e-Mobility for ALL

Design Document

Abbondanza Alessia - 995959
Campo Marco Lorenzo - 103213
De Luca Alessandro - 103542

December 2022

Contents

1 Introduction **4**
- 1.1 Purpose . . . 4
- 1.2 Scope . . . 4
- 1.3 Definitions, Acronyms . . . 5
  - 1.3.1 Definitions . . . 5
  - 1.3.2 Acronyms . . . 5
- 1.4 Revision History . . . 6
- 1.5 Document Structure . . . 6

2 Architectural Design **7**
- 2.1 Overview: High-level components and interaction . . . 7
  - 2.1.1 High-Level Component View . . . 8
- 2.2 Component View . . . 9
  - 2.2.1 Component Diagram . . . 9
  - 2.2.2 More on communication protocols . . . 16
  - 2.2.3 Additional Specification . . . 19
- 2.3 Deployment view . . . 20
  - 2.3.1 More on deployment choices . . . 22
- 2.4 Runtime view . . . 23
- 2.5 Component interfaces . . . 29
- 2.6 Selected architectural styles and patterns . . . 31

3 User Interface Design **33**
- 3.1 Client User Interface . . . 33
- 3.2 CPO User Interface . . . 35

4 Requirements Traceability **36**
- 4.1 e-MALL eMSP Goals mapping . . . 36
- 4.2 e-Mall CPMS Goals mapping . . . 39

5 Implementation, Integration and Test Plan **42**
5.1 Implementation plan 42
5.2 Integration plan 44
5.2.1 eMSP integration plan 44
5.2.2 CPMS integration plan 49
5.2.3 Subsystems integration plan 53
5.3 System testing 54

6 Effort Spent **55**
6.1 Effort required by members 55

7 RASD references **57**
7.1 eMSP Goals 57
7.2 CPMS Goals 57
7.3 Requirements 58

8 References **61**

Chapter 1

Introduction

1.1 Purpose

This is a design document (DD). The goal of this document is to provide a functional description of the e-MALL system previously presented in the Requirements Analysis and Specification Document (RASD). This document is intended for the development team and provides a detailed overview of the major components and their interaction. In addition, the design decisions regarding the architecture are analyzed to provide developers with information about the architecture chosen (client-server) and the software and hardware components used to build the system, taking into account availability, reliability, performance, and security. Data management is also briefly discussed, in terms of the database chosen. The design features contained in this document serve as a guide for the implementation, integration, and testing of the e-MALL system. The main features of this DD are as follows:

1. High-level system architecture
2. Main system components
3. Interfaces provided by most important components
4. Design patterns adopted

The decision making process involved in the design of this distributed system is explained in depth throughout the document.

1.2 Scope

The full list of assumptions and requirements can be found in the associated RASD, but it can be summarized as follows: The e-MALL system should allow users to book, manage, and perform electric charges at registered stations and provide charging station operators

with tools to manually manage these stations and interact with energy providers. The system also uses information provided by electric vehicles and stored data to provide users with an advanced feature for personalized charging plans.
As previously discussed in the RASD, e-MALL will be analyzed as two separate and communicating subsystems which both have specific properties. Both e-MALL CPMS and eMSP will share a few aspects of the architecture choices, but the implementation will differ heavily.
The system is designed for a very large number of users (as described in more detail in the RASD) distributed over a large region, and therefore needs to be distributed to provide fast and reliable access to the data for all connected users.

1.3 Definitions, Acronyms

1.3.1 Definitions

- **e-MALL**: e-Mobility for All, software that will be analyzed in detail in this document
- **eMSP**: e-Mobility Service Providers, software to manage the interaction between users and charging stations
- **CPOs**: Charge Point Operators, who physically sets up the station and operates it. They interact with a CPMS.
- **CPMS**: Charge Point Management System, software used by CPOs to manage the charging station and the operations that take place there.
- **DSO**: Distribution System Operator, outside organization that provides the energy.

1.3.2 Acronyms

- **API**: Application Programming Interface
- **GPS**: Global Positioning System
- **MVC**: Model-View-Controller pattern
- **HTTP**: HyperText Transfer Protocol
- **SSL**: Secure Sockets Layer
- **JSON**: JavaScript Object Notation

- **EV**: Electric Vehicles
- **OCPI**: Open Charge Point Interface
- **OCPP**: Open Charge Point Protocol

1.4 Revision History

- Version 1.0 | 1.8.2023

1.5 Document Structure

1. **Introduction**: This chapter introduces the reader to the concept of the e-MALL system, highlighting the purpose and scope of DD. The section also includes the terms and acronyms used throughout the report.
2. **Architectural Design**: This chapter contains the architectural design decisions made during the development of the e-MALL system with respect to the software and hardware elements used. The Runtime View and Component Interfaces sections contain the main functions provided by the interfaces and the processes that use them.
3. **User Interface Design**: This chapter presents some mockups of the user interface in special sections, multiple for each user type if required.
4. **Requirements Traceability**: This chapter presents the mapping between the requirements defined in the RASD and the architectural elements chosen.
5. **Implementation, Integration and Testing**: This chapter contains plans for implementing, integrating, and testing the various components of the system presented in this document.
6. **Effort Spent**: Provides an overview of the effort spent by each group member in order to complete the DD.

Chapter 2

Architectural Design

2.1 Overview: High-level components and interaction

The e-MALL System adopts a three-tiered client-server architecture, applied differently for each subsystem.
The decision to use a three-tier architecture was made to separate the business logic of the system from the data. The data itself could more usefully be used for other applications and modules in the future, and decoupling facilitates this.
The system has also been designed to be updated and integrated with multiple functions to follow the trend towards new energy sources and technologies. Therefore, a decoupled architecture can help in this regard.
Additionally, it provides one more layer of security, as all data access passes through the middle layer.
The architecture consists in the following layers:

L1: **Presentation layer**: This layer is responsible for the interaction with the user. It contains all the interfaces needed for users and CPOs to communicate with the logic of the system;

L2: **Application layer**: This layer processes the business logic for the application and acts as a bridge between the other two layers;

L3: **Data layer**: This layer is a Database Management System (DBMS) and provides access to the application data. It provides an interface to the Application Layer and stores all relevant data for both user presentation and correct functioning of the back end system.

2.1.1 High-Level Component View

The architecture described above is implemented using three-tier hardware, with each tier associated with a layer. The following image summarizes the chosen architectural style.

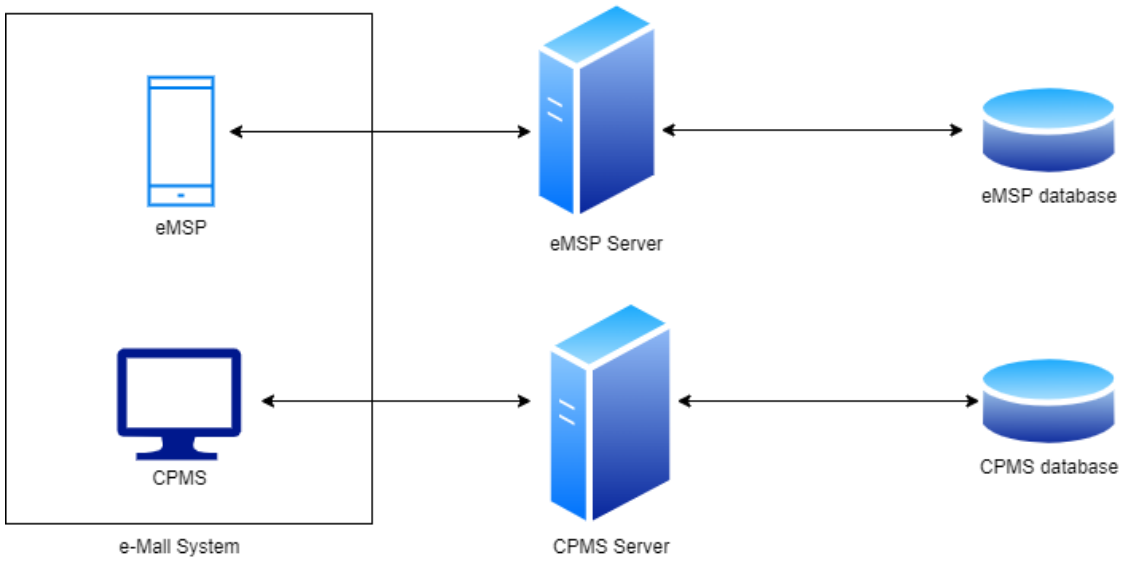

Figure 2.1: Three-tiered hardware architecture

T1: **Client**: Provides an interface for interacting with the system. It can be provided via a desktop (CPMS) or a smartphone (eMSP);

T2: **Application server**: Processes requests provided from the client and provides business logic;

T3: **Data Source**: Contains both the DBMS and any external data sources. Different data sources are queried from eMSP or CPMS servers.

The devices used by different users (desktop or smartphones) correspond to the client level of the hardware architecture. All devices are connected to the correspondent application server. The latter represents the server layer of the architecture and provides interaction with DBMS and (if demanded) external APIs.

2.2 Component View

2.2.1 Component Diagram

The component diagram below describes the organization and wiring of the internal structure of the application server, which contains the business logic of the software, and the interaction between the various components.

