POLITECNICO
MILANO 1863

Politecnico di Milano
A.A. 2016 – 2017
Software Engineering 2: "PowerEnJoy"
Design Document

Marco Festa
December 11, 2016

Contents

1 Introduction

1.1 Purpose

In this document each aspect discussed in the RASD is analyzed in detail and further described. The main project architecture and design pattern is proposed and different runtime behaviors are explained using detailed diagrams.

1.2 Scope

PowerEnJoy needs a digital management system to support his car-sharing service. The system is divided in two logical subsystems: one interacting with the company's customer and the other interacting with the numerous electrical PowerEnJoy cars. All registered user must provide valid credentials and driving license in order to get access to the system. Once they get approved they may start searching for nearby cars and reserve them for a ride. GPS is crucial to provide all important features such as car localization and car opening. The car must be able to identify if certain eco-friendly behaviors are respected by the user to apply a significant discount. The interaction between the user and the system is made possible through 3 different interfaces: the web application, the mobile application (IOS, Android) and the on-board display (a touch-screen monitor located inside the car dashboard). The interaction between the central system and the car is fully automated by the embedded device connected to the on-board screen via the CAN bus and the central system itself through a mobile network internet connection.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- *User (or Registered User):* a person registered to the system. The driving license has been verified and the customer info is correctly added into the database. Registered users are the only entities eligible for car riding.

- *Driver:* the user who physically unlocks an electric car using his own credentials and starts driving it becomes automatically the driver. Registered users are actually the only possible persons to potentially become drivers.
- *Guest:* a person that is not necessarily registered to the system.
- *Safe area:* geographical area where cars are authorized to be parked giving the user the chance to end the ride.
- *Power Grid Station:* inside the safe areas are located several power stations where electric cars can be plugged in and have the central battery recharged.
- *Free Car:* a car which is not being used by any registered user and is not under any pending reservation is considered available or free.
- *Reserved Car:* each registered user has the ability to choose a free car from the smartphone app or web interface and have it reserved for at most one hour. During this phase the car disappears from the list of free cars and can only be opened and unlocked by the user who made the reservation. The reservation state of a car ends either when the car is unlocked by the user who automatically becomes the driver, the one-hour limit is reached, or the state is ended by the user itself.
- *Opening procedure:* a user can open a free car, or one he had reserved, by using the dedicated feature on his smartphone app. An opened and locked car has to be considered a reserved car.
- *Unlocking procedure:* car unlocking is achieved via a PIN code entered through the touch screen display inside the vehicle.
- *PIN code (or PIN):* a 4-digit secret code chosen by the user on his first car ride.
- *In-use Car:* a car that has been unlocked by a registered user and is now able to be turned on by the driver.
- *CAN bus:* the physical network connecting each present electronic device: sensors, micro-controllers, actuators and instruments.

1.3.2 Acronyms

- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document
- **DBMS**: Database Management System
- **DB**: Database
- **PGS**: Power Grid Station
- **CAN bus**: Controller Area Network bus
- **API**: Application Programming Interface: a common way to communicate with other systems.
- **SD**: Sequence Diagram
- **UX**: user experience design
- **BCE**: business controller entity
- **MVC**: model view controller
- **HTML**: Hyper Text Markup Language
- **HTTP**: Hyper Text Transfer Protocol
- **JEE**: Java Enterprise Edition
- **JPA**: Java Persistence API
- **JSP**: Java Server Pages
- **JDBC**: Java Database Connectivity
- **EUCARIS**: European Car and Driving License Information System

1.4 Reference Documents

- PowerEnJoy RASD
- Specification document: "Assignments AA 2016-2017.pdf"
- IEEE Std. 1016-2009, "IEEE Standard for Information Technology – Systems Design – Software Design Descriptions"
- ISO/IEC/IEEE Std. 42010:2011, "Systems and software engineering – Architecture Description"

1.5 Document Overview

Document structure:

- **Section 1 – Introduction**: presentation of the document and product.
- **Section 2 – Architectural Design**
  - **Overview**: high level view of the system's architecture and tiers division.

  - **Component view**: all the logical software components and their interfaces connection are shown in this section.
  - **Deployment view**: the detailed description of how the various logical components will be deployed on different physical machines.
  - **Runtime view**: in this section we use various sequence diagrams to specify core functionalities behaviors and interactions between the different components.
  - **Component Interfaces**: the accurate description of the multitude of interfaces used to interact with our software components.
  - **Selected architectural styles and patterns**: this section contains all of the architectural choices taken during the creation of the application.
- **Section 3 – Algorithms Design**: some of the core and most critical functions of the product are explained in this section with a detailed pseudo-code algorithm description.
- **Section 4 – User Interface Design**: significant mockups along with UX and BCE diagrams are proposed to specify even more the structure and aspect of all the user interfaces.
- **Section 5 – Requirements Traceability**: this section explains the link between the goals identified in the RASD and the solution adopted to achieve them.
- **Section 6 – Appendix**:
  - Tools used.
  - References.
  - Revisions.
  - Working hours division.

2 Architectural design

2.1 Overview

Due to the scalability and expansion needs of PowerEnJoy, we decided to adopt a server-client multi-tiered architecture, with the intent of distributing the various components on different physical machines.

The proposed architecture is divided into 4 core tiers:

1. *Client tier:* the entry point of the user interaction with the central system. The web and mobile applications are considered part of this tier along with the car subsystem. HTTPS protocol is used to communicate with Web Tier. For the web browser application, we use HTML pages and JS scripts. Both the mobile application and car subsystem uses a custom made software application.
2. *Web tier:* all of the client requests are received by this physical tier which basically serves as a mere interface between the client applications and the central system's application. In most components interaction this tier may seem ignored since it doesn't hold any significant logical function. Great effort in hardening the tier's machines is spent in order to increase the overall security. The technology chosen to accomplish all of the tier's task is the Glassfish framework (JSP and Serverlet for the client applications.)
3. *Business tier:* the core tier of the system. It holds the central Application Subsystem responsible for all the sensitive operation and fundamental interactions. The only node directly connected with the DBMS. EJB and JPA modules are deployed to respectively describe the core logic of the system and communicate with the DBMS.
4. *Data tier:* runs the DBMS and holds all of the sensitive application data. MySQL is our default choice for the DBMS implementation. JDBC standard technology is in charge for the interaction with the upper Business tier.

