POLITECNICO
MILANO 1863

PowerEnjoy

Design Document

Version 1.0

Software Engineering 2 (A.A. 2016/2017)

- Simone Boglio (mat. 772263)
- Lorenzo Croce (mat. 807833)

Summary

1. Introduction

1.1. Purpose

This document presents the architecture on which PowerEnjoy will be developed, it describes the decisions taken during the design process and justifies them.
The whole process is reported including all the improvements and modifications to provide additional information in case of future changes of the architecture structure.

1.2. Scope

This document will focus on the non- functional requirements of PowerEnjoy. Since the system architecture defines constraints on the implementation, this document will be used to provide fundamental guidelines in the development phase of PowerEnjoy.

1.3. Definitions, Acronyms, Abbreviations

The following definitions are used in this document:

- Thin client: is a computer that depends heavily on another computer (its server) to perform its computational tasks.
- Server: is a type of server that provides most of the functionality to a client's machine within a client/server computing architecture.
- Event-driven architecture: is a software architecture pattern promoting the production, detection, consumption of, and reaction to events, that is a change of state of an object.
- Availability: is the "status" of a PowerEnjoy car, it can be:
    - Available: the car is ready for a new reservation by the customer.
    - Unavailable: the car is not available in the system, for example when car need external maintenance, or it is moved from one area to another to balance the availability of the cars in the city.
    - In use: the car is used by a customer for a ride.
    - Reserved: the car is waiting for the customer who reserve its and cannot be reserved or use by another user.
- Area: the city of Milan is partitioned in different type of area.
    - Safe Area: the user can finish is ride leaving the car in one of this area.
    - Special Area: like safe area, also in this type of area the car can be left in charge in one of the power grid station.
    - Unsafe Area: the user cannot end is ride in this type of area.

The following acronyms are used in this document:

- JEE: Java Enterprise Edition
- RASD: Requirements Analysis and Specification Document
- UI: User Interface
- API: Application Programming Interface
- CU: Control Unit
- GMapsAPI: Google Maps API
- HTTPS: HyperText Transfer Protocol over Secure socket layer
- JSON: Javascript Standard Object Notation
- EIS: Enterprise Information System

1.4. Reference Documents

- Specification document for Software Engineering 2: PowerEnjoy project.
- Template for the Design Document.
- IEEE Standard 1016-2009 - IEEE Standard on Design Descriptions.
- Requirements Analysis and Specification Document (RASD): PowerEnjoy.

1.5. Document Structure

This document specifies the architecture of the system using different levels of detail, it also describes the architectural decisions and justifies them.
The design is developed in a top-down way, then the document reflects this approach.
The document is organized in the following sections:

1. Introduction
   Provides a brief of the architectural descriptions.
2. Architectural design
   The core of the design document, in this section are presented all the components of the system and the interaction between them, in increasing level of detail starting from a high-level overview.
3. Algorithm design
   Pseudocode and flowchart of the fundamental algorithms of PowerEnjoy.
4. User interface design
   Mock-ups of the UI to understand better how the functionalities will been implemented from a graphical viewpoint.
5. Requirements traceability
   The mapping between the requirements and the components in the architectural design used to satisfy them.
6. References
   List of sources used in this document: internet links, documents.

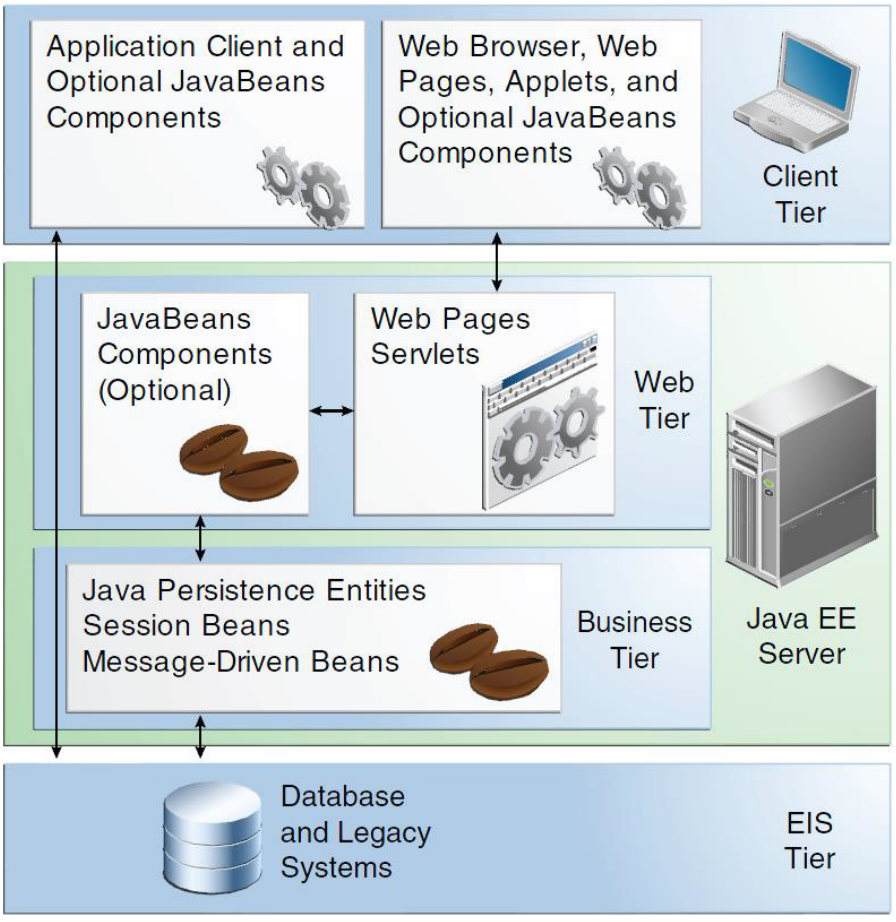
2. Architectural Design

2.1. Overview

In this section, there is a general design description of the PowerEnjoy service.

2.1.1. PowerEnjoy Platform

The service adopts the very-known architecture "Client-Server" that means there are two main parts: the client, it sends a request; the server, it sends back to client the results of his request.
Furthermore, PowerEnjoy adopts a JEE platform.
JEE Platform is organized into four levels (four tier) and these levels interacts with each other to reach the goal. These tiers can be installed in different machines.



The picture above explains how the JEE platform is organized.

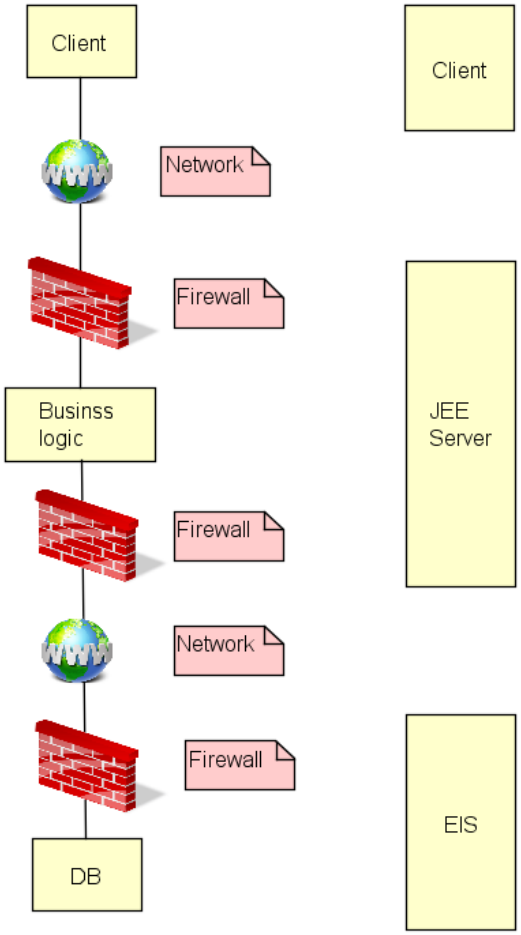
- **Client Tier:** this level is designed to interact directly with the users of the service. Client must use the dedicated app (over his smartphone) or a web browser to access the Power Enjoy's pages.
- **Web Tier:** this level is designed to receive requests from the client tier and send data to the Business Tier. At the end, the Web tier send back to Client tier the response.

- **Business Tier:** this level is designed to receive data from the Web Tier, process they, generate a response and send it back to Web Tier. This tier can access to EIS tier if is it necessary.
- **EIS Tier (Persistence tier):** this level is designed to store all service data (for example in a database)

The Web Tier and the Business Tier can be represented together in one tier called "Business logic tier". The structure is:

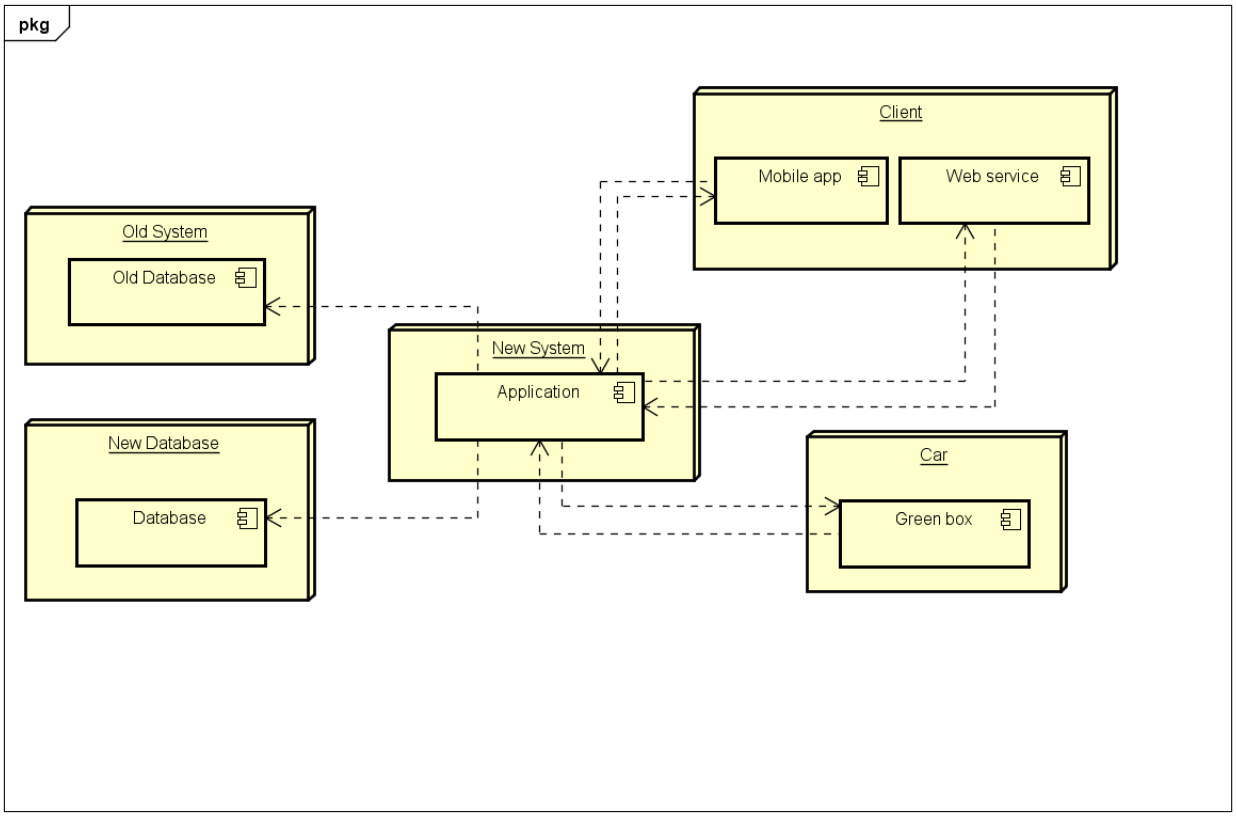
- Client Tier
- Business Logic Tier
- EIS Tier

There are clients, a JEE server and a Database (two, one of this is the old database). Remember that PowerEnjoy is an online service. The picture below explains how the system is distributed.



2.1.2. High-Level Component

Here is showed a high-level representation of the interactions between all the system's parts.



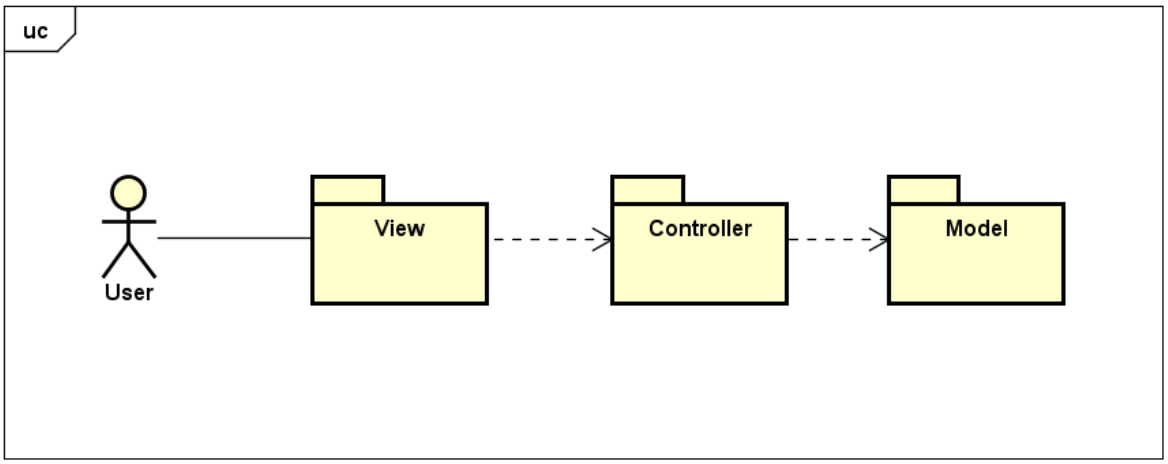
These are the main elements:

- New System
- Client
- Car
- Old System
- New Database

The main element is the "New System". It's a singleton and it's connected with all the other elements. The New system receives request from clients, it processes them and it interacts, if is it necessary, with databases and it send back the results.
The client can access service using the mobile app (Android or IOS) or web browser (by a PC or a smartphone). All requests start from client and are sent to New System.
Every car communicates with New System by a Green Box installed on board, in this way the "central" system can interact with them.
The Old System represent the old database where are stored information about cars, PowerEnjoy will continue to use this system since it is used by other external service (such the car maintenance company for example).
The New Database is used to store information about client registrations, reservations and rides.

2.1.3. MVC

The picture below explains (in a high-level view) how the user can interact with the system.



