POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

eMall – e-Mobility for all

Design Document - DD

Author(s): **Fabio Lusha - 10882532**
Bianca C. Savoiu Marinas - 10684465

Academic Year: 2022-2023

Contents

Contents **i**

1 Introduction **1**
1.1 Purpose . . . 1
1.2 Scope . . . 2
1.3 Definitions, Acronyms, Abbreviations . . . 3
1.3.1 Abbreviations . . . 3
1.3.2 Definitions . . . 4
1.4 Reference Documents . . . 5
1.5 Document Structure . . . 5

2 Architectural design **7**
2.1 Overview . . . 7
2.2 Component view . . . 8
2.3 Deployment view . . . 16
2.4 Runtime view . . . 19
2.4.1 EVD runtime view . . . 20
2.4.2 CPO runtime view . . . 27
2.5 Component interfaces . . . 31
2.5.1 Common interfaces . . . 31
2.5.2 eMSP interfaces . . . 32
2.5.3 CPMS interfaces . . . 35
2.6 Selected architectural styles and patterns . . . 40
2.7 Other design decisions . . . 42

3 User Interface Design **45**
3.1 EVD user interface . . . 45
3.1.1 Log in . . . 45

3.1.2 Register . . . 46
3.1.3 Charge now . . . 47
3.1.4 Visualize stations . . . 48
3.1.5 Booking charging point . . . 49
3.1.6 Terminate charging . . . 50
3.2 CPO user interface . . . 51
3.2.1 Visualize stations and add a general promotion . . . 51
3.2.2 Update price . . . 52
3.2.3 Store energy in battery . . . 53
3.2.4 Update DSO . . . 54
3.2.5 Analyse stations . . . 55

4 Requirements traceability **57**
4.1 Requirements . . . 57
4.2 EVD requirements traceability . . . 61
4.3 CPO requirements traceability . . . 76

5 Implementation, integration and test plan **85**
5.1 Implementation, integration and test plan for the CPMS part of the eMall 85
5.2 Implementation, integration and test plan for the eMSP part of the eMall . 88

6 Effort spent **91**

7 References **93**

List of Figures **95**

List of Tables **97**

1 | Introduction

1.1. Purpose

Widespread electrification of transport is the most efficient way to reach Europe's climate objectives for the sector and electric charging is the main asset to overcome the obstacles of the take-up of electric vehicles (EVs). EVs can reduce CO2 by an estimated annual 600,000 tons by 2030, going towards a carbon neutral Europe, and the importance of this aim raises the problem of having efficient systems that manage the charging services. The eMall is thought as an all-encompassing application that oversees the entire process from the user interaction to the effective recharge of the EV's battery.

The main goal we want to achieve with the eMall software is to help the EVDs (electric vehicle drivers) to have better access to recharge and to be able to book a charging point in order to avoid interference with his daily plans. Another important purpose of the system is to safeguard not only the users but also the providers of the service and this is made through privacy agreements and the actual interaction, that guarantees to supervise both interested parts, in order to get the best possible service and pay for it accordingly, having also a technical and economic exploitation of the charging infrastructures.

In this context there is an increase in the requested electric energy, but large amounts of power in short periods would require investments in the reinforcement of the distribution networks, which have not been designed to accommodate such load. It becomes necessary to introduce new systems and solutions to optimize the operation of the distribution networks. In this context we can identify the DSOs as the suppliers of electricity through the distribution networks. The DSOs interact with the eMall, and in particular with the CPMS (Charging Point Management System) module of the system to be. The CPMS, then, gives the information about the DSO's supply to the CPOs, which are important actors, that use the system in order to manage the charging service. A CPO is represented by an employee or a software, part of the business that owns some charging stations and wants to manage them through the eMall, deciding from where to acquire energy, and how to establish the prices, the special offers and other details about the stations.

The eMall is thought as a software that manages both the interaction with the businesses that offer the charging service and the interaction with the EVDs which want to use these services in order to charge their EVs. Therefore, the eMall provides a mobile application (eMma), which through its interface allows to the EVD to obtain the service, and provides, also, a web application that the CPOs use to manage the charging stations. The EVD interacts, as well, with the charging point interface (eMci), that communicates with the CPMS part of the eMall, in order to start the charging session from the station, plugging then the car to the compatible connector to effectively charge the EV.

By the official definition of the IEEE Std 1016™-2009 standard, the DD is a representation of a software design that is to be used for recording design information, addressing various design concerns, and communicating that information to the design's stakeholders. So, in this document we focus on the design of the system to be, describing the components and the interaction among them and with external systems, through interfaces, in order to achieve the goals and satisfy the requirements explained in the RASD document.

1.2. Scope

The eMall has two main stakeholders: the CPOs and the EVDs. There are different goals to satisfy respectively for the CPOs and the EVDs and different associated requirements to develop. For a detailed description of the domain in which the e-Mall will operate, and to see all the goals and the requirements, the reader should refer to the RASD. While, in this document, we briefly present the main design concerns of the stakeholders, and the main architectural styles.

The stakeholders have goals that, in order to be satisfied, need the storage of a large volume of data, therefore scalable technologies are required to manage the amount of information used by the system. Also, the stakeholders, especially the EVDs, are not expert users, thus in order to have an user-friendly system is necessary to create a graphical interface with which the users will interact. The system will have a GUI for the eMma and a GUI for the web application used by the CPOs, compatible with any browser, in order to facilitate the access to the system. We want also for the user interfaces to be compliant with different devices, regardless of the screen size.

This document will present as main adopted architectural style the 3-layered architecture, that we combine with other architectural choices. For the business logic layer, in fact, we

will present a micro-services architecture, while for the presentation layer we will adopt a client-side rendering architecture, in which the client (the web browser and the mobile app) is responsible for rendering the GUI of the application. These architectural choices will be further explained in the following chapter of the DD.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Abbreviations

- **eMall**: e-Mobility for all
- **eMma**: e-Mall mobile application
- **eMci**: e-Mall charger interface
- **CPMS**: Charging Point Management System
- **CPO**: Charge Point Operator
- **eMSP**: Electric Mobility Service Providers
- **DSO**: Distribution System Operator
- **OCPI**: Open Charge Point Interface
- **DBMS**: Database Management System
- **DBAL**: Database Abstraction Layer
- **OS**: Operating System
- **EV**: Electric Vehicle
- **EVD**: Electric Vehicle Driver
- **RASD**: Requirements Analysis and Specification Document
- **DD**: Design Document
- **UI**: User Interface
- **GUI**: Graphical User Interface
- **DB**: Database
- **DBMS**: Database Management System
- **DBAL**: Database Abstraction Layer

1.3.2. Definitions

- **DSO**: typically the entity responsible for the operation and management of distribution networks – High, Medium and Low Voltage networks
- **CPO**: entity that technically manages all the EV infrastructure assets, depending of existing country regulation – this role can be assured by the DSO or other entity
- **eMSP**: is the entity that can explore the economic side of the EV charging infrastructure, namely by selling energy for charging purposes
- **CPMS**: is a software system that manages the charge point infrastructure – can manage the technical and economic aspects of the charging infrastructures
- **EVD**: person or entity who owns an EV car and can use the public or private facilities for charging purposes
- **Vehicle inlet**: the port on the electric vehicle that receives charging power
- **Rectifier**: an electrical device that converts alternating current (AC) to direct current (DC)
- **eMma**: the eMSP subsystem responsible for the EVD interaction from the mobile app
- **eMci**: the eMSP subsystem responsible for the EVD interaction at the charging point
- **Additional costs**: overtime penalty, deposit for unregistered users
- **Status of the charger**: can be free, occupied, booked and in maintenance
- **DD**: is an SDD (Software Design Description), which is a representation of a software design to be used for communicating design information to the stakeholders and also to guide the development of the system. The standard refers to this document as SDD, but in the following presentation we will call it DD
- **RASD**: is the document that analyzes and presents all the requirements of the system to be, explaining the domain in which the software will operate under some assumptions, and its interactions with the users

