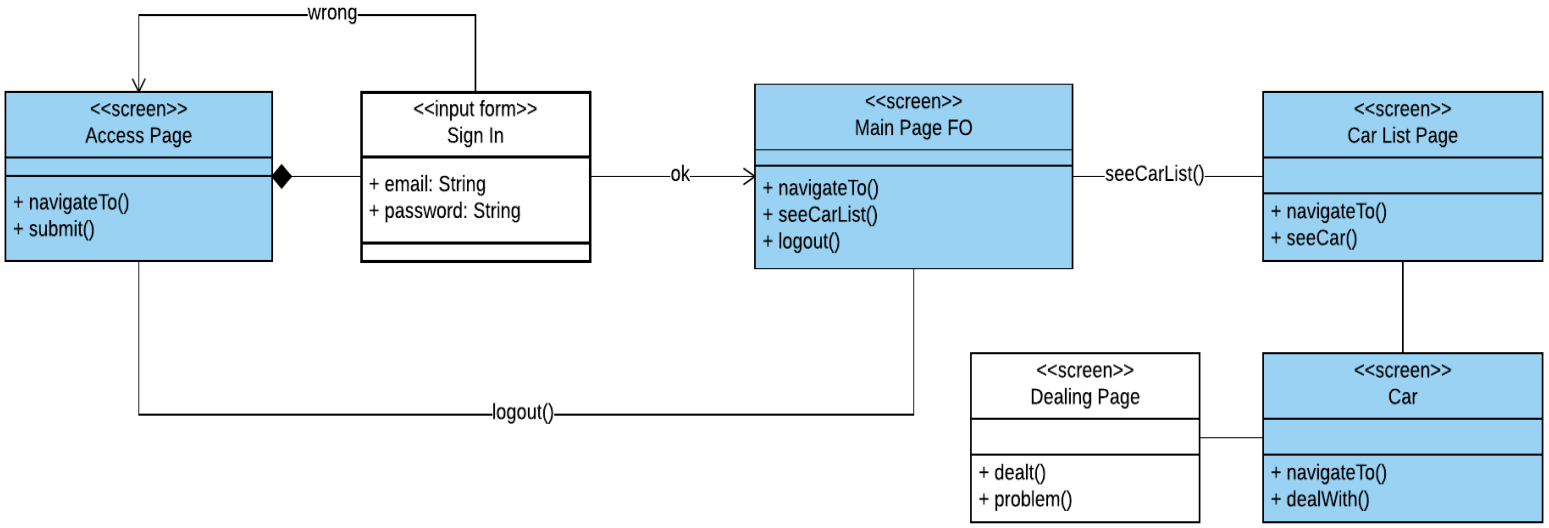


Figure 13: Field Operator UX

The FO accesses to the system like the normal user. The system links the password-email pair to a FO_ID, and the FO is sent to his/her main page, from where he/she can perform mainly two actions: log out (which sends him/her back to the sign in page) and go to the list of the cars he/she is supposed to take care of during his/her shift. Each car shall have the main problem specified.

The FO can choose the "deal with" option, and he/she will be sent to the "dealing page" from where he shall inform the system that the problem has been fixed, or that ulterior problems occurred.

In this first FO's application design we leave place for further improvement. Particularly, the FO's tasks may be improved and increased. This design leaves place for further development of new stakeholders' requests.

5.3 BCE

In this section we will present Boundary Control Entity diagrams (BCE) to better present the interaction of the components of the system. We choose to introduce this diagrams because they represent in detail what is offered to the user and how the system handles the requests from the users.

In the first BCE, *Figure 14*, we see the diagram related to the Login and Register operations for the user as well as the field operator.

From the home screen of the application, they can either choose to login if they are already registered or to register, in case of first time users. To do that, they have to fill in a form where information needs to be inserted. The information inserted is checked by the controllers. Once the user sends the filled in form, the controllers check the integrity of the information inserted. We have two different controllers, the Registration Manager and the Sign-in Manager. As indicated by their names, they handle the registration process and the sign-in process, respectively. They are both sub-components of the Profile Manager presented in the architecture design. A successful sign-in takes the user to the homepage of his/her profile, while if the registration is successful, the user is readdressed to the home page, where he/she is supposed to sign in with the password he/she received by email. We can also see the entities involved in executing of these operations and the changes that they undergo.