The packages roles are:

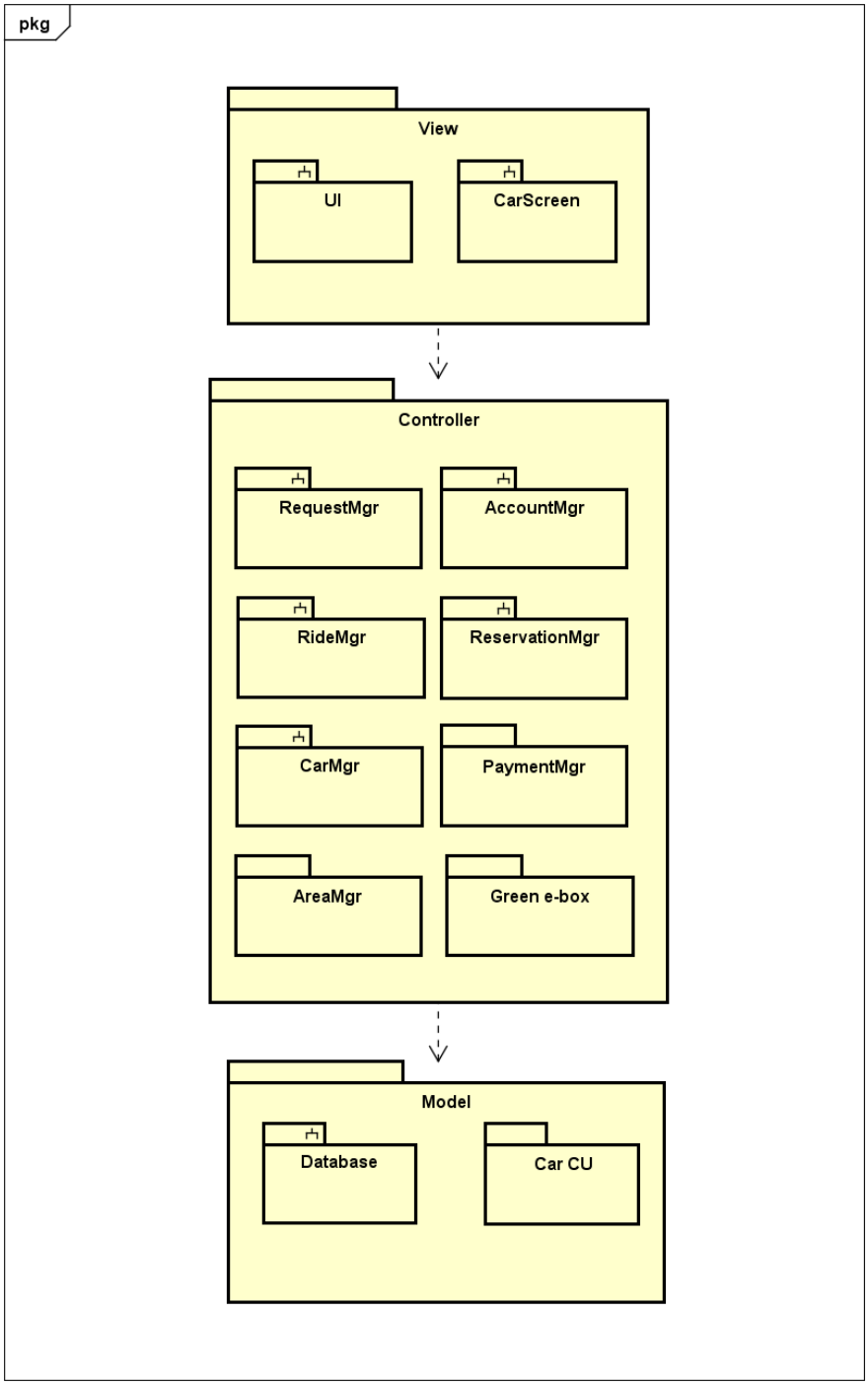
- **View:** this package contains the User Interfaces (UI) and by these interfaces he can interact with the system. This package allows user to send request to the system, obtains response about them and knows information about current ride.
- **Controller:** this package contains the business logic of the system. It receives requests from the view, processes them and sends back a response.
- **Model:** this package contains all data (it is the Persistence level) of the system.

The user can access only to the view. The other packages are hidden to him.

The picture above describes a MVC Pattern used to deploy PowerEnjoy service. (It is described in the rest of this document).

2.1.4. Packages

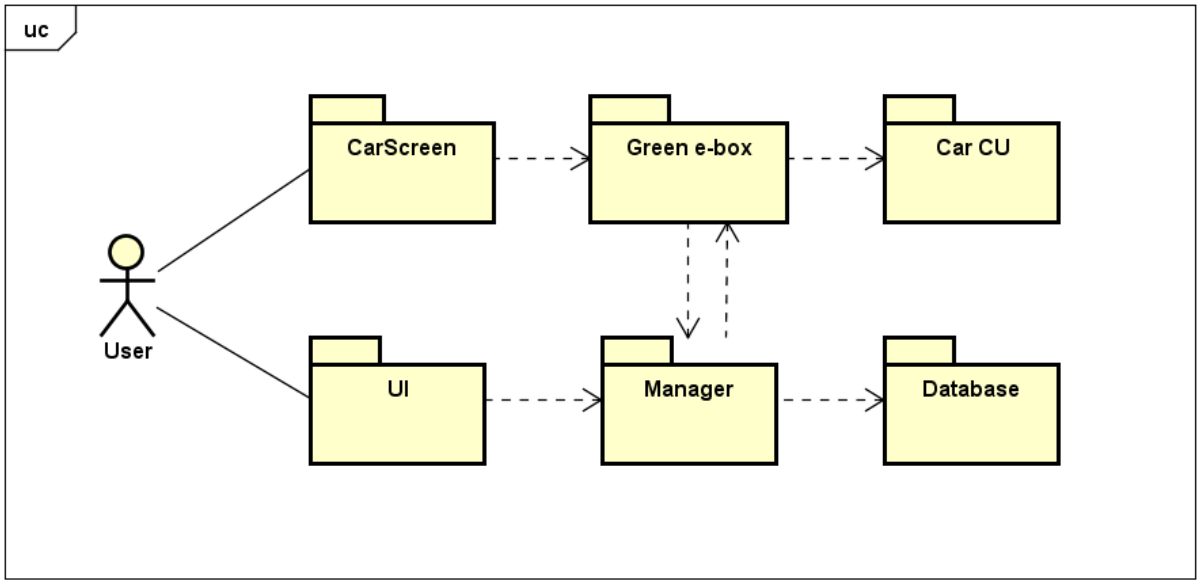
Each package is composed by other packages. Every package is specialized to manage some operation and they are connected to each other.



The "CarScreen", "Green e-box" and "Car CU" packages represent the physical car.
In each car, there is a Green e-box that communicates with the central and sends information about the car CU. Furthermore, the Green e-box shows information to user about the ride by the screen (CarScreen) installed on board.
The presence of the Green e-box is hidden to the user; he can only see the information on the car screen.

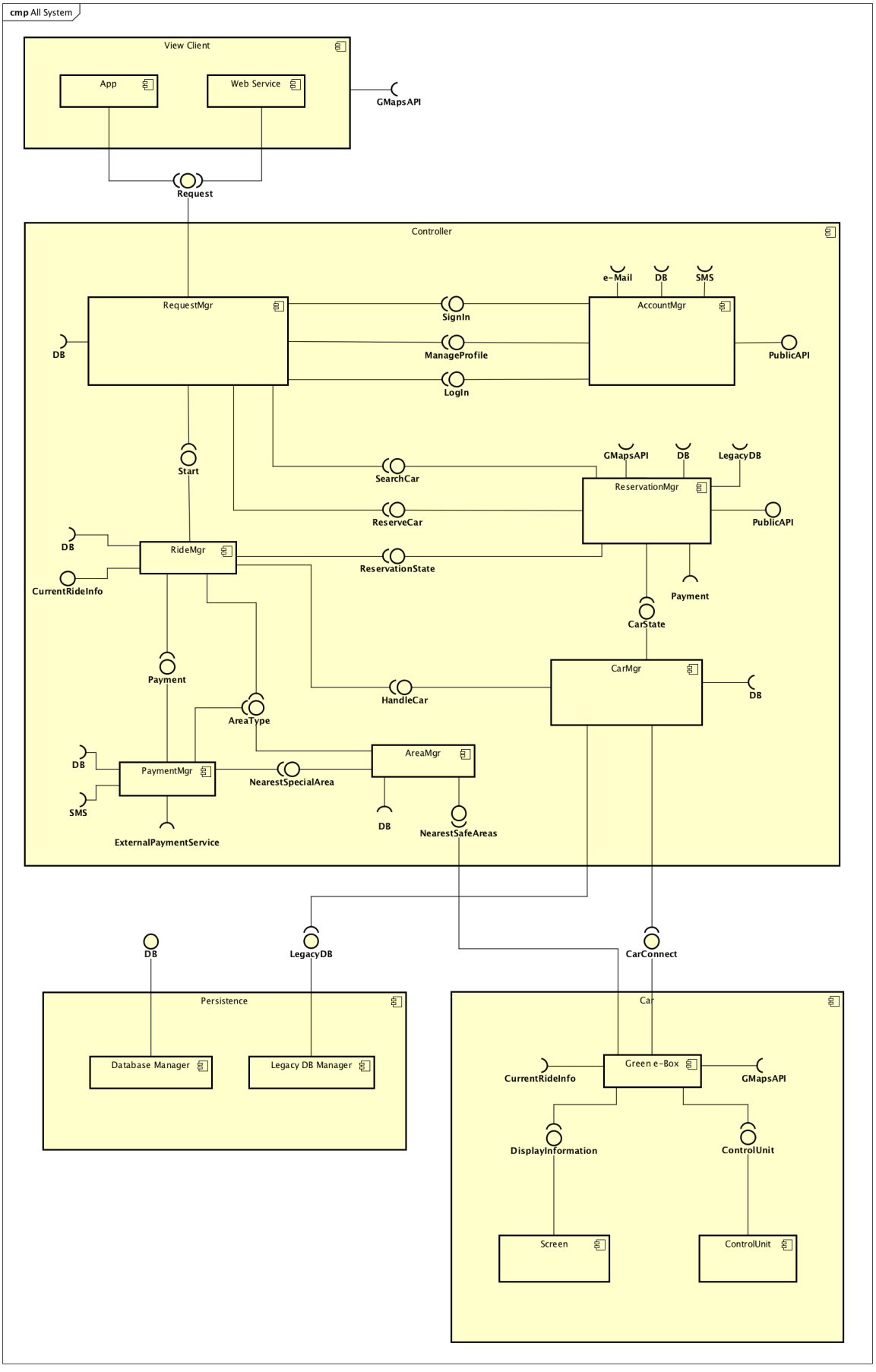
2.1.5. High-Level Packages Interaction

The next picture shows a high-level view of the interaction between user, car and the central system.



2.2. Component View and Interfaces

The picture below represents the main components and interfaces of PowerEnjoy service.



From the component diagram, we can see the main part is the controller of the server, it contains all the logic of the system, it is composed by:

- The RequestManager is the component in charge of binding the incoming requests (from app or web service) with the user sessions present in the system, after this step it dispatches every request in the right component.
- The AccountManager is the component in charge of offering the sign-up functionality to the visitors, the login to the registered users; it also offers the functions related to the managing of the user's profile data.
- The ReservationManager is the component that offers the search of PowerEnjoy cars and allows the reservation of them for a ride.
- The CarManager is the component in charge of manage all the car through the CarConnect interface, also it provides all the necessary data to the RideManager.
- The RideManager is the component that manages all the rides of every user.
- The PaymentManager is the component in charge of calculate the ride cost (applying bonus if necessary) and it exchanges the information about the transaction with the external payment service provider.
- The AreaManager is the component in charge of distinguishes the type of area from a generic GPS coordinate and it also provides the list of nearest safe area from a GPS position.

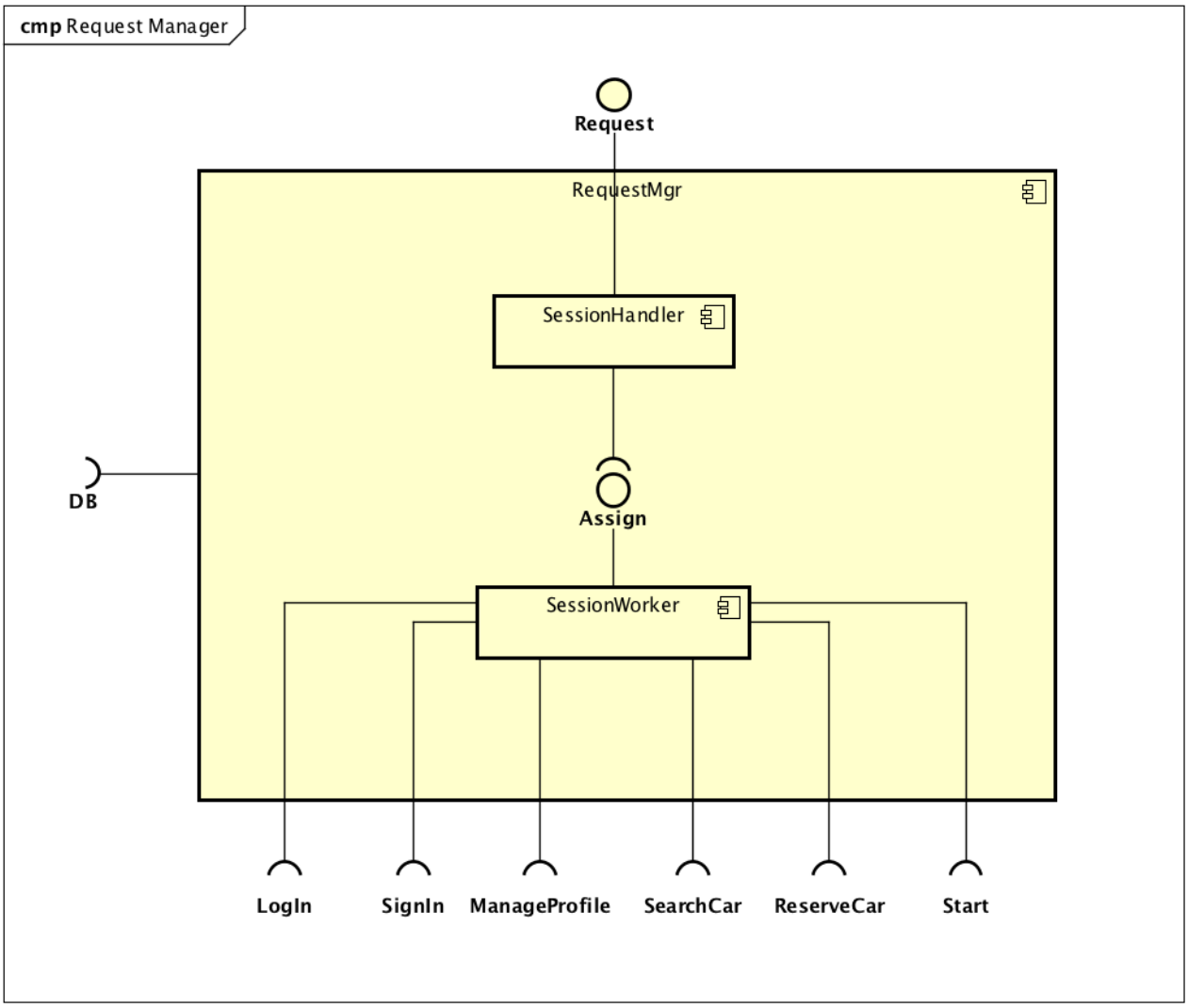
The other parts are:

- The thin-client: it is composed by view (app and web-page) and communicate with the controller through the Request interface.
- The model of the system: the new database and the old database, here are physically stored all the persistent data used by the system.
- The physical car: in according with the MVC it is divided in screen (view), control unit (model), green e-box (controller). The physical car is managed by the server controller through the CarConnect interface provided by the car's Green e-Box.