1.4. Reference Documents

- IEEE Std 1016™-2009 International Standard for Information Technology - Systems Design - Software Design Descriptions: describes how to structure a DD, giving a representation of a software design to be used for communicating design information to its stakeholders. The requirements for the design languages (notations and other representational schemas) to be used for conformant design documents are specified
- ISO/IEC/IEEE 42010 International Standard - System and Software engineering - Architecture description: this International Standard addresses the creation, analysis and sustainment of architectures of systems through the use of architecture descriptions. This International Standard provides a core ontology for the description of architectures. The provisions of this International Standard serve to enforce desired properties of architecture descriptions and also specifies architecture frameworks and architecture description languages (ADLs), in order to usefully support the development and use of architecture descriptions
- RDD: assignment document
- RASD: the document written for the e-Mall, which specifies the goals and the requirements the system to be has to achieve and analyzes the domain of the system
- Electric Vehicle CPMS and Secondary Substation Management by F. Campos, Efacec, Portugal; L. Marques, Efacec, Portugal and K. Kotsalos, Efacec, Portugal (15 October 2018): used to define the interactions between the different parts of the system and the actors; models the EV public infrastructures, the eMSP, the DSO and the CPMS together with the APIs and protocols that allow their communication

1.5. Document Structure

This document mainly follows the guidelines of the IEEE Std 1016™-2009 International Standard for Information Technology - Systems Design - Software Design Descriptions, especially the sections 4 and 5 of the document, ordering the parts in a way that fits best the topics of this document. The document is composed by the following sections:

- the first section, to which this part belongs, provides an introduction of the eMall, similar to the RASD document, introducing also the main architectural choices

further explained in the following parts of the document

- the second section provides a specific description of the architecture, specifying the design decisions regarding the software to be; the section contains a formal description using UML diagrams to show the components of the eMall, the interfaces, and the interactions of the components among them and with external systems; we start with a high-level view of the system and then we decompose it in smaller parts showing their connections and dependencies; the section contains a component interface diagram, a deployment view and sequence diagrams describing the main interactions explained with the use cases of the RASD interactions between components.
- the third section provides mockups of the user interface, explaining the design of the GUI developed for the mobile app and the design of the GUI developed for the web application, describing the interaction of the stakeholders with the interfaces
- the fourth section provides a mapping between the components described in the previous sections and the requirements specified in the RASD, and a component is mapped on the requirement when it contributes to its fulfillment; in order for the document to be self contained we also report the requirements identified in the RASD, and we explain the mapping, thus the reader can understand how the provided architecture reaches the requirements
- the fifth section provides a plan to follow during the implementation of the eMall, specifying from which components to start, which components can be developed in parallel and provides, also, a plan for the integration of the components and the testing, from unit testing, to integration testing and finally the testing of the entire system
- the sixth section reports the effort spent by the components of the group to complete this document
- the last section contains the references used, beyond the ones already specified in this chapter

2 | Architectural design

2.1. Overview

In this section we will provide a high-level view and description of the components that our system is made of. The architecture chosen for our system is a three-tier one. The major advantages of this architectural style is the decoupling of the application logic from the presentation logic and the data persistence concerns. Further details about the characteristics of this architectural style will be given in section 2.6, for now we will proceed with the general overview of the system. In the picture below is illustrated a high-level view of the system with an informal notation, where each rectangular box represents a high-level computational unit of the system, meanwhile the double-edged arrow represents the interaction between two components.

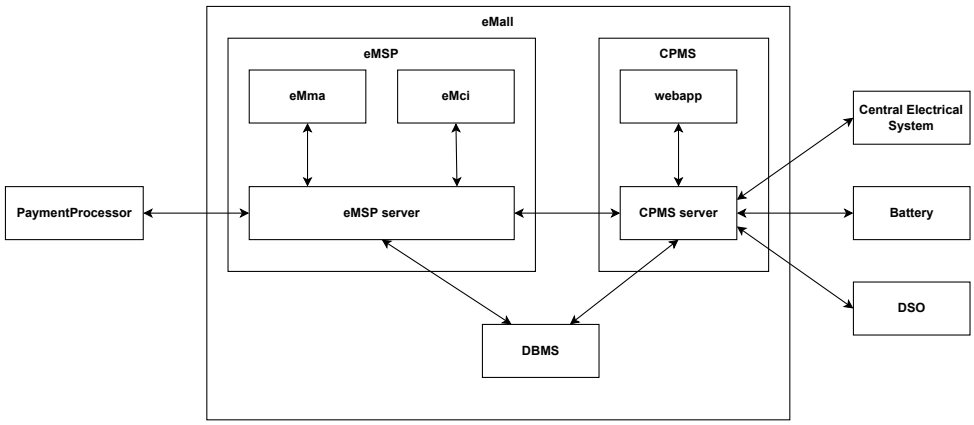

Figure 2.1: High level description of the components and their interactions

The eMall system, illustrated in the figure, is the objective of this design document. It is clear that the system is divided in two main sub-systems: eMSP and CPMS. This choice is driven by an interoperability requirement of the eMSP and the CPMS with different CPMS and eMSP systems respectively, offered by other companies. Nonetheless, this low-coupling of the CPMS with the eMSP doesn't preclude us from reusing components

that have the same functionality in both sub-systems. It is also important to point out that our system, specifically the CPMS sub-system, must be able to interface with the system responsible for managing the technical aspect of the charging point, the system that manages the battery, if present, and the DSO's software system.

As we stated previously, the system has a three-tier architecture. In particular the three tiers are:

Client tier It's the tier closest to the user and its duty is to manage the user interaction. This means that it must handle the visualization of the content to the user and interpret and translate the user interaction in requests to be forwarded to the application tier. We'd like to remark that this tier doesn't contain any application (or business) logic. We will use a client-side rendering software architecture to design this layer.

Application tier This is the part where the core and the business logic of the system is implemented, consequently, this second layer realizes the functionalities required to the system, like the booking service or the charging station management service for the CPO. All this functionalities shall be discussed in more detail in the upcoming sections. As we will see in the following section, a micro-services approach has been used to build such layer.

Data tier The third, and bottom tier, of our system is the data tier, where the persistence concerns of our system are met. The eMall system, both CPMS and eMSP sub-systems, has to handle a large amount of data, which must be carefully stored in order to have a properly working system. The data management is an aspect of software systems that has been thoroughly studied and developed, so the obvious choice is to use an already implemented and tested Database Management System (DBMS).

2.2. Component view

In this section we will discuss and elaborate on the components that compose our system in order to implement the functionalities listed in the RASD document. We will start from a higher level of abstraction to provide a grasp of how the system works, thus even those who are not familiar with the technical aspects of a software system can understand at high level how the system is structured. Afterwards, we will proceed to further analyze the system at finer levels.

