

A.Y. 2022-2023 Software Engineering 2 - DD Project

Document by Felipe Bagni, Felipe Azank and Sabrina Wolff

January 4, 2023

Table of Contents

1 Introduction	1
1.1 Purpose	1
1.2 Scope	1
1.3 Definitions, Acronyms, Abbreviations	2
1.3.1 Definitions	2
1.3.2 Acronyms and Abbreviations	2
1.4 Reference Documents	3
1.5 Document Structure	3
2 Architectural design	4
2.1 Overview: High-level components and their interaction	4
2.2 Component view	5
2.2.1 CPMS	6
2.2.2 eMSP	8
2.3 Deployment view	10
2.3.1 eMSP	10
2.3.2 CPMS	10
2.4 Runtime view	11
2.5 Component interfaces	24
2.6 eMSP	24
2.7 CPMS	25
2.8 Selected architectural styles and patterns	25
2.9 Other design decisions	26
3 User interface design	26
3.1 eMSP	26
3.1.1 Login	26
3.1.2 Register	27
3.1.3 SEPA Register	27
3.1.4 Booking Charging Station	28
3.1.5 Schedule Charging	28
3.1.6 Booking Confirmation	29
3.1.7 View Active Bookings	30
3.1.8 Charging Start	30

3.1.9 Charging Status	31
3.1.10 Notifications	31
3.2 CPMS	32
3.2.1 Login	32
3.2.2 Register	33
3.2.3 Home Page (Automatic Mode)	33
3.2.4 Charging Station View	34
3.2.5 Specific Charging Station View	35
3.2.6 Setting Prices	36
3.2.7 Setting Special Offer	37
3.2.8 Buying Energy from DSOs	38
3.2.9 Battery Management View	39
4 Requirements traceability	39
4.1 Review of requirements	39
4.2 List of abbreviations	42
4.3 Mapping traceability	43
5 Implementation, integration and test plan	45
5.1 Implementation	45
5.2 Integration and Testing	46
5.2.1 DBMS & Model	47
5.2.2 API Handler	47
5.2.3 Service	48
5.2.4 Router	48
5.2.5 Client Application	49
6 Effort spent	49
7 References	50

1 Introduction

1.1 Purpose

The global climate crisis displays one of the biggest threats to humanity and the planet. The loss of biodiversity, the impact on society and extreme weather conditions are only a few potential consequences if we don't act quickly and reduce the emissions of greenhouse gases in every sector possible. One important sector in this fight is the transportation sector, especially cars that emit 27% of total greenhouse gas emissions in the U.S, making it the largest contributor.¹ To deal with this problem more and more people should be encouraged to buy electric cars instead of traditional ones. But one problem with electric cars is that you can't just drive to a gas station and refill in a few minutes. You need to know where charging stations are and moreover if they are available. Depending on the car's battery the charging process can take between 4-10 hours² and therefore needs to be scheduled carefully to fit in the daily schedule of the people using electric cars.

To make electronic mobility more accessible and to reduce the carbon footprint of the transportation sector, eMall tackles this problem by offering a software that allows you to know about the nearest charging station, its prices and if it is free for you to use. It simplifies the charging process of electric cars and makes them therefore more suitable for everyday use.

This document presents the full description of the architectural design for the system, along with each component design and the logic behind its choices. Further in the document, it is possible to find mockups of the main interfaces, implementation procedures and test and integration plans for the software-to-be. With this document, it is expected to establish a clear communication between Requirement Analysts, Software Architects and the Developers responsible for the software creation.

1.2 Scope

eMSP is a subsystem of eMall that allows owners of electric cars to easily find and book a charging station. The CPMS is another subsystem of eMall that allows charging point operators to handle their energy acquisition and distribution and based on this the cost of their service. Moreover the CPMS performs the charging process itself.

As a group of three students we have to assume that the eMSP subsystem interacts with CPMSs of multiple CPOs and the CPMS is connected to multiple DSOs. Therefore the actors we need to consider are the Car owners who want to charge their cars and the charging point operators who want to manage their multiple charging stations.

Furthermore, in order to develop more specific requirements and to have better approximations in internet traffic and cloud capabilities, the target area of the eMall system is initially thought to be Italy.

¹ <https://www.epa.gov/transportation-air-pollution-and-climate-change/carbon-pollution-transportation>

² <https://monta.com/uk/blog/how-long-does-it-take-to-charge-an-electric-car/>

In order to simplify some processes that would compose a complete business plan for the software, the process in which a DSO is added in the energy system to be available to CPOs is outside of our scope. However, the process of adding the DSO to the CPO system is included.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Word	Definition
Charging station	A station with at least one charging socket where electric cars can recharge their batteries. In some stations additional batteries are connected that can store energy or give the energy to a connected car through the charging station.
Charging Socket	A socket is part of a charging station. It is the cable connecting the charging station and electric car.
Type of Charging sockets	There are different charging stations depending on the velocity of their charging process. The options are slow, fast and rapid.
Battery	A battery is connected to a charging station and can store energy or give the energy to a connected car through the charging station.
Charging Process	The charging process describes the process where the charging station gives energy to the car's battery until the car's battery is full.
Charging Station's External Status Information	External information of a charging stations consists of: Number of charging sockets available, their type such as slow/fast/rapid, their cost, and, if all sockets of a certain type are occupied, the estimated amount of time until the first socket of that type is freed
Charging Station's Internal Status Information	Internal information of a charging stations consists of: Amount of energy available in its batteries, if any, number of vehicles being charged and, for each charging vehicle, amount of power absorbed and time left to the end of the charge

1.3.2 Acronyms and Abbreviations

Word	Description
RASD	Requirements Analysis and Specification Document
WP X	World Phenomena number X
SP X	Shared Phenomena number X
GX	Goal number X

DX	Domain assumption number X
RX	Requirement number X
eMall	e-Mobility for All
eMSP	e-Mobility Service Providers
CPO	Charging Point Operator
CPMS	Charge Point Management System
DSO	Distribution System Operators
GUI	Graphical User Interface
User	Car Owner or CPO

1.4 Reference Documents

The requirements and conclusions mentioned in this document are all derived from the specification document “Assignment RDD A.Y. 2022-2023 Software Engineering 2” and the “A.Y. 2022-2023 Software Engineering 2 - RASD Project”.

1.5 Document Structure

This document consists of six sections. The first section was about the purpose of this document. It explained the fundamental definitions and acronyms used in this document and the application domain of the software that is described in the following sections.

The second section consists in the Architectural Design in which is presented how the system-to-be will be organized and how the high-level components should be organized. In addition, it is shown how each component is going to be connected between themselves by a component diagram. Finally, the section ends with the component interface diagram that shows which interfaces need to be developed and a set of sequence diagrams showing how some of the components will interact with one another.

In the third section, it is presented a set of mock-ups of the interfaces presented in the section before, which helps the front-end developers to visualize and understand what needs to be done for the project.

Fourth section shows the Requirement Traceability Matrix, in which it presents the connection between each component and the requirements of the system presented in the RASD. The mapping is a good method to figure out if the architecture is lacking any sort of component or attribute that is needed to fulfill all components completely.

The Fifth section proposes the ideas and methods to implement the architecture and components described as well as how the testing of the system after completion should be done.

The last two sections contain, respectively, the effort spent and the report by each author and the references used in order to make it.

2 Architectural design

2.1 Overview: High-level components and their interaction

The architecture of the eMSP and CPMS follows a two-tier (Client/Server) architecture. This means GUI, application logic and data is distributed on client or server.

For the eMSP we decided on a remote data access approach where the GUI and the application logic is part of the client's side and the data is part of the server's side. Since the eMSP needs to be smart by recommending offers to the client and recently checking the car owner's battery, it makes sense to keep the application logic on the client's side and only decouple the data, so it can be more secure on the server's side.

For the CPMS we decided on a remote presentation approach where only the GUI is on the client's side and application logic and data is on the server's side. The CPMS has also an automatic mode, therefore the client is not needed in this case so it is necessary that we decouple the GUI from the application logic, so that the CPMS can work independently.

Before we continue we will shortly describe the three components of the architecture.

- 1. Graphical User Interface (GUI)**

The GUI will consist of a representation of the functionality offered to the user. The User can interact with it to get the information he needs. For this the GUI will process the inputs and forward them to the application logic.

- 2. Application Logic**

The Application Logic consists of all the functionality of the system. It requests data from the data layer if needed and will process it to send it to the GUI for representation. It is also the layer responsible for communication with external APIs such as the Payment API.

- 3. Data**

The data will store all necessary data the system requires. It can get requests from the application layer.

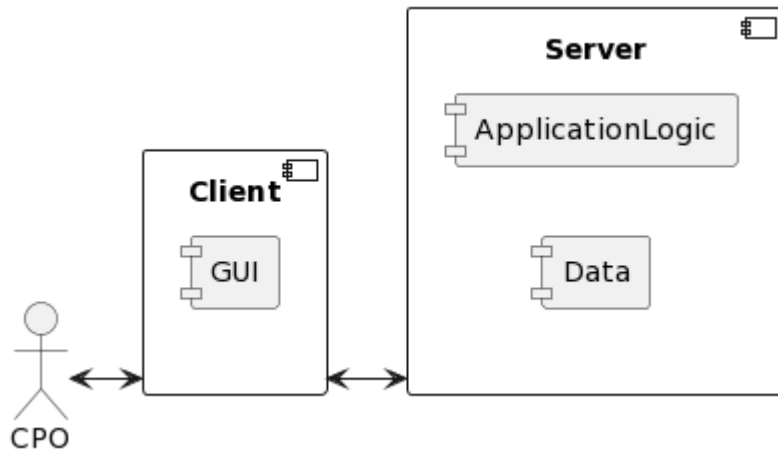


Figure 1: High level architecture for CPMS

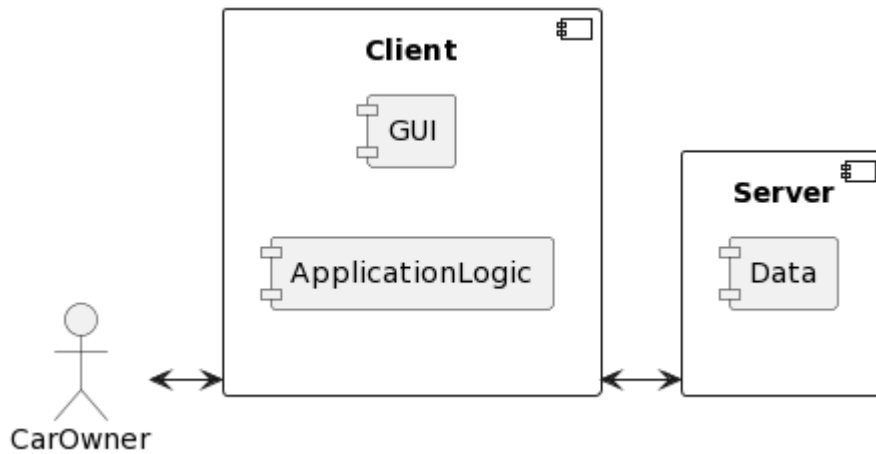


Figure 2: High level architecture for eMSP

2.2 Component view

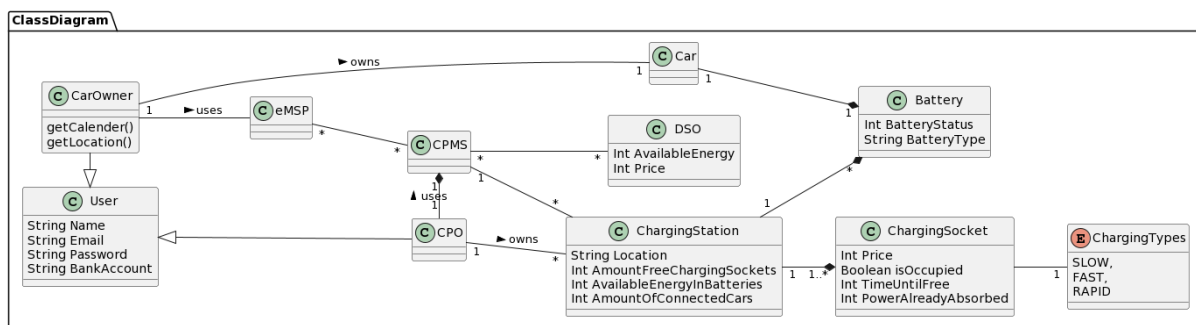


Figure 3: Class diagram

First we see here a class diagram of the different entities in the eMSP and CPMS system. Next we will look into the components more closely. The following diagrams will show the used components first for the eMSP and then for the CPMS. Each one is splitted into client and server side, the GUI is shown in the ClientApplication and the Data in the DBMS.

2.2.1 CPMS

- **Client Application** - Client application component, e.g. mobile phone
- **Router** - Client sends requests to the server, that are rerouted to the correct services
- **Authentication Service** - Component that handles verification and authentication of new and already registered users
- **Model** - Component that handles the database communication by containing an object-relation mapping for other components to use when they request data from the database
- **ChangePriceService** - Handles the changing of prices and special offers for charging stations
- **ActivateAutomaticModeService** - Handles the activation of the automatic mode
- **BuyEnergyService** - Handles the process of buying energy from DSOs
- **Internal/External_StatusInformationService** - Handles the information gain of the internal and external status of all charging stations
- **UseEnergyFromBatteriesService** - Handles the process of using the stored energy from batteries connected to the charging stations to charge cars
- **ChargingService** - Handles the initialisation and completion of the charging procedure
- **PaymentAPIHandler** - Handles the communication between the external Payment component that cares for the correct money transfer
- **ChargingStationAPIHandler** - Handles the communication between the charging stations and the CPMS e.g. fetching status information or initializing a charge
- **DSO API Handler** - Handles the communication between the DSO and CPMS system e.g. to get current price information or buy energy
- **eMSP API Handler** - Handles the communication between the CPMS and eMSP e.g. for the charging procedure
- **DBMS** - Database Management System

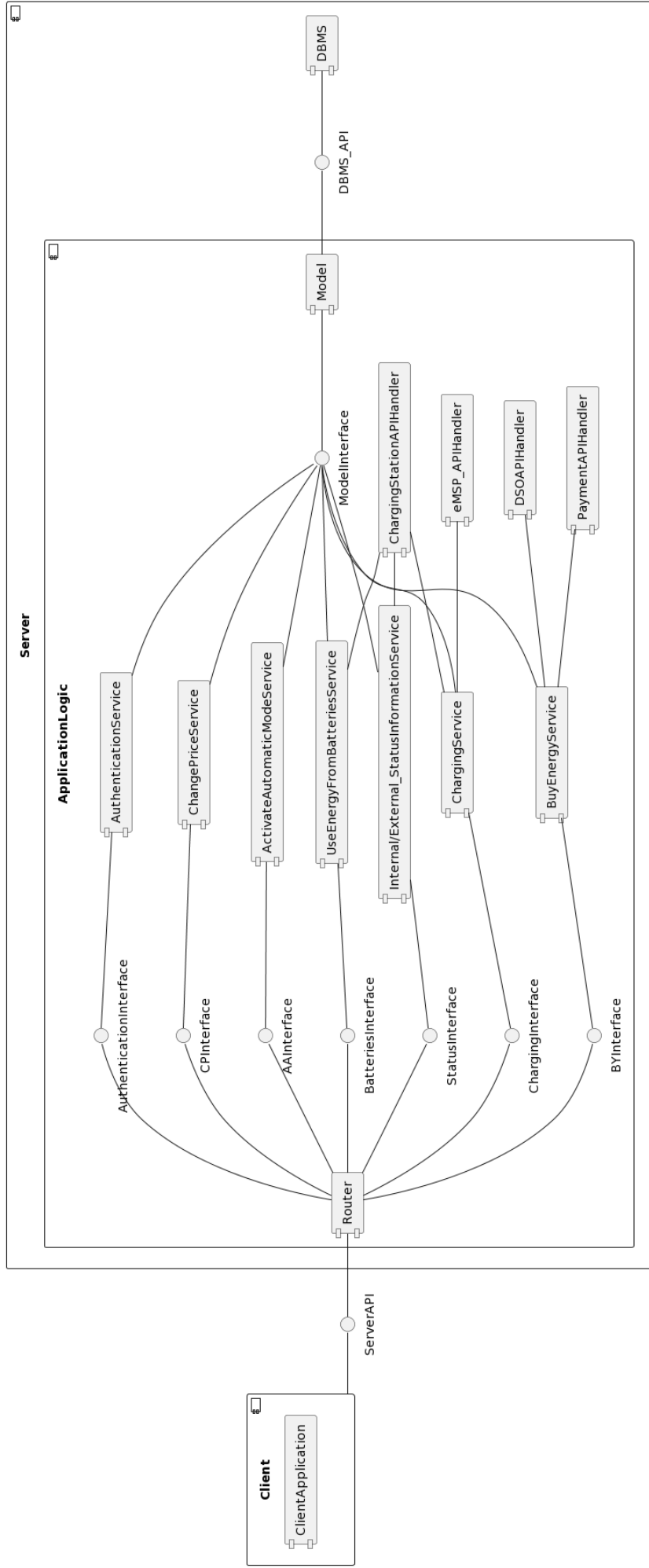


Figure 4: Component View for CPMS

2.2.2 eMSP

- **Client Application** - Client application component, e.g. mobile phone
- **Router** - Client sends requests to the server, that are rerouted to the correct services
- **Authentication Service** - Component that handles verification and authentication of new and already registered users
- **Model** - Component that handles the database communication by containing an object-relation mapping for other components to use when they request data from the database
- **DBMS** - Database Management System
- **NearbyChargingStationService** - Handles the search for charging stations close to the car owner's position
- **PayChargeService** - Handles the paying procedure after booking a charge
- **BookChargeService** - Handles the booking procedure for a charge
- **ChargingService** - Handles the initialisation and completion of the charging procedure
- **SuggestionService** - Handles the proactive suggestion procedure of the eMSP to the user if a fitting charging possibility is found
- **CarBatteryService** - Handles the continuously checking of the car's battery to react on it e.g. by suggesting a charge
- **PhoneDataService** - Handles the checking of the owner's schedule if the car battery is empty enough to consider a recharge (at 30%) and the locationing service of the phone
- **PaymentAPIHandler** - Handles the communication between the external Payment component that cares for the correct money transfer
- **CPMS_APIHandler** - Handles the communication with the CPMS system e.g. to start charging or to book
- **CarBattery_APIHandler** - Handles the communication between eMSP and the car's battery
- **PhoneData_APIHandler** - Handles the communication between the eMSP and the owner's phone for getting location data or schedule information

