Design Document

Sommario

Pietro Crovari,
Enrico Gnecco

Introduction

Purpose

The aim of this document is to precisely define the architectural choices and needs that PowerEnjoy application has to own.
The RAS Document focused on the needs of the stakeholders, on the domain assumptions and on what specifications are for the application; starting from that, the following chapters will introduce an high level overview, detailing more and more what are the modules and needs, reaching what PowerEnjoy really has to satisfy from a software and a hardware points of view.

Scope

From the highest level, some actors can be identified within PowerEnjoy application:

- Users
- Cars
- Operators

All the work operated by the application is to manage and control these entities.
On one hand the system allows the users to reserve cars, it has to calculate fees, manage payments and, to do so, control the position of each vehicle relating it to the position of predefined safe areas and to car status; on the other hand the system needs to manage operators and their relation with vehicles.
Different actors have different interfaces that act concurrently and independently, on the same resources. Therefore the architecture of the system needs a module managing shared data and the connection among all the interfaces, satisfied by a central server and database.
Moreover there is the need to manage the pre-existed system, a server with a database which manages safe areas, hiring operators and information about cars.

The functionalities, briefly described above, will be fulfilled by the architecture chosen. In addition, reliability, efficiency and data protection will be considered during the design process.

Definitions, Acronyms, Abbreviations

RASD: requirements analysis and specification document

DD: design document

API: application programming interface

URL: uniform resource locator

Safe areas: pre-defined areas where the user can park the car, this action will stop the bill charging

Pietro Crovari,
Enrico Gnecco

Reference Documents

- Lectures' Slides

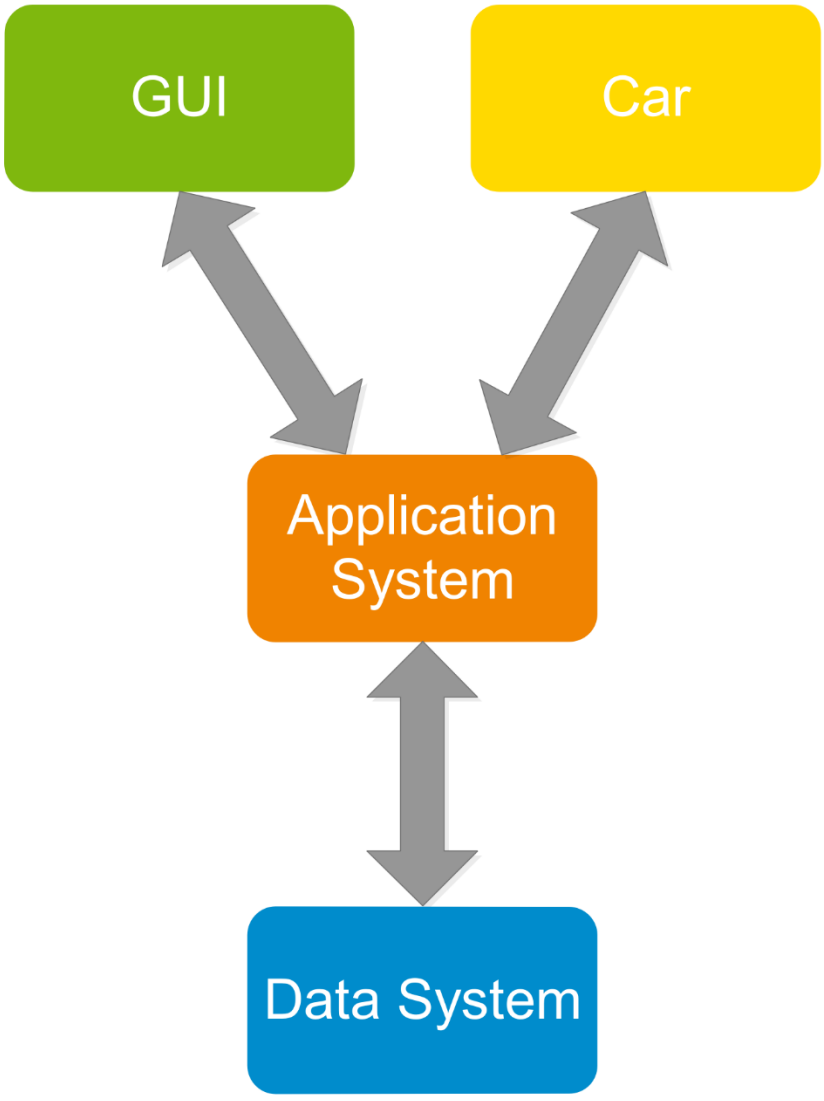
Document Structure

First of all, we consider a high level approach in order to consider all the possibilities and go towards the fulfilment of every requirement previously detected in the RASD. The more the topics are faced, the deeper the approach becomes, until algorithms and low level representations are touched.

Architectural design

Overview

In the previous "Scope" chapter, a high-level approach has been adopted to define which are the main modules and interfaces; here those are schematized and analysed to provide a better comprehension of what are the internal needs and agent inside them.

