POLITECNICO DI MILANO

COMPUTER SCIENCE AND ENGINEERING

MASTER OF SCIENCE

DD

PowerEnjoy

Software engineering II project

Giovanni Agugini, Matteo Foglio, Tommaso Massari

Contents

1 Introduction

1.1 Purpose

By proposing this Design Document, we aim at defining the software of the service Power Enjoy and its components in a more precise and deeper way. Hence, our purpose is to show how and why we took specific design decisions related to the structure(s) of the software, the implementation of some algorithmic problems and the user interface of the service.
This document is adressed to developers, customers and stakeholders and is determined to provide:

- high level architecture
- design patterns used to improve quality and efficiency
- runtime behavior of critical parts
- user experience interface
- database schema

1.2 Scope

The system we are projecting is the software of Power Enjoy, an electric car-sharing service which is based on a web application.
There are two different target of people involved in the system:

- the clients
- the operators

Both of them interact with the system via web application through PCs, tablets and smartphones. The system allows users to reserve an electric car via web app, using their GPS position or manually inserting a specific address to find a car in the same area. The system also interacts with users through the car's on-board computer which gives to clients several information such as traffic indications, discounts, fees and so forth. The clients have to register to the service by uploading an official document, a driving license and a credit card through an apposite module on the web app. Instead, operators access the system in order to approve registration requests from clients. (The system also includes some IoT software features, given that the cars are considered as smart objects with sensors and their own computing engine)
Our main goals are to get an efficient, scalable and usable application and to provide a service which is sustainable for the environment. To achieve these goals, we need to get the user involved in the service by rewarding well behaviours and punishing bad ones (see definitions below).

1.3 Definitions, acronyms, abbreviations

- RAS Document: Requirements Analysis and Specifications Document
- DD: Design Document
- API: Application Programming Interface: it is a common way to communicate with another system
- MVC: Model View Control (pattern)
- Push notification: it is a notification sent to a smartphone using the mobile application, so it must be installed
- Well behaviours: is a set of good actions taken by users while using the service. They include helping the service by minimising the intervention of operators and preventing pollution issues
- Bad behaviours: is a set of bad actions taken by users while using the service. They include leaving the car at more than 3km to the nearest power grid station and leaving the car with more than 80% of battery empty
- REST: REpresantional State Transfer
- RESTful: REST with no session
- ETA: estimated time available; it is the time the taxi needs to arrive to client starting position
- Safe Area: is a set of delimeted positions where users can end the rental
- Unsafe Area: every area that is not a Safe Area
- ER: Entity Relation
- UX: User eXperience design
- BCE: Boundary Control Entity

1.4 Reference documents

- RAS Document produced before
- Assignment Document

1.5 Document structure

- **Introduction**: introduces the design document. It contains a justification of its utility and indications on which parts are covered in this document not included in the RAS Document
- **Architecture Design**: this section is divided into different parts:
    1. Overview : this sections explains the division in tiers of our application
    2. High level components and their interaction : this sections gives a global view of the components of the application and how they communicate
    3. Component view : this sections gives a more detailed view of the components of the applications
    4. Deploying view : this section shows the components that must be deployed to have the application running correctly
    5. Runtime view : sequence diagrams are represented in this section to show the course of the different tasks of our application
    6. Component interfaces : the interfaces between the components are presented in this section
    7. Selected architectural styles and patterns : this section explain the architectural choices taken during the creation of the application
    8. Other design decisions
- **Algorithms Design**: this section describes the most critical parts via some algorithms. Pseudo code is used in order to hide unnecessary implementation details with the purpose of focusing on the most important parts
- **User Interface Design**: this section presents mockups and user experience explained via UX and BCE diagrams
- **Database schema**
- **Requirements Traceability**: this section aims to explain how the decisions taken in the RAS Document are linked to design elements
- **Hours of work:** this section shows how much effort (in terms of hours) each member group has spent in developing the Design Document

2 Architectural design

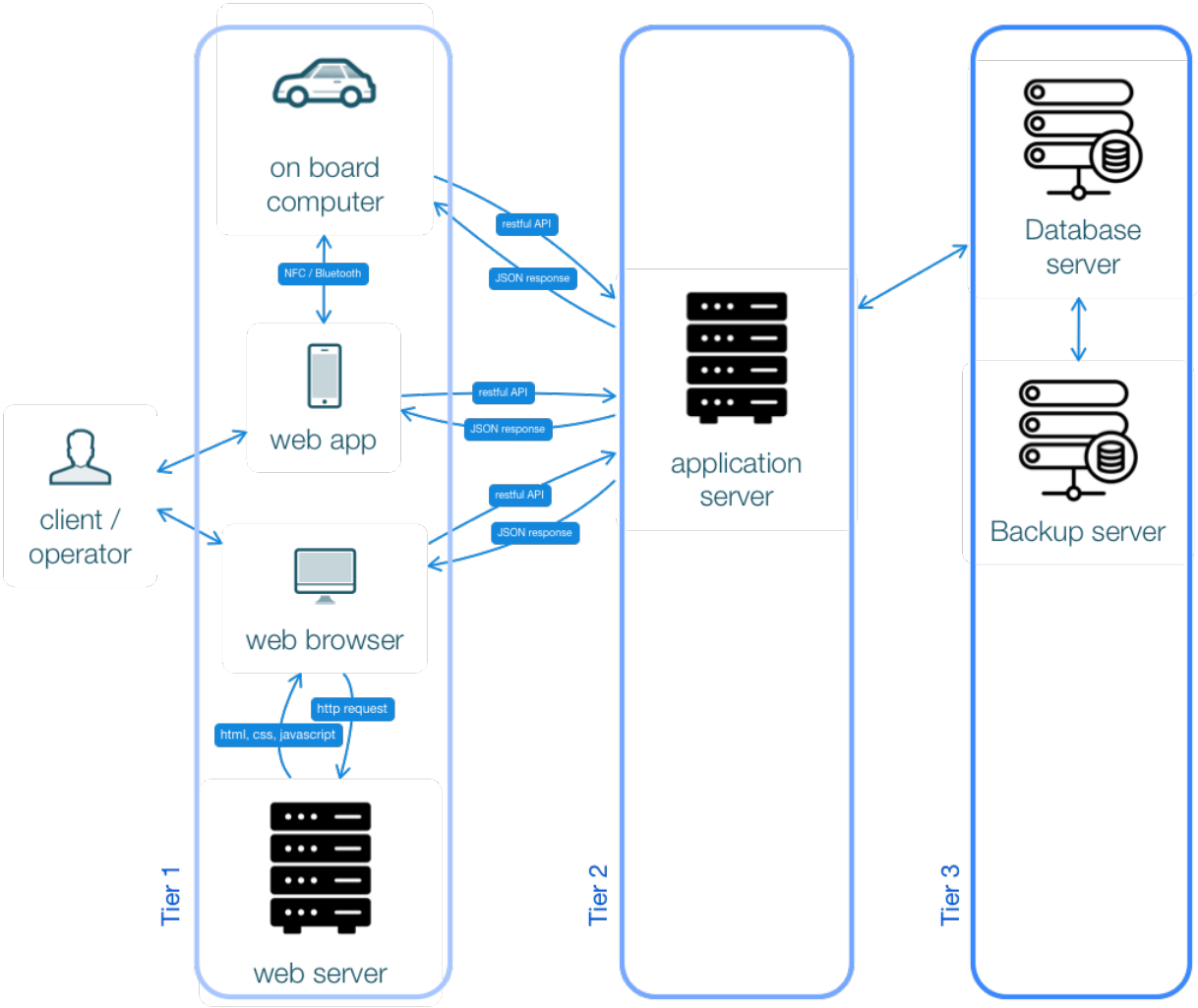
2.1 Overview

The PowerEnjoy adopts a 3 tiers and 3 layers architecture.

- Tier 1: in this tier we collocated the whole Data layer. It's composed by a database server and a backup server for further security and the ability to restore the system after a fault. The backup server communicates only with the DB server and the DB server communicates with the upper tier.
- Tier 2: in this tier we collocated the Application layer. It communicates with the DB server of the tier 3 and with devices of the layer 1.
- Tier 3: this tier is dedicated to the presentation layer. It's composed by:
  - dedicated app running on the on-board computer: this device is in fact a tablet running Android 4+. It can communicate with the web app installed on a proper device through NFC or Bluetooth
  - web app on a smartphone or tablet
  - web browser supported by the web server

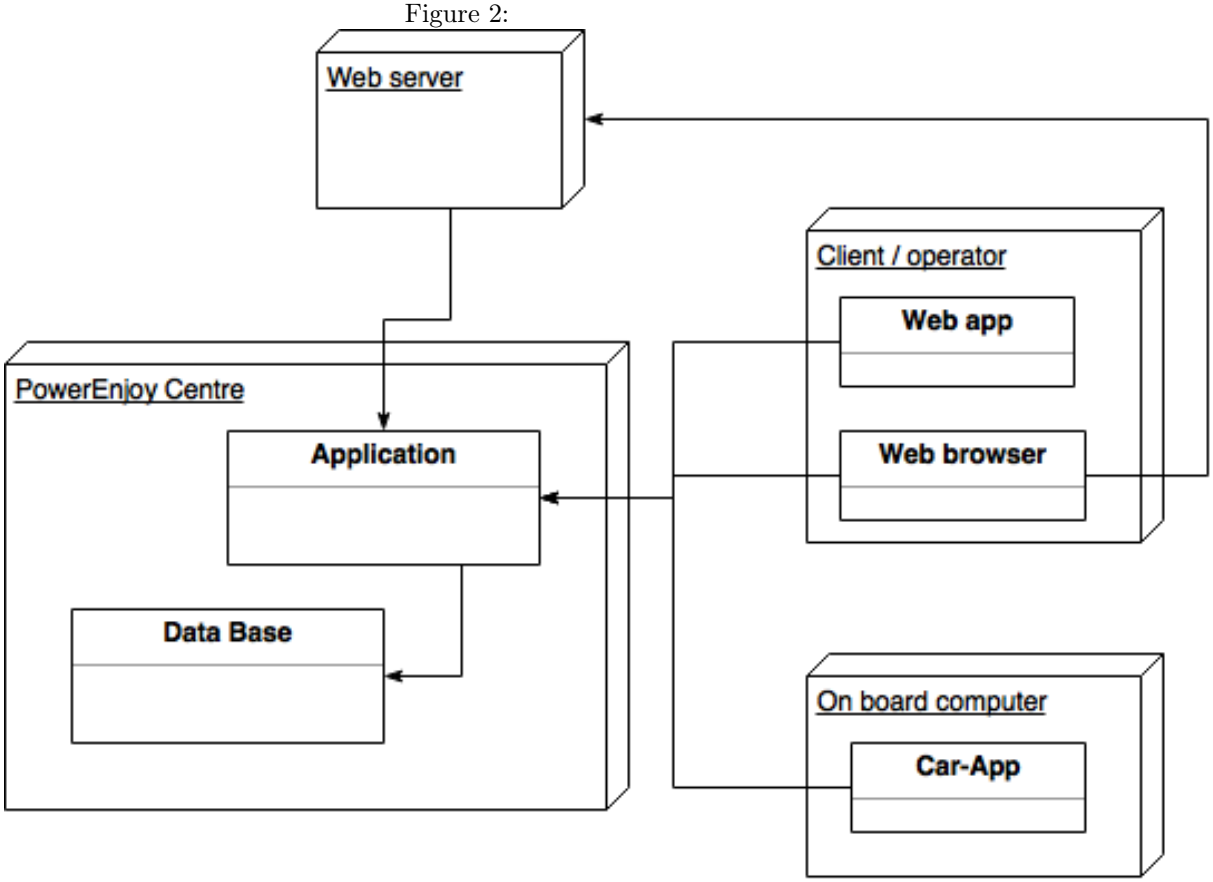
In particular the communication between the application layer and the presentation layer occurs through Resful API requests and JSON responses.

