

POLITECNICO DI MILANO

SOFTWARE ENGINEERING 2

Design Document (DD)

Author:
Claudia BAZ
(10916443)

Author:
Pablo LLORENTE (10904511)

Author:
David QUINTERO (10832704)

Lecturer:
Prof. Matteo CAMILLI

October 21, 2024



POLITECNICO
MILANO 1863

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Scope	3
1.3	Definitions, acronyms, abbreviations	5
1.3.1	Definitions	5
1.3.2	Acronyms	5
1.3.3	Abbreviations	5
1.4	Revision history	6
1.5	Document structure	6
2	Assumptions, dependencies and constraints	7
3	Architectural design	7
3.1	Overview: System architecture	7
3.2	Architectural styles: Application architecture	9
3.2.1	Architectural patterns	9
3.3	Component view	9
3.4	Deployment view	11
3.5	Component interfaces and communication	13
3.5.1	Internal communication	13
3.5.2	External communication	14
3.6	Runtime view	15
3.6.1	eMSP: User's Registration, Login, and Authorization	16
3.6.2	eMSP: External Services Pairing	18
3.6.3	eMSP and CPMS: User's CS Search, Selection and Booking	21
3.6.4	CPMS: Power Pricing and SOs	25
3.6.5	CPMS: Decide CS supply type (internal, external or mixed)	27
3.6.6	eMSP and CPMS: Smart Feature - SO User's Notification	29
3.6.7	eMSP: Users Modifying and Deleting Bookings	31
3.6.8	eMSP: Bookings Payment	33
3.6.9	eMSP and CPMS: Innit Vehicle Charging	34
3.6.10	eMSP and CPMS: Monitoring Vehicle Battery and End of Charging	36
3.6.11	eMSP: Smart Feature - Suggest Charging Schedule	39
3.7	Other design decisions	40
4	User Interface Design	40
4.1	Log In	41
4.2	Register	41
4.3	Main Menu	42
4.4	My Bookings	42
4.5	Profile	43
4.6	Booking Information	43
4.7	Booking map	44
4.8	Booking socket type	44
4.9	Booking socket	45

4.10	Confirm Booking	45
4.11	Monitor Battery Status	46
4.12	CPMS view	46
5	Requirements traceability	47
5.1	Non Functional Requirements	48
5.2	Functional Requirements	49
6	Implementation, integration and test plan	51
6.1	Implementation plan	51
6.2	Integration and test plan	52
7	Reference Documents	55
8	Effort spent	56

1 Introduction

1.1 Purpose

In the last years, Governments' efforts in achieving their emission goals have resulted in the promotion of some incentive schemes to increase electric vehicle (EVs) sales, as they are a crucial means to meet environmental goals. The significant reduction in the prices of battery cells and the 5G rollouts for charging management services are also driving the growth of this market.

However, some factors, such as the lack of infrastructure (charging stations), the still high battery costs, and the multiple vendors with still no standardized charging and payment systems, affect the owners of EVs. As a result, the charging process tends to be time-consuming, forcing drivers to drive around the city searching for a suitable and available socket to charge their vehicles, interfering with their daily schedule.

Therefore, building a system that brings the user closer to the charging stations will improve enormously the user experience. The new system, e-Mall, is composed of two subsystems, e-Mobility Service Provider (eMSP), and Charge Point Management Syst(CPMS). The first faces the users (clients), and the latter faces the charging point operators (vendors), enabling the charging service market. The e-Mall system will allow users to book a charging service, monitor the battery of their vehicles, receive proactive suggestions about where and when to charge, and will ensure the energy supply chain, all from remote.

This document will further expand on the design of the e-Mall system and is a continuation of the Requirements Analysis and Specifications Document (RASD).

1.2 Scope

The domain of the e-Mall system encompasses all the agents directly involved in the supply and consumption of energy for EVs. The system is composed of two sub-systems, the eMSP and the CPMS:

- The eMSP has interfaces with the User (vehicle owner), the Charge Point Management System (CPMS), the third-party Payment system, the Vehicle itself, and the Calendars of the user. This sub-system allows the user to manage bookings of charging services, monitor the battery status of his/her vehicle, and receive suggestions about the charging schedules to follow based on the user's habits. Thus, making the eMSP a smart system.
- On the other hand, the CPMS has interfaces with the Charging Point Operator (CPO) which we treat as another human agent, the Distribution System Operator (DSO) or energy vendors, the charging stations infrastructure (CSs infra), and the eMSP. Among their functions are to provide automatically the information needed by the eMSP for managing the lifecycle of bookings, ensure the energy supply chain and, enable the charging service execution. Besides that, the CPMS let the CPO to take manually some decisions such as setting the price of charging services, deciding the publishing and creation of special offers, and acquiring energy from the DSOs.

Furthermore, the e-Mall system interacts with the following actors:

- CS Infrastructure: The CS Infrastructure is defined by the set of sockets and batteries in the same location that belongs to a unique CPO. They are managed by the CPMS.
- Human agents
 - User: Schedule bookings.
 - CPO: As it is the operator he decides all about the management of CSs, price of charging services, special offers, and the acquisition of energy.
- Third-party integrations
 - DSO: Energy providers. They sell the energy to the CPOs through the CPMS and provide the estimation of the energy price up to 7 days in advance.
 - Payment system: System used to pay for the charging service.
 - Calendar: App where the user schedules his/her events.
 - Vehicle

Multiplicities and relationships between actors and subsystems

- Each User will have one instance of an eMSP that can pair, if existent, his/her multiple vehicles and calendars.
- There are multiple CPOs and each of them can have only one technological platform CPMS to centrally manage its multiple CSs.
- Each CPMS can interact with multiple eMSPs when broadcasting special offers and on the opposite, each eMSP can interact with multiple CPMSs when searching for sockets available.

Summary of main architectural style/choices

The eMSP and the CPMS subsystems will have both a hybrid architecture, a mix of an N-Tier Architecture—a type of multilayered Client-Server Architecture with a separation of Presentation Tier, Application Tier, and Data Tier both logical and physical—and a Service-Oriented Architecture according to business activities.

As a result, the system will be composed of 4 different applications: 2 front-ends (clients) and 2 back-ends (servers). Each of them with its own layered architectural pattern. The MVC pattern was chosen for the “client-side” or Presentation Tier, and the Controller-Service-Repository pattern was chosen for the “server-side”.

All these architecture choices were taken to favor the separation of concerns, to ease the software development lifecycle, and to increase the security of the applications.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

Name	Description
Charging Station	Local where vehicles park to charge their batteries. They collect a group of socket types.
Socket	Physical connection/interface where their vehicles connect to charge their battery.
Socket Type	Speed of charge, there are only 3 types: slow/fast/rapid. This is the only difference between sockets.

Table 1: Definitions

1.3.2 Acronyms

Acronym	Name	Description
DSO	Distribution System Operators	Energy vendors
CPMS	Charge Point Management System	CPO management systems. Control their power supply and list their data to external services
CPO	Charging Point Operators	Company or human operator that provides the charging service and is able to acquire energy from DSOs.
eMSP	e-Mobility Service Provider	User interface that provides the interaction with the CPOs through their CPMS
CS	Charging Station	All the infrastructure that is located in the facility where the User charges his/her Vehicle, such as screens to interact with the client, batteries to store energy, and sockets.
SO	Special Offer	offers provide by the DSO or the CPO of the CS. Also, it can be associated to low prices detected by the CPMS.

Table 2: Acronyms

1.3.3 Abbreviations

Abbreviation	Description
RASD	Requirements Analysis and Specification Document
API	Application Programming Interface
DX	Domain assumption number X
RX	Requirement number X
SD	Sequence Diagram
UC	UseCase

Table 3: Abbreviations

1.4 Revision history

- DD 1st version - January 2023

1.5 Document structure

This document contains eight sections divided in the following manner:

- In the first section, it is described the problem background and the scope of the e-Mall system where some details about the architecture used are given. Besides that, it contains some definitions of key terms, acronyms, and abbreviations used throughout the document.
- The second section contains the assumptions, dependencies, and domain assumptions that we consider while designing the e-Mall system. Originally from the RASD document, they were included here in order to ease the reading of the document.
- The third section is the core of this document. It focuses on the architectural styles and patterns that the e-Mall system and its sub-systems (eMSP and CPMS) follow and how this decision impacts both the implementation in software (SW) and hardware (HW). Consequently, each sub-system was divided into components and their interfaces. After this division, the runtime view of the system shows how the different components interact for the realization of the use cases [4]. Furthermore, the deployment view of the system is depicted, i.e. how the system would be implemented in physical hardware, matching with the architecture style chosen.
- The fourth section encompasses the user interface design where different views of the different sub-systems are shown. We focus more on the eMSP interface since this application is intended for public use, whereas the CPMS does not. This latter is intended for the internal use of the CPO.
- In the fifth section, the requirements traceability matrix is shown, where each of the components described in section 3.3 is mapped to the requirements of the system. For the same reason that brings here section 2, the functional and non-functional requirements are shown.
- In the sixth section, we described the implementation, integration, and test plan to follow.
- Finally, we have both sections, the effort spent —the responsibilities that each team member performed are mentioned and the time it took to do them—, and the references used for the elaboration of this document.

2 Assumptions, dependencies and constraints

The design of the whole system is based on the follow assumptions:

Identifier	Description
D1	There is always 100% of compatibility between the vehicle hardware and the device where the eMSP is running
D2	The charging stations physical sockets have an universal adaptor for all kinds of vehicles. Meaning, the only differences between types of sockets is the charging speed and all cars can physically connect to all sockets.
D3	When a socket is not being used it is consider always available assuming it wont have any physical defect or missfunction
D4	All users leave the station after the battery charge has reach the expected percentage
D3	When a socket is not being used it is consider always available assuming it wont have any physical defect or missfunction.
D5	No user disconnects the vehicle from the socket before the expected time
D6	All users arrive an pay on time their booking
D7	The interaction between the various providers (eMSPs, CPOs, and DSOs) occurs through uniform APIs *
D8	All the eMSPs detect automatically the CPMSs available and when one disconnects
D9	The energy acquired from the DSO is available immediately at the charging station
D10	The estimations of energy and price of charging are very exact, they don't fluctuate with time so the user is not charged twice
D11	The DSO can provide an estimation of the energy price up to 7 days in advance
D12	The CS batteries have infinite capacity, they can buy as much energy as they can afford, with no restriction in storage capacity.
D13	All users configure their app properly, pairing at least one calendar and one vehicle
D14	All vehicles manufacturers provide a Cloud Service where the vehicle posts its updated battery status

Table 4: Domain Assumptions

3 Architectural design

3.1 Overview: System architecture

The architecture of e-Mall follows a hybrid architecture between **N-tier Architecture** (a multilayered Client-Server Architecture) and a **Service-Oriented Architecture** for each of its subsystems (eMSP and CPMS).

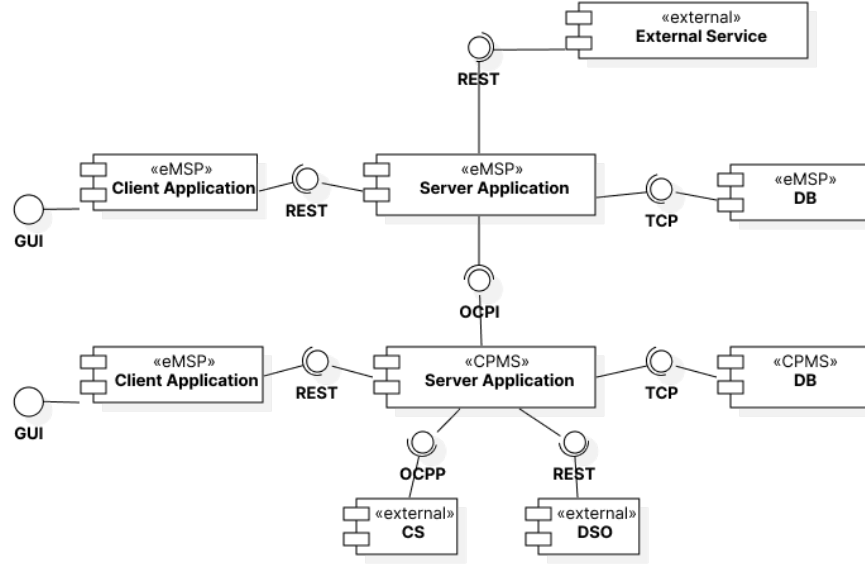


Figure 1: High-level overview of architectural components

There are two sub-systems will have their own Client Application, known as the front-end, for the presentation layer; their own exposed API or back-end, which contains all the HTTP request handling, business logic, and data management; and finally its own data tier physically and logically separated. This can also be seen in Figure 4.

The use of an N-tier Architecture is heavily influenced by the business need of allowing multi-user updates through a Graphical User Interface (GUI) to a shared database (DB), and the security and ease of control management that its characteristic centralization provides. The separation of concerns between the presentation tier, the application tier, and the data tier facilitates the development process and provides that security. A brief description of every tier is presented:

- **Presentation:** The Presentation Tier is in charge of the interaction with the user. It receives the user's requests and forwards them to the Application Tier where they are processed and a response is retrieved.
- **Application:** aka Business logic Tier, here the request processing takes place. The user input is transformed according to his/her request and this normally involves accessing the Data Tier. It is the middleman between the Presentation and the Data Tier. Furthermore, it is in charge of communicating with external services.
- **Data:** aka Persistence Tier. This Tier is mainly in charge of containing the databases and tables that the system needs to operate.

These layers above are not explicitly declared in the component division. However, they guide the general architecture and division of the system.

3.2 Architectural styles: Application architecture

Both the front-end (Client) and the back-end (Server) of each sub-system use a layered architecture for each domain service (inspired by a business activity) to provide a better structure of the code. The “separation of concerns” provides for a better division of the work and improved maintenance.

On the one hand, the separation of the user input, the operations to do, and the actual design play an important role on the Client. Conversely, on the Server, the separation of responsibilities involves separating the HTTP request handling, the core business logic, and the mapping of Data Transfer Objects (DTOs) to entities in the database.

3.2.1 Architectural patterns

Two different architectural patterns inspired by a layered architecture are followed for both the eMSP and the CPMS in the Client and in the Server side:

- **Client side:** The **Model-View-Controller pattern** is used for the presentation.
 - **Model:** It is the central part of the pattern. It manages data and business logic. It defines the data the app should contain.
 - **View:** It handles the layout and visual design of the app.
 - **Controller:** It transforms the user’s input into commands, which it routes to either the Model or the View.

In simple terms, the user sees the View and interacts with the Controller, which in turn manipulates the Model to update the View according to the user input. The Controller can update the View directly also and perform some validation on the user’s input before routing the commands to the Model.

- **Server side:** The **Controller-Service-Repository pattern** is applied.
 - **Controller:** It is the middleman between the input and business logic.
 - **Service:** Here the business logic is implemented, i.e. the transformation of the input according to the business rules, calling external services or the repository. It is the only agent that should interact with the repository.
 - **Repository:** It is a data mapping/access layer. It is in charge of transforming the DTOs into entities in the database. It is an abstraction of database access.

3.3 Component view

As mentioned, e-Mall also uses a Service-Oriented Architecture, that is to provide a better separation of concerns during development, so the business logic of repeatable business activities is decoupled and can be developed in parallel. Some domain services inspired by business activities are the following:

- **eMSP:**
 - Auth: Authentication and authorization handling.

- User: All processes related to User’s data such as personal information, calendars, and vehicles paired.
- Booking: Booking life cycle management.
- Calendar: Manage connection with external calendar services.
- Vehicle: Manage connection with external vehicle services.

- **CPMS:**

- Charging Stations (CS): All the activities related to the interaction with the physical infrastructure of the charging station such as checking the availability of sockets and information about its batteries, etc.
- Charging Sessions (CSessions): Stores the information of each session. In this context, a Session is how the CPMS refers to a charging session associated with a Booking from the eMSP.
- Management: Related to processes that can be done manually by a CPO such as creating Special Offers (SO) and handling the acquisition of Energy.

With that said, the components division takes into account the domain and the separation of concerns seen in the previous section. The CPMS and eMSO components and its internal dependencies are the next:

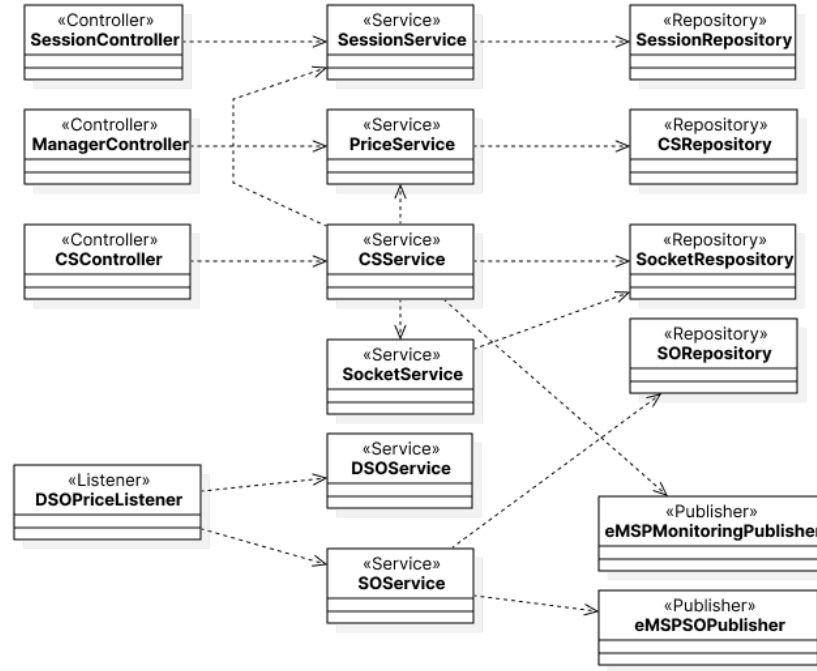


Figure 2: CPMS internal components and dependencies

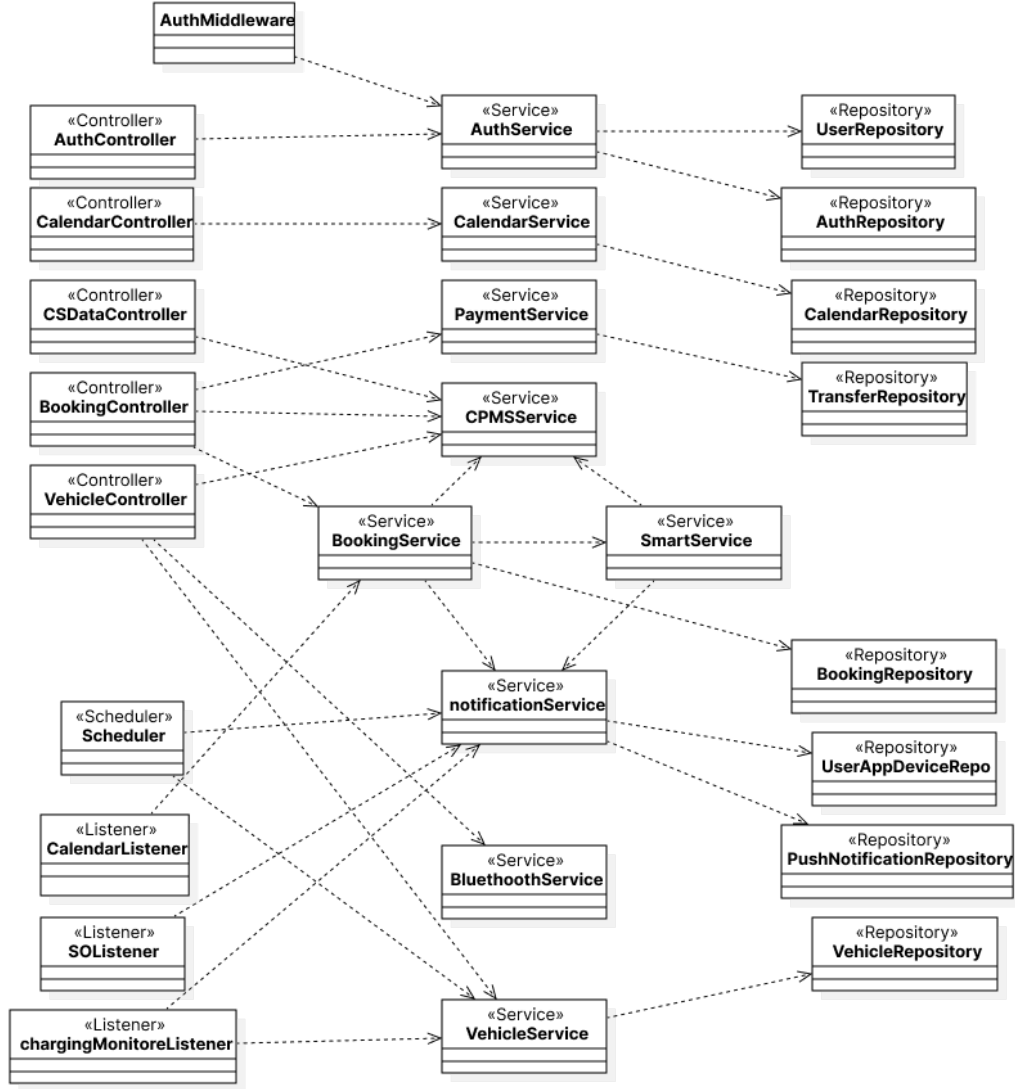


Figure 3: eMSP internal components and dependencies

3.4 Deployment view

The deployment diagram is useful to represent the HW needed for the system, the SW components that run on top of that HW or even other SW components, and the middleware that connects the different HW components.