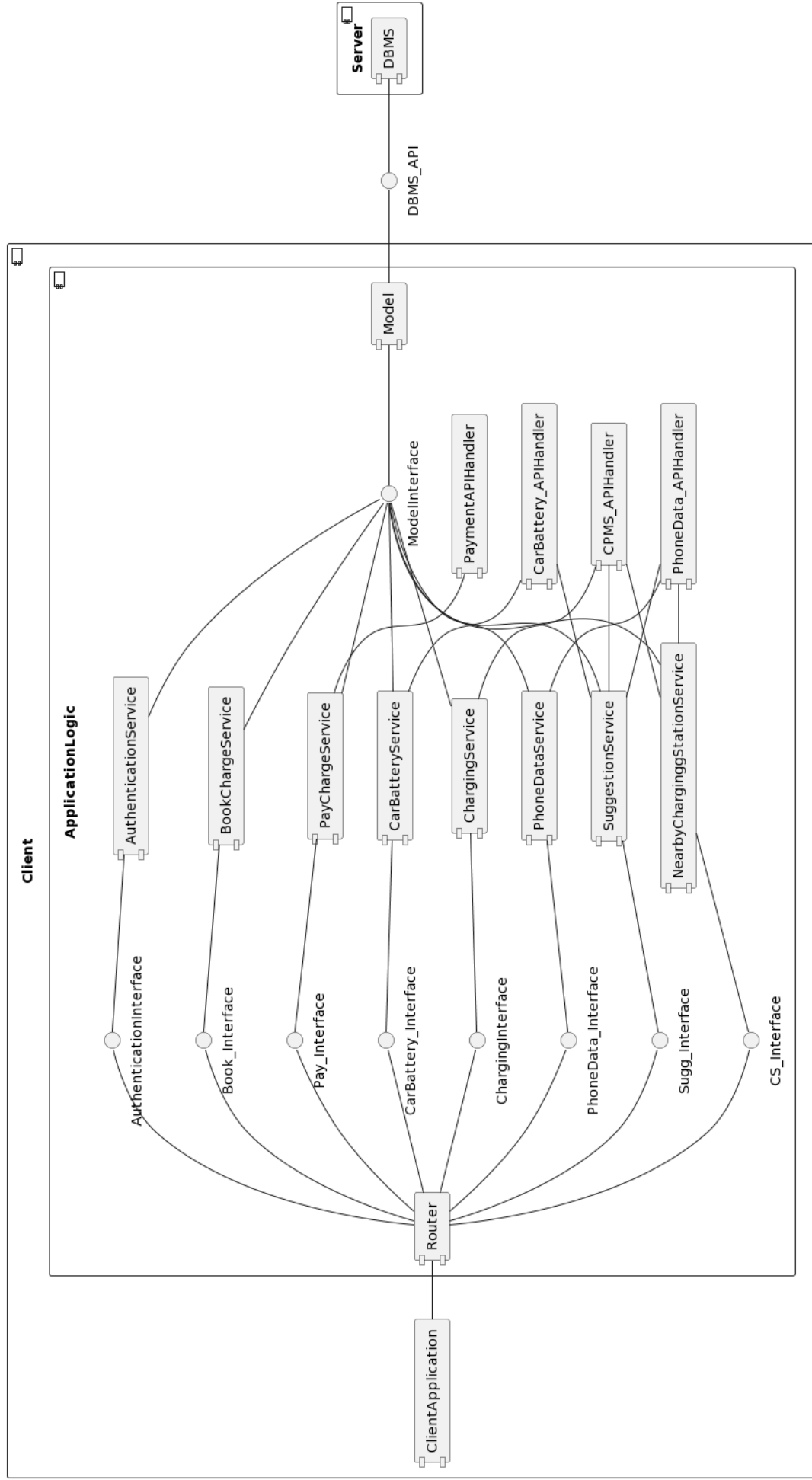


Figure 5: Component View for eMSP

2.3 Deployment view

In this section we will present the deployment view. It shows the decision we made on the used tools and protocols that contribute to the system's functionalities.

2.3.1 eMSP

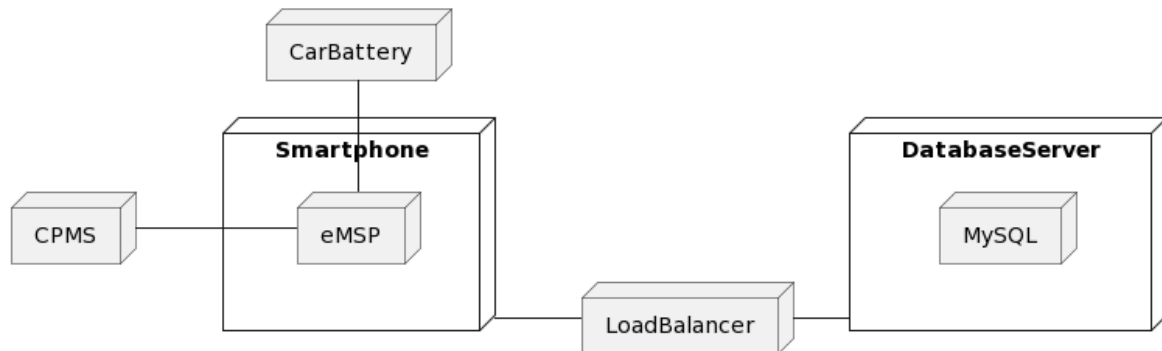


Figure 6: Deployment View for eMSP

- **Smartphone** - Device to access the eMSP application that has a locationing service and a saved schedule
- **LoadBalancer** - Handles incoming requests and balances the workload among multiple Database Server
- **eMSP** - contains the application logic for the eMSP system
- **DatabaseServer** - Hosts the database that is used for the eMSP system
- **MySQL** - Database used for the eMSP system
- **Car Battery** - Battery of the car that is checked by the eMSP regularly
- **CPMS** - System that gives the eMSP information about the charging stations

2.3.2 CPMS

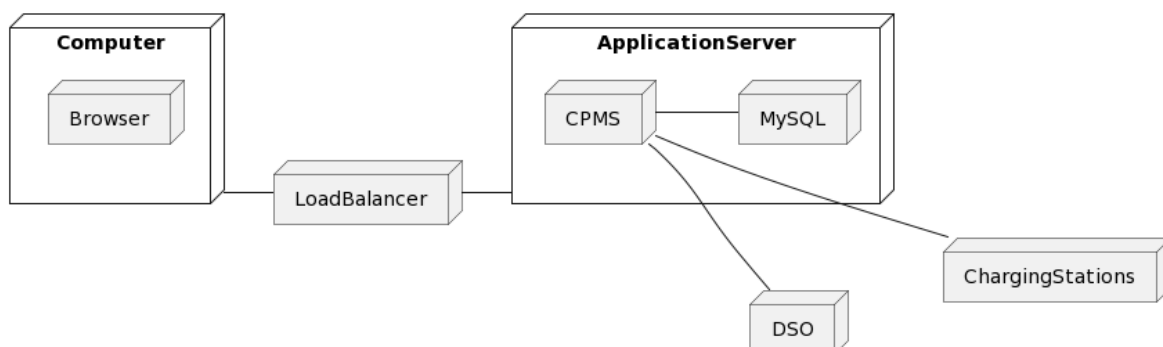


Figure 7: Deployment View for CPMS

- **Computer** - Device used for accessing the CPMS webpage
- **Browser** - Is used to open the CPMS webpage

- **LoadBalancer** - Handles incoming requests and balances the workload among multiple Database Server
- **ApplicationServer** - Contains the CPMS system and the MySQL database
- **CPMS** - contains the application logic for the CPMS system
- **MySQL** - Database used for the eMSP system
- **DSO** - People that sell energy to the CPO
- **ChargingStations** - Stations where a car can charge and that are managed by a CPMS

2.4 Runtime view

1. Getting a view from an application

Here we see a typical procedure if the user requests a specific view. Since this procedure is the same in most cases, we present here an example for a “ApplicationService”. If it works like this, we will simplify this procedure in the following sequence diagrams.

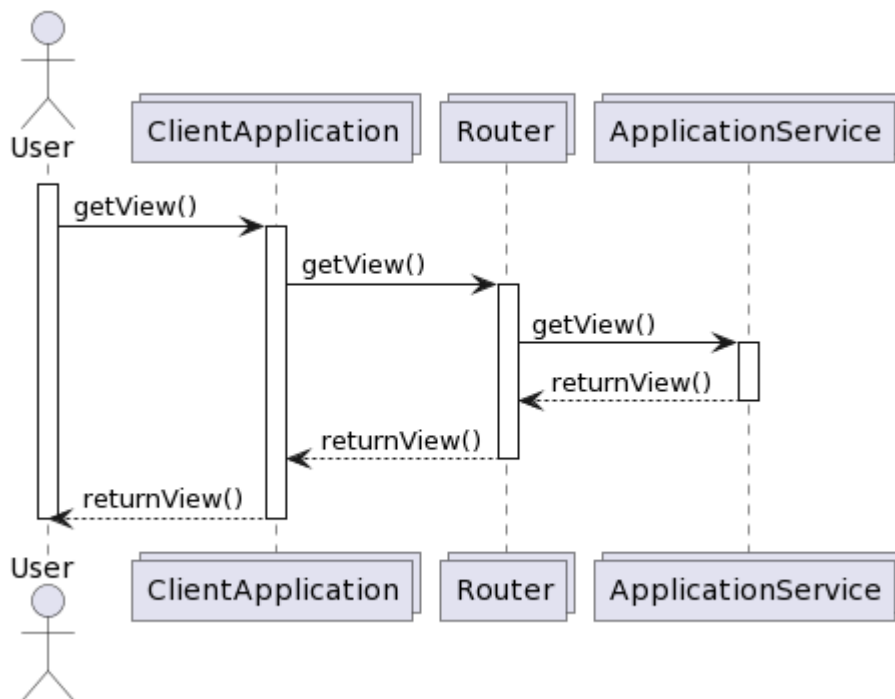


Figure 8: Getting a view from an application

2. Registering a user

Here we see the registration procedure. Since it works the same way for CPMS and eMSP, we use only one view for both systems.

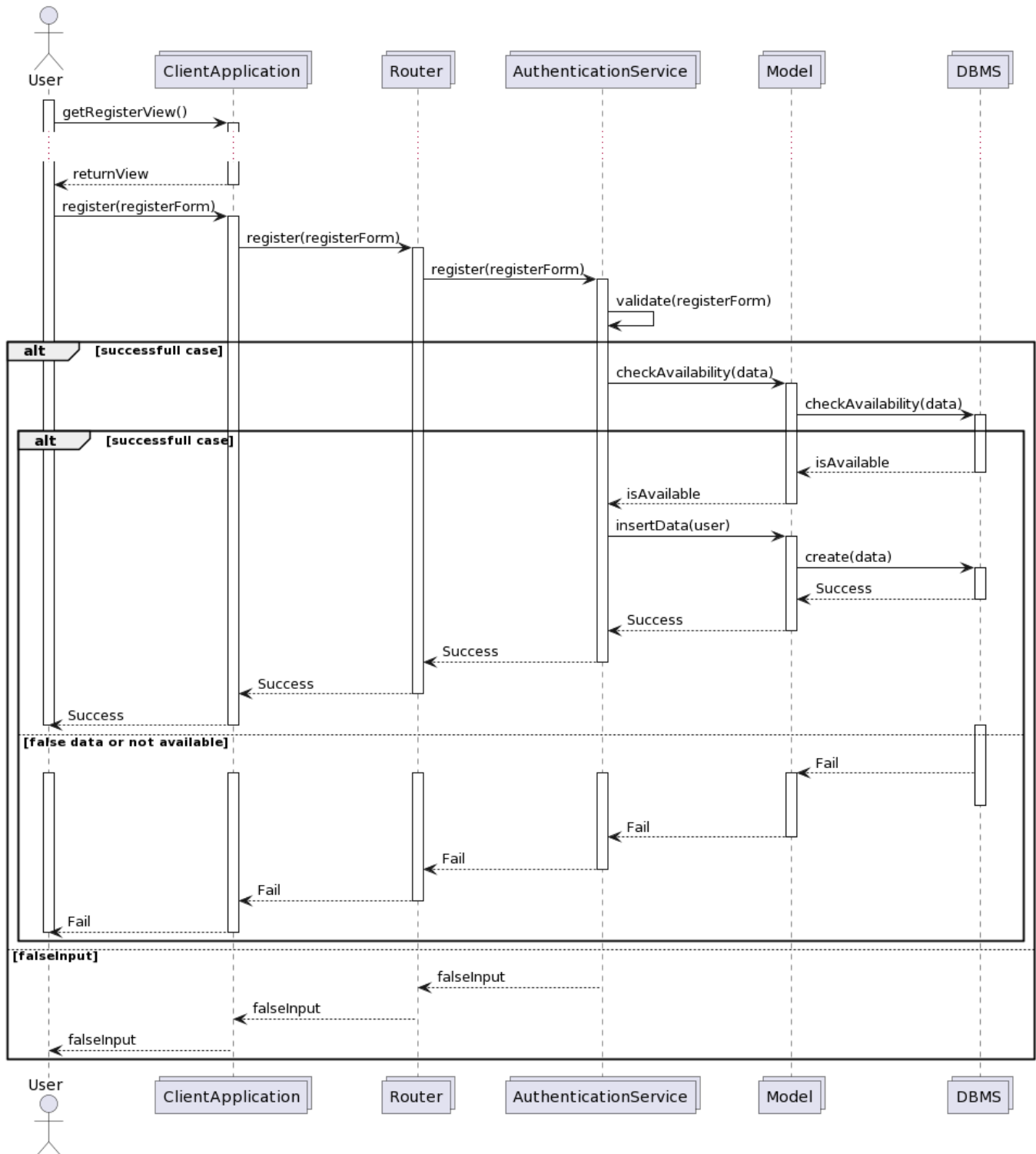


Figure 9: Registering a user

3. Login a user

Here we see the login procedure. Since it works the same way for CPMS and eMSP, we use only one view for both systems