Figure 1: General Atrchitecture



2.2 High level components and their interaction

Figure 2:

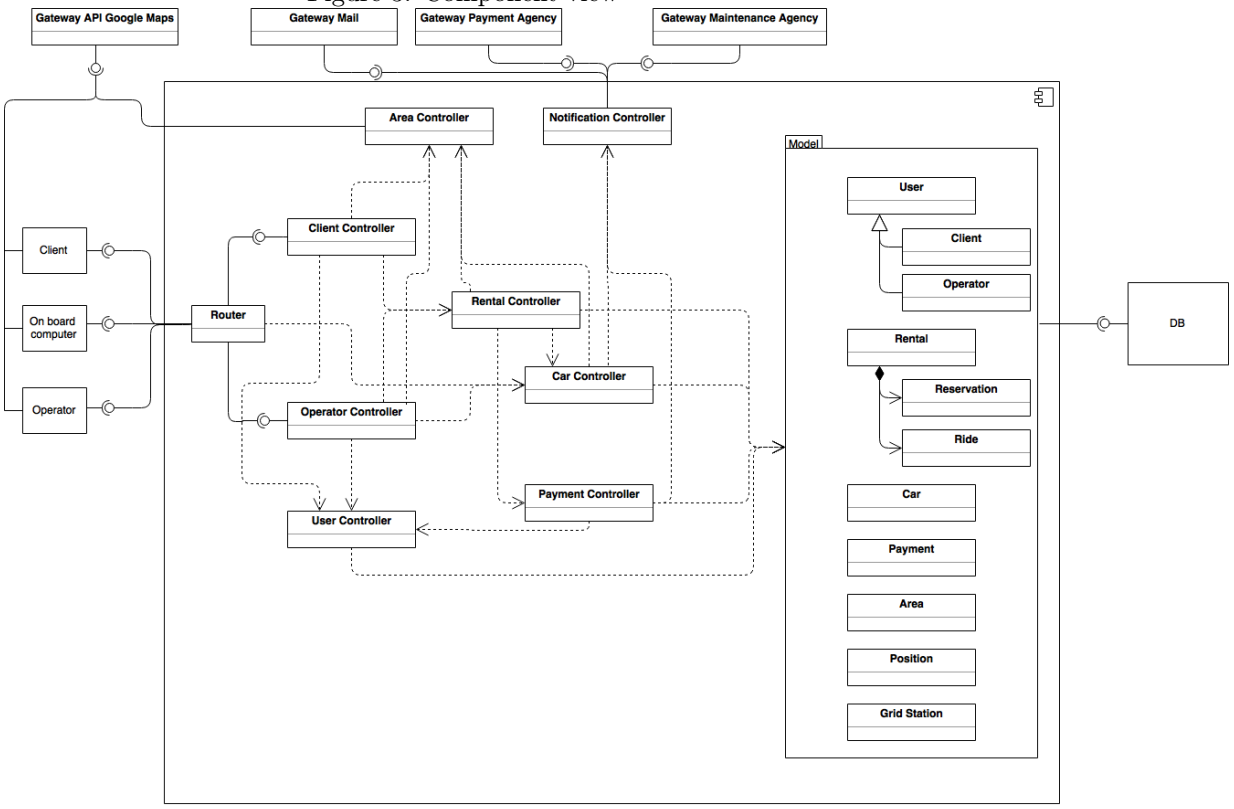


Our goal was to design a system that can be easily improved and expanded in a second time. In order to achieve this result we decided to create different components that can be easily modified with full transparency to the others.
First, the use of thin clients allow us to improve software UX for different devices without any change of the application logic. This structure gives also another advantage: in fact the concentration of the whole application logic allows to reduce the need of updating the apps on the on-board computers and client devices.
Moreover this gives us the chance to reduce security controls to the communications between the several devices (representing the presentation layer) and the unique PowerEnjoy Centre in which all data are stored. The use of a central and unique system for storage of the data allows us to easily back them up by using a dedicated server connected to the DBMS and to avoid problems of data inconsistency due to synchronization issues.

Scalability of the system is supported by the design of different components that can be moved to different devices if more performance are needed due to an expansion of PowerEnjoy business.
User devices are provided of dynamical GUI that is updated with information coming from asynchronous requests to the main PowerEnjoy Centre. The choice of developing a Web App to be installed on clients' and operators' devices leads to a reduction of the project cost: we can design a single app that can be run in different environment such as Android, Ios, Windows or by the use of a web browser. The portability is supported by the use of Phonegap, an open source distribution of Cordova.

2.3 Component view

Figure 3: Component view



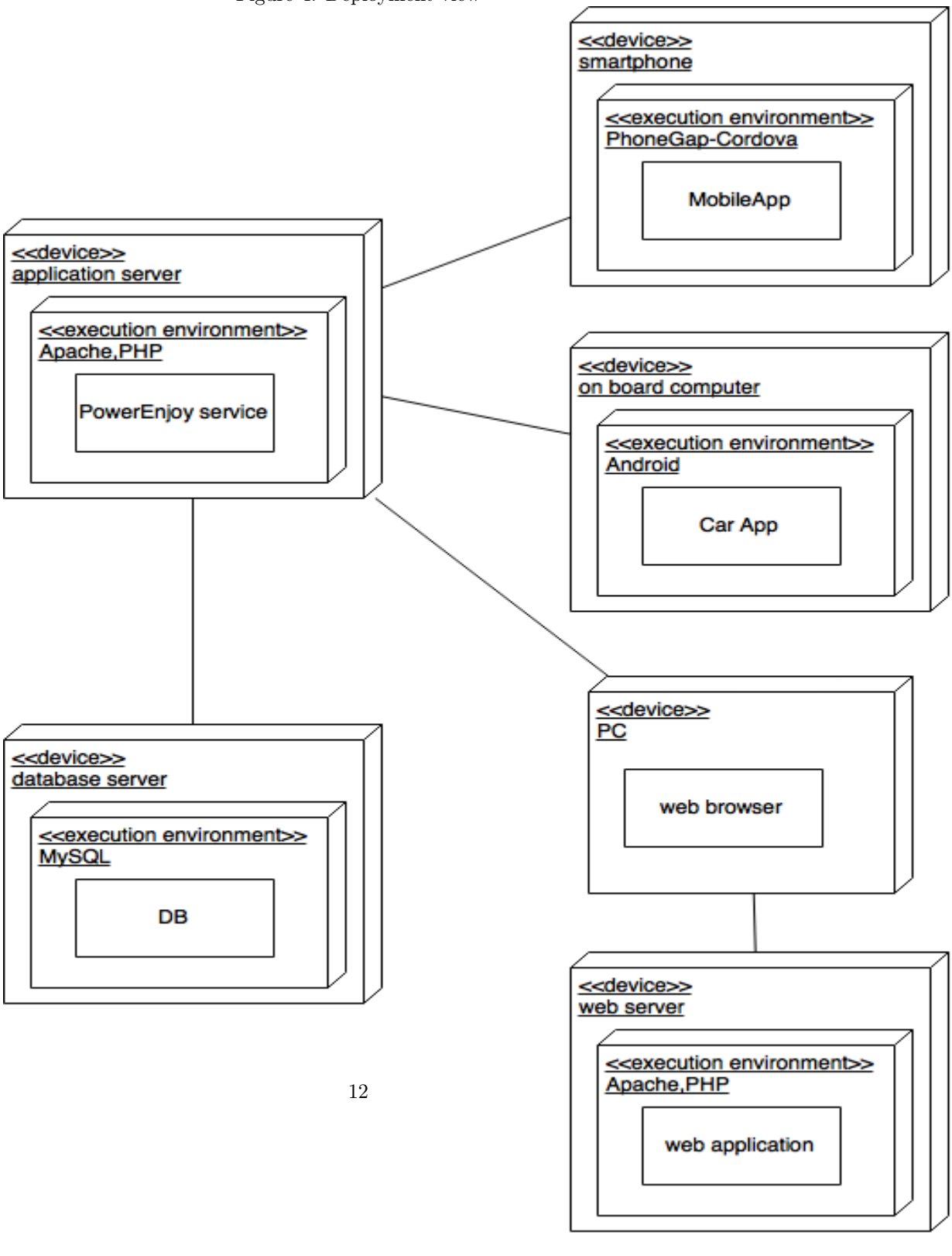
- Model: our representation of the world, the data we interact with
- Router: it manages all the incoming requests from our users, clients and operators, and it routes them to the Client Controller or the Operator Controller, according to the request type. It includes a firewall