The e-Mail system consists of two different sub-systems: the eMSP and the CPMS. For the sake of uniformity, we designed the deployment for each one of these following a similar logical topology.

We will explain the general topology and expand in detail on the particularities of each sub-system in the following.

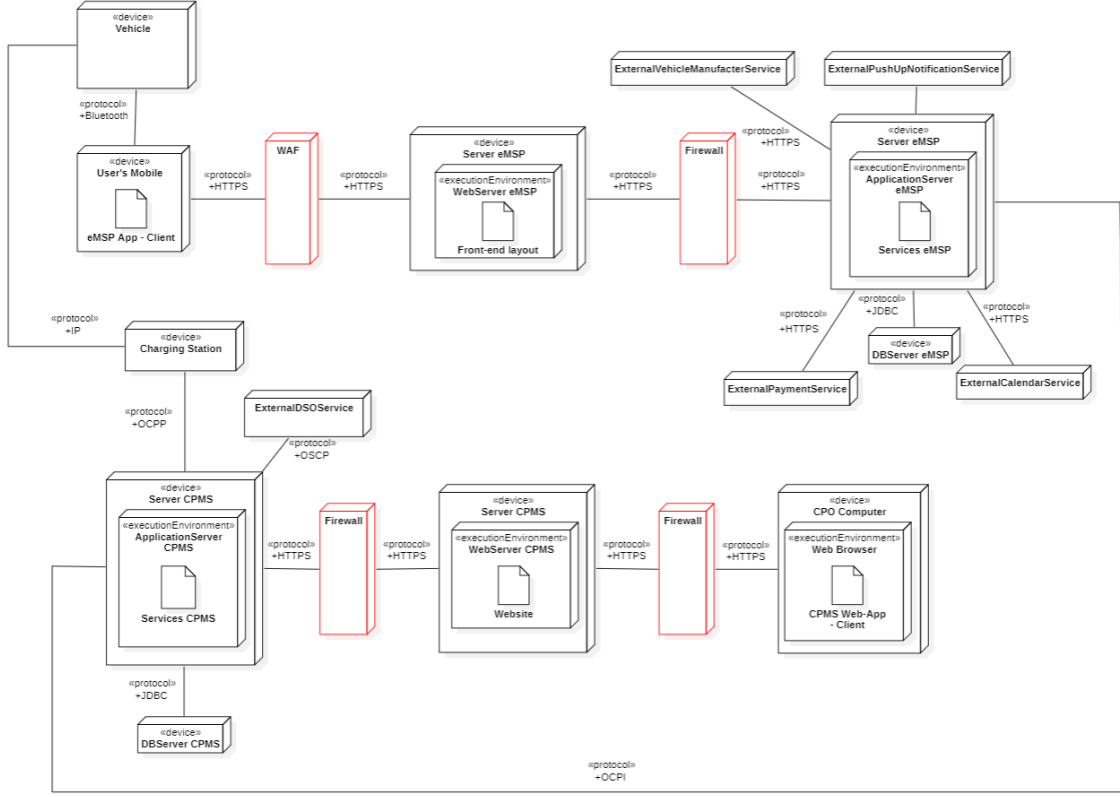


Figure 4: Deployment diagram

The topology exhibits an N-tier Architecture (multilayered client-server architecture), where the user can access the eMSP or CPMS apps either through a Mobile phone or one of a set of selected computers located in the CPO facility, respectively. Then, the HTTP traffic is handled by the Web Server that forwards the request to the Application Server. Once there, the request is served by this server since it stores the business logic and provides access to the data layer. A Demilitarized Zone (DMZ) Configuration secures the AppServer. Furthermore, the WebServer and ApplicationServer are used in tandem to speed up the application since the WebServer can be used for caching previous queries. Each sub-system is connected to the external systems that it requires mostly through their APIs, thus, providing the full functionality of the e-Mail system.

- **Clients:**

- eMSP app which can run on mobile phones or the screen of the Vehicle for the eMSP User.

- CPMS portal which runs only on some specific computers on-premises in the CPO Facility for security concerns.
- **DMZ:** Firewalls are used to protect the devices from outside unauthorized access. In the DMZ configuration, they are placed before and after the web servers, so the App server and the data layer have a double degree of protection. The web server and the application server are deployed in different physical servers to increase security.
 - **eMSP:** We put a Web Application Firewall (WAF) at the edge of the eMSP network since it is a reverse proxy that hides the IP address of the WebServer and provides load balancing and firewall features. Increasing in that way the security of the system. The WAF is justified for the eMSP since it is public to the Internet while the CPMS doesn't. Thus, this latter is good with a Firewall.
- **WebServer:** It is used to take care of the static portion of serving up a website, it displays the site content. They can handle caching and simple HTTP requests.
- **ApplicationServer:** It is used to serve the dynamic content by processing requests according to the business logic, the interaction between user and what is displayed. It consists of the containers with the different services that belong to the App. Some examples of services are User, Vehicle, Booking for the eMSP; and CS and CSession for the CPMS.
- **DatabaseServer:** Physical server where the database of the corresponding sub-system is hosted. It is planned to have one extra database as a backup in case of fail-over, increasing in that way the redundancy of the system.
- **ExternalSystems:** In the eMSP, we can communicate with external systems such as the PushNotificationService, the VehicleManufacturerService, and the CalendarService through REST APIs but we can connect to the Vehicle in two different ways, first by Bluetooth and then through the HTTP protocol to the VehicleManufacturerService. On the other side, the CPMS requires to use some specific protocols to connect with other devices:
 - **Open Charge Point Protocol (OCPP):** For CPMS-CS communication keeping the CPMS updated about events in the CS.
 - **Open Charge Point Interface (OCPI):** For eMSP-CPMS allowing to the driver roaming between different charge points operators. We assume that the connection between eMSPs-CPMSs backends is done through a VPN.
 - **Open Smart Charging Protocol (OSCP):** For CPMS-DSO communication, to exchange information about the energy and its cost.

For the rest of communications, the de-facto secure HTTPS protocol is used but for the DBs, on which JDBC is used.

3.5 Component interfaces and communication

3.5.1 Internal communication

Internal components are those considered inside a subsystem. In this section, it is considered only the interaction between not border components (they do not communicate with external systems). These components are able to interact with each other via interfaces.

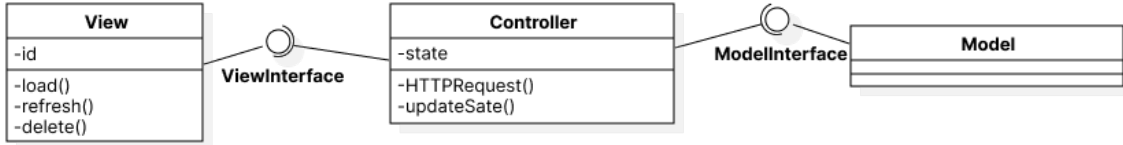


Figure 5: Front-end application not border components interfaces

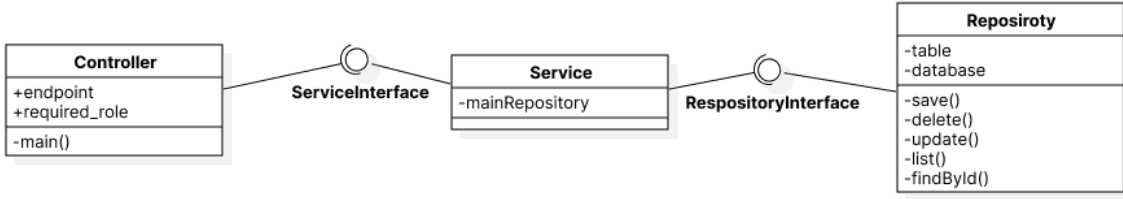


Figure 6: Back-end application not border components interfaces

The figures above present the general components' functions and attributes. All of the components will inherit from these general classes.

3.5.2 External communication

This section will define the interaction between components of different sub-systems and those external.

As explained in Section 3.1, there are two main sub-systems: the eMSP and the CPMS. They can interact in the following ways:

- **Event-driven communication**

For asynchronous calls, the CPMS will act as the publisher while the eMSP as the listener. To keep the scalability, a Listener and a Publisher can have only one topic. Meaning, one eMSP Listener can be subscribed to more than one CPMS Publisher, but all of them must send events of one topic only. This will avoid in the future nested bugs.

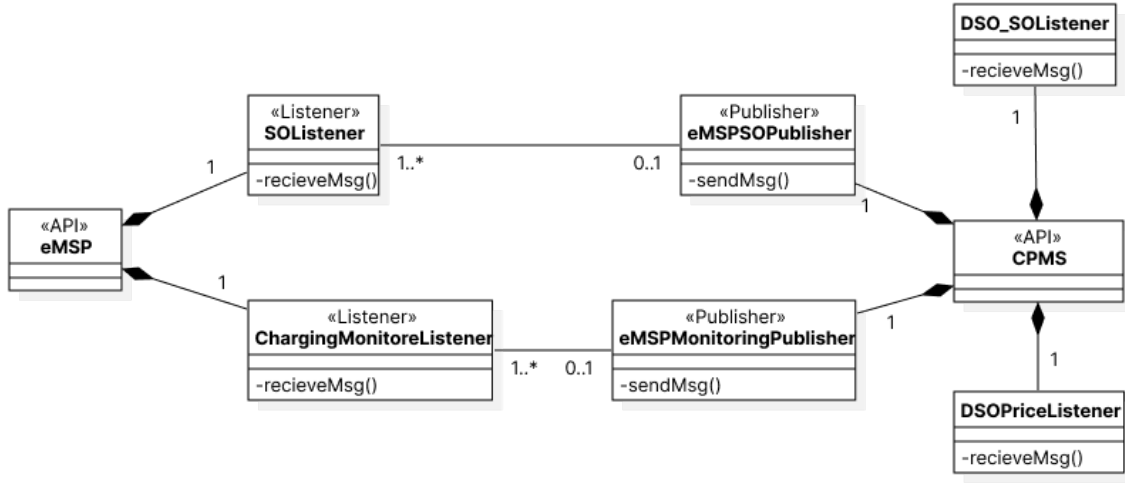


Figure 7: Event-driven communication between the eMSP and the CPMS

- **API REST**

Some of the service components are designed to communicate with external services:

- CalendarService (eMSP)
- VehicleService (eMSP)
- NotificationService (eMSP)
- PaymentService (eMSP)
- CSService (CPMS)

All of them work as integrators and provide an interface to the system's internal components. Meaning, different external services can be added and deleted without modifying how the rest of the components call their functions. For example, all the vehicle manufacturers' cloud services must be called by the same component the *VehicleService*.

This **modularity** will help to keep the responsibility division and keep the potential scalability of the application.

Besides, the CPMS will act as REST API from which the eMSP can consume information. Therefore, for the eMSP, the CPMS is perceived as an external service.

3.6 Runtime view

This section aims to describe the interaction between components at runtime. This means each SD represents the functionality of the system. Most of them are correlated to a UC listed in the previous RASD document [3]. Moreover, one SD can be associated with different UC due to the nested features.

In that manner, the SDs help to understand the systems architecture, its internal and external dependencies, and the functionalities of the components.

Each section can contain both eMSP or CPMS components, or just one of the systems. The system to which it belongs is defined above its name. In this case, the component can be from the eMSP (client or server), the CPMS (client or server), or an external one (DSO, CS, or other services). The external components are generalizations of a third-party system just represented to simulate their interactions with the internal ones.

3.6.1 eMSP: User's Registration, Login, and Authorization

The identification and authorization of the eMSP users is centralized in the *AuthMiddleware* component. In that manner, a user must be registered first to be recognized by the system. Afterward, the user can log in using his/her credentials. Due to the centralization of these nested functions in one single component, different authorization techniques can be used. However, the system was designed thinking of the JWT Token [2] pattern, due to its popularity.

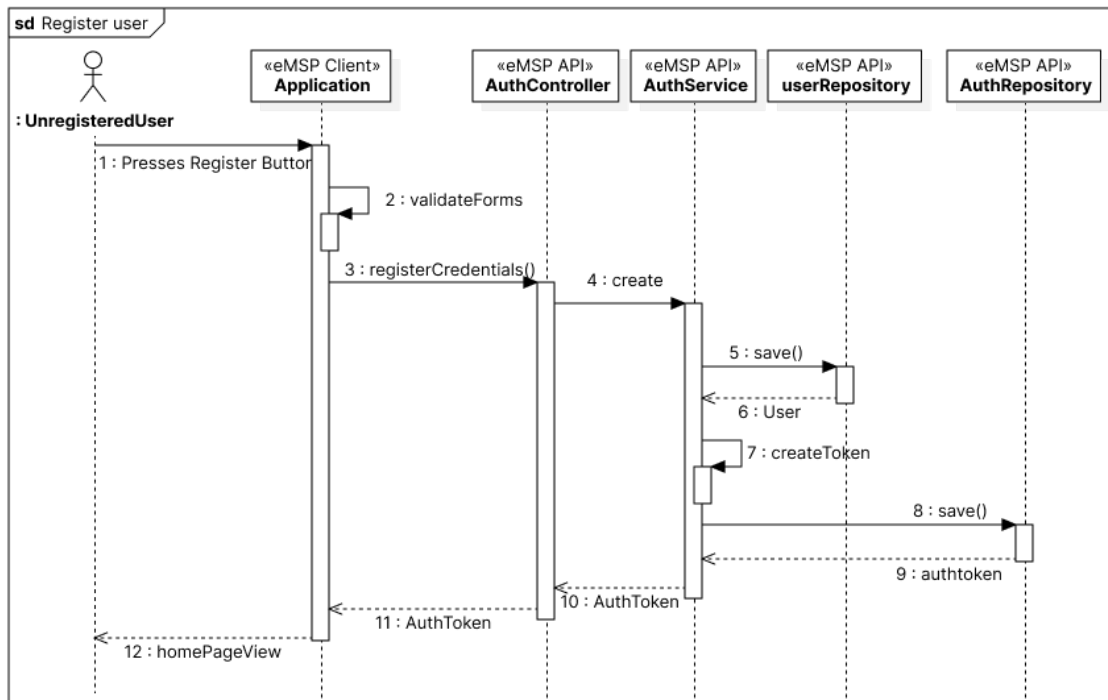


Figure 8: eMSP user registration sequence

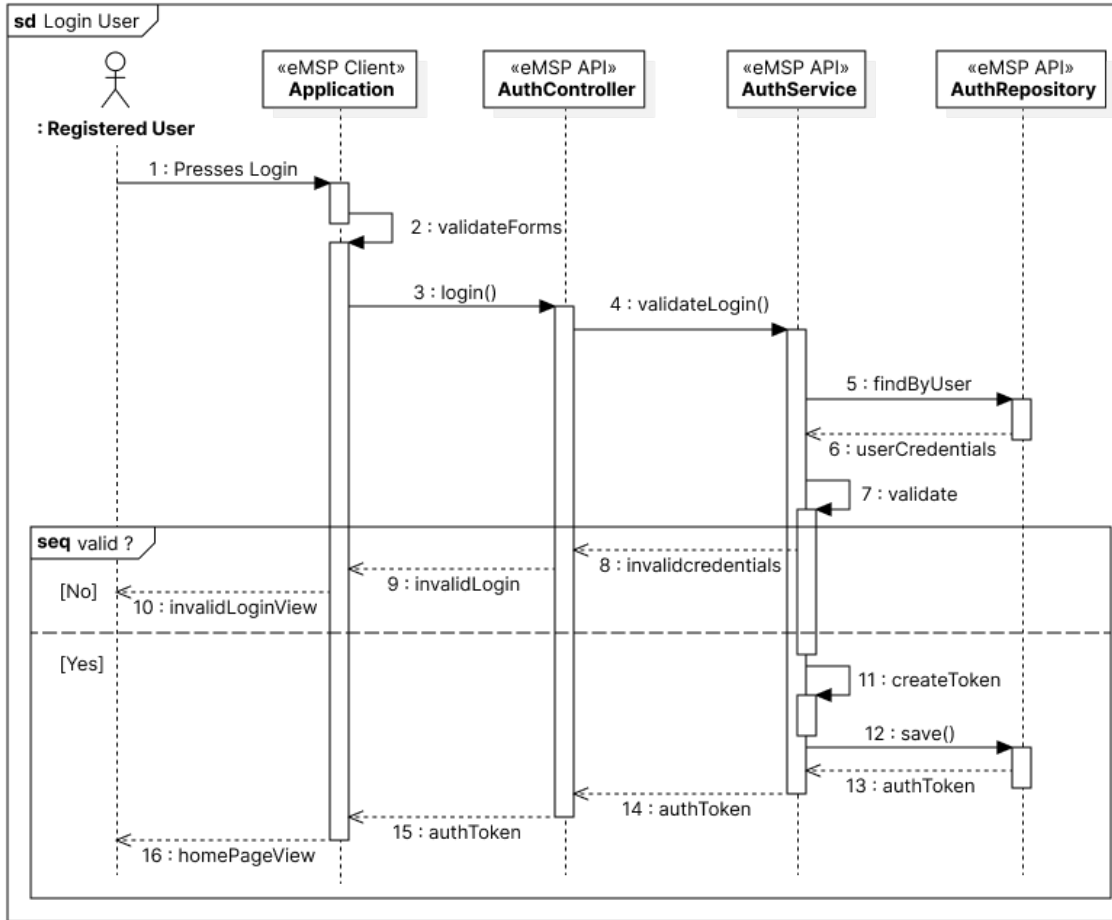


Figure 9: eMSP users login sequence

The next SD describes the previous authorization process for all non-public endpoint access. Meaning, the *General Controller* represents a *black-box* of any feature requested to the system.

