POLITECNICO
MILANO 1863

Politecnico di Milano
2016-2017 Software Engineering 2 Project:
PowerEnJoy
Design Document

Version 1.1

Alessandro Polenghi 879111

Alessandro Terragni 879112

Contents

1.Introduction

1.1. Purpose

The purpose of this document is to give more technical details about PowerEnjoy system in order to provide a single vision of all parts of the software and clearly defining how they work.
This document is addressed to developers and aims to identify:

- The high level architecture
- The design patterns
- The main components and their interfaces provided one for another
- The Runtime behavior

The document will explain the architectural decisions and tradeoffs chosen in the design process and its justifications.
We will use the UML standards and other useful diagrams to show the structure of the system.

1.2 Scope

The system aims to support a car-sharing service that exclusively provide electrical cars.
PowerEnjoy is a service based on mobile application and has a unique target: the clients.
The system allows clients to reserve an electric car via the mobile app, using his GPS position or inserting his position manually to find an electrical car in the same zone.
The clients must be registered to the system to use it.
The registration process is very simple, it requires the credentials, the driving license and the payment data; the system is able to check if this data are valid and it allows the registration only for new users.

1.3. Definitions, acronyms, abbreviations

- **DBMS** = Database Management System
- **JVM** = Java Virtual Machine
- **RDMS** = Relational DataBase Management System
- **Android auto** = Is a smartphone projection standard, developed by Google, to allow mobile devices running the Android operating system
- **JDBC** = Java API which defines how a client may access a database

- **GlassFish** = an open-source application server project for the Java EE platform

For the missing definitions, acronyms and abbreviations please refer to paragraph 1.4 of the RASD document

1.4. Reference documents

- Requirements and Analysis Specification Document (RASD).
- Integration Test Plan Document (ITPD)
- Project Plan (PP)

1.5. Document structure

• **Architecture Design**: this section is divided into different parts:
  - High level components and their interaction
  - Component view
  - Deploying view
  - Runtime view
  - Component interfaces
  - Selected architectural styles and patterns

• **Algorithms Design**

• **User Interface Design**: this section presents mockups and UX diagrams.

• **Requirements Traceability**: this section aims to explain how the decisions taken in the RASD are linked to design elements.

2.Architectural design

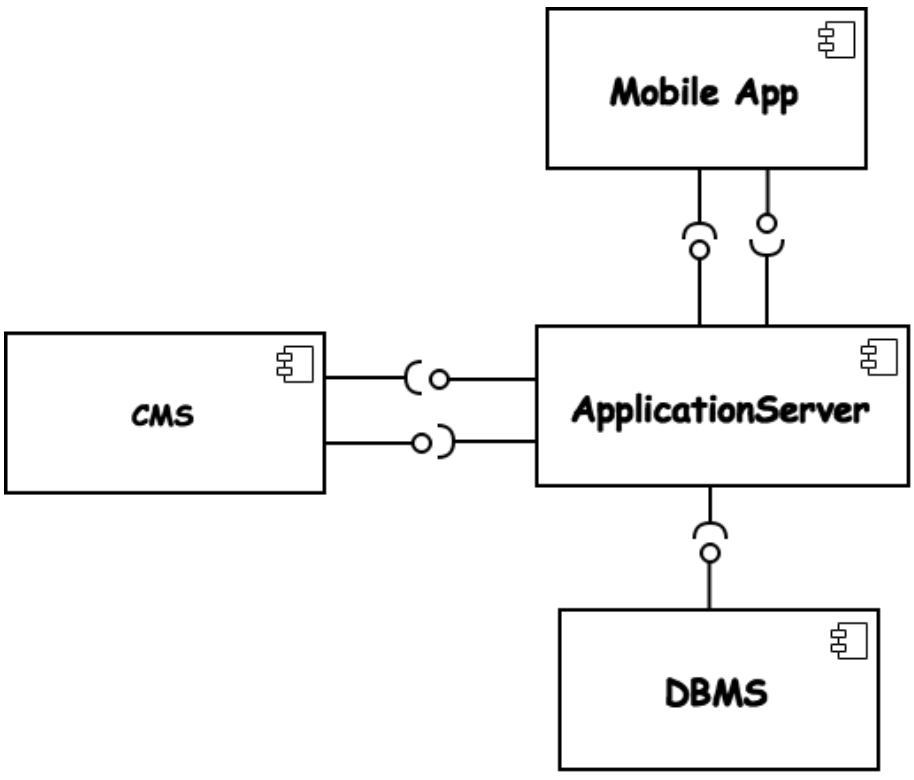
2.1. High Level components

The PowerEnjoy has a three tier architecture that will be described using a top-down approach.
The main high level components of the system are:

1. Mobile App
2. Application Server
3. DBMS

Moreover, there is also an existing system that must interface with the Application Server: the CMS.
We decided to implement a three tier architecture to follow a model view controller pattern, in fact, our application doesn't need a web server because we don't have a web site application.

The main element is the Application Server; it receives requests for reservations from the clients through the Mobile App.
The server communicates also with the DBMS, which store the data about the clients and the reservations.
Furthermore, the Application Server communicates also with an existing system called CMS: it provides the cars data to the Server and the notifies it about some changes of the cars.



2.2. Component view

The Application Server has some specific components that come from a continuous refining of the sequence diagrams. We started from the main functionalities and we imagine how they can be executed, adding new components every time we discovered that a new one was needed. These are the result of our analysis:

- **SearchManager**
  - It manages the search of cars available
- **ReservationManager**
  - It manages all the reservations
- **RegistrationManager**
  - It acquires data for the user registration and create new accounts
- **LoginManager**
  - It manages login of all clients
- **PaymentsManager**
  - It manages the payment process, executes the bank transaction and send the notification to the client
- **RideManager**
  - It calculates the ride bill, including fine and discounts
- **CarManager**
  - It handles the car data
- **ClientManager**
  - It manages the client accounts and checks also the correctness of the data provided
- **EmergencyManager**
  - It manages the emergency procedure

All these components are strictly connected to each other, as it is showed on the components diagram in the next paragraph.

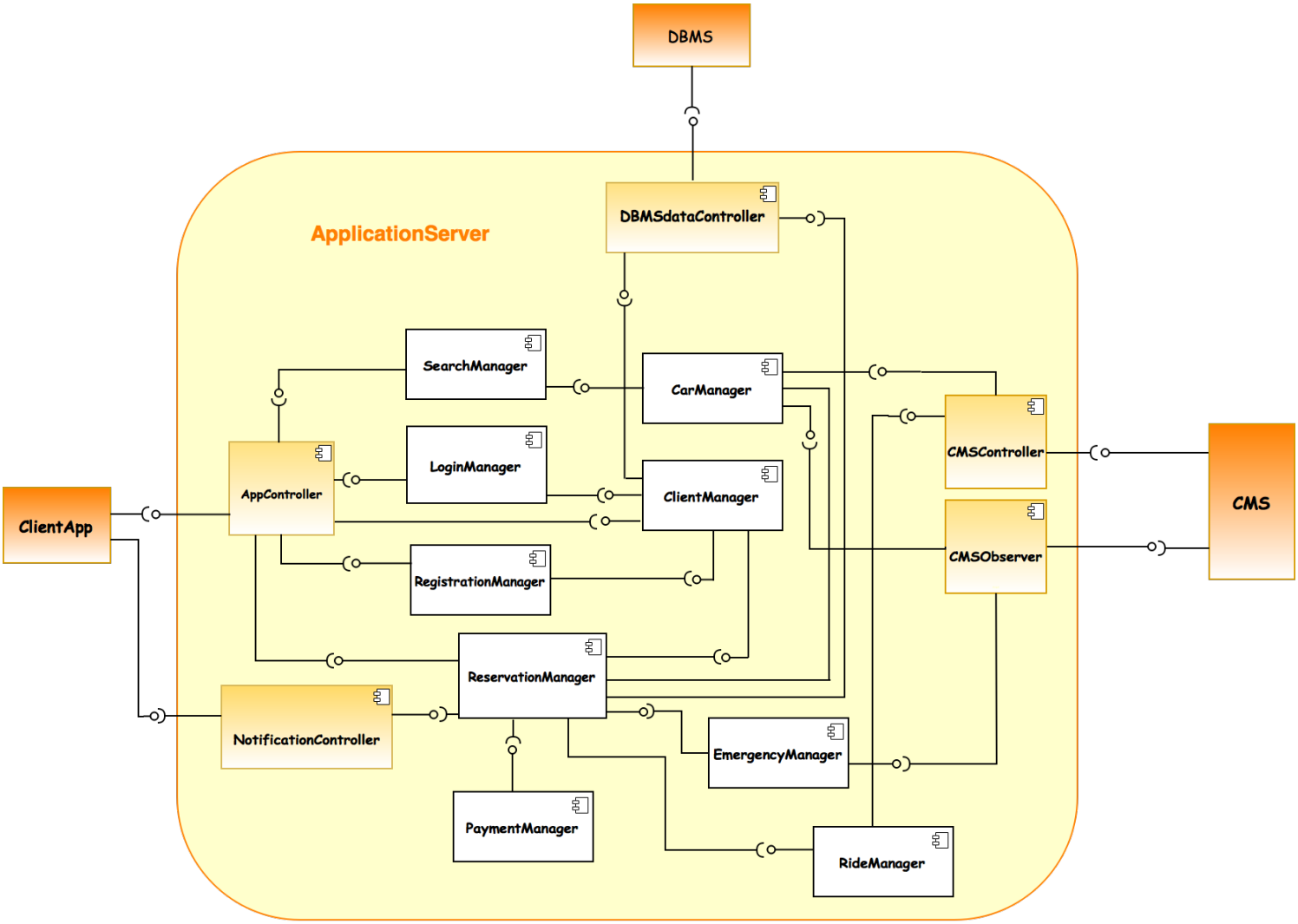
Moreover, there are some components that act as interfaces with the high level components:

- **DBMSDataController**

- Interacts with the DBMS making the queries on the database

- **CMSController**
  - Interacts with the CMS making the queries on the CMS Data Base

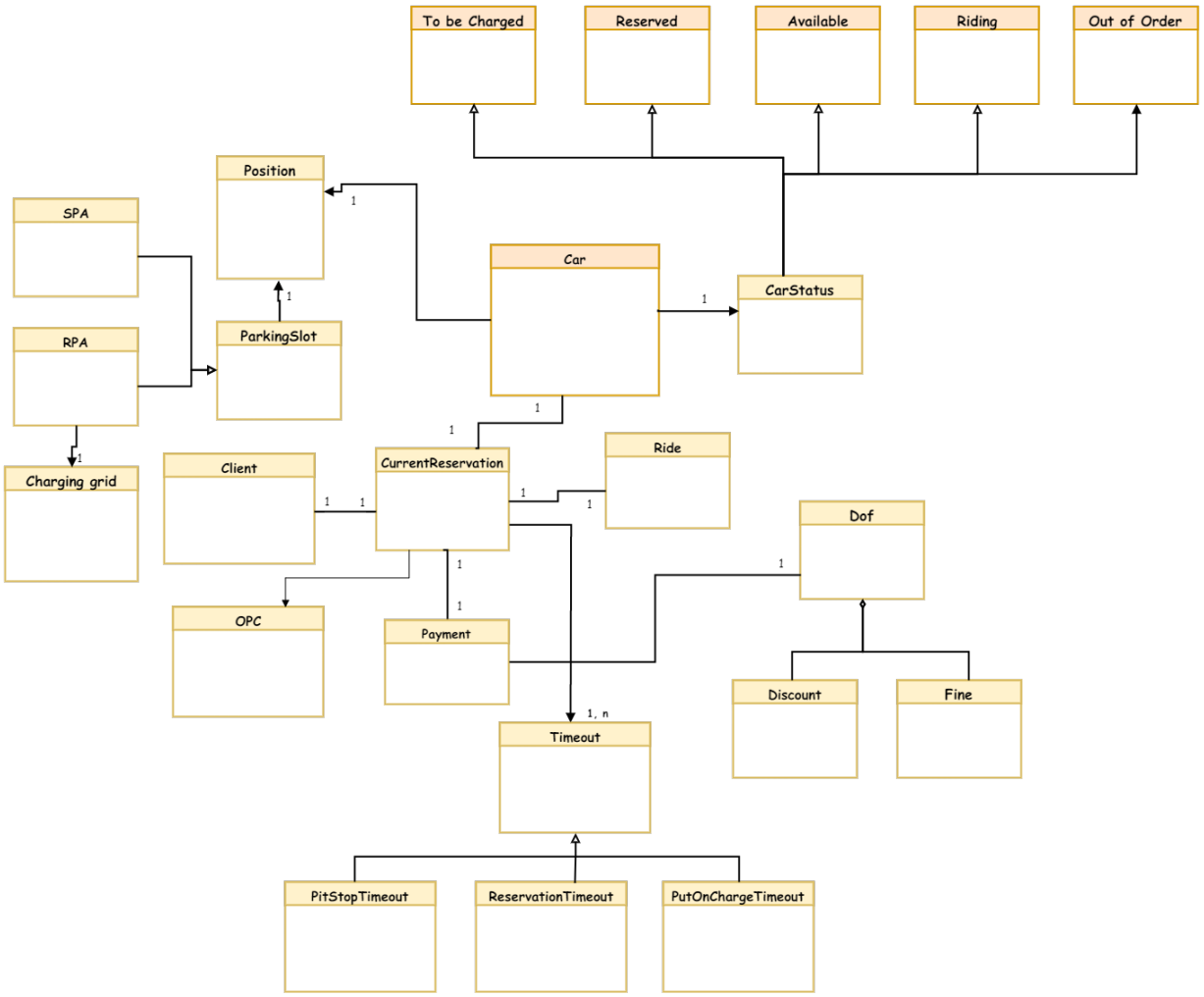
- **CMSObserver**
  - Interacts with the CMS to respond to CMS request
- **AppController**
  - Interacts with the Mobile App
- **NotificationController**
  - Sends notification to the Mobile App



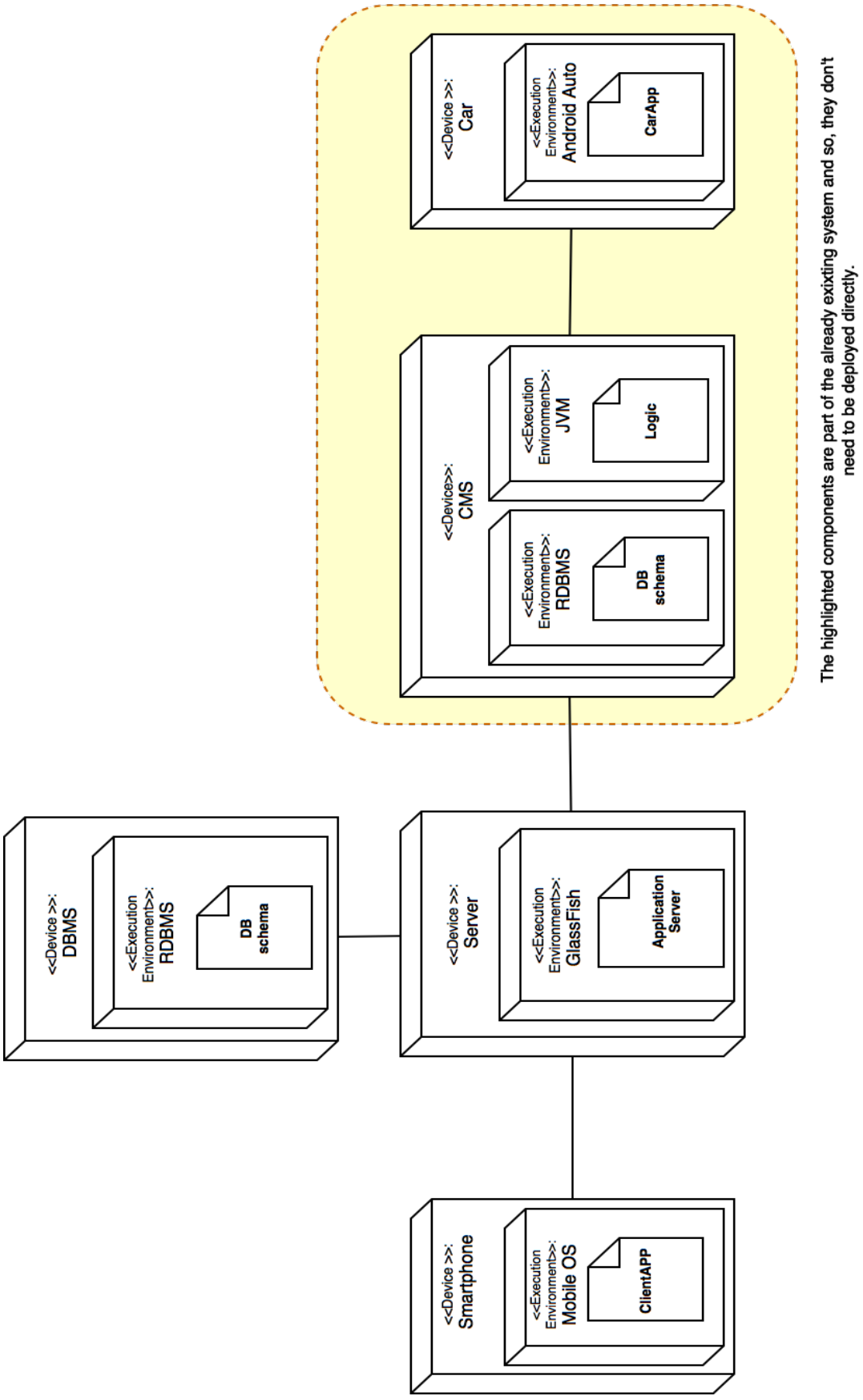
These components will be described better in the component interfaces paragraph.