In the following chapters, we analyse the main components more deeply, we explain all the interactions and the interfaces used by each one.

Note: in the next component diagrams for simplicity when we write that a component receives a request from the user, we omit the fact that the request is processed before by the request manager.
For example, if a user wants reserve a care we say the ReservationManager receives the request from the user, but in real, all the steps are:
User -> View -> RequestManager -> ReservationManager

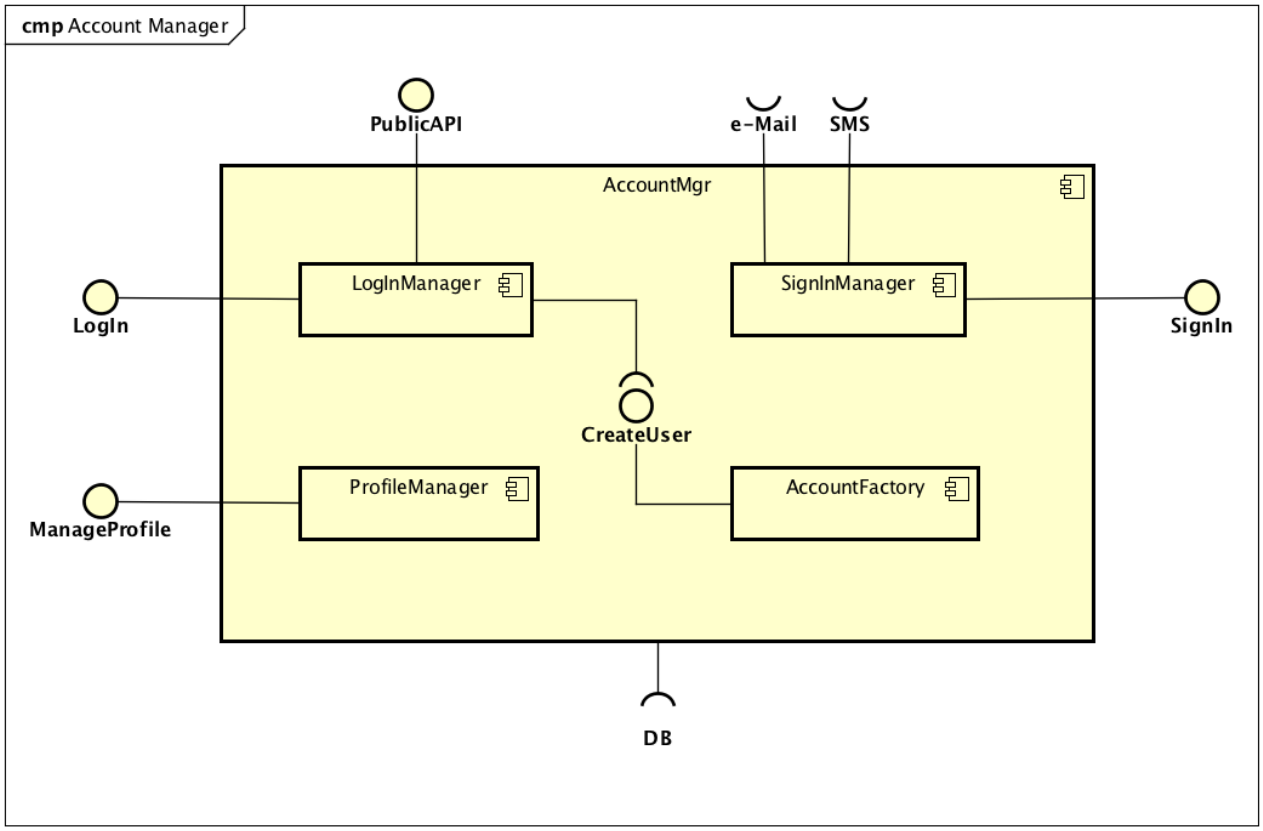
2.2.1. Request Manager



SessionHandler	
Definition	Component that manage all the request from the registered users and unregistered users.
Responsibilities	This component starts a SessionWorker for each user and dispatch the request in the right worker during all the users' session (binding each time the users' request with the respective RequestWorker). It stores and retrieve data from the DB.
Interaction	With the user client (app or web) and the DB.
Interfaces offered	• Request for the View component (App or WebService)
Interfaces required	• Assign of the SessionWorker
Implementation	Factory method pattern and pool manager

SessionWorker	
Definition	Component that check and dispatches the request of one user (registered and unregistered).
Responsibilities	This component manages the user session data and dispatch the request in the right component of the system.
Interaction	With the AccountManager, the ReservationManager, the RideManager.
Interfaces offered	Assign for the RequestHandler
Interfaces required	• Login • SignIn • ManageProfile • SearchCar • ReserveCar • StartRide • DB
Implementation	Multi instance: one for each user active session.

2.2.2. Account Manager



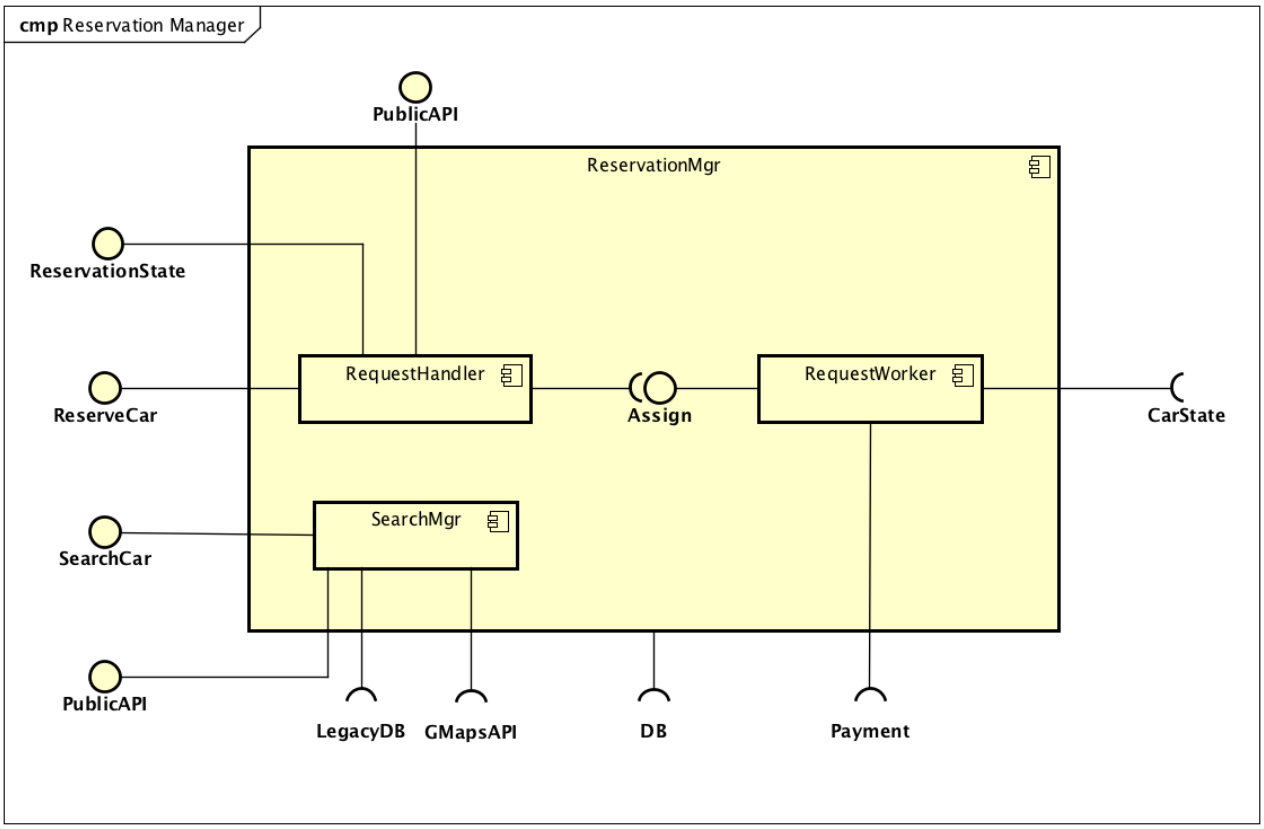
SignInManager	
Definition	Component that controls the sign in of the unregistered users.
Responsibilities	This component allows visitors to sign in into PowerEnjoy service. It connects to the DB to stores the credential and it requires email and SMS interfaces to send back the password to complete the sign in process of the visitor.
Interaction	With the visitors, the DB, the email service and the SMS service.
Interfaces offered	• SignIn for the visitors
Interfaces required	• DB • External e-mail service • External SMS service
Implementation	Static class

LoginManager	
Definition	Component that controls the login of the registered user.
Responsibilities	This component allows registered user to login into PowerEnjoy service. It is connected to the DB to verifies the credentials. It also offers the possibility to login in the service through third part apps that use the public API offers by this component.
Interaction	With the registered users, the DB, external services (API), account factory.
Interfaces offered	• Login for the registered user • Public API for third part apps
Interfaces required	• DB • CreateUser of the AccountFactory
Implementation	Static class

ProfileManager	
Definition	Component that control the users' profile.
Responsibilities	This component allows the user to edit is profile, change personal information, change password and credit card data. It retrieves and save data in the DB.
Interaction	With the registered user of the system and the DB.
Interfaces offered	• ManageProfile for the registered user
Interfaces required	• DB
Implementation	Multi instance: one for each user session (where session is intended as the entire period of time in which the user is logged in).

AccountFactory	
Definition	Components that instantiates the logged users.
Responsibilities	This component is in charge of create an instance in the system for every logged user.
Interaction	With the LoginManager and the DB.
Interfaces offered	• CreateUser for the LoginManager
Interfaces required	• DB
Implementation	Factory method pattern

2.2.3. Reservation Manager

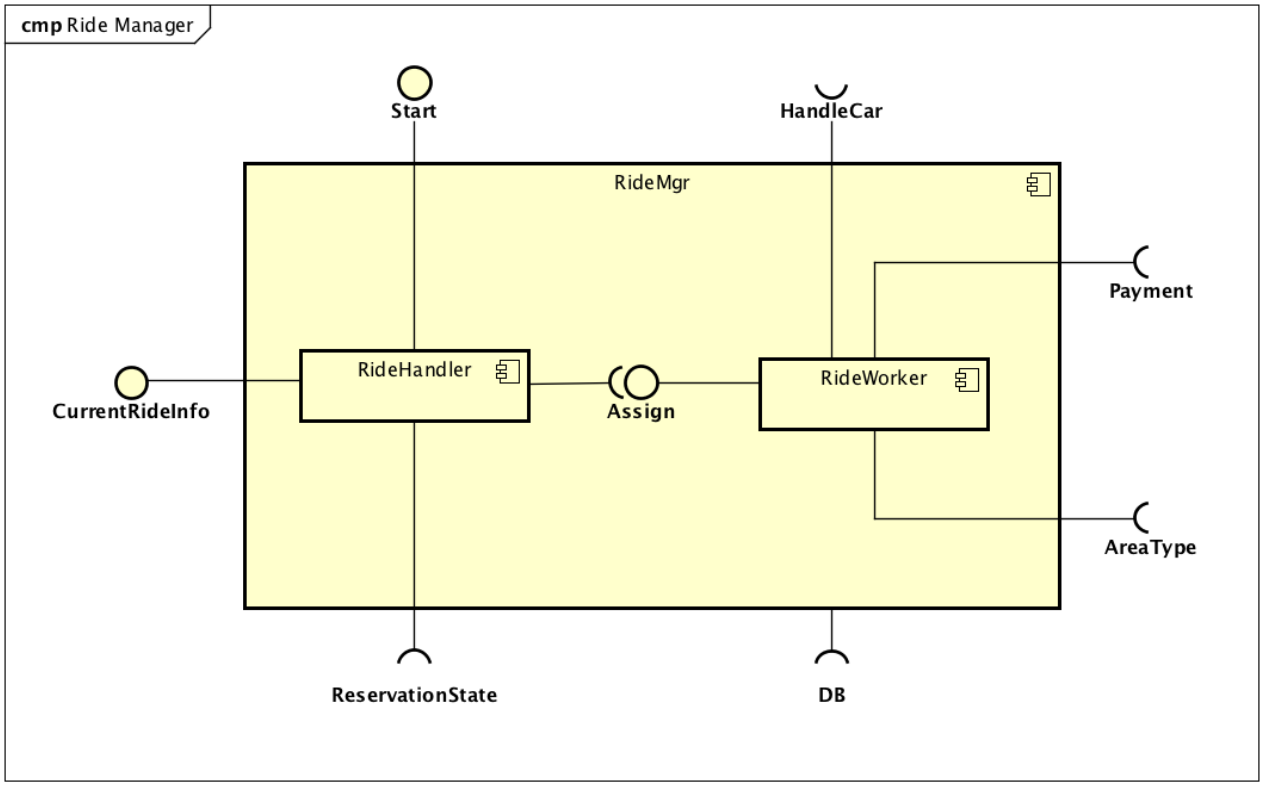


RequestHandler	
Definition	Component that receives the users' reservation.
Responsibilities	This component receives the request of reservation of the clients and for each of them starts a worker that will handle them singularly. It also offers API to allows third part apps to send the reservation request. It offers an interface to manage the state of the reservations.
Interaction	With the registered user of the system, the third part apps, the DB, the RequestWorker.
Interfaces offered	• ReserveCar for the registered user • PublicAPI for third part apps • ReservationState for the RideManager
Interfaces required	• Assign of the RequestWorker • DB
Implementation	Factory method pattern