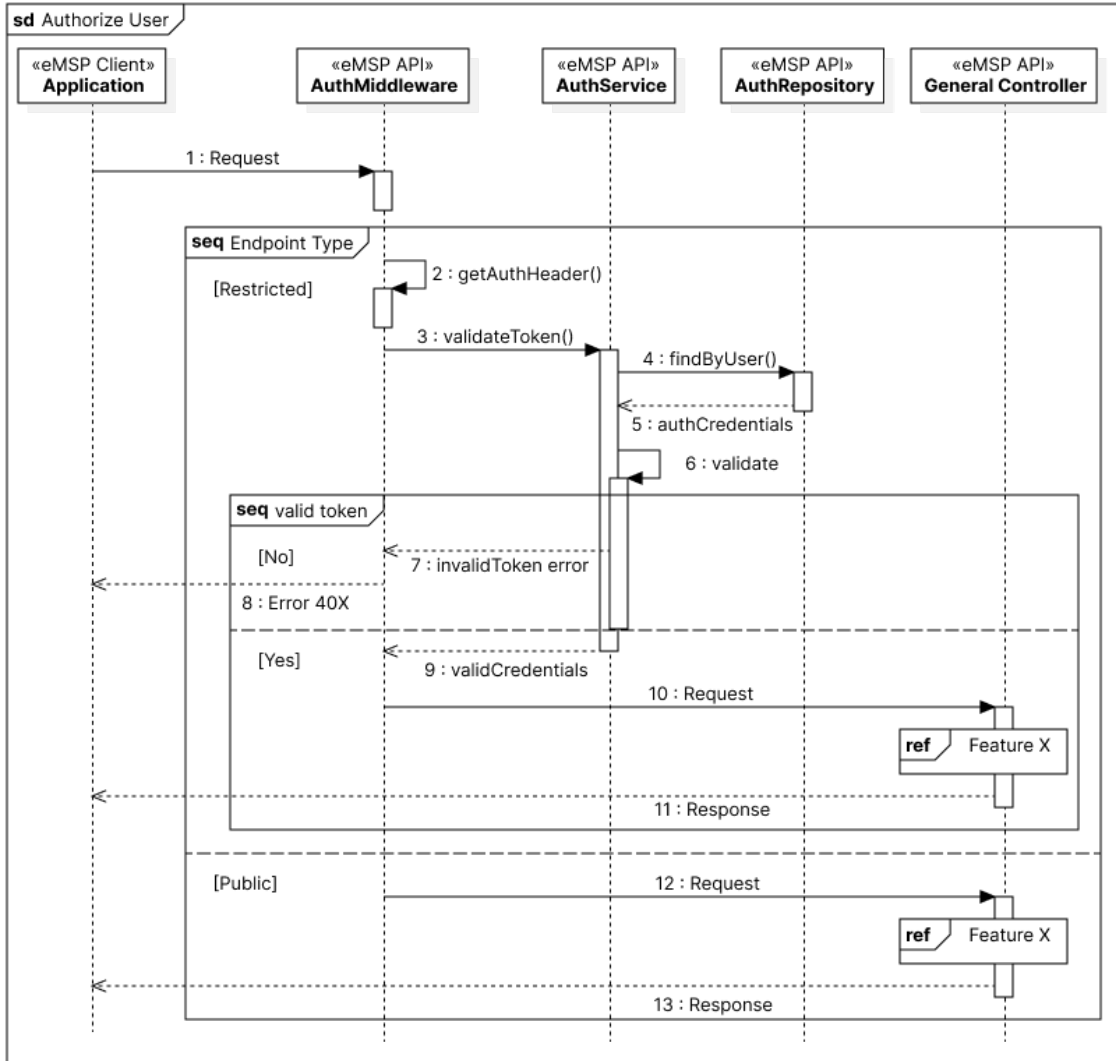


Figure 10: eMSP user authorization sequence

Therefore, the only public endpoints of the eMSP API are the Register and Login features from the *AuthController*. All the rest, private endpoints, must be filtered first by the *AuthMiddleware* component.

3.6.2 eMSP: External Services Pairing

The eMSP has to consume external APIs to be able to connect to the users' device (via push-up notifications), vehicle (via its cloud service), and personal calendar. The last feature relies on D14, see Table 2.

The eMSP pairing components are designed to consume different external APIs. It's vital to keep its modularity in favor of the eMSP applications scalability. With time, different external APIs (of vehicle manufacturers and personal calendars) can be added and deleted to improve the user experience. In that manner, the next SD draws the vehicle pairing feature with the user eMSP application:

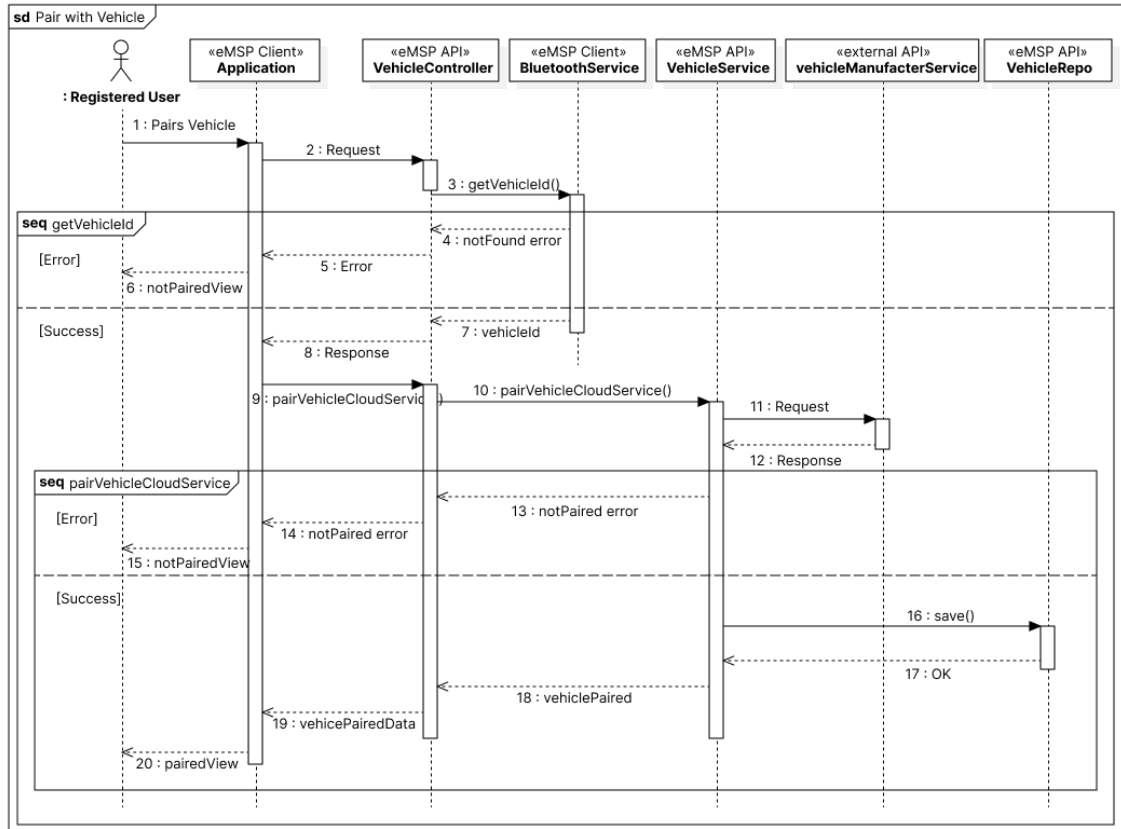


Figure 11: Pairing between the eMSP client application and the user's vehicle

To access the vehicle cloud service, to load its battery status, the vehicle must be identified first. The *BluetoothService* connects to the car to get its key identification or VIM (Vehicle Interface Module - the vehicle production unique identification). Afterward, the eMSP uses it to pair with the vehicle manufacturer cloud service.

Moreover, the *VehicleService* must actuate as an integrator and interface to unify all the different manufacturer APIs for the other internal eMSP components.

The *BluetoothService* implements a native feature of the user's device. Therefore, it's not considered an external service. However, it might be required to consume a third-party library to implement the Bluetooth [1] functions. For this project, a special version is not required.

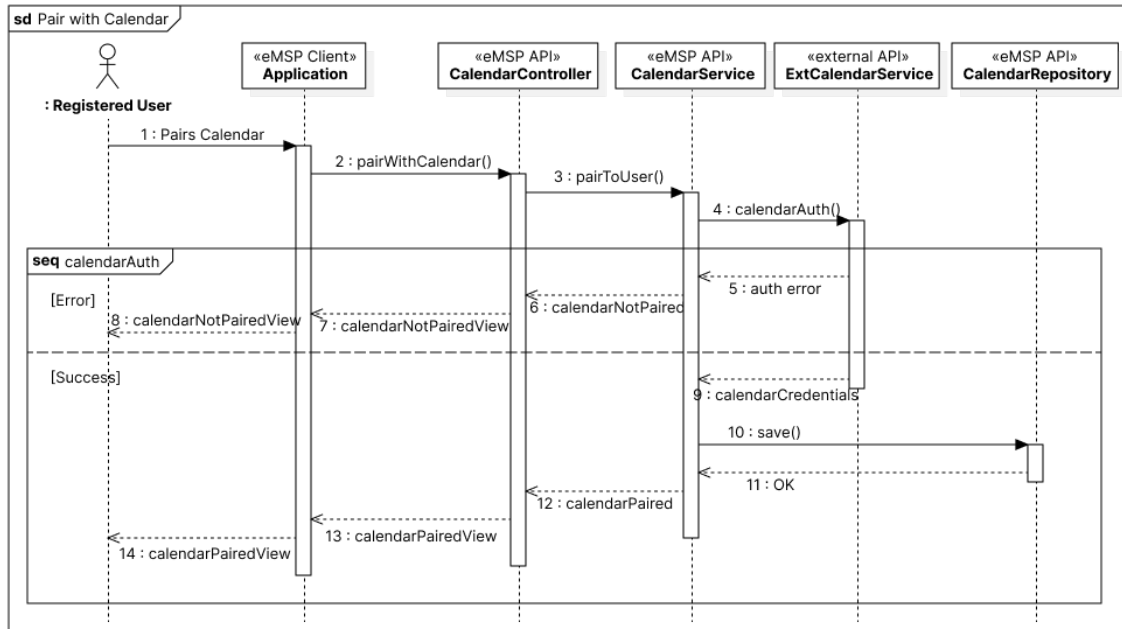


Figure 12: Pairing between the eMSP client application and the user's personal calendar

Like the *VehicleService*, the *CalendarService* works as an integrator and interface to the different personal calendars the user might have.

3.6.3 eMSP and CPMS: User's CS Search, Selection and Booking

These SDs include the connection between the CPMs and the eMSP. As explained in Section 3.5, the communication between these subsystems can be done via Publisher/Listener subscription or via API REST.

Via API REST, the *CPMSService* (eMSP component) is responsible for composing the requests and processing the responses of the CPMS API. Therefore this service is designed to communicate the eMSP internal components with the CPMS controller. Via Publisher/Listener subscription, the eMSP Listeners receive the CPMS Publisher messages.

The next SD describes the process of loading the CS data:

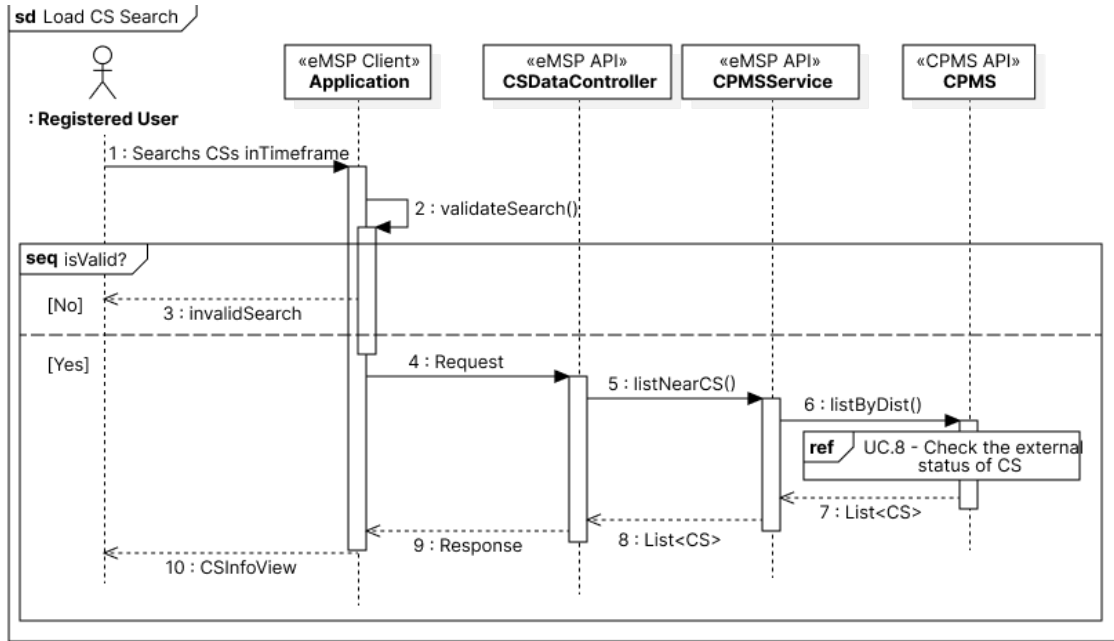


Figure 13: User requests CS information

After loading the charging stations information the eMSP Application Client will display the *MapView* 39. Sequentially, the user can choose a CS to get more specific data about its sockets and prices. Finally, the user has to choose the socket and timeframe from the CS chosen to complete the booking information. These choices are done in the eMSP Application Client in the frames 40, 41, and 42.

The booking process is divided between the two subsystems. While the eMSP process the user data and the booking the CPMS will reserve the physical infrastructure of the CS. The next two SDs show how this process is done:

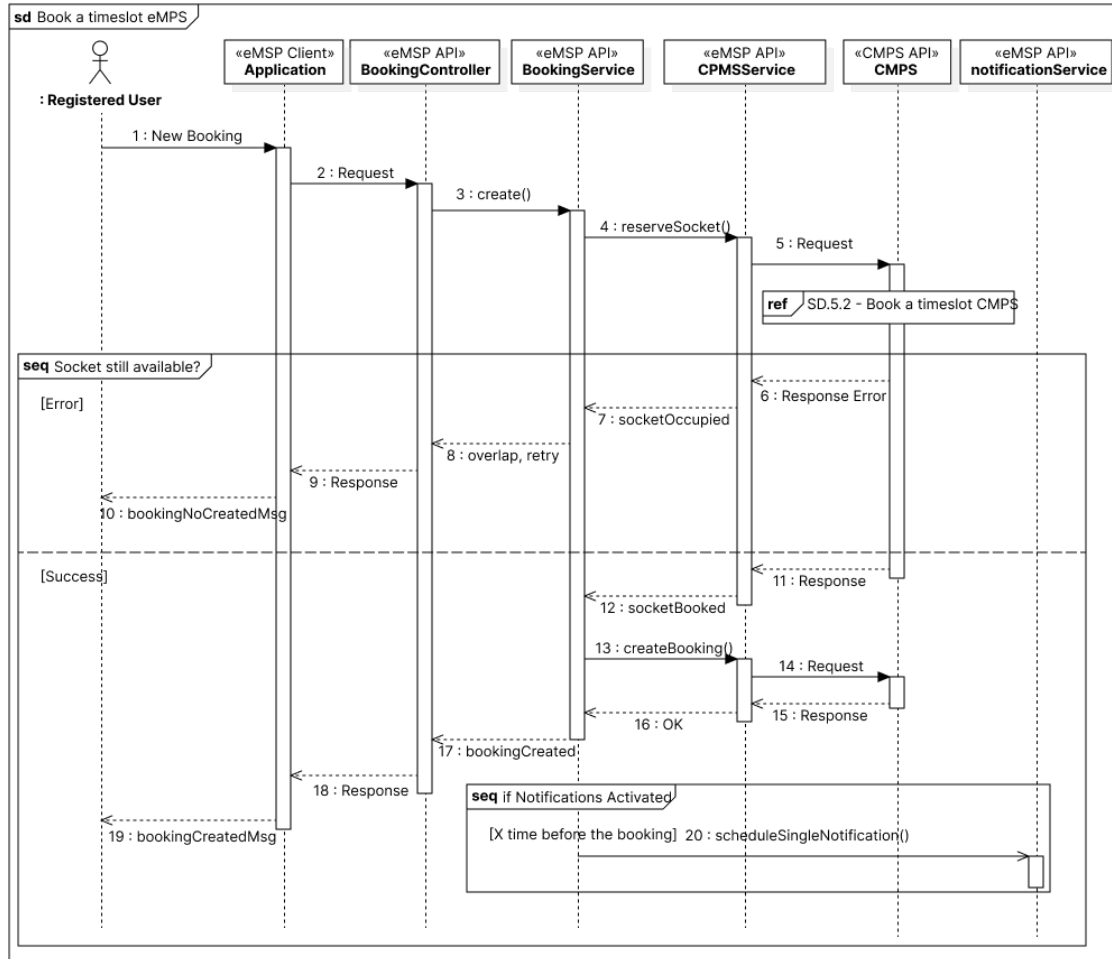


Figure 14: User creates booking through the eMSP Client Application

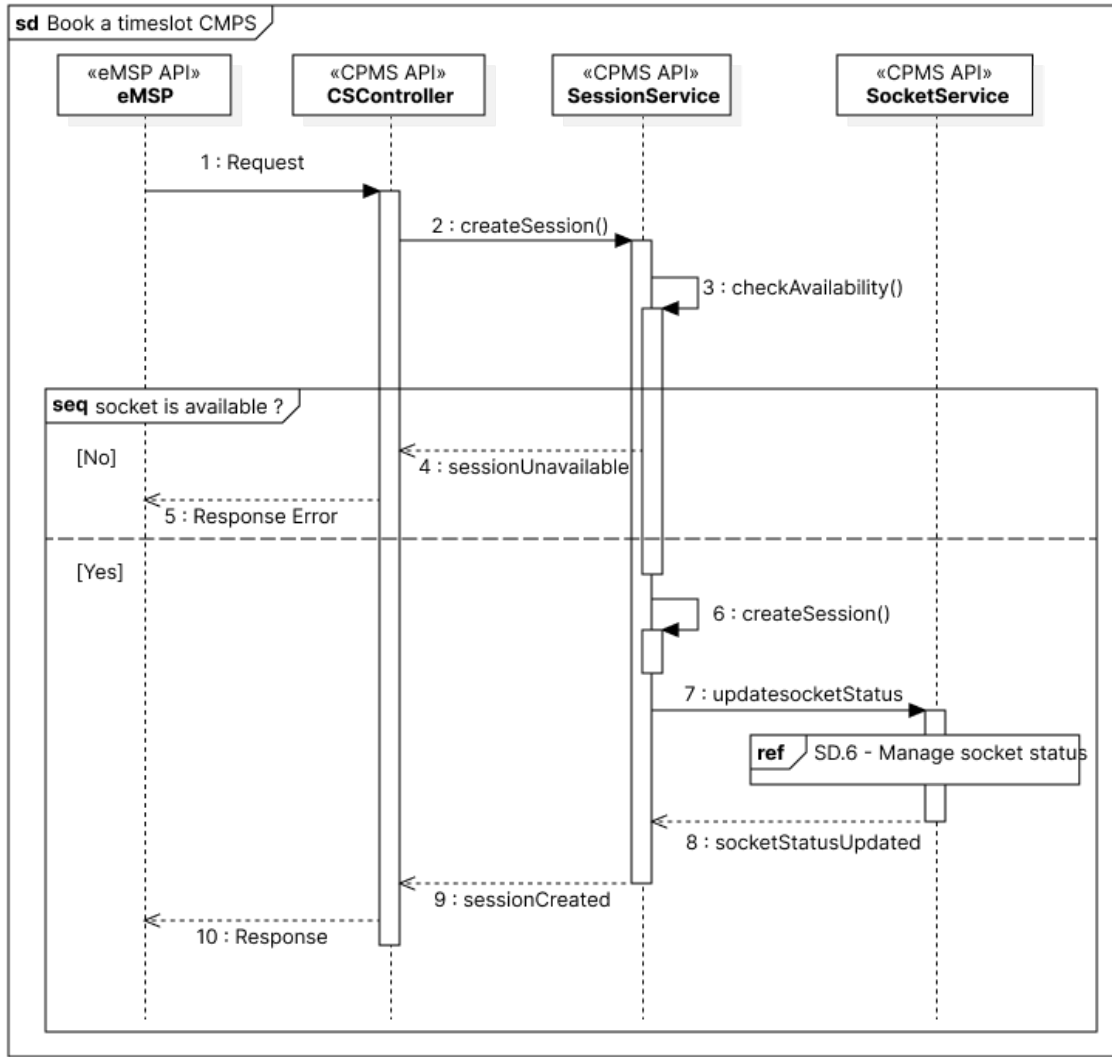


Figure 15: The CPMS system receives and processes the booking created at the eMSP

Therefore, the eMSP is the entrance for the user input, processes it, and requests the CPMS to reserve a socket based on the booking details. It's a sequential and synchronous process. The eMSP must receive the confirmation of the socket reservation by the CPMS. The next SDs are auxiliary to divide the SD complexity. They represent the functionalities of the SDs above:

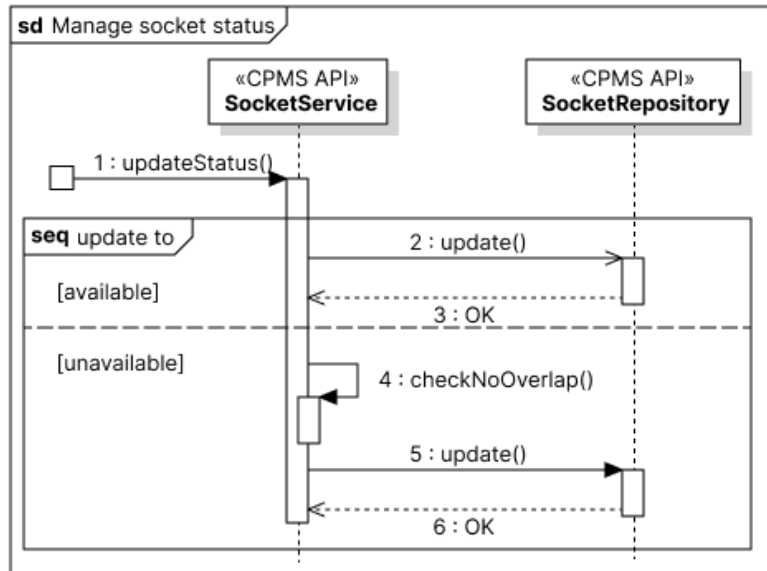


Figure 16: The CPMS updates the CS socket status

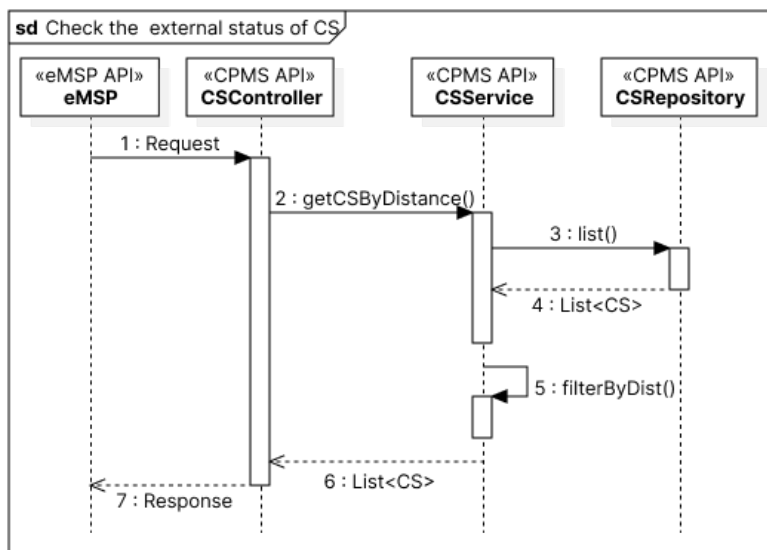


Figure 17: The CMPS loads CSs information

3.6.4 CPMS: Power Pricing and SOs

As the CPMS is the system responsible for managing the power distribution of the CSs it has an internal process design to fulfill its requirements. The next SDs describe:

- How the CPMS updates the CS's external energy supply prices based on the DSO data
- How the CPMS updates the CS's internal energy supply cost
- How the CPMS updates the total cost of charging, based on: the socket type, the internal and external supplies

All of this information is saved asynchronously in relation to its consumption. Meaning, the CPMS constantly updates the supply prices based on the DSO messages and the CS updates. This data is persisted in the CPMS database through the *CSRepository*.

This process is totally independent of the creation of the session (booking), where the *CSService* will consume it to list the CS information (see Figure 17).

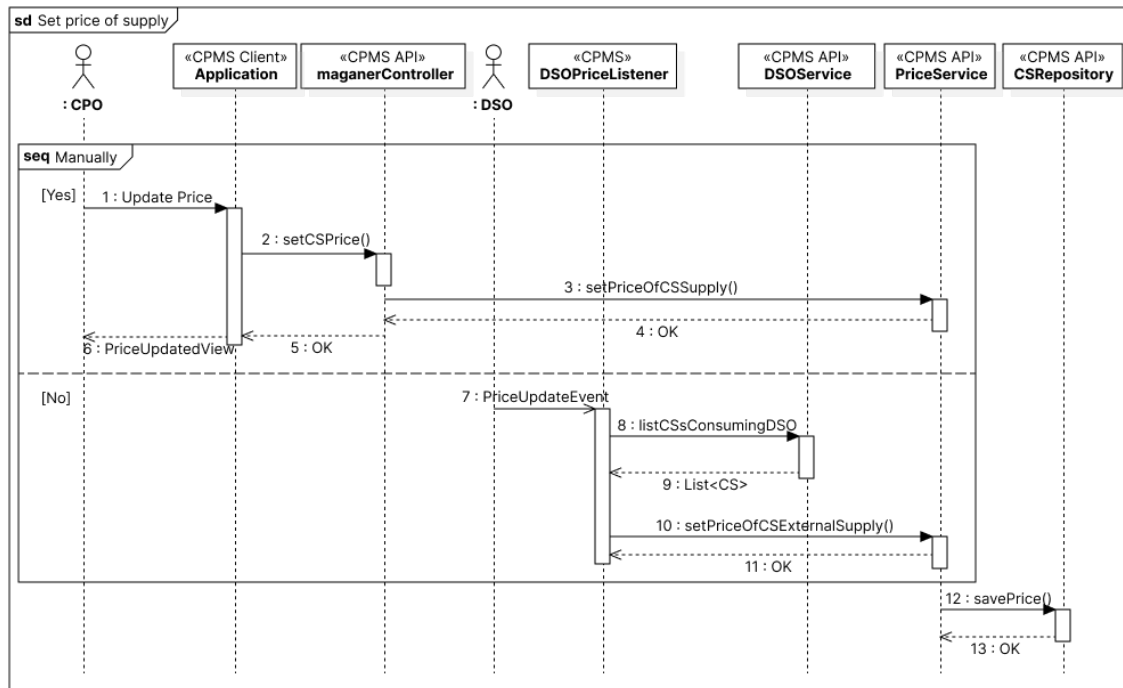


Figure 18: CPMS sets prices of power supply

The SD above describes the CPO action as a CS regulator. The CPMS will only take into account the DSO data to calculate the prices. However, the CPO can manually modify them. This will update the charging cost automatically.

Also, the CPMS expects messages with updates on the CS internal battery status. This information will modify the cost of charging automatically:

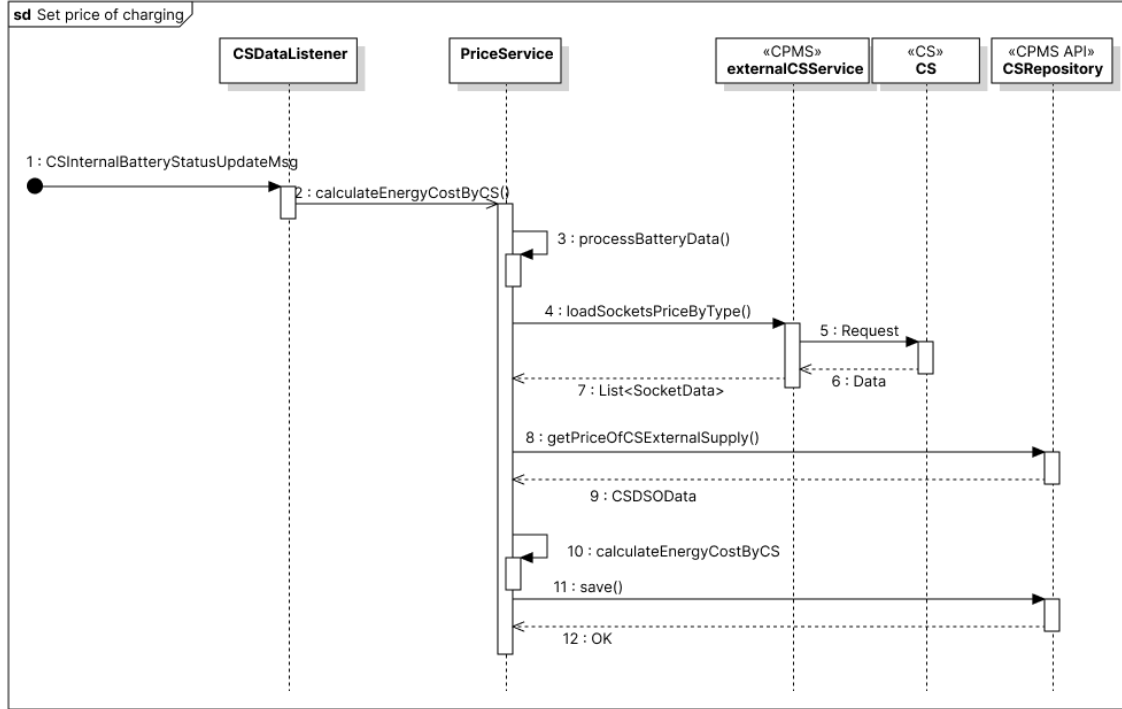


Figure 19: CPMS set final cost of charging

Finally, either the CPO or the DSO can create SOs to lower the charging cost. This event will be detected by the CPMS and published for the eMSP to notify the users. To see how the eMSP process this event see Figure 22.

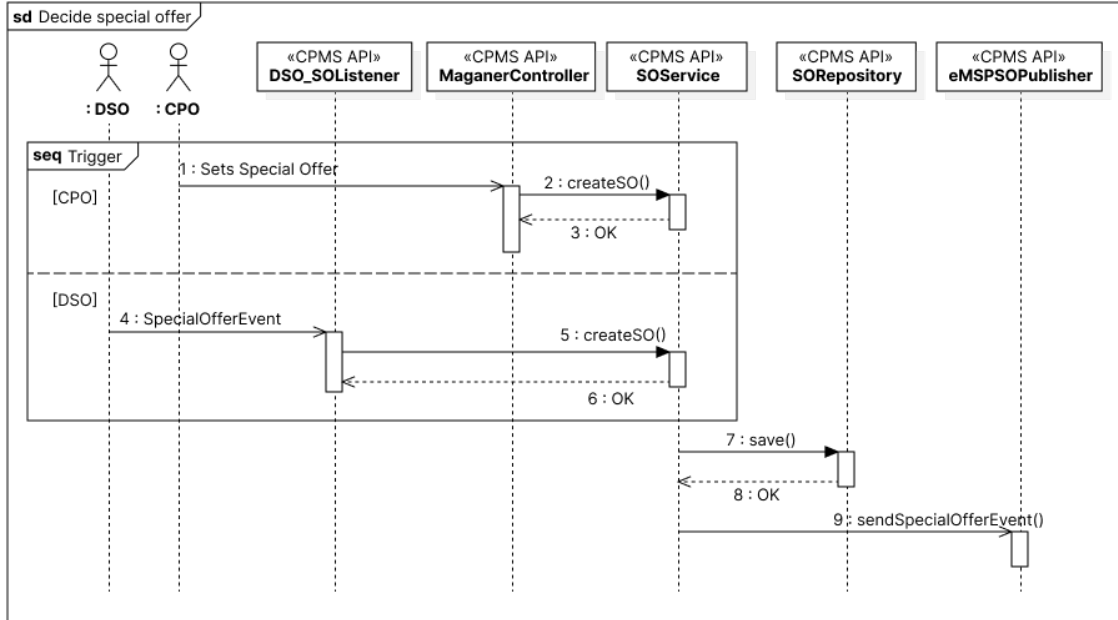


Figure 20: CMPS receives SOs

The charging price is updated by these processes and set for the booking at its creation.

3.6.5 CPMS: Decide CS supply type (internal, external or mixed)

As seen, the prices are being updated automatically. This information helps the CPMS decide how the CS will obtain its energy. Based on the external supply prices (DS) and the internal (CS batteries) it can decide:

- to buy energy from the grid (DSO supply)
- to reserve energy from internal batteries (internal supply)
- to do both

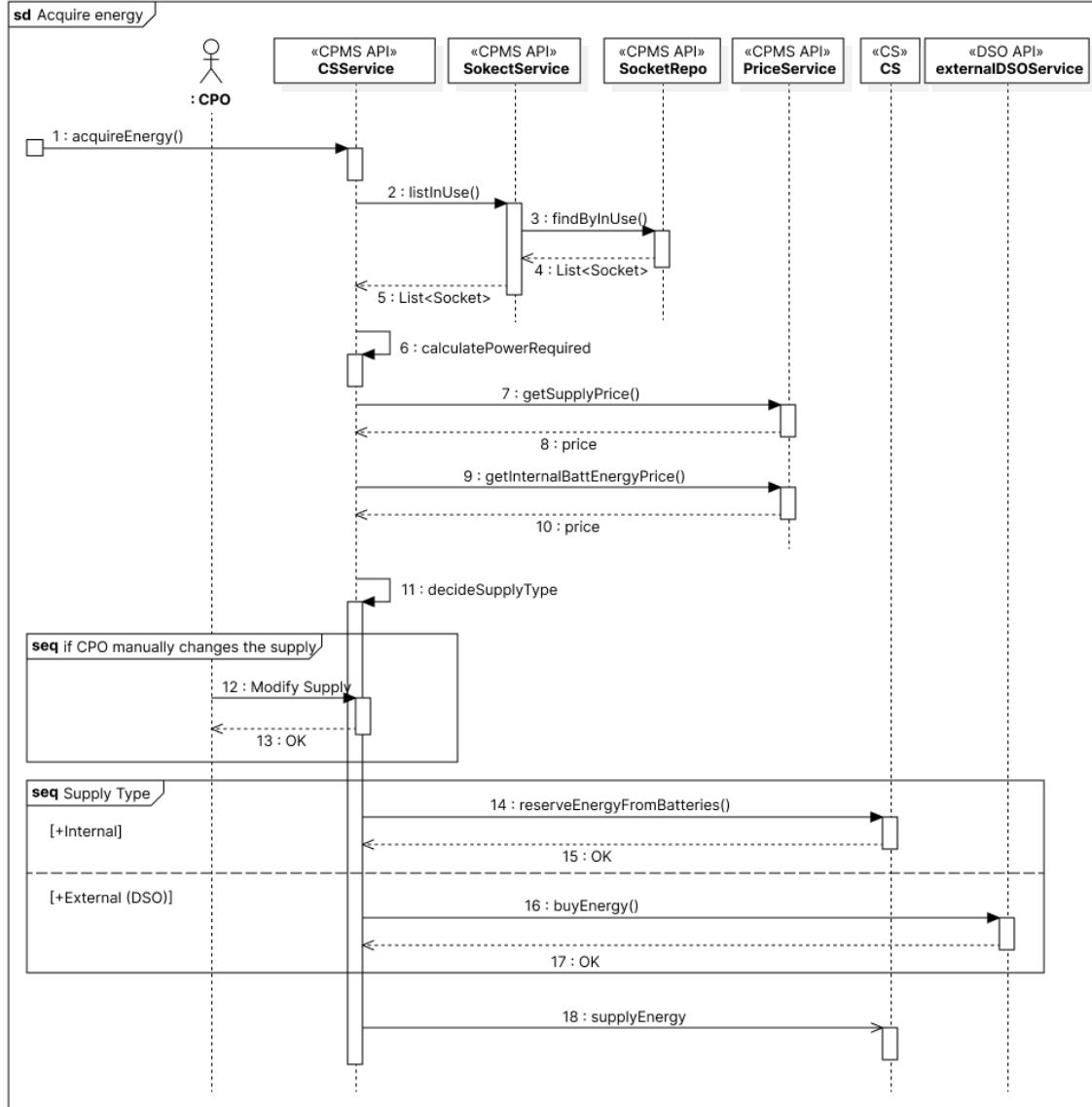


Figure 21: CPMS decides the power supply type (external, internal or mix)

Again the CPO can modify manually this automated process. The energy will be acquired to start the charging session. To see how this SD fits in that process see Figure 27.