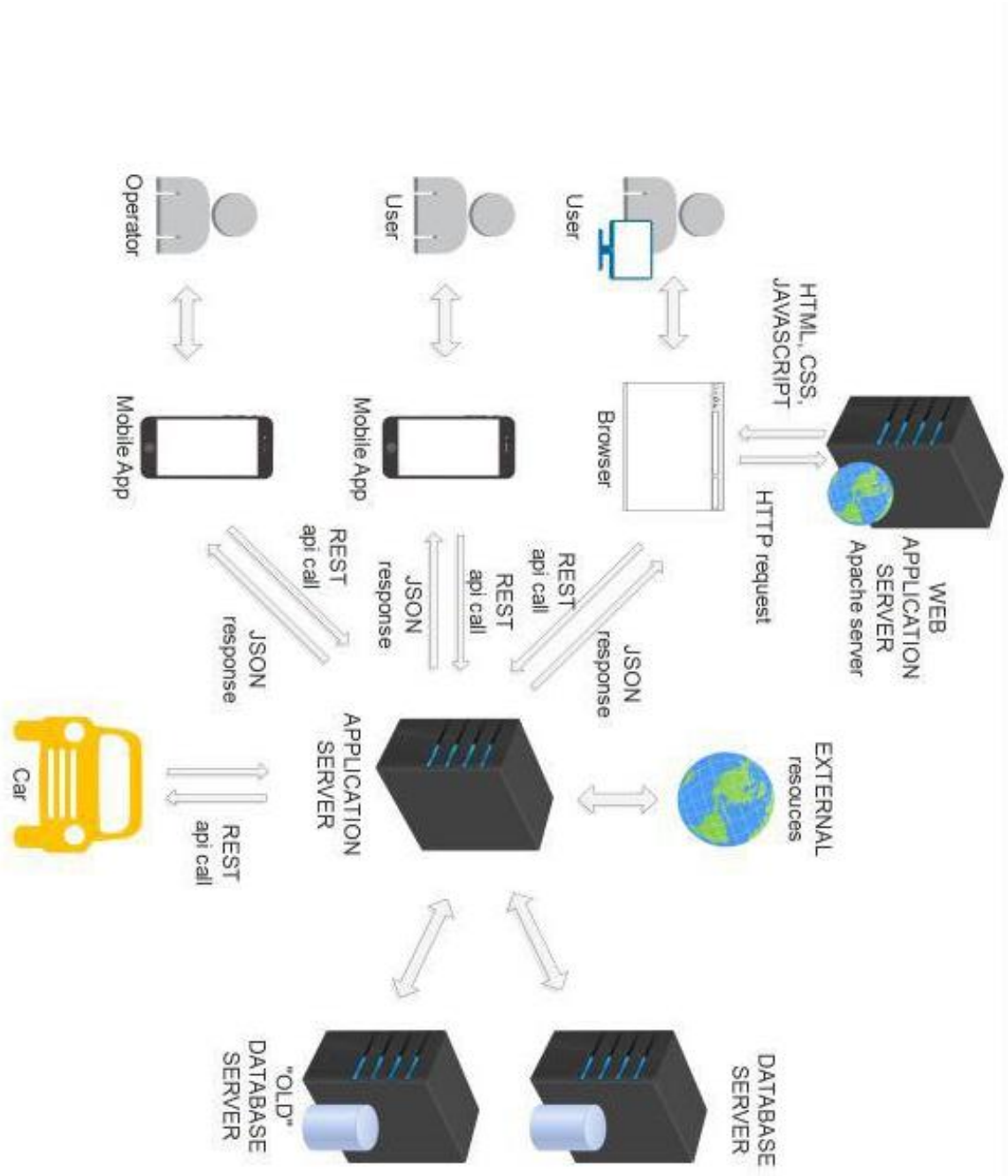


Pietro Crovari,
Enrico Gnecco

6 main components compose our Architecture:

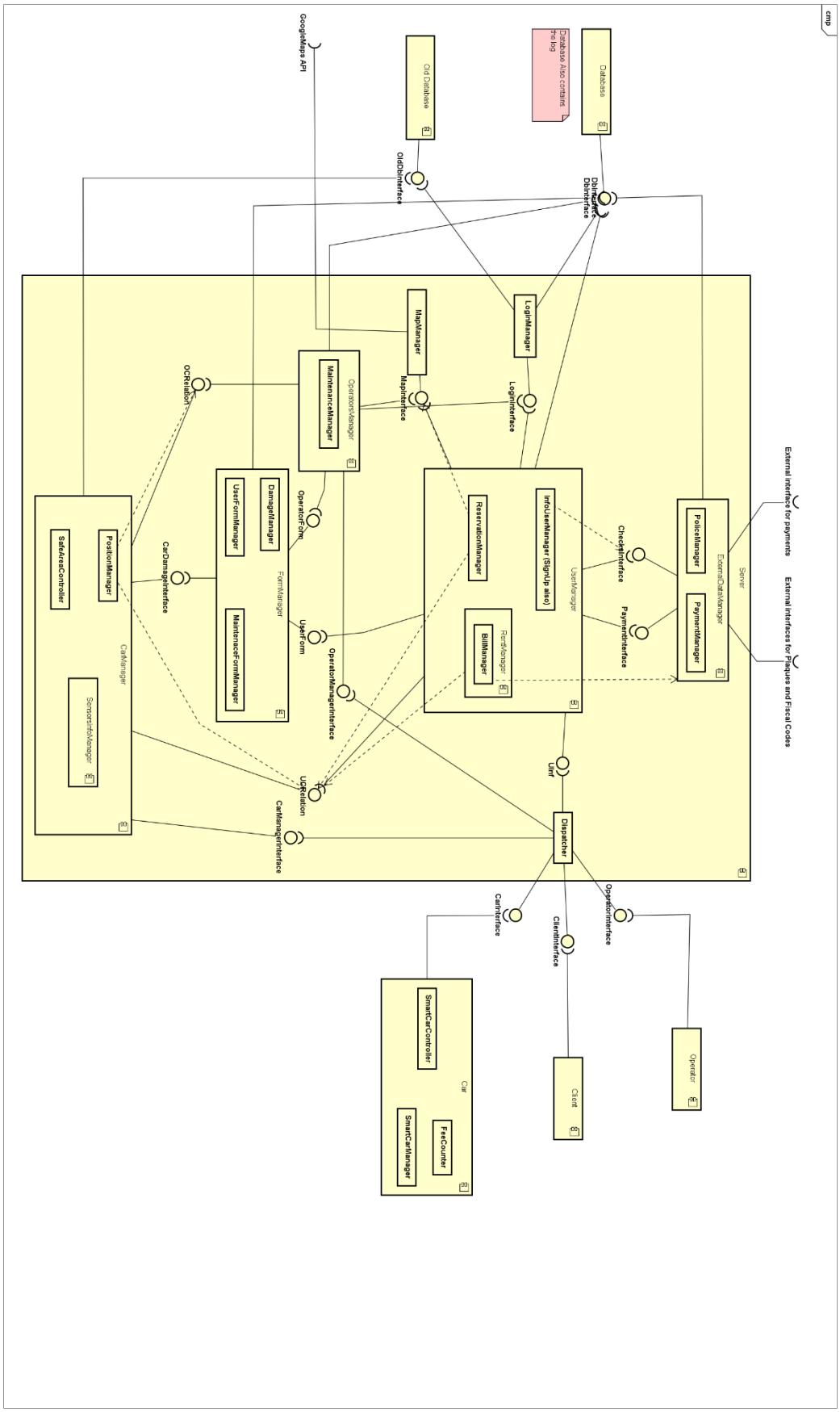
1. A central Application;
2. The client's interface;
3. The Operator's interface;
4. Car's Computational Unit;
5. A database;
6. The "old" database;

In practice, the central Application is the core unit of the System. It receives clients' requests, made by the app or the website and answer them properly, sending them either the information required or the confirmation of the reservation of a car. It is also responsible for the vehicle management, collecting information from them, such as their position and all the other data (battery left, status and so on), and sending all the required communications, for instance massages for unlock the doors or change its status. Also operators, through their mobile app, deal with the central Application, receiving information about cars that require maintenance and being allowed to access to vehicles. The central unit is not responsible for storing data, but the database in charge to do it. The application requires it polls it every time he needs some information. Finally, the "old" database, it contains all the safe areas and the pre-existing data, it will interface to the central application too, when necessary. The communication between the central unit and cars, clients and operators is synchronous, in fact when a device (either a client or an operator's one) send a request to the Application, it waits for a confirmation message. In the same way Every communication with the car requires an acknowledgement.



Pietro Crovari,
Enrico Gnecco

Component view



Pietro Crovari,
Enrico Gnecco

UserManager: the management of all the data concerning the users is given to this module, as submodules there are: a BillManager, dealing with the payment processing, a ReservationManager, dealing with the reservation processing, and finally a InfoUserManager, supposed to control the flow for everything regarding the personal data of the users.

FormManager: forms are processed, and we thought of a suitable unit in our schema in order to deal with this particular aspect of the whole. Forms are offered, processed and sent to be stored to the new database, supposed to contain our log.

OperatorManager: as a duality with respect to the UserManager, this is supposed to control every information flow regarding operators. The MaintenanceManager is a sub-unit thought to control the works operated.

CarManager: as the cars are actors of our world, we thought that a dedicated unit would have been the suitable choice for the design of the server. It needs three sub-component: a PositionManger, a SafeAreaManager and a SensorInfoManager; the distinction of them is appropriately chosen to maximize the division inside the unit and to keep the structure as simple as possible.

ExternalDataManager: this component is supposed to deal with external APIs, in particular, with the bank and the police ones; this choice follows the paradigm, already shown, of keeping different tasks separated in order to simplify the structure on the whole, obtaining a linear behaviour.