In the second BCE, *Figure 15*, we can see the diagram related to the Search and Reserve operations. A user that is logged in can decide to search for available cars. He/she fills the form, indicating the location (being that the location found by the application through his/her mobile or tablet GeoLocation or him/her manually inserting the desired location) or he/she simply select one of their recent searches. Once the user sends the filled in form, the Search Manger handles the elaboration of the request. The resulting cars are shown to the user or in the case where no cars available cars are found a message is shown to the user. The user can at this point either decide to do another search or reserve one of the available cars or.

In the first case, we restart the process again.

In the second case, he/she needs to insert some more information, the time in which he/she desires to pick up the car. After that, the Reservation manager handles the request. The user's information is checked, to evaluate whether he/she can reserve a car or not. If the user is not allowed to reserve a car, a message is shown to the user through the application, otherwise the operation is finalized. A reservation is created and an e-mail is sent to the user with the necessary information. The entities involved in this two operations are also shown in the diagram as well as the changed they might undergo.

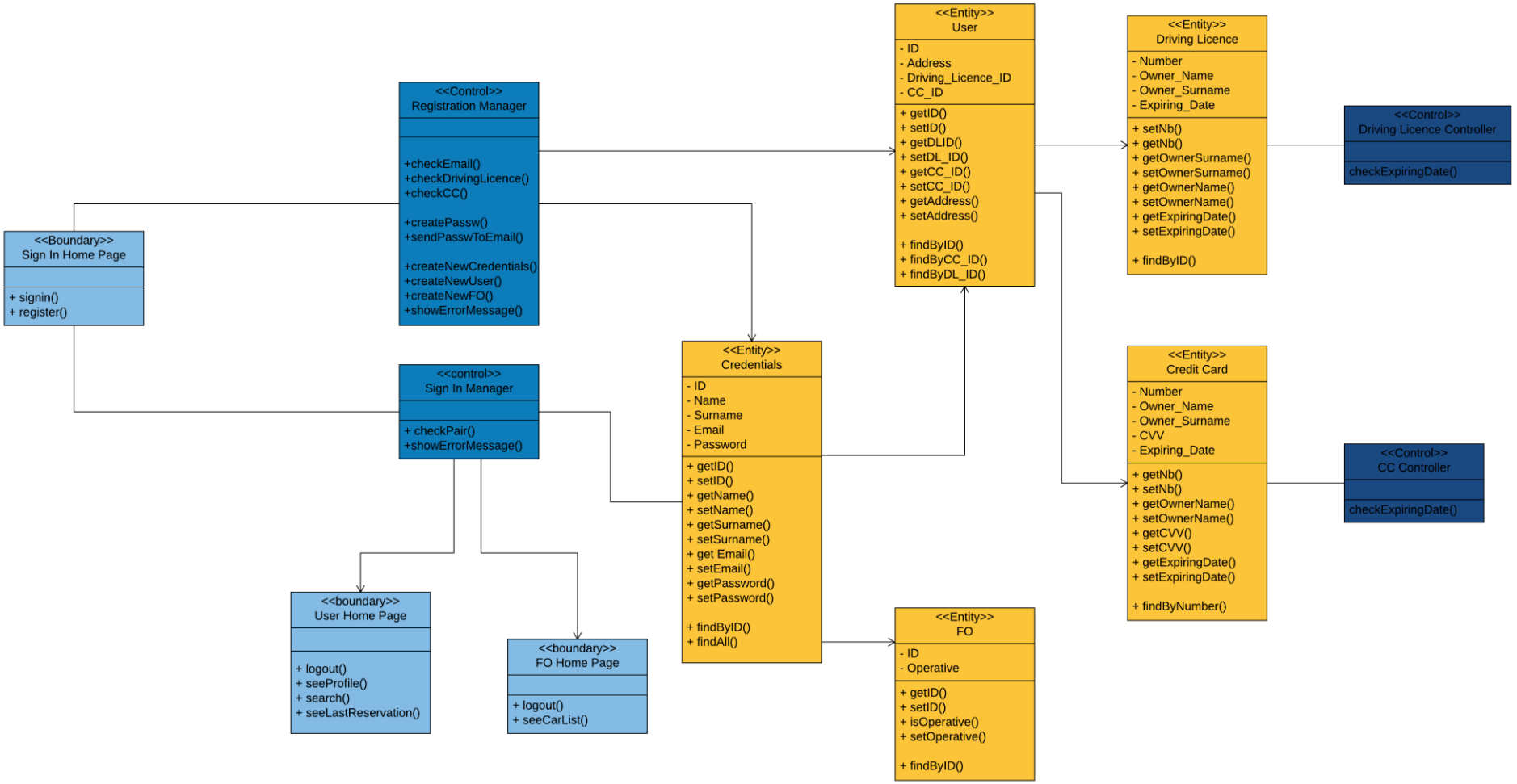


Figure 14: BCE-LogIn