3.6.6 eMSP and CPMS: Smart Feature - SO User's Notification

The eMSP is able to send push notifications to users. For this, it uses an external service and the notification process is centralized in the *NotificationService*:

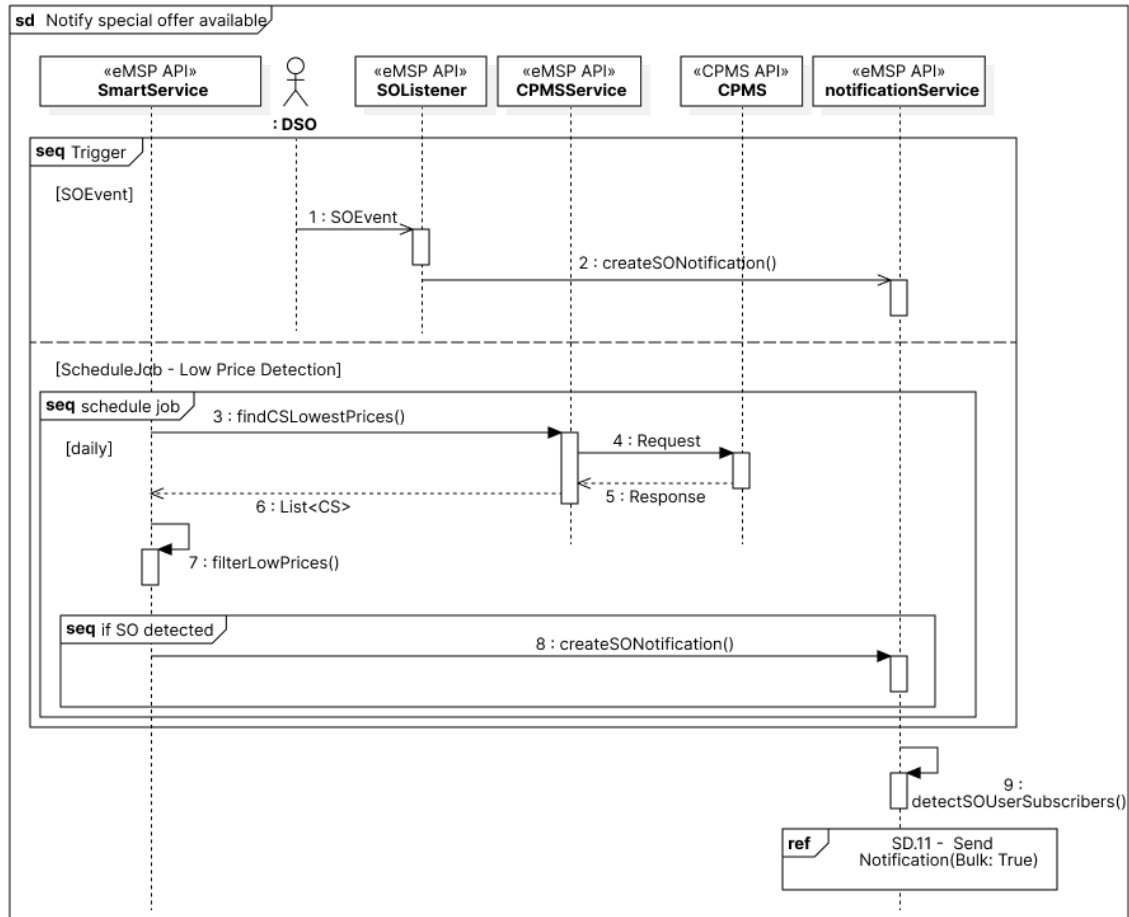


Figure 22: SO send push notification to the user

In this case, it is to announce an SO.

The eMSP is able not only to receive SOs from DSO and CPOs but to detect low moment prices. This is done at the *SmartService*, where a scheduled job is constantly supervising the charging prices to detect low ones. Not only it will detect them, but it will also filter which users must receive them based on its records.

The push notification system works as follows:

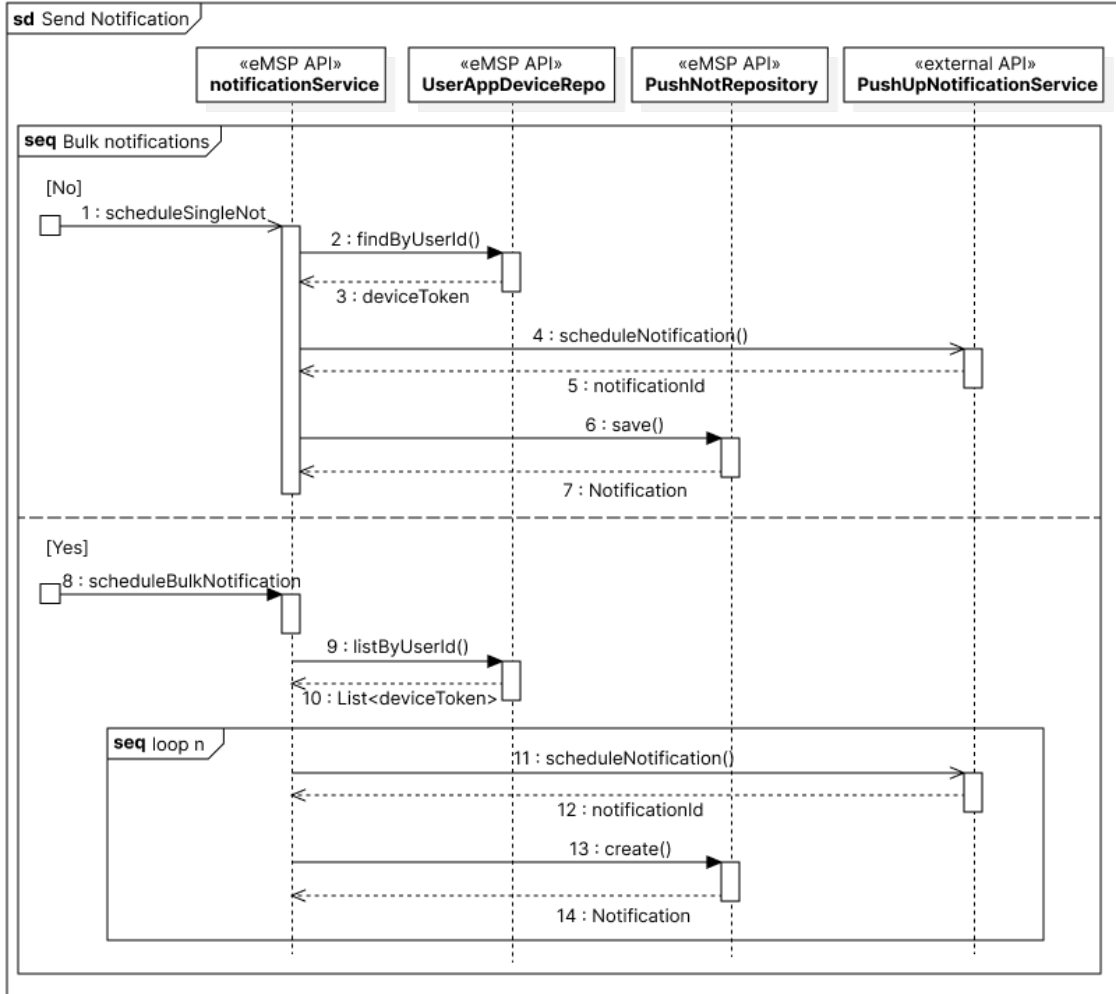


Figure 23: eMSP push notification process

The *PushUpNotificationsService* represents an external system. In this manner, the *NotificationService* can be extended to work with other external services. It works as an integrator and interface to the internal eMSP components.

Again, modularity is key to keep scalability. With this approach, another type of notifications such as email or desktop alerts can be added just by connecting them with the *NotificationService* component.

3.6.7 eMSP: Users Modifying and Deleting Bookings

The user can modify and delete his/her booking before he/she pays. This will update the socket status at the CPMS system:

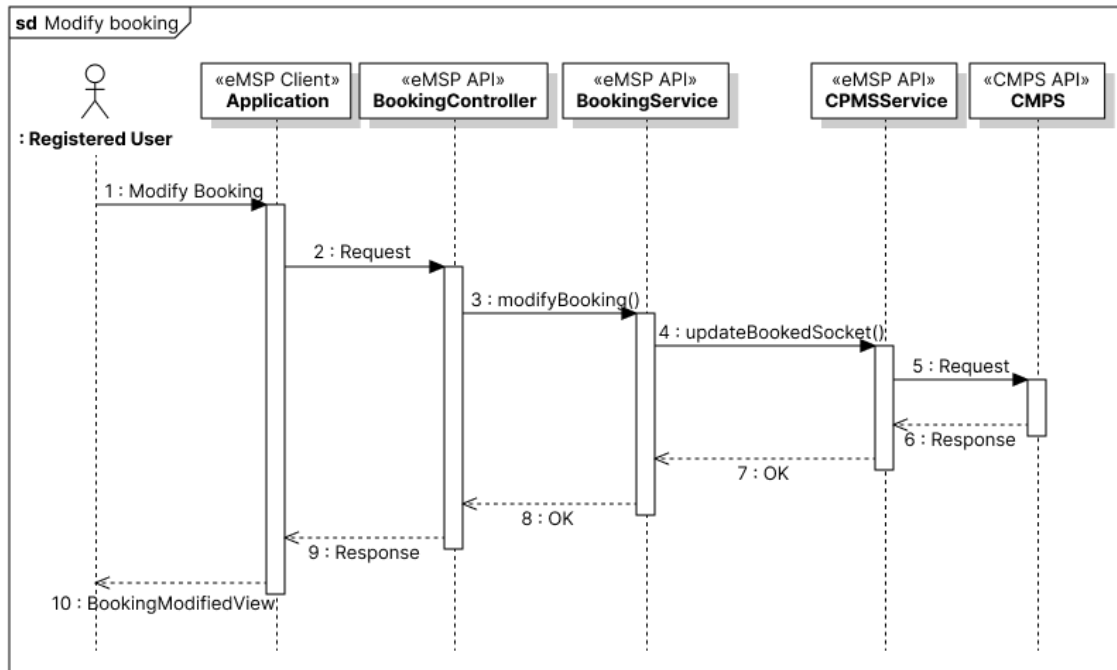


Figure 24: User modifies booking

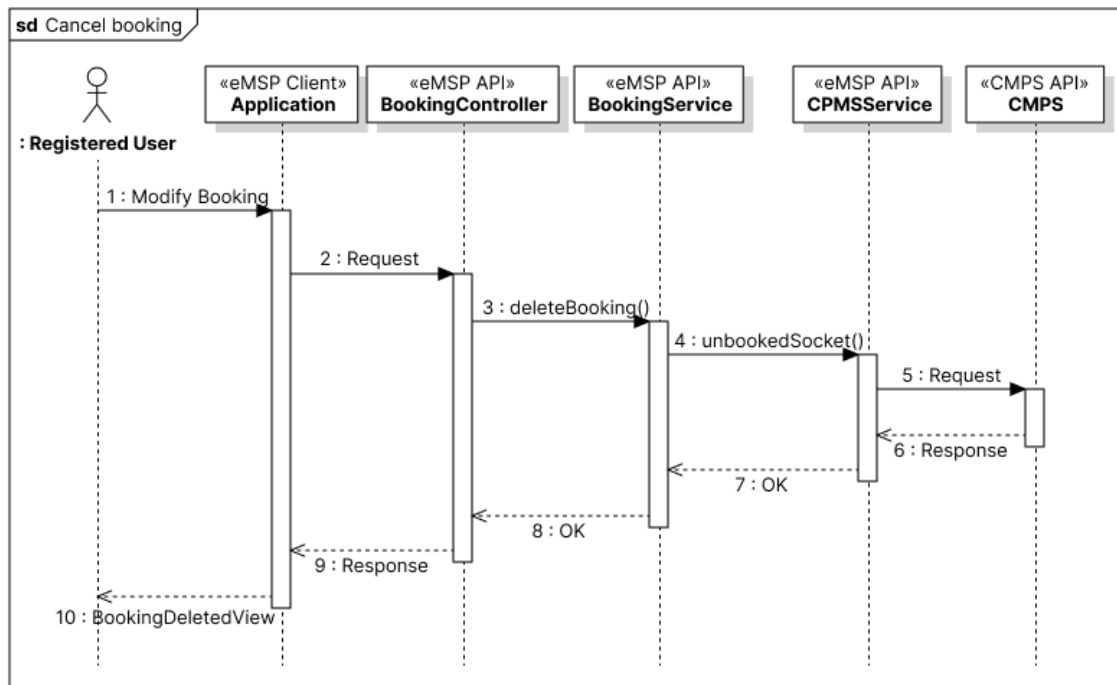


Figure 25: User deletes booking

3.6.8 eMSP: Bookings Payment

The booking payment is done before the charging starts. The charging starts with a code authorization process, see Figure 27. But to get the code, the eMSP will first verify that the payment has been approved.

In this context, though simply this process, the payment feature was designed to consume an external *PaymentService*. This service will transfer the money from the user's account to the eMSP. The bank credentials will be loaded to the eMSP system to be able to initiate the bank transfer.

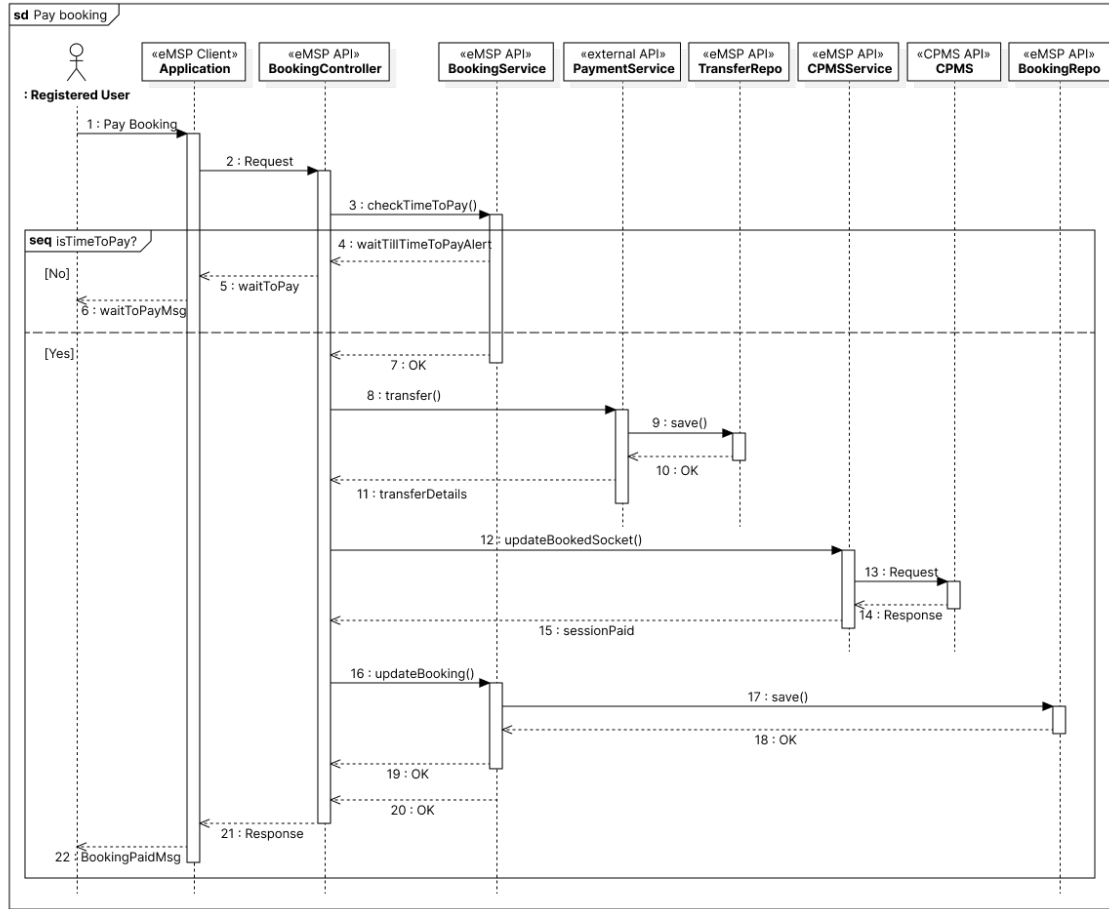


Figure 26: Booking payment

All the transfers will be saved in the eMSP database to keep the record via the *TransferRepository*. After the transfer has been done, the *BookingService* will update the booking state to paid.

3.6.9 eMSP and CPMS: Innit Vehicle Charging

After the payment has been done, the next step is to authorize the charging. For this, the eMSP will ask the CPMS for the session code:

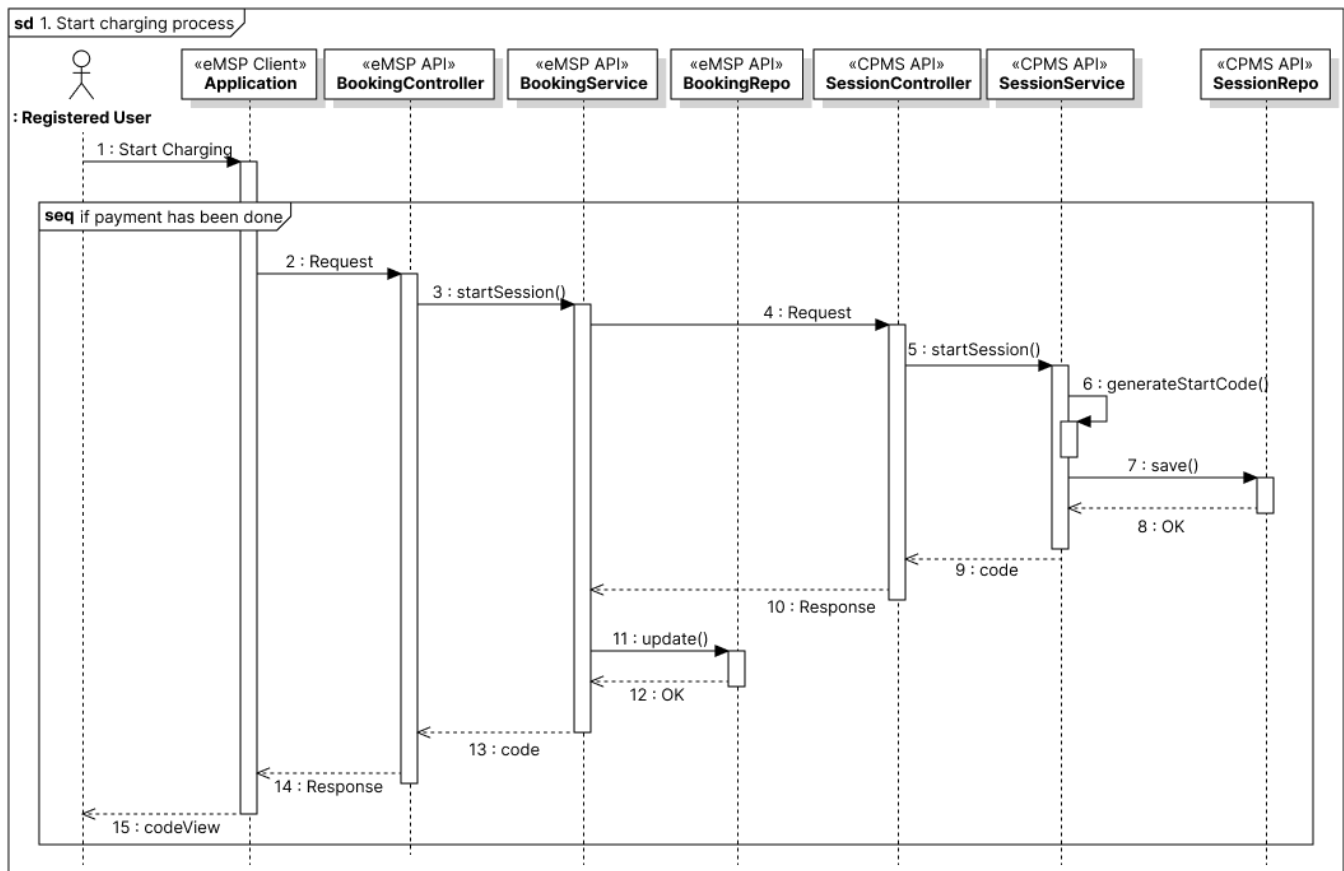


Figure 27: Authorize charging

To start the charging, the user must input the code in the CS socket. Finally, the CS will verify with the CPMS the code. If correct, it will acquire the energy and start supplying energy to the vehicle.