- Client Controller: it manages the client requests, this includes a legality checks. It communicates with User Controller when the request type is about reading or modifying user data (registration, login, document upload, etc). It communicates with the Rental Controller when the request type is rental (reservation, door opening, drive, end reservation, end drive, etc)
- Operator Controller: it manages the operator requests, this includes the legality checks. It communicates with the User Controller when the request type is about modifying data (log in, client validation, edit data, etc). It communcates with the Rental Controller when the request type is about a rental (car moving, car charging, etc). It communicates with the Car Controller when the request type is about cars (i.e. car state change)
- User Controller: it manages users data of both clients and operators
- Payment Controller: it manages the payments and the communication with the external Payment Agency creating outgoing messages and interpreting incoming messages. All the messages passes through the Notification Controller. It comunicates with the User Controller to block a user in case of unsuccessful payment
- Rental Controller: it receives requests from Client Controller and Operator Controller. Moreover it manages the rental in general by communicating with the Model, the Payment Controller and the Car Controller
- Car Controller: it manages the cars and the communication with the Maintenance Agency, creating outgoing messages and interpreting incoming messages. All the messages passes through the Notification Controller
- Notification Controller: it haldles all the incoming messages from the partner agencies and outgoing ones. It allows us to have just one interface to the external world for security reasons and it includes a firewall
- Area Controller: it manages all the controls and computations related to areas and positions
- Client: the client's device (web-app or web browser)
- Operator: the operator's device (web-app or web browser)
- On board computer: it's the device installed onto the cars
- DB: the database where the persistent data are stored
- Gateway Payment Agency: it manages the messages with the Payment Agency
- Gateway Maintenance Agency: it manages the messages with the Maintenance Agency

- Gateway API Google Maps:
- Gateway Mail: it manages the sending of e-mails to clients such as payment confirmation e-mails or payment request e-mails