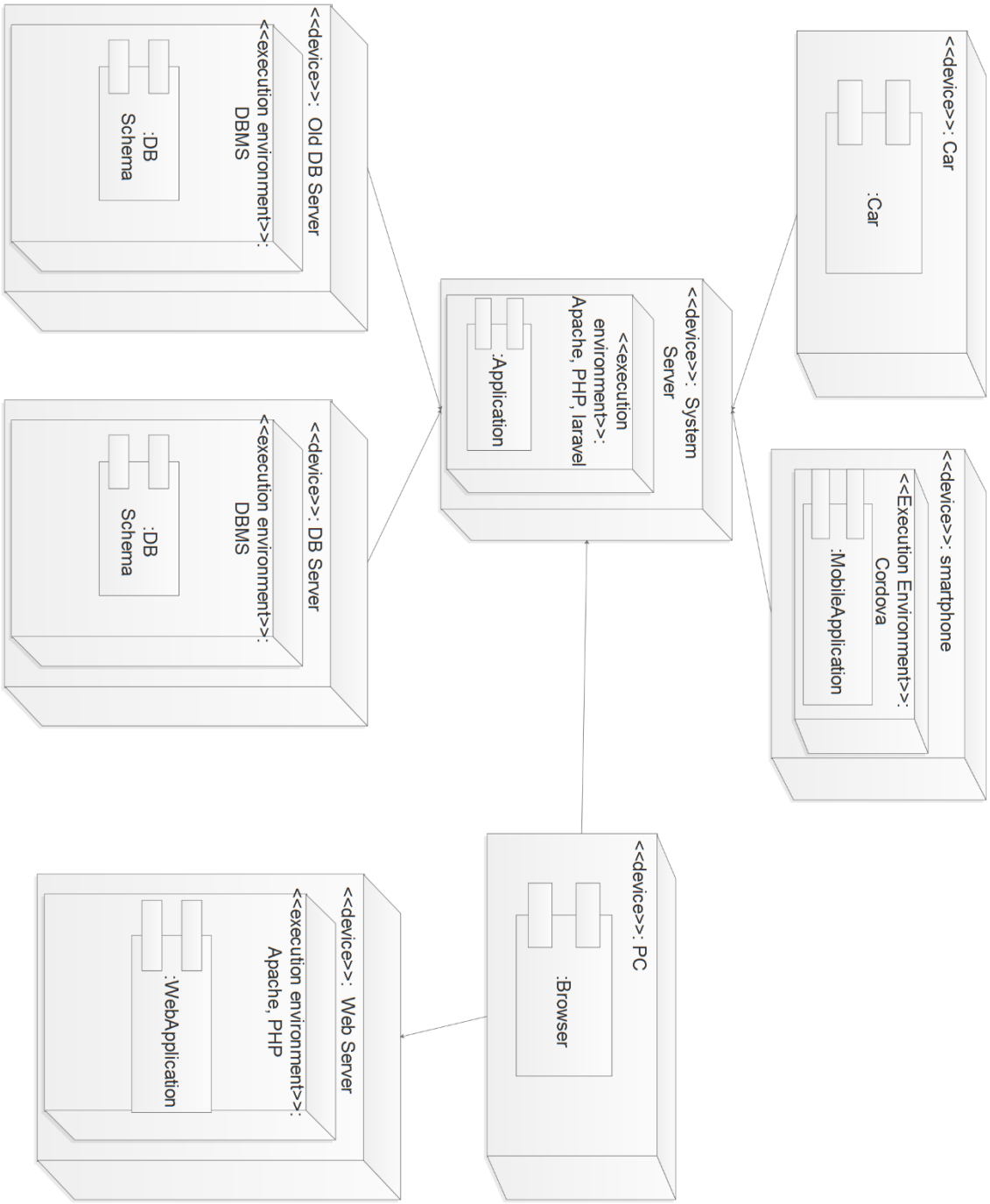
MapManager: the system needs the creation and correct integration of maps; this component is thought with this intent. The external APIs chosen here are the Google's ones, or equivalent. There will be a brief discussion later about this topic.

LoginManager: as both operators and users need this sub-component, we thought to separate it from the respective one in order to better control the whole behaviour. Its function is merely to offer a way for the login to the two other components.

Even if the map manager relies on external services, we decided not to including it in the ExternalDataManager component mainly for 2 reasons:

- Driving Licence check and payment system must work with local realities, we want our system to be portable, hence transparent to the particular interface of police system in a specific city. Maps APIs, instead, are the same for every city where our system will be located. Hence, we don't need an additional layer to guarantee transparency.
- PoliceManager and PaymentManager are only based on a pre-existing function; in fact, they only have to use external interface to fulfil their functions, the MapManager, instead, must build the map interacting more with the external services. He is even in charge of modifying the result of the APIs call, adding the vehicles to the generated map.