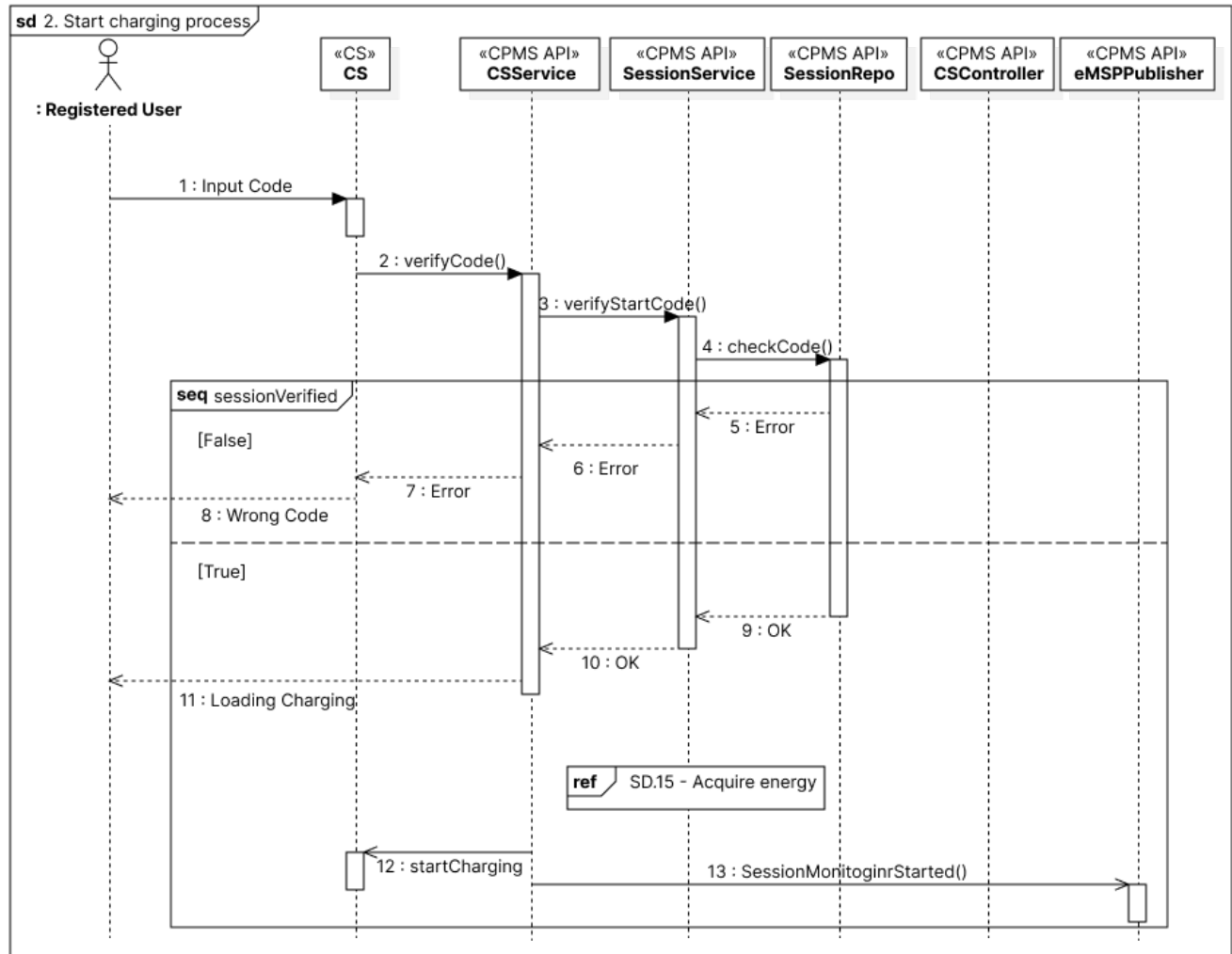


Figure 28: Init the vehicle charging

3.6.10 eMSP and CPMS: Monitoring Vehicle Battery and End of Charging

The user can check the vehicle battery status in two different scenarios: off charging and while charging. Both have different processes and components involved. In both, the user can check the battery level independently of its location

While the car is not charging, the user can visualize its battery status, see View 43. The communication with the car is done via its manufacturer's cloud service (external). Firstly, the user must pair the vehicle with his/her eMSP account, see Figure 11.

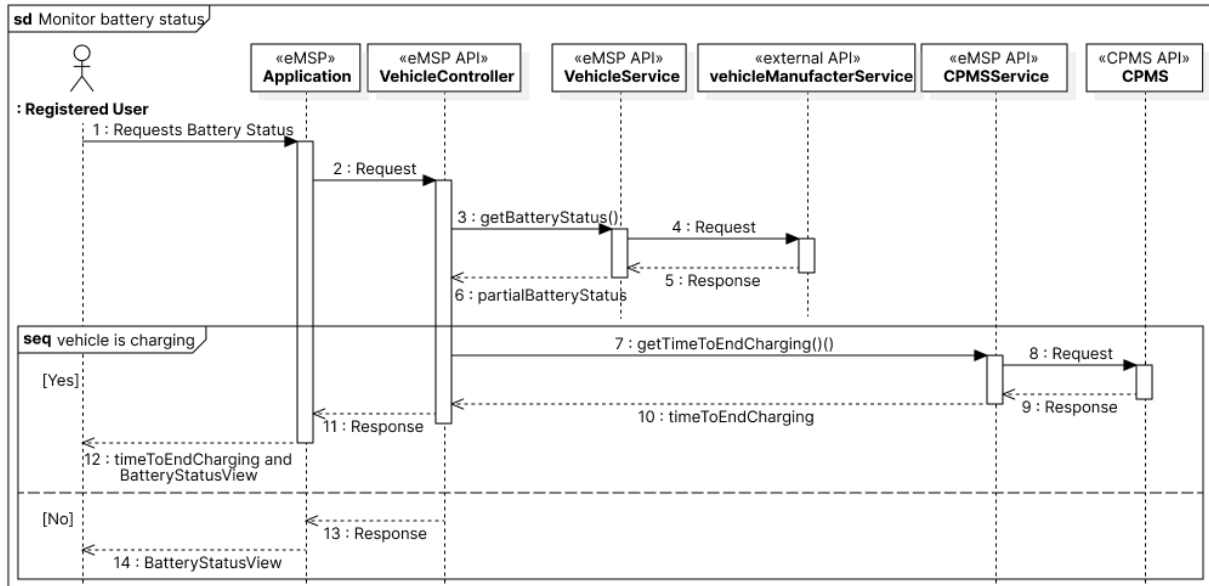


Figure 29: eMSP monitors the vehicle battery while off charging

Therefore, the status is updated every time the user loads the View 43.

In the second scenario, the user can verify the vehicle battery status while charging. This process is supervised by the CPMS. In that manner, the CPMS will publish an event when the charging is about to end. This will trigger a push notification in the user device by the eMSP:

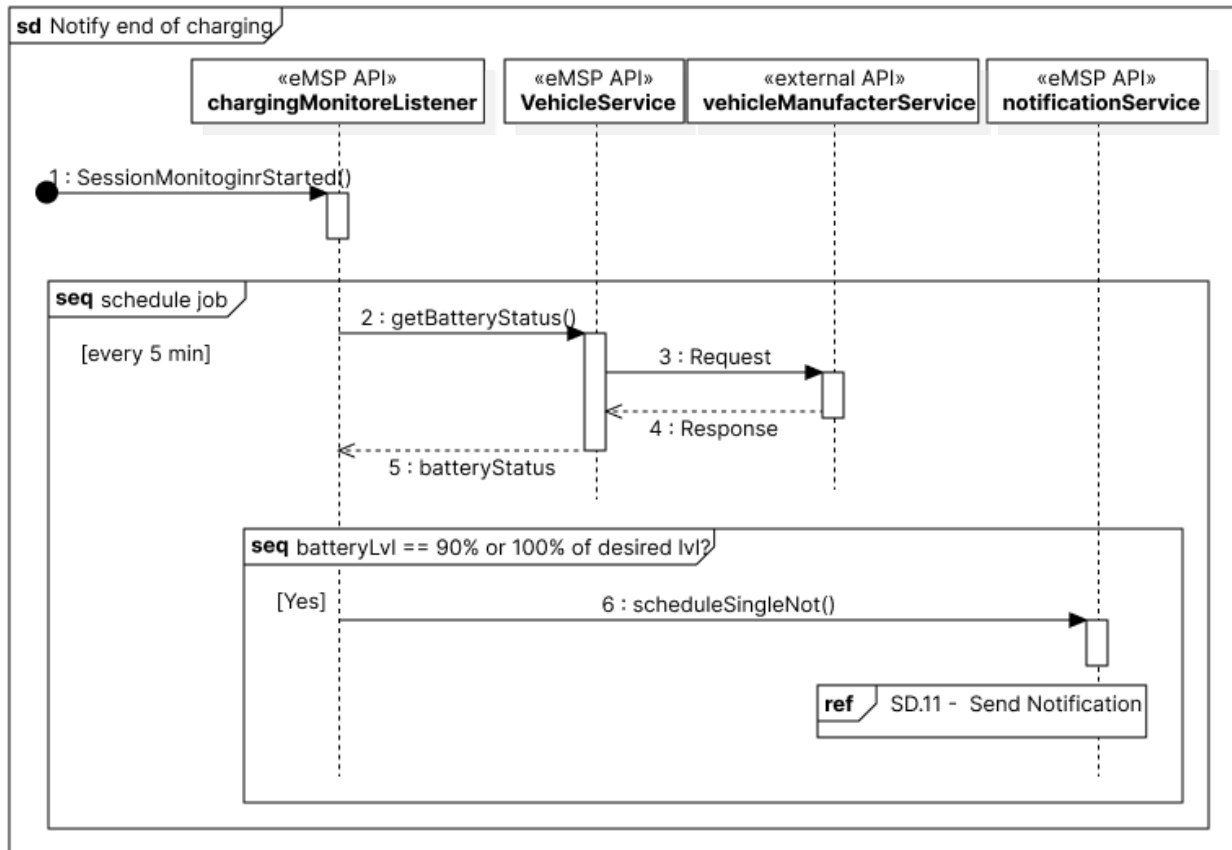


Figure 30: eMSP notifies the end of charging

Finally, the eMSP must be able to notify the user when the vehicle battery is low. This case is again while charging. This process is defined as a scheduler that will run a job every X minutes to check the user vehicle battery status.

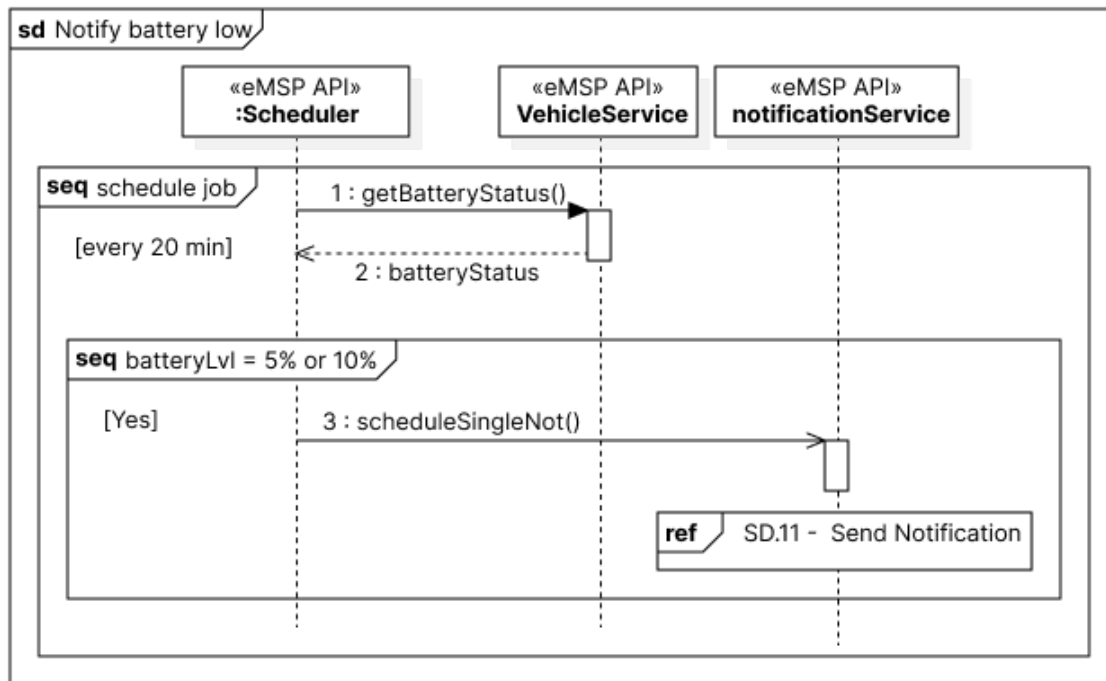


Figure 31: eMSP notification of low battery

3.6.11 eMSP: Smart Feature - Suggest Charging Schedule

Between others, the *SmartService* is able to structure charging scripts. Meaning, based on the user locations and timeframe is capable of listing a sequence of possible CSs and sockets to charge the vehicle. All is done based on the user's past story. To this, it can be added notifications to remember the user to create these bookings, with the possibility of modifying them in case he/she changes his/her plans.

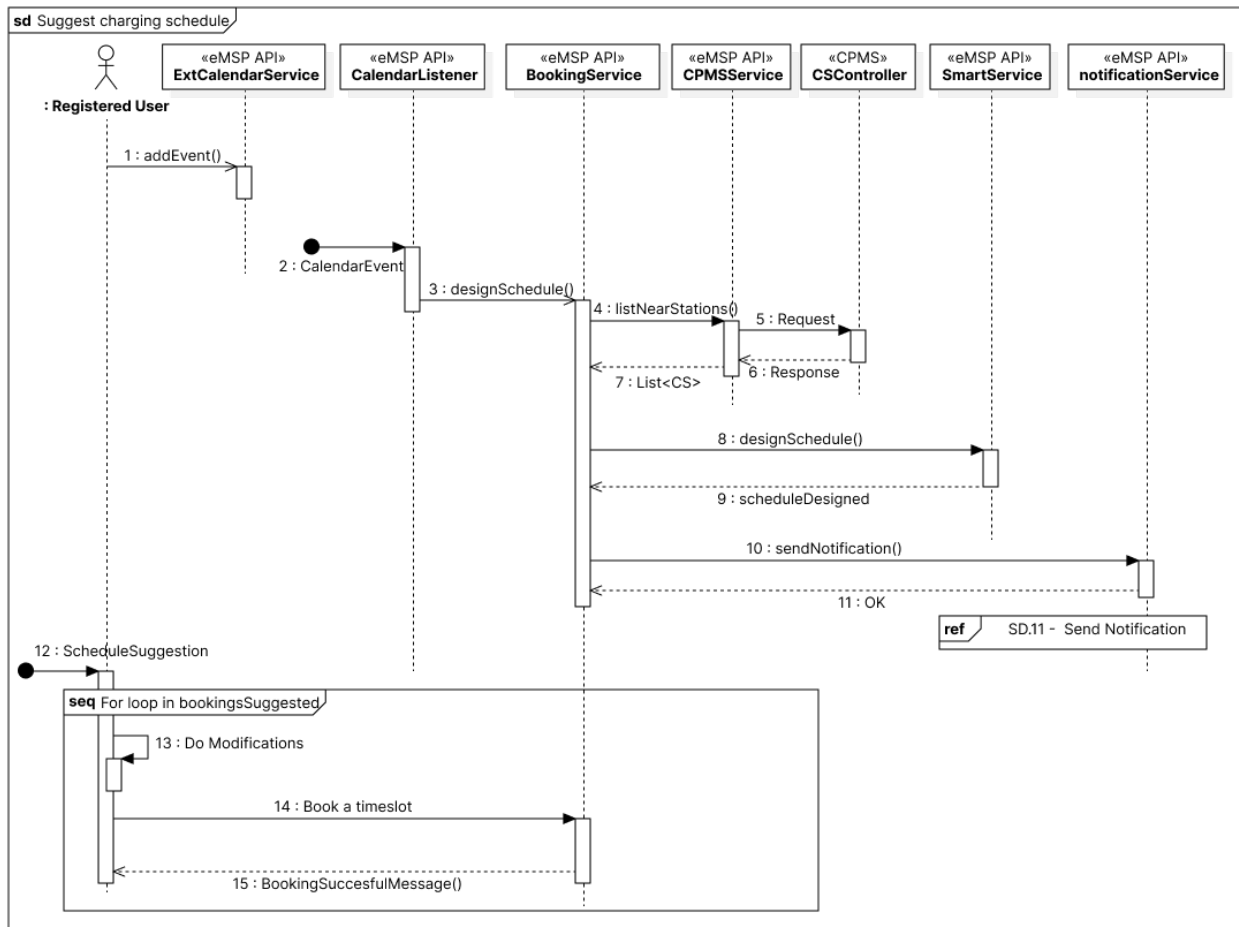


Figure 32: Suggest Charging Schedule

3.7 Other design decisions

Besides using the sub-system's own services, each sub-system calls external services when they need to pull information or add functionality not provided by its internal components. This process can be done either through the use of external REST APIs or by using Event-driven APIs aka Listeners. Their use is justified since third-party services can increase the functionality of our system by re-using them, making the development process more efficient and providing the system with more functions.