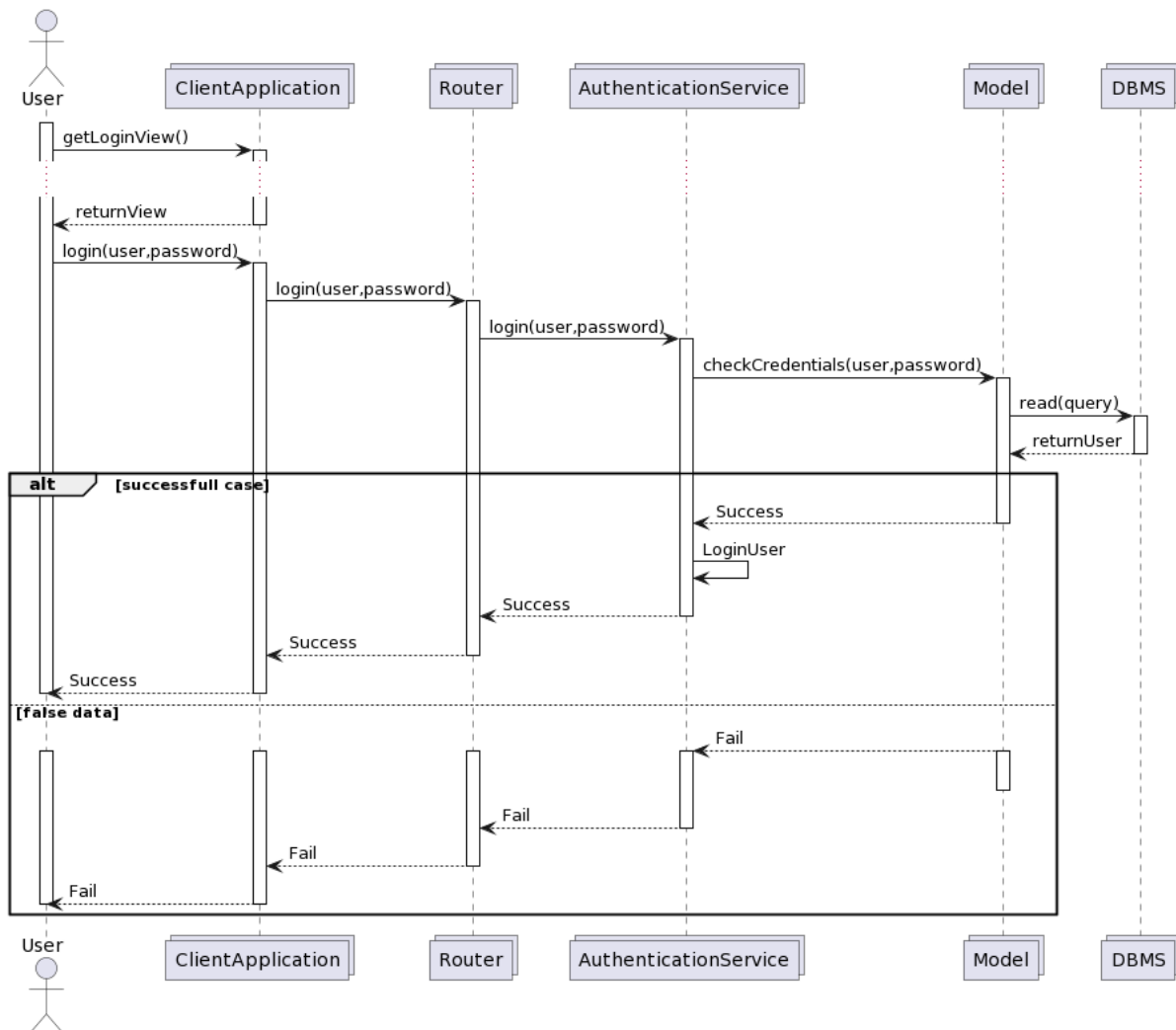


Figure 10: Login a user

4. Car Owner books a charge

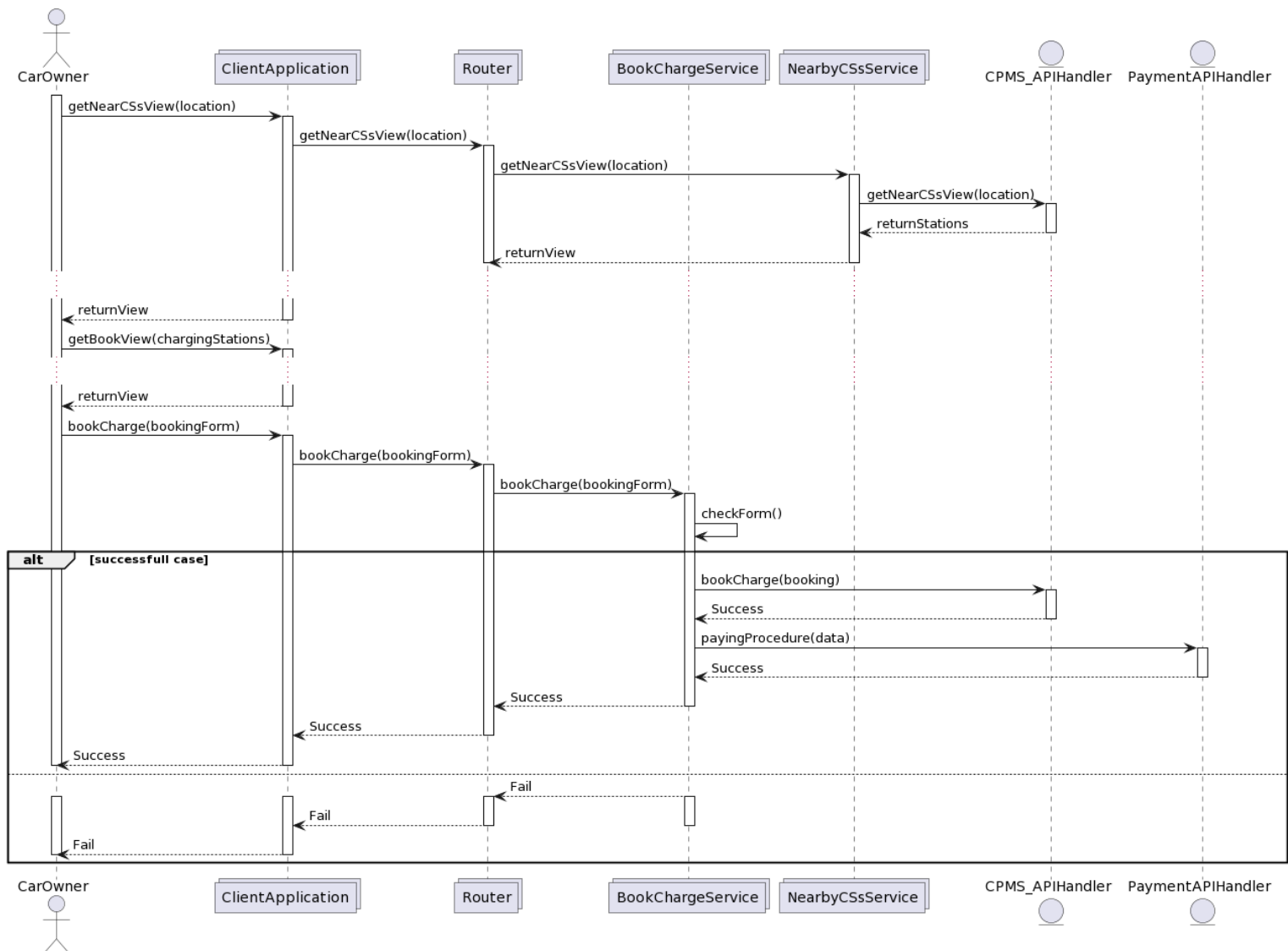


Figure 11: Car Owner books a charge

5. Start charging procedure

Here we see the charging procedure inside the eMSP in the first diagram. We left out the failure case since it is the same as the failure cases we already saw. Simply, a failure message is returned. Inside the CPMS_APIHandler, the eMSP_APIHandler of the CPMS is called which is displayed in the second diagram.

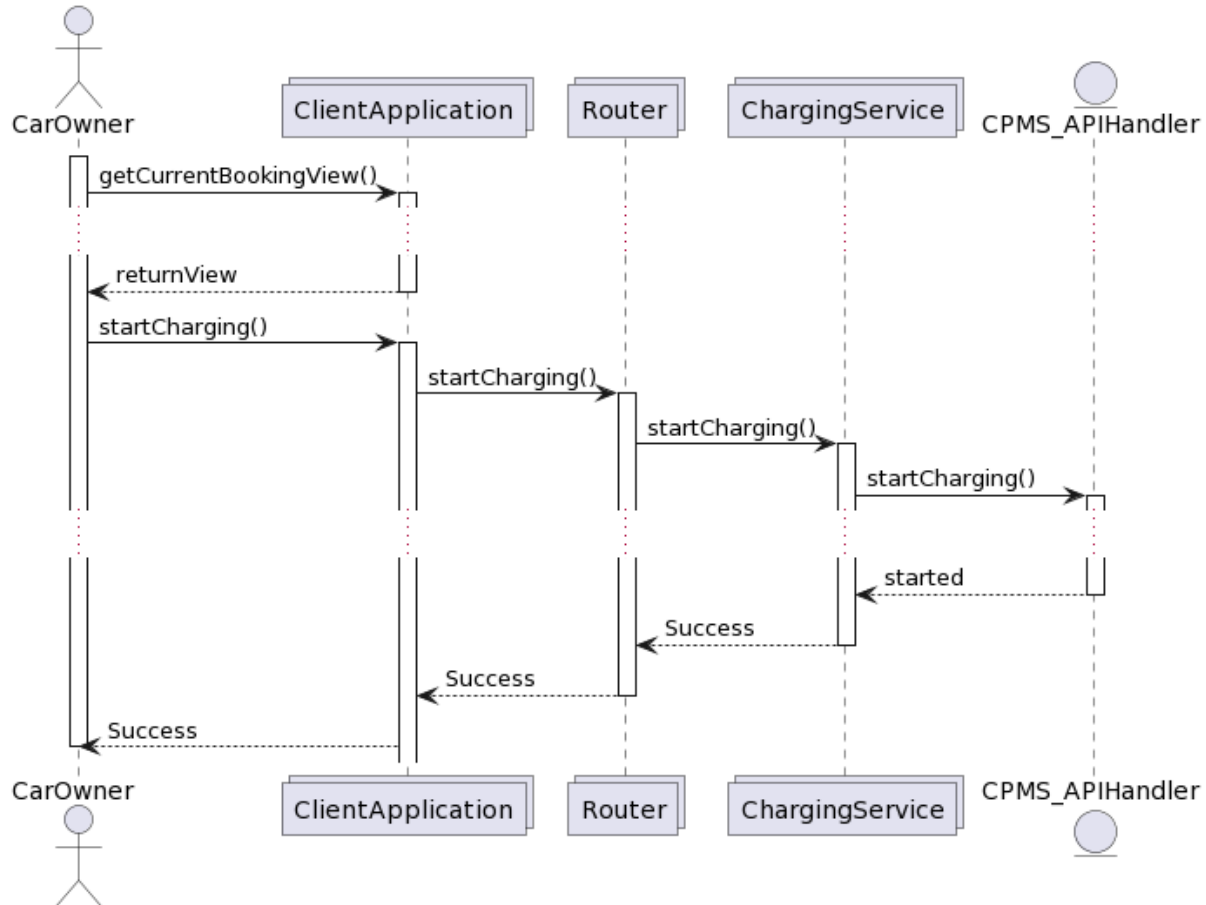


Figure 12: Start charging procedure eMSP view

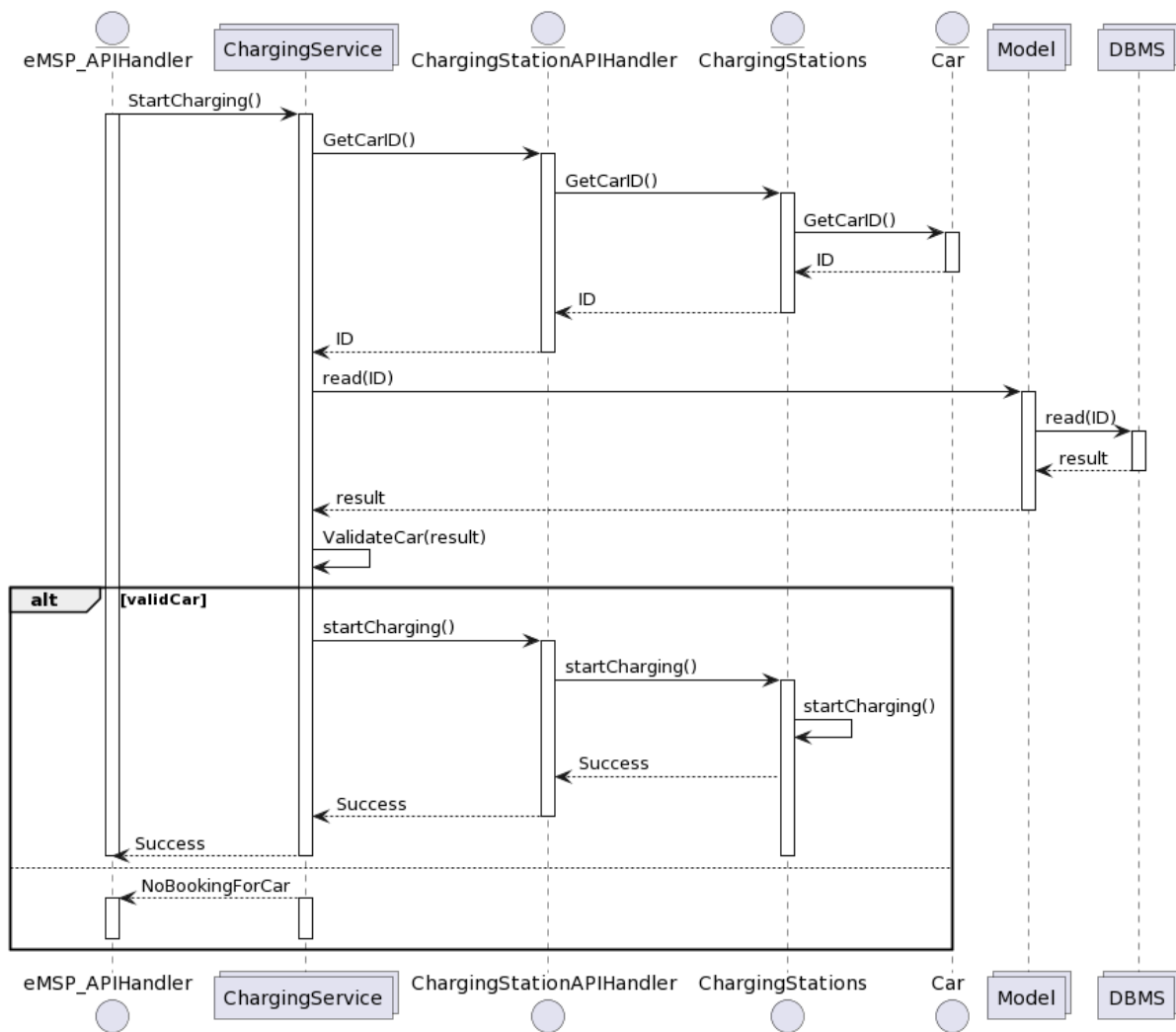


Figure 13: Start charging procedure eMSP view

6. Stop charging procedure

In the first diagram we see the stopping of the charging procedure initiated by the Charging Station inside the CPMS system. Then, in the second diagram, we see how the eMSP system reacts.