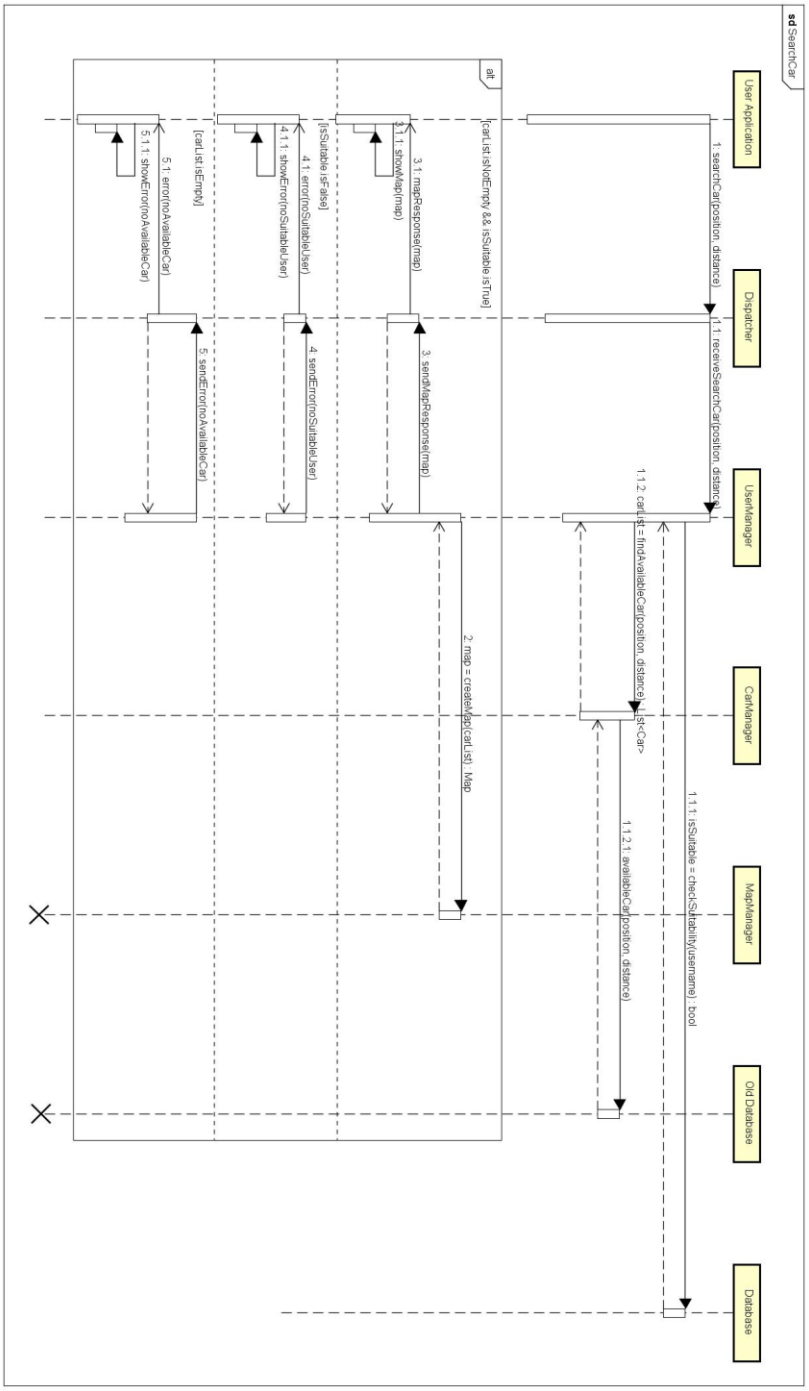
Deployment view



Pietro Crovari,
Enrico Gnecco

Runtime view

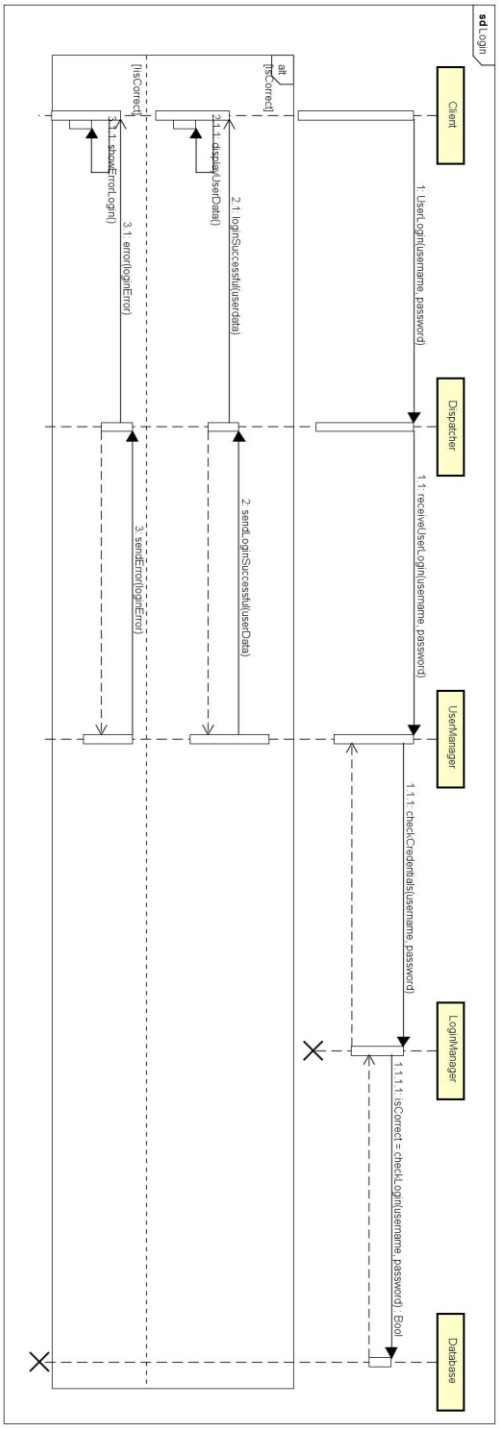
Search Car



In this diagram, we can see what happens when a User wants to look for an available car. Once he has sent the request to the server, it firstly check if the Customer is eligible of booking a car, that is if he has a valid driving licence and no pending payments on his account. If he passes these checks then the servers asks the database the list of cars which satisfy Users' parameters (availability and position within the selected range from a determinate position). If there are, they are returned to the server. At this point, the User Manager invoke the Map manager's method to build a map displaying all the car, and then sends it to the User's

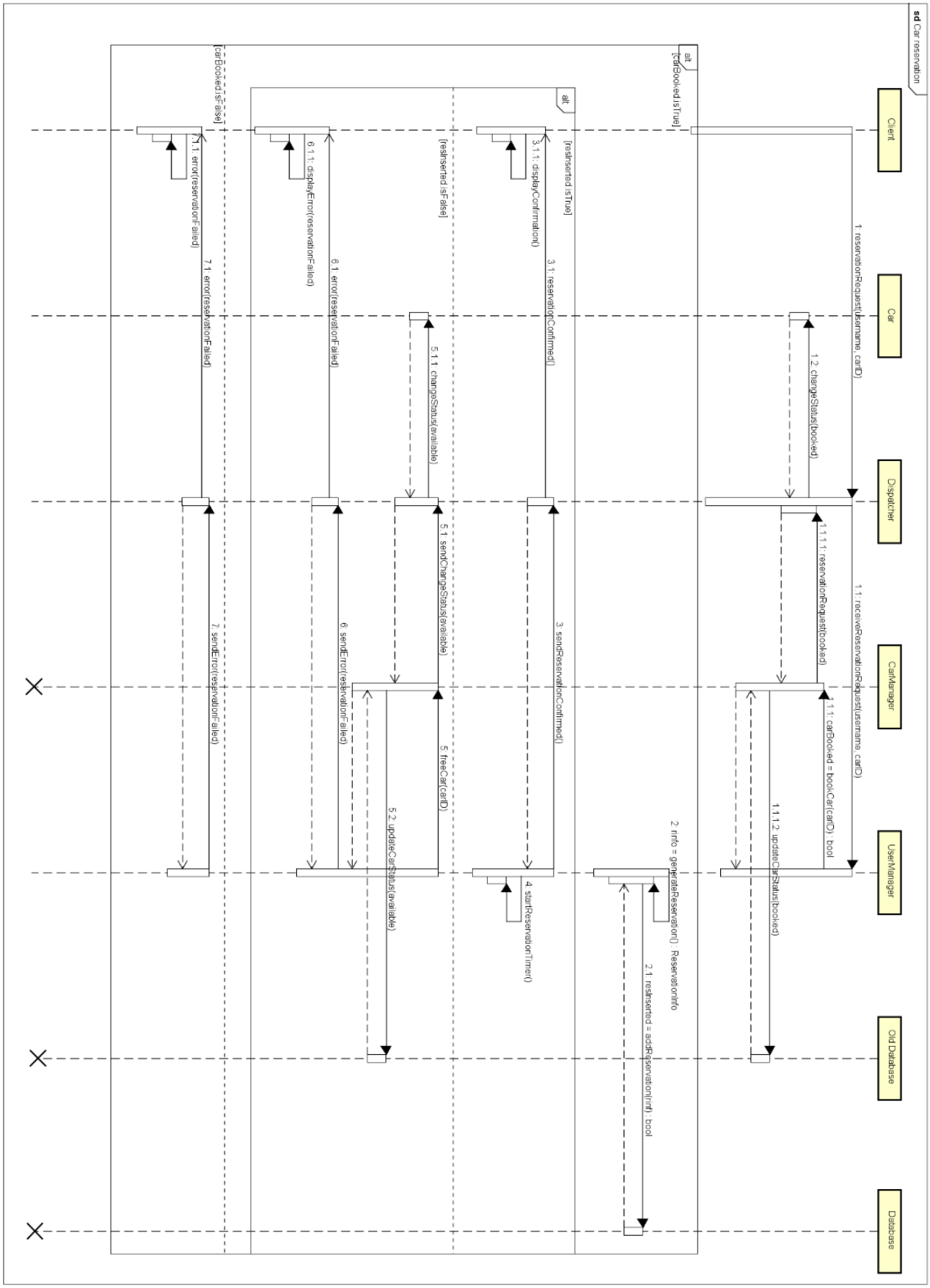
device. If there are no available vehicles, the Client shows an error, message, inviting the user to try another research.