2.2.1 Server Data Structures

Internally, the Server maintains a data structure that connects together the data contained in the DBMS and the ones contained in the CMS.
We didn't add details about the attributes because at this stage of the Design we prefer to give an overall view of the data structure and leave the implementation choices to the programmer.



2.3. Deployment view

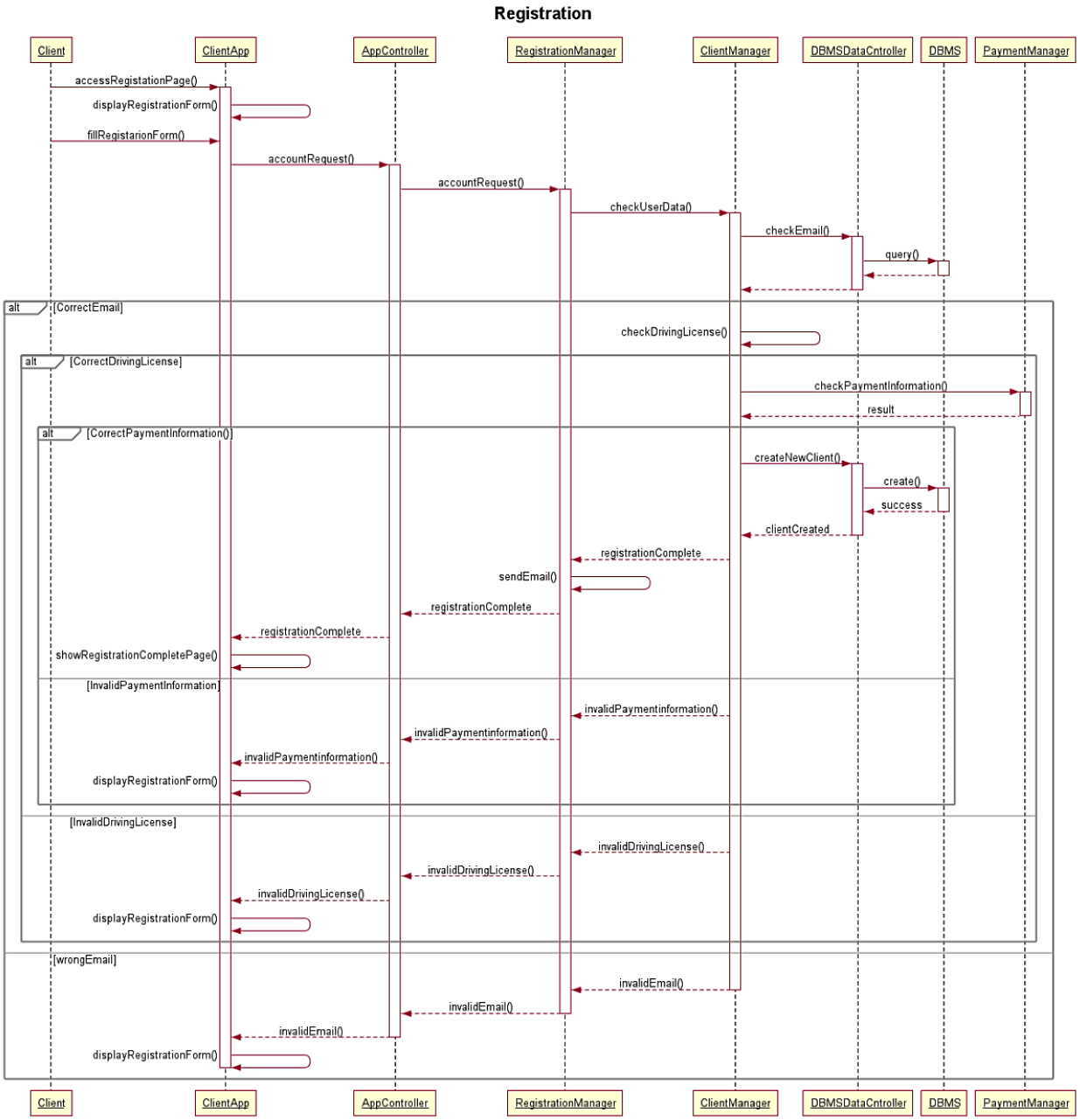


The technological implementation choices of the execution environment are indicative and will be discussed more precisely during the implementation phase.