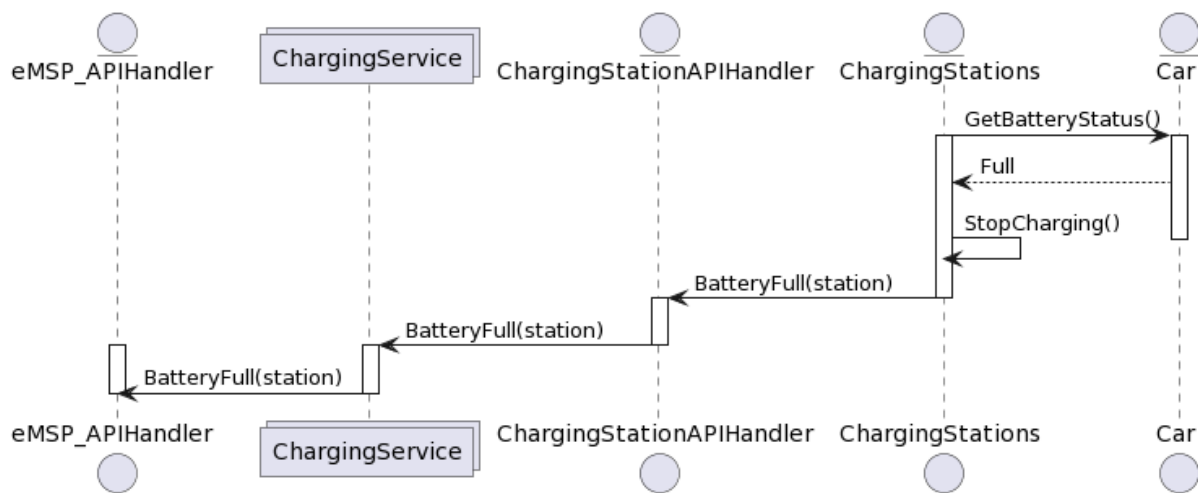


Figure 14: Stop charging procedure from CPMS view

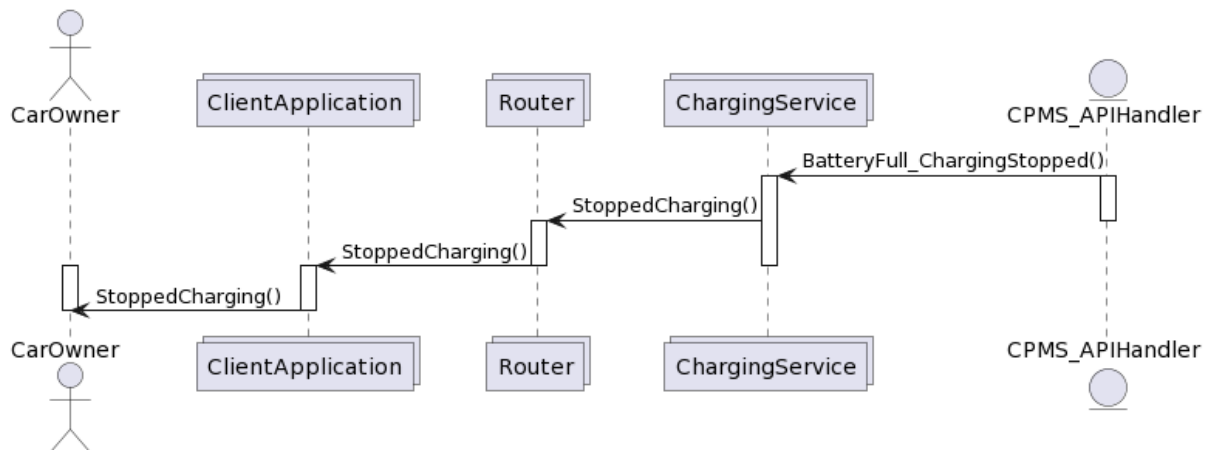


Figure 15: Stop charging procedure from eMSP view

7. eMSP system suggests charging offer to Car Owner

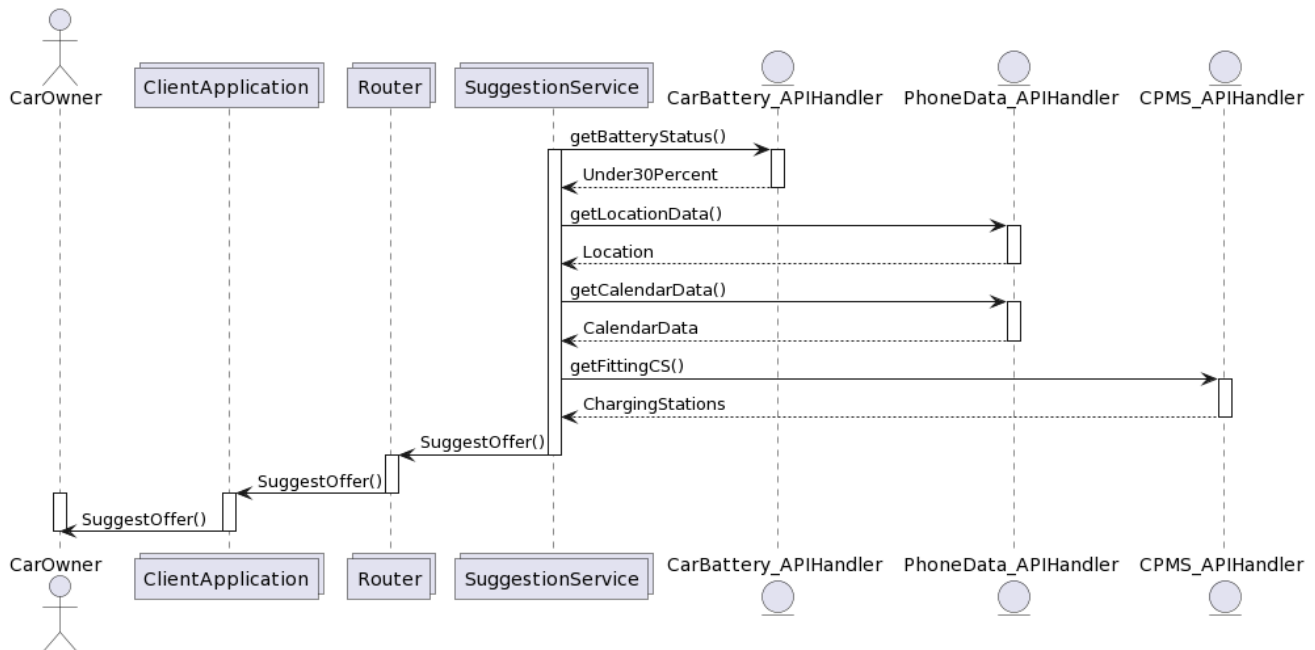


Figure 16: eMSP system suggests charging offer to Car Owner

8. CPO checks status information about charging stations

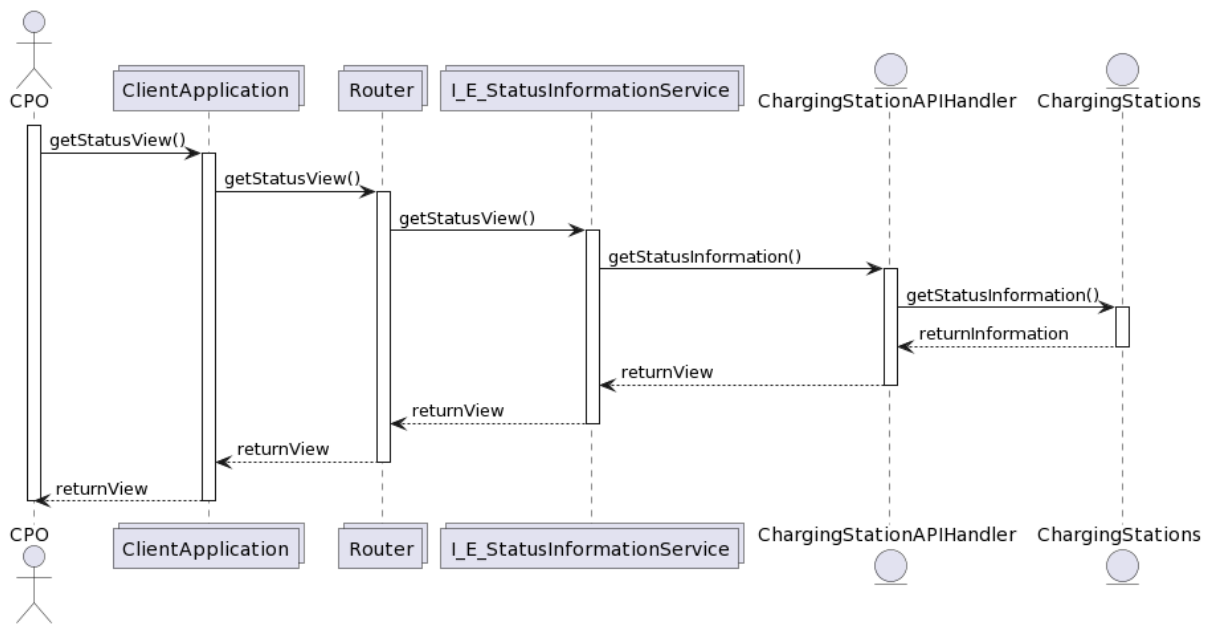


Figure 17: CPO checks status information about charging stations

9. CPO buys energy from DSO

We left out the failure case since it is the same as the failure cases we already saw. Simply, a failure message is returned.

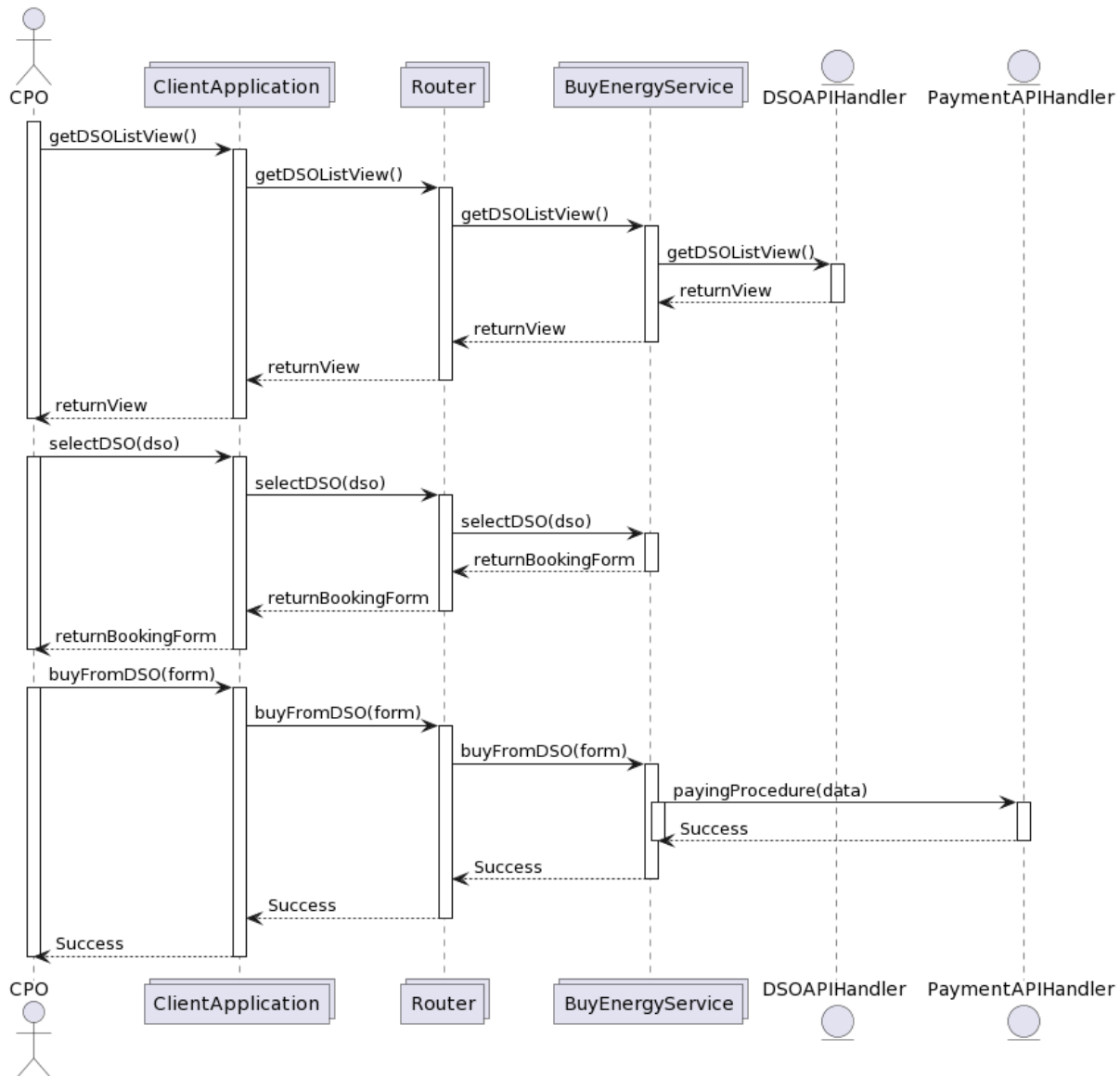


Figure 18: CPO buys energy from DSO

10. CPO chooses to acquire energy from Charging Station's batteries

We left out the failure case since it is the same as the failure cases we already saw. Simply, a failure message is returned.

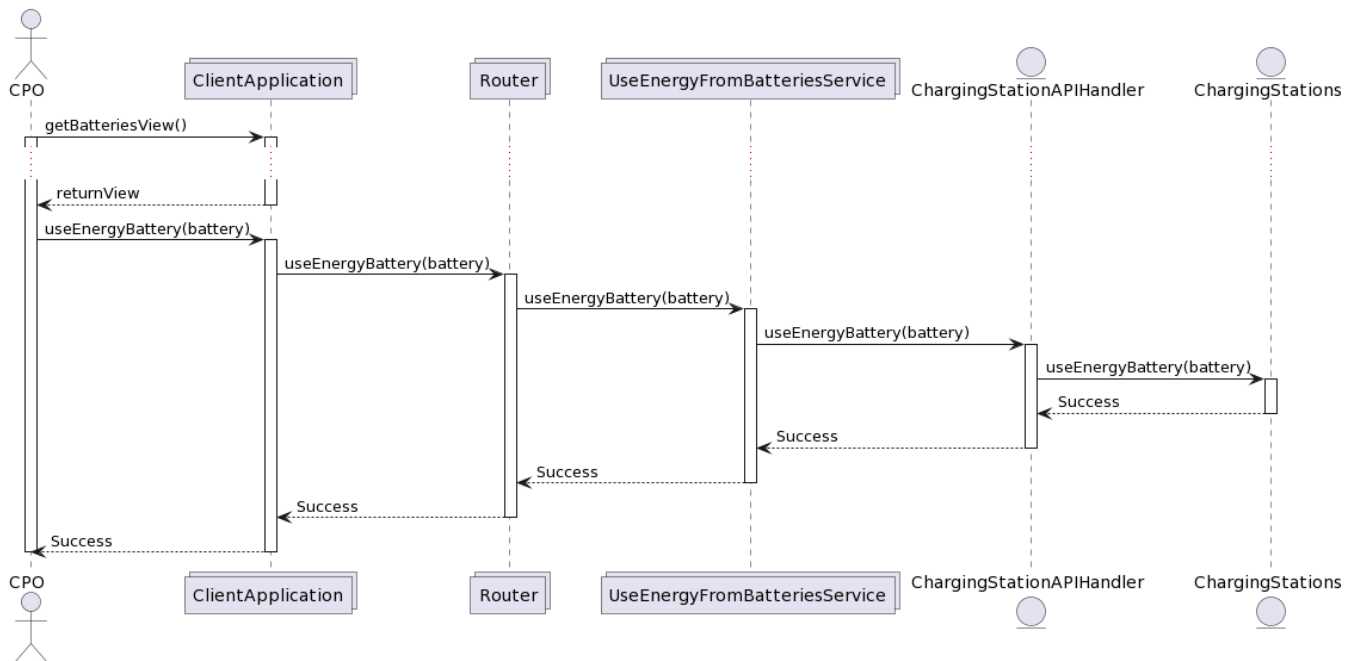


Figure 19: CPO chooses to acquire energy from Charging Station's batteries

11. CPO sets "Special offer" for his Charging Stations

We left out the failure case since it is the same as the failure cases we already saw. Simply, a failure message is returned.

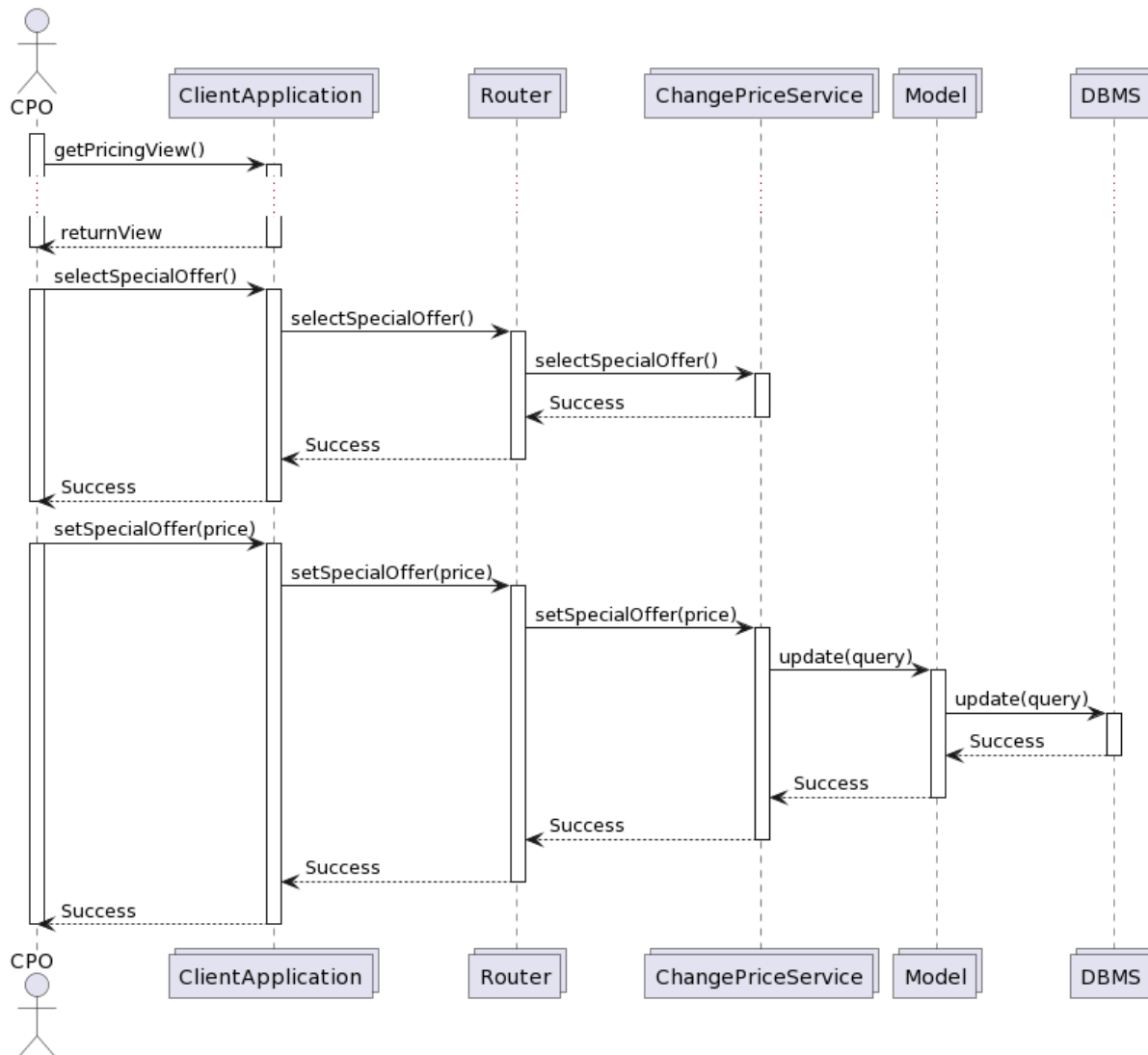


Figure 20: CPO sets "Special offer" for his Charging Stations

12. CPO sets Price for his Charging Stations

We left out the failure case since it is the same as the failure cases we already saw. Simply, a failure message is returned.