Login



Once the Customer has inserted his credentials, they are sent to the server, that check their correctness and, if they correspond to the ones on the database, it allows the Client to access to the system's functionalities

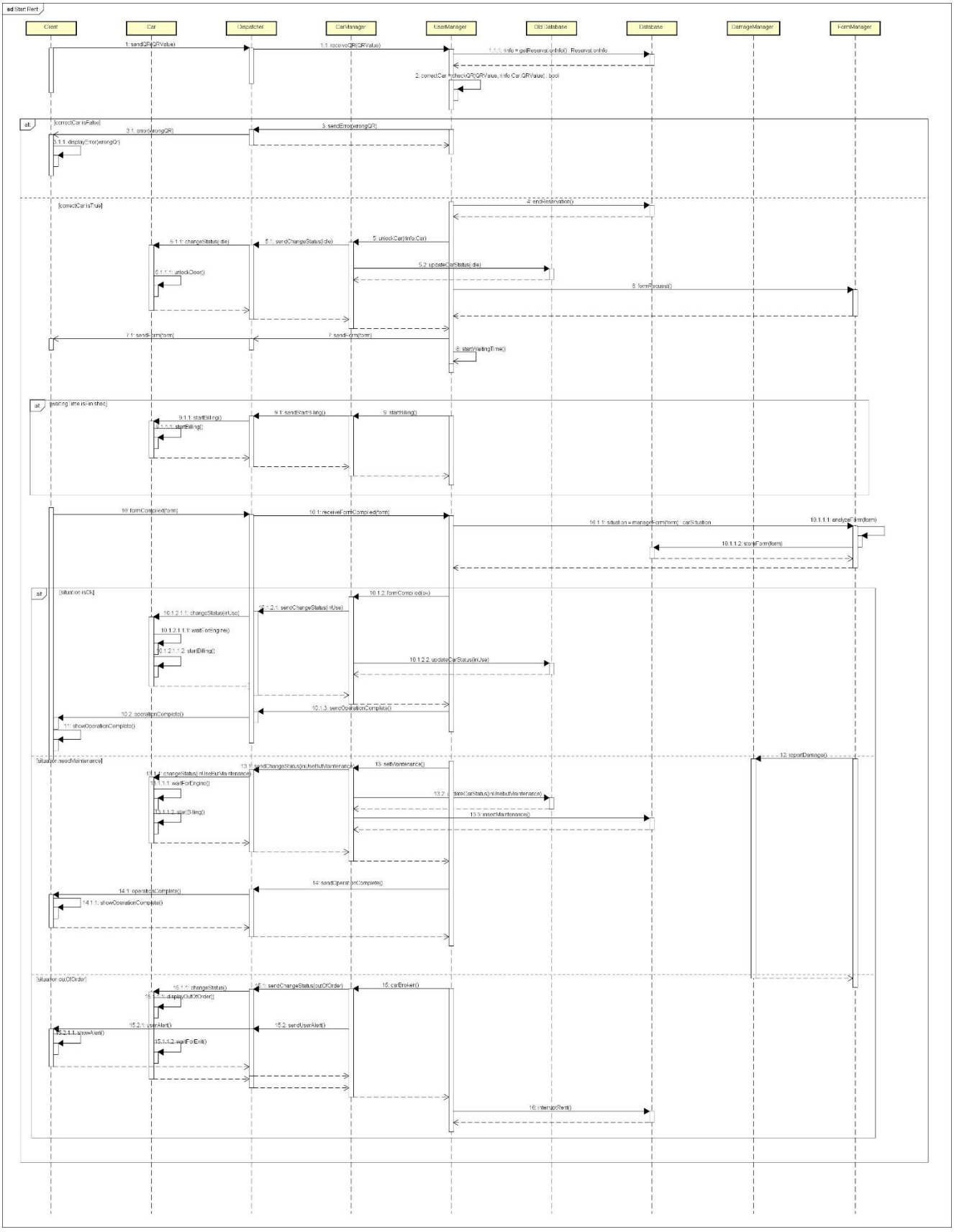
Car Reservation



Pietro Crovari,
Enrico Gnecco

When a Customer wants to book a Car, the UserManager asks the CarManager to change the status of the vehicle to booked. If this one manage in updating it (also on the Old Database, the one containing all the information of the cars), a confirmation message is sent to the Customer. Otherwise, the car is made available again and a message of error invites the customer to try the reservation again.

Start Rent

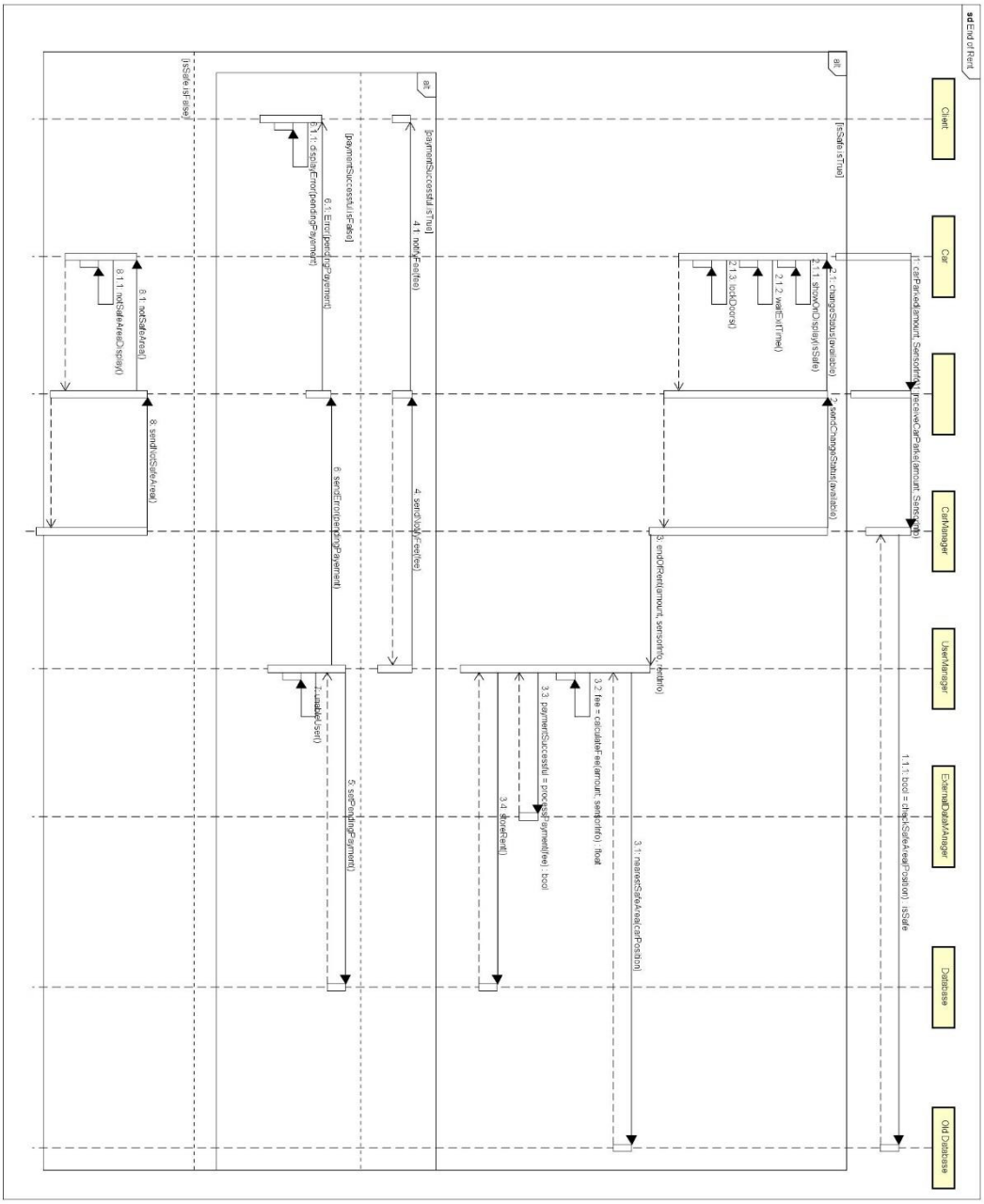


Pietro Crovari,
Enrico Gnecco

The rent starts when a customer send the QRcode on the car he has booked to the system. Once the server receives it, the ClientManager asks the CarManager to unlock that vehicle, updating its status to idle.