RequestWorker	
Definition	Component that manages individually the active reservation of the registered user.
Responsibilities	There is an instance of this components for every active reservation generated by the registered users. It changes the state of the car from free in reserved (and in free from reserved when time expired). It applies the fee when the reservation expired (the user don't pick up the car in one hour).
Interaction	With the DB to store and to retrieve data, the CarManager to change the state of the cars, the PaymentManager to process the fee amount.
Interfaces offered	• Assign for RequestHandler
Interfaces required	• CarState of the CarManager • Payment of the PaymentManager • DB
Implementation	Multi instance: one for each reservation

SearchManager	
Definition	Component that allows user to search the PowerEnjoy cars in a determinate zone.
Responsibilities	This component allows users and third part apps (through API) to search the PowerEnjoy cars by a GPS position or address. It provides the list of all the nearest car with the respective GPS position.
Interaction	With the user and third part apps, the legacy DB (query for retrieve the nearest cars list), GMaps API to convert name address in GPS position.
Interfaces offered	• SearchCar for the user • PublicAPI for third part apps
Interfaces required	• GMaps API • Legacy DB
Implementation	Static class

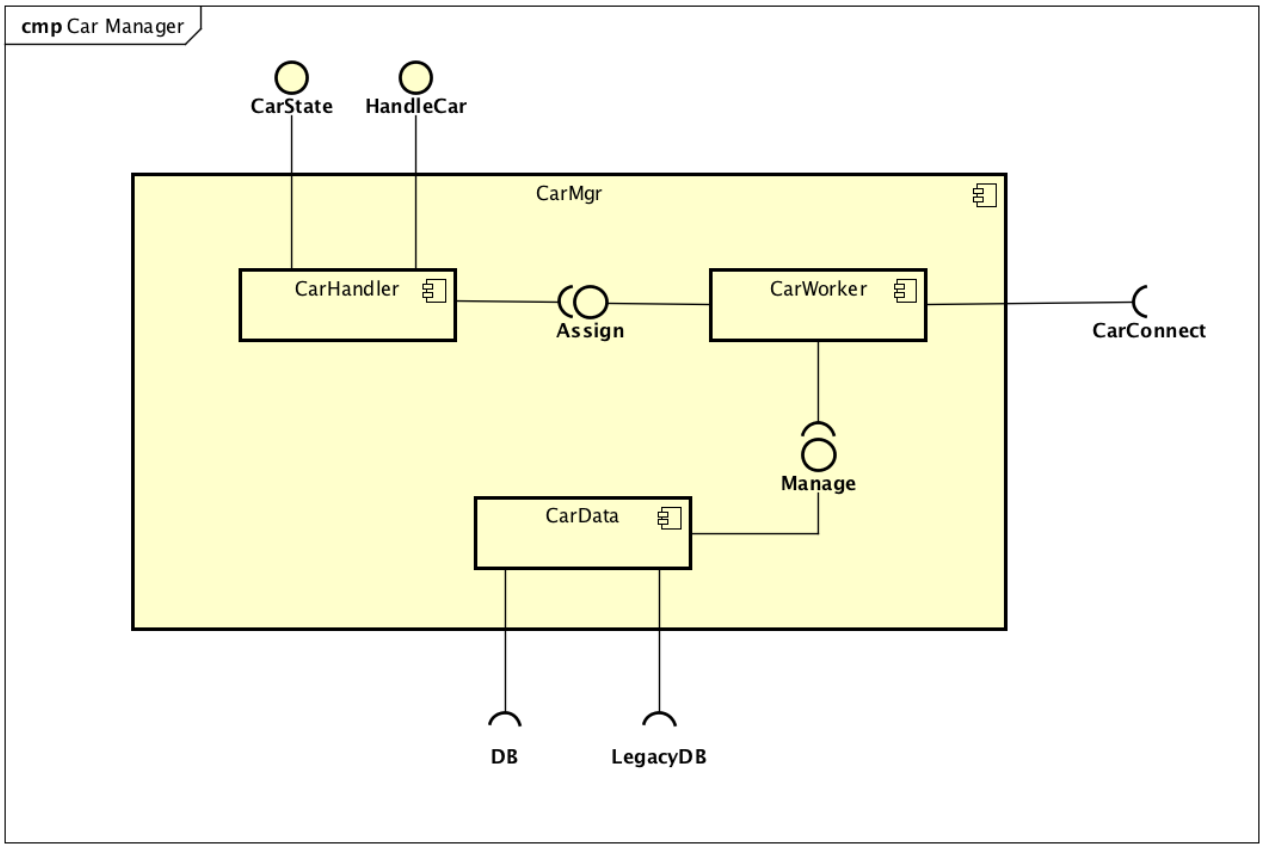
2.2.4. Ride Manager



RideHandler	
Definition	Component that receive the users' start/end ride command and manage all the ride.
Responsibilities	This component receives the "start" command from the registered user that already reserved a car and for each of them starts a worker that will handle them singularly. It offers an interface CurrentRideInfo for the car to provide it the current information about the requested ride like the partial amount of the ride or the safe areas nearest the user actual position.
Interaction	With the RideWorker, the user, the ReservationManager to set as expired the reservation when start the ride, the cars that retrieve the current ride information and display the data to the users.
Interfaces offered	• StartRide for the user • CurrentRideInfo for the Green e-Box car component
Interfaces required	• ReservationState of the ReservationManager • Assign of the CarWorkers
Implementation	Factory method pattern

RideWorker	
Definition	Component that manages individually the active ride of the registered user.
Responsibilities	There is an instance of this components for every active ride. It manages and collect all the necessary data about the ride for the whole duration of it. When it starts, it requests the car to the CarManager and tells it to unlock the car. The ride is active and now the component collects all the relevant data. When the ride end, it checks the position (look if the car is in a Safe Area), it checks the car values (door state, engine state and others), send the ride data to PaymentManager that process the payment, and last it stores all the ride data in the DB.
Interaction	The DB where it stores all the data, the RideHandler that manage every RideWorker, the AreaManager that provide the type of area by a GPS position, the CarManager that provides the HandleCar interface used to receive car information and to send the command of unlock the car.
Interfaces offered	• Assig for the RideHandler
Interfaces required	• DB to store data • HandleCar of the CarManager • AreaType of the AreaManager • Payment of the PaymentManager
Implementation	Multi instance: one for each ride

2.2.5. Car Manager

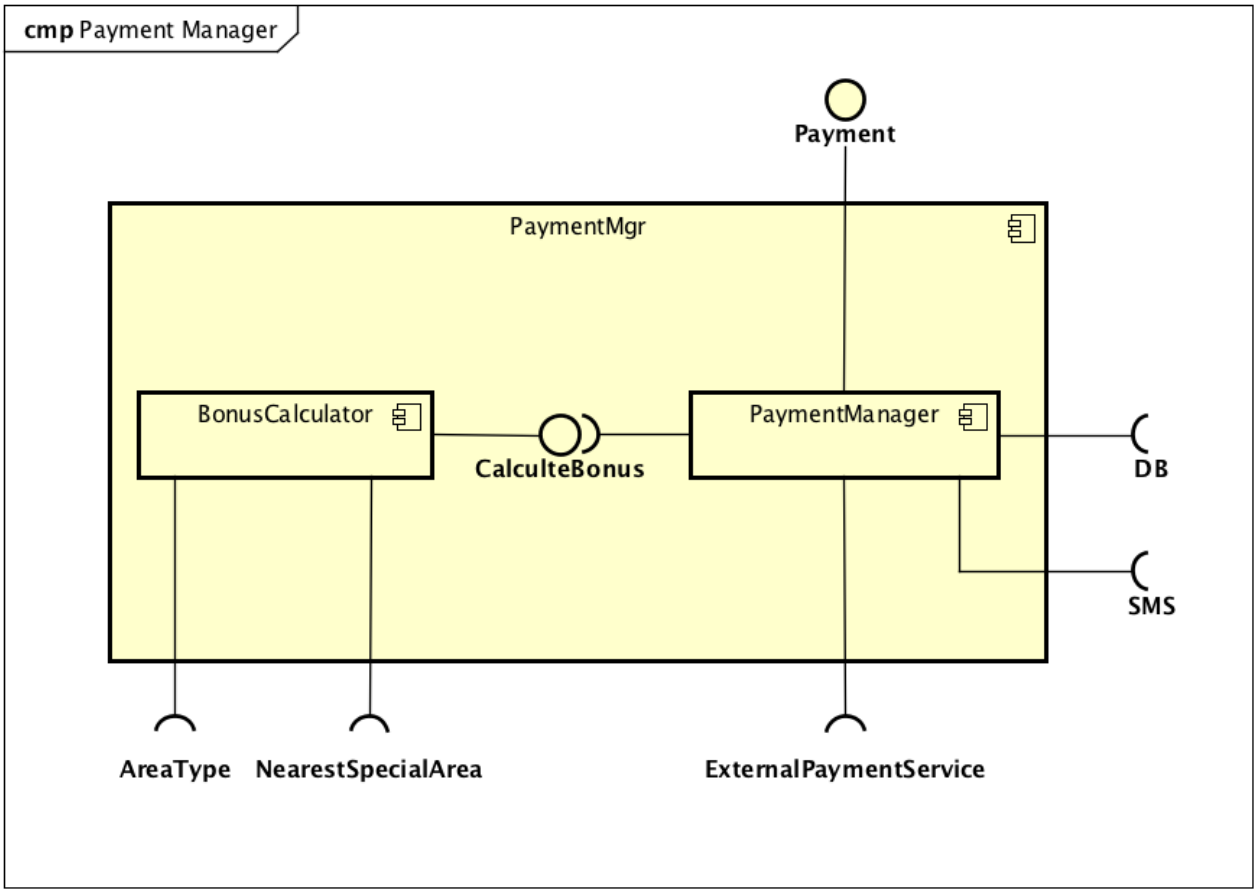


CarHandler	
Definition	Component that manages all the car.
Responsibilities	This component receives external request from other component to manage the car or just edit the state of a PowerEnjoy car. It starts all the workers that manages singularly the car and manage them. It also dispatches the external request in the right CarWorker.
Interaction	With the ReservationManager that can edit the state of a car. With the RideManager that need communicate with the car (read state values and manage the door state of the car). With all the instances of CarWorker that it manages.
Interfaces offered	• CarState for the ReservationManager • HandleCar for the RideManager
Interfaces required	• Assign of the CarWorker
Implementation	Factory method pattern and pool manager

CarWorker	
Definition	Component that manages individually a PowerEnjoy car.
Responsibilities	This component is in charge of allows the communication between a physical PowerEnjoy car and the central control system through the CarControl interface. It stores and retrieves car data thanks to CarData component. It offers the Assign interface that allows the CarHandler to handle it.
Interaction	With the CarHandler, the CarData, the Green e-box in the car.
Interfaces offered	• Assign for CarHandler
Interfaces required	• CarConnect of the Green e-Box car component • Manage of the CarData
Implementation	Multi instance: one for each PowerEnjoy car

CarData	
Definition	Component that manages the car data and retrieves/stores the data in the right position.
Responsibilities	This component is in charge of manage the car data and retrieve/store them in the right position, since the compatibly with the LegacyDB is required and the new additional data are stored in the new DB. It offers an interface used by all the CarWorker that use the car data in a transparent way.
Interaction	With DB, LegacyDB and the CarWorker.
Interfaces offered	• Manage for the CarWorker
Interfaces required	• LegacyDB • DB
Implementation	Static class

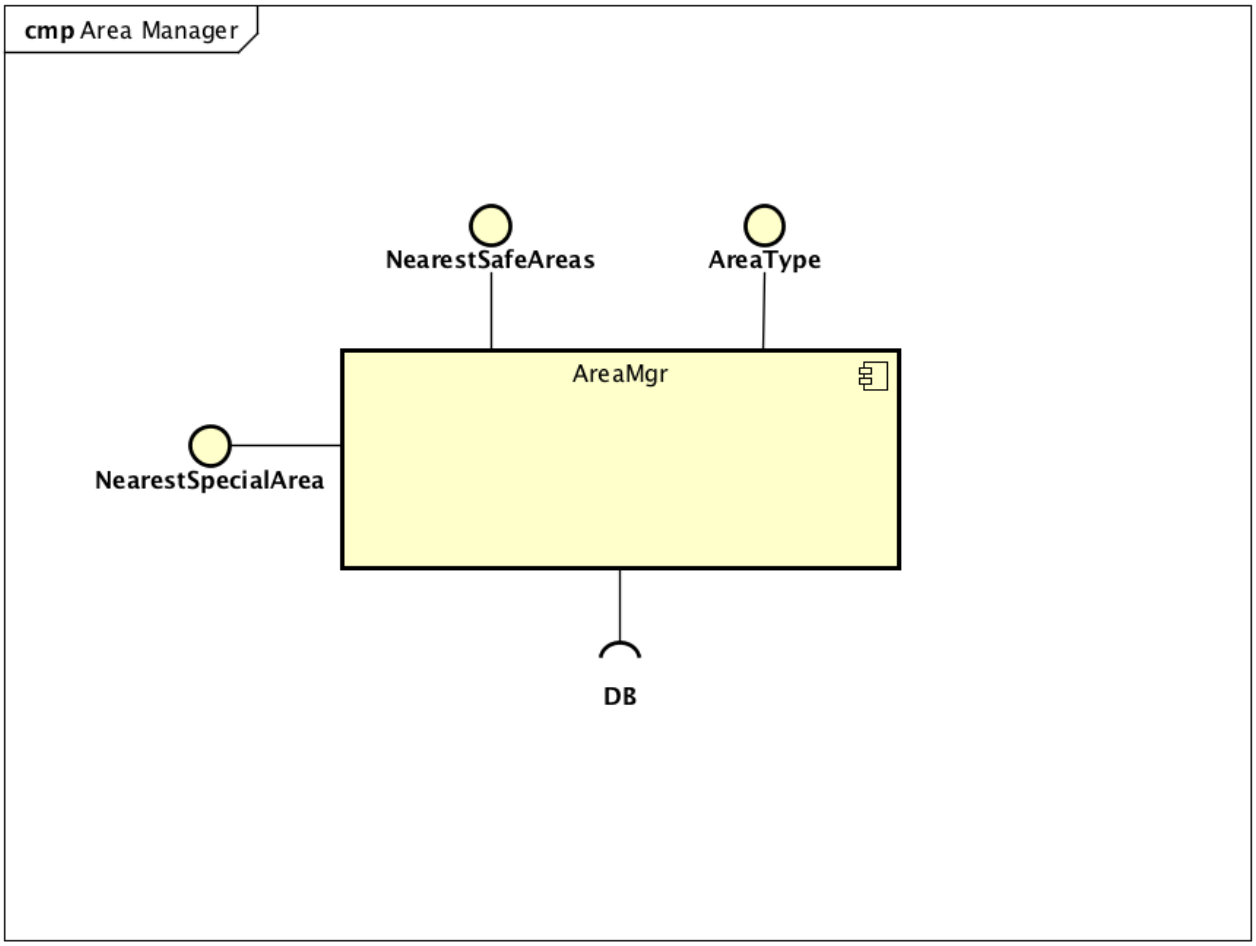
2.2.6. Payment Manager



PaymentManager	
Definition	Component that controls the payment process.
Responsibilities	This component is in charge of computing the total cost of the ride. It interacts with the RideManager to receive information about the ride and with the BonusCalculator for calculate the right percentage of bonus. It uses an interface to process the payment with external services. At the end, it sends a SMS to the user with the payment information.
Interaction	With RideManager that provides all the information about the ride, the BonusCalculator that apply the bonus, the DB to store data and read the user credit card, the ExternalPaymentService that process the payment, the SMS service to send a message with information about the payment.
Interfaces offered	• Payment for the RideManager
Interfaces required	• CalculateBonus of the BonusManager • DB • ExternaPaymentService • SMS
Implementation	Static class