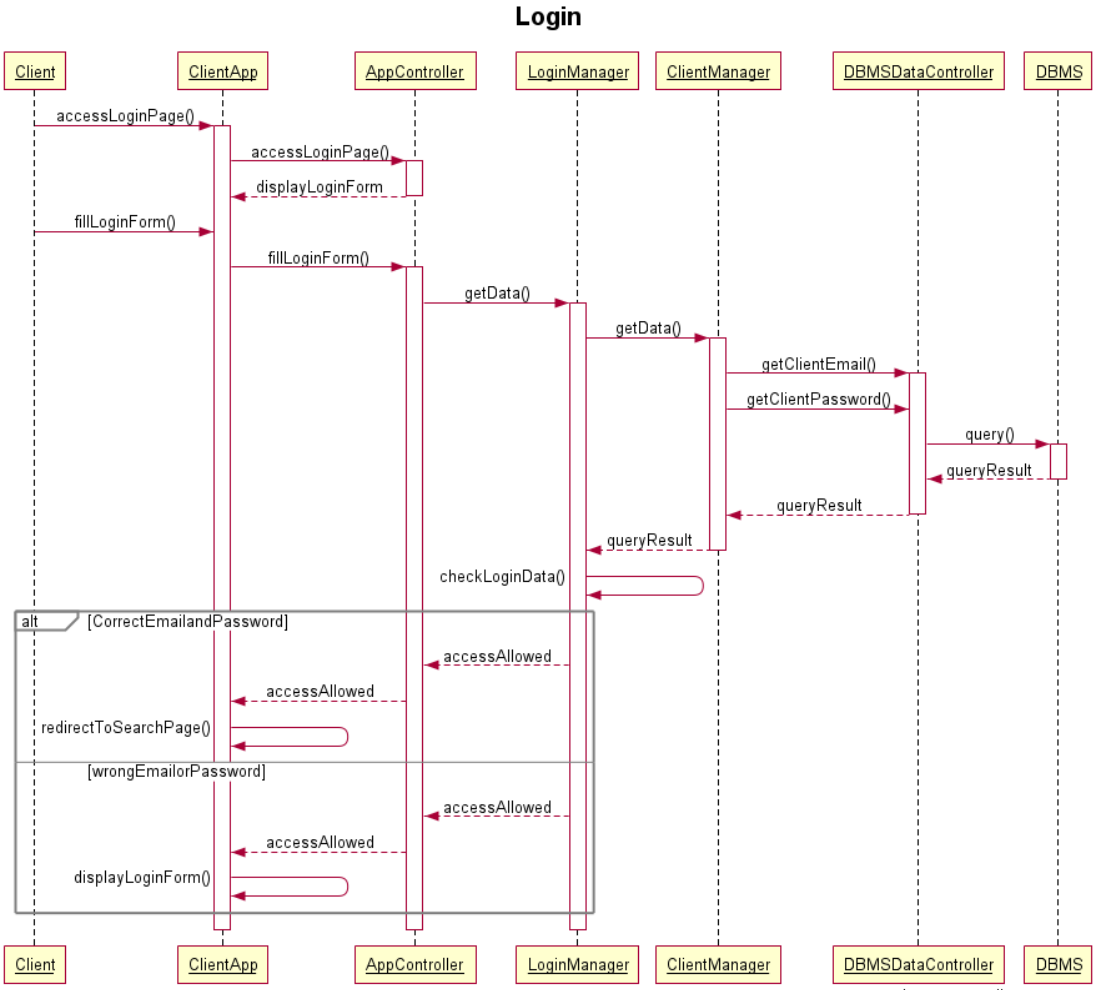
2.4. Runtime view

This section describes the dynamic interaction of the system in the most relevant cases.