We will use integrator interfaces when integrating with external services so that the interface describes the functionality to be extracted from that service without sticking to a particular third-party service provider, favoring decoupling and flexibility. For instance, the "vehicleManufacturerService", as its name suggests, allows connecting to the REST APIs provided by each vehicle manufacturer and, by passing the vehicle id, pulls data such as the vehicle battery status, regardless of the vehicle manufacturer. A similar situation happens for the Calendar service.

Although the use of third-party services could make the different sub-systems more prone to failure, we will use reliable and supported APIs provided by mature and recognized companies. In that way, they enrich the offered features of our system and ease the development process. Their use requires a periodic review seeking possible API updates to improve its functionality in terms of performance, and security, among others.

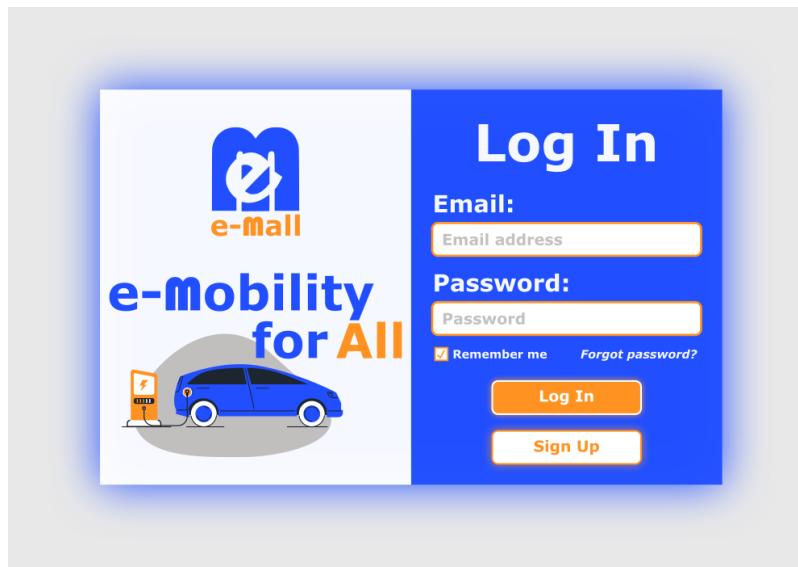
4 User Interface Design

In this section, it is shown how different interfaces that would be displayed to the final users. The main focus of the chosen interfaces is the main purpose of the project, booking at a charging station. Furthermore, some other possible interfaces of the application are indicated, such as the log-in and register screens, monitor battery status screen, profile screen, and CPMS main menu if the user is a CPO.

Focusing on the booking phases, it is shown seven interfaces. First, the main menu, where the user can book a socket or check past bookings. Second, if the user enters "My Bookings", he or she can see dates, hours, and locations in the map of their bookings and modify and cancel them if necessary. Third, once the user enters the booking interface he or she will select the date, hour, and % battery wanted and continue with the Book button. Fourth, the user will select a charging station by double-clicking on one of the shown on the Booking map. Fifth, he or she will select the desired type of socket double clicking again. Sixth, the user will access the number of sockets available of that desired type in the selected charging station at the date and hour specified, taking into consideration, that it may happen that all of them are occupied, so the user should go back and change some of the information to final select an available socket. Finally, there is a Confirm Booking interface where he or she can check all the information picked and confirm.

Finally, all of the interfaces are designed for a tablet's screen size, as it is the perfect fit to show everything in a document like this one. However, this application can be scaled down to a mobile size or scaled up to a computer screen. This is why, one press or click remarks the blue buttons in orange and the second click means the final selection.

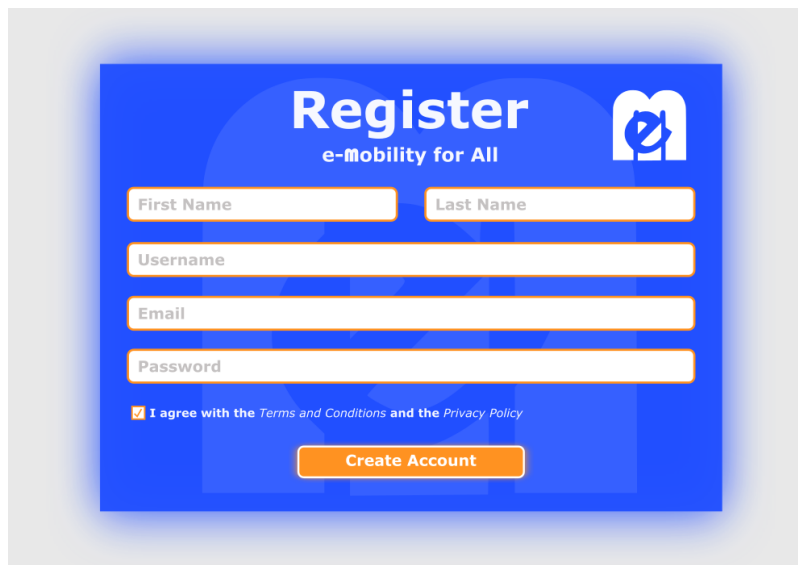
4.1 Log In



The Log In interface is a blue rectangular form. On the left side, there is a logo consisting of a blue 'M' with a white plug icon inside, and the text 'e-mall' in orange below it. Below the logo, the text 'e-mobility for All' is displayed in blue and orange, with an illustration of a blue car plugged into a charging station. On the right side, the title 'Log In' is in large white font. Below it, there are two input fields: 'Email:' with a placeholder 'Email address' and 'Password:' with a placeholder 'Password'. Below the password field, there is a checked checkbox for 'Remember me' and a link 'Forgot password?'. At the bottom, there are two orange buttons: 'Log In' and 'Sign Up'.

Figure 33: Log In interface

4.2 Register



The Register interface is a blue rectangular form. At the top, the title 'Register' is in large white font, with 'e-mobility for All' in smaller white font below it. On the right side, there is a logo consisting of a blue 'M' with a white plug icon inside. Below the title, there are four input fields: 'First Name', 'Last Name', 'Username', and 'Email'. Below the email field, there is a 'Password' field. Below the password field, there is a checked checkbox for 'I agree with the Terms and Conditions and the Privacy Policy'. At the bottom, there is an orange button labeled 'Create Account'.