Meanwhile it requests to the FormManager to produce the form to send to the Customer. When this operation is completed it sends that to the Client's device and starts the timer used to avoid an improper use of the car. Once the form is sent back compiled, the FormManager analyzes to understand either the car needs maintenance or not. If the car is unusable, then the Customer is asked to exit the vehicle, it is locked and set as Out of Order. Otherwise its status is updated to inUse or inUseButMaintenance (according to either there has been any report on the form) and the engine is allowed to ignite. If the timer expires before the ignition of the car, the vehicle starts billing the rent anyway.

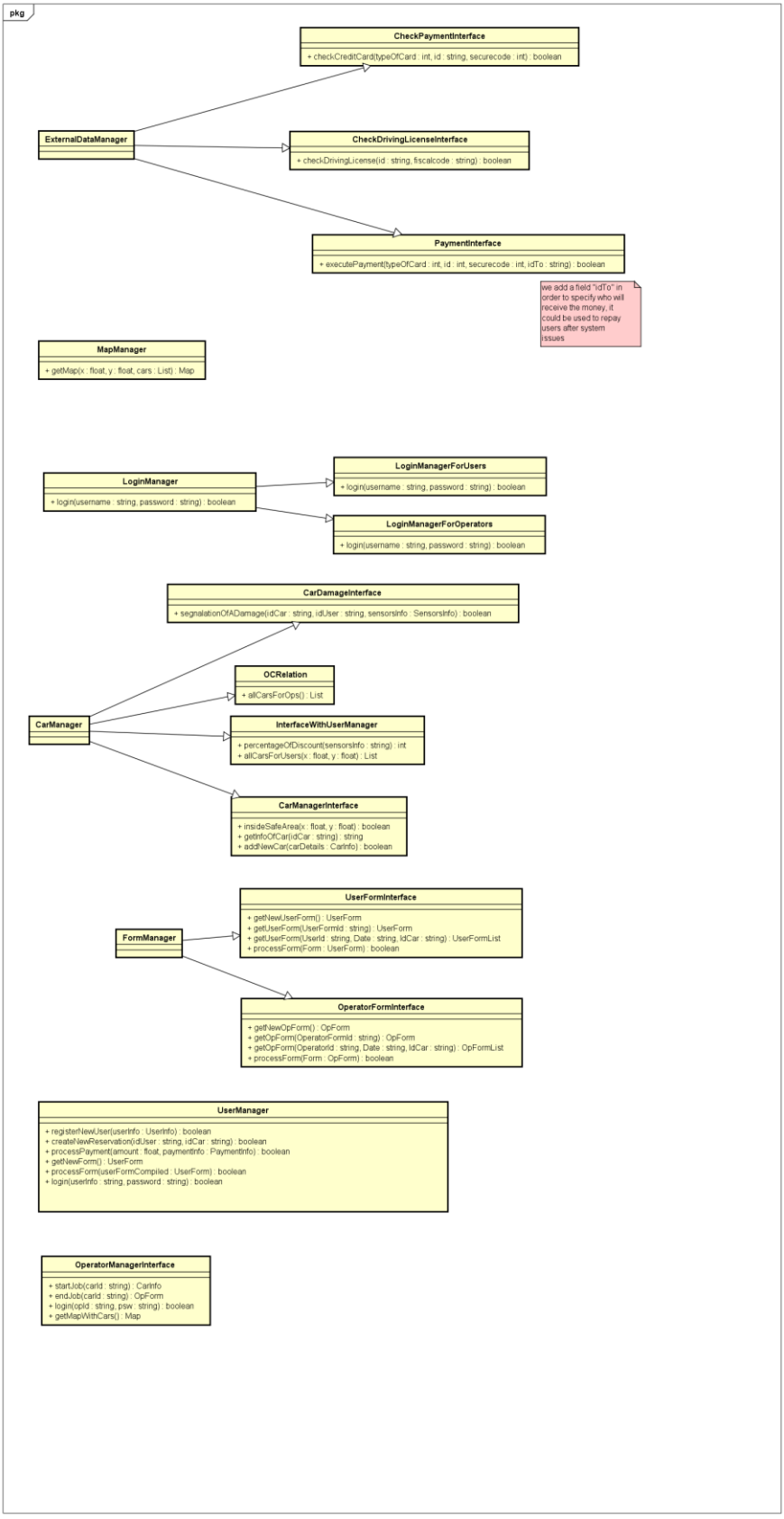
End Of Rent



When a car is parked, it communicates it to the CarManager, which immediately checks either it is in a safe area. If so, the Customer is invited, through the display, to exit the car. When he is gone, the vehicle locks itself and communicates it to the Server. Here the UserManager calculates the fee of the rent, keeping in consideration all the information received by the CarManager about the requirements to obtain dome discount. Then, the ExternalDataManager processes the payements. If it is not successful it is notified to the Customer, and his account is modified adding the pending fee. He won't be able to book another car

until he doesn't extinguish his debt. On the other hand, if the procedure finishes successfully, the User his notified with the payed amount through his personal device.

Component interface



Selected architectural styles and patterns

A multi-tier architecture has been chosen for fulfilling the requirements detected.
There can be detected a MVC, Model View Controller, as a part of the architecture and a three-tier system, with a dispatcher dedicated to an internal event-based structure, as the other part of the whole.

From the MVC point of view: on one hand, a distinct server for the presentation of data, so HTML, CSS, and everything regarding the web app visualization, is given; on the other, a server is entirely dedicated to the management of data; this aims to separate the data and how it will appear.
Moreover, the communication client-server is managed by a dispatcher using an event-based approach; this has been chosen to give more elasticity to the usage of the system and to fulfil the needs of the customers.

From the three-tier point of view there can be detected: a client, a logic server and a storage unit. This consists in two databases: one is assumed to be the old, pre-existent unit dedicated to cars, safe areas and operators, the newest instead is designed to store all the other information, regarding users, payments, the log and so on. This has been chosen to take in consideration the previous system adding what requested.

Maintainability, security and reliability has also been considered designing the system. This is clear looking at the internal structure, constructed over modules, easier to be managed.

Other design decisions

It must be underlined the great usage of external APIs, indeed an entire unit is dedicated to the interaction with external services as Credit Cards and Driving Licenses. Moreover, a unit is dedicated to the generation of maps using Google APIs.