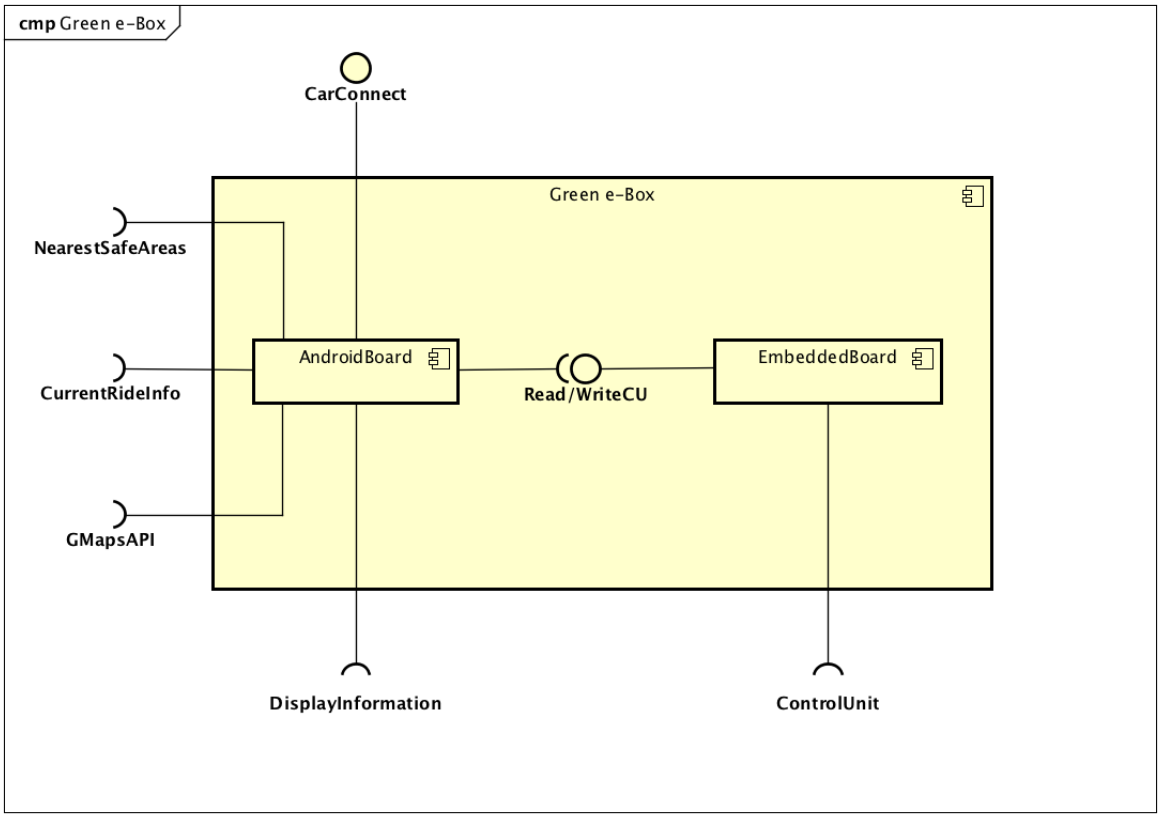
BonusCalculator	
Definition	Component that calculate the bonus percentage that are to be applied on the ride amount.
Responsibilities	This component is in charge of calculate the right percentage of the “behaviour bonus” (for more information about this look the RASD chapter 3.2.10 [G10])
Interaction	With the payment manager, it provides all the information about the ride, this information is used to check the possibility of apply every single bonus.
Interfaces offered	• CalculateBonus for the PaymentManager
Interfaces required	• AreaType of the AreaManager • NearestSpecialArea of the AreaManager
Implementation	Static class

2.2.7. Area Manager



AreaManager	
Definition	Component that provide information about the areas.
Responsibilities	Component that receive GPS coordinate (lat,lng) and send back the type of the area that includes the coordinate through the AreaType interfaces. It also provides (through the NearestSafeArea interfaces that receives a GPS coordinate) a list with all the position of nearest safe areas and for each of them specify if it is a simple safe area or a special area.
Interaction	With the DB where are stored the Safe Area position and Special Area positios and with the Ride Manager that needs the information about the areas type.
Interfaces offered	• AreaType for the RideManager and the PaymentManager • NearestSafeAreas for the Green e-Box car component • NearestSafeSpecialArea for the PaymentManager
Interfaces required	• DB
Implementation	Static class

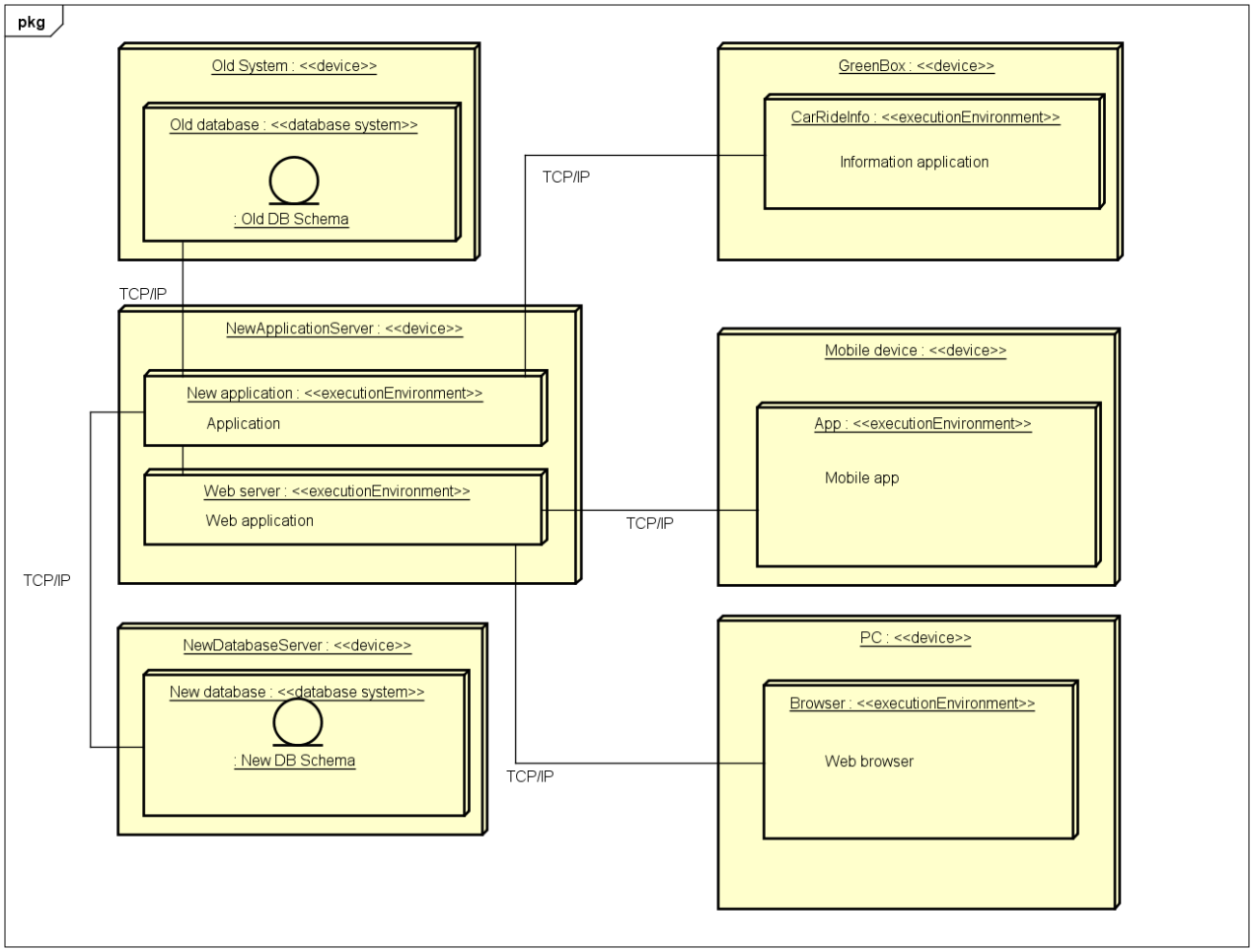
2.2.8. Green e-Box



AndoridBoard	
Definition	Component that controls the physical car and it allows the remote communication between the car and the central system.
Responsibilities	This component reads and writes data from the CU (doors state, number of passengers, engine state and other), it displays information on the car screen (current charge and nearest safe areas, these last showed in a map thanks to GMapsAPI).
Interaction	With the EmbeddeBoard that allows the Green e-Box to communicate with the CU at high level. With the CarWorker that manages and controls the car remotely through the CarConnect interfaces. With the user that sees the information on the car display (this information are provided by the RideManager, AreaManage and the GmapsAPI).
Interfaces offered	• CarConnect for the CarWorker
Interfaces required	• Read CU of the EmbeddedBoard • Write CU of the EmbeddeBoard • Display Information of the car display • CurrentRideInfo of the RideManager • NearestSafeAreas of the AreaManager • GmapsAPI
Implementation	One physical AndoridBoard for each PowerEnjoy car

EmbeddedBoard	
Definition	Component that communicate with the CU of the cars at low level.
Responsibilities	This component receive request from the Android board and communicate at low level with the CU to retrieved the state values (car engine state, door state, number of passengers and others). It can also actuate the high-level command of lock/unlock the doors of the car.
Interaction	With the AndroidBoard and with CU of the car.
Interfaces offered	• Read CU for the AndoridBoard • Write CU for the AndroidBoard
Interfaces required	• ControlUnit to physical communicate with the CU of the car
Implementation	One physical EmbeddedBoard for each PowerEnjoy car

2.3. Deployment View



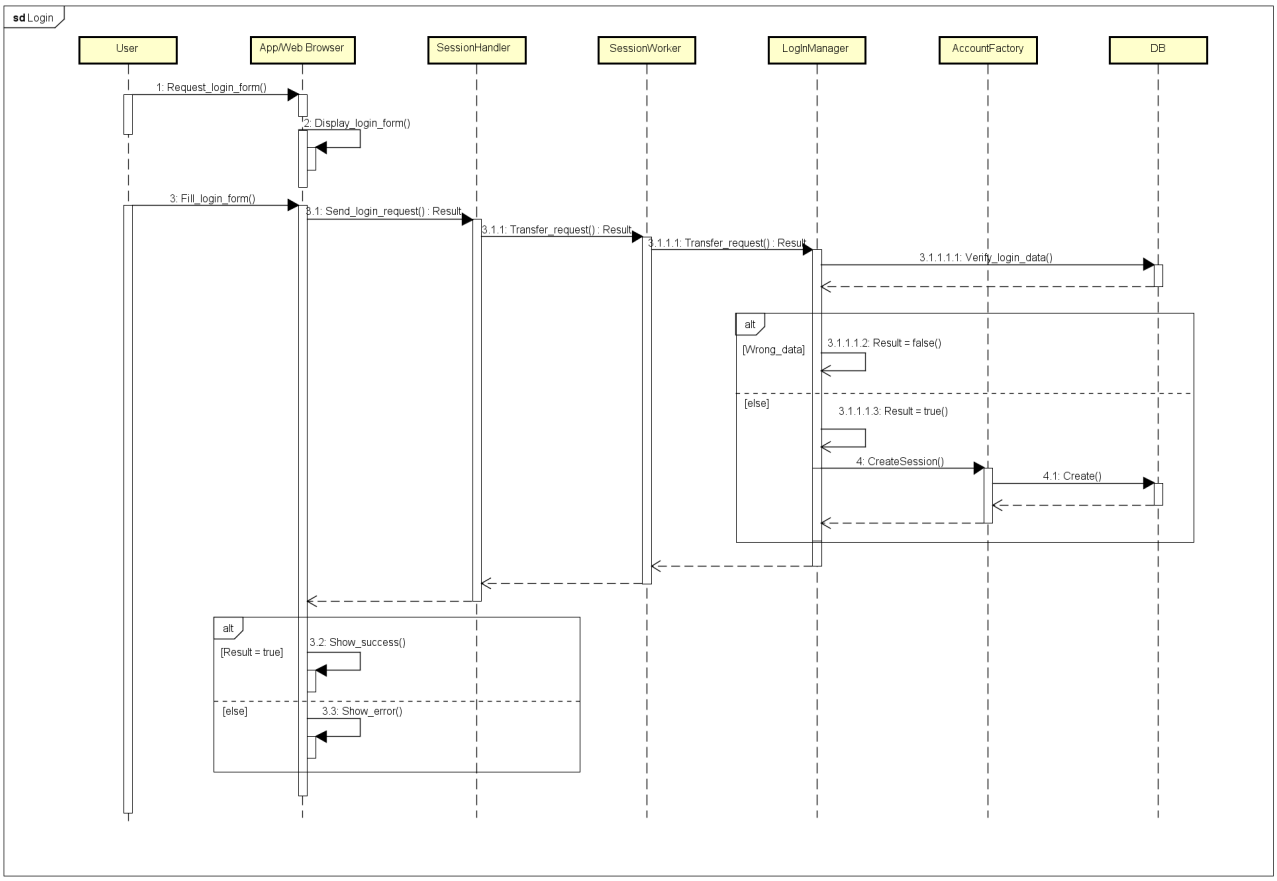
The deployment diagram above explains which component executes which software.
The NewApplicationServer is dedicated to execute the business logic (New Application) and the web server (Web server) and this is the central node of the entire system.
The databases are stored in the NewDatabaseServer and in the OldSystem.
The mobile app is executed by a Mobile device, instead the Web browser is executed by a PC.
The mobile app and web server are the two ay that the user should use the service.
The NewApplication can communicate with the CarRideInfo installed on the GreenBox. In this way, the system can “control” all the cars.
The protocol used for the communication between software components is the TCP/IP.