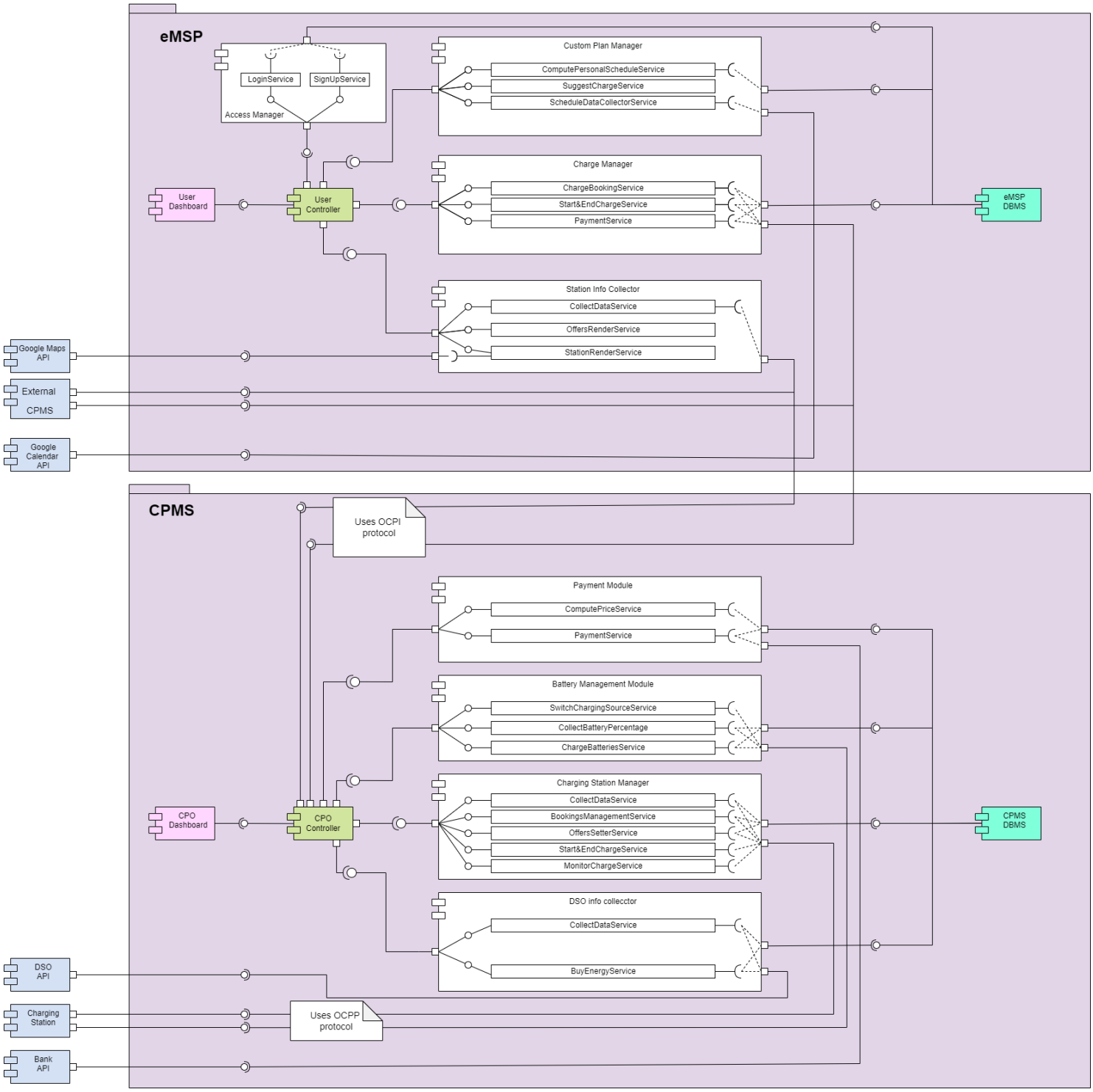

Figure 2.2: Component diagram of the whole e-MALL system

eMSP Component Diagram

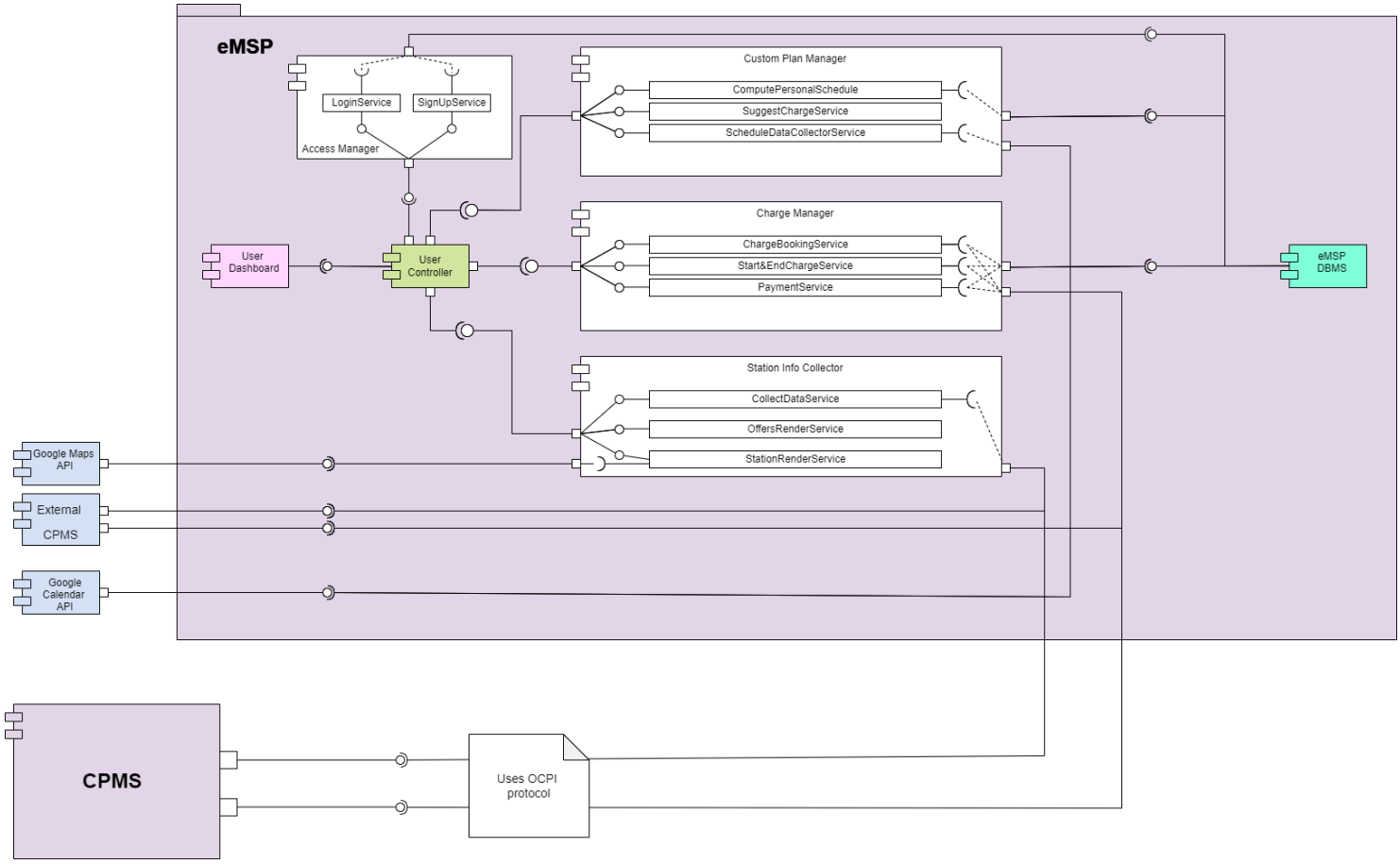

Figure 2.3: Component diagram of the e-MALL eMSP

C1. **Google Maps API**: APIs used by the e-MALL eMSP to get the rendering of a specific geographical area

C2. **Google Calendar API**: APIs used by the e-MALL eMSP to get the user's personal schedule

C3. **External CPMS**: Contact interface provided by external CPMS. e-MALL eMSP can communicate with them in order to provide better coverage

C4. **User Dashboard**: Component that allows users to interact with the eMALL system, provided by a GUI

C5. **User Controller**: Component that redirects the user's input to the correct module depending on the user's actions

C6. **Access Manager**: Component that manages sign up and sign in actions

C7. **eMSP DBMS**: Database management system that supports the user side of the system. This component collects data to provide both a more personalized experience and less communication with external components when they are not needed.

C8. **Custom Plan Manager**: This component is responsible for creating and managing the "smart schedule" advanced feature offered by the e-MALL system

- C8.1 **ScheduleDataCollectorService**: Service responsible for querying the Google Calendar API to obtain the data about the user's personal schedule
- C8.2 **SuggestChargeService**: Service Responsible for notifying the user when a recharge is needed (a custom plan is being followed, a new offer has been published, the estimated battery level is low)
- C8.3 **ComputePersonalScheduleService**: Service that implements logic to adapt a charging schedule to the user's schedule. This service takes into account the usual routes and activities repeated during the week in order to create the most feasible loading schedule possible

C9 **Charge Manager**: This component is responsible of managing the charging process, from its booking to the payment

- C9.1 **ChargeBookingService**: This service takes user input and notifies the CPMS that a charge is requested at a specific charging station for a specific charging location. After confirmation by the CPMS, it sends the result to the user, either "accepted" or "denied"
- C9.2 **Start&EndChargeService**: This service is responsible of actually starting and ending the charging process
- C9.3 **PaymentService**: This service is responsible for providing the user with the price of a charge, receiving the payment information, and returning it to CPMS for processing

C10 **Station Info Collector**: This component is responsible of showing the user all the information relative to the charging stations

- C10.1 **CollectDataService**: This service connects to CPMS to get information about the external status of charging stations (availability of charging spots, time needed for a spot to become available, type of charging spots available, GPS coordinates of the charging station, available offers)
- C10.2 **OffersRenderService**: This service is responsible of rendering all the available offers

C10.3 **StationRenderService**: This service interfaces with the Google Maps API and information collected by the CollectDataService to display maps of charging stations and provide the corresponding information

CPMS Component Diagram

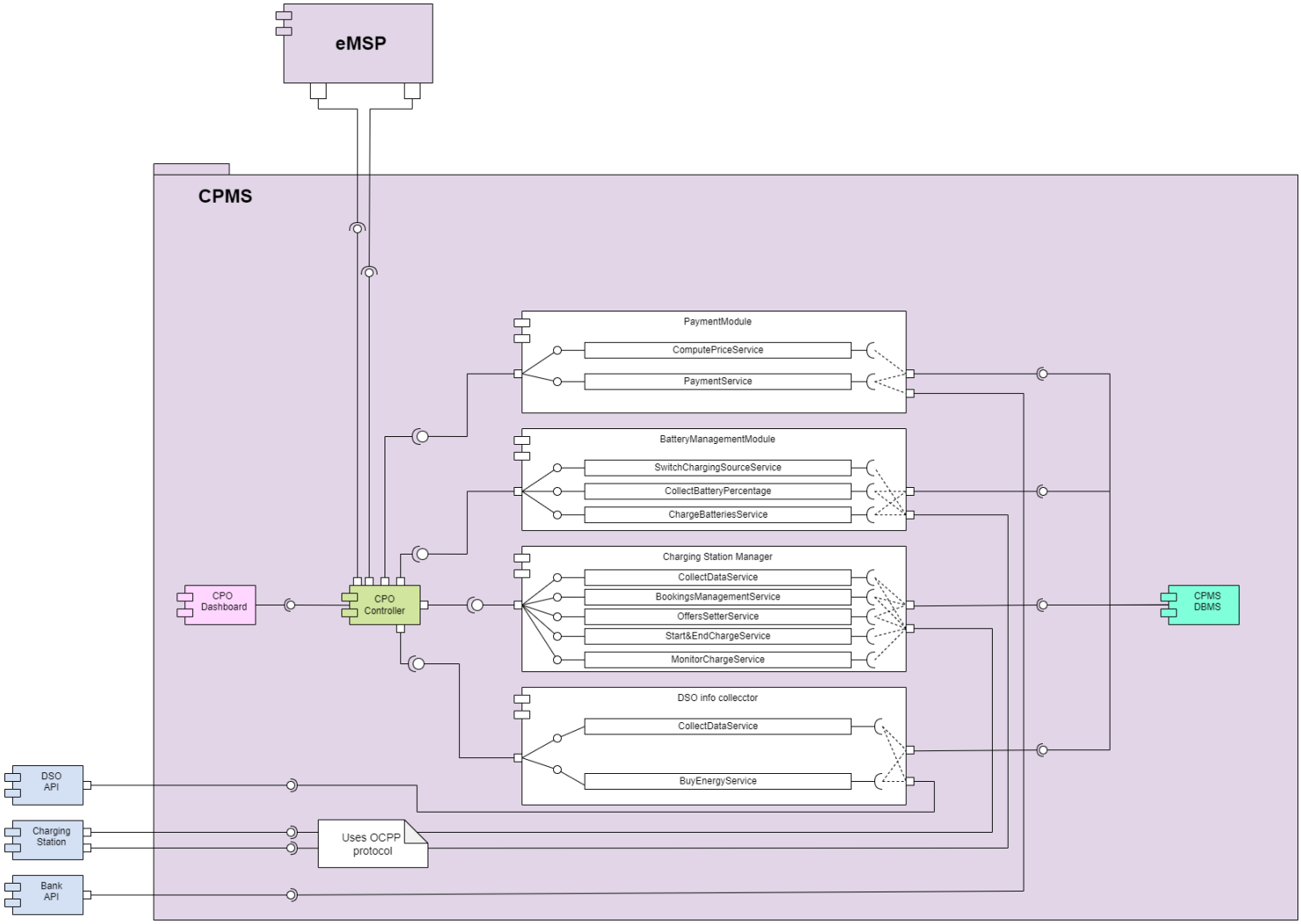

Figure 2.4: Component diagram of the e-MALL CPMS

C1 **DSO API**:API to retrieve information about energy prices and purchase energy from a DSO

C2 **Charging Station**: interface of the charging station that sends information to the CPMS via certain communication protocols

C3 **Bank API**: API of the bank used to validate and execute payments

C4 **CPO Dashboard**: component that allows CPOs to interact with the CPMS and charging stations

C5 **CPO Controller**: controller used to route input from CPO and eMSP to the correct modules of the CPMS

C6 **CPMS DBMS** : Database management system that supports the CPMS side of the system. This component collects data both for market analysis and diagnostics and to reduce communication with external components when not needed

C7 **payment module**: This module is responsible for handling all payment-related tasks

- C7.1 **ComputePriceService**: This service checks all information about an executed load (energy prices, type of load, time, amount of consumed energy) and calculates the final price of this load
- C7.2 **PaymentService**: This service receives the user's payment information, sends it to the bank for verification, and responds to the user whether the transaction was approved or not

C8 **battery management module**: This module manages all functions related to the additional batteries present in a charging station

- C8.1 **SwitchChargingSourceService**: Service to switch the energy source from that provided by the DSO to that stored in the charging station batteries and vice versa
- C8.2 **CollectBatteryPercentage**: Service that monitors the remaining energy in the charging station batteries
- C8.3 **ChargeBatteriesService**: Service used to initiate the charging of the batteries

C9 **Charging Station Manager** : Module that manages all functions related to the charging station

- C9.1 **CollectDataService**: Service that collects all the information a charging station needs to know its internals (number of vehicles to be charged, amount of energy absorbed by each vehicle to be charged, time remaining until the end of the charging process)
- C9.2 **BookingManagementService**: Service that manages bookings, validates booking requests (checks that there are no overlaps) and correctly classifies the specified time slot as occupied for the user who booked it, otherwise responds with an "input not valid" message
- C9.3 **OffersSetterService**: Service that allows CPO to post an offer
- C9.4 **Start&EndChargeService**: Service that takes user input to startand stop a charging process and forwards it to the correct charging location

C9.5 **MonitorChargeService**: Service that monitors the charging status of a vehicle connected to a time slot

C10 **DSO Info Collector**: Moduele responsible for managing communication with DSOs

C10.1 **CollectDataService**: Service that connects to a DSO to collect information about the energy such as price, means of production, and availability

C10.2 **BuyEnergyService**: Service that contacts a DSO to purchase energy from it

2.2.2 More on communication protocols

As described in the RASD, the system implements the three main communication protocols: OCPP, OCPI, DIN/ISO. These are the *state of the art* in electrical charges.

Future assumpions

For simplicity, all messages exchanged between eMSP and CPMS are sent in the format required by OCPI and all messages exchanged between CPMS and EV charger are sent in the format required by OCPP, even if this isn't specified. Also the message exchange between EV charger and CPMS isn't shown in all provided diagrams, because the OCPP communication protocol is very straightforward and the notation can easily become redundant and cumbersome.

OCPI Protocol

The OCPI implementation used by e-MALL is version 2.2, and the implemented architecture is *Hub and Peer-to-Peer*. Not all Platforms will only communicate via a Hub. There might be different reasons for Platforms to still have peer-to-peer connections or the Hub might not yet support new functionalities. This also allows for better routing and control over the OCPI message exchange.

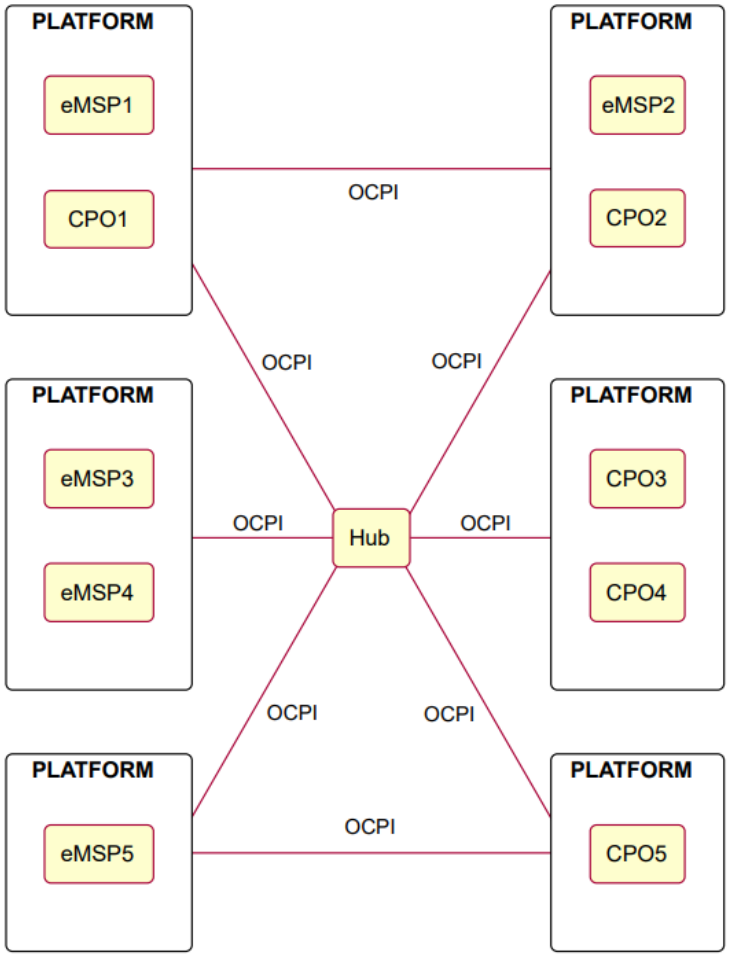

Figure 2.5: Platforms connected via a Hub and directly topology example

Message routing functionality also makes it possible to support platforms that host multiple roles. Many of the stakeholders are not just CPO or eMSP but both. Some parties have a platform where they host services for other CPOs/eMSPs. Some providers are CPO themselves and provide CPO services to others, while other providers (themselves) do not play a CPO or other role in the EV charging landscape, but provide services to CPOs/eMSPs.
For this reason, having a very versatile infrastructure can benefit the system, especially considering future functionalities that may need to be implemented.