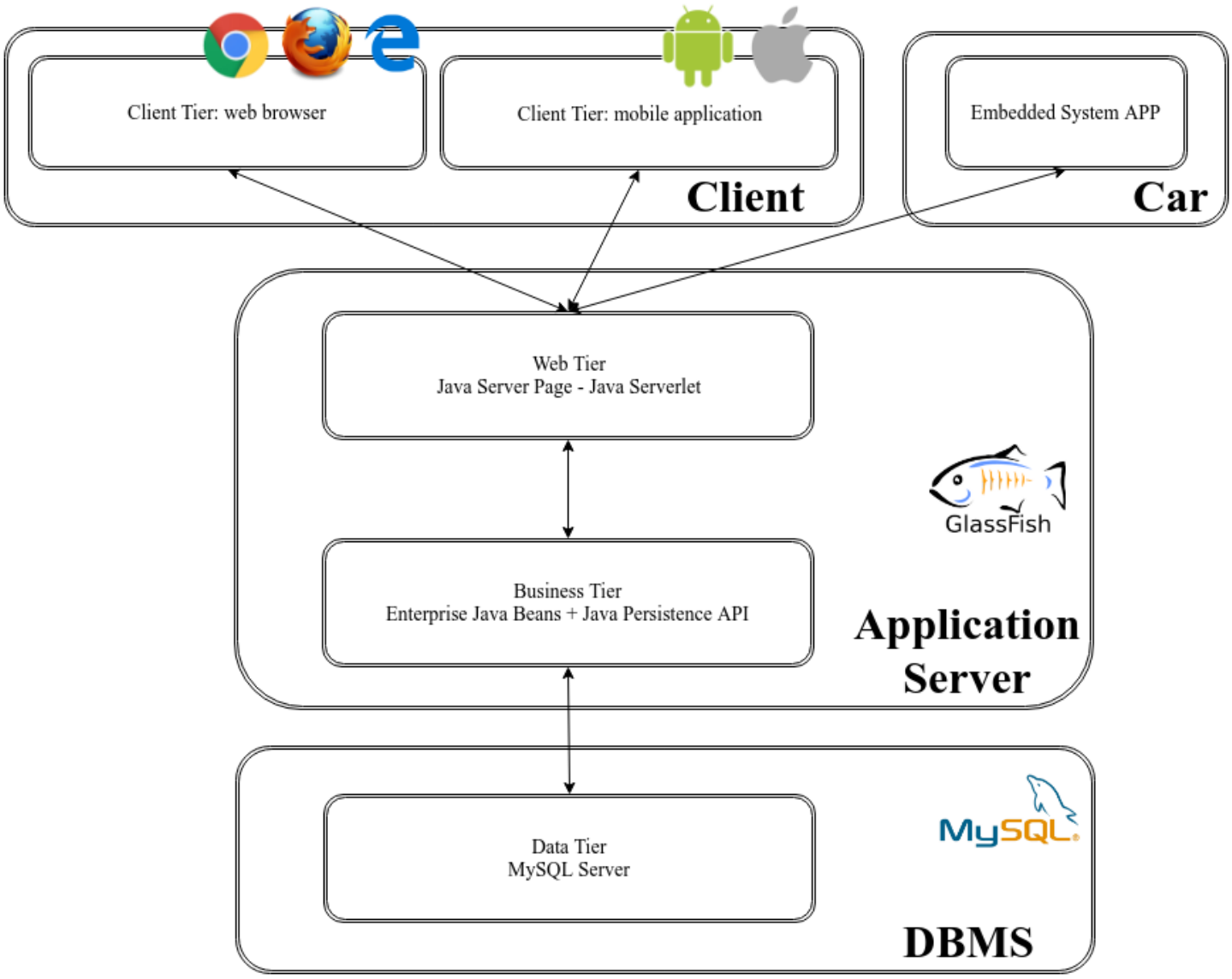

Figure 1: Architecture Overview

2.2 Component View

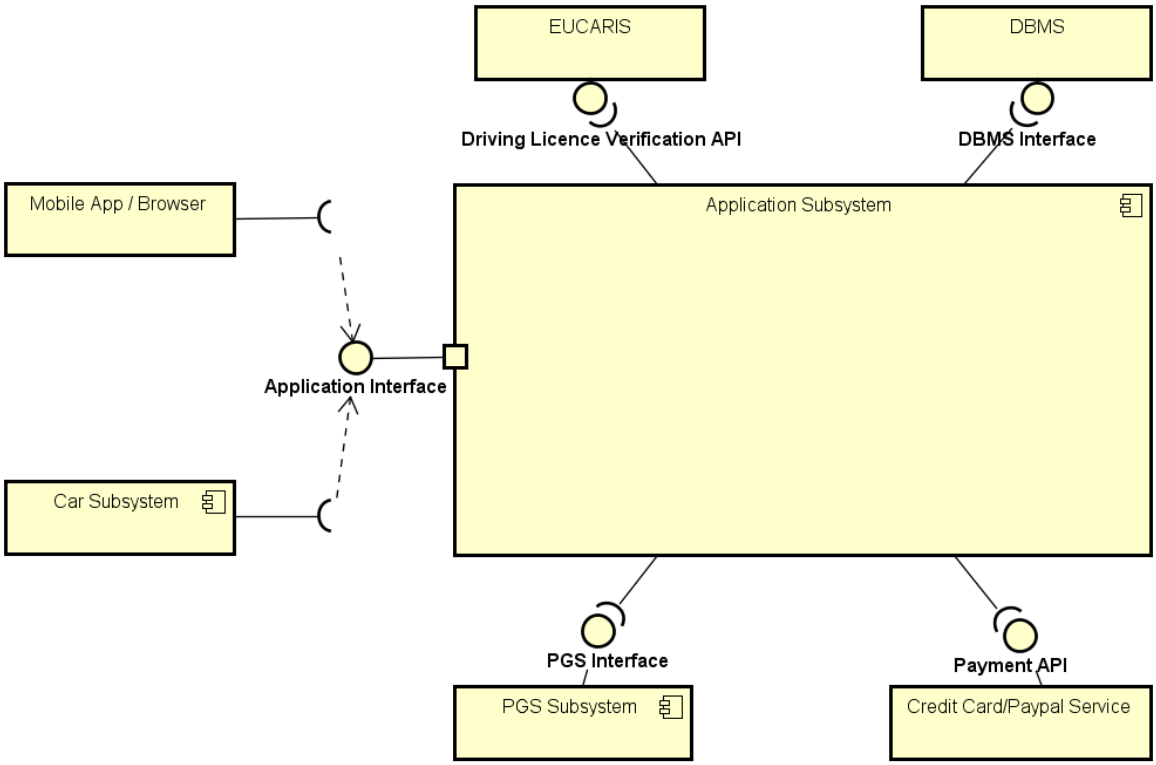

Figure 2: Component Diagram Overview

- **EUCARIS**: the application system implements API to interact with the Driving License Information System in order to verify user's documents during registration phase.
- **PGS Subsystem***:* the application running on the Power Grid Stations. Since all of this technology is implemented and designed by an external company we don't have much detail about its specifications. Interaction between the components is still possible due to a custom made interface.
- **Credit Card/PayPal Service:** through our implemented payment API we are able to connect to one of the numerous real-time payment services available. (e.g. Stripe)