2.4. Runtime View

In this section, there are some sequence diagrams that represent the main features of the PowerEnjoy system.
The components represented in the following diagrams are referred to the components represented in the Component view.

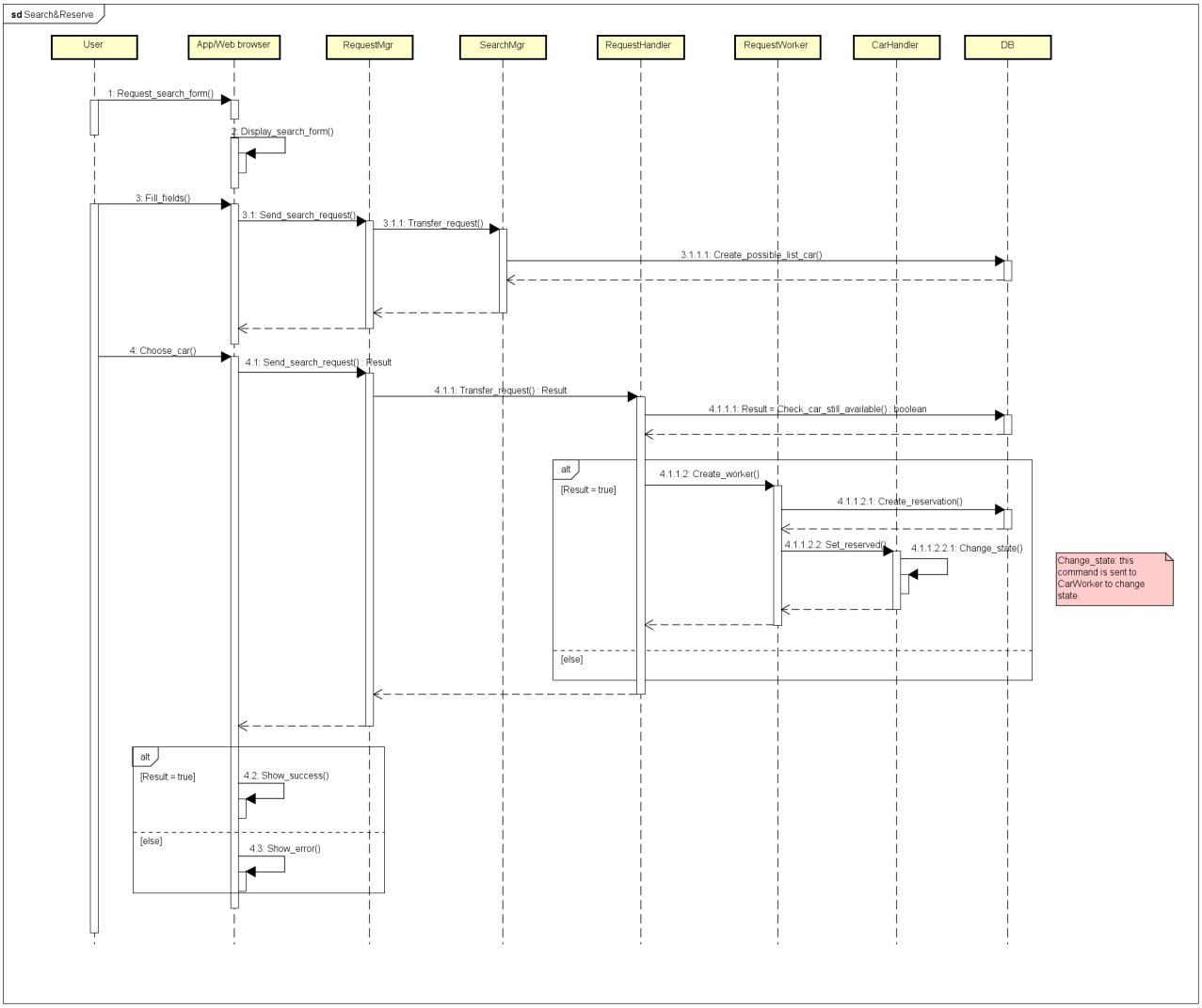
2.4.1. Login

This feature allows user to log to the service. The user can log to the service only if he is already registered to PowerEnjoy. He can use the app or the web service. The incoming request in transferred from the SessionHandler to the SessionWorker that chooses the right way for the log request. From the SessionWorker the request is sent to the LoginManager that checks if the credentials are correct and then, if credentials are correct, the Accountfactory creates a session for the user and saves it to the DB.



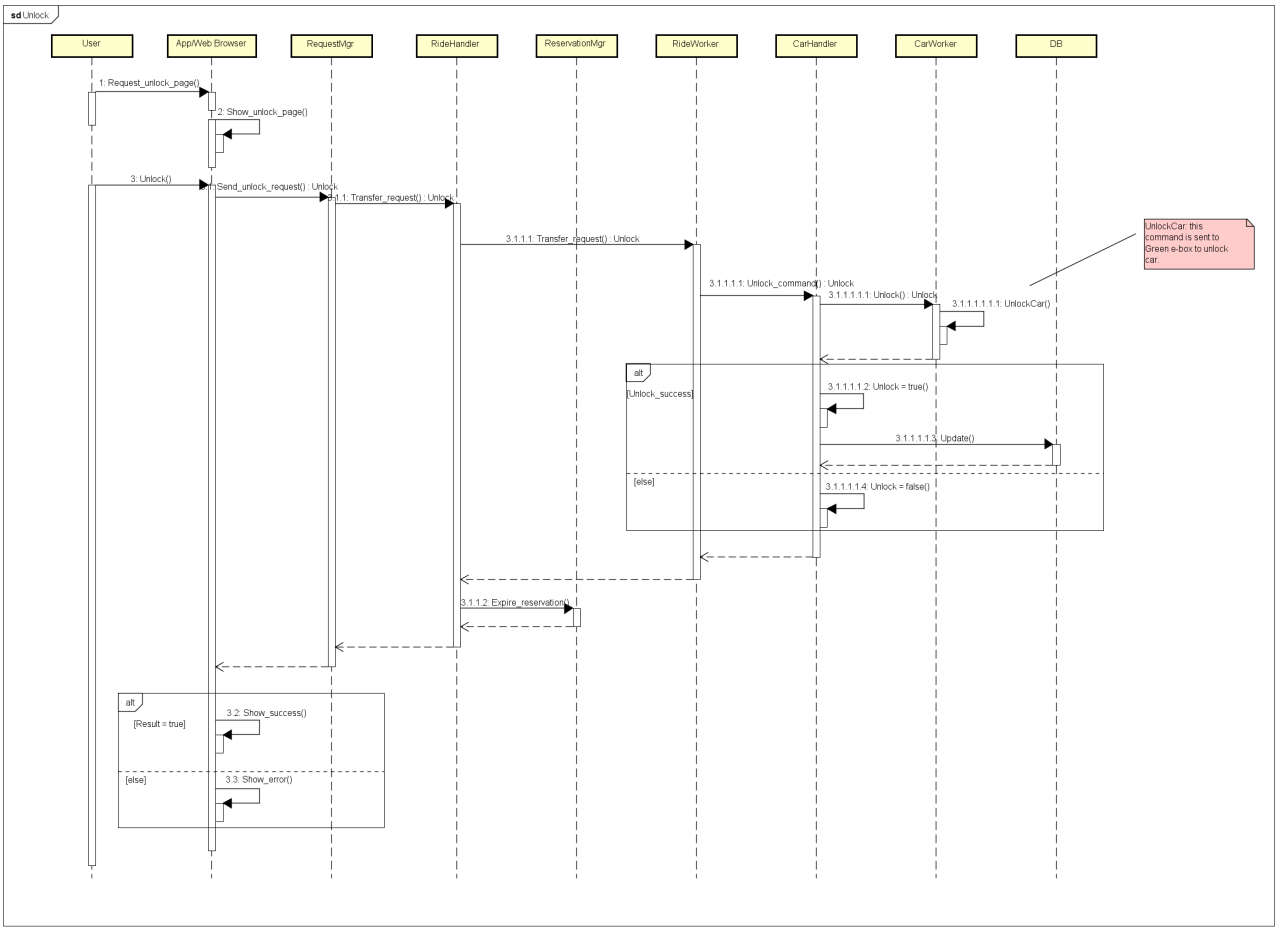
2.4.2. Search & Reserve

This functionality allows user to search and reserve a car. He can do this using the app or the web application. The request arrives to the RequestMgr that dispatches it to the right component. The SearchMgr questions the DB and sends back the list of cars. When user sends the reserve request, it is transferred by the RequestMgr to the RequestHandler. The RequestHandler creates the RequestWorker and the latter creates the reservation and sends the Set_reserved command to the CarHandler.



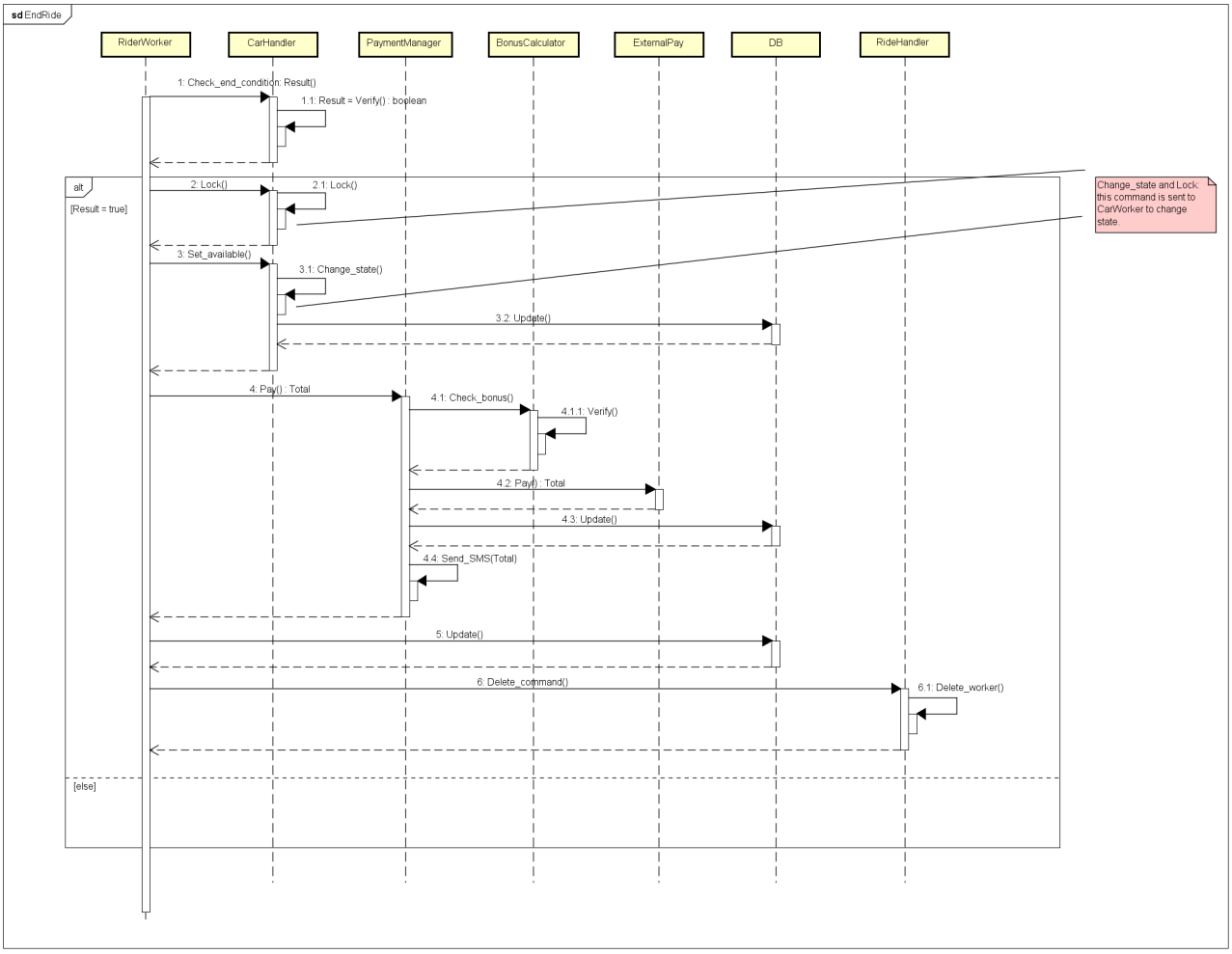
2.4.3. Unlock

This feature allows user to send the unlock command. When request is arrived from the app or the web service, the RequestMgr select the component to manage it. RideHandler transfers the request to RideWorker that sends unlock command to CarHandler. CarHandler transfers command to CarWorker and is executes the command. CarHandler update the DB and sends back the result. Now RideHendler can expires the reservation.



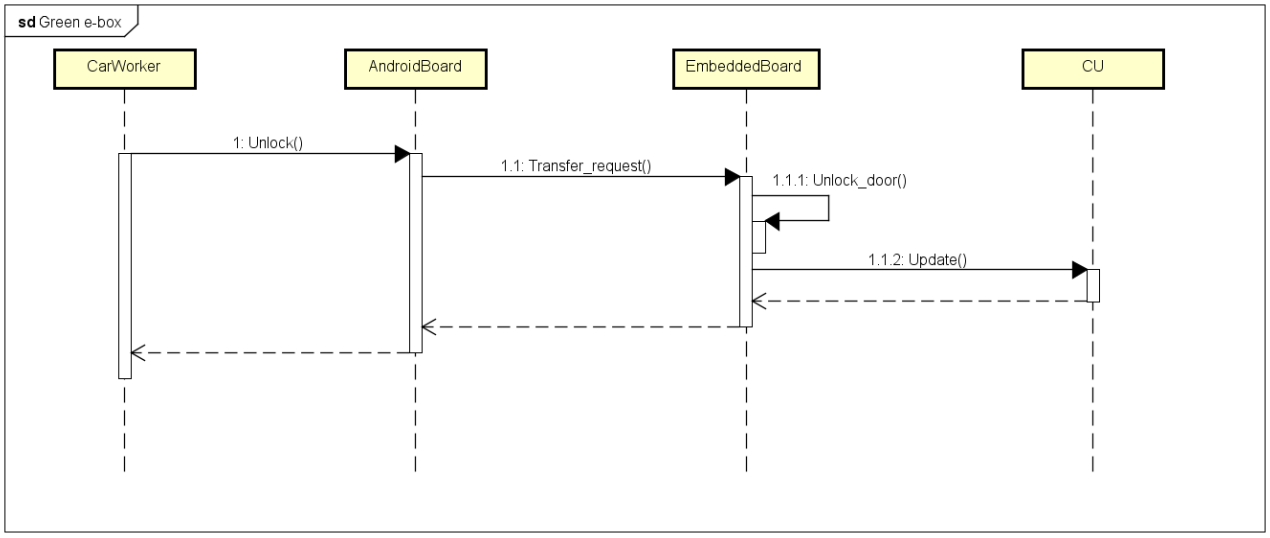
2.4.4. EndRide

This functionality allows system to stop charging, lock the doors and account ride to user. When the end conditions are verified, the RideWorker tell CarHandler to lock the doors and to change state. The it sends to PaymentManager pay command and the latter transfer the command to the external payment service. The PaymentManager and RideWorker update the DB.