The OCPI protocol is based on HTTP, with SSL and server-side token-based authentication, and uses the JSON format. It follows a RESTful architecture for web services where possible.

OCPP Protocol

e-MALL has implemented OCPP 1.6J version, which means using JSON over WebSocket.
The OCPP, Open Charge Point Protocol, is the standardized framework for communication between EV chargers, Electric Vehicle Supply Equipment (EVSE), and back end software. It sends information to the back end regarding start/stop/status messages for charging sessions, EVSE reservations, and firmware updates.

The protocol was chosen against competitors because it is free to use and hardware-agnostic, making the e-MALL CPMS service's target as big as it can possibly be. Also, OCPP is the most common communication protocol implemented in the EV charging landscape, so frequent patches and new functionalities are to be expected.
The last major update concerns security. In version 1.6, the connection between the electric vehicle supply equipment and the backend of the network required a VPN to encrypt the entire communication channel. Now, data packets are encrypted at the protocol level, and no third party is needed for a secure connection. All big steps towards a secure and functional electrified infrastructure.

The version chosen is the 1.6J which is the most commonly used one. In addition to the features mentioned above, 1.6 enables smart charging, the ability for the Network Operator to set restrictions like power level or time limits on an individual charger, TriggerMessages, allowing the Network to query information from the charger, and Authorized Local Lists, to determine who is allowed to charge on each unit.

2.2.3 Additional Specification

To ensure that the application provides high reliability and availability, the Application Server, which is the server tier, is replicated. Replication provides redundancy in case both the software and hardware fail. Load balancers are available to balance requests and prevent system overload, so that users can have the best possible experience depending on their location.

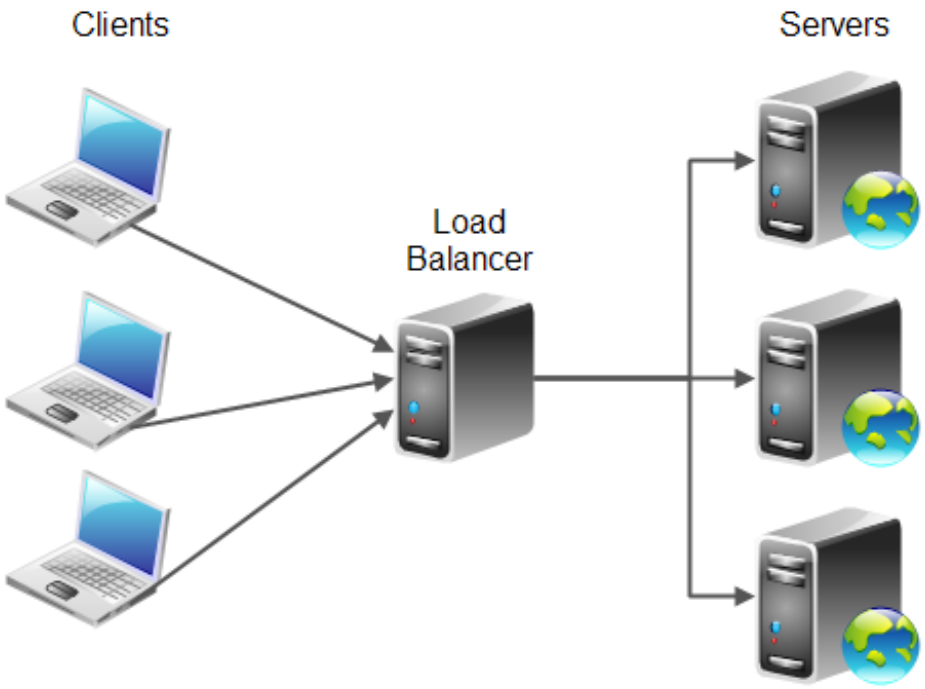

Figure 2.6: *Load Balancer* and *Replication* architecture

To ensure more reliability and higher performance, the use of caches is essential as it will reduces the overall load on the servers. The clients specified in the system architecture are thin clients, as they are designed to interact with the application server and do not provide any additional functionality without this connection. This allows the web application to verify the correctness of the compiled form by executing mobile code, while maintaining a lightweight manner required when dealing with low bandwidth.

2.3 Deployment view

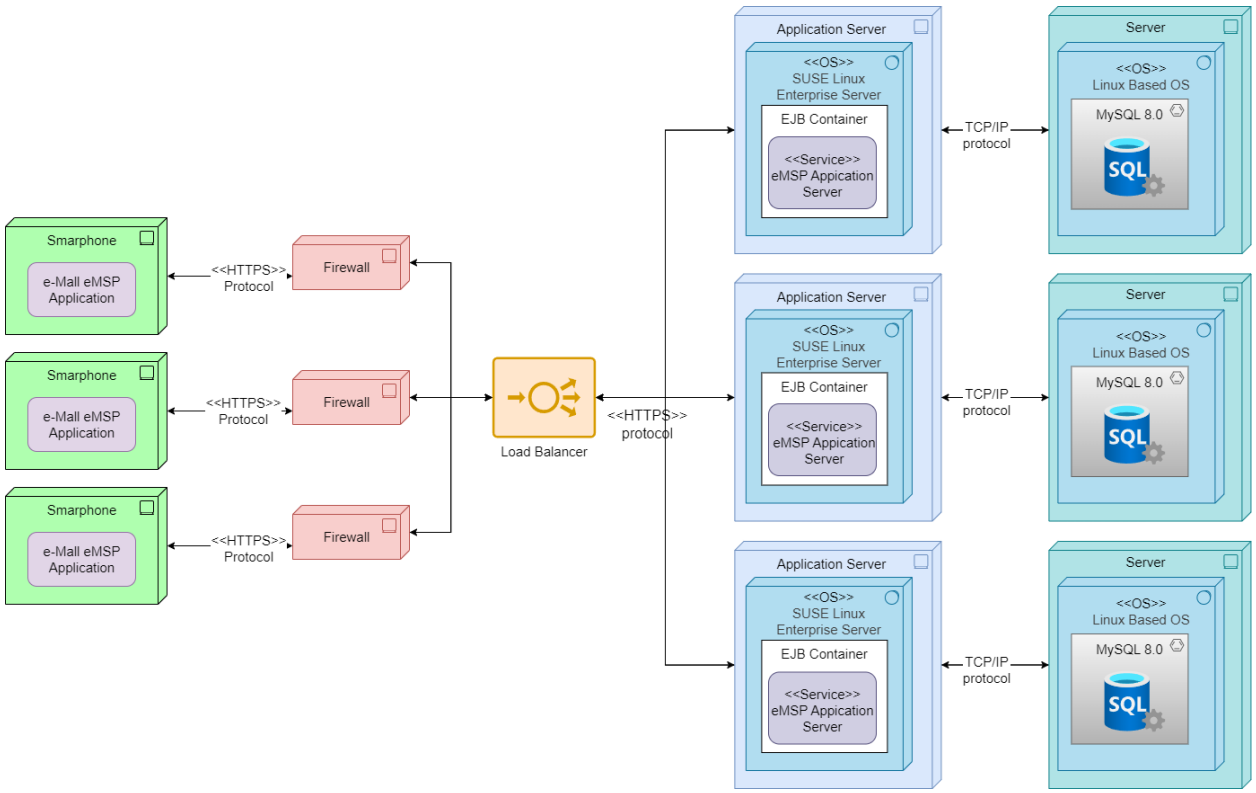

Figure 2.7: Deployment diagram of the eMSP subsystem, three clients and *server redundancy*

Here are the main components of the system:

C1: **Smartphone**: User access device. It is used to connect to the Internet and perform various actions, such as: logging in/registering, booking and managing a charge, viewing the user's charge history, and custom plans. It can also display an overview of charging stations nearby or at a specific location. This is done by connecting to Google Maps' public APIs (which are not covered in this document);

C2: **Firewall**: Usual connection device in all domestic and enterprise networks. Prevents and controls inbound/outbound traffic;

C3: **Load Balancer**: It increases the reliability of the system by managing the distribution of the workload among the different servers;

C4: **Application Server**: Contains the business logic of the application and acts as *middleware* placed between user and DBMS;

C5: **Database Management System**: It serves as an interface between the databases and the services run on the e-MALL application server.

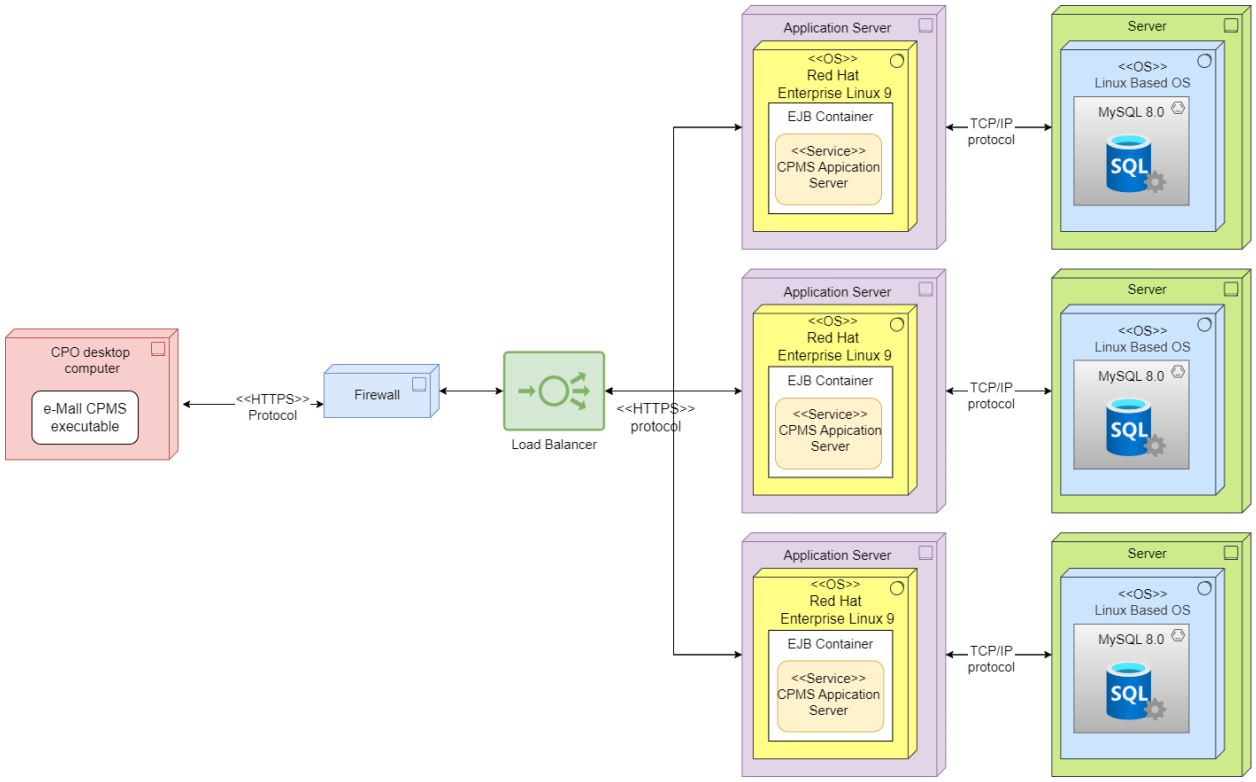

Figure 2.8: Deployment diagram of the CPMS subsystem, one clients and *server replication*

C1: **Desktop computer**: Used by CPOs to connect to the Internet and gain control of the connected charging station. It is also the portal through which operators can advertise a new special offer and connect to DSOs for manual purchase of energy (through the APIs provided);

C2: **Firewall**: *above mentioned*;

C3: **Load Balancer**: *above mentioned*;

C4: **Application Server**: Contains the business logic of the application and acts as *middleware* placed between CPOs and DBMS;

C5: **Database Management System**: It serves as an interface between the databases and the services run on the e-MALL application server.

2.3.1 More on deployment choices

1. SUSE Linux Enterprise Server (SLES) was chosen for the eMSP application server because of its versatility. It is designed for large workloads in big data centers, but also for single-server environments. Also, because it is subscription-based, it gives access to patches, fixes and security updates through the SLES customer portal, making future fixes and management easier, while keeping the system as reliable as possible.

2. The DMBS of choice for implementation is MySQL 8 which can be easily scaled and follows a client-server architecture.

3. Red Hat Enterprise Linux 9 was chosen for the CPMS application server because of its popularity, reliability and constant security patches, updates and support.

4. A relational database was chosen because the data stored is by no means large or aggregated:

   4.1 **eMSP data**:

   i. Usercode and Password
   ii. Bookings data: time frame, location, duration, prices, socket type, etc.
   iii. Custom charging plans
   iv. User's charging history
   v. Payment information:encrypted using the md5 encryption algorithm in order to guarantee all security measures

   4.2 **CPMS data**:

   i. Station's information
   ii. Bookings data: user, time frame, socket type, etc.

   Simple and predictable data types can provide good segmentation and storage, which facilitates maintenance and allows scalability.

2.4 Runtime view

In this section sequence diagrams will be used to describe the interaction between the most important components of the system, in such a way that developers can understand the functionalities to be implemented.

1. **User books a charge**

   This sequence diagram represents the process of a user booking a charge on the e-Mall application. The request is sent to the User Controller, and then to the BookingManagementService. Then it is forwarded to the CPO controller, in particular to the ChargeBookService, that verifies it and sends back the messages of confirmation.