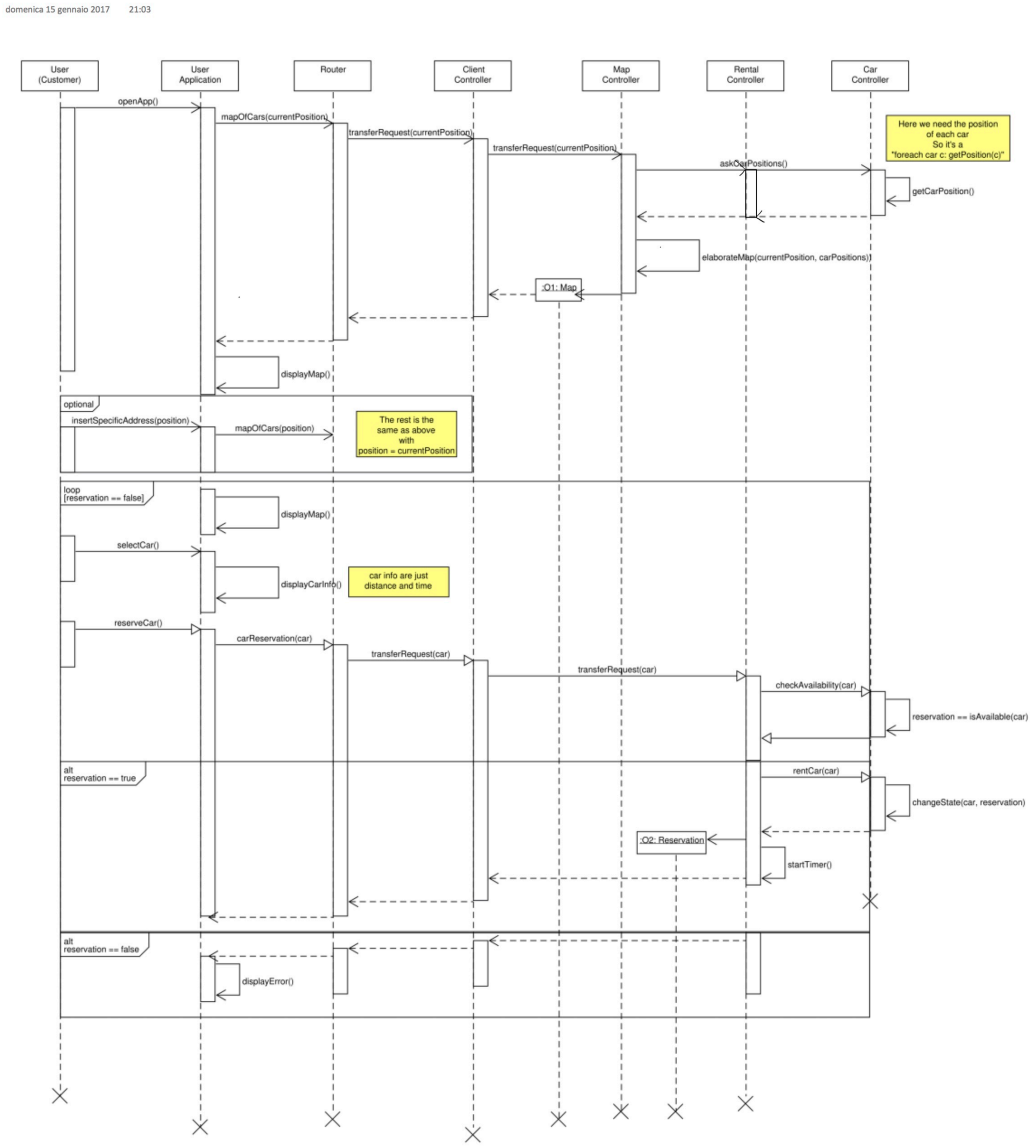
2.4 Deploying view

Figure 4: Deployment view



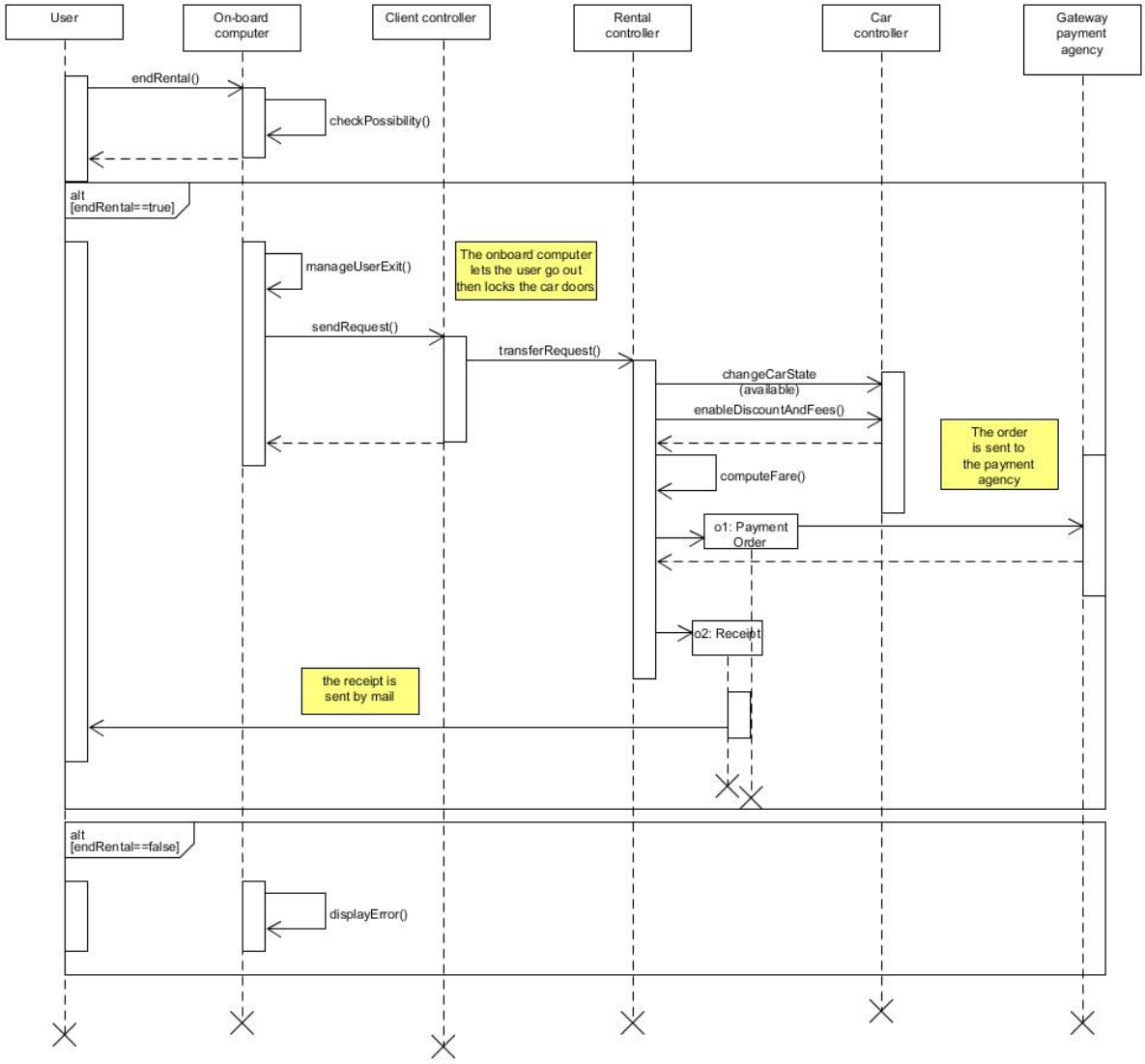
2.5 Runtime view

Figure 5: reservation sequence diagram



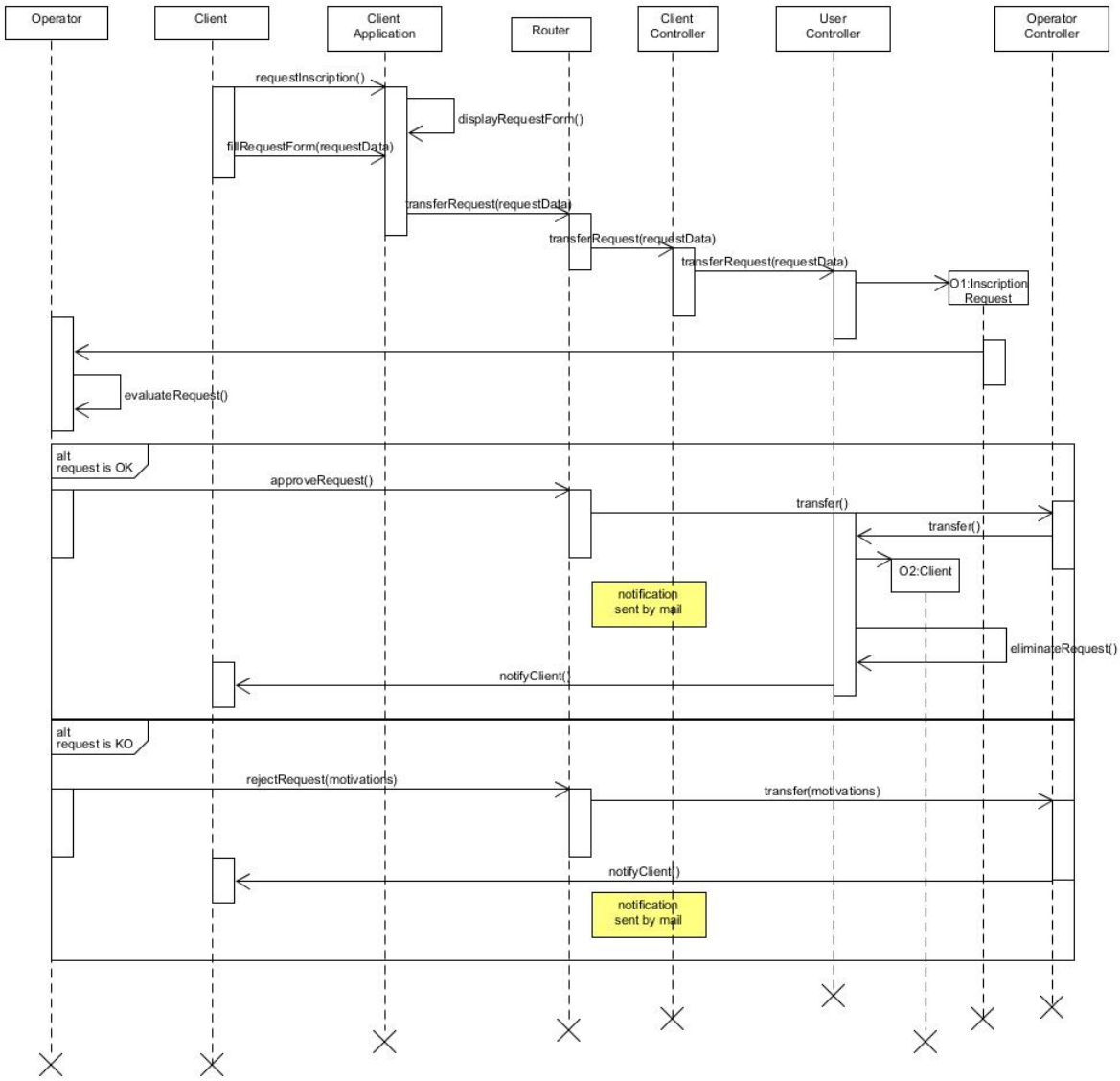
In this sequence diagram you can see how a reservation process is made by our system. Assuming that the user is already logged in, he opens the app and sends his/her current location to the Router that passes the request to the Client Controller. The Client Controller passes the parameter to the Map Controller which interrogates the Car Controller to know the position of each car. Afterwards the Map Controller elaborates a map for the client. The client selects a car among those shown on the map and decides to rent it: in this case the Rental Controller has to ask the Car Controller if the car is still available. If the answer is yes, a new reservation is made; unless the app returns to show the map

Figure 6: endRental sequence diagram



In this sequence diagram we show how software components interact with each other when a client decides to end the rental. First of all, when the client wants to end the rental, the On-board Computer of the car checks if it is possible to end the rental (e.g. the car is in a safe area). If the check goes well the system lets the client go out the car and locks the doors, then the On-board Computer sends the rental data (e.g. the price of the ride without discounts and fees, indications of well/bad behaviuours) to the Car Controller, which passes the data to the Rental Controller and changes the state of the car to "Available". The Rental Controller has the function of applying possible discounts and fees and creating a Payment Order with the final price. The Payment Order passes through the Notification Controller which sends the order to the Gateway Payment Agency. Assuming in this case that the payment goes well, the Gateway Payment Agency returns the positive answer of the Payment Agency to the Notification Controller who sends it back to the Rental Controller. This one creates a Receipt of the rental that is sent to the user via mail. If the check doesn't go well, the On-board Computer displays an error to the client (e.g. "you cannot end the rental in your situation").

Figure 7: inscription Request Sequence Diagram



In this sequence diagram it is shown the process of approving inscription requests from clients. The Client demands the inscription through the Client Application which shows the request form. The client fills the request and the Client Application sends it to the Router that sorts it to the Client Controller. The Client Controller instantiates a new object called “Inscription Request” . This object is sent to the Operator who decides to approve or reject the request. If the Operator approves the request he/she sends to the Operator Controller through the Router the positive answer. The Operator Controller deletes the object and creates a new object called “Client” to be stored in the Model. Then the approval is communicated to the Client through notification via mail. Otherwise, if the Operator rejectes the request, the negative answer is before passed to the Operator Controller through the request and after notified to the Client by mail.

2.6 Selected architectural styles and patterns

2.6.1 Overall Architecture

Our application is divided into 3 tiers:

1. Client: thin client
2. Application Logic
3. DataBase

2.6.2 Protocols

The proposed system relies on the following protocols:

- PDO (PHP Data Objects) used by the application server to communicate with the DBMS -
- MySQL
- Restful API with JSON used for the communication between users' app or mobile browser and the PowerEnjoy centre. To ensure the proper level of security, these requests can rely of the SSL protocol

2.6.3 Design patterns

MVC We adopt the Model-View-Controller as explained in the architectural considerations

Visitor We use the visitor pattern to recognize the the all the different request types so can be redirected to the dedicated controller.

Client - server The communication between tier 1 and 2 is entirely based on client-server model. The client can be a user web-app or web browser, eather a client's or an operator's. The client is a thin one and it just handles the presentation layer. In the case of the web browser the presentation is distributed between the web browser and the web server. All the application logic is executed by the server in the application layer. This decision leads to the following advantages:

- the application can run on low-resources devices
- maintenability: high maintenability due to the unique server
- security: no access to data without the central server approval
- scalability: the number of connected clients can be scaled

2.7 Other design decisions