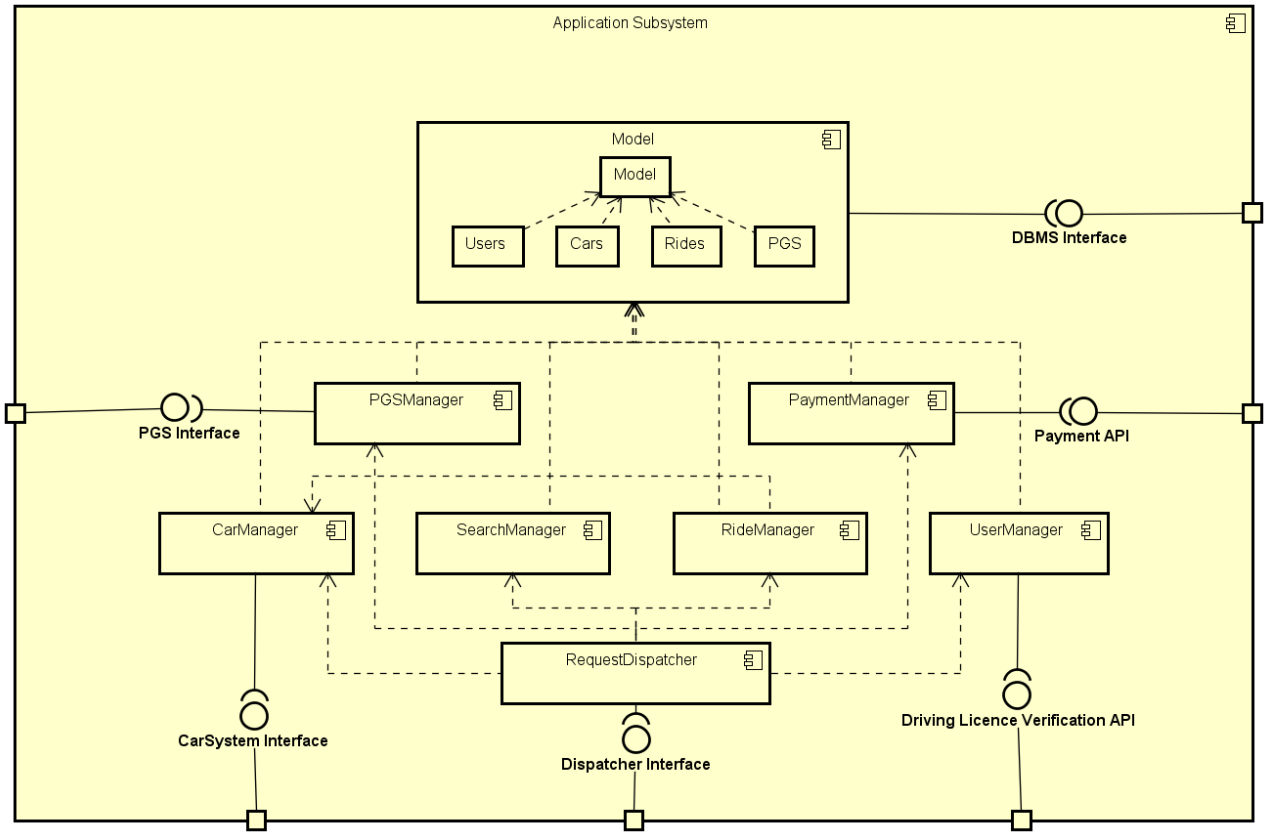

Figure 3: Application Subsystem Component Diagram

- **RequestDispatcher:** this component accepts all of the requests issued by the user applications.
- **CarManager:** the only component able to communicate directly to the car subsystems through its dedicated interface.
- **SearchManager:** this component elaborates the user position or target address and replies to his request providing all the nearby available elements (being these cars or PGS.)
- **UserManager:** handles login and registration processes by verifying data correctness and document validation.

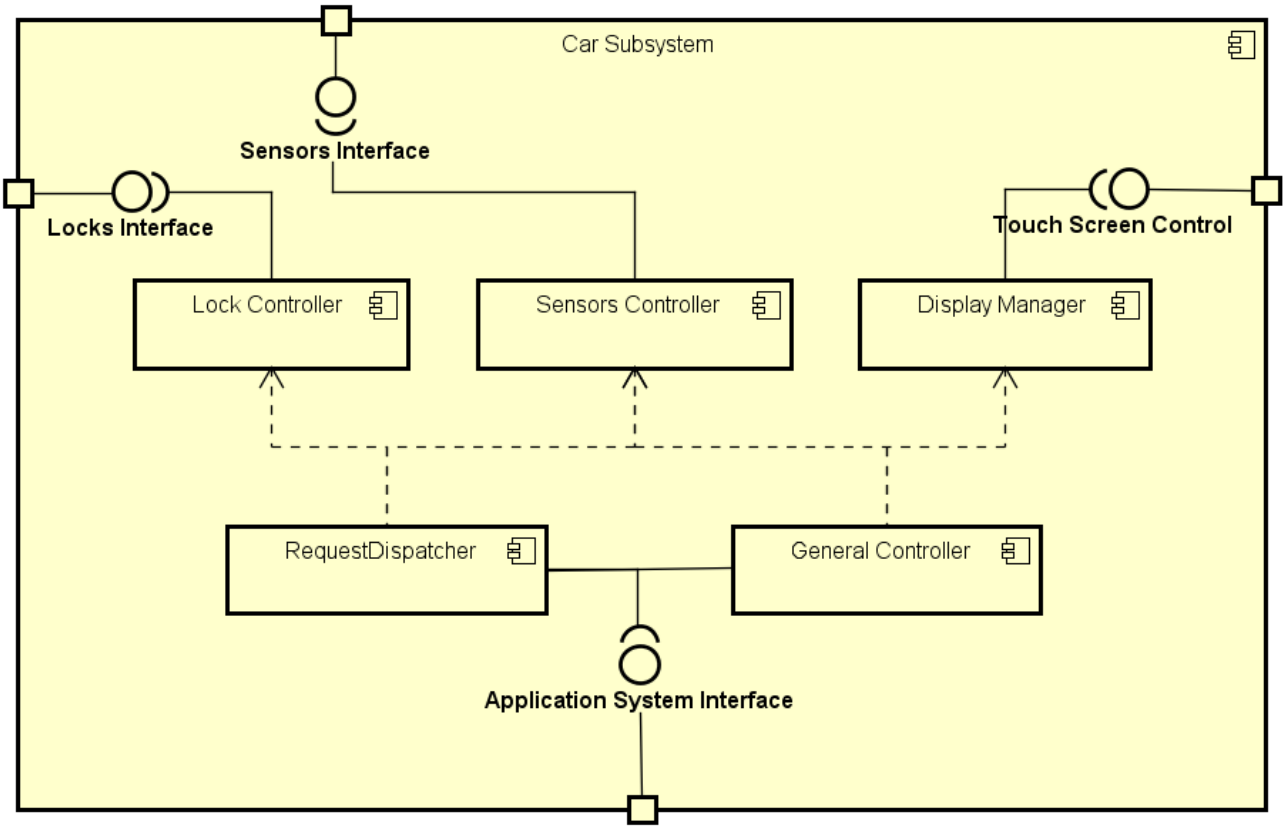

Figure 4: Car Subsystem Component Diagram

- **General Controller:** this component takes care of all the basic and continuous operations necessary for the car idle state (e.g. send car location to Central System.)
- **Lock Controller:** an entire software component is dedicated to car's door-closing system since this represents a more sensitive and critical operation.
- **Sensors Controller:** this component is directly connected to the CAN bus ready to read and interpret each sensor output data. It can then notify the General Controller in order to display requested information (e.g. battery charge level, car accurate position.)