2.4.5. Green e-box: CU Interaction

The Green e-Box manages all the CU request in the following way.
The following sequence diagram shows one of this interaction, the unlocked of the car.
The request arrives from the CarWorker and it is sent to the AndroidBoard. Here the request is transferred to the EmbededBoard that unlocks the doors and update the CU.



2.5. Architectural Styles and Patterns

As it was written before, PowerEnjoy service is based on the client-server architecture. The server provides services and manages requests arrived from the clients. It also interacts with the databases and all the cars.
Instead the clients visualize pages and sends requests to the server through them. The server must guarantee a high level of parallelism because it must be able to accept and process multiple client's requests at the same time.
The clients are thin-client, it means that all data and all business logic are located into the server.

Talking about pattern, the principals are:

- Model View Controller (MVC): this pattern allows to separate the software application into three main parts: the model, the controller and the view. The model represents the data of the application; the controller has all the business logic and the view visualizes the model content and deals to interact with the user.
- Factory: this pattern defines an interface to create an object but leaves to sub-classes the task to create the object. In PowerEnjoy this pattern is used by handler components to create the worker components.
- Singleton: this pattern allows to instantiate only one time a determinate object. It is used when an object must be the only. The handler components use this pattern.
- Façade: this pattern allows to provide a simplify interface to a set of subsystems. In PowerEnjoy service this pattern is used for example in the RequestManager that provides only one interface to the user.

Multithreading is highly used in this project because the application must be able to manage a high number of tasks at the same time. Every "worker" is an independent thread.
When a request arrived to a handler, the latter creates and runs a worker. When the worker finishes its task, the handler deletes it.

2.6. Other Design Decisions

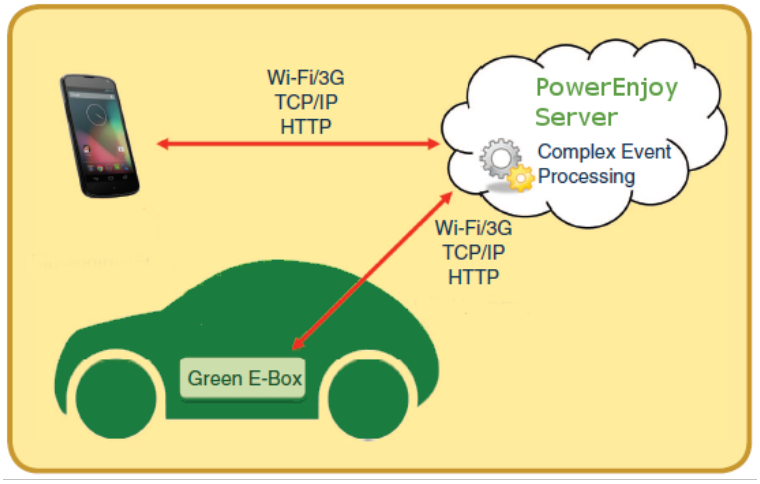
2.6.1. Set of Safe Areas

The set of safe areas is pre-defined and it is stored in the database, in this way it can be retrieved easily from others components (if is it necessary).

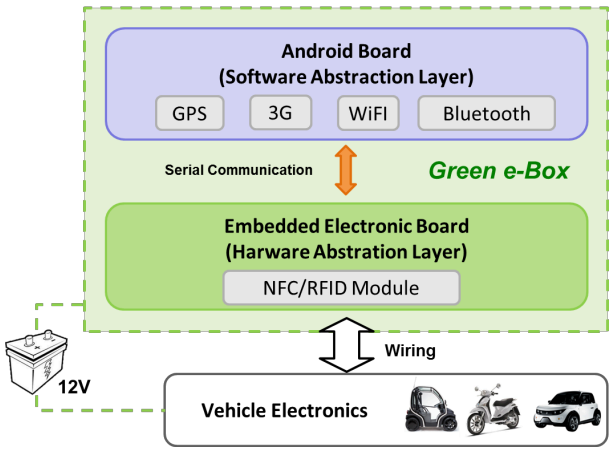
2.6.2. Green e-Box

A Green e-Box is a device installed in every car and this component allows to the central system to communicate with the car. It also retrieves information about the CU.
The Green e-Box is provided by GPS, 3G and Wi-Fi connection to communicate with the central system. It uses Android as OS.
The Green e-Box is also connected to the screen installed into the car. This screen is used by the user to see the amount of the ride and the positions of the safe areas.

Interaction between the car, the Power Enjoy central system and the user's smartphone:



Green e-box architecture and interaction with car CU:



2.6.3. APIs

PowerEnjoy provide APIs to external services to allow interaction with the system.
The access to these functionalities is provided using the HTTPS protocol.
To exchange data is used JSON.

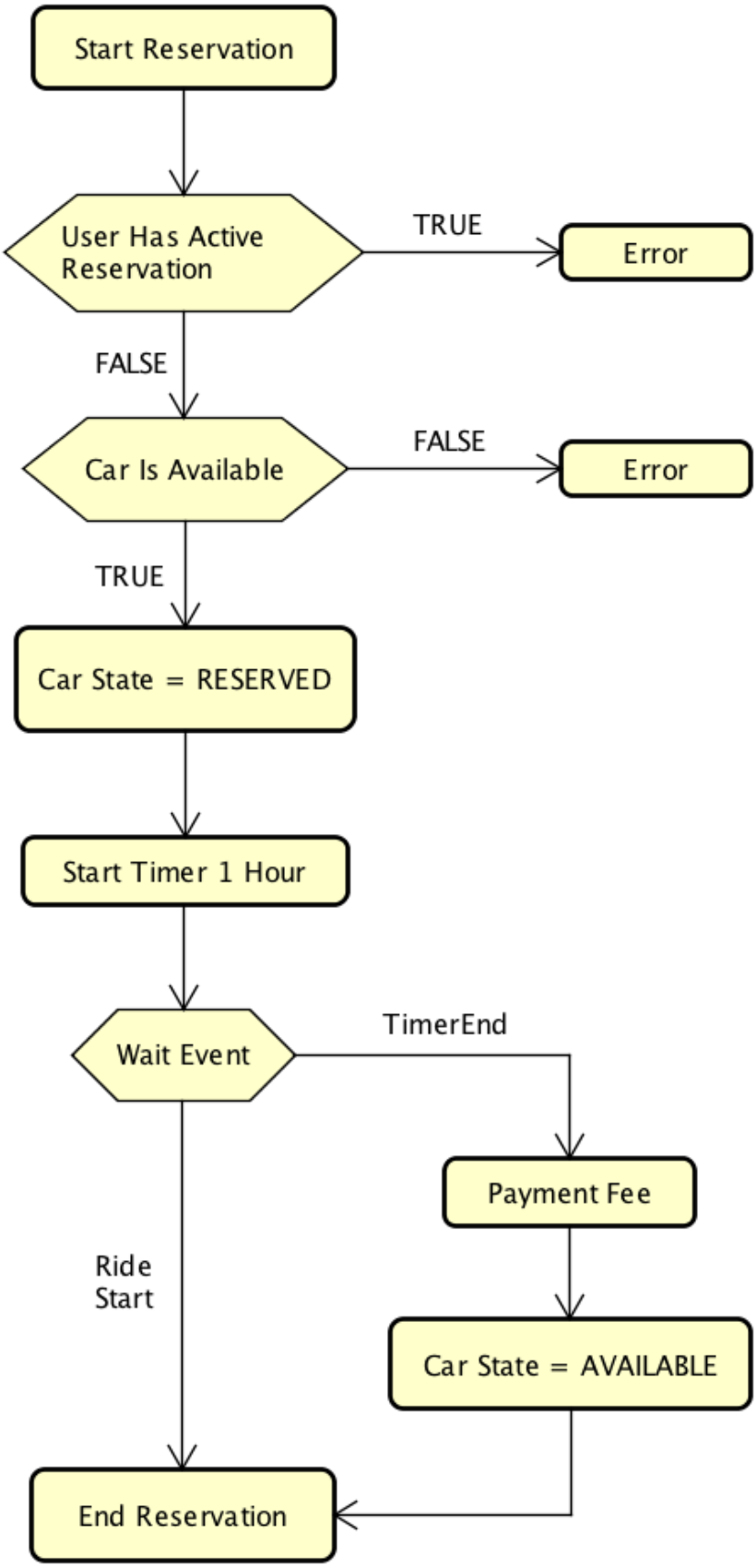
3. Algorithm Design

Here we give just an idea of the most interesting parts, this isn't the complete code.

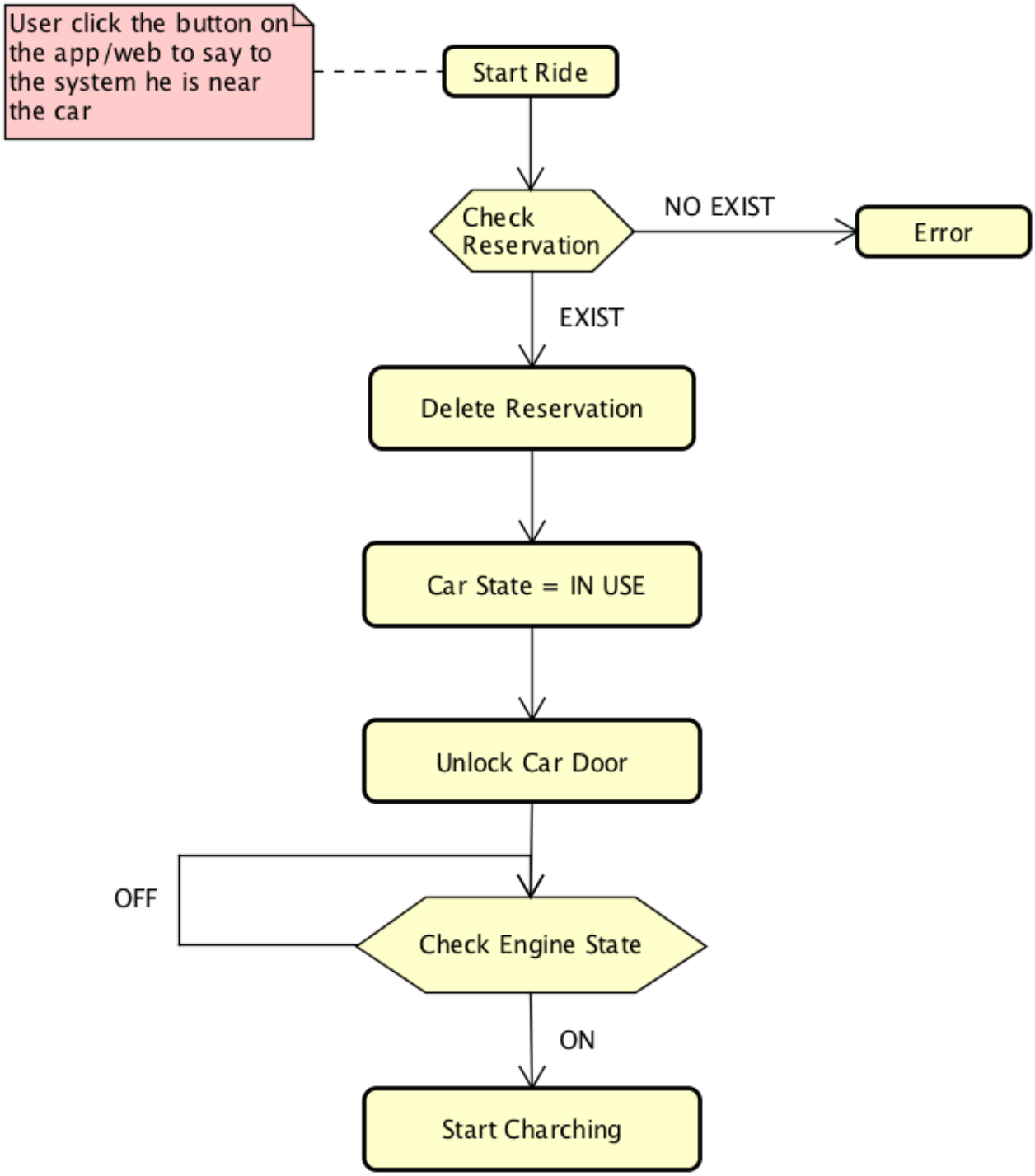
3.1. Algorithm: Bonus Percentage

```
private static final double BONUS_PASSENGERS = 0.1;
private static final double BONUS_BATTERY = 0.2;
private static final double BONUS_CAR_IN_CHARGE = 0.3;
private static final double PENALTY_DISTANCE_LOWBATTERY = -0.3;

private static final double MAX_DISTANCE_SPECIAL_AREA = 3.0;
private static final double MIN_BATTERY_LVL_PENALTY = 0.2;
private static final int NUMBER_PASSENGERS_BONUS = 2;
private static final double MIN_BATTERY_LVL_BONUS = 0.5;

public static int calculatePercentageBonus(Ride ride){
        double bonus;
        bonus = 0.0;
        if(ride.getNumeberOfPassengers >= NUMBER_PASSENGERS_BONUS)
                bonus += BONUS_PASSENGERS;
        if(ride.getFinalBatteryLvl >= MIN_BATTERY_LVL_BONUS)
                bonus += BONUS_BATTERY;
        if(AreaManager.areaType(ride.getEndPosition) == AreaManager.SPECIAL_AREA &&
        ride.isCarInCharge == true)
                bonus += BONUS_CAR_IN_CHARGE;
        if(AreaManager.distanceNearestSpecialArea(ride.getEndPosition) >=
        MAX_DISTANCE_SPECIAL_AREA || ride.getFinalBatteryLvl < MIN_BATTERY_LVL_PENALTY)
                bonus += PENALTY_DISTANCE_LOWBATTERY;
        if(bonus >= 1.0)
                bonus =1.0;
        return (int) (bonus*100);
}
```