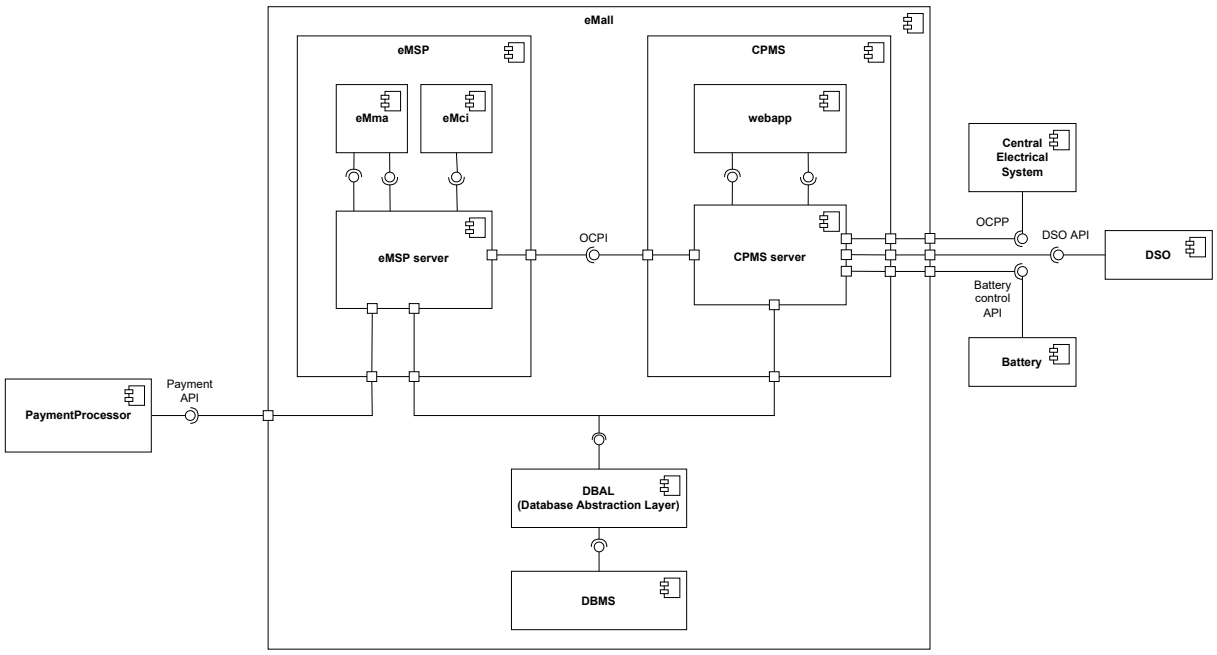

Figure 2.2: Component view of the system

This component diagram provides a higher level view of how the system is composed. It distinguishes the parts that the development team must build (the eMall component), and lists the external system with their respective interfaces with which the eMall must interact to provide its functionalities. Furthermore, it's apparent that the system is composed by two main sub-systems, the eMSP and CPMS, that must interact with each other through the OCPI interface, and each one of these sub-systems is structured with a client-server architecture. The OCPI interface is an open and free interface that standardizes the interface of a CPMS, in other words it standardizes the services a CPMS offers and how to make requests for these services to the CPMS. As a consequence of using the OCPI protocol we allow our eMSP to be able to interact with different CPMS*s*, and our CPMS to interact with different eMSP*s*, as per requirement. Finally, there are the DBMS and DBAL components, that compose the third and bottom tier of our architecture. The presence of the DBAL component tells us that a level of indirection from the DBMS has been added, allowing the system to be independent from any particular DBMS. Regarding the external components with which the eMall shall interact we denote with *Central Electrical System* the software system responsible for managing the technical aspect of the charging station, like unlocking the charging point, controlling the flow of electricity at a charging point and connecting (likewise disconnecting) electrical components to the electrical. With Battery we denote the software responsible for managing the battery, like monitoring the charging status or the level. Next, there is the DSO, with denotes the software system of DSO

that will be used by the CPSM, to gather information about energy price and conclude electricity purchase contracts. Finally, there is the PaymentProcessor to which the eMSP will forward the payment request from the user to pay for the charging sessions.

In the next paragraphs we shall discuss in more detail the composite structure of the main sub-systems of the eMall: the eMSP and CPMS.

eMSP

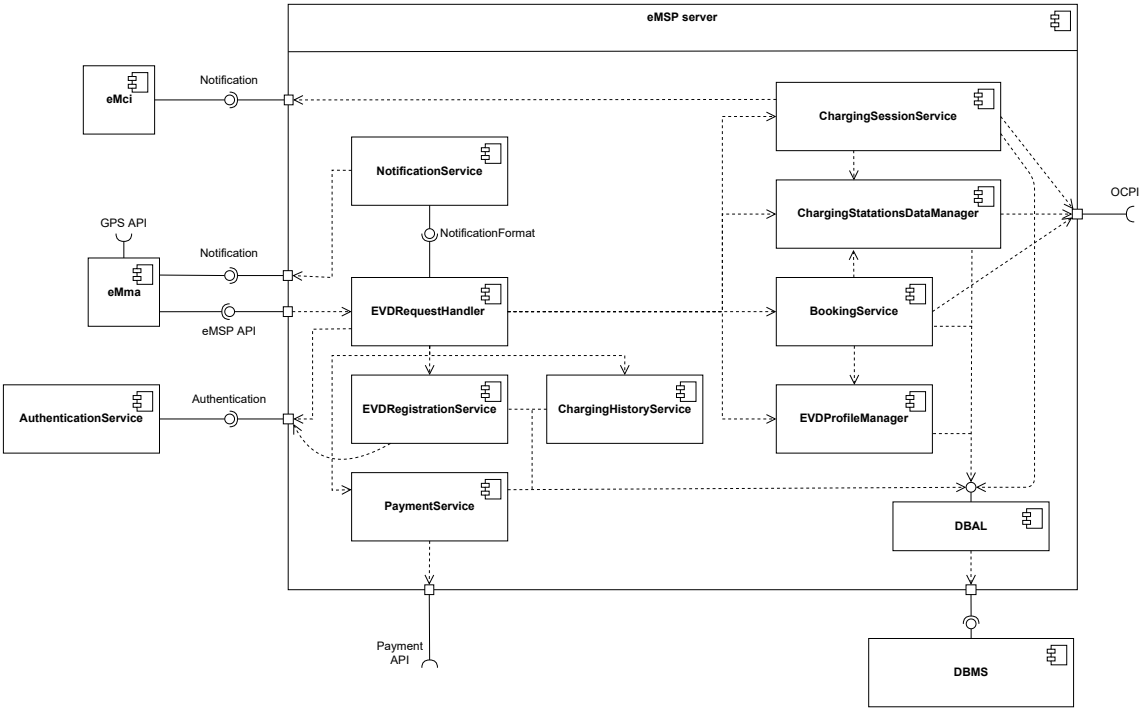

Figure 2.3: eMSP composite structure

The eMSP is the system that will interact with the EVD and forward his charging request to the CPMS. As we can see the eMSP isn't only a middleman between the EVD and the CPMS but offers other services like profile management, payment services and charging history services. Below we provide a brief description of each component of the eMSP:

- **eMma**: eMall mobile app. It is the client componet of the eMSP, and its task is to act as a view and partially as a controller for the EVD interactions. Specifically, it will provide a GUI to the user from which he can make his requests to the eMSP. Furthermore, it will encode the user input and interaction with the view to the corresponding functions of the eMSP server interface. Finally it has to interact with the GPS module of the OS in which it will run to provide location information to the eMSP
- **eMci**: eMall charging interface. This component's role is to provide the graphical interface the EVD will be using at the charging point when he wants to start a charging session and is located on an embedded device on the charging point. Its job is, then, to identify the charging point in which it resides so the EVD can inform the eMSP of which charging point he wants to use