Figure 34: Register interface

4.3 Main Menu

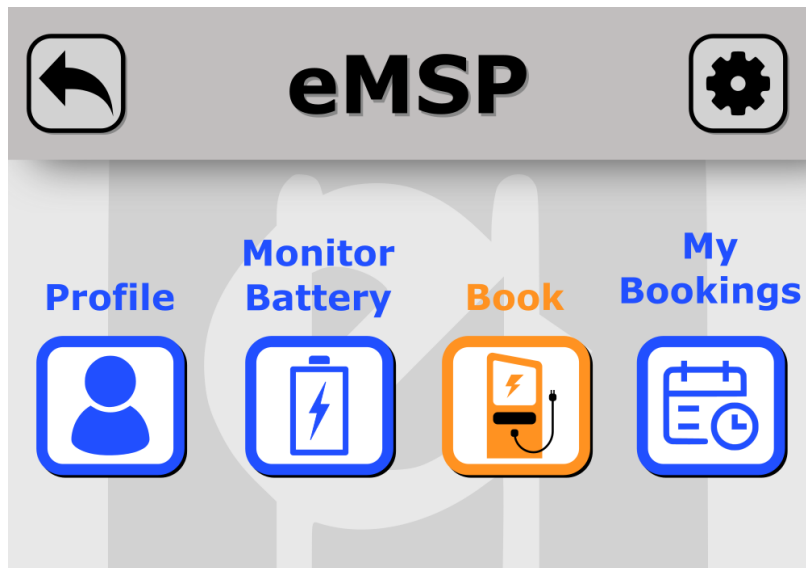


Figure 35: Main menu interface

4.4 My Bookings

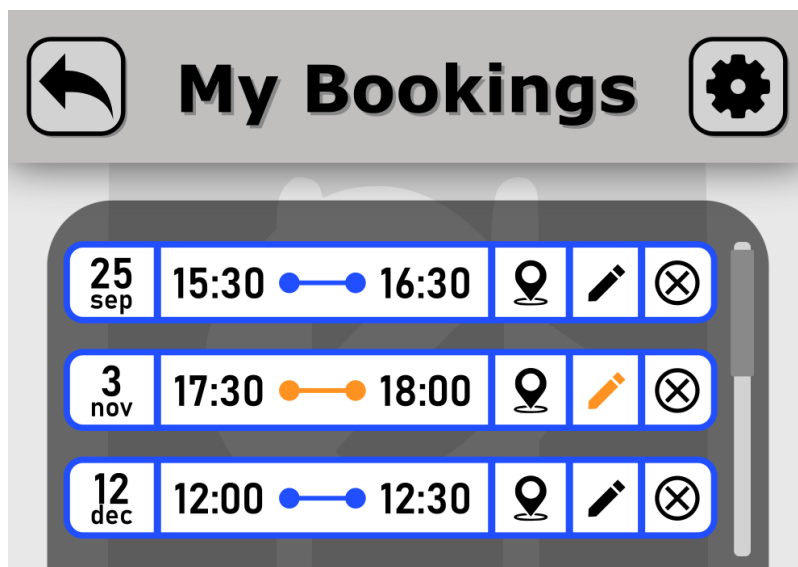


Figure 36: My Bookings interface

4.5 Profile

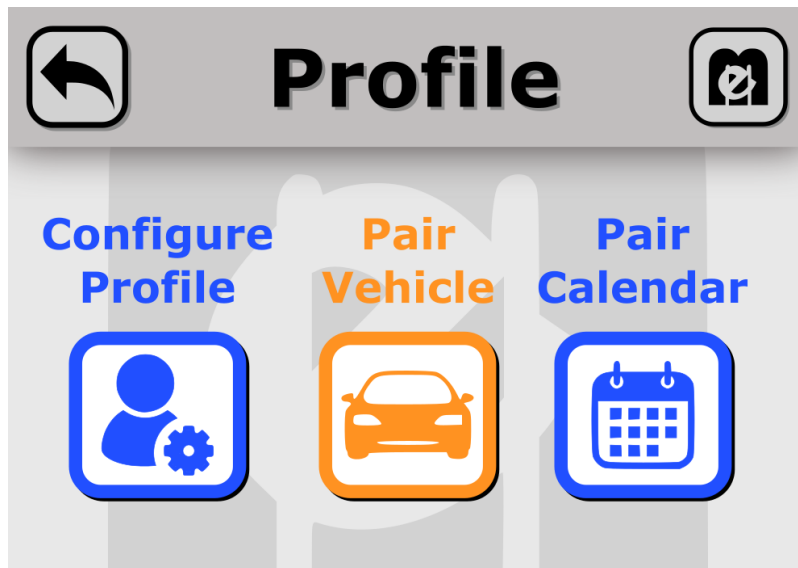


Figure 37: Profile interface

4.6 Booking Information

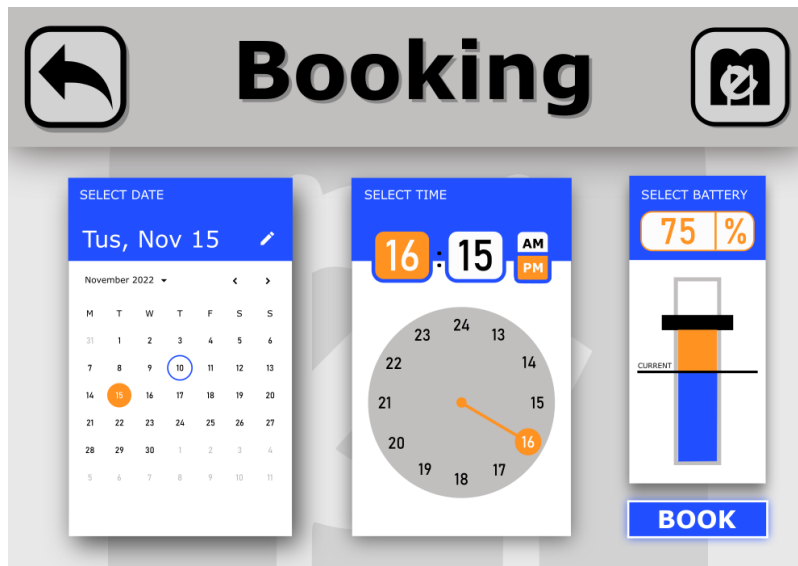


Figure 38: Booking information interface

4.7 Booking map

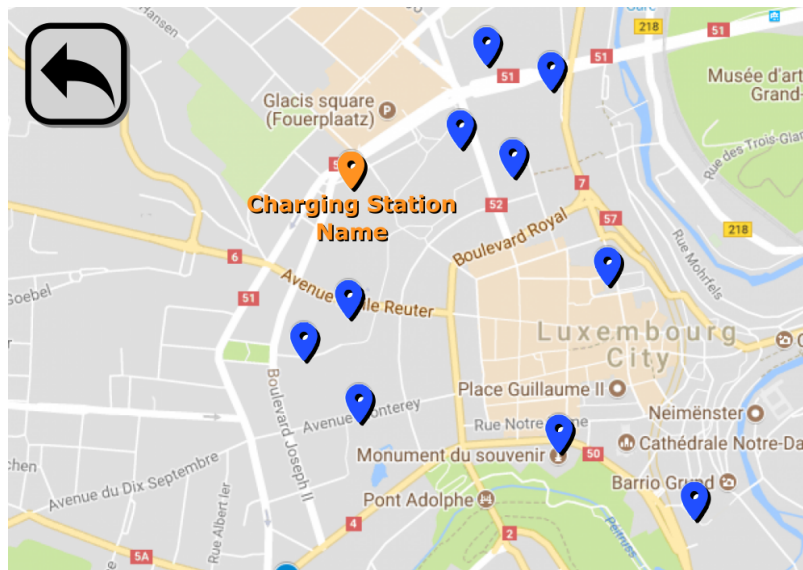


Figure 39: Charging station selection interface

4.8 Booking socket type

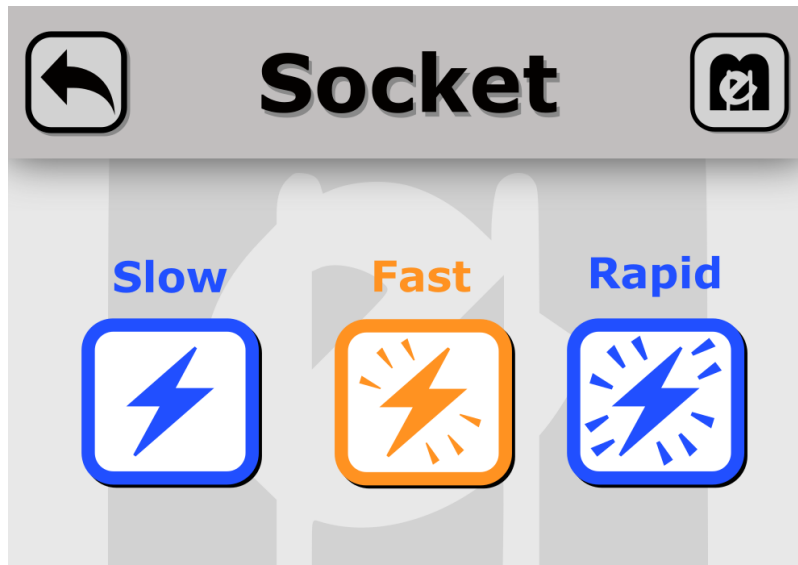


Figure 40: Socket type selection interface

4.9 Booking socket

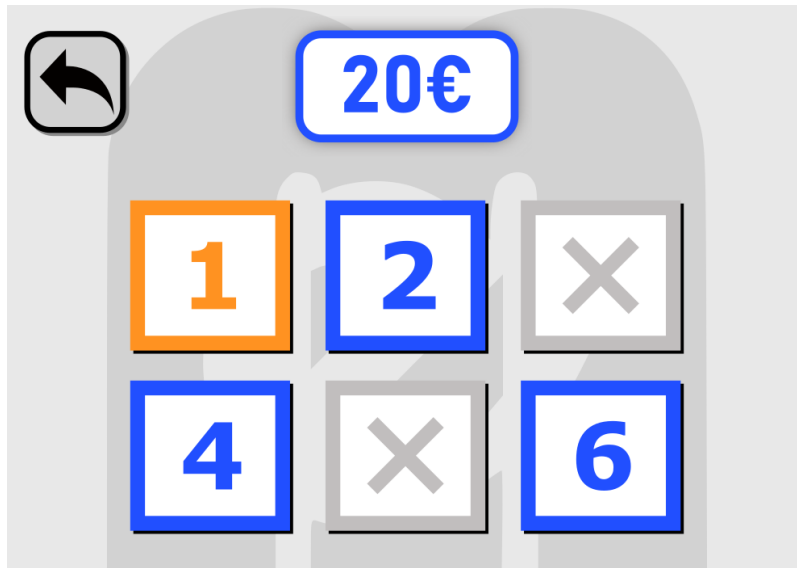


Figure 41: Final socket selection interface

4.10 Confirm Booking

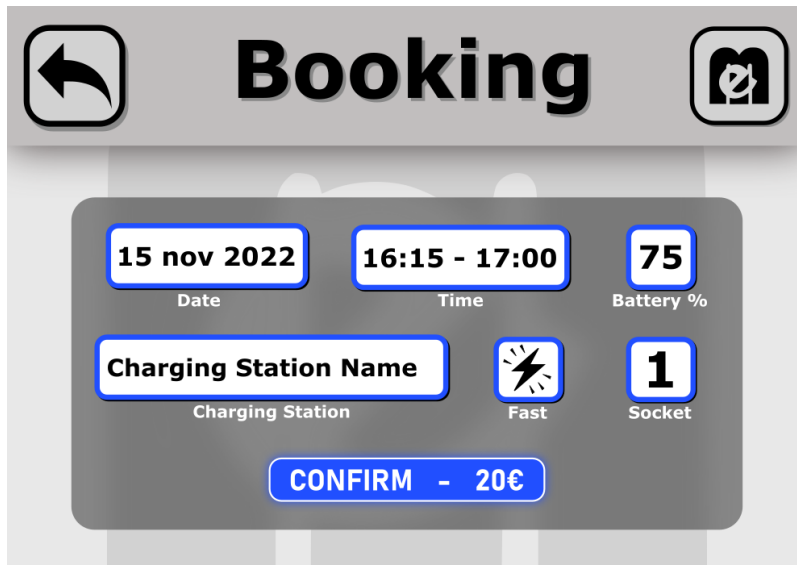


Figure 42: Confirm Booking information interface

4.11 Monitor Battery Status

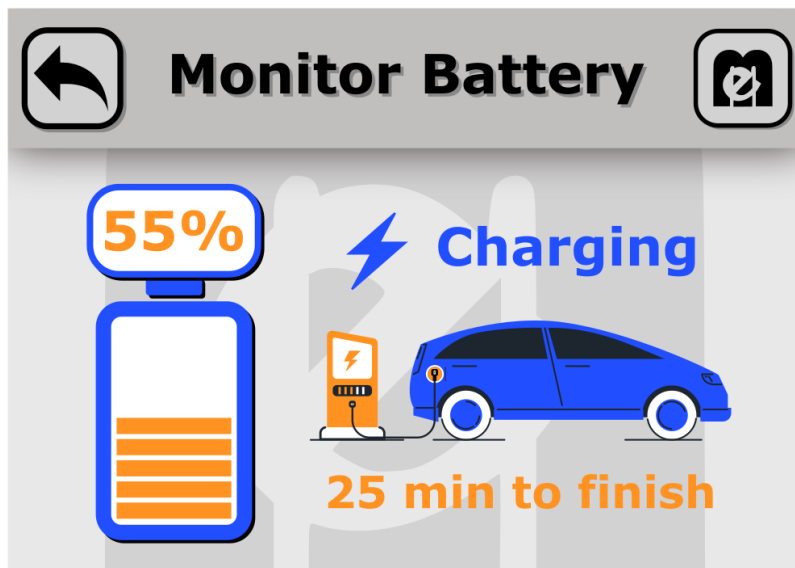


Figure 43: Monitor battery status interface

4.12 CPMS view

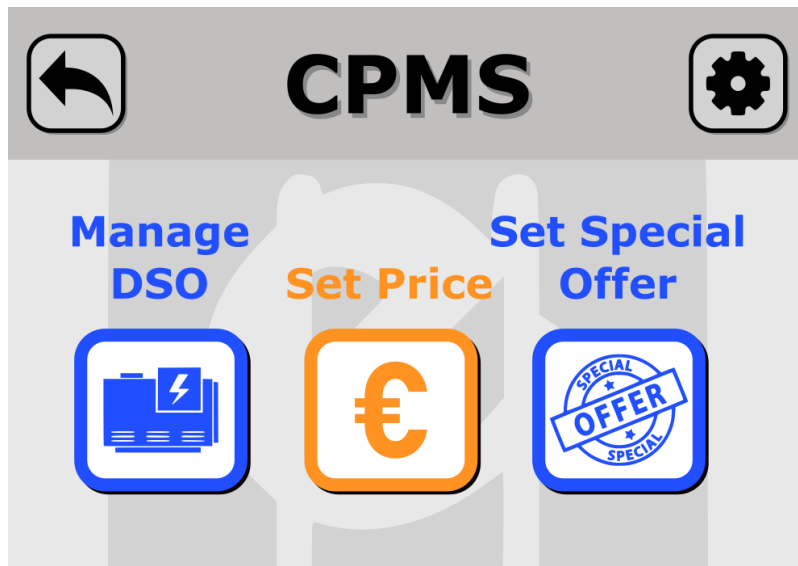


Figure 44: CPMS main menu interface

5 Requirements traceability

This section aims to map the relationship between the requirements listed in the past RASD document [4] and the sub-system components. For this section each component will have its abbreviation:

eMSP	
Component	Abbreviation
AuthController	eAC
VehicleController	eVC
CalendarController	eCC
BookingController	eBC
CSDDataController	eCDC
AuthService	eAS
BluetoothService	eBS
VehicleService	eVS
BookingService	eBOS
NotificationService	eNS
SmartService	eSS
CPMSService	eCPS
UserRepository	eUR
AuthRepository	eAR
VehicleRepository	eVR
CalendarRepository	eCR
UserAppDeviceRepository	eUAR
PushNotificationRepository	ePR
BookingRepository	eBr
TransferRepository	eTR
SOListener	eSOL
ChargingMonitoreListener	eCML
AuthMiddleware	eAM

Table 5: eMSP components abbreviations

CPMS	
Component	Abbreviation
CSController	cCC
ManagerController	cMC
SessionController	cSC
SessionService	cSS
SocketService	cSKS
CSService	cCS
DSOService	cDS
PriceService	cPS
SOService	cSO
SORepository	cSOR
SocketRepository	cSKR
CSRepository	cCR
SessionRepository	cSR
DSOPriceListener	cDL
DSO_SOListener	cSOL
eMSPSPublisher	cSOP
eMSPMonitoringPublisher	cMP
Scheduler	cS

Table 6: CPMS components abbreviations

The requirements are listed ahead:

5.1 Non Functional Requirements

Identifier	Description
R22	The eMSP must have an interface to interact with the vehicle (hardware or software) to compile information about its battery status
R23	The eMSP must provide an authentication to identify their users securely and to save their credentials in the device: automatic login
R24	The eMSP must have a mobile push notification system to send information about the battery status of the vehicle and charging stations with offers and lower prices
R25	The eMSP must interact synchronously with the CPMSs for the booking feature
R26	The eMSP must consume the CPMSs charging station information every time the user opens its interface to show updated information
R27	The eMSP must be able to interact with a third party payment system
R28	The CPMS must be able to interact with the physical hardware of the CSs infrastructure to unlock the sockets
R29	The eMSP must be able to consume data from the calendar/navigation system of the user
R30	The eMSP must have an interface integrator to interact with multiple CPMSs from different CPOs

Table 7: Non Functional Requirements

5.2 Functional Requirements

Identifier	Description
R1	The eMSP interface shall allow the user to select a charge station from a list based on its location
R2	The eMSP interface shall allow the user to select an available socket from a charge station list with its socket types, availability and prices
R3	The CPMS must be able to identify and list the location of a charging station, the number of its available and unavailable sockets, their type (charge speed), their cost, and, the estimated amount of time until the first socket of a type is freed
R4	The eMSP shall allow the user to book freely only available sockets based on a determined timeframe and a desired percentage of battery to be reached,
R5	The CPMS must change the socket status to unavailable the timeframe it has been booked
R6	The eMSP must notify the CPMS to start charging the vehicle only if the user booking starts and has paid for energy to be consumed
R7	The eMSP shall allow the user to cancel a book
R8	The eMSP must provide a graphical interface where the user can consume information about the battery status of his/her vehicle
R9	The eMSP must not let a user to book two sockets with overlapping times
R10	The eMSP must collect user charging habits such as the mean charging time of a book, frequented charging stations, and type of sockets used.
R11	The CPMS must be able to list the DSOs' energy prices
R12	The CPMS must be able to set the charging price, create special offers and acquire energy automatically
R13	The CPMS must be able to let the CPO set the charging price, create special offers and acquire energy manually
R14	The CPMS must be able to know the amount of energy available in the charging stations batteries and decide the source of energy: either from a DSO, the stations internal batteries or a mix of both
R15	The eMSP shall filter the offers listed by CPMS relevant to the user (is consider relevant, if its booking price is lower than the usual, considering the type of socket the user tends to select and the frequented charged stations)
R16	The eMSP shall notify the user when the vehicle battery has only 10 and 5 percent of charge
R17	The CPMS must monitor the amount of power supplied by the socket when a vehicle is charging to infer when the battery has reach the expected percentage and the time left to end of the charge
R18	The eMSP must interact with the CPMS charging monitoring process to notify the user when the charging service has been 90% and 100% completed
R19	The eMSP must monitor the vehicle when its driving, turn on and off to keep the user informed about its charge
R20	The eMSP shall allow the user to pay before the charging service
R21	The eMSP shall allow the users to do login and register

Table 8: Functional Requirements

eMSP										CPIMS																																			
Req	Controller					Service					Repository					Other					Controller				Service				Repository				Other												
	eAC	eVC	eCC	eBC	eCDC	eAS	eBS	eVS	eBOS	eNS	eSS	eCS	eCPS	eUR	eAR	eVR	eCR	eUAR	ePR	eBR	eTR	eSOL	eCML	eAM	cCC	cMC	cSC	cSS	cSKS	cCS	cDS	cPS	cSO	cSOR	cSHR	cCR	cSR	cDL	cSOL	cSOP	cMP	cS			
R1	x		x	x	X	x	x	x				x	x	x									x	x																					
R2				x	x							x	x																																
R3																																													
R4				x		x			x			x	x	x					x					x	x																				
R5																																													
R6				x		x			x			x	x						x					x																					
R7				x		x			x			x	x						x					x																					
R8				x		x		x																																					
R9																																													
R10				x		x		x	x		x								x																										
R11																																													
R12																																													
R13																																													
R14																																													
R15						x		x	x		x	x	x	x	x	x	x	x	x	x																									
R16				x		x		x				x	x	x																															
R17																																													
R18																																													
R19																																													
R20						x																																							
R21						x																																							
R22																																													
R23																																													
R24																																													
R25																																													
R26																																													
R27																																													
R28																																													
R29																																													
R30																																													

Figure 45: CPMS sets prices of power supply

6 Implementation, integration and test plan

In this section, the “Implementation Plan” and the “Integration and Test Plan” are presented for each of the two sub-systems: eMSP and CPMS. A similar approach will be followed for both.

6.1 Implementation plan

After having a preliminary design with each sub-system divided into its components, the methods that their interfaces expose, and the runtime view of the system, the implementation step becomes straightforward. The implementation of each sub-system will be based on its respective component diagram. The general implementation will follow a **thread strategy approach** by implementing first the portions of code needed to satisfy each of the runtime use cases exposed in the “Runtime view” section, where each case will correspond to a feature that should be mapped to one or in some cases several endpoints to be developed. As a result, the implementation of the sub-systems will be characterized by the implementation of endpoints, allowing expedited external visibility of the process and delivering faster a Minimum Viable Product (MVP). By combining the **thread** strategy with the **critical modules** approach, we can identify the critical threads (endpoints in our case). Therefore, the basic functionality will be implemented first, and the smart features will be implemented last. This combination of approaches fits in large and complex subsystems.

Some important notes are:

- Each sub-system (eMSP and CPMS) can be developed in parallel to each other.
- Since some endpoints involve accessing different methods from different services, all the domain services can be developed in parallel.
- Each domain service is composed of Controller-Services-Repository, here we will follow a bottom-up strategy for most of the cases starting from the Repository, passing through the Services (including External Services) and Controllers, and ending with the ClientApplication itself aka front-end.
- The services will be implemented incrementally, method by method according to what methods a specific endpoint (functionality inspired by use cases) calls.

The components should be implemented in the following order:

1. **DBMS and Repository:** aka Persistence layer. Here the database tables and the Repositories needed for the endpoint are created. The repositories are the gateway between the business layer and the database, where domain data is mapped to database entities.
2. **Services (also external):** Here is where the business logic resides. The services are the middle layer between the controller and repositories.
3. **Controllers:** The controllers oversee handling the HTTP requests to an endpoint, transforming the user input by passing it through the services.
4. **ClientApplication:** This is the front end where the user interacts with the system. This component is only in charge of the presentation layer.

6.2 Integration and test plan

The integration of the components will follow the same order as the one from the implementation described in the previous section. That is, we will follow a **thread + critical modules** strategy for integration and testing of each endpoint. To this end, we will follow in most cases a bottom-up approach for incrementally testing each component needed for the endpoint, trying to integrate and test each of the components as soon as they are finished and all unit tests run smoothly, starting with the repository, and finishing with the ClientApplication. By finished, we mean that the component has the bare minimum portion of code needed to execute the endpoint successfully.

Furthermore, not only a bottom-up strategy will be used but also some sort of top-down approach. It will be used in integration tests that require to access external (third-party) services by creating stubs or mocks of them e.g., calendar, vehicle, payment services, DB accesses. In this way, we reduce the cost in which we might incur if we use those services continuously during testing. Considering that there could be hundreds of testing cases where access to those services is needed, so running them could be reasonably costly. In the following, we will show different diagrams that explain how to run the integration tests. The graphics will depict the Booking endpoint but for the rest of the endpoints, a similar approach can be followed for both the e-MSP and the CPMS.

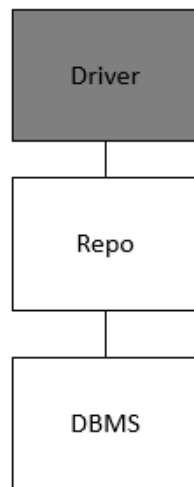


Figure 46: Integration test of Repository

Here the mapping between domain entities in the programming language of choice to the database entities is tested. Operations such as save, update, and delete are performed.

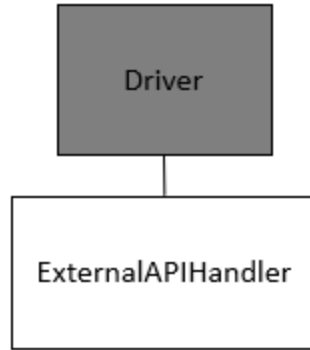


Figure 47: Integration test of External APIs

Examples of external APIs are the ExternalVehicleManufacturer-, the Calendar-, the ExternalPayment-, the ExternalPushUpNotification, and the ExternalDSO services. We will test that the functionalities that we need from them are provided in their APIs and were implemented correctly.

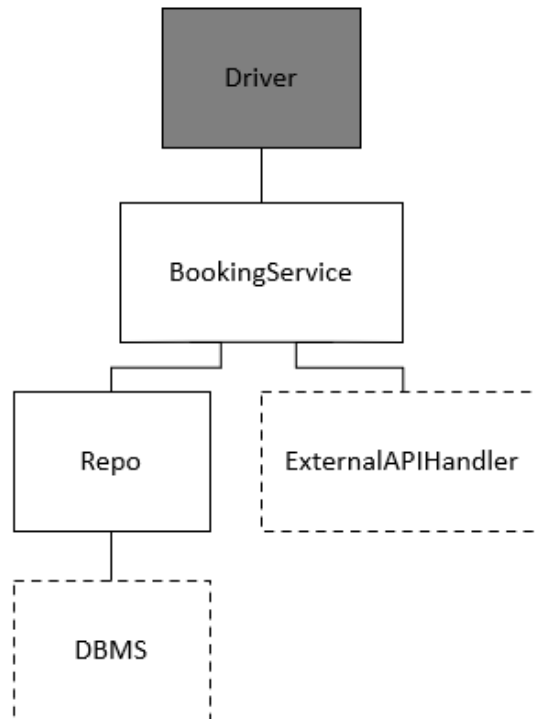


Figure 48: Integration test of Service (Booking example)

Here it is not needed to have the full functionality of the service. Only the portion needed to

run a particular endpoint. The dashed rectangles indicate that those modules are stubbed. We stub the DBMS and the ExternalAPIHandler while testing services to reduce the possible costs incurred and the delay added when calling those operations.

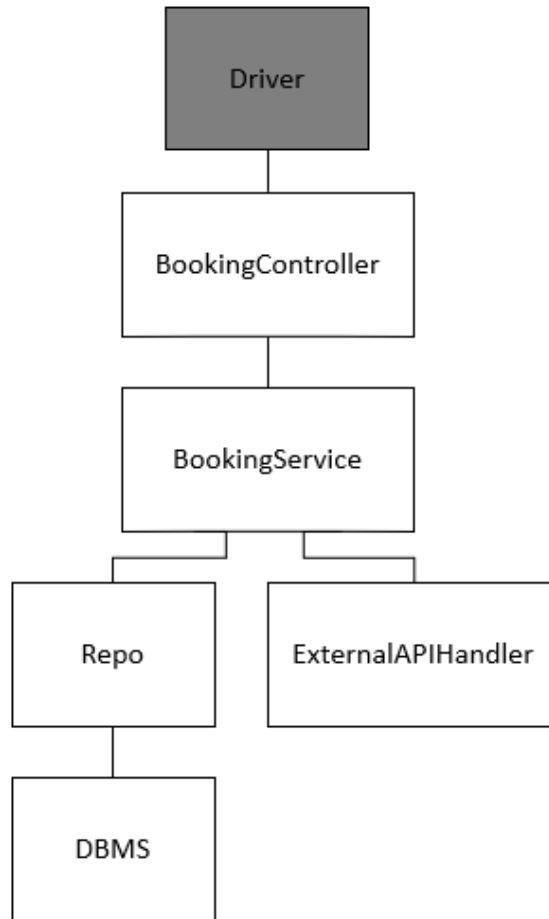


Figure 49: Integration test of Controller (Booking example)

Other controllers would lie parallel to Booking and all depend on their corresponding Services and Repository components. Some would also depend on one or more external services. In this stage tools to test endpoints such as Postman can be used.

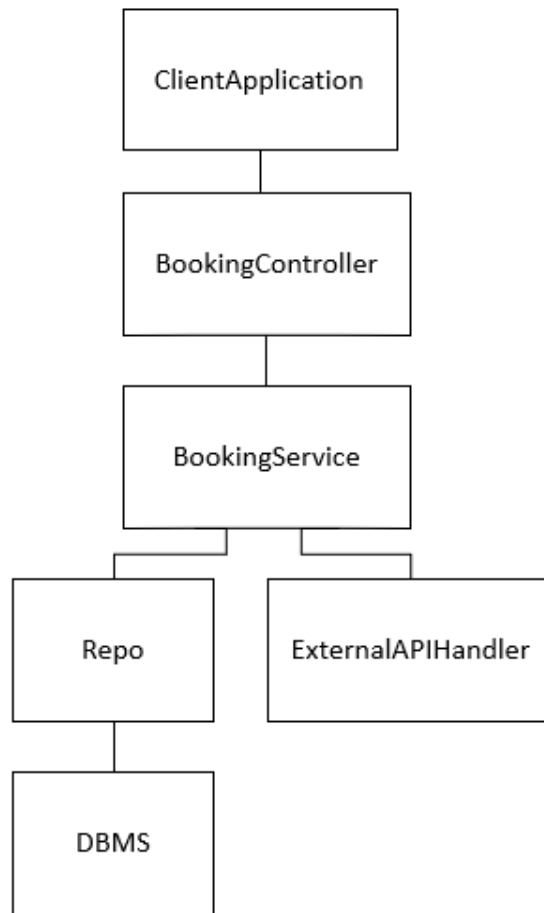


Figure 50: Integration test of ClientApp (Booking example)

Full integration of the front-end and the back-end. This is the end result that will be delivered to production and can be used by final users to do validation of the system.

After integrating the system, system testing can be performed in order to test functional and non-functional requirements.

7 Reference Documents

References

- [1] *Bluetooth Official Website*. <https://www.bluetooth.com/>. [Online; accessed 08-January-2023].

- [2] *JSON Web Token*. <https://jwt.io/>.
- [3] *Previous Document - RASD: e-Mall Problem*. <https://github.com/claualc/BazLlorenteQuintero.git>.
- [4] *Requirement Engineering and Design Project: goal, schedule, and rules*. 2022.

8 Effort spent

Task	Time spent
Introduction	0.5
Team Discussion	6 h
Architectural design	36.5h
Requirements Traceability	3h
Total	46h

Table 9: 10916443 - Claudia Baz Alvarez

Task	Time spent
Introduction	1
Team Discussion	6 h
Architectural design	30 h
Implementation, integration and test plan	8 h
Total	45 h

Table 10: 10832704 - David Alfredo Quintero Olaya

Task	Time spent
Team Discussion	1 h
User Interface Design	35 h
Total	36 h

Table 11: 10904511 - Pablo Llorente Cerezuela