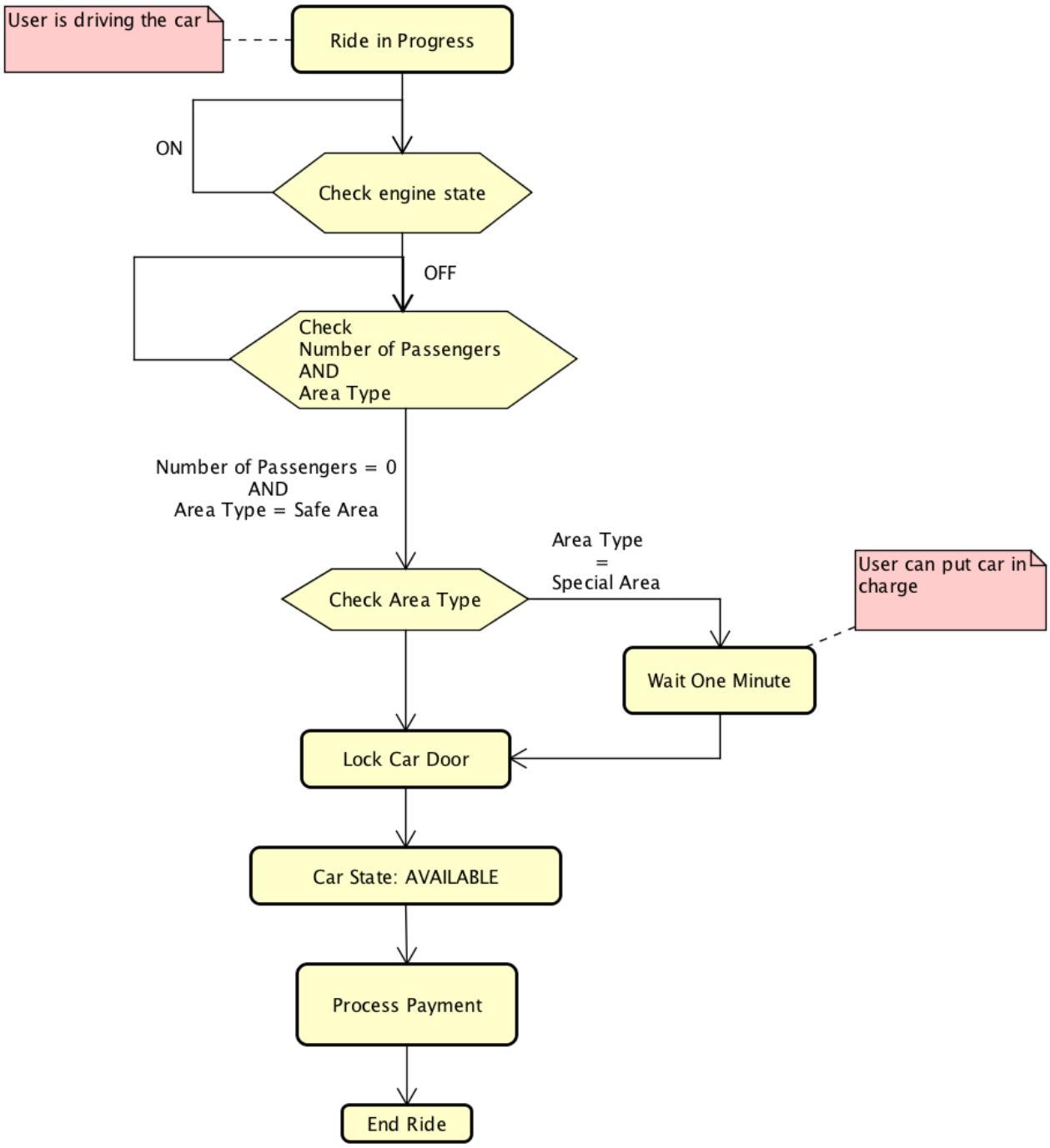
3.2. Flow Chart: Car Reservation



3.3. Flow Chart: Start Ride



3.4. Flow Chart: End Ride



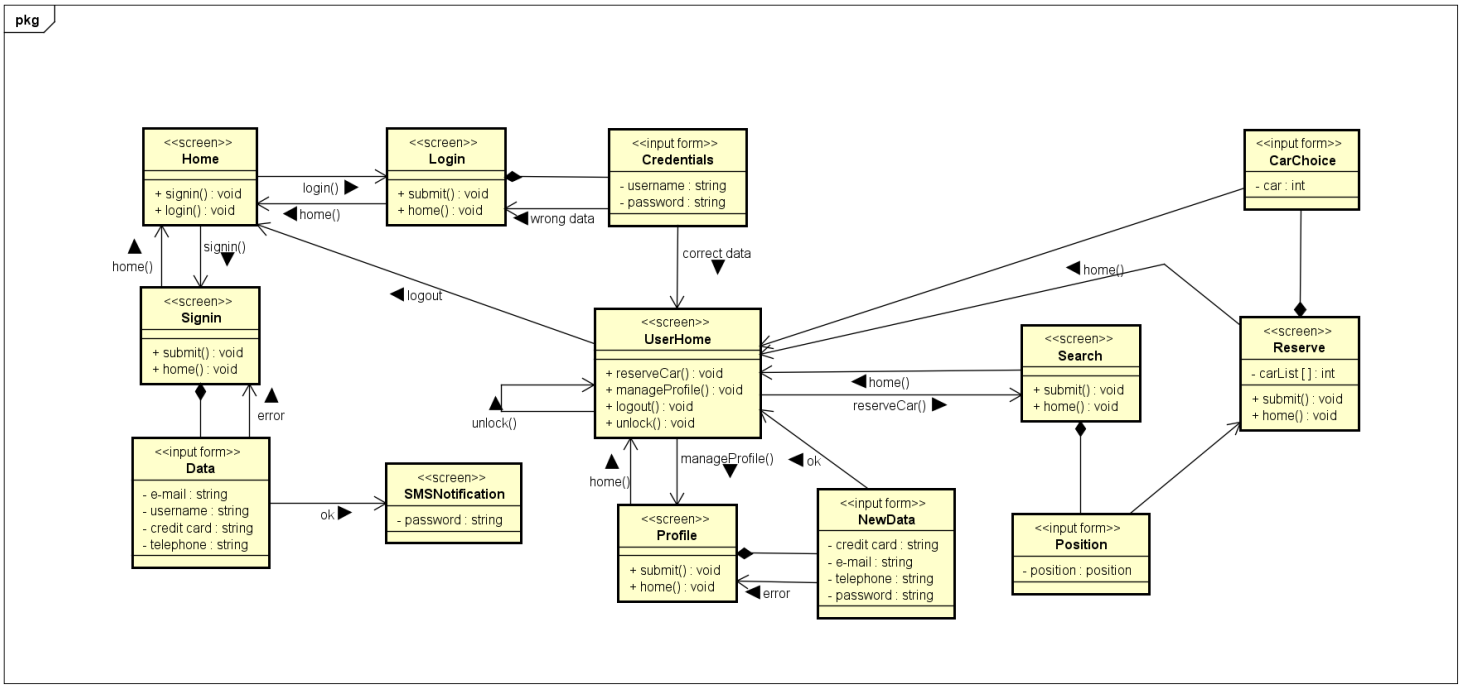
4. User Interface

4.1. Mockups

In the RASD there are the mockups that provide an overview of user interfaces (RASD section 3.1.1).

4.2. UX Diagram

The follow picture represents an UX diagram and show how the user can interact with the functionalities.



5. Requirements Traceability

- **[G1] The client shall be able to access the service through web service.**
  ⇒ [R1] The system must provide two different ways to access to the service with same functionalities: web page and mobile app.
  ✓ The View component offers the two type of interfaces (App and Web Page Service), these have the same functionalities and can send the same requests to the system.

- **[G2] The client shall be able to sign in and log in to the service.**

  ⇒ [R1] The client must be able to submit a valid email address, phone number and choose an available username, this fields are mandatory for successfully complete the registration.
  ✓ The AccountManager component satisfies this requirement.

  ⇒ [R2] The system must be able to send back to the new user his password.
  ✓ The AccountManager component satisfies this requirement.

  ⇒ [R3] The user can be able to log in into the service submitting his username and his password.
  ✓ The AccountManager component satisfies this requirement.

  ⇒ [R4] The user must be able to submitted his credit card number in according with the privacy and security policies.
  ✓ The AccountManager component satisfies this requirement.

  ⇒ [R5] The unregistered user can be see only sign in and log in pages.
  ✓ The RequestManager component satisfies this requirement.

- **[G3] The user shall be able to manage his profile.**
  ⇒ [R1] The logged user must be able to modify his profile.
  ✓ The AccountManager component satisfies this requirement.

  ⇒ [R2] The logged user must be able to edit his credit card number.
  ✓ The AccountManager component satisfies this requirement.

- **[G4] The user shall be able to search cars in a specific zone.**
  ⇒ [R1] The user must be able to search cars near his current position (using GPS coordinate).
  ✓ The ReservationManager component satisfies this requirement.

  ⇒ [R2] The user must be able to search cars near a specific address.
  ✓ The ReservationManager component satisfies this requirement.

  ⇒ [R3] The system must be able to verify that the submitted position is in the city.
  ✓ The ReservationManager component satisfies this requirement.

  ⇒ [R4] The system must be able to refuse request of searching car in position out of the city.
  ✓ The ReservationManager component satisfies this requirement.

  ⇒ [R5] The system must be able to provide a list of available cars in according with the position indicated by the user.
  ✓ The ReservationManager component satisfies this requirement.

- **[G5] The user shall be able to reserve a car from a list up to one hour.**
  ⇒ [R1] The user must be able to see the list of available cars received as response to his previous search request.
  ✓ The ReservationManager component satisfies this requirement.

  ⇒ [R2] The user must be able to select and reserve a car from the list for up to one hour.
  ✓ The ReservationManager component satisfies this requirement.

  ⇒ [R3] The user can reserve only one car by one.
  ✓ The ReservationManager component satisfies this requirement.

  ⇒ [R4] If after one hour of car reservation the user dot pick up the car, reservation expired and the user pays a fee of 1 EUR.
  ✓ The ReservationManager and the PaymentManager components satisfy this requirement.

- **[G6] The user shall be able to picks up and drives the reserved car.**
  ⇒ [R1] The user who reserved a car must be able to tell the system he is near the car.
  ✓ The View component satisfies this requirement.

  ⇒ [R2] The system must be able to know if car's engine is power on.
  ✓ The CarManager and Green e-Box components satisfy this requirement.

  ⇒ [R3] The system must be able to notifies user of the current charges.
  ✓ The RideManager and Green e-Box components and the satisfy this requirement.

  ⇒ [R4] The system must be able to unlock the doors of the car.
  ✓ The CarManager and Green e-Box components satisfy this requirement.

- **[G7] The user shall be able to know where are the safe area for parking the car.**
  ⇒ [R1] The system allows to user to see the position of safe areas.
  ✓ The AreaManager and Green e-Box components satisfy this requirement.

  ⇒ [R2] The system allows to user to see the position of special areas.
  ✓ The AreaManager and Green e-Box components satisfy this requirement.

- **[G8] The user shall be able to know the current charges during the ride.**
  ⇒ [R1] The system allows to user to see the current charges by a screen installed on the car.
  ✓ The Green e-Box component satisfies this requirement.

  ⇒ [R2] The system must be able to send to the car the current charges.
  ✓ The RideManager component satisfies this requirement.

- **[G9] The user shall be able to end the ride and pay it when he leaves the car in safe area.**
  ⇒ [R1] The system must be able to know if the car's engine is power off and the car is parked in a safe area.
  ✓ The CarManager, the Green e-Box, the RideManager and the AreaManager components satisfy this requirement.

  ⇒ [R2] The system must be able to stop charging user if the car is parked in a safe area.
  ✓ The RideManager component satisfies this requirement.

  ⇒ [R3] The system must be able to notify the user the account of the ride.
  ✓ The PaymentManager component satisfies this requirement.

  ⇒ [R4] The user must be able to see the amount of the ride.
  ✓ The PaymentManager notify the amount through SMS service and user sees the account on its terminal.

  ⇒ [R5] The system must charge the total amount of the ride on the credit card user.
  ✓ The PaymentManager through external payment service.

- **[G10] The system must incentivize the virtuous behaviours of the users.**
  ⇒ [R1] The system must be able to know the number of passenger in the car.
  ✓ The CarManager and the Green e-Box components satisfy this requirement.

  ⇒ [R2] The system must be able to know the battery level of the car.
  ✓ The CarManager and the Green e-Box components satisfy this requirement.

  ⇒ [R3] The system must be able to know if the car is in a special area.
  ✓ The CarManager, the Green e-Box, the AreaManager components satisfy this requirement.

  ⇒ [R4] The system must be able to detect the car's position.
  ✓ The CarManager and the Green e-Box components satisfy this requirement.

  ⇒ [R5] The system must be able to apply a discount of 10% on the last ride if it detects that there are at least two other passengers onto the car.
  ✓ The RideManager, the CarManager, the Green e-Box and the PaymentManager components satisfy this requirement.

  ⇒ [R6] The system must be able to apply a discount of 20% on the last ride if it detects that the car is left with no more than 50% of the battery empty.
  ✓ The RideManager, the CarManager, the Green e-Box and the PaymentManager components satisfy this requirement.

  ⇒ [R7] The system must be able to apply a discount of 30% on the last ride if it detects that the car is parked in a special area and it is plugged into a grid power.
  ✓ The RideManager, the PaymentManager, the AreaManager, the CarManager, the Green e-Box components satisfy this requirement.

  ⇒ [R8] The system must be able to apply a charge of 30% more on the last ride if it detects that the car is left at more than 3Km from the nearest power grid station or with no more than 80% of the battery empty.
  ✓ The RideManager, the PaymentManager, the AreaManager, the CarManager, the Green e-Box components satisfy this requirement.

  ⇒ [R9] Bonus for one ride are cumulative.
  ✓ The PaymentManager component satisfies this requirement.

  ⇒ [R10] The discount percentage can't exceed 100% of the total amount.
  ✓ The PaymentManager component satisfies this requirement.

- **[G11] The system should offer public APIs to enable the possibility to develop additional services on top of the basic ones.**
  ⇒ [R1] The system offers APIs to third party applications using web APIs as technology.
  ✓ The LoginManager and the ReservationManager components satisfy this requirement.

  ⇒ [R2] The system replies using current industry standards.
  ✓ The LoginManager and the ReservationManager components satisfy this requirement.

6. References

- Material from Wikipedia
  - Thin client: https://en.wikipedia.org/wiki/Thin client
  - Event-driven architecture: https://en.wikipedia.org/wiki/Event-driven architecture
  - JEE platform: https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition
- Documents
  - UML e Design Pattern – M. Bigatti
  - Paper on the green move project.pdf
  - Second paper on the green move project.pdf

7. Appendix

7.1. Software and Tools Used

- Microsoft Office Word: to redact and format this document.
- Astah Professional 7.0 (http://astah.net/editions/professional): to create Use Cases Diagrams, Sequence Diagrams, Class Diagrams and State Machine Diagrams.

7.2. Hours of Work

- Simone Boglio: 32 hours.
- Lorenzo Croce: 27 hours.