- **DBMS**: Database Management Service. It is the usual system with the task of managing the insertion, storing, deletion and update of the data generated by the system. This component is exceptionally complex and given the widely availability of this kind of systems on the market, using an already build and thoroughly tested one will be mandatory
- **DBAL**: Database Abstraction Layer. It functions as an additional layer of indirection form the DBMS. In this manner the eMSP will be independent of the DBMS used and if this component changes in the future the eMSP won't need to be modified to integrate the interface of the new DBMS
- **AuthenticationService**: Component that has to mange user's authentication to the system and authorization to use the services
- **PaymentService**: This component, through the usage of external Payment APIs offered by external providers, allows the eMSP to handle the payment requests from the users
- **ChargingHistoryService**: The component which has to process the data saved in the DB to present the user with a functional information regarding his charging history
- **EVDRegistrationService**: Component which has to handle the user registration process. Obviously it shall use the DBAL to store the user information once the process has finished. Additionally it also has to use the component AuthenticationService in order to login the user once the registration process has successfully ended
- **EVDProfileManager**: Component responsible for managing and processing the data concerning the EVD personal information and the EVD's data about his EVs
- **ChargingStationsDataManager**: This is one of the main components of the system. It has to interact with the CPMS to gather data useful for the EVD and more specifically that will be used and processed by the other components. Moreover it acts as an interface for other components that need information about the stations, so they do not need to depend on the data structure used to store the stations
- **ChargingSessionService**: Component responsible for unlocking the charging point, starting the charging session and terminating it. Obviously to offer this services it must interact with the CPMS through the OCPI interface. Additionally it uses the DBAL interface in order to save the data about the charging session, which will be used by the ChargingHistoryService component, and can perform checks when the

user is trying to unlock the wrong charging point by cross-checking the data stored in the DB about the charging points and the code it receives by the eMci component

- **BookingService**: Component that provides the booking functionality. It must use the EVDProfileManger and the ChargingStationsDataManager components to get the specific information about the user and the station respectively in order to make the appropriate request through the OCPI interface. Moreover it stores this information in the DB so it can next be used if necessary to cancel a booking
- **NotificationService**: Component that provides the eMSP server with the capability of sending notification to the EVD through the eMma component, consequently this component must use the communication interface offered by eMma
- **EVDRequestHandler**: All the users' requests pass through this component. This centralized approach is need so the system can check if the requests have the right permissions (i.e. request for services allowed only for registered user). This explains the dependency with the AuthenticationService component. Obviously it also need to use all the other components to grant the user the services offered by the system

CPMS

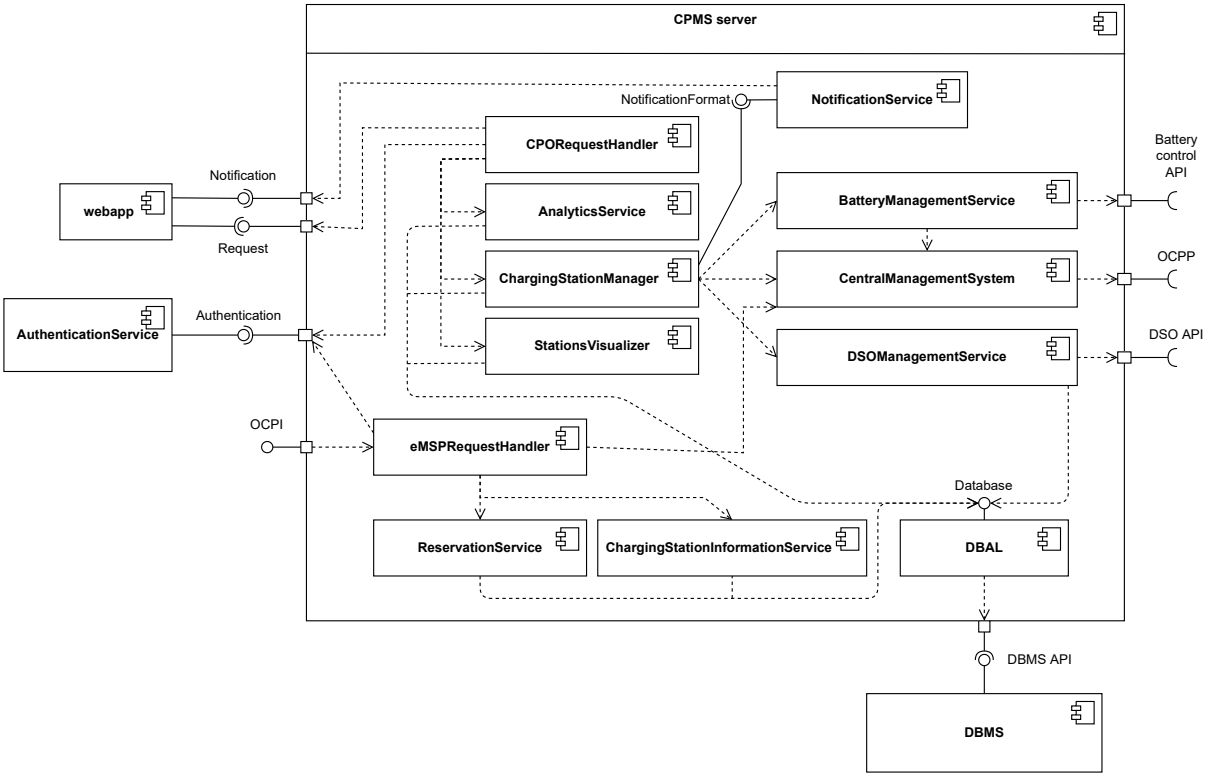

Figure 2.4: CPMS composite structure

The CPMS is the subsystem responsible for managing the charging stations, offering charging services to EVDs and securing the electricity supply from the DSOs. To offer all this functionalities this sub-system of the eMall must collaborate with other systems external to our own, which can be viewd in the general overview. Specifically, the CPMS needs to interfaces to the system responsible of managing the battery by the `Battery control API` interface, with the DSO IT system through it's API and with the the system managing the charging points devices through the protocol OCPP, which is an open standard. Like we have done for the eMSP, we will proceed describing each component of the CPMS sub-system (components AuthenticationService, DBAL, DMBS and NotificationService have the same functionality as in the eMSP system and shall not be repeated here):

- **CentralManagementSystem**: Component responsible for interacting with the central electrical system of the charging station. This component will query the central electrical system through the OCPP protocol, asking for the status of the charging points, and invoking request like starting (likewise terminating) the flow

of electricity of a particular charging point to start a charging session

- **DSOManagementService**: Component responsible for collecting the data from the various DSO and concluding sales contracts with them if the API allows it
- **BatteryManagementService**: Component responsible for managing the battery of the charging station if one is present. This component must interface with the software system managing the technical aspects of the battery through its control API (here called Battery control API) and must use the methods provided by the CentralManagementService component to allow the central system to switch to the battery for charging or charging the battery when requested
- **StationsVisualizer**: Component that allows the CPO to visualize and select one of the charging stations managed by him
- **AnalyticsService**: Component that provides the CPO with the tools to do some statistical analysis on the data collected by each charging station
- **ChargingStationManager**: Main component responsible for the interaction with the CPO. It implements all the functionalities related to the business aspect of the charging station (setting electricity price, promotions, etc.) and uses the components described before for managing the technical aspects of the charging station. This component also uses the NotificaionService component to notify the CPO about any necessary information
- **CPORequestHandler**: It is the component where all the CPO requests must first pass through in order to perform security checks and allow only authorized users to access the services of the CPMS. To offer this functionality the component makes use of the AuthenticationService component to login the user and to check if the requests coming from the webapp have the necessary authorization. If a request passes the controls it's forwarded to the respective component
- **ChargingStationInformationService**: Component in charge of processing the data of the charging station, stored in the DB, in a suitable format for the eMSP and compatible with the OCPI interface
- **ReservationService**: Component responsible of managing the reservation of the charging points