Algorithm design

```
function calculateFee(amount : float, rentInfo : RentInfo) : float {
        //Following the flow diagram done in RASD
        if(rentInfo.isCharging()){ amount = 0.7*amount; }
        if(rentInfo.carBatteryPercentage >= 50){ amount = 0.8*amount; }
        else{
                if(rentInfo.carBatteryPercentage <= 20){ amount = 1.3*amount; }
                else if(rentInfo.carBatteryPercentage > 20){
                        distance = rentInfo.distanceNearestPowerGridPosition;
                        if(distance > 3km){ amount = 1.3*amount; }
                }
        }
        if(rentInfo.numberOfpassengers >= 2){ amount = 0.9*amount; }
        return amount;
}

function checkSafeArea(carPosition : (float, float), radiusArea : float, radiusAccuracy : float) : boolean{
        //Because there can be multiple safeAreas in a district, we take all of them and we check if the car
        is near to one of them
        List<(float, float)> possibleSafeAreasPositions = getFromDbSafeAreas(carPosition, radiusArea);
        bool check = false;
        foreach((x,y) in possibleSafeAreasPositions){ check = (check or particularCheckSafeArea(carPosition,
(x,y), radiusAccuracy)); }
        return check
}
```


Pietro Crovari,
Enrico Gnecco

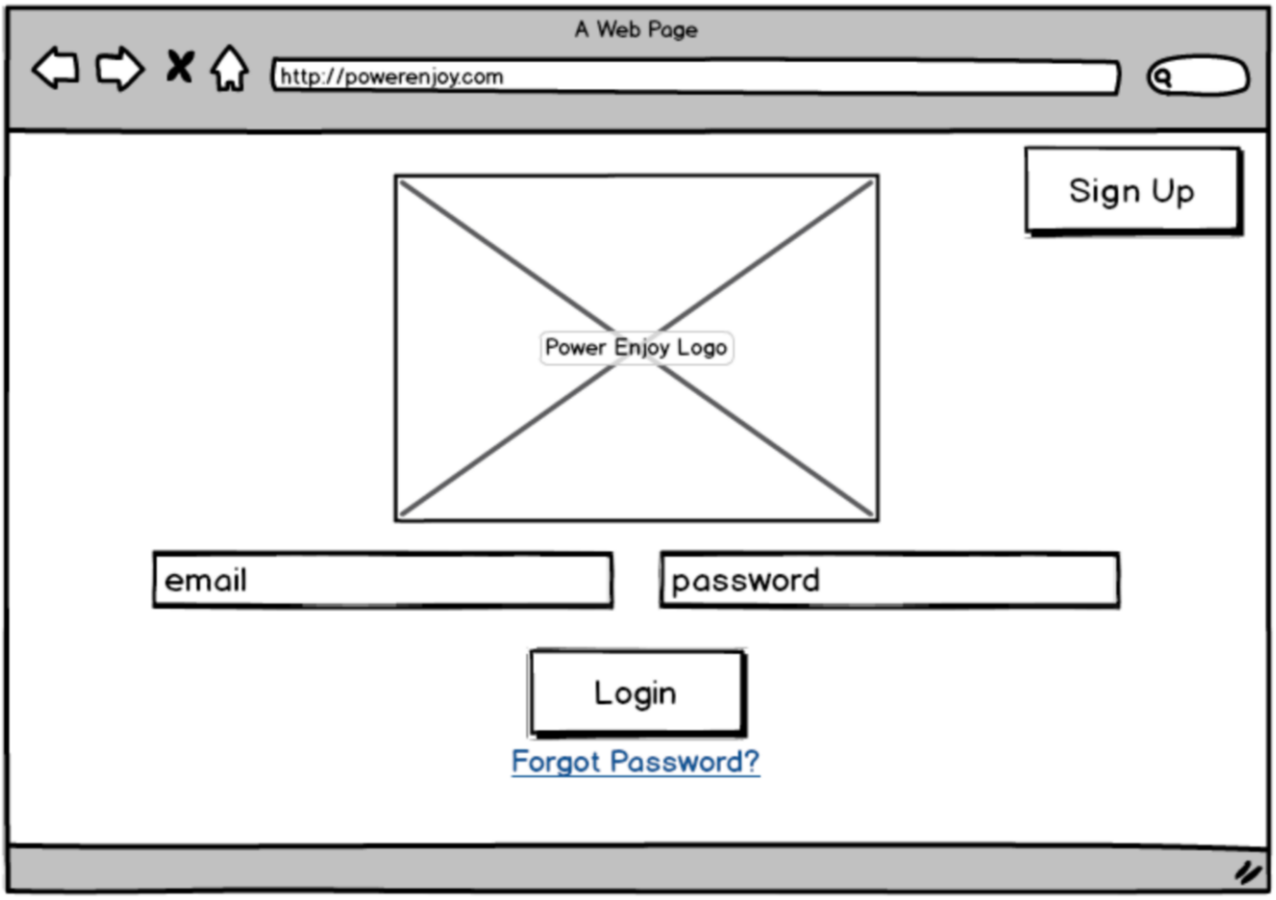
```
function particularCheckSafeArea(carPosition : (float, float), centerSafeArea : (float,float), radius : float) :
boolean {
        xOk = false;
        yOk = false;
        if((carPosition.x >= centerSafeArea.x - radius) && (carPosition.x <= centerSafeArea.x + radius)){ xOk
= true; }
        if((carPosition.y >= centerSafeArea.y - radius) && (carPosition.x <= centerSafeArea.y + radius)){ yOk
= true; }
        return (xOk && yOk);
}

function signUp(userInfo : UserInfo) : string {
        DBDealer db = new DBDealer();
        if(db.alreadyPresent(userInfo.username)){ return "Please change the username, it's already
present!"; }
        string userInfo.password = password(userInfo.password);
        bool check = signUpUser(userInfo);
        if(check == true){ return "Your password is " + userInfo.password + ". SignUp completed"; }
        else{ return "An error occurred, please retry"; }
}

function signUpUser(userInfo : UserInfo) : boolean{
        //Check the validity of the information given
        ExternalDataManager extManager = new ExternalDataManager();
        bool checkDrivingLicense = extManager.checkDrivingLicense(userInfo.idDrivingLicense,
userInfo.fiscalcode);
        bool checkCreditCard =
extManager.checkCreditCard(userInfo.typeOfCard,userInfo.idCreditCard,userInfo.securecode);
        //Insert into the database the new information given
        DBDealer db = new DBDealer();
        bool insertion = db.insert(userInfo).commit;
        return (checkDrivingLicense && checkCreditCard && insertion);
} //Once the user signed up in the system, he will use the psw he has decided, otherwise the system will
generate one
//If the user has not a valid Credit Card or Driving License, we prevent him from registering
```

```
function password(pswGivenByUser : string) : string {
        Random r = new Random();
        if(pswGivenByUser.empty()){
                return r.newRandomString();
        }
        else{
                return pswGivenByUser;
        }
}
```

User interface design

While designing the user Interface, we decided to focus on functionalities of our system, paying attention to the presence of graphical elements rather than their aspect.