All the map services, such as the map visualization and navigation, are provided by Google Maps APIs for the web app and the web browser as well as the on board computer. Our system is only in charge of managing the set of areas and the uniform distribution of cars.

To develop the web application we will use

- PhoneGap: it's a distribution of Cordova powerd by Adobe
- JQuery and Bootstrap frameworks

3 Algorithm design

3.1 Uniform distribution

```
# This function returns the destination to guarantee an uniform distribution.
# It is given the user destination userDestination and a graph safeAreasGraph,
# which contains safe areas as macro zones of the city. Each safe area contains
# a vector of car parked in that area and a number of cars required to guarantee an uniform distrubution
# (this number may vary from area to area, depending on the density of usage per zone
# and other factors such as traffic issues).The algorithm is a BFS algorithm with the variant that
# if the number of cars parked in a specified vertex of the graph is less than the number required for
# a uniform distribution, then the function returns a destination in that vertex (safeArea)

function uniformDistribution(userDestination, safeAreasGraph){

    # WHITE = not explored;
    # GREY = just explored;
    # BLACK = explored --> means that every adjacent node is explored;
    variable List greyList; # this is the list that contains all the grey nodes
    variable SafeArea tmpArea;
    variable SafeArea[] adjacentAreas;

    # finds the safeArea of the userDestination
    foreach(safeAreasList as safeArea){
        if(userDestination is in safeArea){
            tmpArea = safeArea;
            break;
        }
    }

    # if the tmpArea has not a number of cars that is correct for guaranteeing an uniform distribution
    # the function returns the userDestination as a correct destination for the uniform distribution
    if(tmpArea.vectorCars.lenght() < tmpArea.distributionNumber)
        return userDestination;
    else{

        foreach (safeArea in safeAreasGraph.getSafeArea()){
        color(safeArea)= WHITE;
        distance(safeArea) = infinity;
```

```
        predecessor(safeArea) = NIL;
        }

        color(tmpArea) = GREY;
        distance(tmpArea)=0;
        predecessor(tmpArea) = NIL;
        greyList.add(tmpArea);

        while (greyList != empty){

            tmpArea = greyList.head(); # the head of the list, assuming that is a FIFO list

            adjacentAreas = safeAreasList.getAdjacent(tmpArea);
            foreach(adjacentAreas as safeArea){
                if(safeArea.vectorCars.lenght() < safeArea.distributionNumber){
                    return generateDestination(safeArea);
                    break;
                }
                else{
                    if(color(safeArea)= WHITE){
                        color(safeArea)= GREY;
                        distance(safeArea) = distance(tmpArea) + 1;
                        predecessor(safeArea)= tmpArea;
                        ENQUEUE(greyList, safeArea);
                        }
                    }
            }
            DEQUEUE(greyList); # removed from the queue: it has already been expanded
            color(tmpArea) = BLACK;
        }

    function generateDestination(safeArea){..}
    # This function generates a parking in a safe area depending on the position of the other cars
    # in the area and it is linked to some GPS features
```

3.2 Make ride and calculate fees

```
# The function checks if the boolean are true and assigns the discount or fee associated to
# behaviour to the final price.
function calculateDiscountsAndFees(priceWithoutDiscountsAndFees, indicationsAboutDiscounts, indicationsAboutFees){

    variable finalPrice;
    finalPrice = priceWithoutDiscountsAndFees;
    foreach( indicationsAboutDiscounts as behaviour){
        if(behaviour.indication == 'UNIFORM + CHARGE'&& behaviour.state == true){
            finalPrice = finalPrice - 0.35*finalPrice;
            break;
        }
        if(behaviour.indication == 'CHARGE'&& behaviour.state == true){
            finalPrice = finalPrice - 0.3*finalPrice;
            break;
        }
        if(behaviour.indication == 'BATTERY 50'&& behaviour.state == true){
            finalPrice = finalPrice - 0.2*finalPrice;
            break;
        }
        if(behaviour.indication == 'PASSENGERS'&& behaviour.state == true){
            finalPrice = finalPrice - 0.1*finalPrice;
            break;
        }
    }
    foreach( indicationsAboutFees as behaviour){
        if(behaviour.indication == 'BATTERY 80'&& behaviour.state == true){
            finalPrice = finalPrice + 0.3*finalPrice;
            break;
        }
        if(behaviour.indication == '3KM'&& behaviour.state == true){
            finalPrice = finalPrice + 0.3*finalPrice;
            break;
        }
    }
```