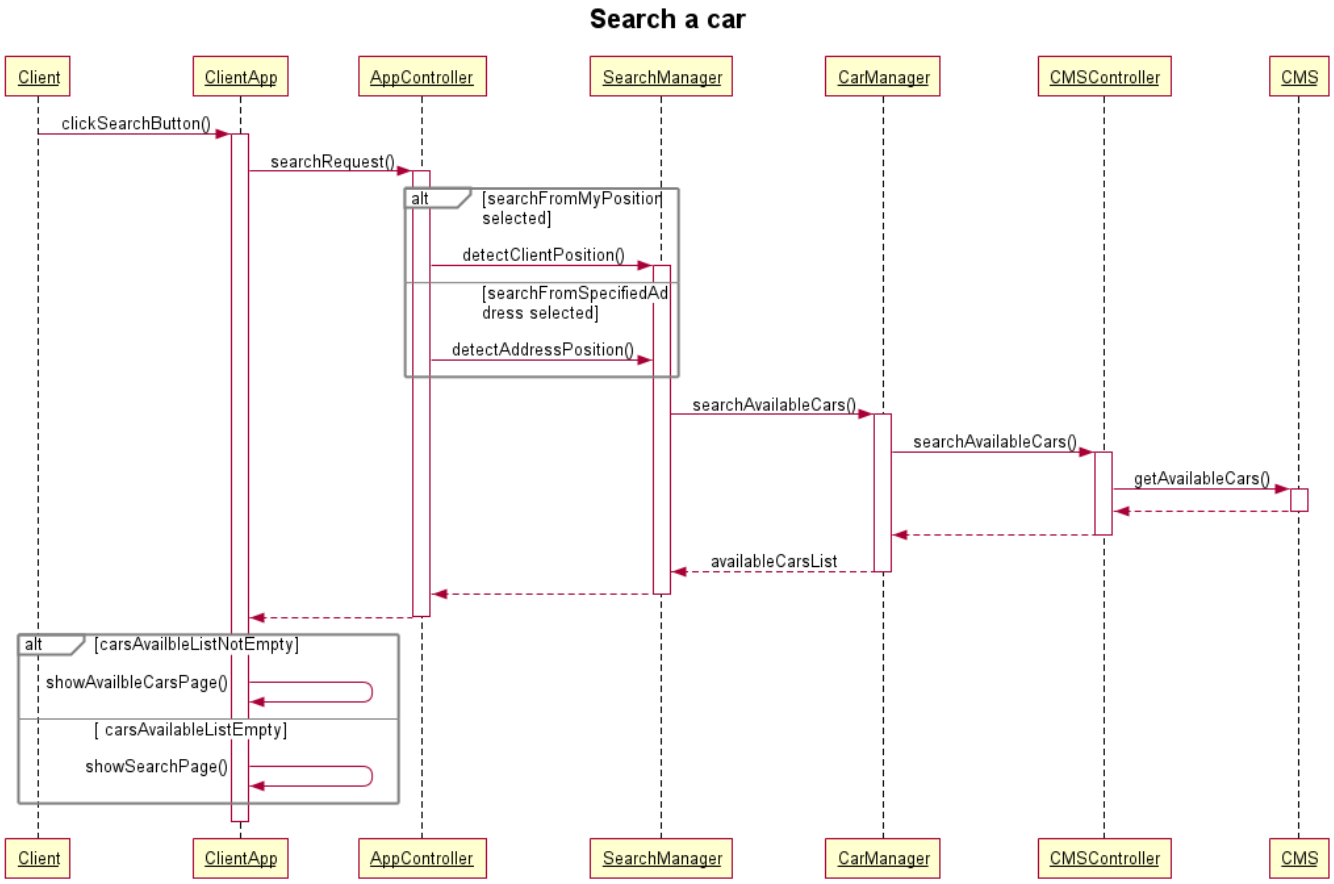
2.4.1 Registration



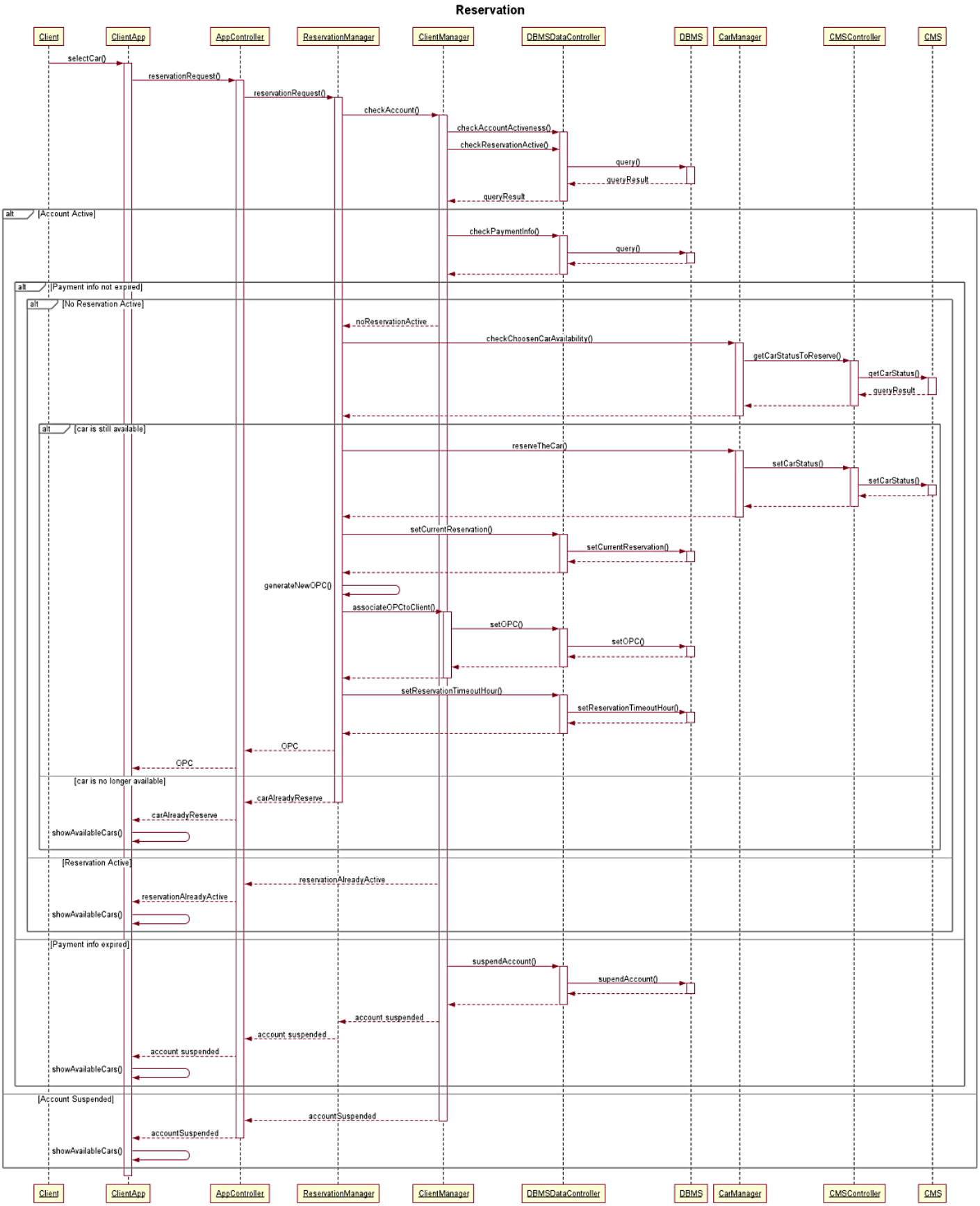
2.4.2 Login


Login
Client
ClientApp
AppController
LoginManager
ClientManager
DBMSDataController
DBMS
accessLoginPage()
accessLoginPage()
displayLoginForm
fillLoginForm()
fillLoginForm()
getData()
getData()
getClientEmail()
getClientPassword()
query()
queryResult
queryResult
queryResult
checkLoginData()
alt
[CorrectEmailandPassword]
accessAllowed
accessAllowed
redirectToSearchPage()
[wrongEmailorPassword]
accessAllowed
accessAllowed
displayLoginForm()
Client
ClientApp
AppController
LoginManager
ClientManager
DBMSDataController
DBMS

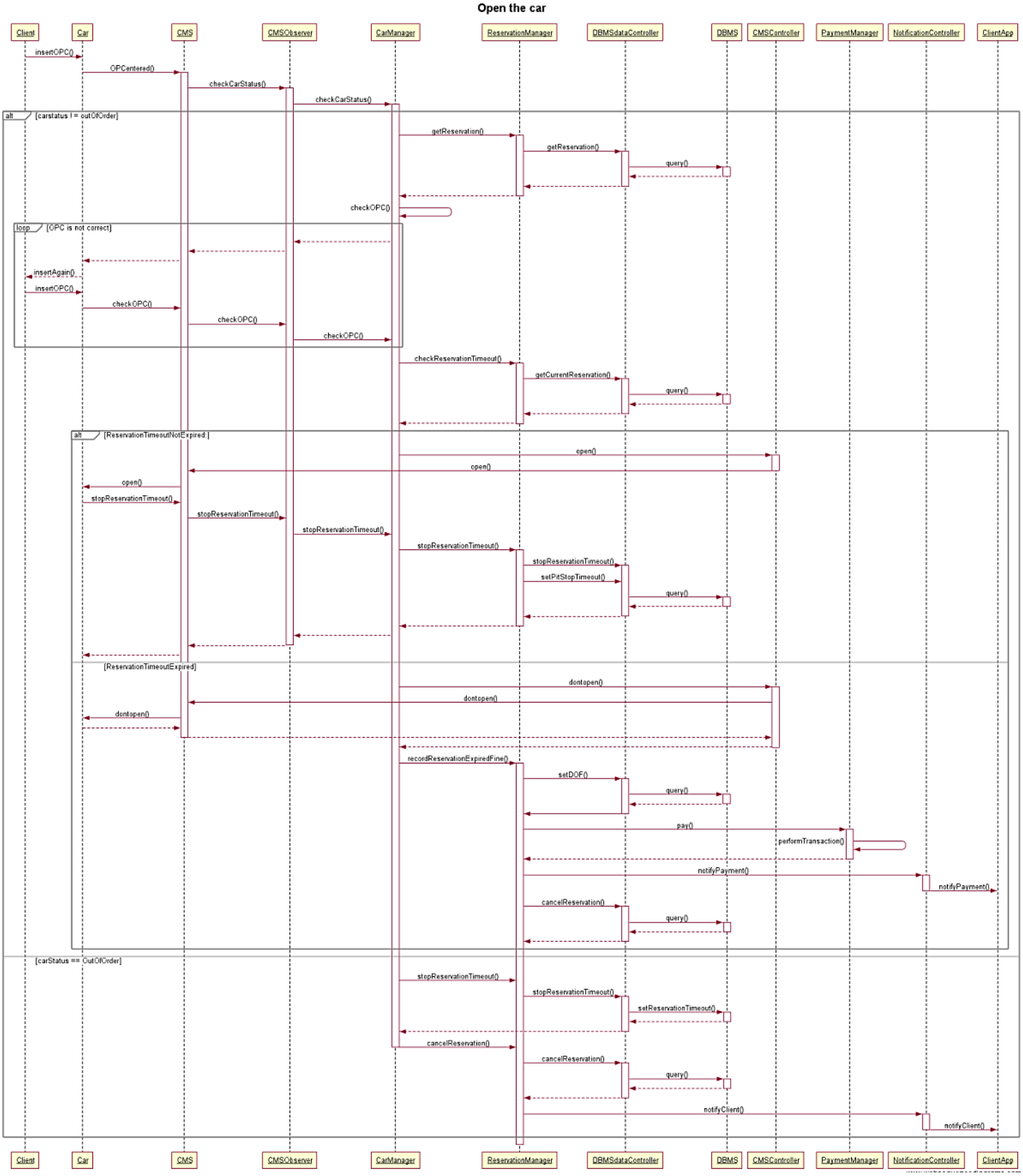
2.4.3 Search


Search a car
Client
ClientApp
AppController
SearchManager
CarManager
CMSController
CMS
clickSearchButton()
searchRequest()
alt
[searchFromMyPosition selected]
detectClientPosition()
[searchFromSpecifiedAddress selected]
detectAddressPosition()
searchAvailableCars()
searchAvailableCars()
getAvailableCars()
availableCarsList
alt
[carsAvailbleListNotEmpty]
showAvailbleCarsPage()
[carsAvailableListEmpty]
showSearchPage()