2.3 Deployment View

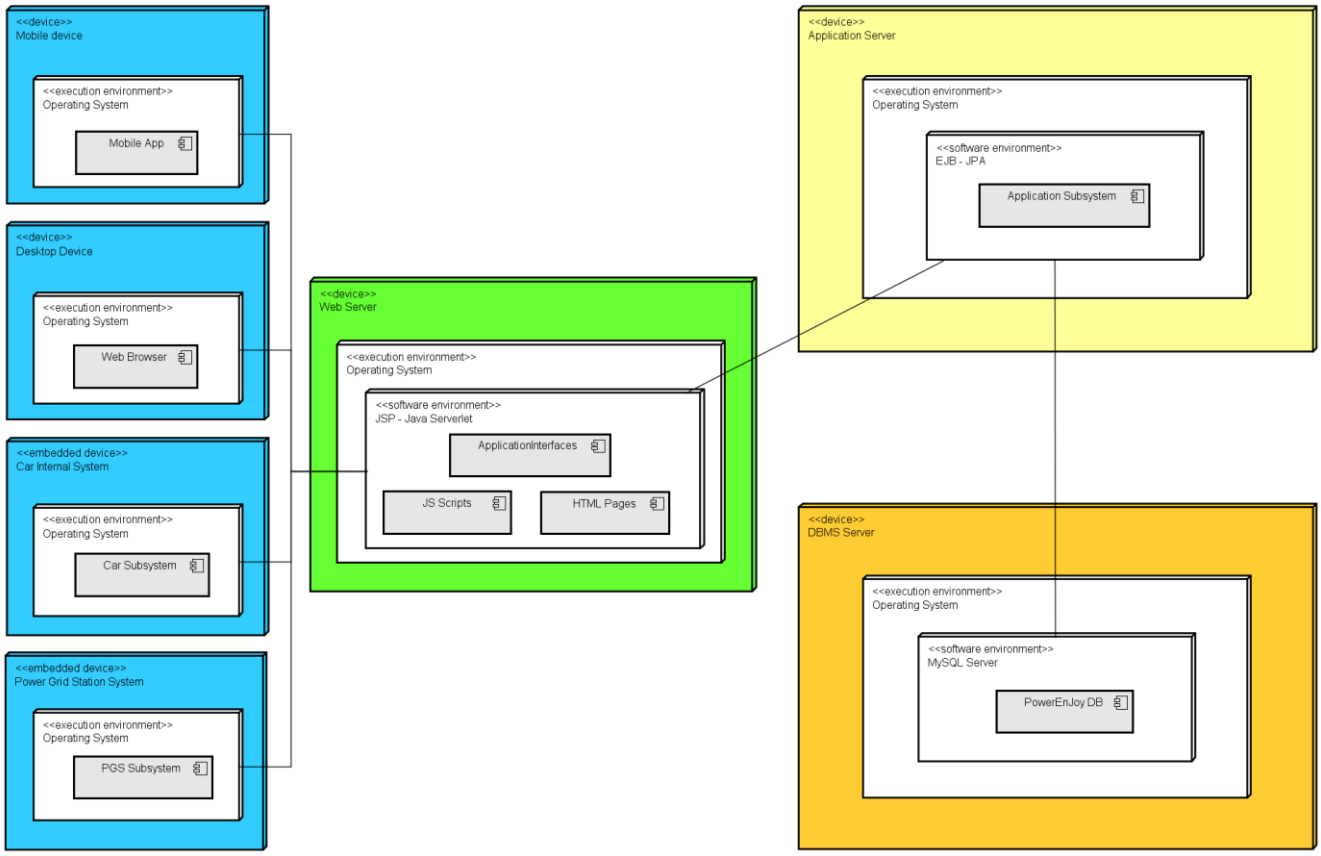

Figure 5: Deployment Diagram

2.4 Runtime View

Some of the most significant operations are represented in this section with the help of detailed sequence diagrams. Interfaces nodes and connections are omitted to get a more significant understanding of the logical process behind the proposed tasks.