4 User interface design

4.1 UX diagrams

Figure 8: Ux client

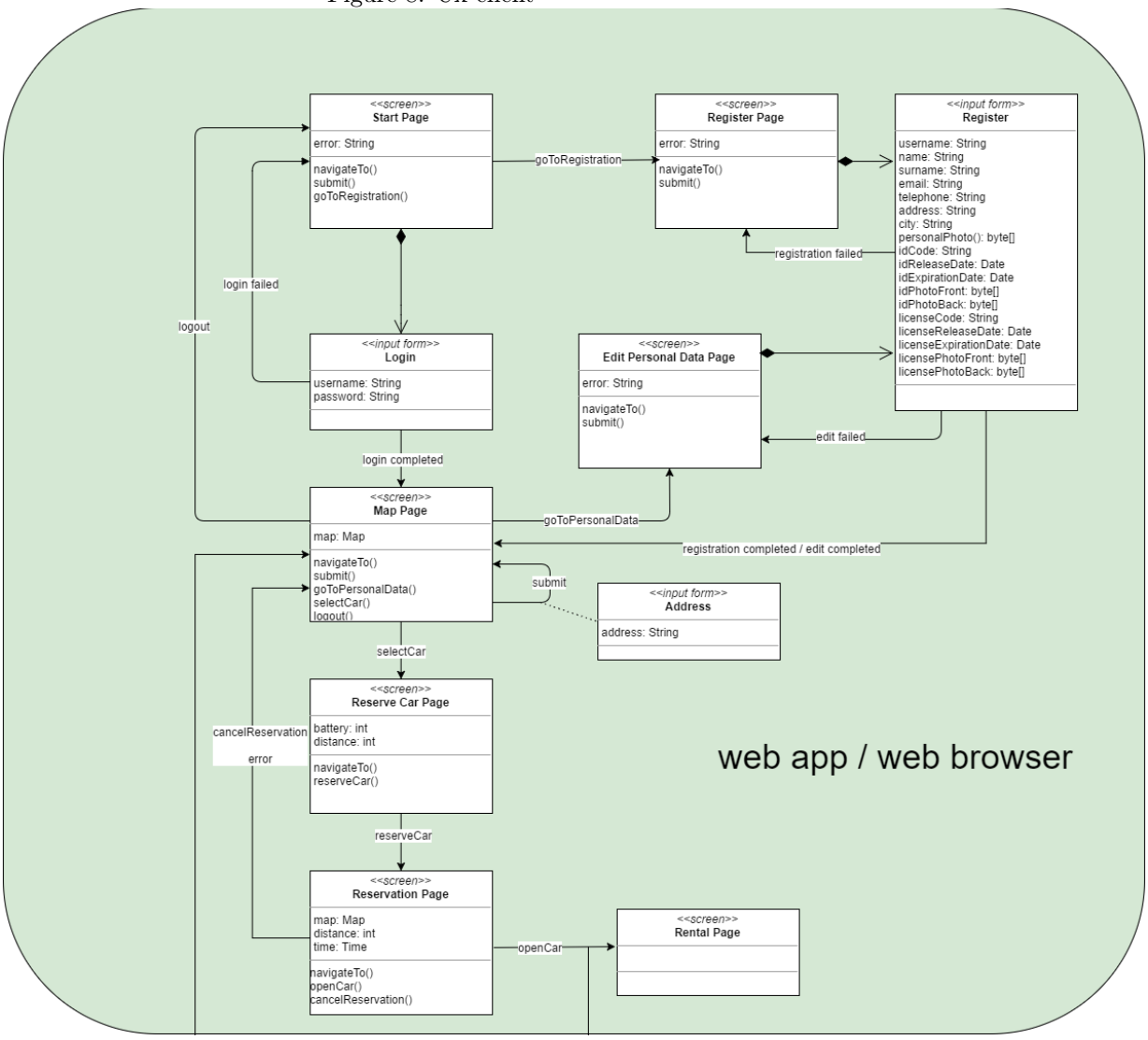


Figure 9: part 2

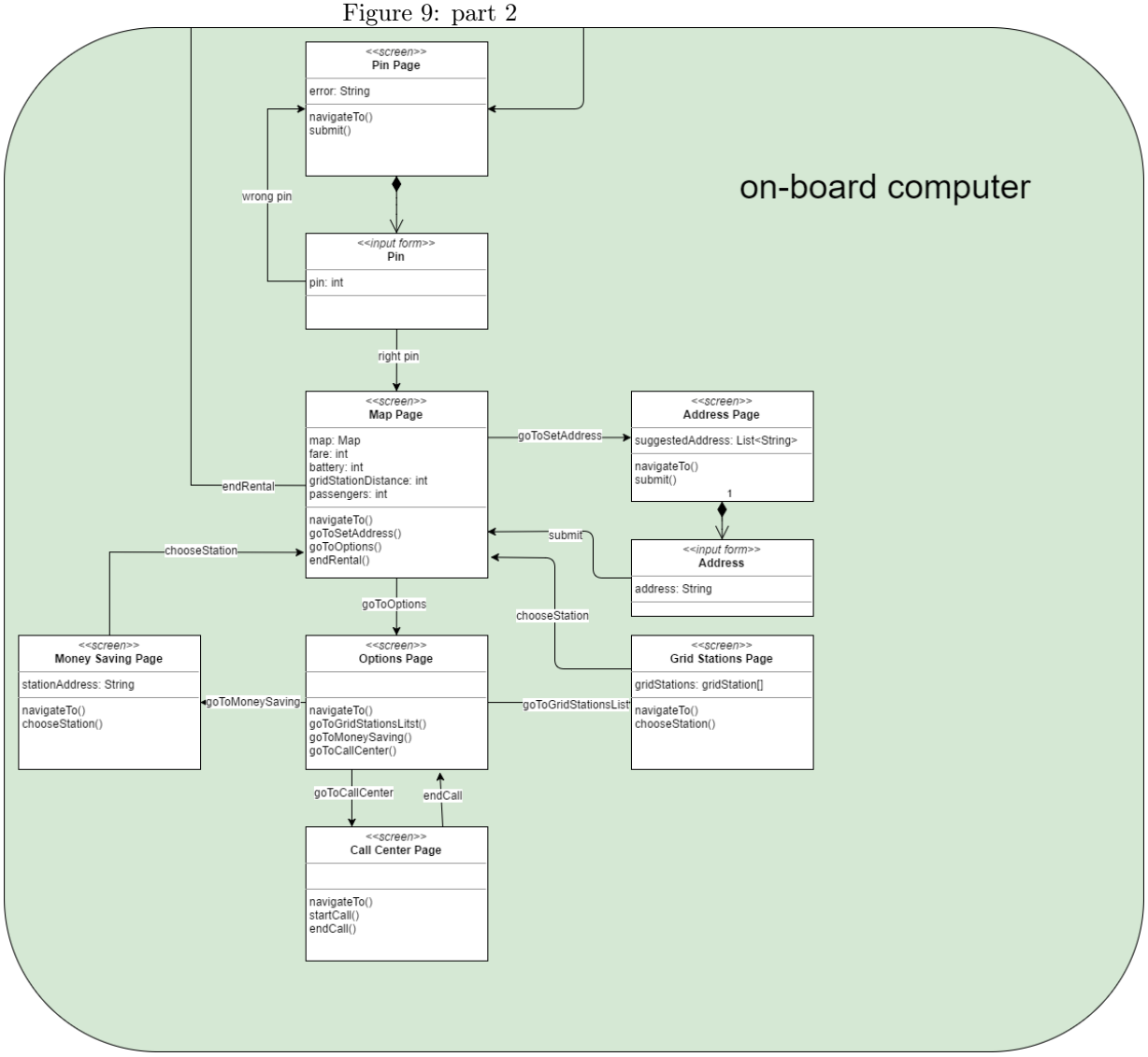


Figure 10: UX operator

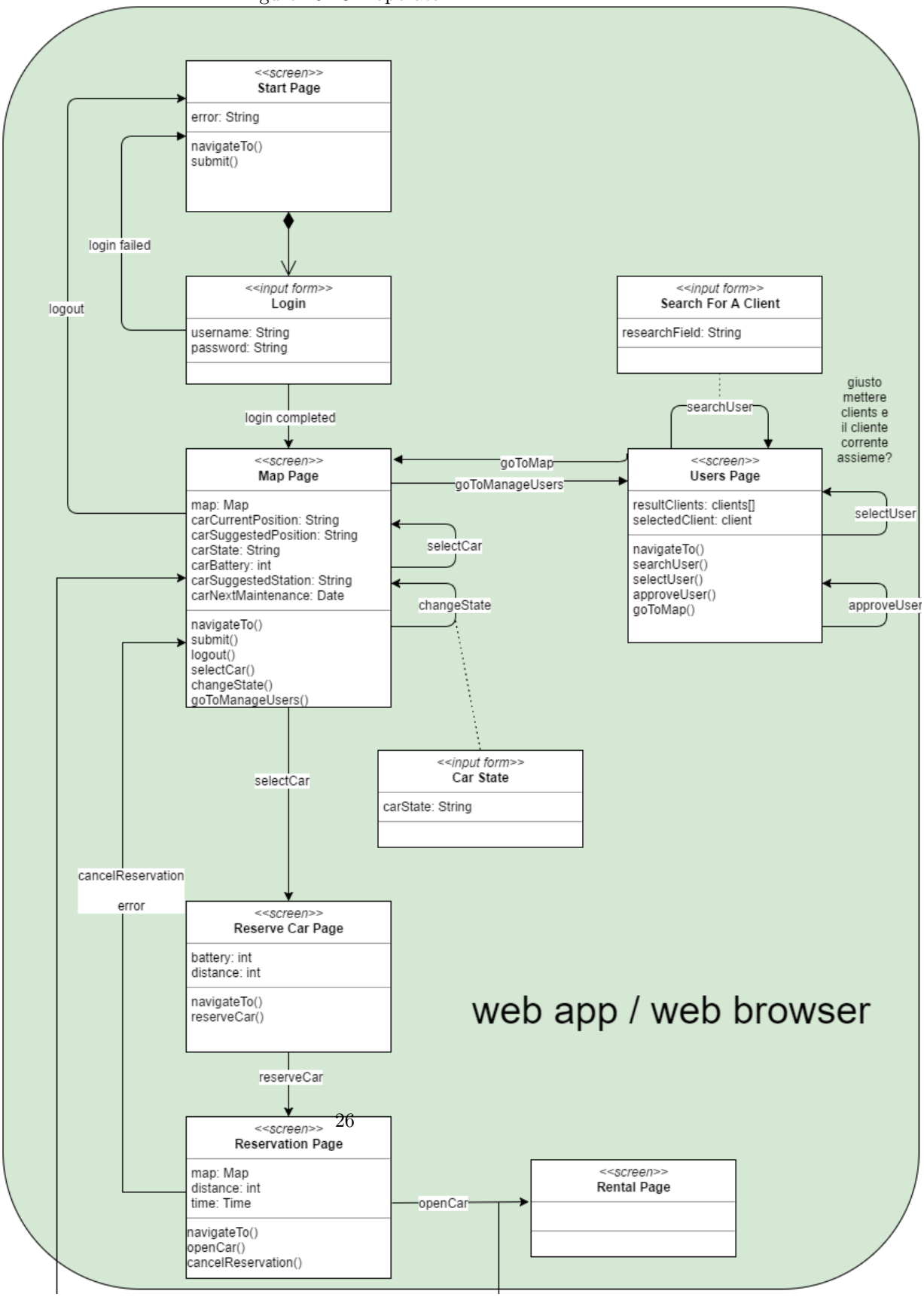
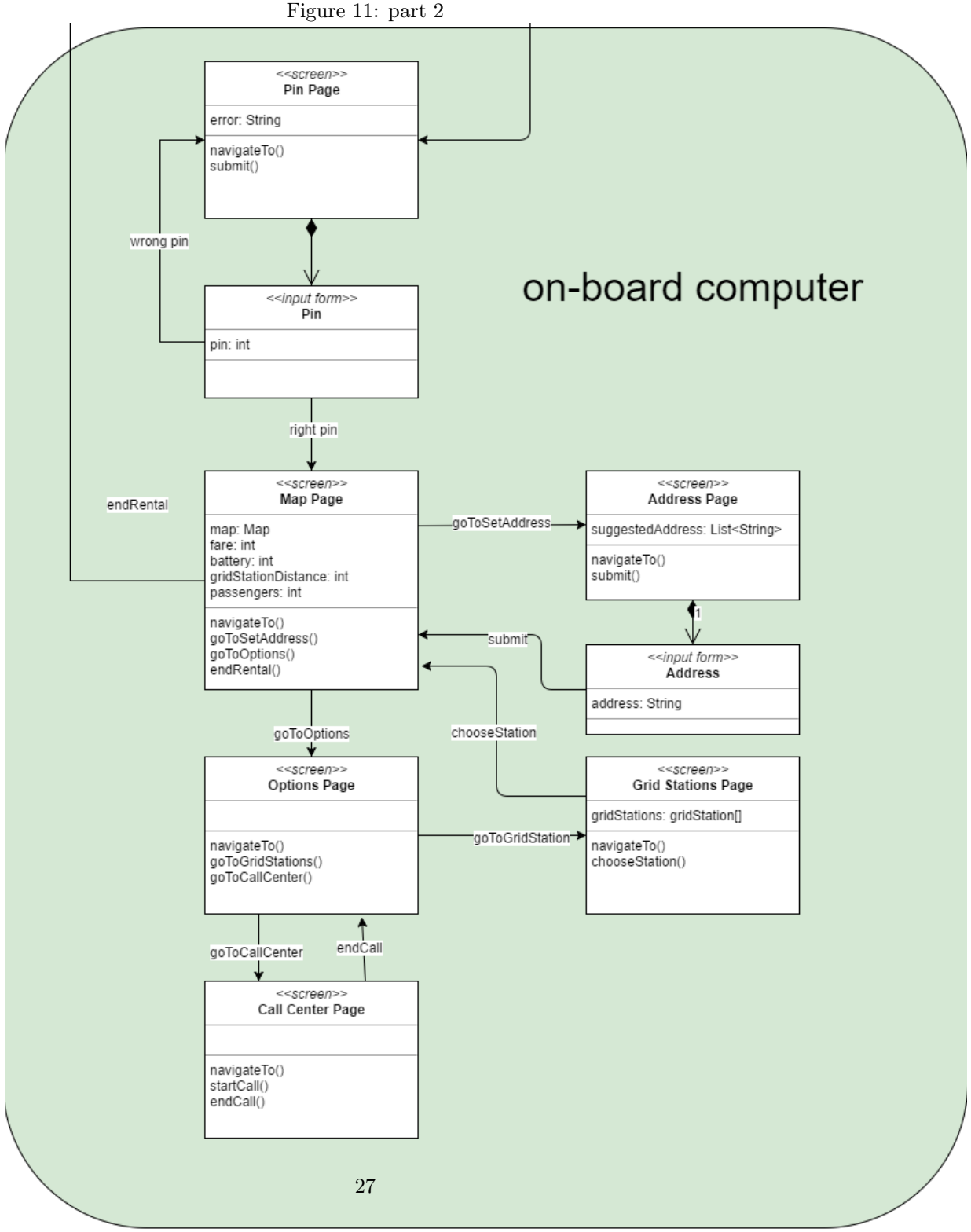