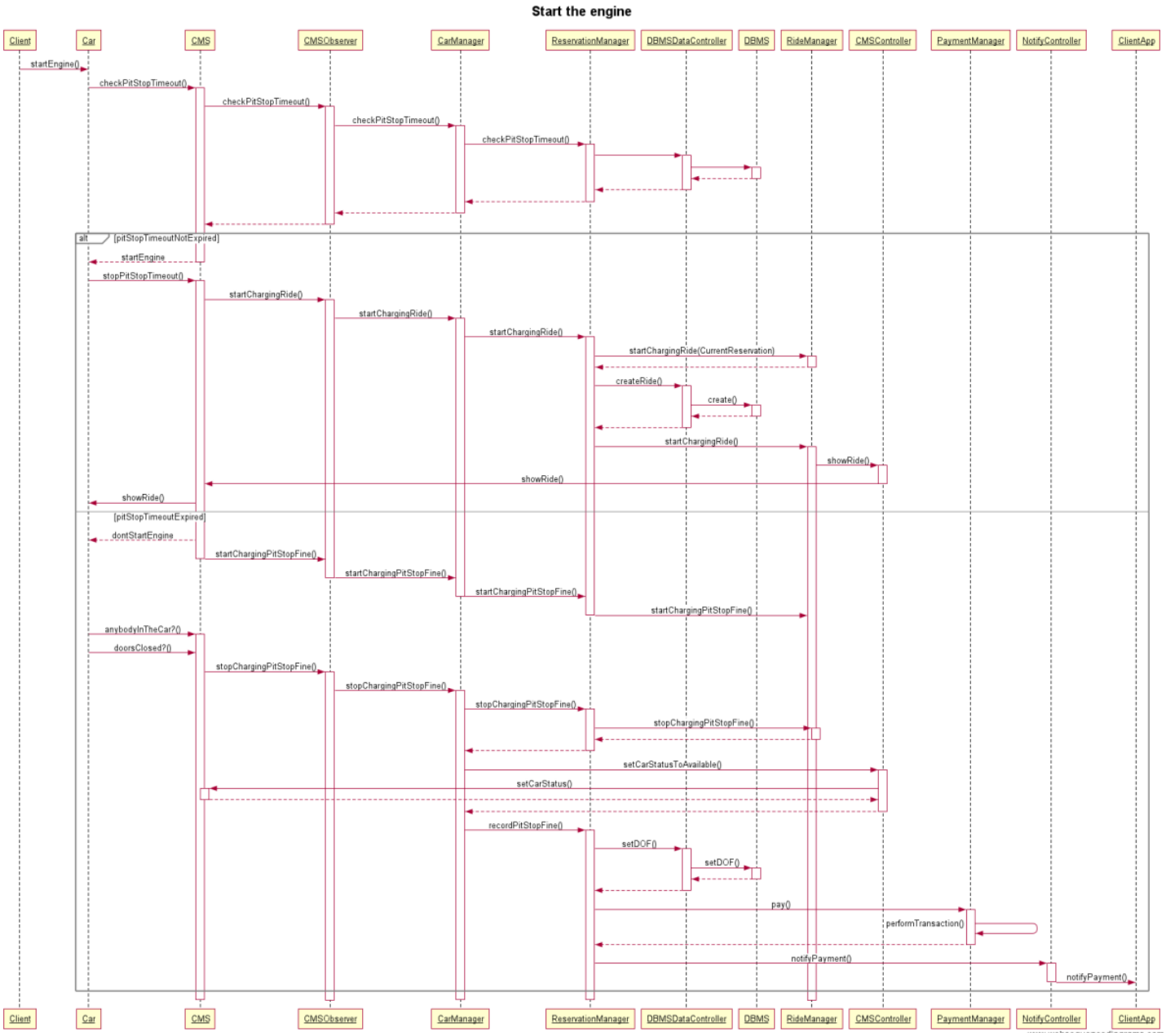
2.4.4 Reservation



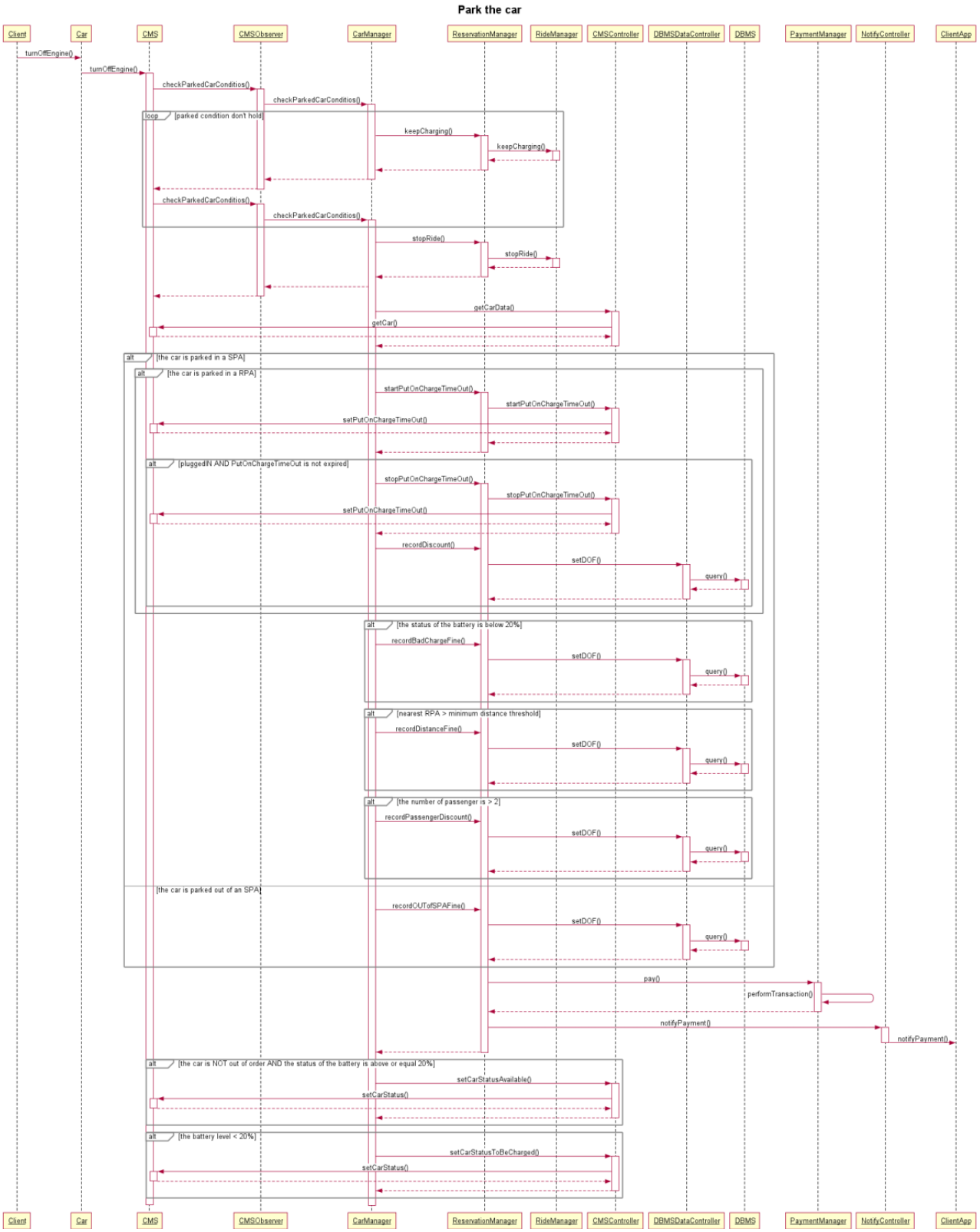
2.4.5 Open the car



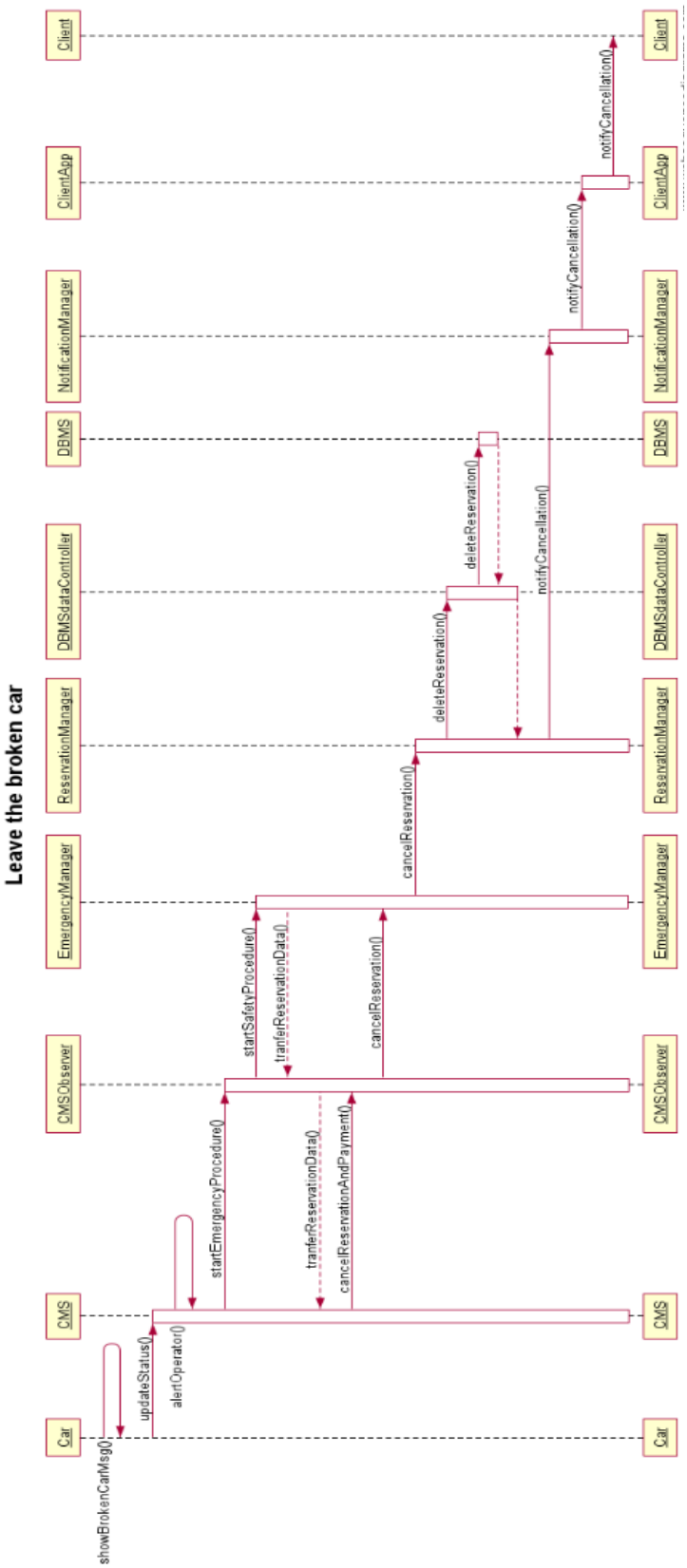
2.4.6 Start the engine



2.4.7 Park the car



2.4.8 Leave broken car



2.5. Component interfaces

The Application Server is connected with the other components through three controllers:

- **DBMSDataController**: connects the Application Server with the DBMS through a JDBC interface.
  It is used to make queries on the DBMS: for example, it retrieves data of the clients, adds new one and creates reservations.