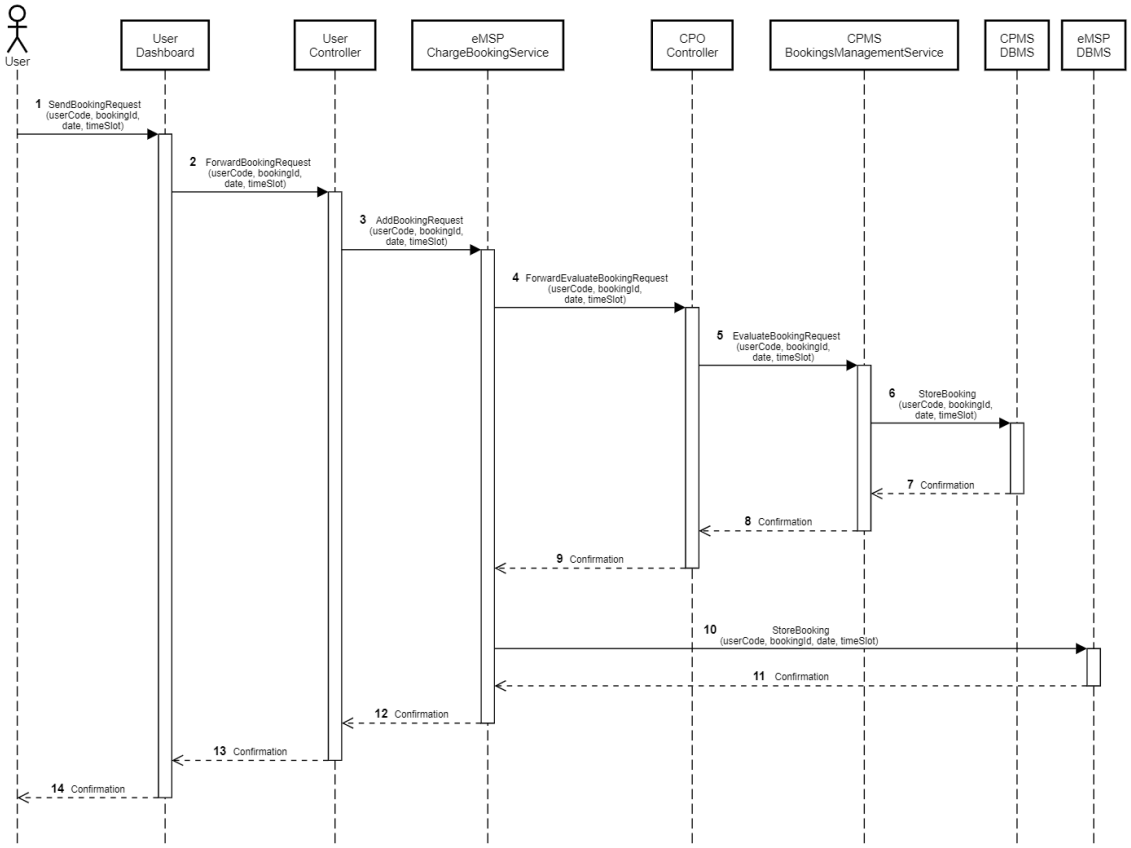

Figure 2.9: Sequence diagram for a user booking a charge

2. **User starts a charge**

The following sequence diagram displays the flow of a user starting a charge of the vehicle, communicating with the eMSP Start&EndChargeService, which, in turn, communicates with the CPMS Start&EndChargeService.

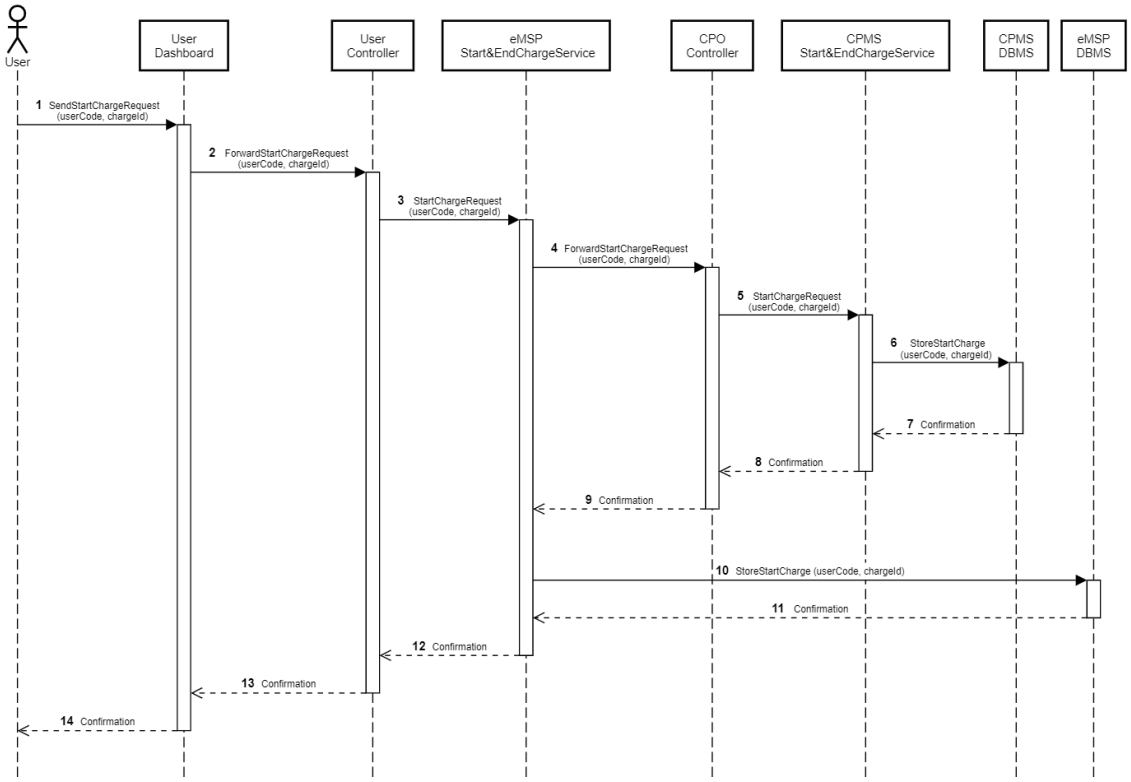

Figure 2.10: Sequence diagram for a user starting a charge

3. **User pays for a charge**

This sequence diagram shows the process of a user paying for a charge, sending the payment request to the eMSP PaymentService, which is forwarded to the CPMS PaymentService that asks for permission to the Bank API. If the payment is verified from the bank, the transaction is successful.

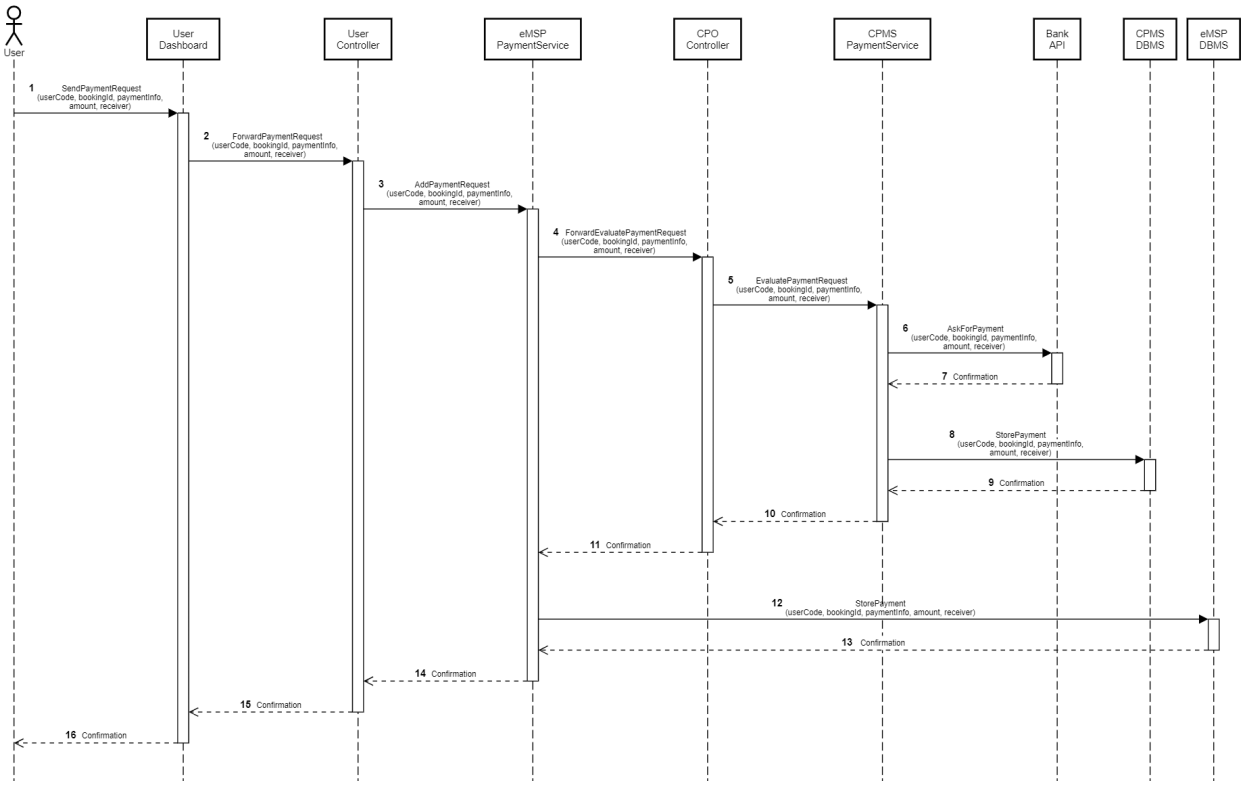

Figure 2.11: Sequence diagram for a user paying for the charge

4. **CPO operator sets a special offer**

   This sequence diagram shows the flow generated when a CPO opertor tries to set a special offer for a specific charging station. The request is sent through the CPMS OfferSetterService, which adds it to the available offers.

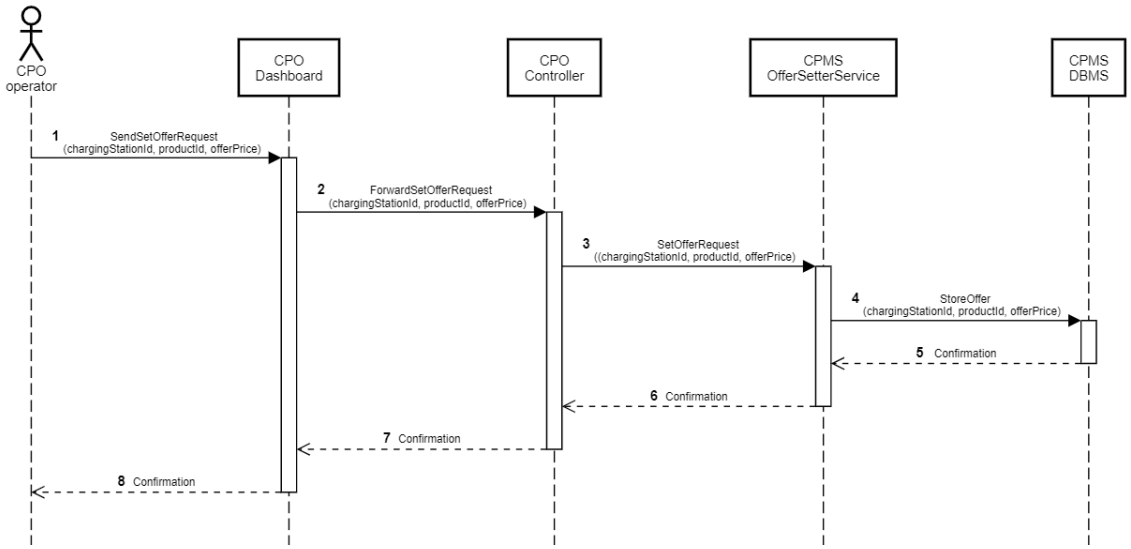

Figure 2.12: Sequence diagram for a CPO operator setting a special offer

5. **User views custom daily plan**

   The following sequence diagram displays the process that happens when a user asks for the custom daily plan. The request of the plan is sent to the Compute-PersonalScheduleService, which, in turn, asks for the calendar of the user to the GoogleCalendar API, and then proceeds to the creation of the plan, that is showed to the user.

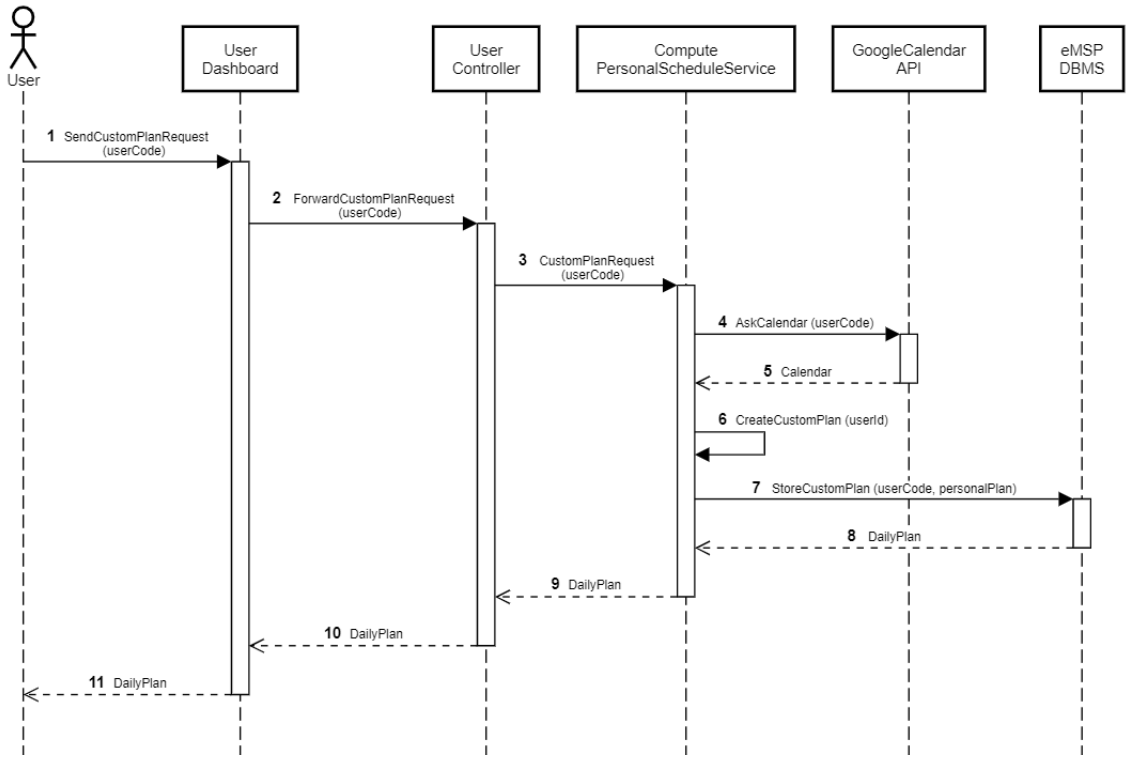

Figure 2.13: Sequence diagram for a user asking for the daily plan

6. **CPO operator buys energy from a DSO**

   This sequence diagram shows the flow of a CPO operator buying a certain quantity of energy from a DSO, forwarding the request to the CPMS BuyEnergyService, that contacts the DSO API to buy the energy.