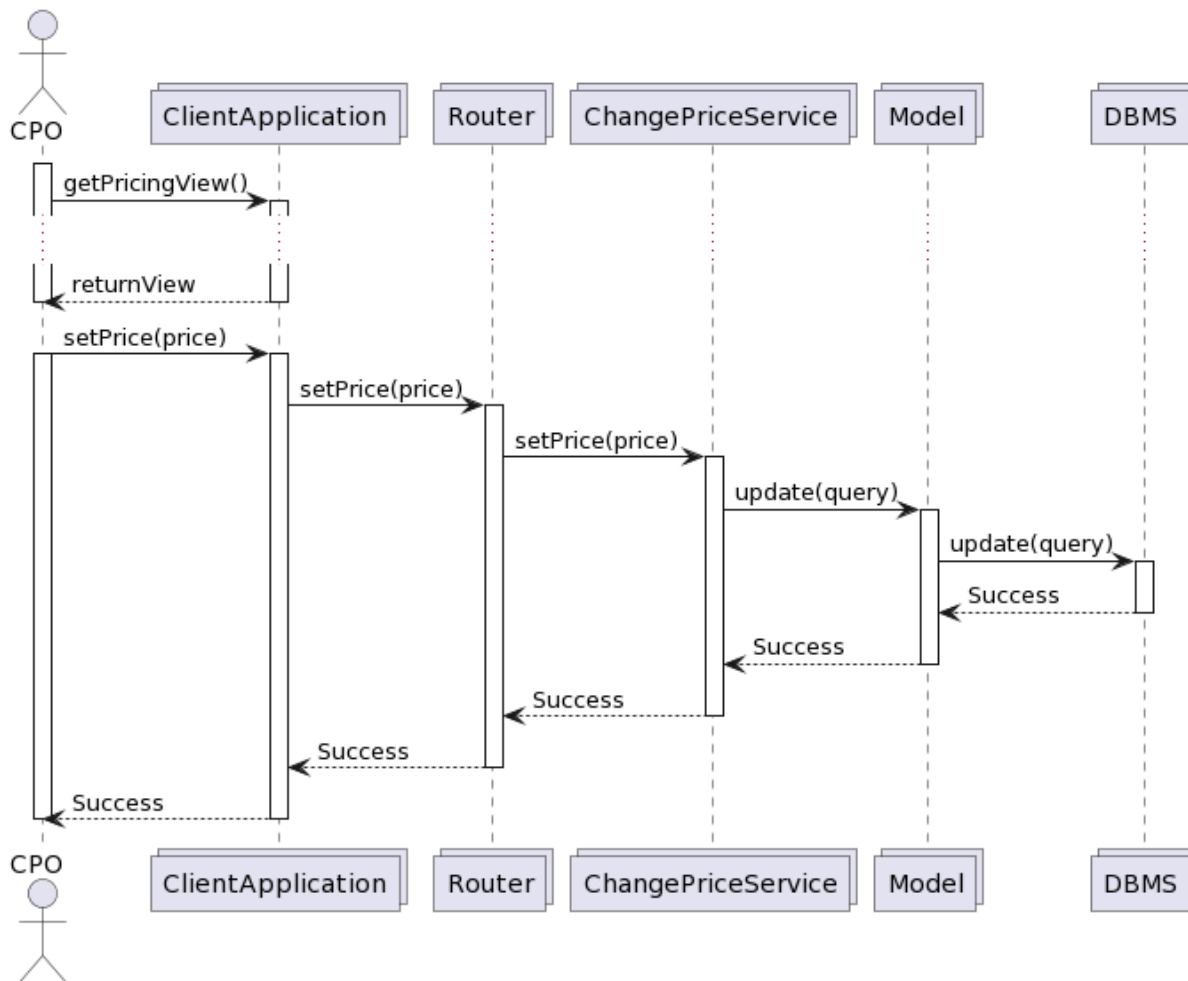


Figure 21: CPO sets Price for his Charging Stations

13. CPO activates automatic mode

We left out the failure case since it is the same as the failure cases we already saw. Simply, a failure message is returned.

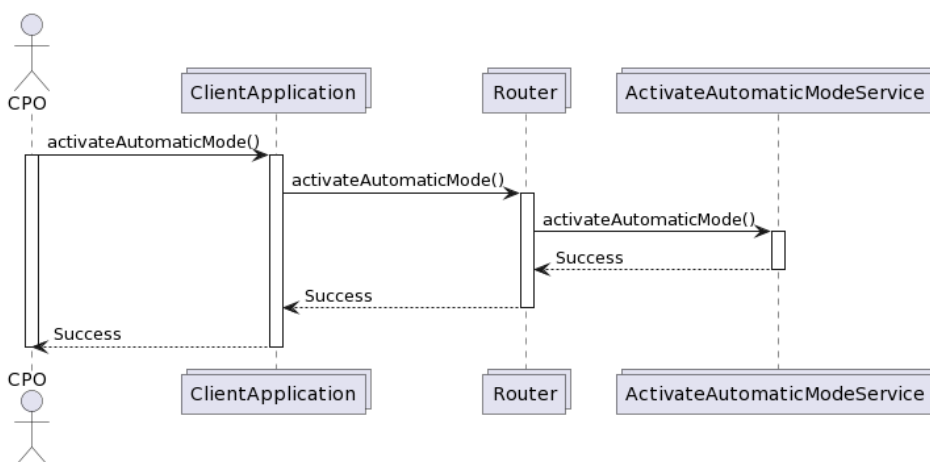


Figure 22: CPO activates automatic mode

14. CPO stores energy in Charging Station's batteries

We left out the failure case since it is the same as the failure cases we already saw. Simply, a failure message is returned.

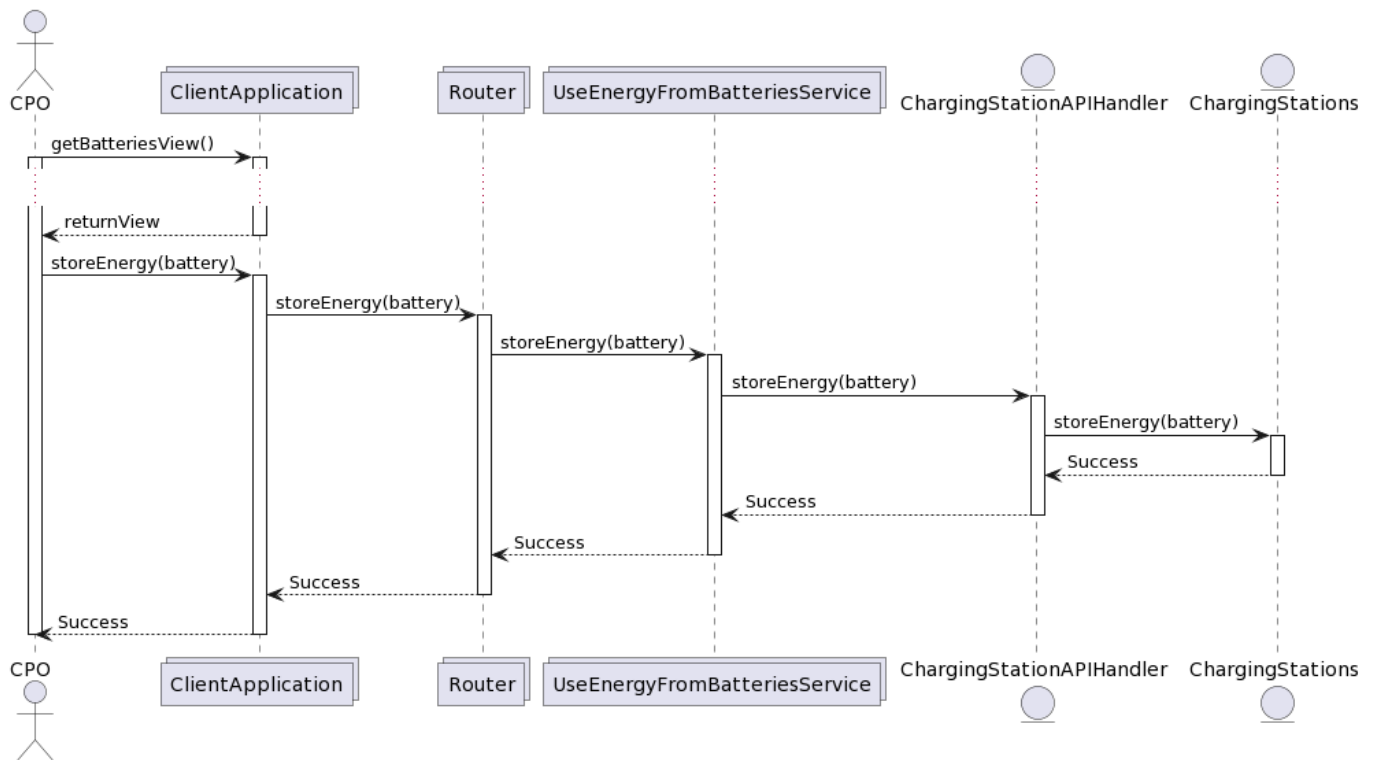


Figure 23: CPO stores energy in Charging Station's batteries

2.5 Component interfaces

2.6 eMSP

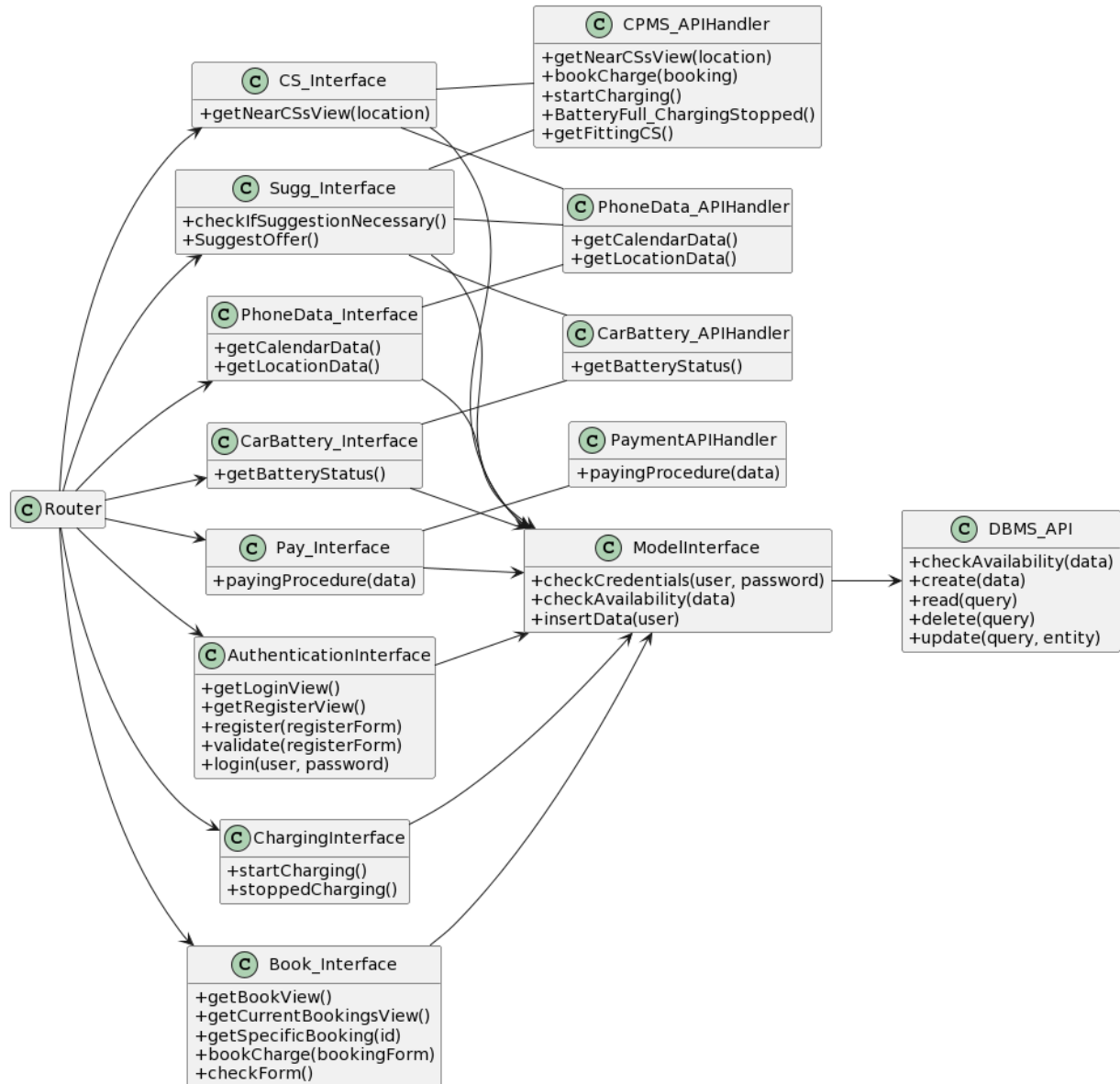


Figure 24: Component Interfaces eMSP

2.7 CPMS

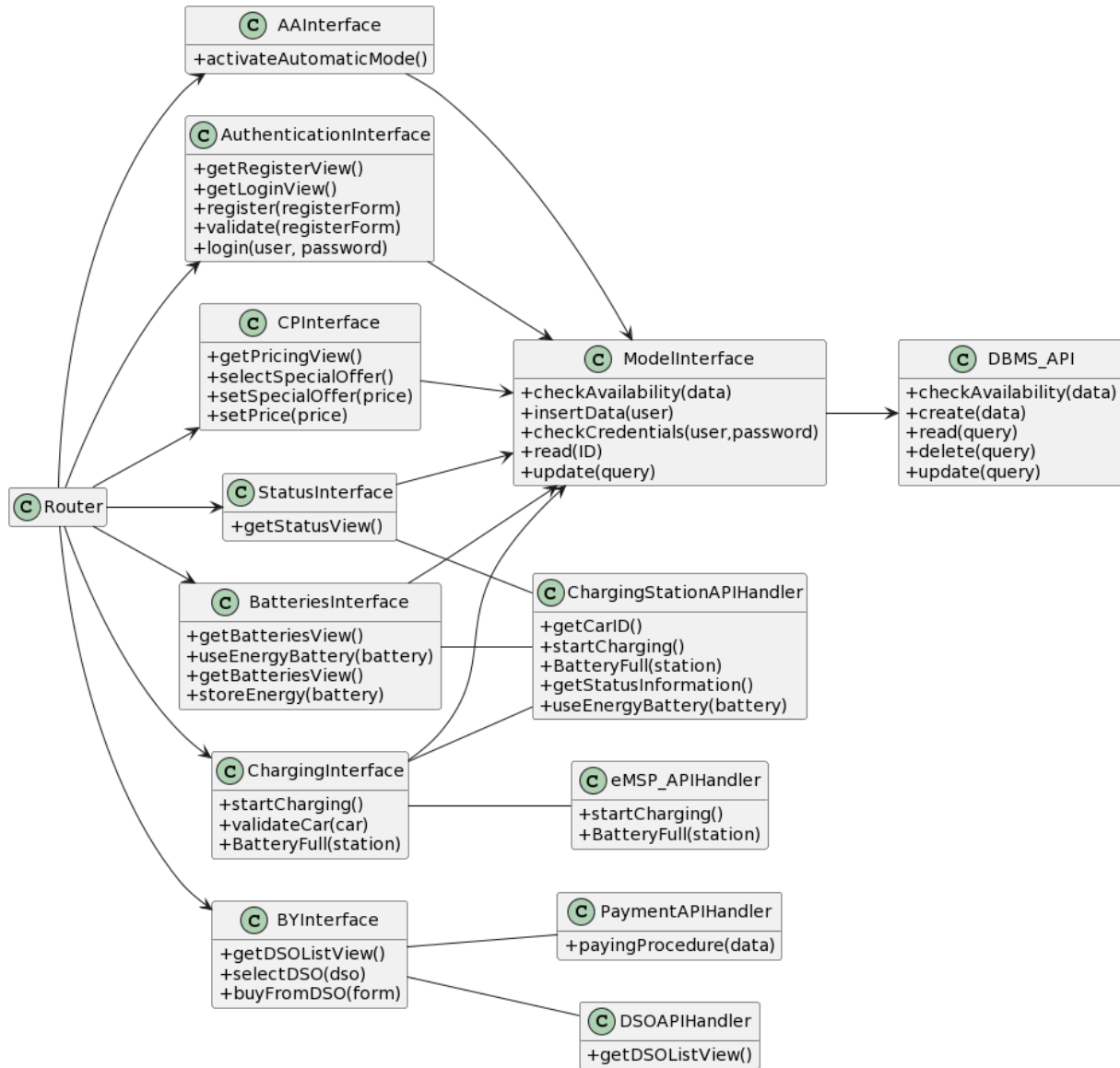


Figure 25: Component Interfaces CPMS

2.8 Selected architectural styles and patterns

- **Two tier architecture**

We structured both of our systems into two-tier architecture. The CPMS uses the variant, in which only the GUI lays on the side of the user, whereas application logic and data lay on the server side. We choose this variant because the CPMS offers an automatic mode for which no CPO interaction is necessary and therefore application logic on the CPO side would complicate this mode.