- **CMSController**: connects the Application Server with the CMS through a JDBC interface.
  It is used to make queries on the DBMS of the CMS to retrieve data about the cars.

- **CMSObserver**: connects the CMS with the Application Server.
  It is used to send request to the server when some changes occur on the car data: these requests are driven by triggers on the DBMS of the CMS.

- **AppController:** connects the ClientApp with the Application Server.
  The ClientApp sends requests to this interface, that dispatches them to the right component of the Application Server.

- **NotificationController:**
  It is used by the Application Server to send asynchronous notifications to the client App.

2.6. Selected architectural styles and patterns

2.7. Design patterns

- **MVC:** Model-View-Controller pattern has been widely used in our application.
  In a broader view, the mobile App represents the View, the Server represents the Controller, and the DBMS and the CMS Database represent the Model
- **Adapter:** Adapters are used in our system to adapt the MobileApp, DBMS and CMS to the server application through the AppController, DBMSDataController and CMSController
- **Client-Server**: The application is strongly based on a Client-Server communication model.
  In particular, this model is used:
  - Between the MobileApp and the Server
  - Between the CMS and the Server
  - Between the DBMS and the Server

2.8 Other design decision

To deploy our application, we also need to use some external APIs:

- **Google Maps** for the map services
- **Push service(s)** APIs to send push notifications to clients
- **Email APIs** to send emails to clients.
- **Payment API** to perform bank transactions and check payment data

3.Algorithm design

The only tricky algorithm that the developer will encounter during the implementation of the system will be the calculation of discount and fees on the overall price of the ride.

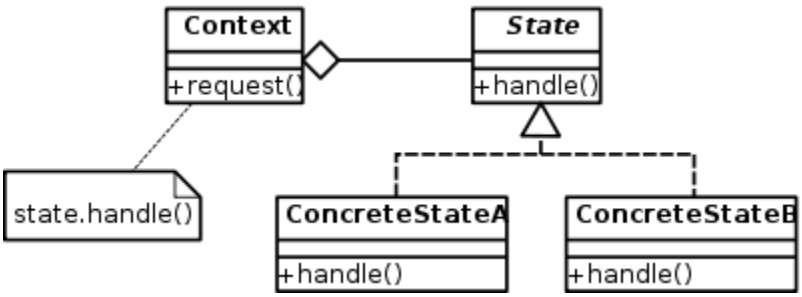
The developer should follow the state chart diagram, present in the section 5.3.2 of the RASD document, to calculate the DOF.

To avoid an endless sequence of nested if condition, a state pattern can be used.

In this case, the state pattern is used to encapsulate varying behaviours for the same object DOF based on its internal state.

This can be a cleaner way for an object to change its behaviour at runtime without resorting to large monolithic conditional statements and thus improve maintainability.

UML Class Diagram of the State Pattern



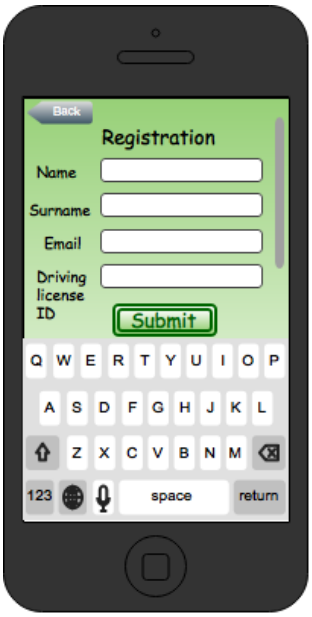
4.User interface design

4.1. Mockups

We have drawn some simple mockups of the ClientApp component to show how it should appear.

This mockup is represented by the Home screen in the UX Diagram.

Picture 1 - Home Page



This picture shows the Registration Page, it refers to the **sequence diagram 2.4.1.**

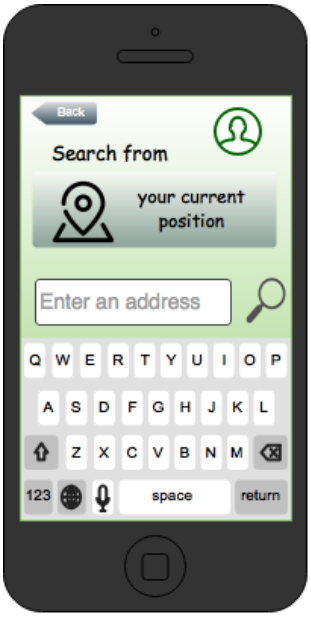
This mockup is represented by the **Registration screen** and the **RegistrationForm** input form in the UX Diagram.

Picture 2 – Registration Page

Picture 3 - Login Page

This picture shows the Login, it refers to the **sequence diagram 2.4.2.**

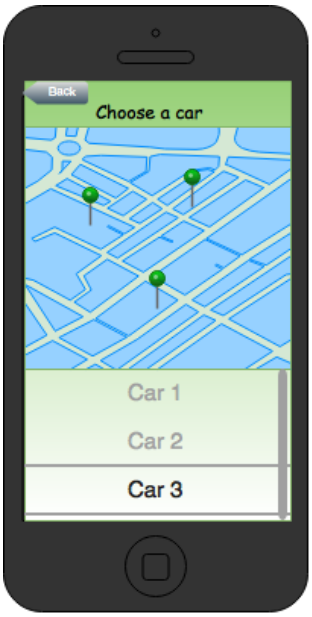
This mockup is represented by the **Login screen** and the **LoginForm** input form in the UX Diagram.



Picture 4 - Search Page

This picture shows the Search, it refers to the **sequence diagram 2.4.3.**

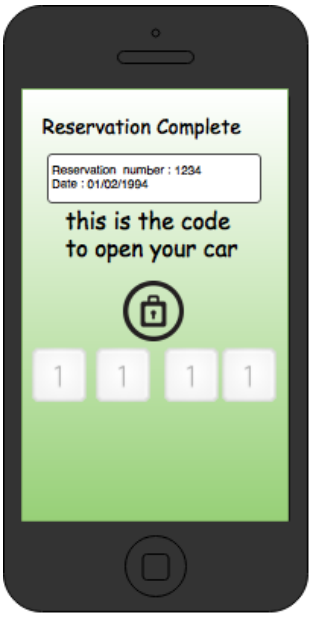
This mockup is represented by the Search Page screen and the **SearchForm** input form in the UX Diagram.



Picture 5 Choose a car page

This picture shows the Choose a car, it refers to the **sequence diagram 2.4.4.**

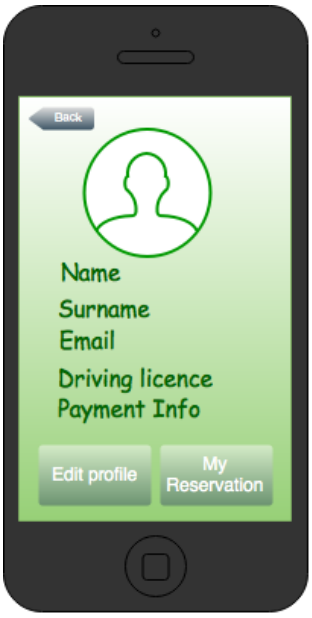
This mockup is represented by the Car Page screen in the UX Diagram.