2.3. Deployment view

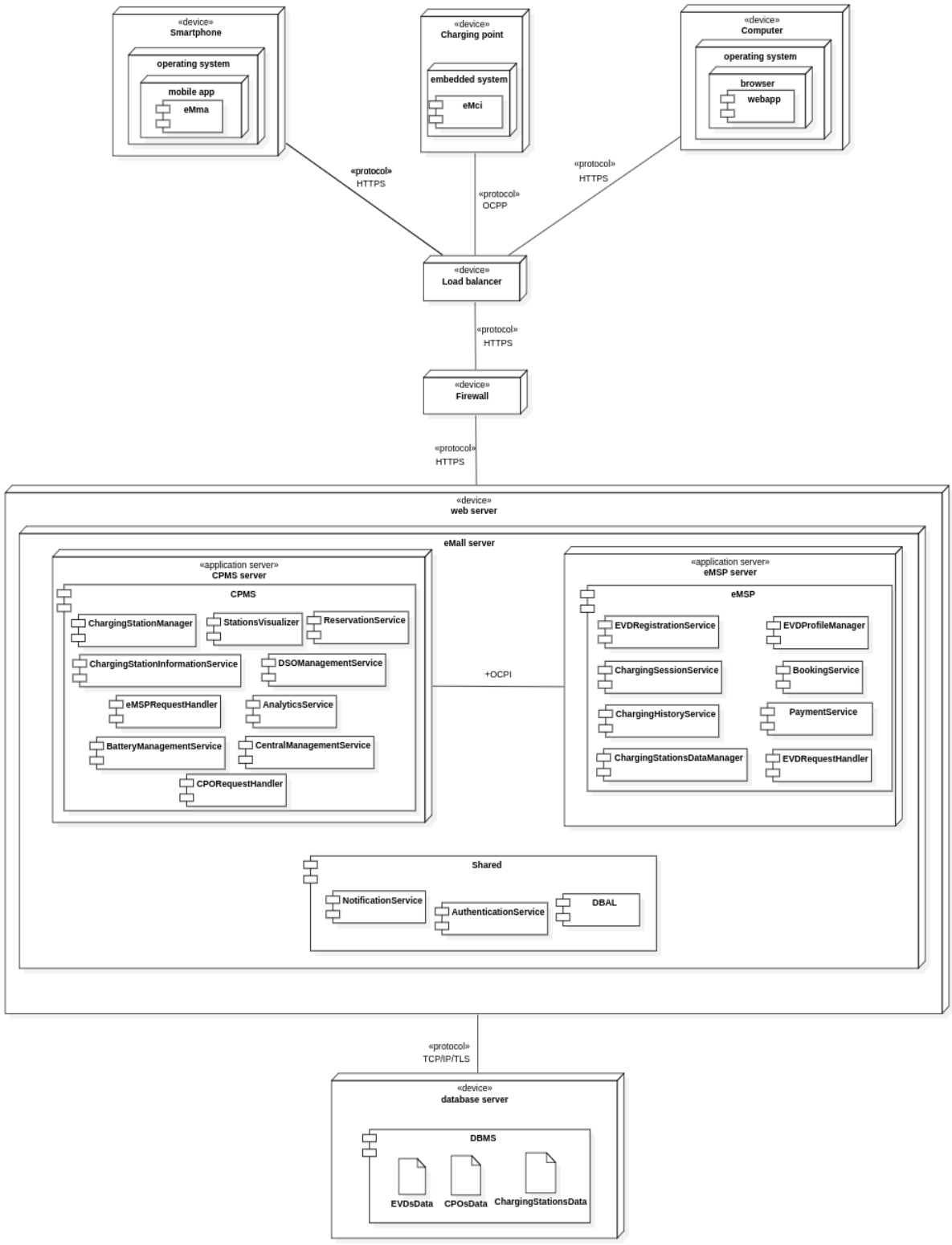

Figure 2.5: Deployment view of the system

At the client level we can see three different devices interacting with the system:

- **Smartphone** - The smartphone runs the mobile application of the eMall (the eMma) and has internet access in order to send the HTTPS requests to the system. This is the kind of device used by the EVDs that use the eMall
- **Computer** - The computer runs the web application of the eMall, and also has internet access to manage the service through HTTPS requests to the system. This is the kind of device used by the CPOs of the companies that use the eMall to manage their charging stations
- **Charging Point** - The charging point is the specific device used by the EVDs to charge the EVs and it has an embedded system, which through the OCPP protocol communicates with the CPMS part of the eMall system, in order to correctly provide the charging service

Between the client level and the application level, we have some architectural elements that allow to achieve some non-functional requirements, such as better performance, scalability, availability and security:

- **Load balancer** - The load balancer is a network device that distributes incoming requests across a group of servers to help improve the performance and availability of the application. A load balancer can help to improve the performance of the application by distributing incoming requests across multiple servers, rather than routing all requests to a single server. This can help to prevent any single server from becoming overloaded, which can improve the overall responsiveness and performance of the application. A load balancer can make it easier to scale the application horizontally by allowing to add or remove servers as needed and this can be useful if we need to add more capacity to handle a growing number of users. A load balancer can, also, help to improve the availability of the application by routing traffic to a healthy server in the event that one of the servers becomes unavailable. We can see that the load balancer is useful to improve different non-functional requirements of the eMall, that can be implemented using different servers to have better overall characteristics
- **Firewall** - The firewall allows to protect the network from external threats and unauthorized access, blocking incoming traffic that does not meet the security rules. The firewall is necessary in order to comply with certain regulations and industry standards, because we are handling sensitive data (financial information, personal data), so is necessary to protect the data

The eMall web application and mobile application provide both static and dynamically generated content, so the system runs web servers for the static content and application servers to generate content dynamically. The load balancer sends to the web server the HTTPS requests that need only static content, and on the other side sends to the correct application server the requests to generate the dynamic content and accomplish more complex functionalities due to the interaction of the eMall components. At the application level the deployment diagram shows in a simplistic way the following elements:

- **Web server** - For the web server we have a computer that stores software and website raw data, such as HTML files, images, text documents, and JavaScript files. The hardware of the web server is connected to the web and supports the data exchange with different devices connected to the Internet
- **Application server** - In the deployment diagram on the same hardware we also have the application servers, one for the CPMS and one for the eMSP, with also the shared services. The application servers contain different micro-services, that interact among them and with external APIs, as shown in the previous component diagram. The micro-services could also be implemented on more servers, splitting the eMSP and CPMS application servers in more servers, or creating a redundancy of the available services on different machines to improve performance and availability, exploiting even better the load balancer
- **Shared components** - The shared components shown in the diagram are components that belong to the two application servers, but also to the web server. The web server of the eMall handles part of these functionalities, sending the rest to the application servers

Finally at the lowest level we have the persistent part of the system, which interacts with the eMall through TCP/IP over TLS and is hidden to the higher levels due to the use of the DBAL. The Database level is composed by:

- **Database server** - It hosts the database of the system and manages the different data through a DBMS
- **Database artifacts** - The different database artifacts shown in the diagram represent physical implementations of the DB, an implementation for the data regarding the EVDs, one for the companies and CPOs data and finally an implementation for the data regarding the charging stations and any other related data

For a more secure system we consider not only a DB, but also some replicas, distributed on different machines to guarantee more availability, fault tolerance and disaster recovery.

2.4. Runtime view

In this section a dynamic view of the system is provided, displaying the interactions between the components identified in section 2.2 by mean of sequence diagrams in UML notation.
For the sake of conciseness, repetition of interactions that exhibit the same, or similar, behaviour has been avoided, hence, only the most important interactions emerging from the use case presented in the RASD document are shown here. Furthermore, we mainly present the main interaction of the use cases, disregarding the exceptions, which are shown only in some situations to underline important differences in the flow of events.