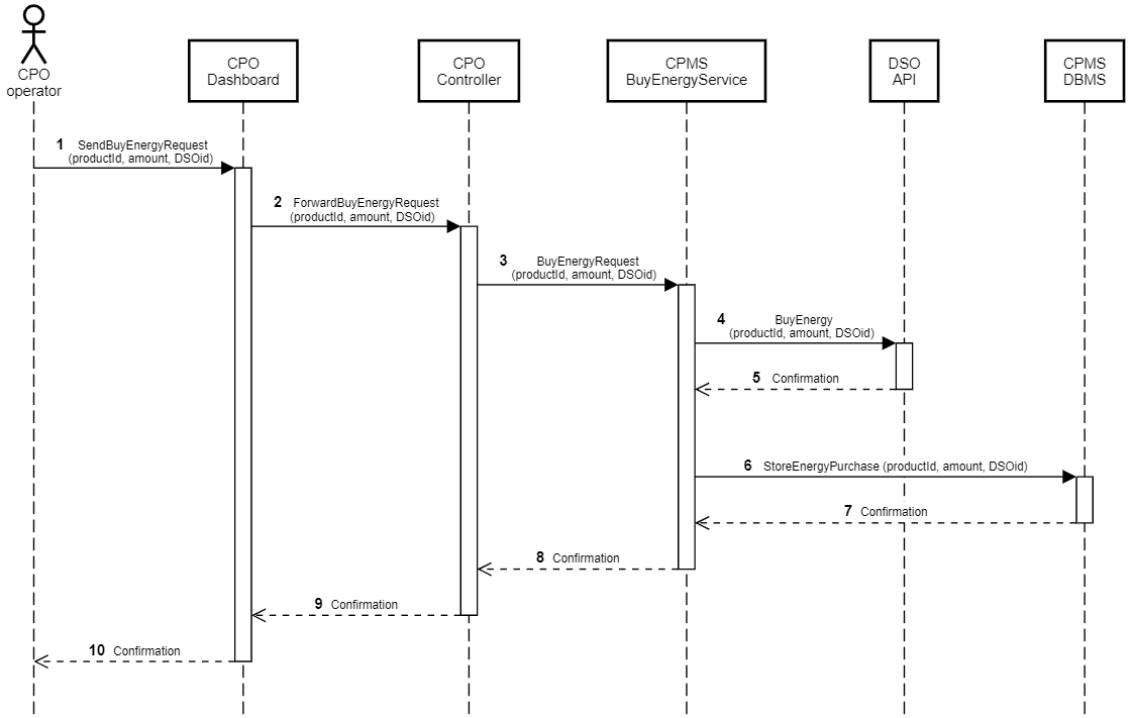

Figure 2.14: Sequence diagram for a CPO operator acquiring energy from a DSO

2.5 Component interfaces

The following diagram shows the main methods of the components interfaces of the system. These methods are useful to give a general idea of the functionalities that each part should provide.

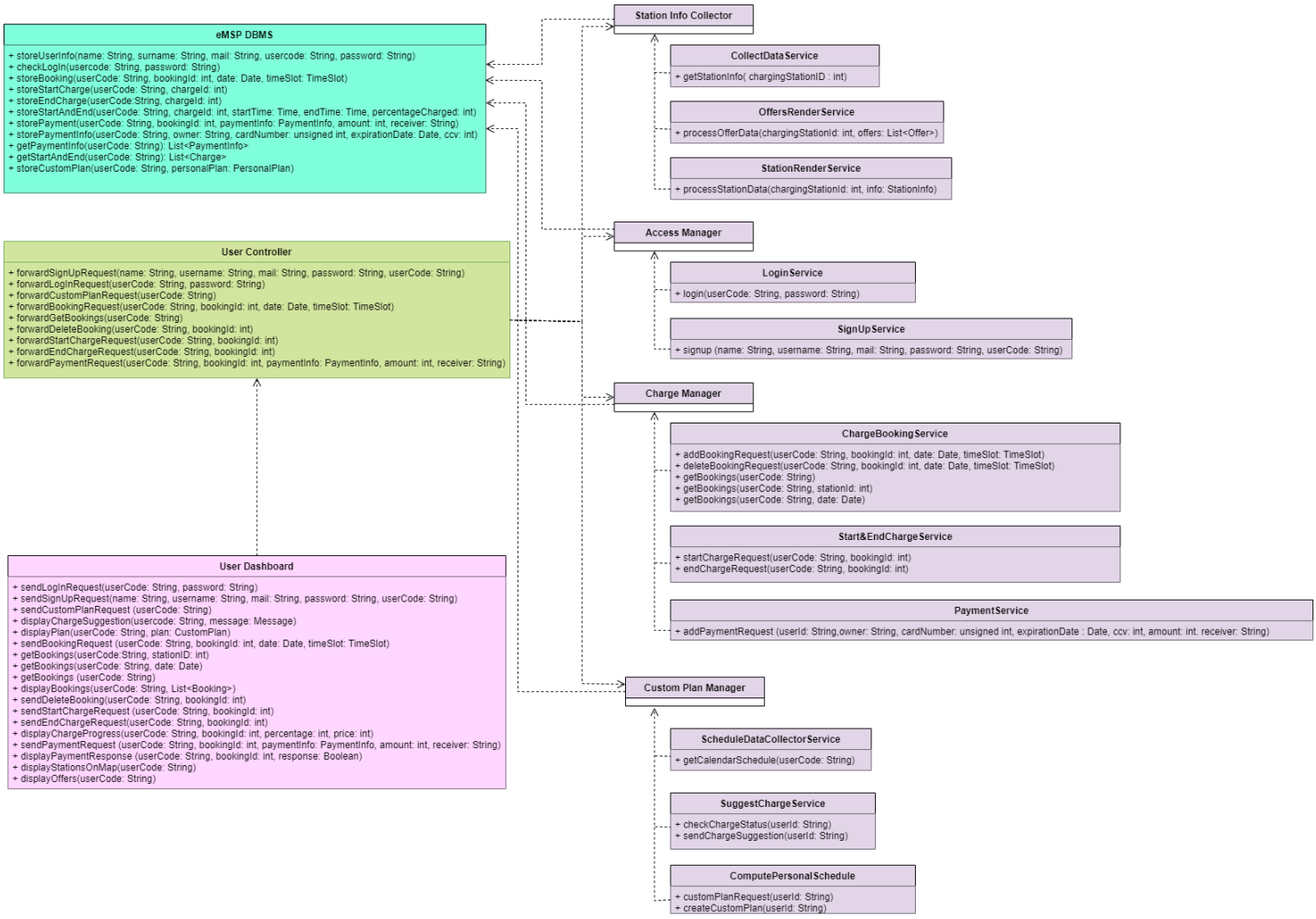

Figure 2.15: eEMSP interfaces

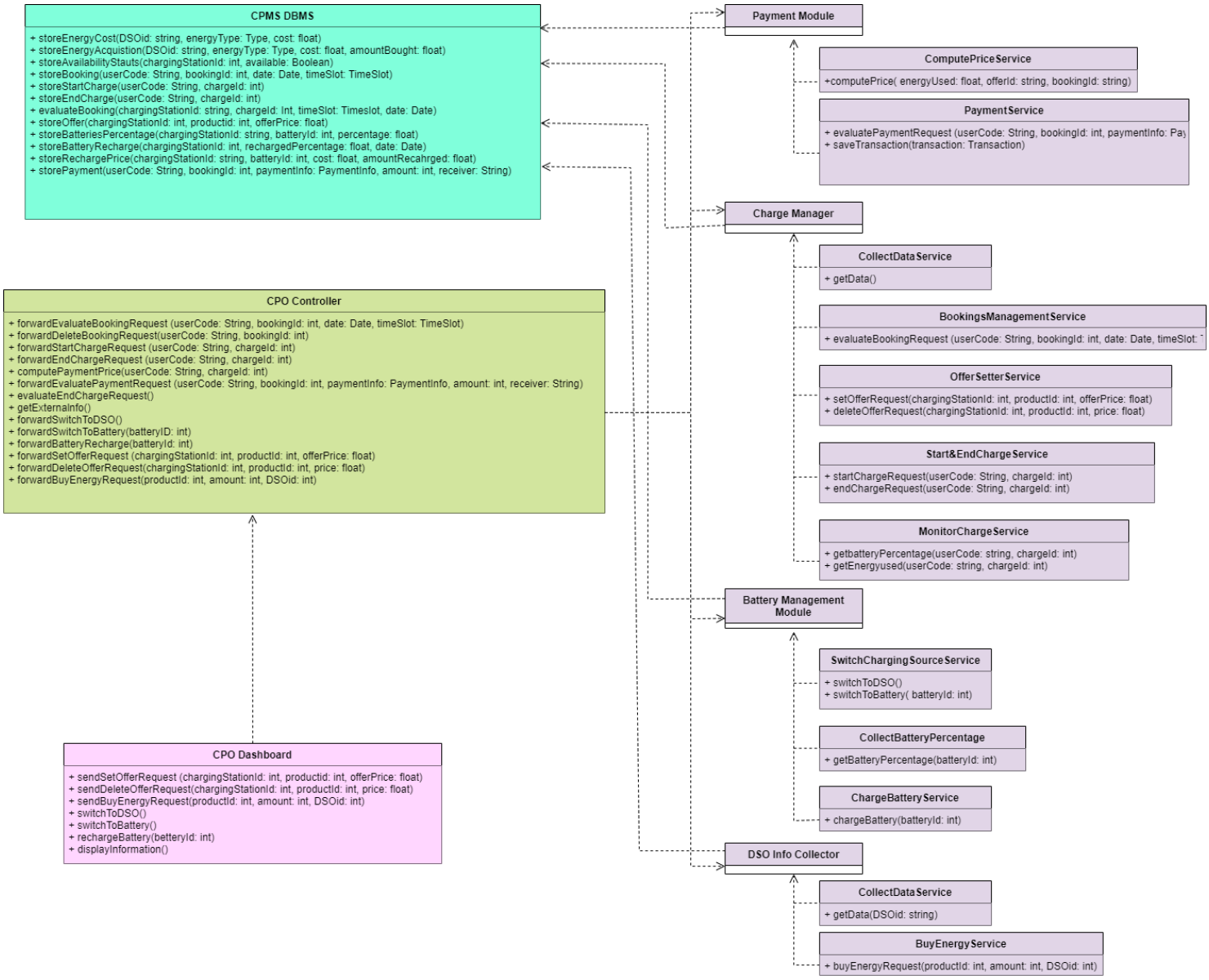

Figure 2.16: CPMS interfaces

2.6 Selected architectural styles and patterns

Client-Server architecture

As mentioned in the architecture overview, a three-tier client/server architecture was chosen. This architectural style offers the advantage that all data is stored in a central location, namely the server, which greatly facilitates the development and maintenance of the application.
Client/server architectures restrict communication to defined interfaces between clients and servers. This restriction simplifies communication between clients and servers, reducing potential vulnerabilities in the architecture. Another advantage of this architectural style is the distinction between three tiers. The three-tier client/server architecture decouples the logic of the data and the logic of the presentation, which increases the security and protection of the data, as well as making the development of the system easier.
Smartphones and computers are also capable of storing small amounts of data in cache memory. Cache memory is used to reduce the number of requests made by the client to the server.

HTTPS protocol

HyperText Transfer Protocol Secure (HTTPS) is the selected protocol for data communication between the components of the system. HTTPS encrypts the data to protect privacy and increase the security of the system to prevent Man In The Middle (MITM) attacks and phishing campaigns. The data transmitted in the network is in a JSON format which is a less verbose language, making communication and debugging easy to manage.

Model-View-Controller pattern

MVC (Model-View-Controller) is a pattern in software design commonly used to implement user interfaces, data, and control logic. It emphasizes the separation between the business logic of the software and the display. This "separation of concerns" provides better division of labor and improved maintenance.
Another important aspect of the MVC pattern is the independence of the three parts, each one can be developed and later integrated with the others.
The three parts of the MVC software design pattern can be described as follows:

- **Model**: Manages data and business logic. Includes DBs, DBMSs, data structures, and services for storing, managing, and processing user inputs.
- **View**: Takes care of the layout and display of the system. Any state-changing action is provided to the rest of the system through the GUIs (Views).

- **Controller**: Relays commands to the model and view parts. Acts as an intermediary between the view and the model, translating the input provided by the user and the model's responses.

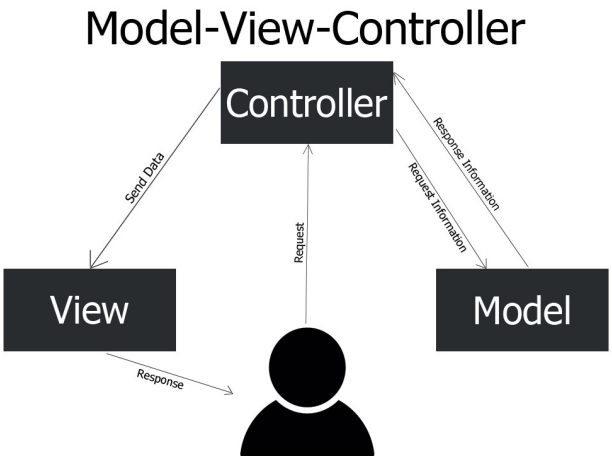

Figure 2.17: *Model-View-Controller* conceptual schema

Chapter 3

User Interface Design

This section focuses on the user interfaces that will be to interact with the system. In particular the most imporant features will be displayed.