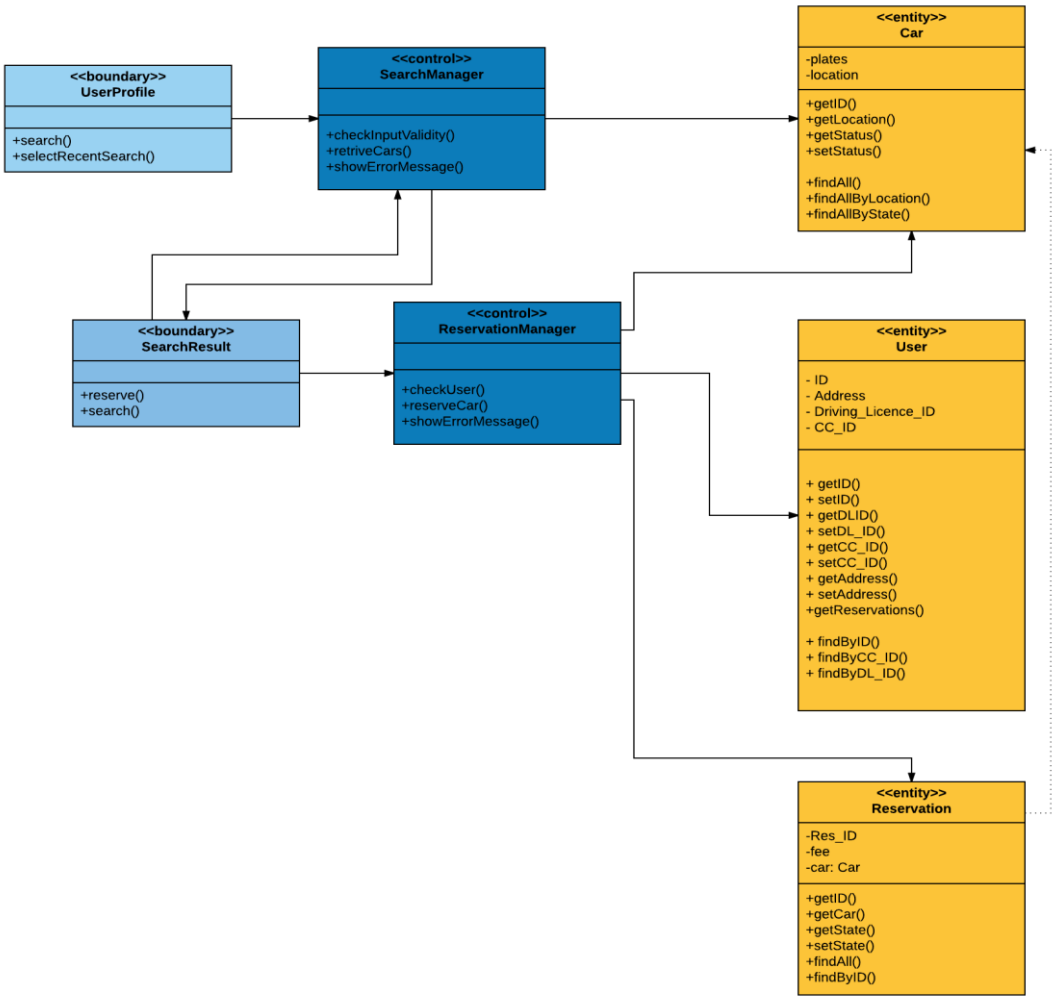


Figure 15: BCE-Search and Reservation

6. Requirement Traceability

In this section will explain the mapping of the requirements introduced in RASD with the components introduced this document, how they cover the functionalities offered by the system to the clients. For each component, we will show the functional requirements covered by said component and how that is achieved.

- User Profile and Profile Manager: These are two components, the first one of the Client tier and the second of the business tier. Even though they belong to different tiers and handle tasks designated for them, they still apply to the same area of functional requirements. The requirements they cover are:
  - Management of the user's profile:
    - [F.R.1]: A guest can benefit the application's services only if he/she is registered.
    - [F.R.2]: A user's registration is confirmed if and only if his/her credentials match the credentials associated to the driver licence inserted.
    - [F.R.3]: A user can log in if and only if he/she inserts the right combination of e-mail address and password.
    - [F.R.4]: Once logged in a user can view his personal information.
    - [F.R.5]: A user should be able to request a password recovery.
    - [F.R.6]: A user should be able to update his/her payment method.
    - [F.R.7]: A user should be able to update his/her driving licence information.
    - [F.R.8]: A user should be able to update his/her e-mail address.