2.4.1 Login process

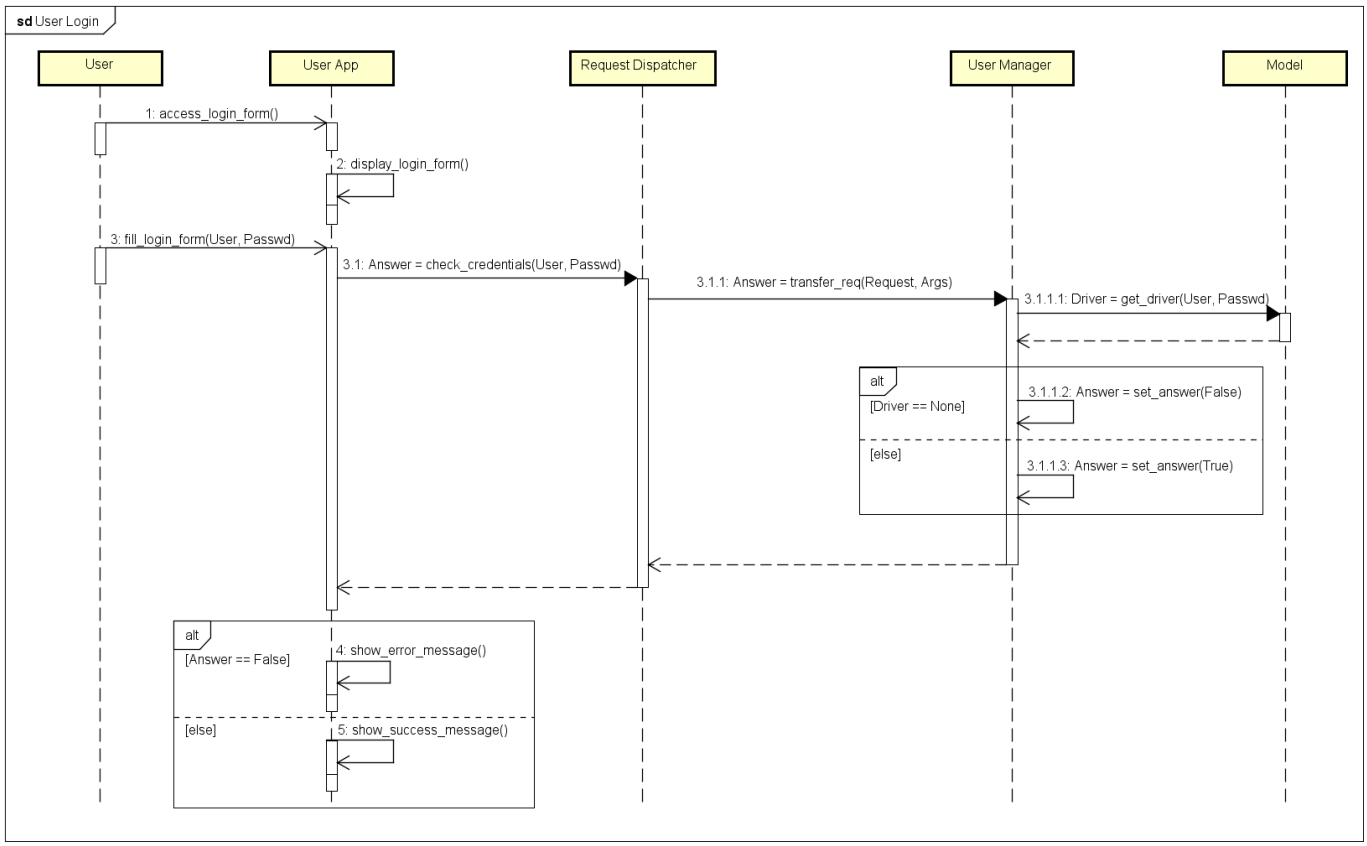

Figure 6: Login

2.4.2 User Car Search

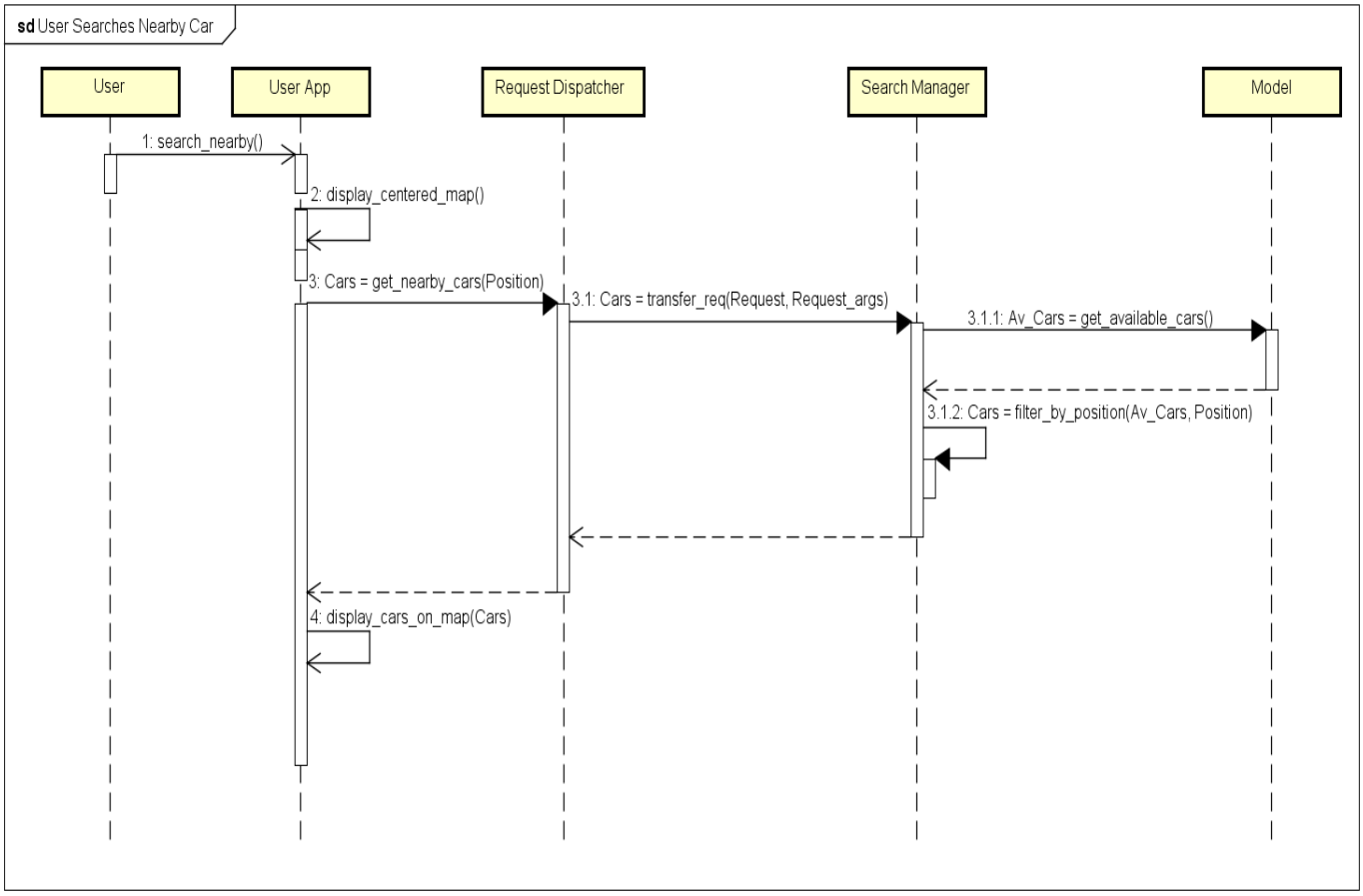

Figure 7: User Car Search

2.4.3 Car Sends Position

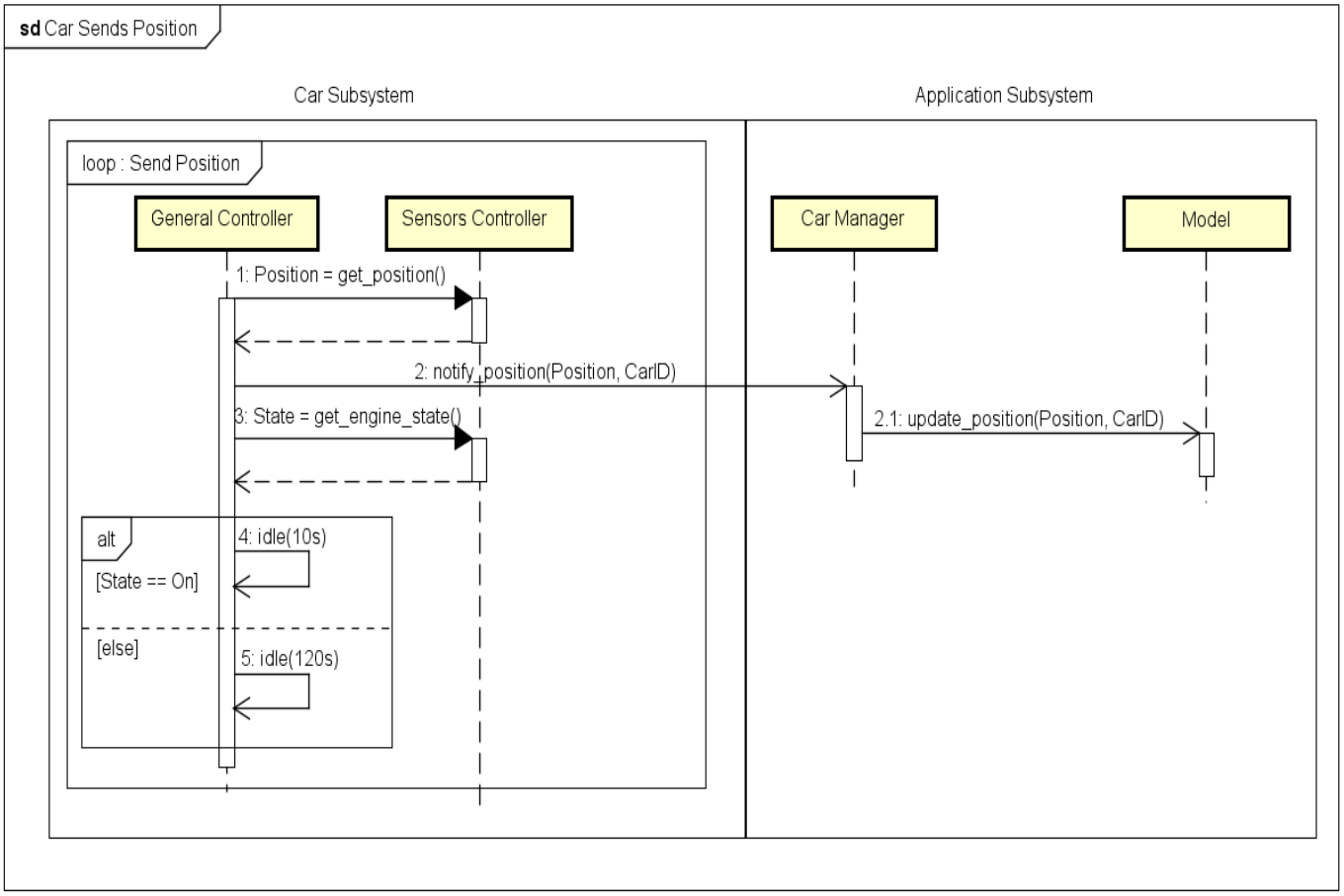

Figure 8: Car Sends Its Position

2.4.4 User Selects Car

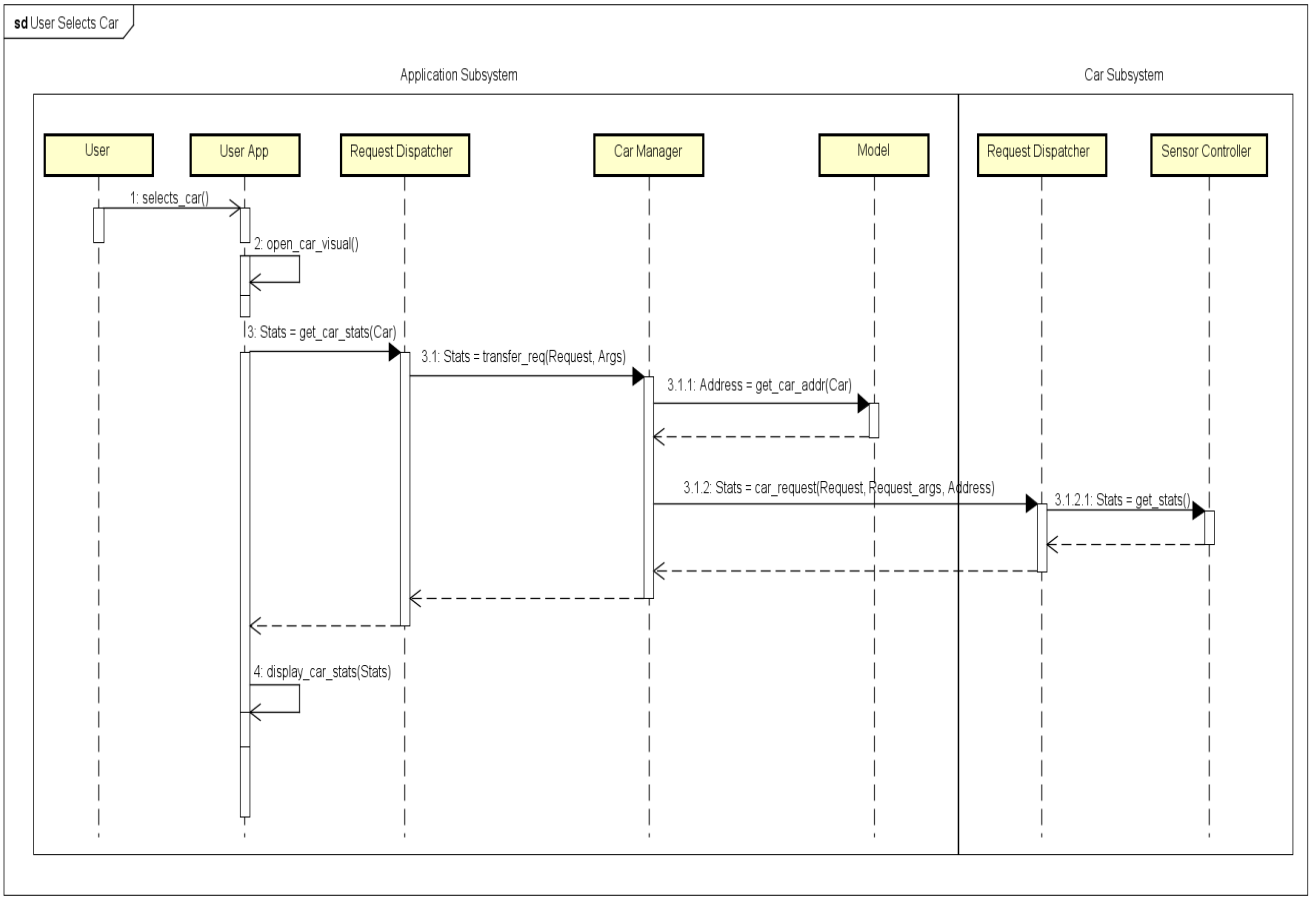

Figure 8: User Selects Car

2.4.5 User Reserves Car

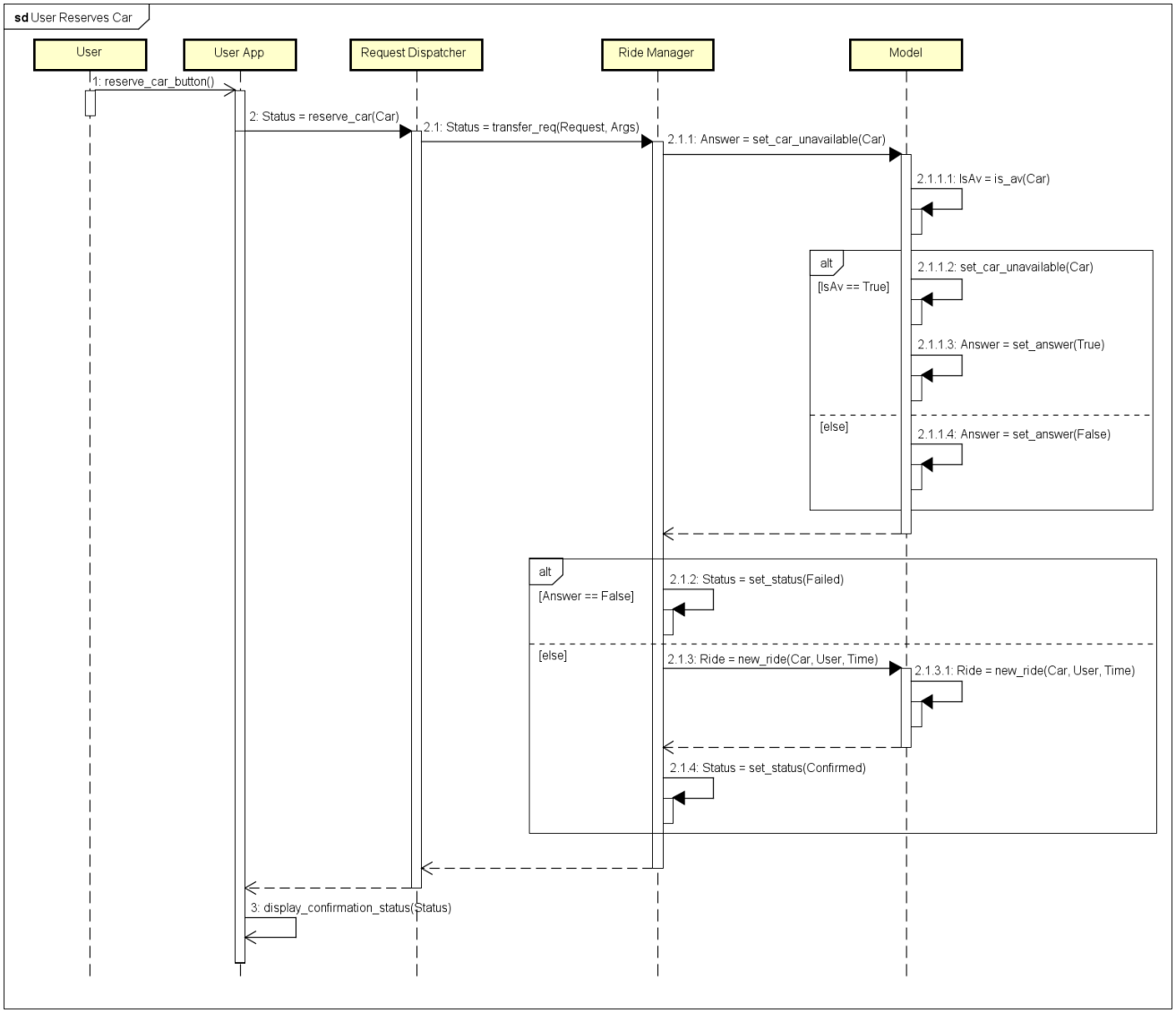

Figure 9: User Reserves Car

2.5 Component Interfaces

Figure 10: Component Interfaces

2.6 Selected Architectural Styles and Patterns

2.6.1 Architectural Style

- **Client-Server Architecture**: the well-known model is chosen due to its typical advantages from *scalability*, *accessibility* and *maintenance* points of view.
- **Service Oriented Architecture**: SOA architecture is chosen to ensure flexibility between the various system components.
- **Four Tiered Architecture**: the reasons behind this choice (better described in section **2.1**) are essentially to improve performance and security for the whole system.

2.6.2 Design Patterns

- **Model, View, Controller (MVC):** this design pattern allows to successfully and efficiently relate the user interface with the underlying data models. The Model, which constantly represents the state of the system is updated by the Controller which later asynchronously dispatches the View for user interface changes. In our specific implementation the View section is completely separated from the other components and resides on the user applications, Model and Controller are part of the Application Subsystem.

2.6.3 Protocols

All of the external interfaces are hosted by the Web Server Machine on what we defined as our Web Tier. All clients send HTTPS requests to a specific endpoint and receive JSON formed responses.

Authentication is performed using session tokens to send along the HTTPS request to access login-required pages and functionalities.

Along with the JSON response object a response code is returned to express the request outcome:

- 200 OK: success
- 400 Bad Request: request failed or not found
- 401 Authorization Required: auth token not found
- 500 Internal Server Error: server problem

Listed below some common application requests:

- *Registration:*
  - *Path*: /registration
  - *Params*: email, name, surname, license_id, licence_pic, dob, address, payment_method
  - *Response*: 200 OK if successful
- *Login:*
  - *Path:* /login
  - *Params:* email, password
  - *Response:* authentication_token
- *Find cars:*
  - *Path:* /find_cars
  - *Params:* latitude, longitude
  - *Response:* (lat, long, car_id) [ ]
- *Get car info:*
  - *Path:* /car_info
  - *Params:* car_id
  - *Response: (battery_level, address)*

3 Algorithm Design

In this section a couple of application algorithms are proposed using a standard pseudocode format.