We decided on the two tier architecture to increase reusability and scalability. It makes it easier to divide and replace parts or change the resources a part takes. A three tier architecture would have also been possible but since we have only simple communication with the database, we choose the two tier architecture for simplicity.

For the eMSP we choose the variant in which the application logic lays on the user side because the system offers charging suggestions which is more easy when the logic lays on the user side, since the system needs to do regular checks on the user's car battery.

2.9 Other design decisions

The whole payment system of the application is thought to be performed by a payment API through a payment gateway. This decision was held given the increasing complexity of account verification and fraud detection. Creating a verification system specifically for the application would imply a lot more work. software development and, therefore time spent.

3 User interface design

In this section, it is possible to find the mockups of the user interfaces developed. The eMSP interfaces are seen in mobile view (used by the Car Owners) while the CPMS views are presented in the PC view.

3.1 eMSP

3.1.1 Login

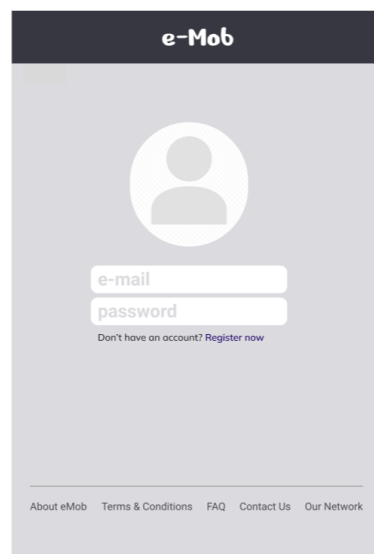
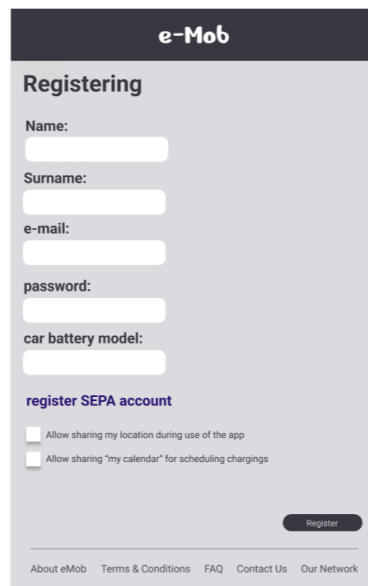


Figure 26: UI for Login

When a Car Owner opens the app, it is needed to inform the basic informations in order to have access to the app's functionalities. The screen is related to Runtime View 3 (Login a user) from section 2.4.

3.1.32 Registering

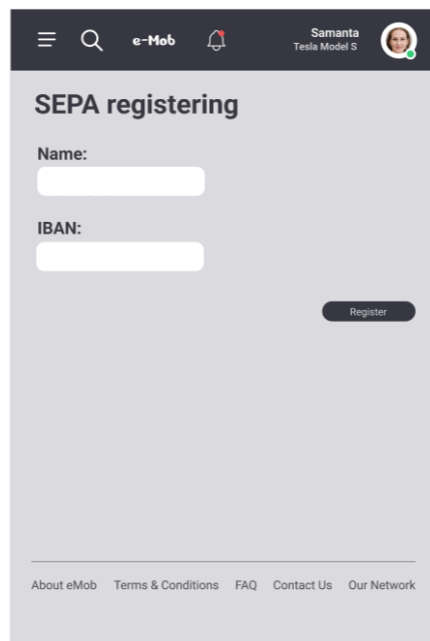


The image shows a mobile app interface for registering an account. At the top is a dark header with the 'e-Mob' logo. Below it, the title 'Registering' is displayed. The form contains several input fields: 'Name:', 'Surname:', 'e-mail:', 'password:', and 'car battery model:'. Below these fields is a section titled 'register SEPA account' with two checkboxes: 'Allow sharing my location during use of the app' and 'Allow sharing "my calendar" for scheduling chargings'. A 'Register' button is located at the bottom right of the form. At the very bottom, there is a footer with links: 'About eMob', 'Terms & Conditions', 'FAQ', 'Contact Us', and 'Our Network'.

Figure 27: UI for Register

If the user does not have an account yet, it is possible to create an account by clicking “Register Now” in the login screen. The Interface above would appear and will request the given data and SEPA registering. Furthermore, it is also necessary to accept the terms of use and data privacy practices. This UI is related to Runtime View 2 (Registering a user).

3.1.3 SEPA Register



The image shows a mobile app interface for SEPA registering. At the top is a dark header with a menu icon, a search icon, the 'e-Mob' logo, a notification bell, and a user profile section for 'Samanta Tesla Model S'. Below the header, the title 'SEPA registering' is displayed. The form contains two input fields: 'Name:' and 'IBAN:'. A 'Register' button is located at the bottom right of the form. At the very bottom, there is a footer with links: 'About eMob', 'Terms & Conditions', 'FAQ', 'Contact Us', and 'Our Network'.

Figure 28: UI for the registering of SEPA data in order to make purchases

Also part of the register, this separated UI is used to receive the data in order to allow payments using the SEPA.

3.1.4 Booking Charging Station

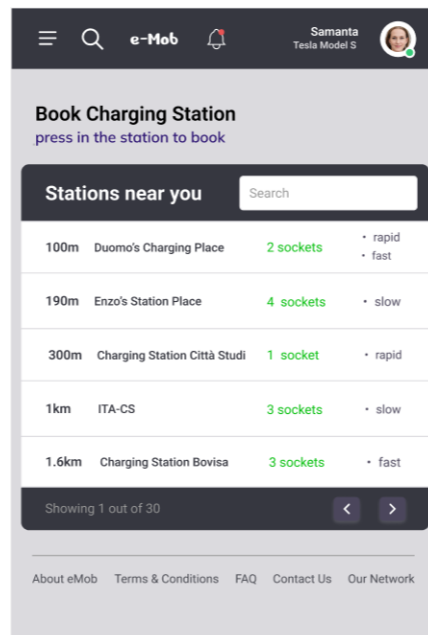


Figure 29: UI for Booking a charge

In this interface, the car owner is able to view the charging stations nearby and select one of them. After clicking in one, the next screen is shown in order to schedule an appointment. This Interface is related to Runtime view 4 (Car owner books a charge).

3.1.5 Schedule Charging

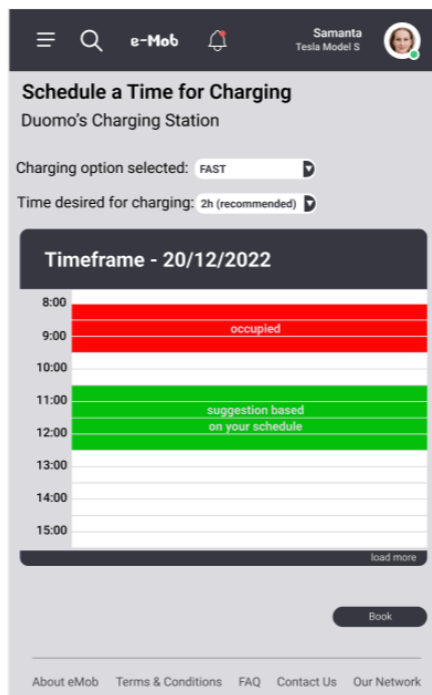


Figure 30: UI for selecting the period the user wants to use a charging socket

After selecting the type of socket and the amount of time desired, the car owner can choose, from the given calendar, a period that fits with their calendar (feature enhanced thanks to the sharing of data). This UI is part of Runtime view 4 as well.

3.1.6 Booking Confirmation

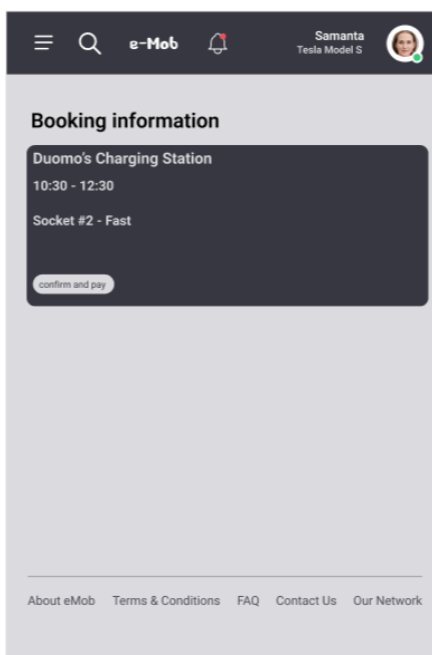


Figure 31: UI for booking information and confirmation of schedule

After scheduling the charging process, this UI is presented in order to show all data from the booking. The “confirm and pay” button will lead to the API payment interface, in which the payment (via SEPA) will be concluded. After that, the booking will be seen in the “Active Bookings” list.

3.1.7 View Active Bookings

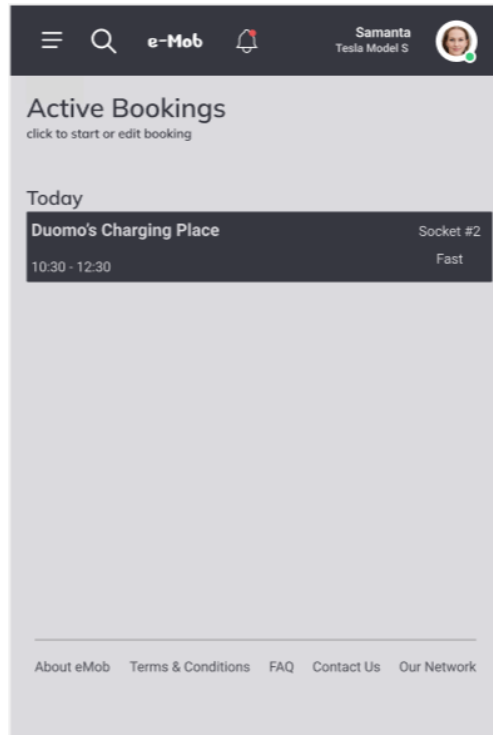


Figure 32: UI of Active Bookings page

This interface is related to view 5 (Start charging procedure). After clicking an active booking, the pop-up below is shown.

3.1.8 Charging Start

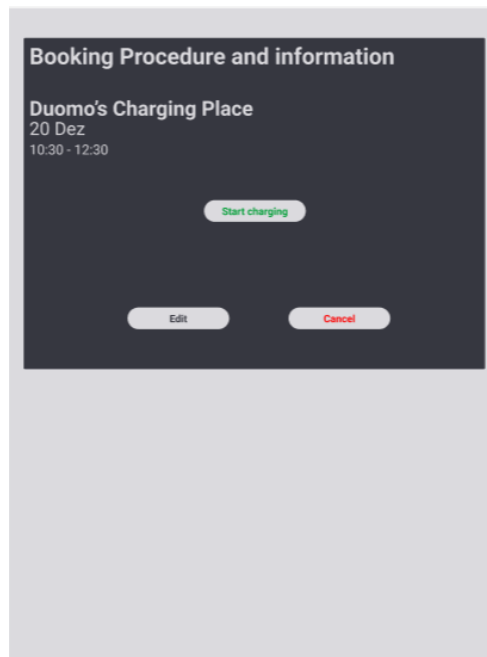


Figure 33: UI of the pop-up to start charging process

3.1.9 Charging Status

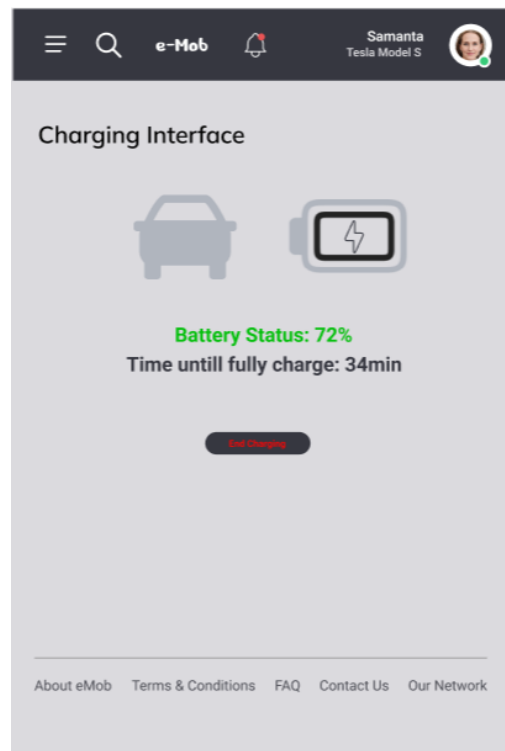


Figure 34: UI of the Charging Status Interface

During the charging process, it is possible to visualize in this view the current status of the charging (Runtime views 5 and 6).

3.1.10 Notifications

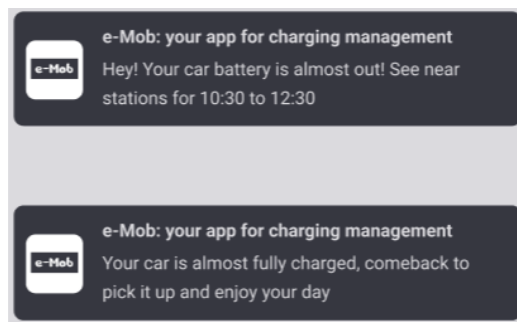


Figure 35: UI of possible notifications from the app

The system will also be able to inform the user via notifications. In the image above, the first one is shown when the car battery is below 30% and uses the calendar information to suggest a possible time (Runtime view 7). The second notification is shown when the charging process is complete.

3.2 CPMS

3.2.1 Login

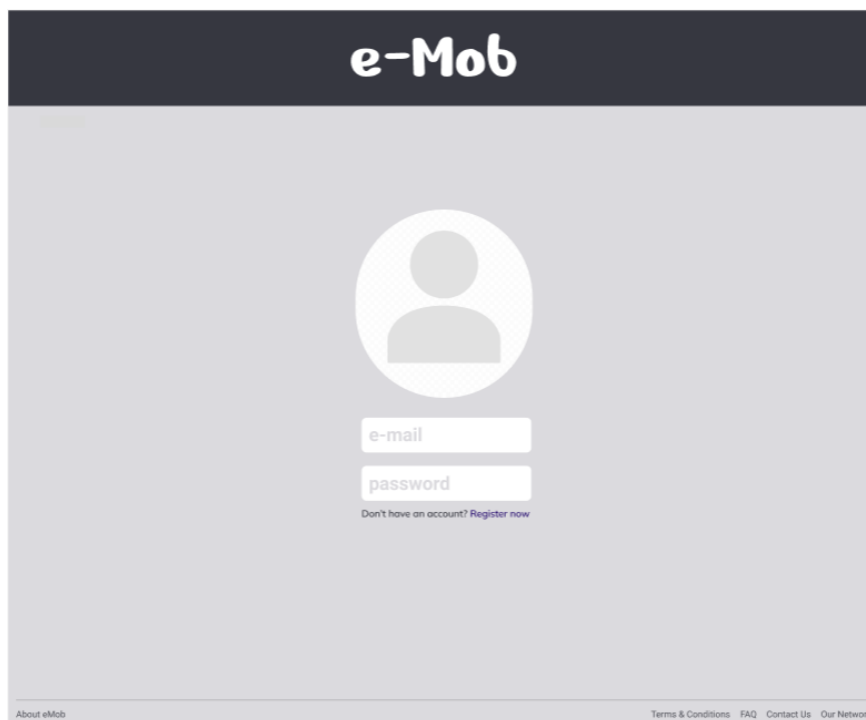


Figure 36: UI for login

Similar to the process for the car owners, the login interface requires the basic data from the user and contains a button for unregistered users. This UI is related to Runtime view 3 from section 2.4.