3.1 Client User Interface

The images below are a mockup of what to expect by the client-side eMALL application. The features shown are the login interface, the charging station search, the booking, offers, and charge interface and the smart plan feature interface.

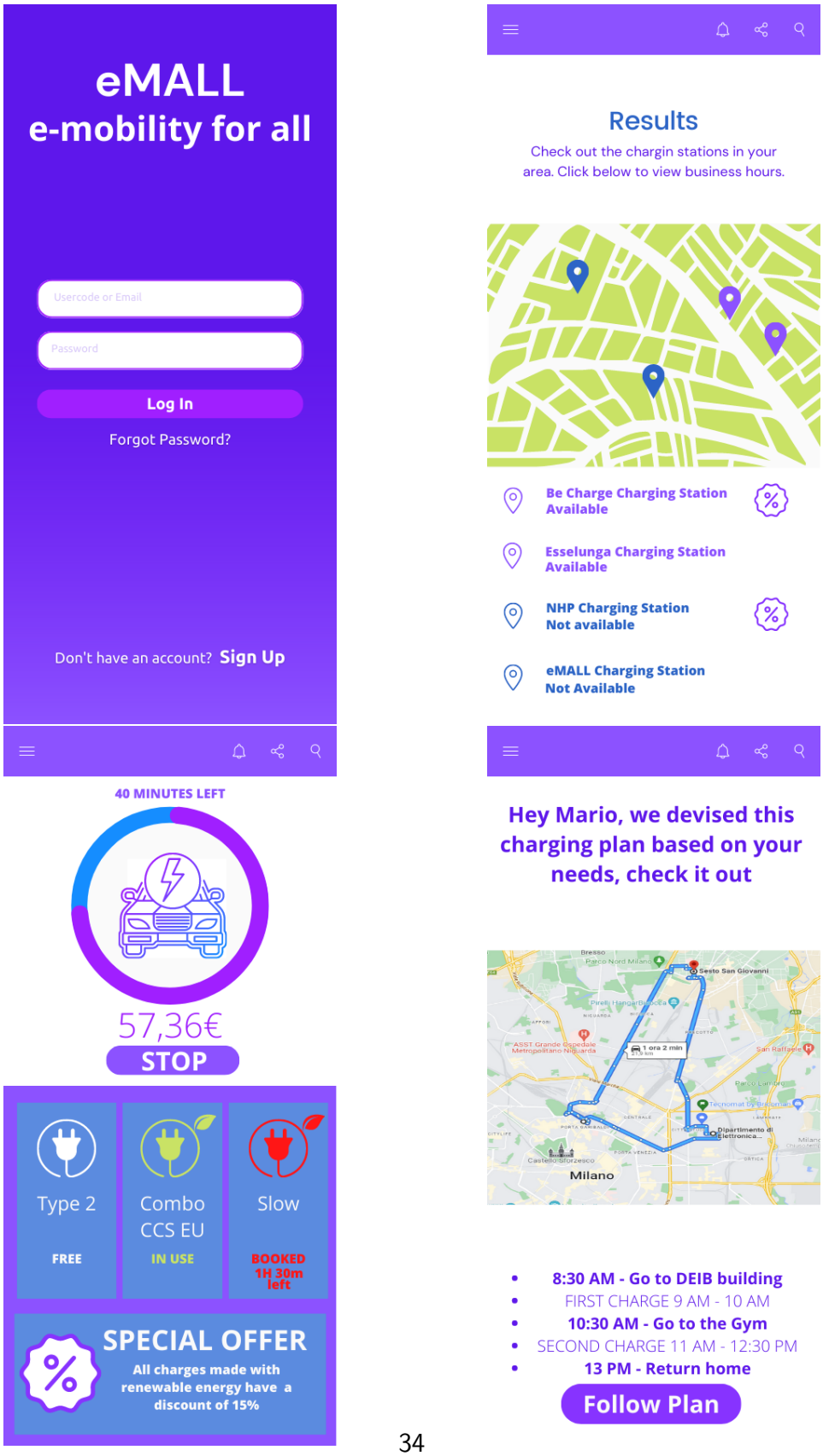

Figure 3.1: User interfaces

3.2 CPO User Interface

The figure below shows a mockup of the CPO Dashboard of the CPMS.

Figure 3.2: CPO interfaces

Chapter 4

Requirements Traceability

4.1 e-MALL eMSP Goals mapping

Mapping of goals and related requirements defined in the RASD and associated requirements to the implemented components:

Goal 1	Allow users to know the location, status, prices and special offers provided by charging stations
	Requirements: R1, R2, R3, R4, R5, R6, R7, R38, R41, R42
	eMSP Components: - User Dashboard - User Controller - Station Info Collector *CPMS Components*: - CPO Controller - Charging Station Manager – CollectDataService – MonitorChargeService - CPMS DBMS

Table 4.1: Mapping of Goal 1

Goal 2	Allow users to manage bookings of a charge in a specific location and time frame
	Requirements: R1, R2, R8, R9
	eMSP Components: - User Dashboard - User Controller - Station Info Collector - ChargeManager – ChargeBookingService - eMSP DBMS *CPMS Components*: - CPO Controller - Charging Station Manager – CollectDataService – BookingsManagementService - CPMS DBMS

Table 4.2: Mapping of Goal 2

Goal 3	Allow users to start, monitor and end the charging process
	Requirements: R1, R2, R10, R11, R12, R13, R30, R39
	eMSP Components: - User Dashboard - User Controller - Station Info Collector - ChargeManager – Start&EndService - eMSP DBMS *CPMS Components*: - CPO Controller - Charging Station Manager – CollectDataService – Start&EndService – MonitorChargeService - CPMS DBMS

Table 4.3: Mapping of Goal 3

Goal 4	Allow users to pay for the obtained service
	Requirements: R1, R2, R14, R15
	eMSP Components: - User Dashboard - User Controller - ChargeManager – PaymentService - eMSP DBMS *CPMS Components*: - CPO Controller - PaymentModule - CPMS DBMS

Table 4.4: Mapping of Goal 4

Goal 5	Suggest users to charge their vehicle based on the charge status, the schedule of the users, special offers provided by the CPOs and availability of the charging slots at the identified charging stations
	Requirements: R1, R2, R17, R18, R19, R20, R 39, R40
	eMSP Components: - User Dashboard - User Controller - Custom Plan Manager - Charge Manager - eMSP DBMS *CPMS Components*: - CPO Controller - Charging Station Manager – CollectDataService – BookingsManagementService

Table 4.5: Mapping of Goal 5

4.2 e-Mall CPMS Goals mapping

Goal 6	Provide a complete "external" overview of the charging station (location, availability, charging options...)
	Requirements: R16, R21, R22, R23, R24, R33, R38, R41, R42
	eMSP Components: - User Dashboard - User Controller - Station Info Collector – CollectDataService *CPMS Components*: - CPO Controller - Charging Station Manager – CollectDataService – MonitorChargeService - CPMS DBMS

Table 4.6: Mapping of Goal 6

Goal 7	Manage and correctly execute a charge for a specific time frame
	Requirements: R16, R21, R22, R23, R24, R25, R26, R27, R28, R29
	eMSP Components: - User Dashboard - User Controller - Station Info Collector – CollectDataService - eMSP DBMS *CPMS Components*: - CPO Controller - Charging Station Manager – CollectDataService – MonitoeChargeService - CPMS DBMS

Table 4.7: Mapping of Goal 7

Goal 8	Know the "internal" status of a charging station
	Requirements: R16, R34, R35
	CPMS Components: - CPO Dashboard - CPO Controller - Battery Management Module – CollectBatteryPercentageService - Charging Station Manager – CollectDataService – MonitorChargeService - CPMS DBMS

Table 4.8: Mapping of Goal 8

Goal 9	Handle the payment of a charge
	Requirements: R16, R30, R31
	CPMSComponents: - CPO Controller - Payment Module – PaymentService - CPMS DBMS

Table 4.9: Mapping of Goal 9

Goal 10	Provide the eMSP with the most convenient charging options by choosing to use locally stored energy or acquire it from DSOs
	Requirements: R16, R32, R36, R37
	CPMS Components: - CPO dashboard - CPO Controller - Battery Management Module – SwitchChargingSourceService - DSO Info Collector – CollectDataService - CPMS DBMS

Table 4.10: Mapping of Goal 10

Chapter 5

Implementation, Integration and Test Plan

This section contains a description of the implementation plan and the order in which the subcomponents will be implemented and integrated. It also contains a plan for system testing.

5.1 Implementation plan

As discussed previously and in the RASD, e-MALL is a complex system consisting of *two communicating subsystems*, which proved to be a challenge during development. For this reason, e-MALL CPMS and eMSP were designed as two almost separate software components whose interaction should be as platform independent as possible. This was also considered a requirement, as e-MALL eMSP should interact seamlessly with third-party CPMS, so designing them as independently as possible seemed as the most optimal approach.

A bottom-up approach was used to implement each system. In a bottom-up approach, the entire system can be built incrementally by developing each component independently and gradually integrating it into the overall system. Each component can be tested separately before being integrated into the system to prevent the occurrence of errors as early as possible.

e-MALL eMSP implementation

The first services to be implemented are the storage ones, so the setup of a DB and DBMS, and the development of the interfaces between the system and the databases. The eMSP sub-system is made of three main components:

1. **Charge manager**: Used to store user related data in the DB. Also interacts with external banking APIs to provide a payment platform.
   User wise, it provides all the tools to start, end and book a charge.

2. **Custom plan manager**: Used to create an *user-tailored* charging plan via data mining, as well as proactively suggest charging options.

3. **Station info collector**: Used to connect to Google Maps' APIs and CPMS in order to provide a complete representation of the stations along with the corresponding special offers.

The first modules to be implemented and tested were [1] and [3]. This allowed to test and debug an almost working system. Module [2] is defined as an *advanced feature* of e-MALL, which was considered an "optional" during the development phase, being completely independent from the other services; as well as not having any side-effect on the system.
Each service has been developed from scratch and initially provides all the methods for communicating with DBMS and performing all the necessary queries to store and retrieve data.
The next step was the development of the *User Controller*, which acts as a translation layer between the data processing modules and the *User Interface*. It is the module responsible for executing all service calls based on the user input.
Last but not least, we designed the *User Dashboard*, a graphical user interface designed to be as simple and engaging as possible.

e-MALL CPMS implementation

As mentioned earlier, the priority was to set up DB and DBMS to enable data storage and retrieval. After that, the main components were designed:

1. **Payment module**: Used to manage payments provided by users and is connected to banking APIs.

2. **Battery management module**: Used to manage local batteries. They can be recharged during a charging operation or provide additional charge when needed.

3. **Charging station manager**: Used to provide a complete overview of the charging station by displaying available, booked or free outlets, and local batteries status (if available). Also allows for manual interaction during a charge.

4. **DSO info collector**: Used for communication with DSOs and power procurement.

As stated in the RASD, manual intervention by an operator is supported because the CPO must be able to manage his own charging station. However, this aspect was only implemented in the final stages of development, as testing and debugging a complete and autonomous implementation was key to making the system manually usable later.
The first component to be implemented was [3], it enables the most important functions of the CPMS, therefore its correct implementation would eliminate the major bugs and vulnerabilities. The choice of the next component to implement was [2], in order to easily manage the whole station correctly. [1] and [4] were then implemented in parallel, as they both rely on external interfaces and don't interact with other modules for state-changing tasks.
The next component to be implemented was the *CPMS controller* and finally CPO's GUI (also called *CPO dashboard*).
A similar approach as stated in the eMSP section was used, first all services that perform data queries were designed, using, as always, a *bottom-up* approach.
Lastly, the interaction between the system and external APIs is tested.

5.2 Integration plan

The following diagrams represent the integration of the various components. The diagrams are ordered by the priority with which they are to be implemented, as indicated in the previous section.
The integration was split into *levels*. Each level follows the *bottom up* approach.
Despite different components, the subsystems are structured similarly, so the integration strategy is the same.
All components are implemented and tested by a *Mock object* that emulates the components not yet implemented.

5.2.1 eMSP integration plan

Level 0

The first component to be implemented is the DBMS Service, essential for all the other components.

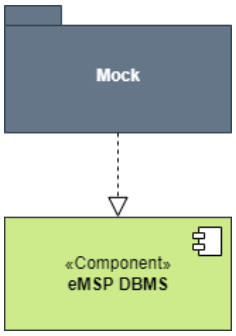

Figure 5.1: **Level 0**: integration of the eMSP DBMS services

Level 1

After the DBMS services are implemented, all components that interact with them can be integrated and tested. The StationInfoCollector module can also be developed in parallel, since it does not interact with DBMS, but is integrated later.