3.1 Discount calculation

The algorithm below shows how the discount rate applied if the user finished his ride with more than one passenger on-board.

```
import rideController as rC
import carController  as cC

def check_discount(ride):
      # Number of passengers at each acquisition
      passengers_num = []
      # Number of good acquisition (more than 2
      # passenger, diver included)
      good_acquisitions = 0
      # The car object is save in the ride
      # argument passed to check_discount()
      ride_car = rC.getcar(ride)
      # The ride status is updated by the car
      # it self so reading it from the ride element
      # is safe to assume correct
      ride_status = rC.getStatus(ride)

      while(ride_status == rC.status.IN_USE):

      passengers_num.append(cC.getPassengers(ride_car))
            sleep(20)   # sleeps 20 seconds
            ride_status = rC.getStatus(ride)

      for acquisition in passengers_num:
            if acqusition > 2 :
                  good_acquisitions += 1

      # More than 2 passenger must be present for
      # more than 60% of acquisition
      if(len(passengers_num) * 0.6 >=
good_acquisitions):
            return False
      else:
            return True
```

4 User Interface Design

4.1 Mockups

Mockups can be found in RASD, section **3.1.1**

4.2 UX Diagrams

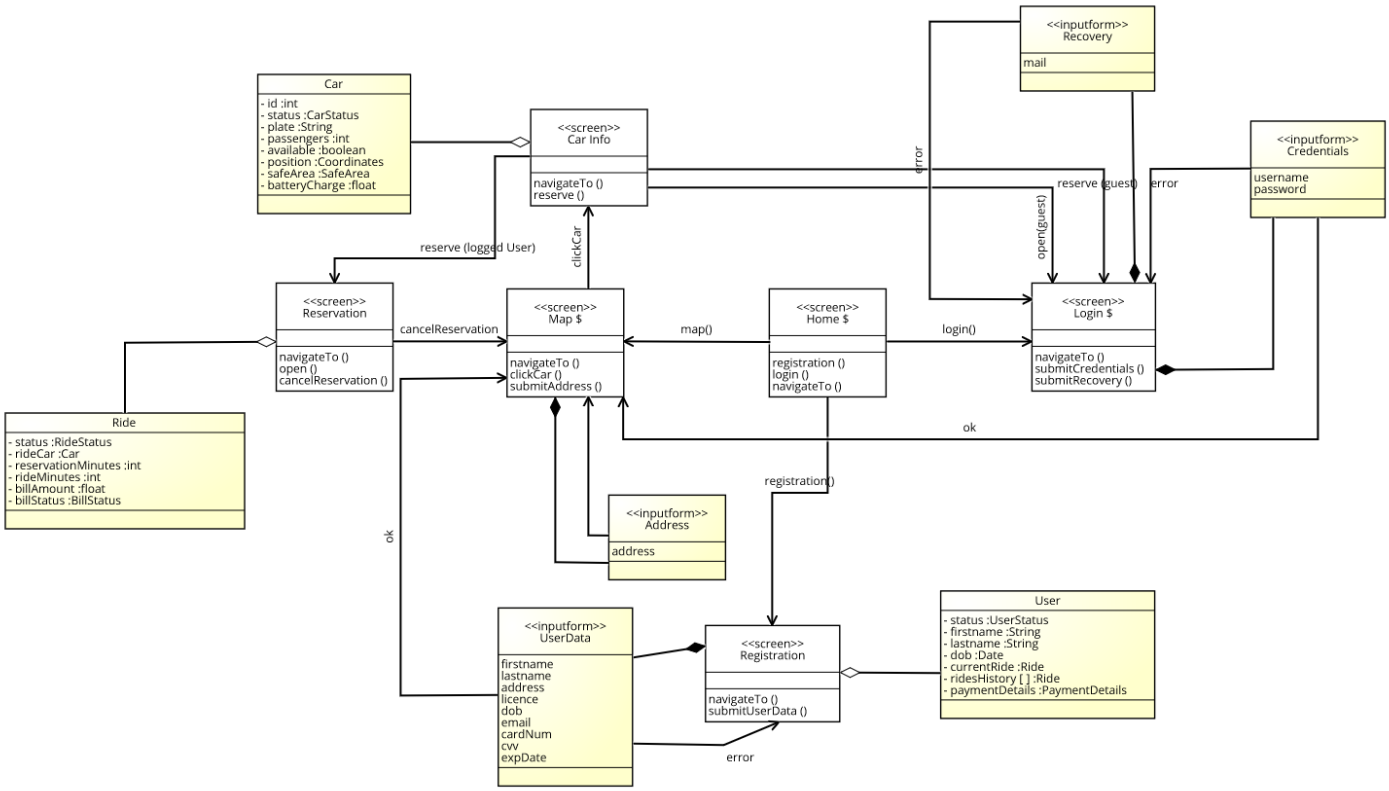

Figure 11: Web Application UX Diagram

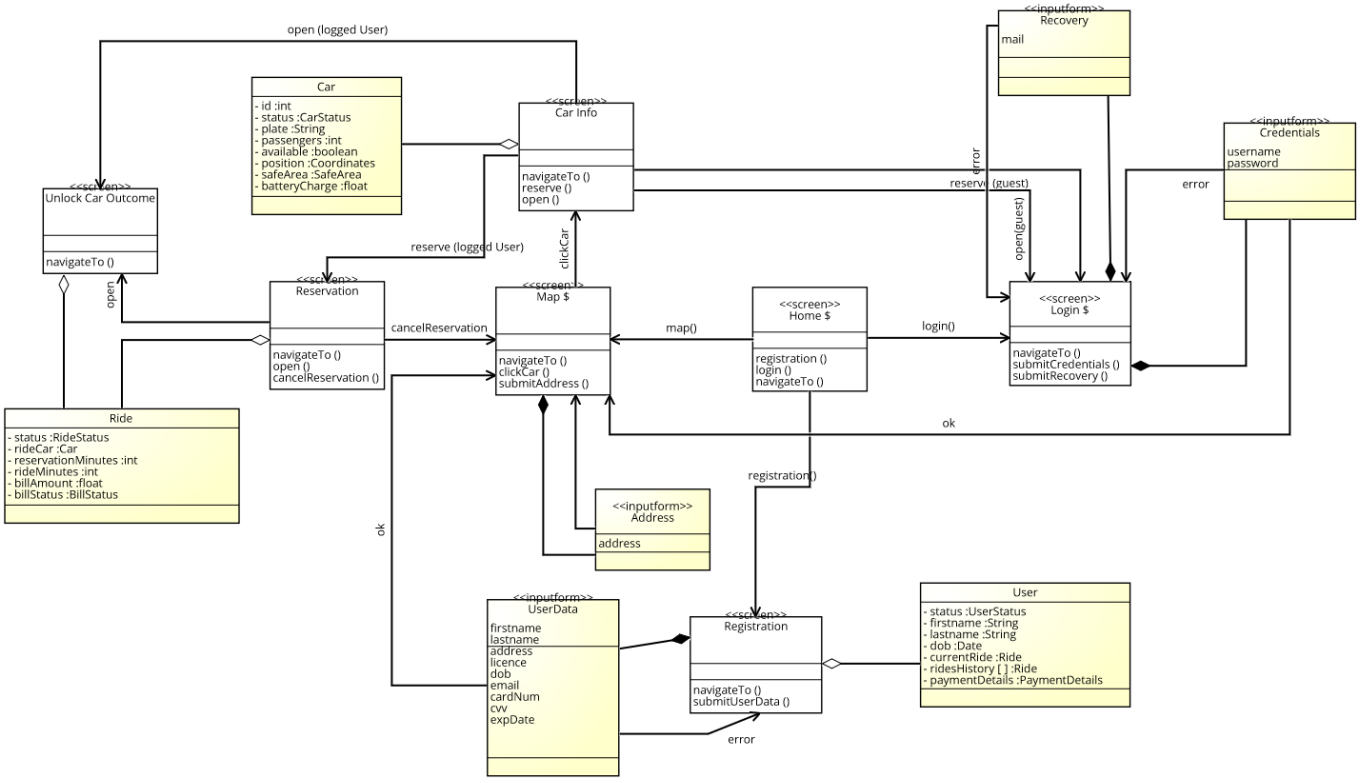

Figure 12: Mobile Application UX Diagram

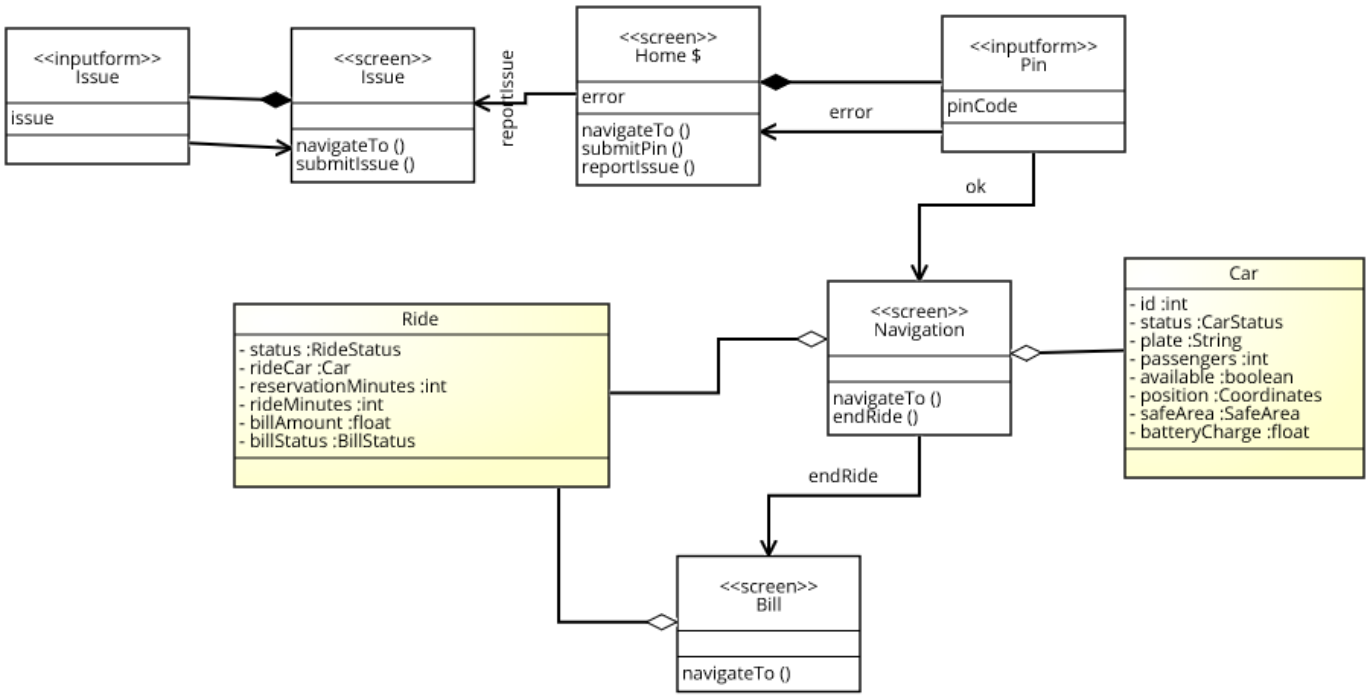

Figure 13: Car Application UX Diagram

4.3 BCE Diagram

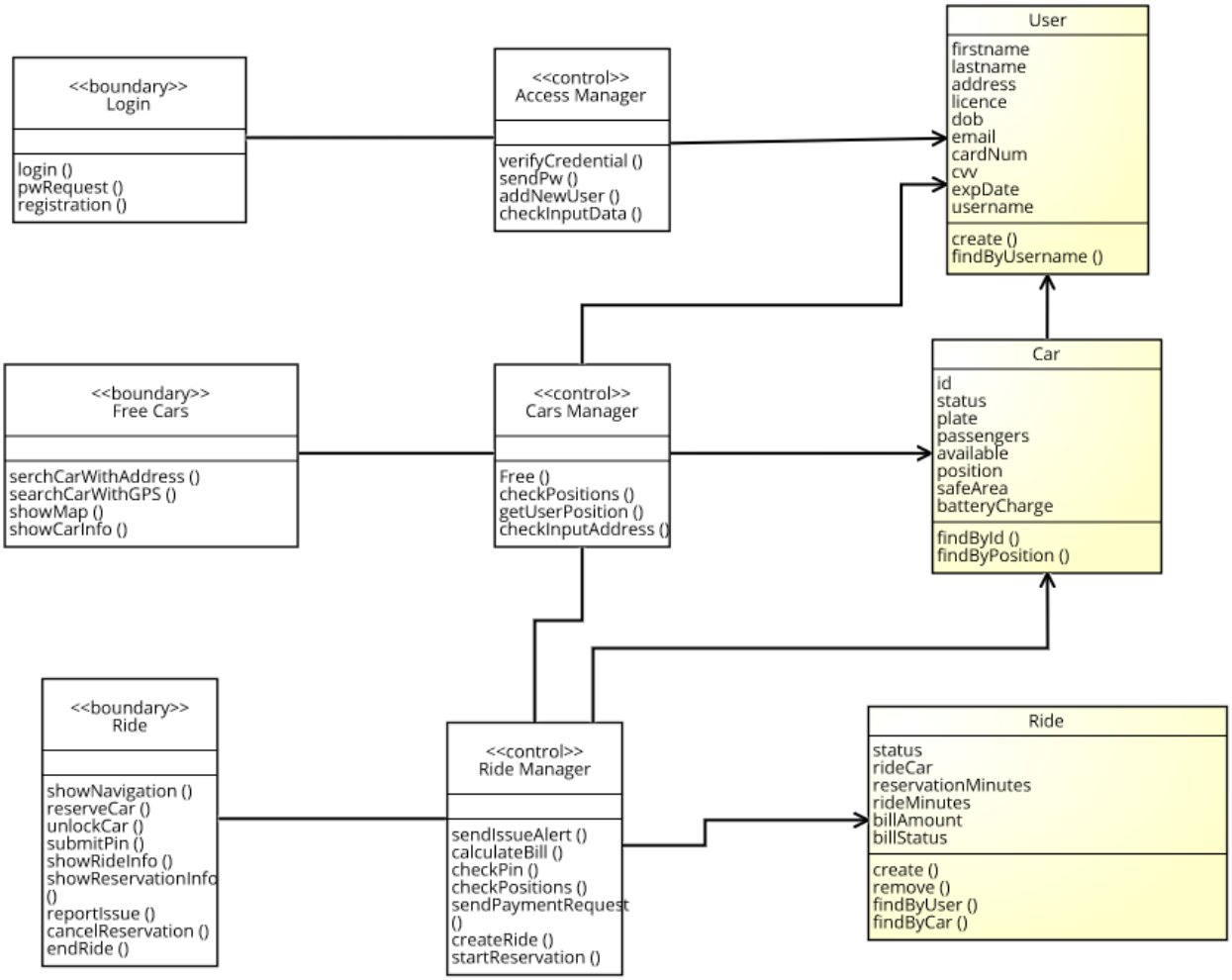

Figure 14: BCE Diagram

5 Requirements Traceability

6 Appendix

- **Tools used:**
  - Signavio for UX diagrams
  - Astah for Deploy, Architecture, Sequence diagrams
- **Working hours division:**
  - Marco Festa: 55 hours of work
- **Revisions:**
  - DD v1.0 published December 12, 2016