Figure 11: part 2



4.2 Screen Boundary Correspondency diagrams

Figure 12: SB client

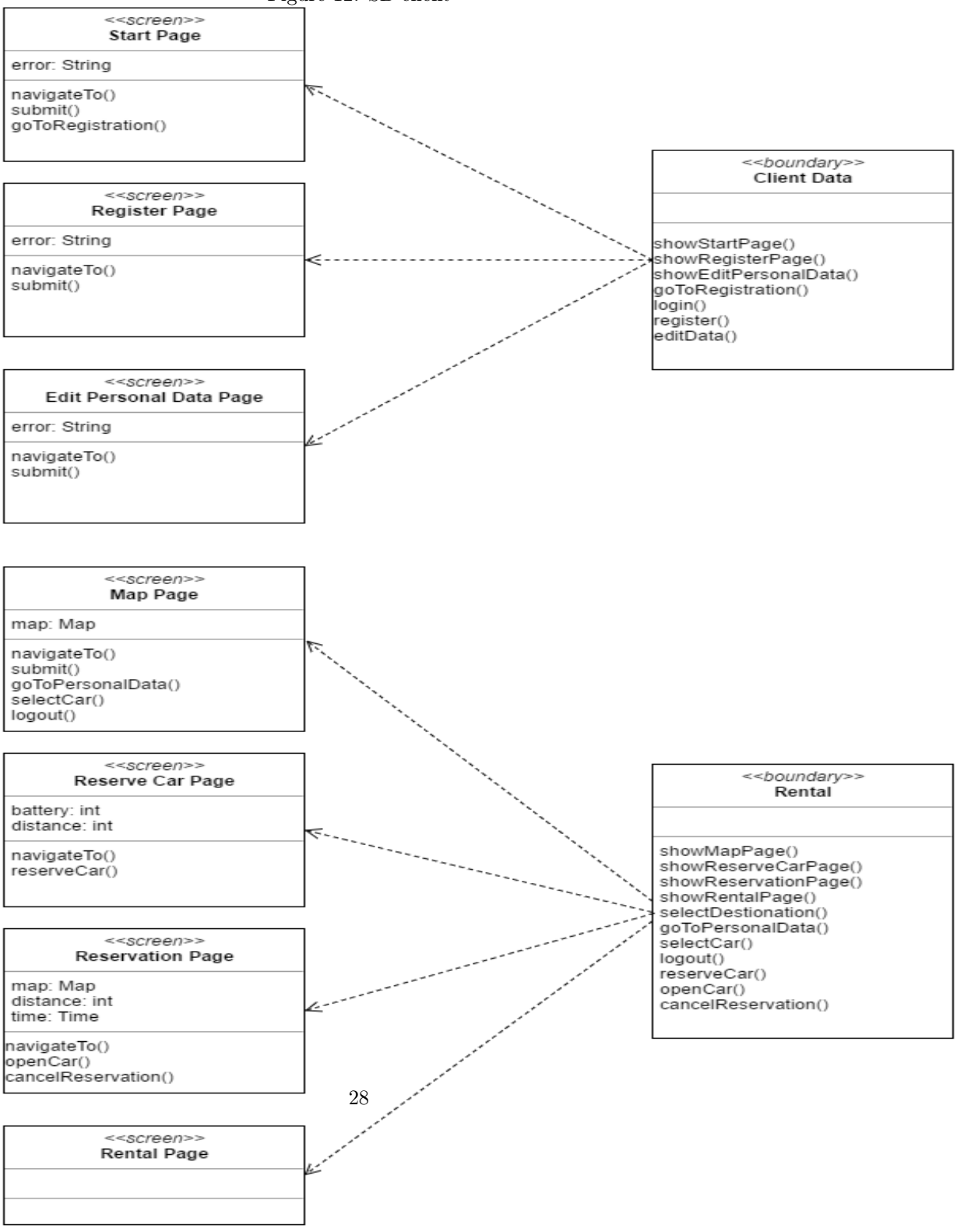


Figure 13: part 2

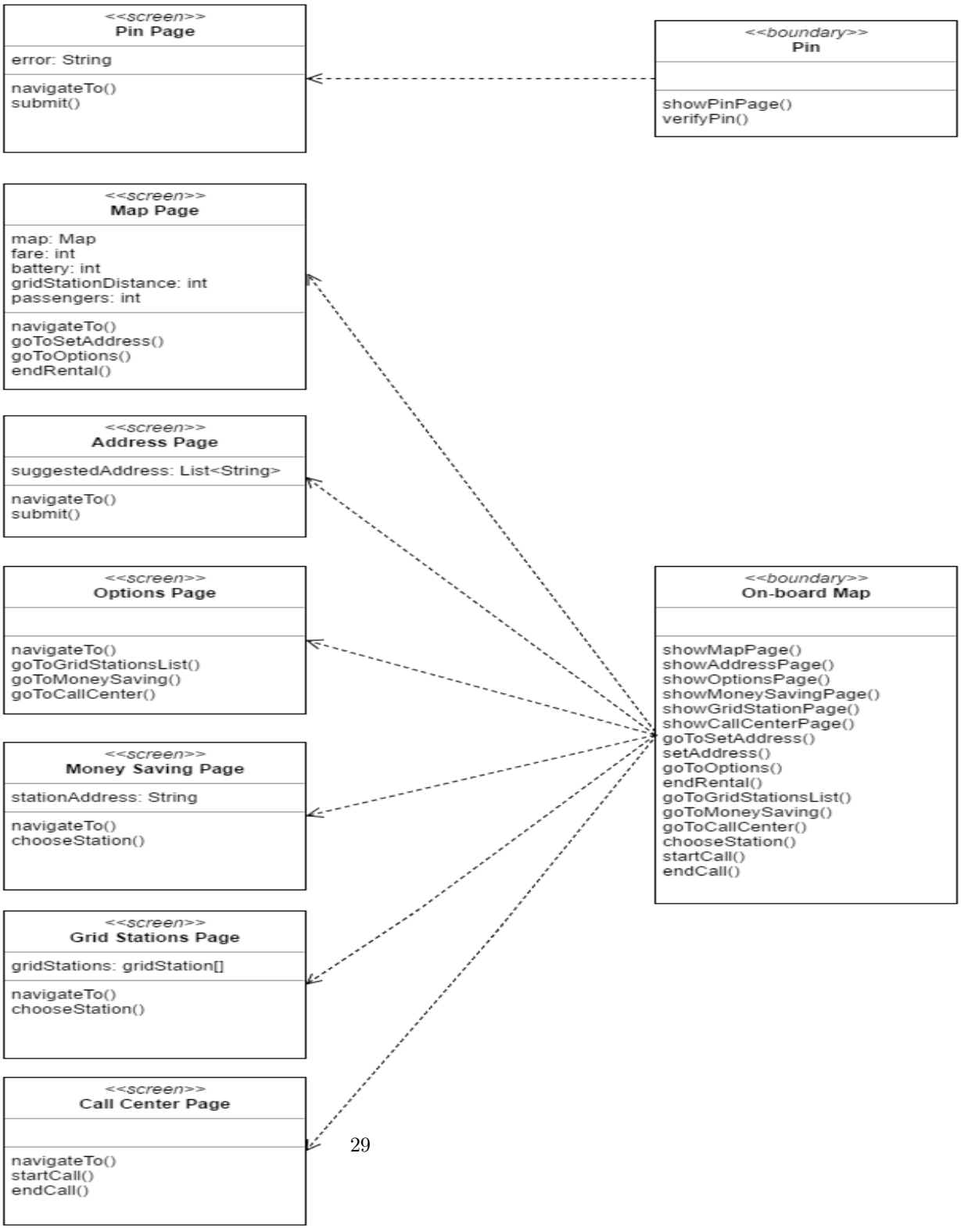


Figure 14: SB operator

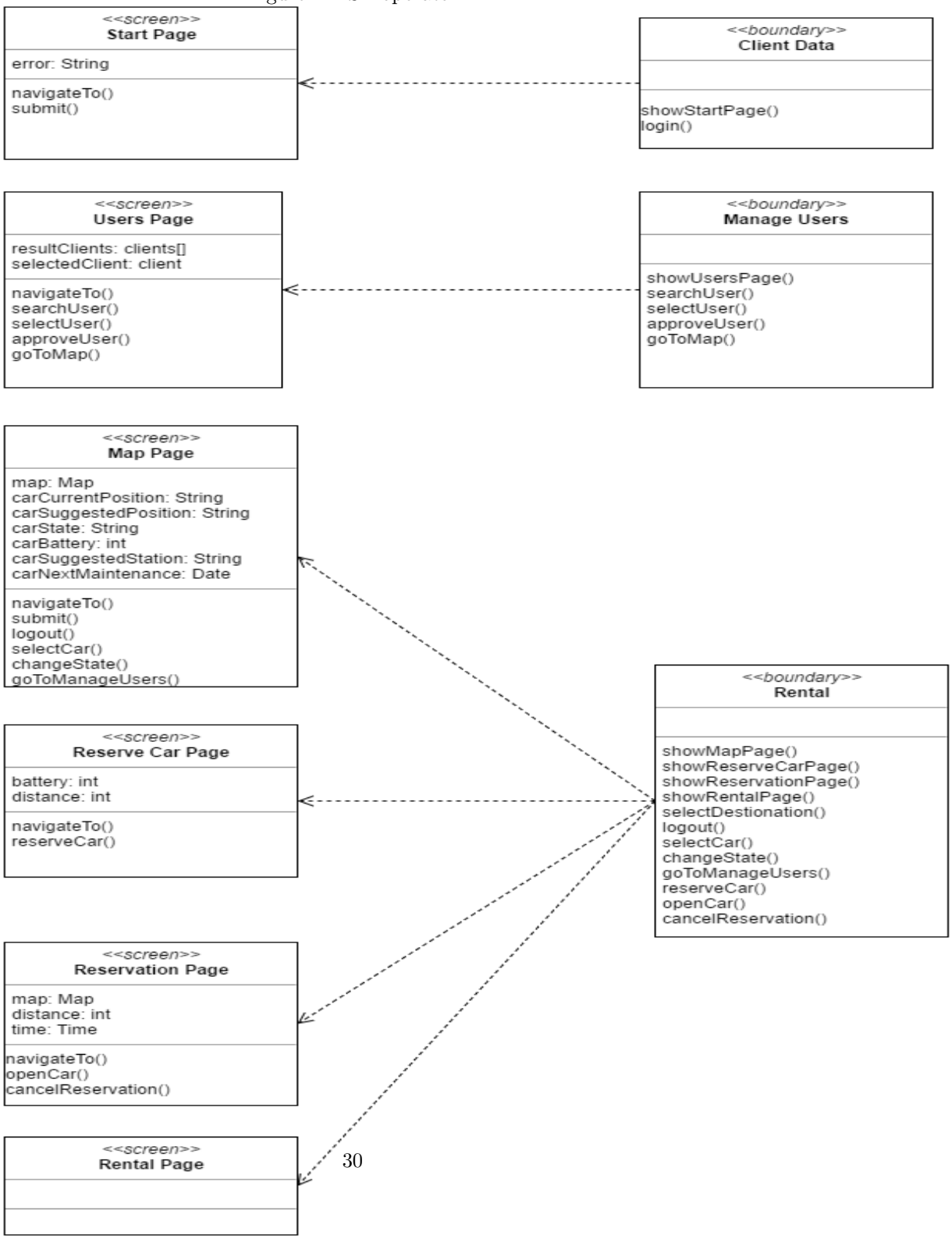
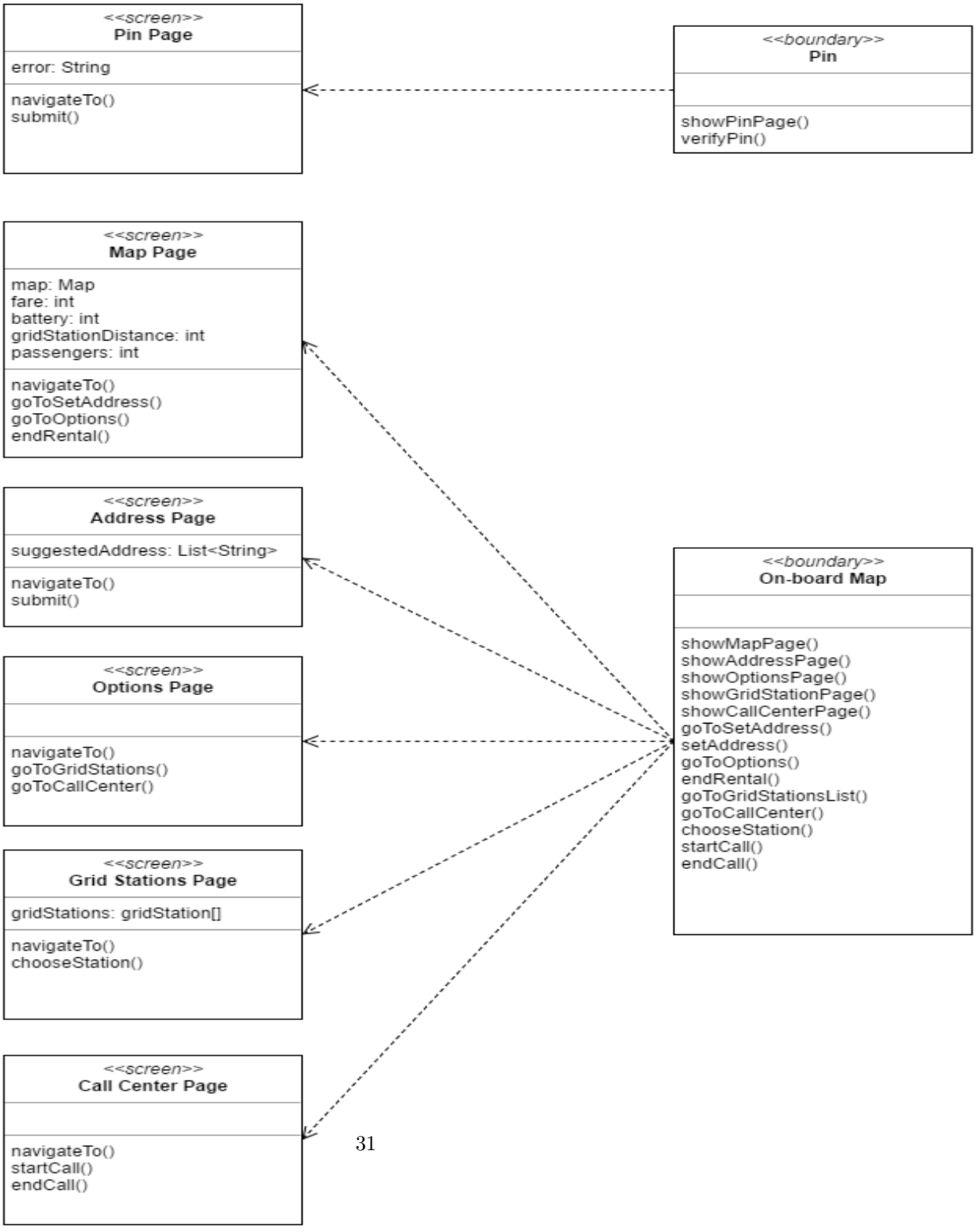
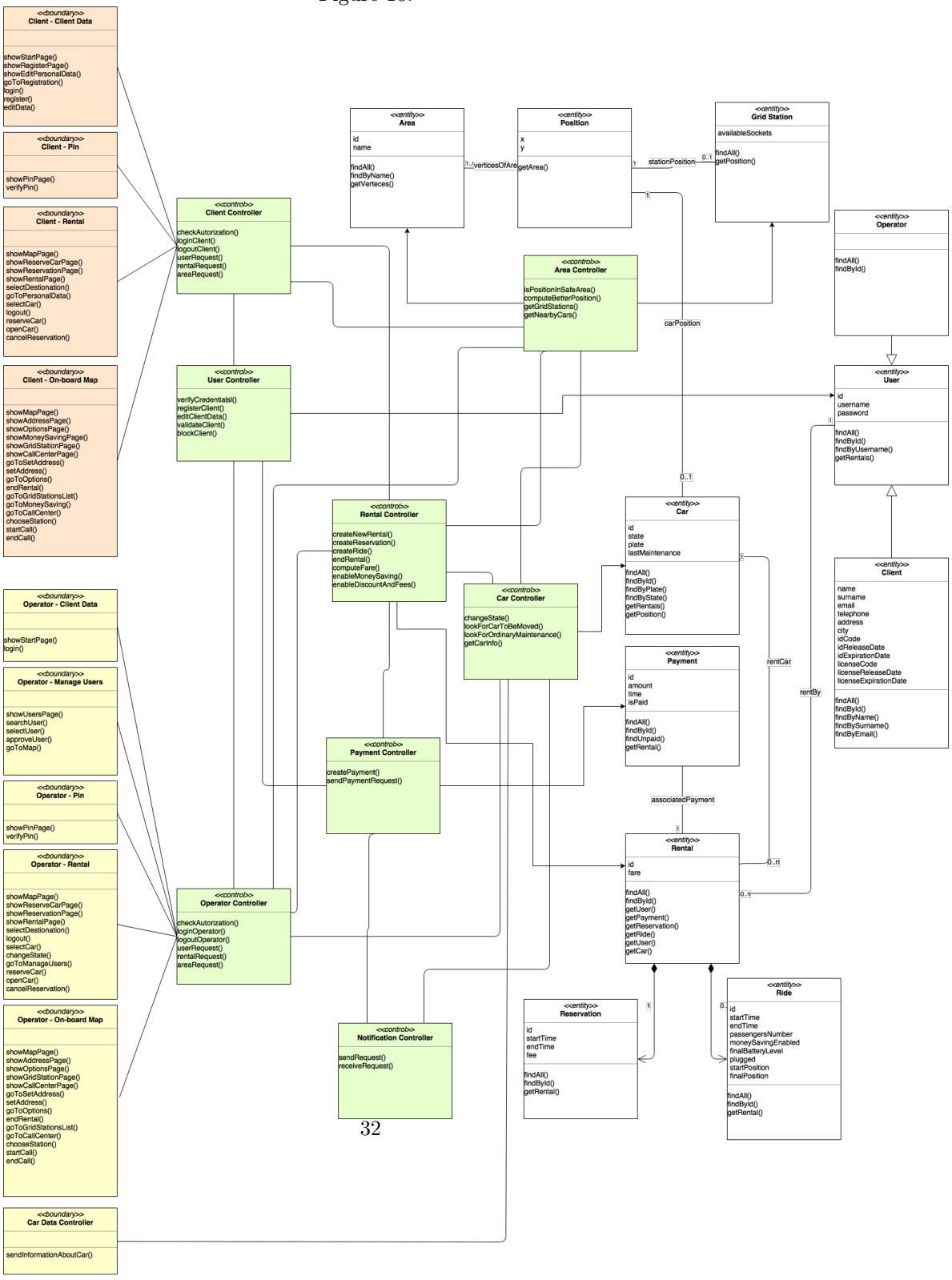