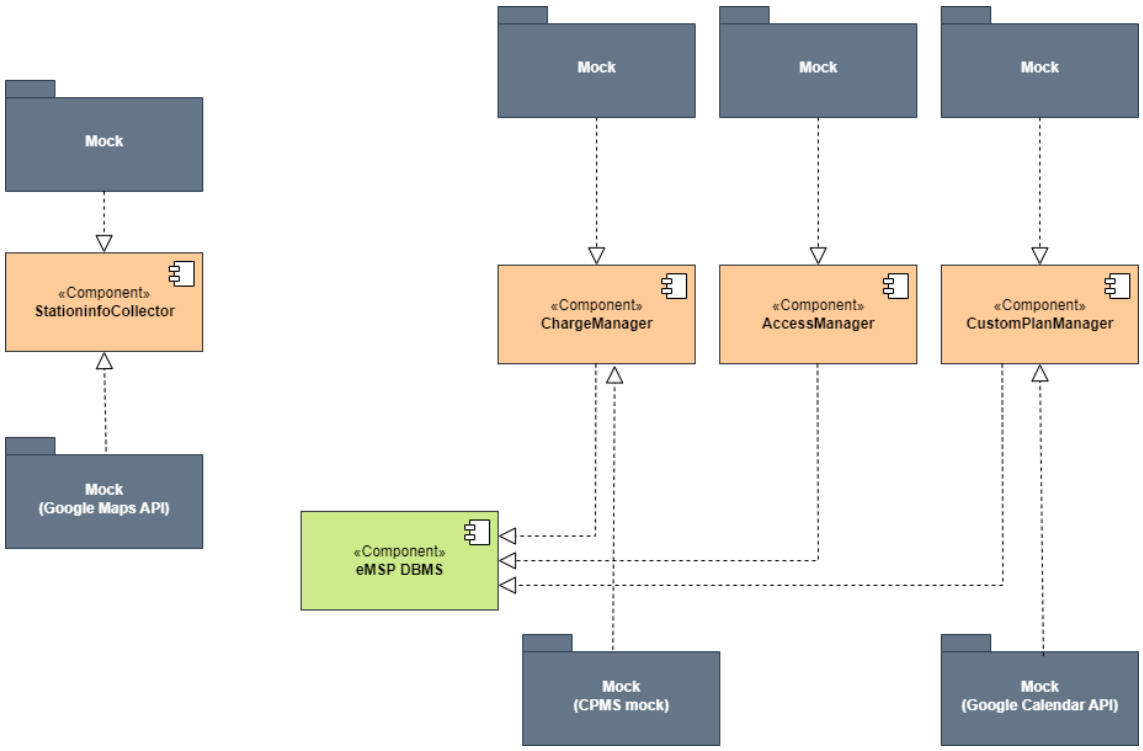

Figure 5.2: **Level 1**: integration of the main eMSP services

Level 2

Now that the main services have been tested and integrated, the UserController, which is responsible for calling the main services, can be tested and integrated.

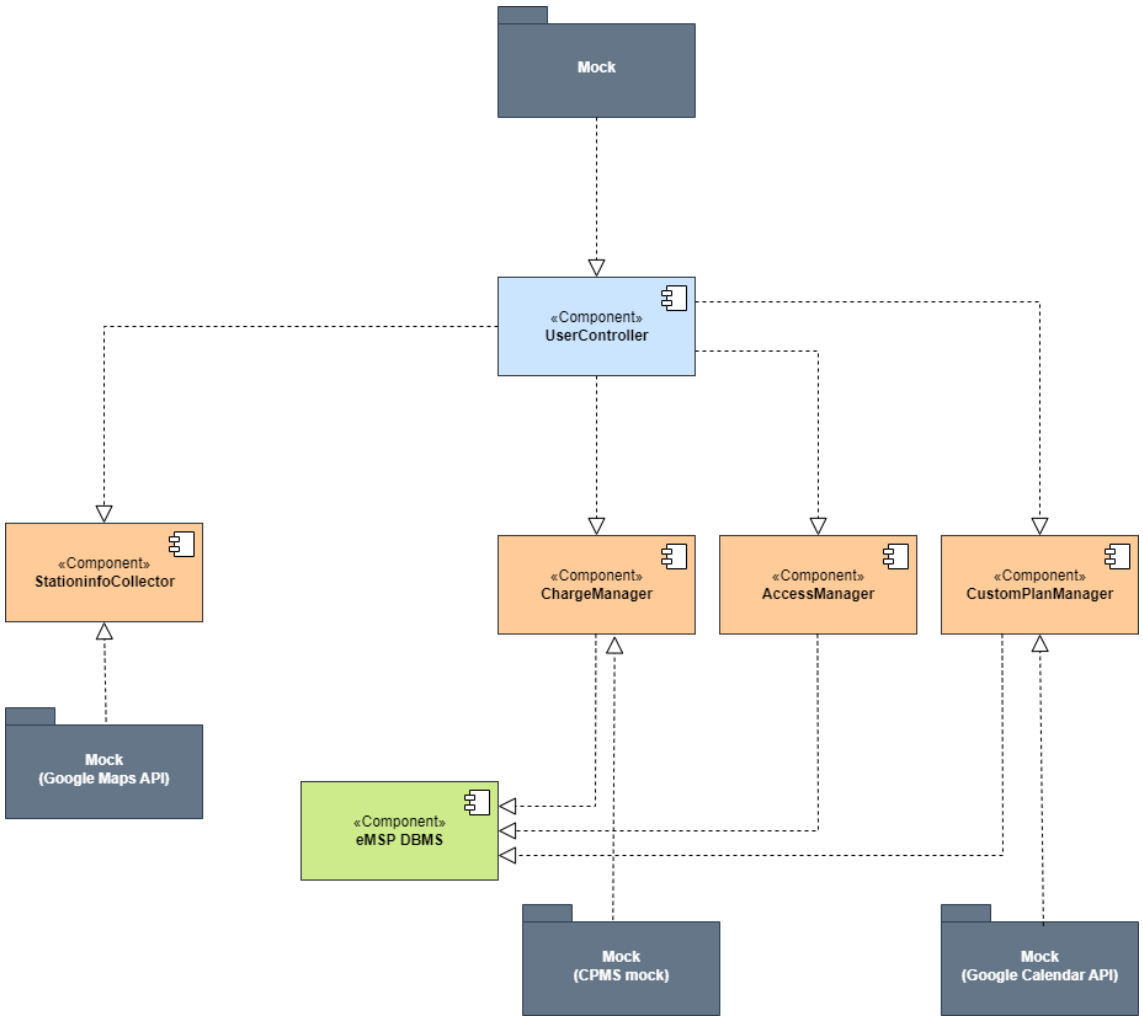

Figure 5.3: **Level 2**: integration of the UserController

Level 3

Now that the UserController has been tested and integrated, the UserDashboard, which is responsible for receiving the user inputs, can be tested and integrated.

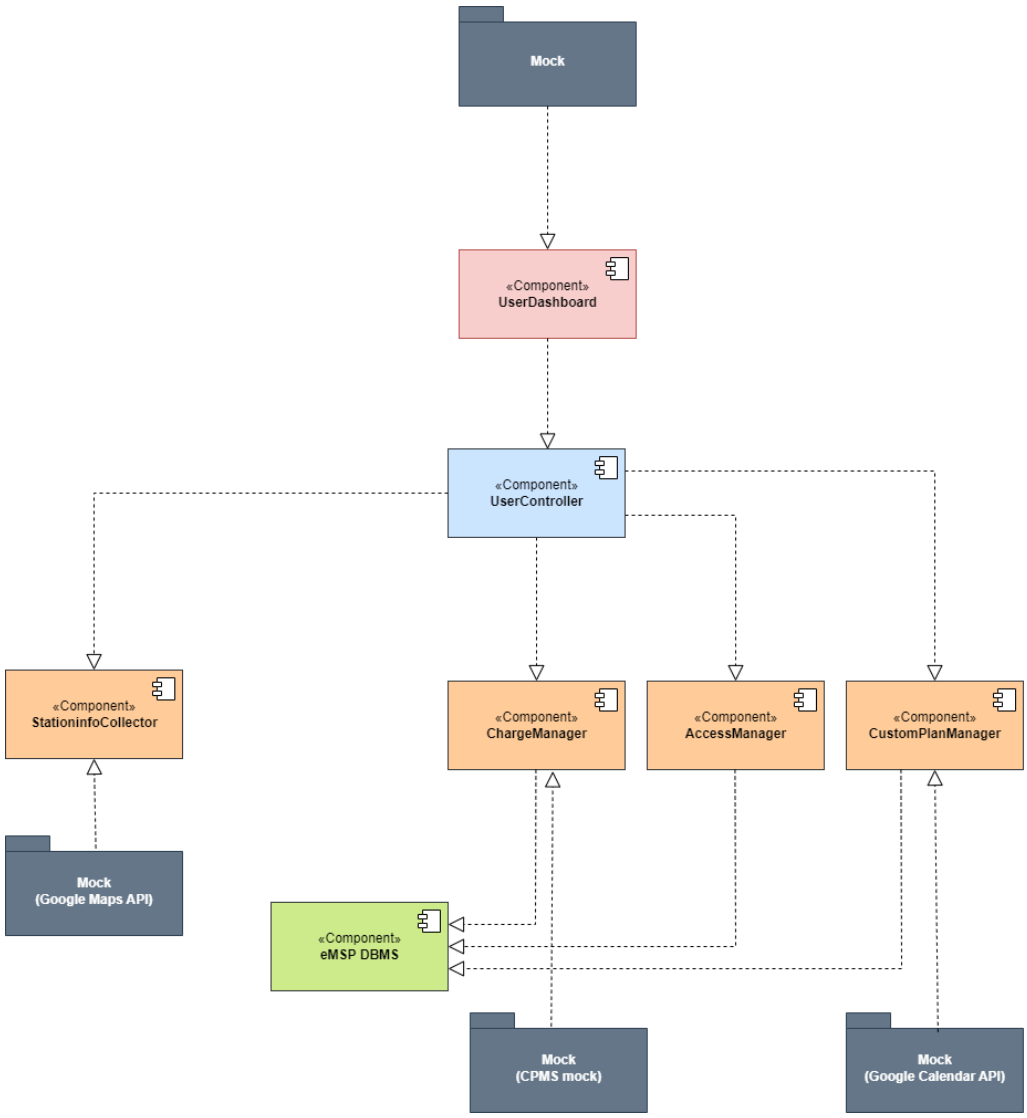

Figure 5.4: **Level 3**: integration of the UserDashboard

5.2.2 CPMS integration plan

Level 0

The first component to be implemented is the DBMS Service, essential for all the other components.

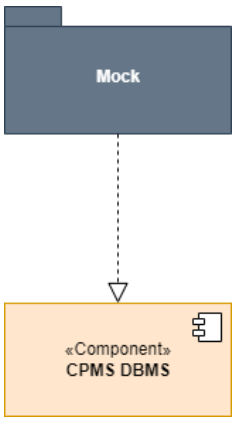

Figure 5.5: **Level 3**: integration of the CPMS DBMS Services

Level 1

After the DBMS services are implemented, all components that interact with them can be integrated and tested.

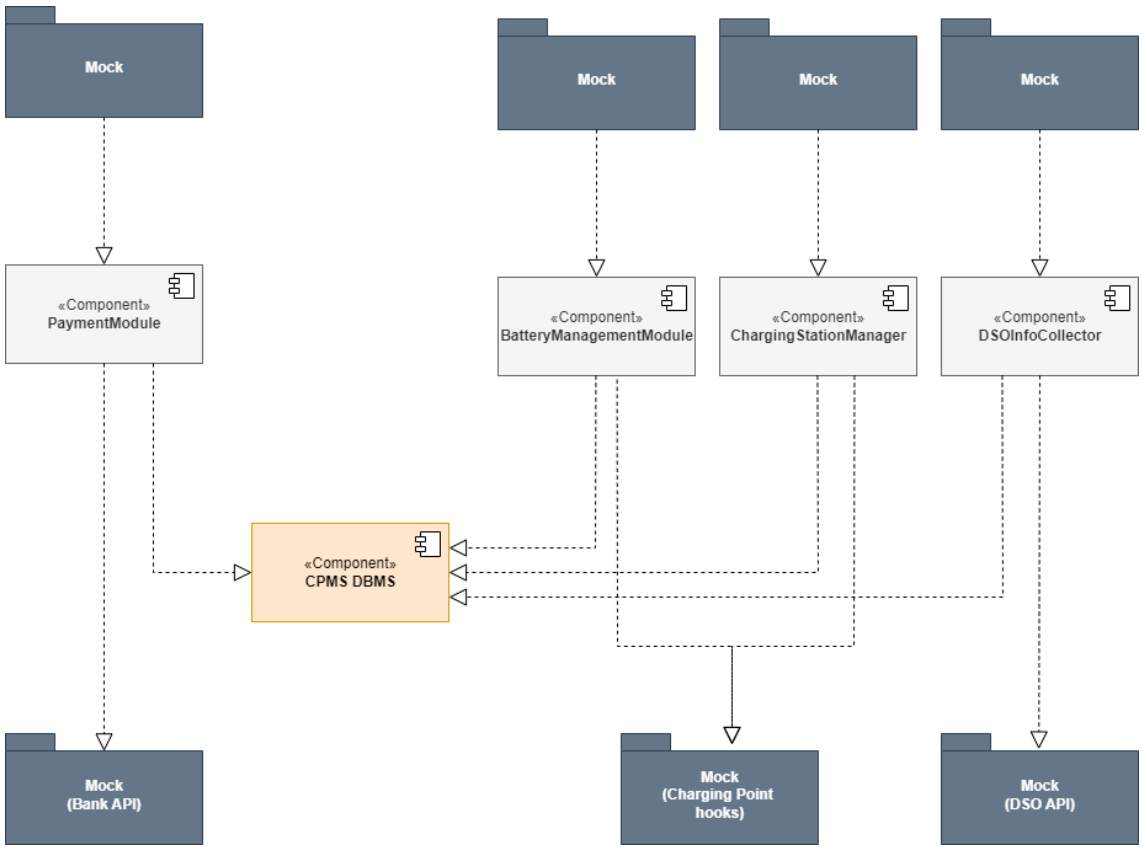

Figure 5.6: **Level 1**: integration of the main services

Level 2

Now that the main services have been tested and integrated, the CPOController, which is responsible for calling the main CPMS services, can be tested and integrated.