2.4.1. EVD runtime view

EVDregistration

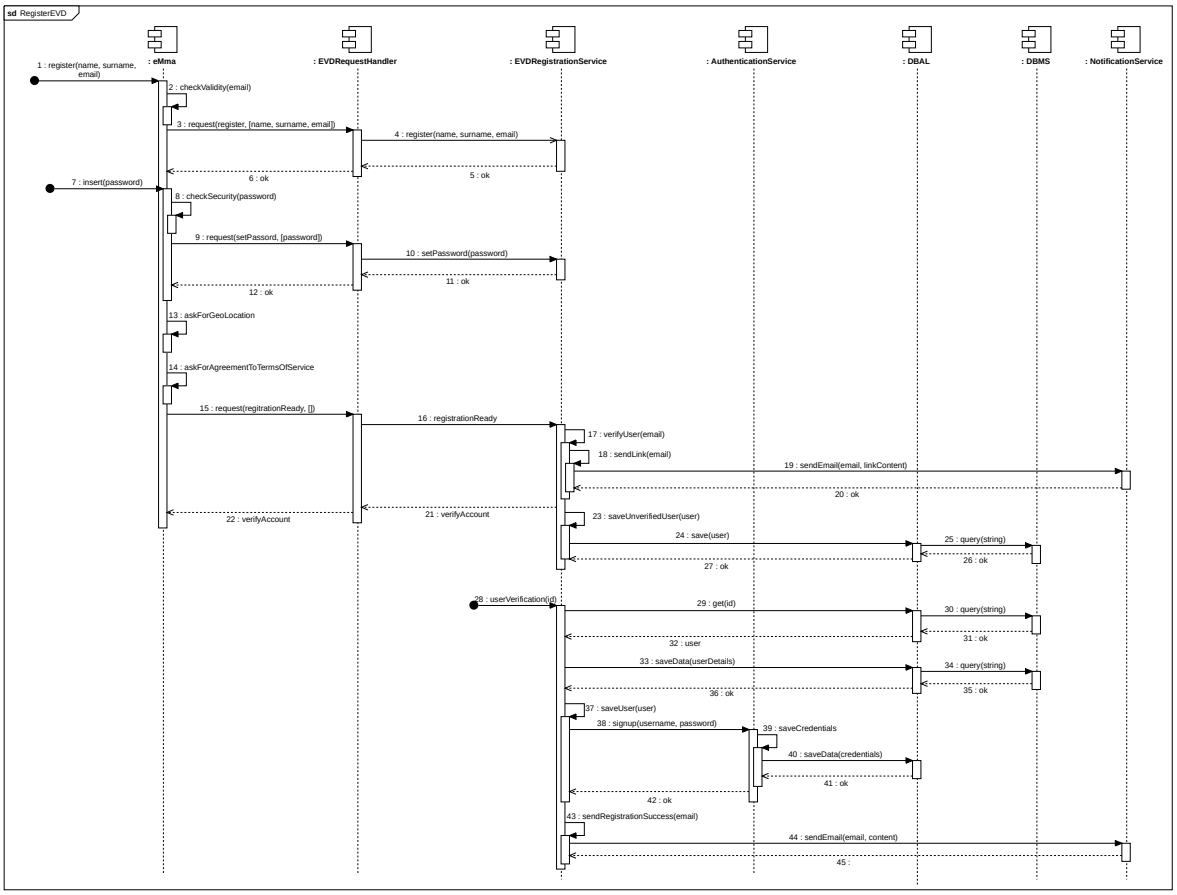

Figure 2.6: EVD registration sequence diagram

This diagram shows the interactions taking place between the eMma component, responsible for the interaction with the user, and the EVDRegistrationService, tasked with handling the registration process. As shown in the component diagram (section 2.2) all the interactions between eMma and the component of the eMSP server must pass through the EVDRequestHandler (ERH from now on) so it can check whether the operations needs special authorization or not. In this specific case the operations do not need any special authorization so the ERH will only dispatch the requests to the right component. The first "found message" represents the user interaction on eMma to start a registration process. eMma, tries first to do a simple check on the validity of the email, by checking if it has the right format. If the controls are successful, eMma starts the interaction with the eMSP server by sending the request with the data to the eMSP server. This request starts the

registration process in the EVDRegistrationService (ERS from now on). In this first step the ERS will acknowledge the request and temporary store the data. The next step is for eMma to check for the password when the user inserts it, and then sends the data to the ERS. Next, eMma requests the user to share his geolocation and to agree to the terms of service. If the user agrees eMma sends a message to the ERS to notify him that the data necessary to the registration were gathered. Now the ERS starts a process of verifying the user by sending him an email with a link to verify his data. After he email is sent, the ERS sends the verifyAccount message to eMma and then saves the EVD data into the DB so they can be retrieved later when the user verifies his identity. When the user does so, the ERS proceeds to save the user by asking the AuthenticationService to validate the user. Finally the ERS sends an email to the user notifying him the registration was successfully completed.

Login

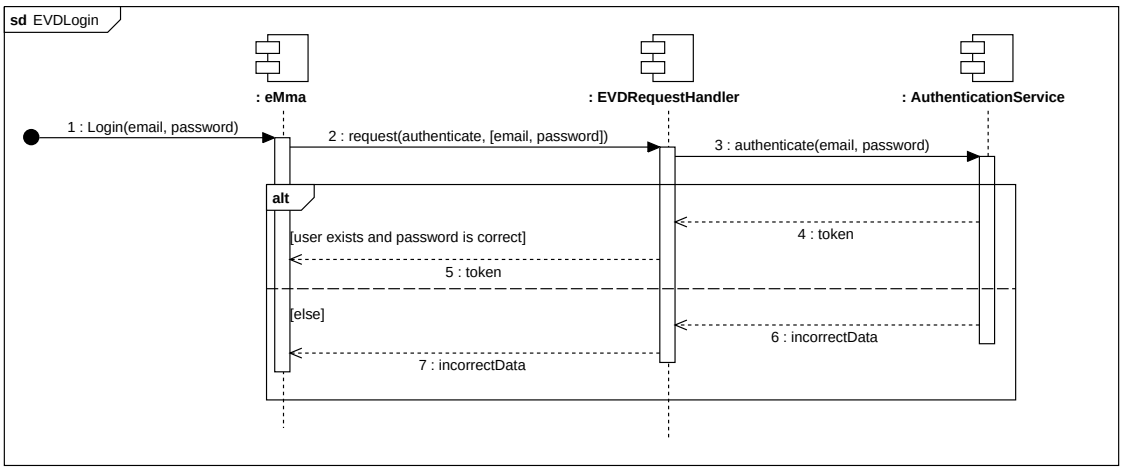

Figure 2.7: EVD Login sequence diagram

This set of interactions is as simple as shown in the diagram. The user starts the interaction by requesting to login in the system and providing the email and password. eMma forwards this request to the eMsp server through the ERH, which passes it to the AuthenticationService. If the email exists and the password is correct then the AuthenticationService returns a token that will be used to access the system services, if not it returns an error message.