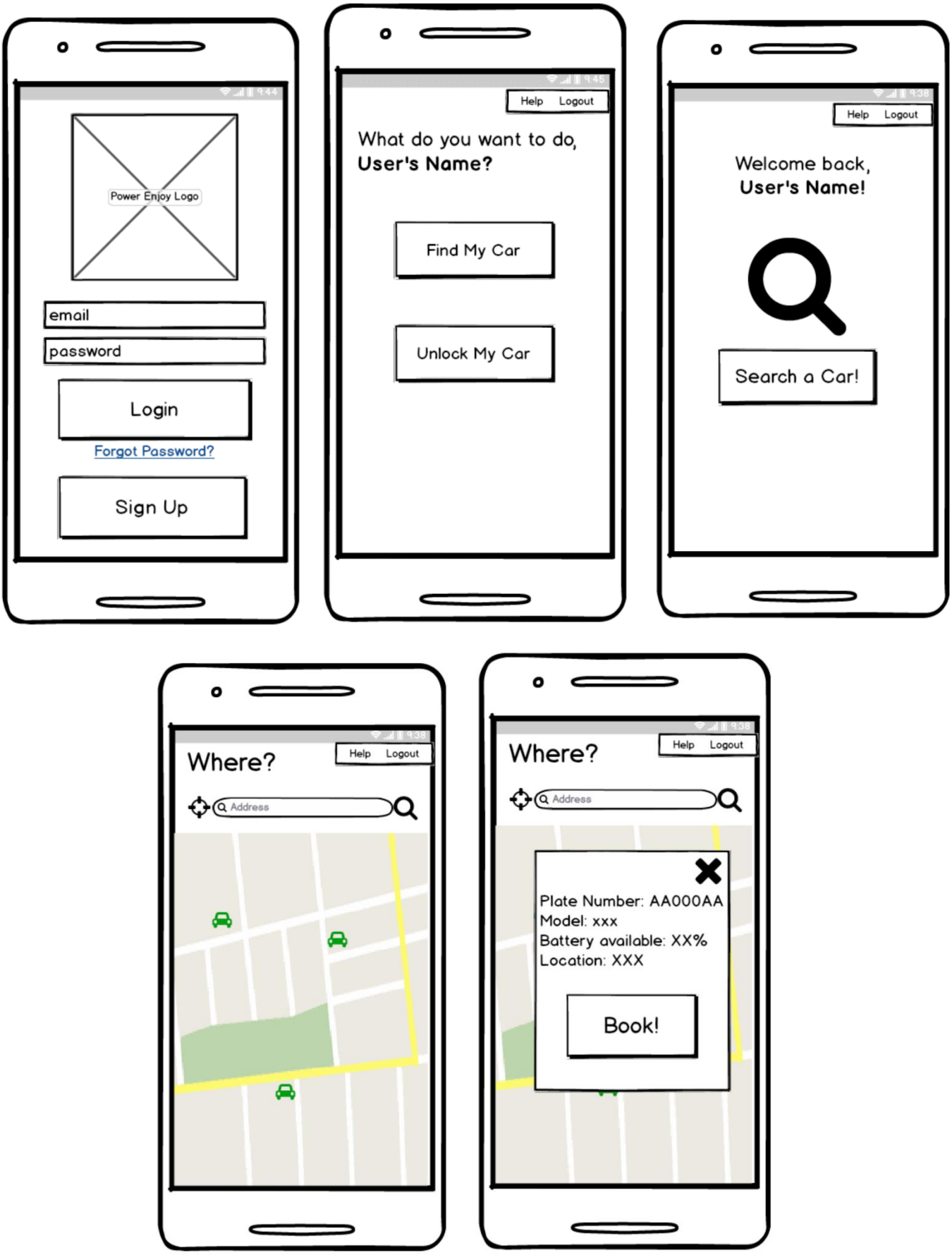
Users' Web Application



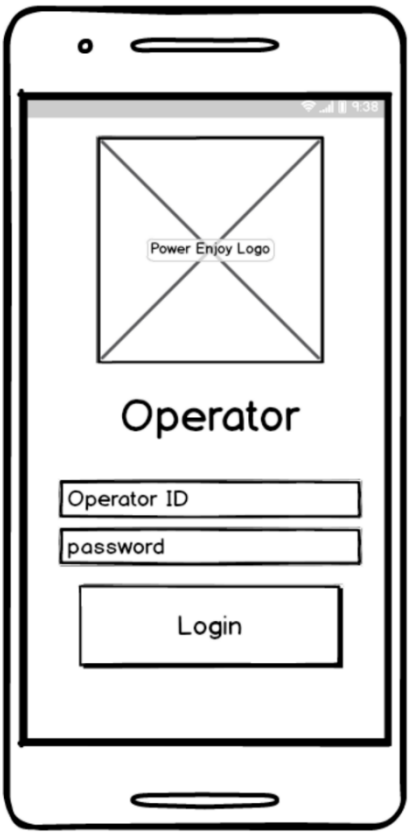
Pietro Crovari,
Enrico Gnecco

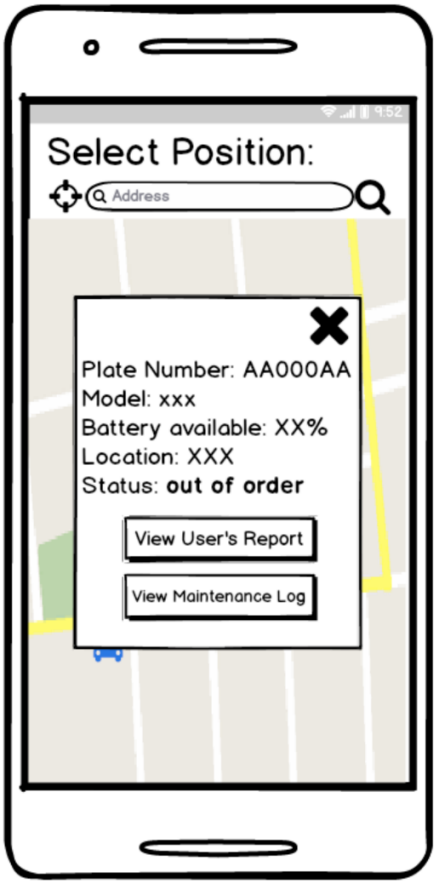

A Web Page
http://powerenjoy.com
Power
Search a Car
Piazza Leonardo da Vinci, Milan
Use Current Position
Plate Number:
AA000AA
Model: xxx
Battery available: XX%
Location: XXX
Book!

Users' Mobile Application



Operators' Mobile Application





Requirements Traceability

Functional Requirements

1	• Algorithms, signUp of the user
2	• Algorithms, signUp of the user
3	• Algorithms, signUp of the user and architectural choice for communicating
4	• Database in the Architecture Schema
5	• Random procedures in PHP libraries
6	• Search Car Sequence Diagram
7	• Mobile Application in the Architecture Schema
8	• Function “startReservationTimer()” in Car Reservation Sequence Diagram
9	• “StartBilling()” function in Start Rent Sequence Diagram
10	• Presence of the display on the car, the amount is calculated directly on the car (as in Start Rent Sequence Diagram) • “CalculateFee()” function in End of Rent Sequence Diagram
11	• Presence of sensors on the car • “carParked()” function in End of Rent Sequence Diagram • “waitExitTime()” and “LockDoor()” functions in End of Rent Sequence Diagram
12	• Notify of correct or missed payement in End of Rent Sequence Diagram
13	• Presence of Safe Areas in the Old Database
14	• External Data Manager Component, with all the relative interfaces
15	• Operators mobile Application Mockup • OperatorManager Component
16	• Operators mobile Application Mockup
17	• Operators mobile Application Mockup
18	• Operators mobile Application Mockup
19	• Operators mobile Application Mockup
20	• “setPendingPayement()” function in End of Rent Sequence Diagram • “checkSuitability()” function in Search Car Sequence Diagram
21	• “changeStatus()” and “updateCarStatus()” functions in End of Rent Sequence Diagram
22	• “checkSuitability()” function in Search Car Sequence Diagram
23	• Condition of state transition on CarStatus State Machine
24	• Different status on the CarStatus state machine
25	• Internal structure of the car

Non-functional Requirements

1	• “StartBilling()” function in Start Rent Sequence Diagram
2	• Internal structure of the car
3	• Fee algorithm is calculated by the server
4	• “Fee algorithm”
5	• “Fee algorithm”
6	• “Fee algorithm”
7	• “Fee algorithm”
8	• “Fee algorithm”
9	• Sequence diagram – start rent
10	• Sequence diagram – end of rent

Effort spent

Crovari: 30 h

Gnecco: 30 h

References

- en.wikipedia.org
- https://beep.metid.polimi.it/polimi

Revisioning Versions

- V1.2:
    - Added Tier Diagram and explanation;
    - Added Revisioning Section;