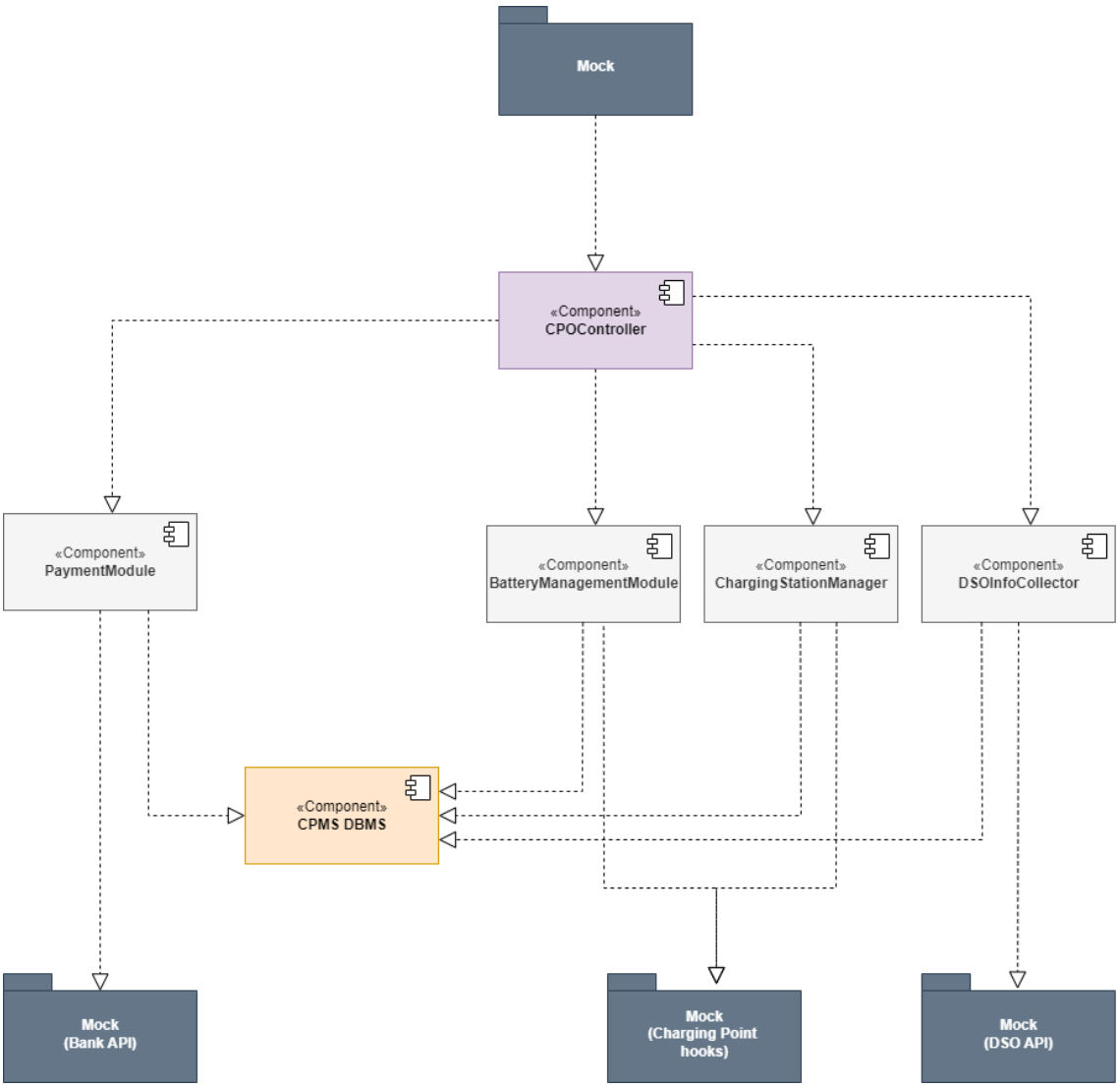

Figure 5.7: **Level 2**: integration of the CPOController

Level 3

Now that the CPOController has been tested and integrated, the CPODashboard, which is responsible for receiving the CPO inputs, can be tested and integrated.

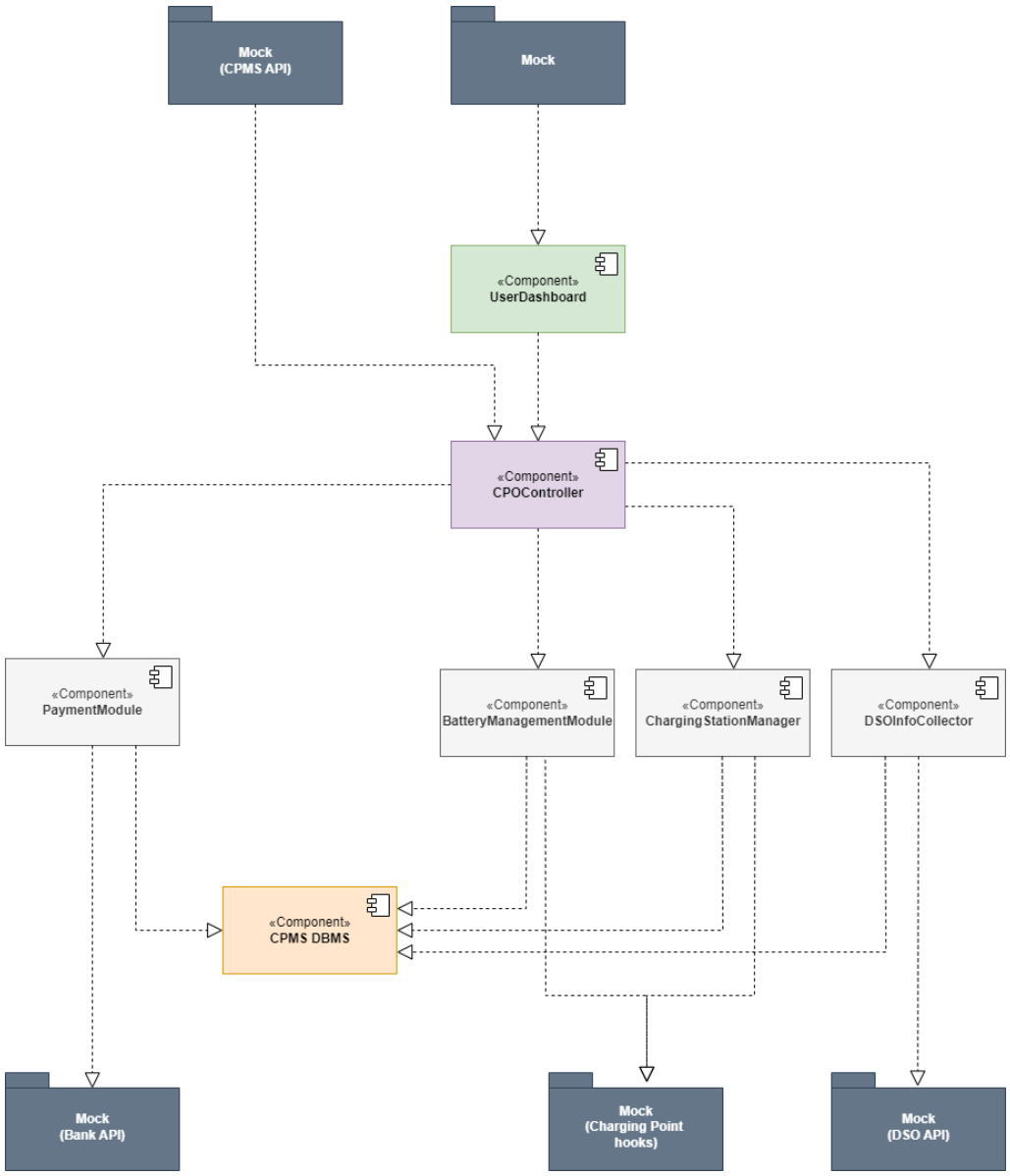

Figure 5.8: **Level 3**: integration of the CPODashboard

5.2.3 Subsystems integration plan

Now that all components of the two subsystems have been tested and integrated, they can be integrated again to test the communication between them.
The API calls are still present, as they will be removed once the communication has been thoroughly tested.

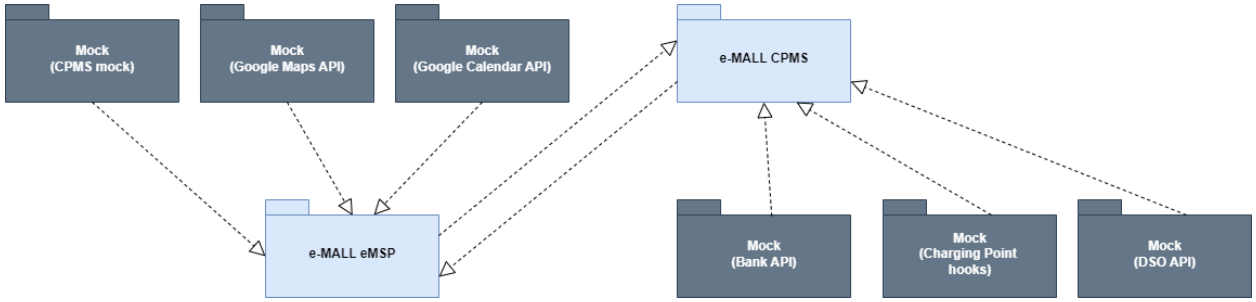

Figure 5.9: Integration of the two subsystems

5.3 System testing

Unit tests are the first step. Each implemented class must work as intended, including covering all possible corner cases.
The next step is integration testing with increasing levels of aggregation, ensuring that communication and data exchange work as intended.
The main goal of testing is to guarantee that all promised features work, so extensive testing is performed on all features.
All testing is done with a *bottom up* approach, meaning every small class is thoroughly tested before implementation.
A special effort is also made to guarantee *non-functional* requirements:

- **Performance testing**: testing speed, scalability, stability and reliability of the system
- **Load testing**: testing the behavior of the system under high loads
- **Stress testing**: testing the robustness of the system under varying loads and intensity

Chapter 6

Effort Spent

6.1 Effort required by members

Task	Time
Introduction	2h
High-Level Component view	2h
Deployment view	3h
Requirements traceability	2h
Implementation, Integration and Test Plan	3h
Rewriting and general fixes	4h
Final review	4h

Table 6.1: Marco Lorenzo Campo

Task	Time
Introduction	2h
High-Level Component view	2h
Runtime view	6h
Component Interfaces	5h
Rewriting and general fixes	3h
Final review	2h

Table 6.2: Alessia Abbondanza

Task	Time
Introduction	2h
Component view	3h
Component Interfaces	3h
User Interface Design	4h
Requirements traceability	2h
Implementation, Integration and Test Plan	3h
Rewriting and general fixes	1h
Final review	2h

Table 6.3: Alessandro De Luca

Chapter 7

RASD references

Here's a summary of goals and requirements provided in the RASD:

7.1 eMSP Goals

G1. Allow users to know the location, status, prices and special offers provided by charging stations

G2. Allow users to manage bookings of a charge in a specific location and time frame

G3. Allow users to start, monitor and end the charging process

G4. Allow users to pay for the obtained service

G5. Suggest users to charge their vehicle based on the charge status, the schedule of the users, special offers provided by the CPOs and availability of the charging slots at the identified charging stations

7.2 CPMS Goals

G6. Provide a complete "external" overview of the charging station (location, availability, charging options. . .)

G7. Manage and correctly execute a charge for a specific time frame

G8. Know the "internal" status of a charging station

G9. Handle the payment of a charge

G10. Provide the eMSP with the most convenient charging options by choosing to use locally stored energy or acquire it from DSOs

7.3 Requirements

R1. The e-MALL application should allow users to register by providing their email address

R2. The e-MALL application should allow users to login using the credentials input at registration time

R3. The e-MALL application should retrieve locations of charging stations from their CPMS

R4. The e-MALL application should show availability of charging stations from their CPMS

R5. The e-MALL application should retrieve prices of charging stations from their CPMS

R6. The e-MALL application should retrieve special offers of charging stations from their CPMS

R7. The e-MALL application should acquire locations of charging stations from their CPMS

R8. The e-MALL application should allow the user to book a charging station for a specific location and time frame

R9. The e-MALL application should allow the user to delete a previously booked charging station

R10. The e-MALL application should register a start signal when the "start charge" button is pressed

R11. The e-MALL application should forward the start signal of the charging to the CPMS

R12. The e-MALL application should register an end signal when the "end charge" button is pressed

R13. The e-MALL application should forward the end signal of the charging to the CPMS

R14. The e-MALL application should allow users to insert their banking credentials

R15. The e-MALL application should be able to communicate with the user's bank

R16. The e-MALL application should connect to external CPMSs through API

R17. The e-MALL application should know the charge status of the user's vehicle

R18. The e-MALL application should know the daily schedule of the user

R19. The e-MALL application should be aware of special offers provided by the CPO

R20. The e-MALL application should be aware of the availability of charging slots at the identified station

R21. The CPMS should collect real time information about available charging sockets

R22. The CPMS should collect real time information about the charge speed provided by each charging socket (classified as slow, fast or rapid)

R23. The CPMS should collect real time information about the cost of a charge

R24. The CPMS should collect real time information about the estimated time to completion of a charge currently happening

R25. The CPMS should set an "available" socket as "unavailable" for 15 minutes after it receives a booking

R26. The CPMS should set a booked socket as "available" if no charge has been started for 15 minutes

R27. The CPMS should set a booked socket as "available" if the eMSP deletes the relative booking

R28. The CPMS should set a socket as "unavailable" as long as a charge is in process

R29. The CPMS should stop dispensing power when the plugged battery is fully charged

R30. The CPMS should provide real time expenses of a charge

R31. The CPMS should receive payments from the eMSP

R32. The CPMS should communicate with DSOs' API to retrieve the cost of energy

R33. The CPMS should choose the most convenient DSO based on the eMSPs' charge requests

R34. The CPMS should know the CPOs' local energy availability

R35. The CPMS should know if batteries are available in the charging station

R36. The CPMS should compute where to dispense energy from (local batteries, DSOs or a mixture of the two), to provide the most convenient options

R37. The CPMS should allow operators to manually handle the energy acquisition and dispense procedure

R38. The CPMS should communicate the current special offers to the e-MALL application

R39. The e-MALL application should know the charge status of the user's vehicle

R40. The e-MALL application should know the daily schedule of the user

R41. The e-MALL application should be aware of special offers provided by the CPO

R42. The e-MALL application should be aware of the availability of charging slots at the identified station

Chapter 8

References

- J2EE Web Servers Load Balancing.
- William Stallings and Lawrie Brown. Computer Security: Principles and Practice. Prentice Hall Press, USA, 3rd edition, 2014.
- Ingegneria del software. Binato, A., Fuggetta, A. and Sfardini, L., 2006. Milano: Pearson education Italia.
- Link: Relational vs. Non-Relational Database: Pros & Cons | The Aloa Blog, accessed: 2022-01-09, Aloa Blog, Pawlan David.
- What is HTTPS? | Cloudflare.
- MDN Web Docs Glossary: Definitions of Web-related terms - MVC
- OCPI 2.2: Open Charge Point Interface -document version 2.2-d2, 12-06-2020
- Setting The Standard: OCPP 1.6 vs 2.0.1 - Noah Newfield, October 22, 2020