Authorized request

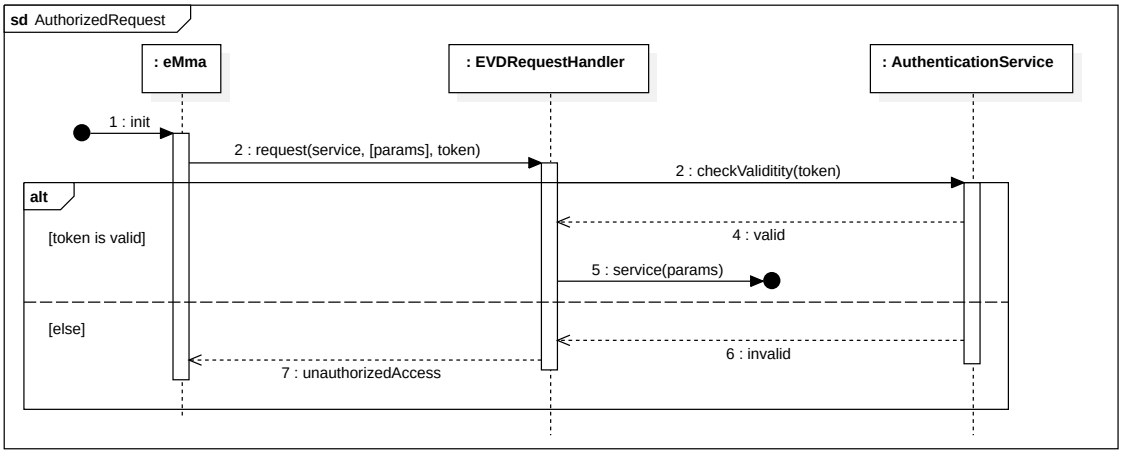

Figure 2.8: Authorized request sequence diagram

This sequence of interactions doesn't map to any particular use case, but it's an important flow of events that shows how an authorized request is controlled to have the right access permissions. This sequence of interactions will always occur for any authorized request arriving to the system, for both eMSP and CPMS, and for the sake of conciseness it will be considered implicit in the following diagrams. The diagram is left generic precisely because how general this sequence of actions is. As always the interactions start with an outside message that triggers eMma to invoke a service of the eMSP. Along with the arguments needed to request the service, for authorized operations, eMma needs also to yield the token obtained during the login phase. Secondly, the ERH asks the AuthenticationService to check if the token is valid: if it is, then it proceeds to call the methods of the required service with the arguments it received from eMma. If the token is invalid, ERH sends an *unauthorizedAccess* message to eMma.

Booking

For this interaction we have decided to split the diagrams in two, showing first the interactions that occur in the eMSP subsystem, and afterwards showing the sequence of actions that happen in the CPMS.

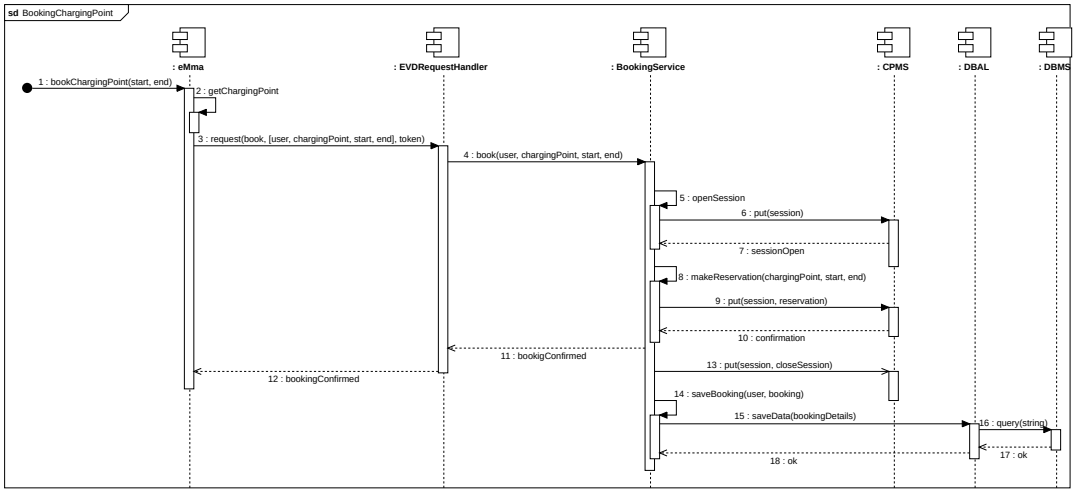

Figure 2.9: Charging point booking sequence diagram

The interaction starts when the user inserts the start time and end time of the booking (notice that this are timestamps, so the date is already encoded). Next, eMma retrieves the data of the charging point the user has selected in the view and sends the request to the eMSP. The request, after validating the token, is forwarded to the BookingService. The latter component must interact with the CPMS to make the reservation. The interactions between the BookingService and the CPMS follow the standard stated in the OCPI protocol. When the reservation is confirmed by the CPMS, the BookingService sees to send the confirmation message back to eMma.

From the perspective of the CPMS we show the sequence of actions, considering the session is already started, hence the interactions start from the PUT method from the eMSP making a request for a reservation. The eMSPRequestHandler validates the request analyzing the token embedded in session object (like we have shown in diagram 2.8) and then dispatches the request to the ReservationService component, which unpacks the reservation object to extract the data he needs and makes an availability control on the charging point and the time span requested.

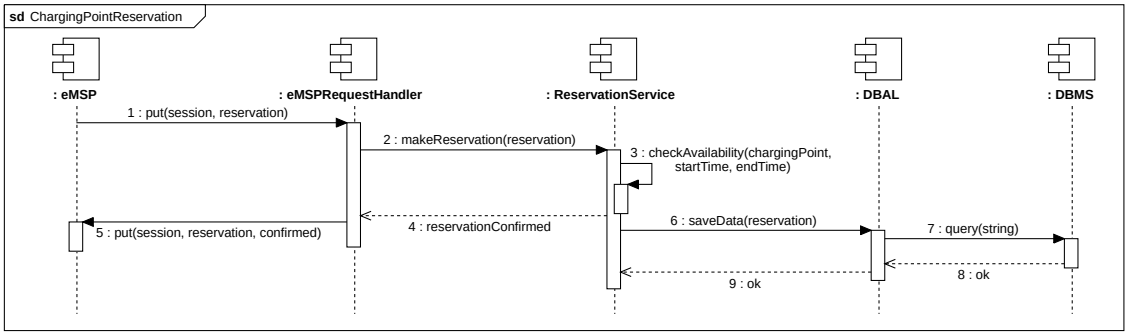

Figure 2.10: Charging point reservation sequence diagram

If the controls are positive then the ReservationService proceeds to send a confirmation message back to the eMSP and saving the reservation in the DB to update the information about the station.

VisualizeStations

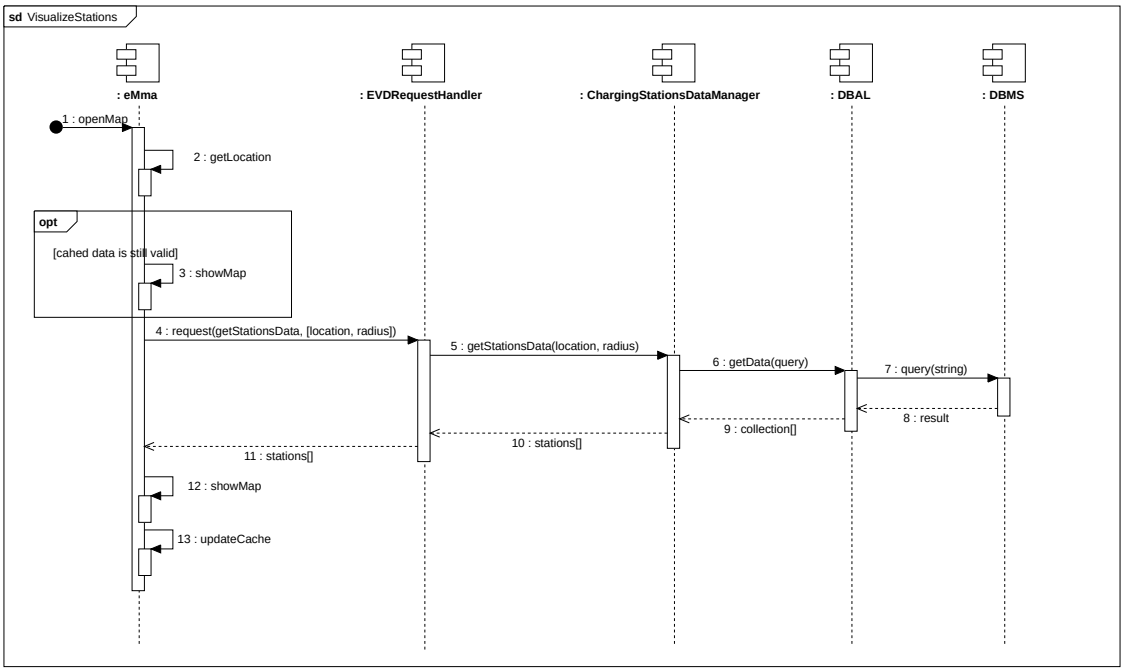

Figure 2.11: Visualize stations sequence diagram

The process of visualizing the stations starts when the user opens the app and is greeted with a map. Next, eMma gets the location information from the user smartphone as shown with the self call *getLocation*. Before asking the stations' data to the eMSP server, eMma tries to first check if the data it has cached is still valid and if it is, then it generates the map based on this data. If the cache is invalid, eMma requests to the eMSP server the data about the stations. Please note that the request doesn't need authorization so the token is not passed as an argument. The request is then dispatched to the ChargingStationsDataManager, which through a series of methods call to the DB retrieves the necessary data and returns it back to eMma.