3.2.2 Register

The screenshot displays the 'e-Mob' web application's registration interface. On the left is a dark sidebar with a menu containing 'Menu', 'Charging Stations', 'Energy Acquisition', 'Pricing', and 'Settings'. The main content area is titled 'Registration form' and includes input fields for 'Name:', 'Password:', and 'IBAN:'. Below these is the 'Charging Station Registration #1' section, which contains a 'location:' field with the value 'piazza Leonardo, 32'. Under 'charging sockets:', there are three entries: '#1' with 'charging type: FAST', '#2' with 'charging type: RAPID', and '#3' with 'charging type: FAST'. Each entry has a corresponding '+ add socket' button. Below the sockets is a 'battery:' section with a red error message '• No battery added' and a '+ add battery' button. At the bottom of the form is a '+ add charging station' button. The footer includes 'About eMob' and links for 'Terms & Conditions', 'FAQ', 'Contact Us', and 'Our Network'.

Figure 37: UI for register

In this submission, the fundamental data from the charging station is required. In which it is possible to add sockets, their types, add batteries and set addresses. This Interface is part of Runtime view 2.

3.2.3 Home Page (Automatic Mode)

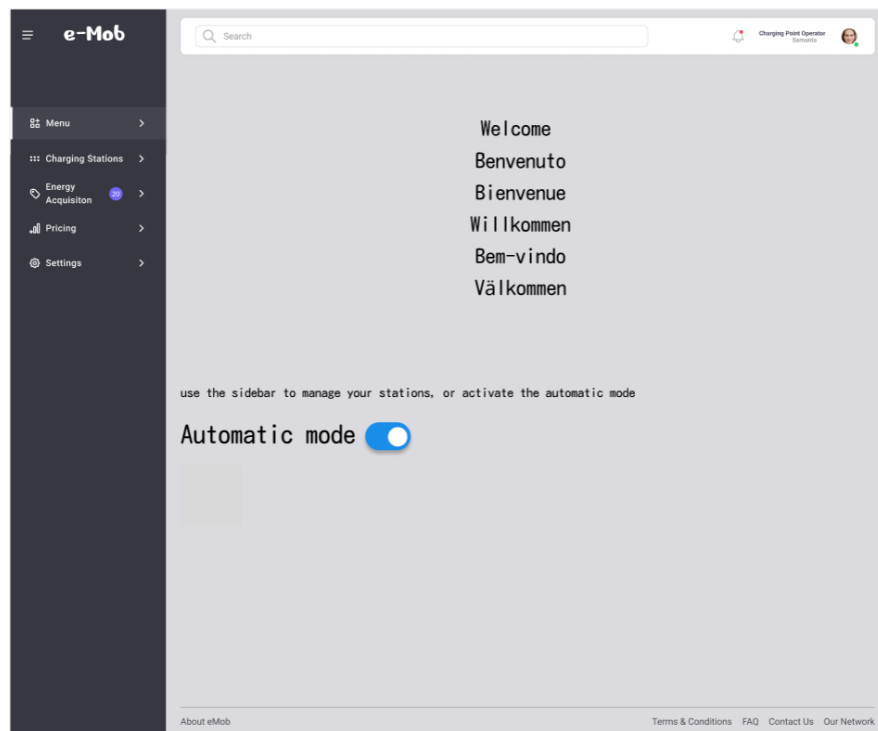


Figure 38: UI for home page

If a user is already logged in, this is the home page seen. To navigate throughout the features, it is necessary to use the left tab. The UI also contains the automatic mode button, that activates the automatic mode of the system (presented in the Runtime view 13).

3.2.4 Charging Station View

Charging Stations List		
Your Charging Stations		
#1 Duomo's Charging Place	2 connected cars	3 sockets available
#2 Enzo's Station Place	4 connected cars	4 sockets available
#3 Charging Station Città Studi	1 connected cars	9 sockets available
#4 ITA-CS	3 connected cars	no sockets available

Figure 39: UI Charging Station table

In the “Charging Station” tab, it is possible to see the charging stations currently controlled by the registered CPO. The table shows the availability data of the stations and sockets. For further information about a specific station, it is necessary to click on it. UI present in view 8.

3.2.5 Specific Charging Station View

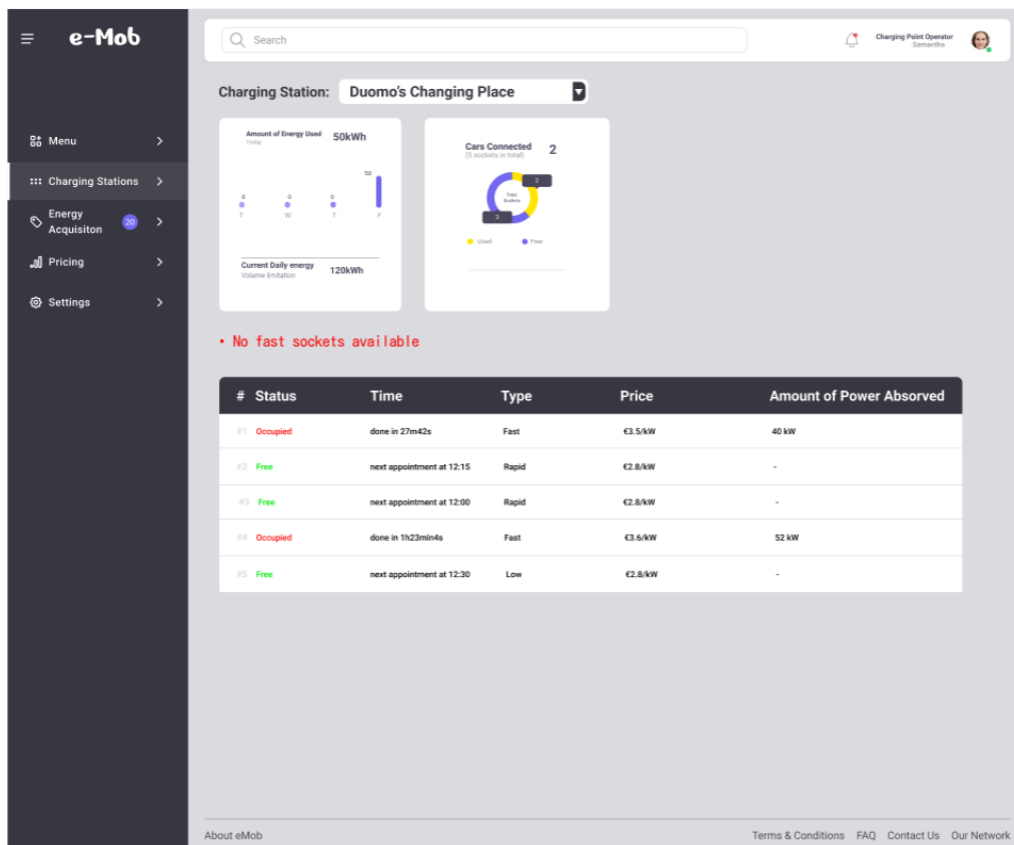


Figure 40: UI for specific station view

After selecting a charging station, the above screen is shown, with specific data about what is currently happening in each charging socket of the station. This continues to be part of Runtime view 8.

3.2.6 Setting Prices

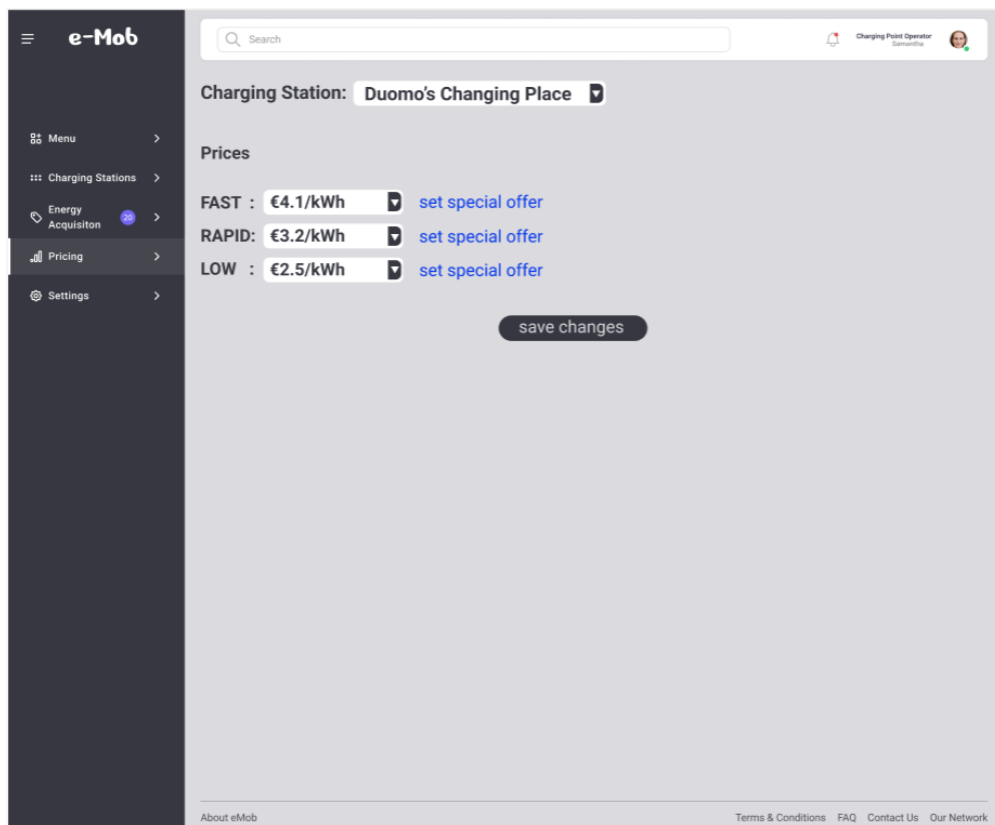


Figure 41: UI for setting charging prices

In the “Pricing” tab, it is possible to select each of the charging stations managed by the CPO and set different prices for each socket type. After completion, it is possible to update the prices by clicking the “save changes” button, describing Runtime view 12 (CPO sets Price for his Charging Stations). Furthermore, the “set special offer” button opens a new interface seen below.

3.2.7 Setting Special Offer

The screenshot shows the e-Mob web application interface. On the left is a dark sidebar with a menu containing: Menu, Charging Stations, Energy Acquisition, Pricing (highlighted), and Settings. The main content area has a light gray background. At the top, there is a search bar and a user profile icon labeled 'Charging Point Operator'. Below the search bar, the 'Charging Station' is set to 'Duomo's Changing Place'. The 'Select Special Price - RAPID' section shows a price of '€4.1/kWh'. The 'Select Timeframe' section shows a date range from '20/12/2022' to '10/01/2023'. A dark 'add offer' button is positioned in the lower right of the main area. The footer contains links for 'About eMob', 'Terms & Conditions', 'FAQ', 'Contact Us', and 'Our Network'.

Figure 42: UI for special offer

This interface allows the CPO to set a special offer in a type of socket and select the period in which this offer will stand. This interface is related to view 11 (CPO sets “Special offer” for his Charging Stations). After clicking “add offer” the offer is created and offered to the car owners.

3.2.8 Buying Energy from DSOs

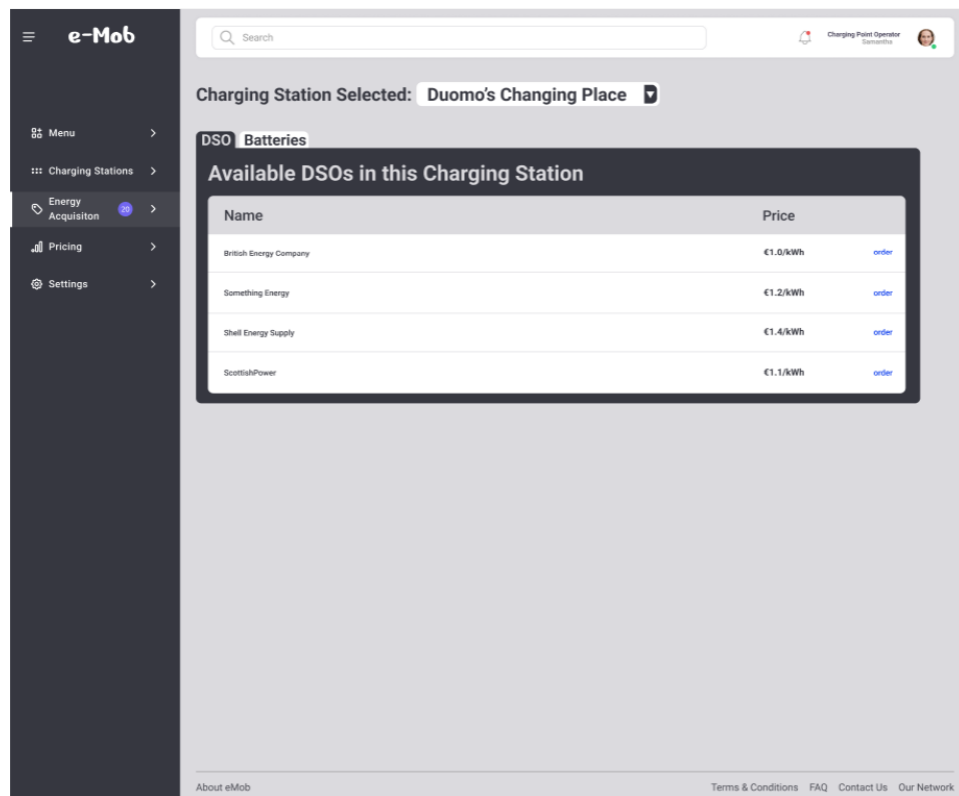


Figure 43: UI for buying energy form DSOs

In the “Energy Acquisition” tab, it is possible to see all energy offerings for each charging station owned. To buy one of the energy offers, the user must select “order” and the payment will be processed by the payment API. This Interface expresses the Runtime view 9 from section 2.

3.2.9 Battery Management View

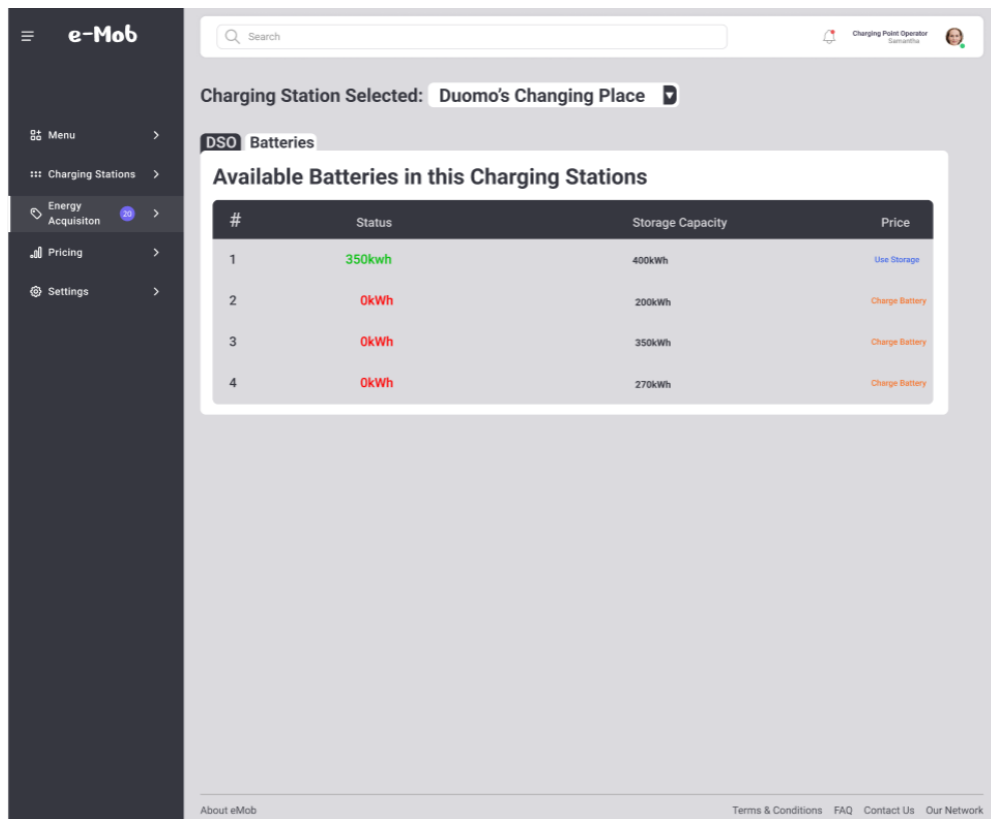


Figure 44: UI for battery ordering and management

In the tab “Batteries” it is possible to visualize the current charging status of every battery available. In order to acquire energy for the battery (view 10) the CPO clicks the option “Use Storage”. If the CPO wants to store energy in another battery (view 14), it is necessary to click on “Charge Battery”.