Picture 6 – Reservation complete Page

This picture shows the Reservation complete, it refers to the **sequence diagram 2.4.4.**

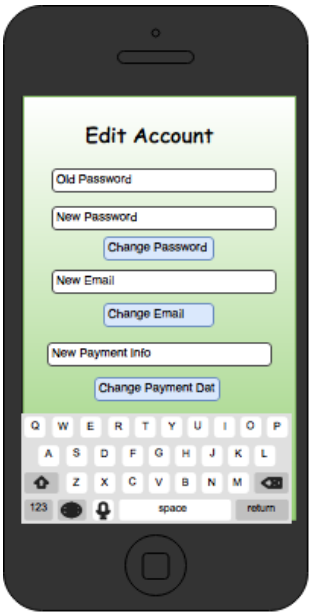
This mockup is represented by the Reservation screen in the UX Diagram.



Picture 7: Account Page

This picture shows the Account Page**.**

This mockup is represented by the Account Page screen in the UX Diagram.

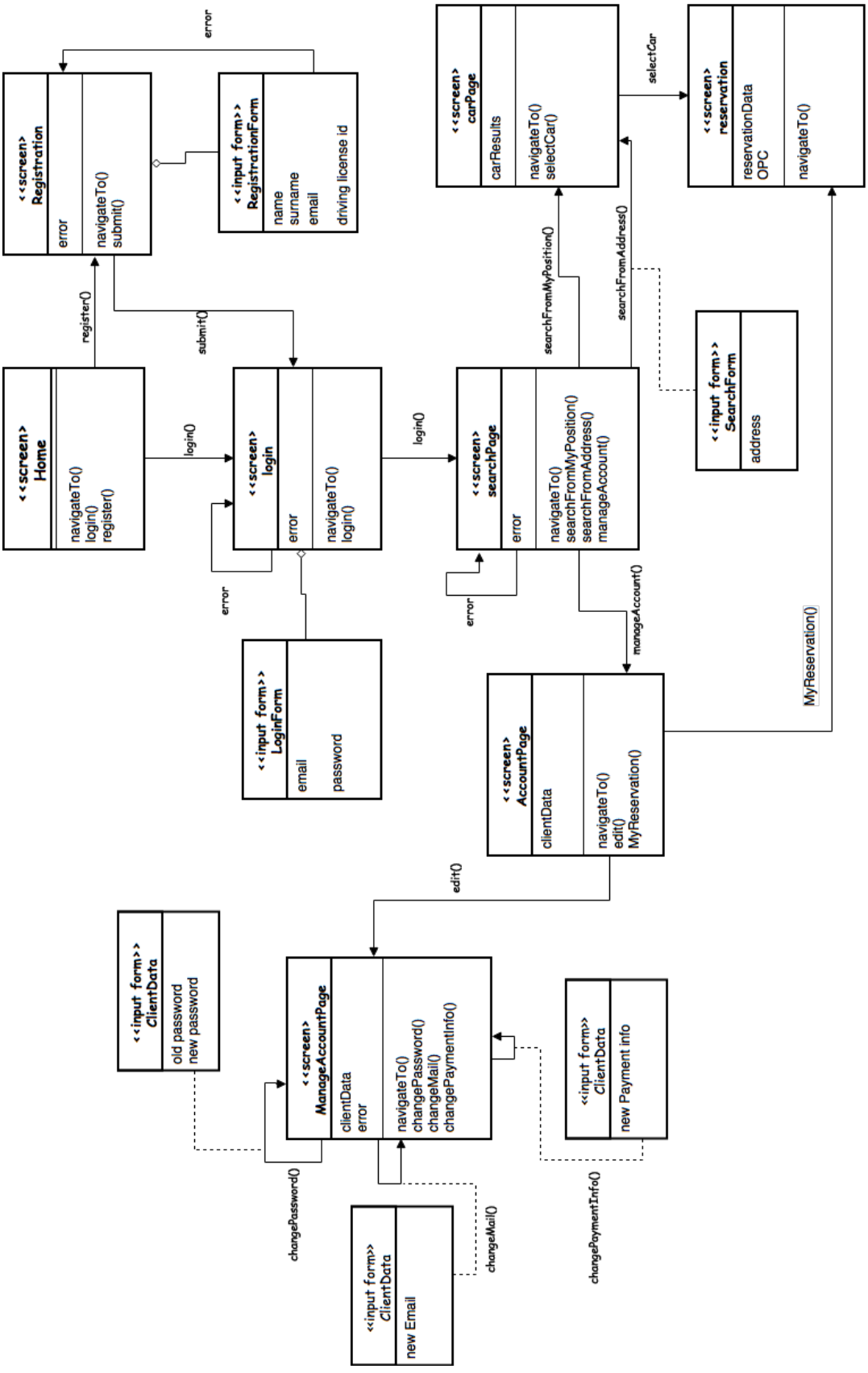


Picture 8 Manage Account Page

This picture shows the Manage Account Page**.**

This mockup is represented by the Manage Account Page screen and the **ClientDataForm** input form in the UX Diagram.

4.2 UX Diagrams



5.Requirement traceability

The design of this project was made aiming to fulfill optimally the requirements and goals specified in the RASD.
The reader can find below the list of these requirements and goals and the designed component of the application which will assure its fulfillment.

- [G1] Users must be able to register to the system by providing their credentials and payment information
    - ClientApp
    - AppController
    - RegistrationManager
    - ClientManager
    - DBMSController
    - DBMS

- [G2] Clients must be able to log-in in the service
    - ClientApp
    - AppContoller
    - LoginManager
    - DBMSdataController
    - DBMS

- [G3] Clients must be able to find the locations of available cars in a prefixed range from their current location or from a specified address.
    - ClientApp
    - AppController
    - SearchManager
    - CarManager
    - CMSController
    - CMS

- [G4] Among the available cars in the range mentioned in the G3, user must be able to reserve a car
    - ClientApp

- AppController
- ReservationManager
- ClientManager
- DBMScontroller
- DBMS
- CarManager
- CMSController
- CMS

- [G5] The client must be able to pick up the car he has reserved
  - CMS
  - CMSobserver
  - CarManager
  - ReservationManager
  - DBMSdataController
  - DBMS
  - CMScontroller

- [G6] The system must be able to charge the user for using the car
  - CMS
  - CMSobserver
  - CarManager
  - ReservationManager
  - DBMSdataController
  - DBMS
  - RideManager

- [G7] The system should be able to fine the client
  - CMS
  - CMScontroller
  - CarManager
  - ReservationManager
  - DBMSdataController
  - DBMS

- [G8] The system should be able to apply discounts to the clients
  - CMS
  - CMScontroller
  - CarManager
  - ReservationManager

    - DBMSdataController
    - DBMS
- [G9] The system should be able to make the client pay
    - ReservationManager
    - RideManager
    - PaymentManager
    - NotificationManager
    - ClientApp

- [G10] The system must be able to detect when an incident happened and when damages or failure occurs
    - CMS
    - CMSobserver
    - EmergencyManager
    - ReservationManager
    - DBMSdataController
    - DBMS
    - NotificationManager
    - ClientApp

- [G11] The system must be able to allow the client to change its profile information
    - ClientApp
    - AppController
    - ClientManager
    - DBMSdataController
    - DBMS
    - PaymentManager

6. Appendix

6.1 Tools used

- Microsoft World
- Dropbox
- Draw.io
- websequencediagrams.com

6.2 Hours of work

Alessandro Polenghi

- 30/11/16 1.30 h
- 2/12 6.30 h
- 3/12 6.30 h
- 4/12 3h
- 5/12 1.30 h
- 6/12 2.30 h
- 7/12 6.30 h
- 9/12 1h
- 10/12 1.30h
- 13/1 1h

Alessandro Terragni

- 30/11/16 1.30 h
- 2/12 6.30 h
- 3/12 6.30 h
- 4/12 3h
- 5/12 1.30 h
- 6/12 2.30 h
- 7/12 6.30 h
- 9/12 1.30h
- 10/12 1h
- 13/1 1h

6.3 Change Log

- v1.1
  - fix High Level Component diagram
  - fix Component View diagram
  - add some explanation about the design process reasoning
  - fix a name of input form in the UX Diagram