Figure 15: part 2



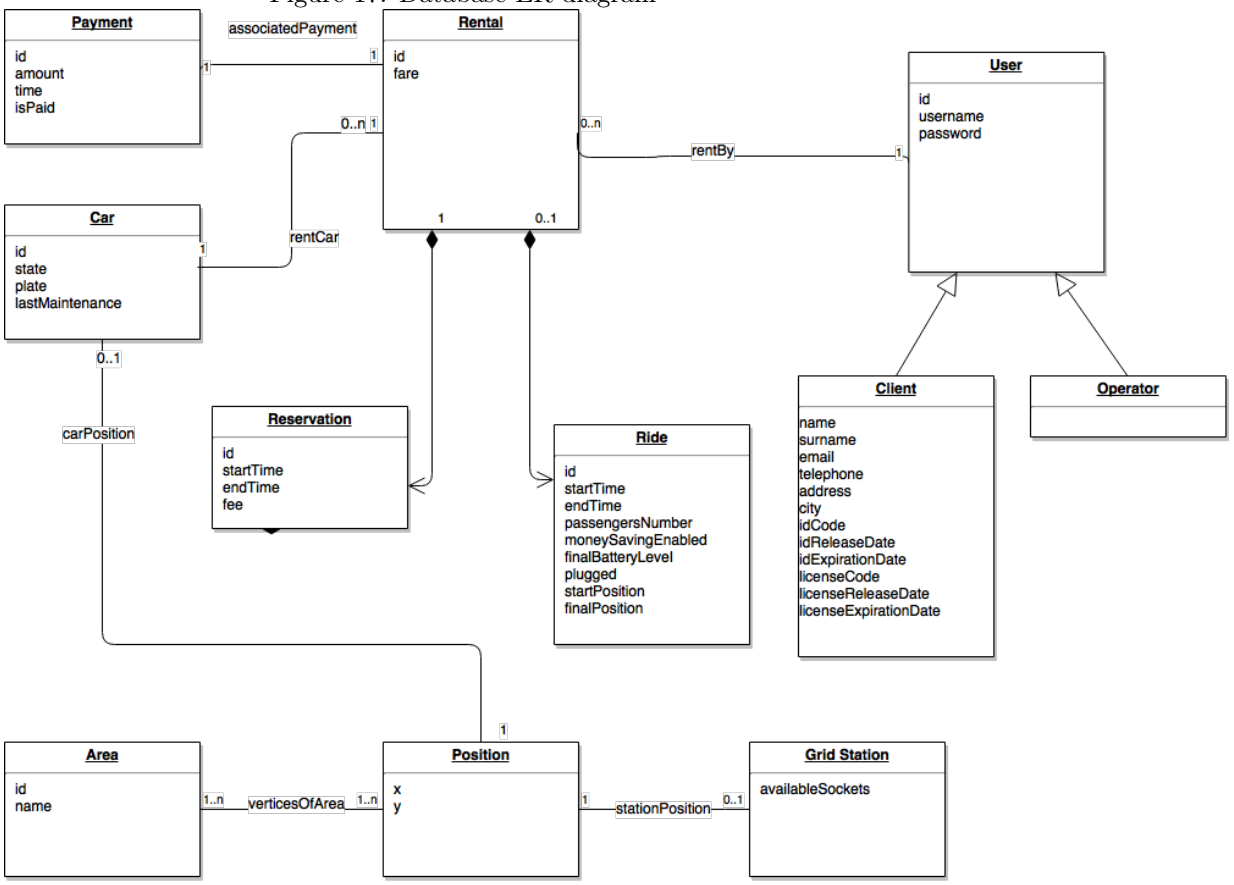
4.3 BCE diagrams

Figure 16:



5 Entity Relation Diagram

Figure 17: Database ER diagram



5.1 Other decisions

Note that not every information is stored in the database: in order to improve the performance of our database system, photos of clients and of their documents will be stored in encrypted folders. The locations of documents related to a particular user is univocally determined by the user's id.

6 Requirements traceability

- [G1] Allows clients to find the location of the cars and the distance from their current position or from a specified address
  - Client controller
  - Area Controller
  - Gateway Google Maps API
- [G2] Allows clients to reserve an available car
  - Client controller
  - Rental controller
  - Car controller
- [G3] Allows clients to access the vehicle when a reservation is made and they are nearby
  - Client controller
  - Rental controller
  - Car controller
- [G4] Allows users to use the GPS service of the on-board computer in order to get traffic indications on both the destination and the power grid stations
  - Client controller
  - Operator controller
  - Area Controller
  - Gateway Google Maps A
- [G5] Encourages clients to have an eco-friendly behaviour applying discounts or fees associated to specific actions
  - Clients controller
  - Rental controller
  - Payment controller
- [G6] Allows clients to visualize the current fare during the drive
  - Clients controller
  - Rental controller
- [G7] Allows the client to enable the "money saving" option in order to guarantee an uniform distribution of vehicles in return for discount
  - Clients controller

  - Rental controller

- [G8] Allows operators to localize cars with empty battery in order to put them in charge
  - Operator controller
  - Car controller
  - Area controller
- [G9] Suggests the operators where to move cars in order to have a uniform distribuition of vehicles
  - Operator controller
  - Car controller
  - Area controller
- [G10] Allows operators to get access of all cars
  - Operator controller
  - Rental controller
- [G11] Allows operators to change the state of a car
  - Operator controller
  - Car controller
- [G12] Allows operators to approve the request of registration of a client
  - Operator controller

7 References

7.1 Used tools

- Umlet - sequence diagram
- Notepad++ - code in algorithm design
- draw.io - diagrams

8 Hours of work

Giovanni Agugini

- Lunedì 28/11 : 2h aggiornamento e general overview
- Giovedì 1/12 : 3h lettura esempio DD e inizio stesura testo
- Giovedì 1/12 : 2h stesura testo e formulate domande su Beep
- Venerdì 2/12: 1h abbozzo schema architetturale e visione del doc
- Sabato 3/12: 3h JEE Overview e schema per car system
- Domenica 4/12: 2h 30 visione esempio DD
- Giovedì 08/12: 3h sequence diagram su reservation
- Sabato 10/12: 4h sequence diagram su end rental e inscription request
- Domenica 11/12: 2h algorithm

Matteo Foglio

- 24/11: 2h
- 25/11: 2h
- 26/11: 3h
- 28/11: 2h
- 29/11: 3h
- 30/11: 6h
- 1/12: 5h
- 2/12: 2h
- 3/12: 5h
- 4/12: 4h
- 5/12: 4h
- 6/12: 2h

Tommaso Massari

- 24/11: 2h
- 25/11: 2h
- 26/11: 2h
- 27/11: 3h
- 29/11: 2h
- 30/11: 2h
- 1/12: 4h
- 2/12: 2h
- 3/12: 3h
- 4/12: 4h
- 5/12: 5h

9 Revision History

Table 1:

Version	Date	Author(s)	Summary
1.0	5/12/2016	all group members	initial release
1.1	13/01/2017	all group members	fixex in the BCE document