4 Requirements traceability

This section contains a table explaining what components are required in order to fulfill each of the requirements specified in the RASD. First, a review of the requirements presented in RASD is shown, followed by the list of the components, which were given abbreviations, and finally the mapping in a table format with the components contributing to the achievement of these requirements.

4.1 Review of requirements

eMSP system

Requirement	Description
-------------	-------------

R1	The eMSP system shall allow an unregistered car owner to register an account by entering name, Email, password, his car's battery data, his bank account information and allows access to his phone's locationing service and calendar
R2	The eMSP system shall allow a registered car owner to search for charging stations nearby
R3	The eMSP system allows the registered car owner to login with his registered account by entering his Email address and password
R4	The eMSP system allows the registered car owner to book a charge
R5	The eMSP system allows the registered car owner to start a booked charge
R6	The eMSP system suggests the registered car owner a suitable offer based on his location and calendar information
R7	The eMSP system must be able to notify the car owner of incorrect data during the registration
R8	The eMSP system must be able to notify the car owner of wrong login data during the login
R9	The eMSP system must be able to notify the car owner when his selected timeframe in the booking is not possible
R10	The eMSP system must be able to notify the car owner if there is no charging station nearby when he searches for them
R11	The eMSP system must be able to notify the car owner if the payment procedure during the booking failed
R12	The eMSP system must be able to notify the car owner if he plugged his car into a wrong charging station
R13	The eMSP system must be able to notify the car owner if his car does not charge
R14	The eMSP system must be able to notify the car owner if he forgets to unplug his car
R15	The eMSP system must be able to access the locationing system of the car owner
R16	The eMSP system must be able to access the calendar information of the car owner
R17	The eMSP system must be able to communicate with an API handling the communication with the bank accounts to handle the payment procedure
R18	The eMSP system must be able to communicate with all the CPMS systems of the charging stations nearby
R19	The eMSP system shall allow the car owner to view his active bookings

R20	The eMSP system shall allow the car owner to start a charge
R21	The eMSP system shall notify the car owner when his car is ready to unplug
R22	The eMSP system checks for charging suggestions when the car battery is below 30%
R23	The eMSP system must be able to check the car owner's car battery

CPMS system

Requirement	Description
R24	The CPMS system must be able to communicate with an API showing the DSO information
R25	The CPMS system must be able to communicate with an API handling the communication with the bank accounts to handle the payment procedure
R26	The CPMS system must be able to notify the registered CPO of incorrect data during the registration
R27	The CPMS system must be able to notify the registered CPO of wrong login data during the login
R28	The CPMS system must be able to notify the registered CPO if there are no charging stations attached to his account
R29	The CPMS system must be able to notify the registered CPO if the payment procedure during the energy acquisition failed
R30	The CPMS system must be able to notify the registered CPO if he tries to charge batteries but there aren't any
R31	The CPMS system must be able to notify the registered CPO if he tries to charge batteries but there isn't enough energy
R32	The CPMS system allows the registered CPO to check status information about his charging stations
R33	The CPMS system allows the registered CPO to buy energy from DSO
R34	The CPMS system allows the registered CPO to save energy in batteries at his charging stations
R35	The CPMS system allows the registered CPO to use the energy from the batteries connected to his charging stations to charge connected cars
R36	The CPMS system allows the registered CPO to login with his registered account by entering his Email address and password
R37	The CPMS system shall allow an unregistered CPO to register an

	account by inserting name, Email, password, his car's battery data, his bank account information and relevant data about his charging stations (amount of sockets, charging velocity, whether a battery is connected, location, technical data regarding the charging process itself)
R38	The CPMS system shall start a charge by giving the signal to the charging station to start charging the connected car
R39	The CPMS system shall inform the eMSP system when the car battery is fully charged
R40	The CPMS system shall allow the registered CPO to see a list of the available charging station batteries
R41	The CPMS system shall allow the registered CPO to see a list of the available DSOs
R42	The CPMS system can check if the connected car is the one for which the charge was booked
R43	The CPMS system shall allow the registered CPO to set a special offer
R44	The CPMS system shall allow the registered CPO to set the price for his charging stations
R45	The CPMS system shall allow the registered CPO to activate the CPMS' automatic mode

4.2 List of abbreviations

eMSP system

- **CA** - Client Application
- **RO** - Router
- **AS** - Authentication Service
- **M** - Model
- **DBMS** - DBMS
- **NCSS** - NearbyCharginggStationService
- **PCS** - PayChargeService
- **BCS** - BookChargeService
- **CS** - ChargingService
- **SGS** - SuggestionService
- **CBS** - CarBatteryService
- **PDS** - PhoneDataService
- **PAH** - PaymentAPIHandler
- **CPMSAH** - CPMS_APIHandler
- **CBAH** - CarBattery_APIHandler
- **PDAH** - PhoneData_APIHandler

CPMS system

- **CA** - Client Application
- **RO** - Router
- **AS** - Authentication Service
- **M** - Model
- **CPS** - ChangePriceService
- **AAMS** - ActivateAutomaticModeService
- **BES** - BuyEnergyService
- **SIS** - Internal/External_StatusInformationService
- **UEFBS** - UseEnergyFromBatteriesService
- **CS** - ChargingService
- **PAH** - Payment API Handler
- **CSAH** - ChargingStationAPIHandler
- **DSOAH** - DSOAPIHandler
- **EMSPA** - eMSPAPIHandler
- **DBMS** - DBMS

4.3 Mapping traceability

eMSP system

R	CA	RO	AS	M	DBMS	NCSS	PCS	BCS	CS	SGS	CBS	PDS	PAH	CPMSAH	CBAH	PDAH
R1	✓	✓	✓	✓	✓											
R2	✓	✓	✓			✓								✓		✓
R3	✓	✓	✓	✓	✓											
R4	✓	✓	✓	✓	✓			✓								
R5	✓	✓	✓	✓	✓				✓							
R6	✓	✓	✓	✓	✓					✓		✓		✓	✓	✓
R7	✓	✓	✓	✓	✓											
R8	✓	✓	✓	✓	✓											
R9	✓	✓	✓	✓	✓			✓								
R10	✓	✓	✓			✓								✓		✓
R11	✓	✓	✓	✓	✓		✓	✓					✓			
R12	✓	✓	✓	✓	✓				✓					✓	✓	
R13	✓	✓	✓								✓				✓	
R14	✓	✓	✓	✓	✓				✓					✓	✓	
R15																✓
R16																✓
R17													✓			

R18														✓		
R19	✓	✓	✓	✓	✓			✓								
R20	✓	✓	✓	✓	✓				✓							
R21	✓	✓	✓	✓	✓						✓				✓	
R22	✓	✓	✓			✓				✓				✓	✓	✓
R23											✓				✓	

The component CA and RO contribute to all requirements that involve the Car Owner and AS when it is required login/registration. As the components are described on section 2.2, the other components presented on the table above follow the same logic, for instance NCSS when it is required to know nearby Charging Stations, PCS when it is needed to pay the charge, which usually it is accompanied by the PAH. When the requirement concerns booking a charge, the component BCS is involved and when it concerns the charging service itself CS is involved. The same applies for SGS, CBS and PDS for suggestion, car battery and phone data, respectively. When the component uses information on the database, M and DBMS are involved. And finally, there are the API Handlers CPMSAH, CBAH and PDAH that are involved in requirements related to CPMS, Car Battery and the data from the phone (location and calendar).

CPMS system

R	CA	RO	AS	M	CPS	AAMS	BES	SIS	UEFBS	CS	PAH	CSAH	DSOAH	EMSPA	DBMS
R24													✓		
R25											✓				
R26	✓	✓	✓	✓											✓
R27	✓	✓	✓	✓											✓
R28	✓	✓	✓	✓				✓		✓					✓
R29	✓	✓	✓				✓				✓				
R30	✓	✓	✓	✓				✓	✓	✓		✓			✓
R31	✓	✓	✓	✓				✓	✓	✓		✓			✓
R32	✓	✓	✓	✓				✓				✓			✓
R33	✓	✓	✓				✓				✓		✓		
R34															
R35	✓	✓	✓						✓			✓			
R36	✓	✓	✓	✓											✓
R37	✓	✓	✓	✓											✓
R38										✓		✓			
R39	✓	✓	✓							✓		✓		✓	
R40	✓	✓	✓	✓				✓				✓			✓
R41	✓	✓	✓				✓						✓		

R42	✓	✓	✓	✓				✓				✓			✓
R43	✓	✓	✓	✓	✓										✓
R44	✓	✓	✓	✓	✓										✓
R45	✓	✓	✓	✓		✓									✓

The component CA and RO contribute to all requirements that involve the CPMO and AS when it is required login/registration. When the component uses information on the database, M and DBMS are involved. The requirement that involves the activation of the automatic mode is related to AAMS. Moreover, CPS, BES, SIS, CS and UEFBS are related respectively to the requirements that involve change in prices, buying energy from DSOs, access to internal or external status information, the initialisation and completion of the charging procedure, and the use of energy from batteries in the charging station. And finally, there are the API Handlers PAH, CSAH, EMSPA and DSOAH that are involved in requirements related to payment, charging station, eMSP and DSOs.

5 Implementation, integration and test plan

In this section the plans to follow for the implementation, integration and testing of the systems are described.

5.1 Implementation

The implementation of this system would follow a planned structure, dividing the components into different groups to maintain a clear development path. This classification is heavily based on the component diagram. The system should be developed according to the following five sub-parts, aiming to follow a bottom-up development with respect to the threads.

It is worth mentioning that for the eMSP, we chose a remote data access approach, where the GUI and application logic are part of the client side and the data is part of the server side. Whereas for CPMS, we opted for a remote presentation approach where only the GUI resides on the client side and the application logic and data resides on the server side. The CPMS also has an automatic mode, so the client is not needed in this case, so it is necessary to decouple the GUI from the application logic to allow the CPMS to work independently. But still, both the eMSP and the CPMS can be divided into these 5 subgroups as they share a similar structure on the components diagram presented in section 2.2. This division is made in order to achieve a modularized system, with high cohesion and low coupling.

1. DBMS & Model

This part includes the creation of the data model. Database tables with the specified rows are created together with the Model component.

2. API Handlers

The API Handlers represent the “middleman” between the services and the APIs. So if any change occurs in the APIs, it is simpler to maintain the system, as the API Handlers are modular and contain all interactions directly between the system and the API.

3. Services

The services will be developed w.r.t the component schematic. All services are independent and can be developed in order to follow the threads strategy for integration and testing.

4. Router

There are various libraries and standard solutions for this. Remember that the router must authorize the request before forwarding it to the appropriate service.

5. Client Application

It is the view part of the system where all user interactivity is developed. Due to the modular structure of both eMSP and CPMS, this component should be responsible for the graphical user interface (GUI) and nothing else.

5.2 Integration and Testing

The sequence in which we integrate components will follow the implementation order described in the previous section. So, it will follow an incremental integration, so it occurs while components are released. For instance, as soon as a first version of components X and Y is released, we integrate them and test the integration. Then, if tests pass, we also integrate the first version of Z that has been released in the meanwhile and such new integration is tested. It is worth mentioning that the components should pass all unit tests before integration.

So the strategy adopted for integration and testing will be based on the hierarchical structure of the system, which is a bottom-up strategy, with the strategy using threads, which are portions of several modules that offers a user-visible program feature, with it we can integrate different threads in order to maximize visibility for the user and minimize stubs and drivers, as well as doing incremental releases of the platform.

Below are explanation graphics that show the sequence in which to run the integration tests for more clarity. The graphics only display part of the integrated controllers, but the others operate similarly because they follow the same logics concerning threads and the hierarchy.

5.2.1 DBMS & Model

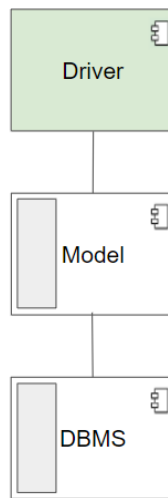


Figure 45: Integration test of Model

5.2.2 API Handler

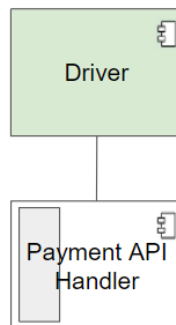


Figure 46: API Handler integration test, in this case, only the PaymentAPI Handler is being shown, the other API Handlers will be dealt with in parallel and the tread process will be done similarly.

5.2.3 Service

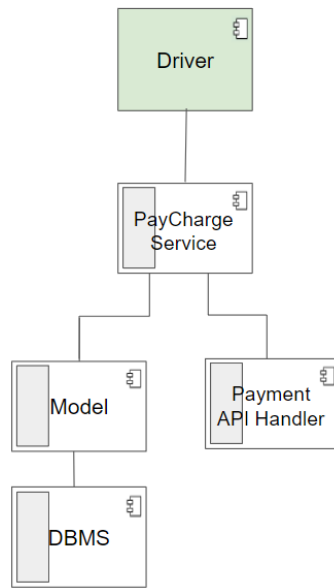


Figure 47: Service Integration test, in the same way that was shown in the API Handler case, the service integration is represented by the PayChargeService, the other services would lie parallel to the presented service above.

5.2.4 Router

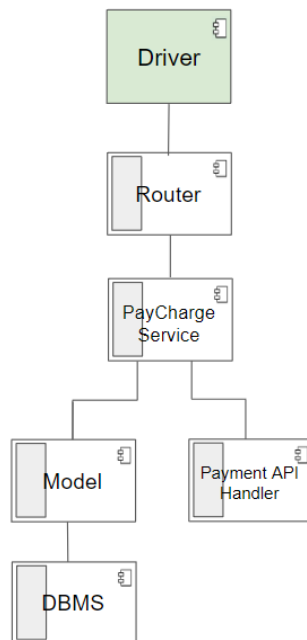


Figure 48: Router integration test

5.2.5 Client Application

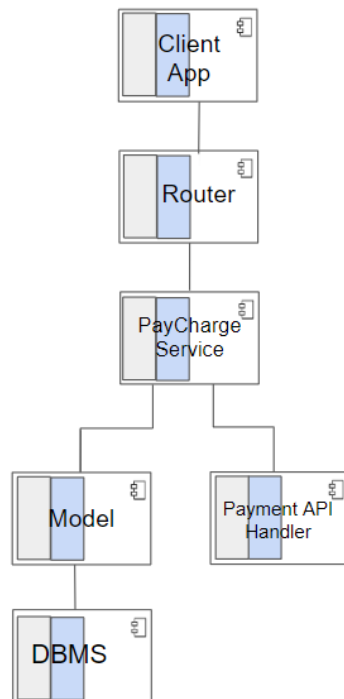


Figure 49: Client Application integration test

As said in the beginning of the section and shown in the diagrams above, the implementation and testing will be done with a bottom-up strategy but counting with a threaded development, so that external visibility of the process is not limited as well as making it more clear the feature relation between the components.

6 Effort spent

1. Azank

Task	Time spent
Introduction	2 h
Architectural design	3 h
User Interface Design	15 h
Requirements Traceability	2 h

Implementation, Integration and Test Plan	10 h
Total	32 h

2. Bagni

Task	Time spent
Introduction	3h
Architectural design	9h
User Interface Design	4h
Requirements Traceability	11h
Implementation, Integration and Test Plan	10h
Total	37 h

3. Wolff

Task	Time spent
Introduction	0 h
Architectural design	10 h
User Interface Design	20 h
Requirements Traceability	4 h
Implementation, Integration and Test Plan	4 h
Total	38 h

7 References

1. Len Bass, Paul Clements & Rick Kazman, Software Architecture in Practice, Addison-Wesley, 1998.
2. Frank Buschmann, Régine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stahl, Pattern-Oriented Software Architecture - A System of Patterns, Wiley and Sons, 1996.
3. Christine Hofmeister, Robert Nord, Dilip Soni, Applied Software Architecture, Addison-Wesley 1999.
4. Eric Gamma, John Vlissides, Richard Helm, Ralph Johnson, Design Patterns, Addison-Wesley 1995.