- Reservations and Reservation Manager: These two components handle the reservations made in the system through the mobile app. The moment the client selects reserve after having done a search, the main controller is called and sends the request to the Reservation Controller. It is the reservation controller that generates the reservation ID and that changes the state of the car picked by the client for said reservation, if the client doesn't have any other ongoing reservation. The e-mail confirmation is sent to the client using the E-mail gateway with a recap of all the information inserted as well as the reservation ID. A notification is also sent to the client through the app, notifying him that the reservation was done successfully. The functional requirements that these components cover are:
  - Management of car reservation [G.2]:
    - [F.R.14]: A user can reserve a car if and only if there is not another car reserved by him/her in the desired time interval.
    - [F.R.15]: To reserve a car the user must insert the reservation time.
    - [F.R.16]: The reservation time must be: $t_{RESERVATION} <= t_{CURRENT} + 1$ Hour
    - [F.R.17]: The picking up time must be: $t_{PICKUP} <= t_{RESERVATION} + 1$ Hour
    - [F.R.18]: The user should be able to cancel a reservation for up to 10 minutes before the reservation time

- Recent Search and Search Manager: these components handle the car searches done by the clients. Once the client has inserted the necessary information for searching a car, the main controller calls the search controller. The search controller goes through the list of all the cars offered by the system, finding the available ones, in the desired location inserted by the client

and in the range indicated and then sends said information to the client through the main controller. The functional requirements covered are:

- Search of a car [G.1]:
  - **[F.R.9]:** A user can search a car if and only if he/she is registered.
  - **[F.R.10]:** The system shows to the user only the cars that are coherent with his/her search.
  - **[F.R.11]:** A car is coherent with a search if and only if it's available and its position is within the search range.
  - **[F.R.12]:** A car can be selected if and only if it's shown in the search.
  - **[F.R.13]:** A selected car can be reserved.

- Payment Manager: this component is responsible for calculating the final fee for the ride requested and also communicating with the external software of the Payment Gateway in order to finalize the payment of the fee. After the payment is done, an e-mail is sent to the user with a recipe, using the E-mail Gateway. Interacting with the reservation controller as well as the car controller (that of the car associated to the reservation), the payment controller covers the following functional requirements:
  - Payment [G.4] [G.5]:
    - **[F.R.21]:** The system calculates the fee once the ride is concluded.
    - **[F.R.22]:** The ride is concluded when the user parks in a safe area, turns off the car and exits from the car without choosing the freeze option.
    - **[F.R.23]:** The fee is calculated based on a fixed amount of money per minute. *The amount of money to be paid is chosen by the administrators of the system.*
    - **[F.R.24]:** Before applying the fee, the system checks if the ride is eligible for a discount.
    - **[F.R.25]:** If for the whole ride there were more than three travellers, the system applies a 10% discount.
    - **[F.R.26]:** If the user connects the car to a power grid of one of the special parking areas after the ride, the system applies a 30% discount.
    - **[F.R.27]:** If the user leaves the car with more than 50% of the charge left, the system applies a 20% discount.
    - **[F.R.28]:** If the user leaves the car more than 3 km away from the nearest power grid station, the system charges him/her 30% more.
    - **[F.R.29]:** If the user leaves the car with less than 20% of the charge left after the ride, the system charges him/her 30% more.

- Operator Manager: this component is responsible for handling the operators of the system. The cars which are under maintenance, meaning the cars that are in need of fixing, being that either charging, internal and/or external damages or any other kind of problems presented, are shown to the operators, synchronizing everything with the operators' shifts. The modifications made by the field operator through the application are transmitted to the controller through the main controller.

- Car and Car Manager: even though the physical car is not a component of the system, it still plays a crucial role for the application. The car manager is responsible for the communication with the physical car, meaning that it handles the interaction with the software installed in the car.

7. Effort Spent

In this section we give the effort spent by each member on the assignment, specifying what each one of the members worked on.

Adriana Cano, parts covered: Architectural Design: Overview and all the subsections, Components View, Components Interfaces, BCE Search-Reserve diagram, SD: Search, Requirement Traceability, Layout

Hours: 25

Agnese Bruschi, parts covered: Database representation, Mockups, UX diagrams, BCE Login diagram, SD: Login, layout

Hours: 25

Daniel Botta, parts covered: Algorithms, Runtime view diagram, Deployment view diagram, partook in the decisions regarding the architecture of the system

Hours: 15