ChargeNow

The starting of a charging session is an operation that involves the eMSP and the CPMS as well. As we did before we will split the flow events in two, showing fist the interaction from the eMSP point of view, afterward we will display the interactions from the CPMS point of view.

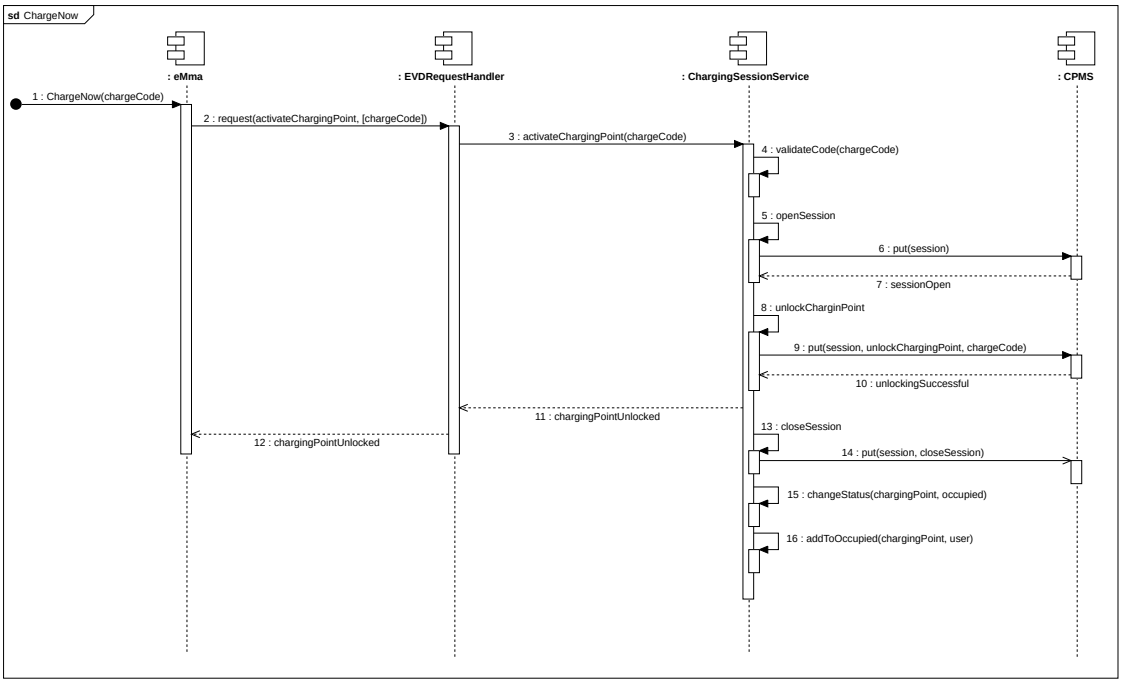

Figure 2.12: Start Charging session sequence diagram

The interaction starts with the insertion of the chargeCode by the EVD, which he gets from the display of the eMci device installed on the charging point. The eMma forwards the request of activating the charging point with the respective code to the eMSP server, which handles it through the EVDRequestHandler. The latter, given the request, invokes

the right method on the ChargingSessionService component (CSS from now on). After the method is invoked on it, the CSS performs a preliminary check on the format of the chargeCode and if the chargeCode exist in the system. If the controls pass the CSS opens a session with the CPMS to request the unlocking of the charging point associated with the aforementioned code. This sequence of actions is very similar to the ones already described in the Booking sequence diagram, thanks to the uniform OCPI interface. If the CPMS unlocks the charging point correctly, then the CSS proceeds to send back to eMma the reply message for the correct conclusion of the request. However, the CSS task doesn't end here, obviously he has to close the session with the CPMS but he also needs to change the status of the charging point to occupied and he has to keep record of which user is using which charging point, so it can show the charging status of the EV to the respective EVD.

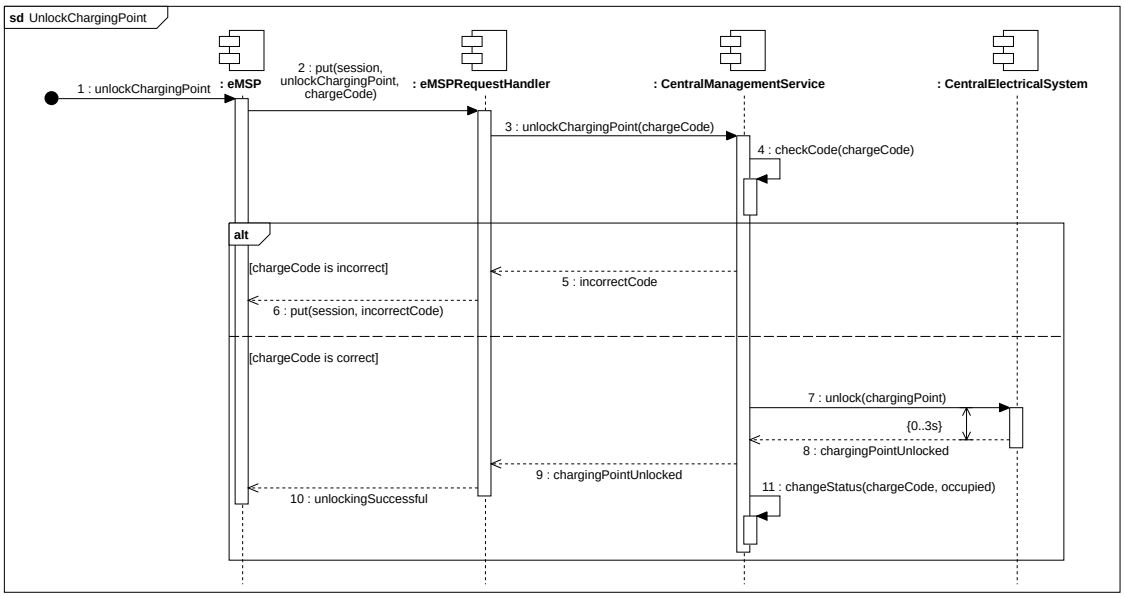

Figure 2.13: Unlock charging point sequence diagram

The diagram starts directly when the CPMS receives the PUT message to unlock a specific charging point delineated by the chargeCode. The request is handled by the eMSPRequestHandler which, after doing all the security checks, dispatched the request to the CentralManagementService (CMS) which is the component responsible for it. The first operation of the CMS is to check if the code refers to a charging point actually present in the station. If this is not true, then an error message is returned. If the test passes, then the CMS starts the operations of unlocking the charging point, which consist of preparing a correct data structure of the charging point, starting from the chargeCode, to be sent as the content of the message to the CentralElectricalSystem, which we remind is the

system that manages all the technical aspects of the charging station. Next, the CMS sends the reply message and proceeds to change the status of the specific chargingPoint to occupied.

2.4.2. CPO runtime view

VisualizeGraphicalInformation

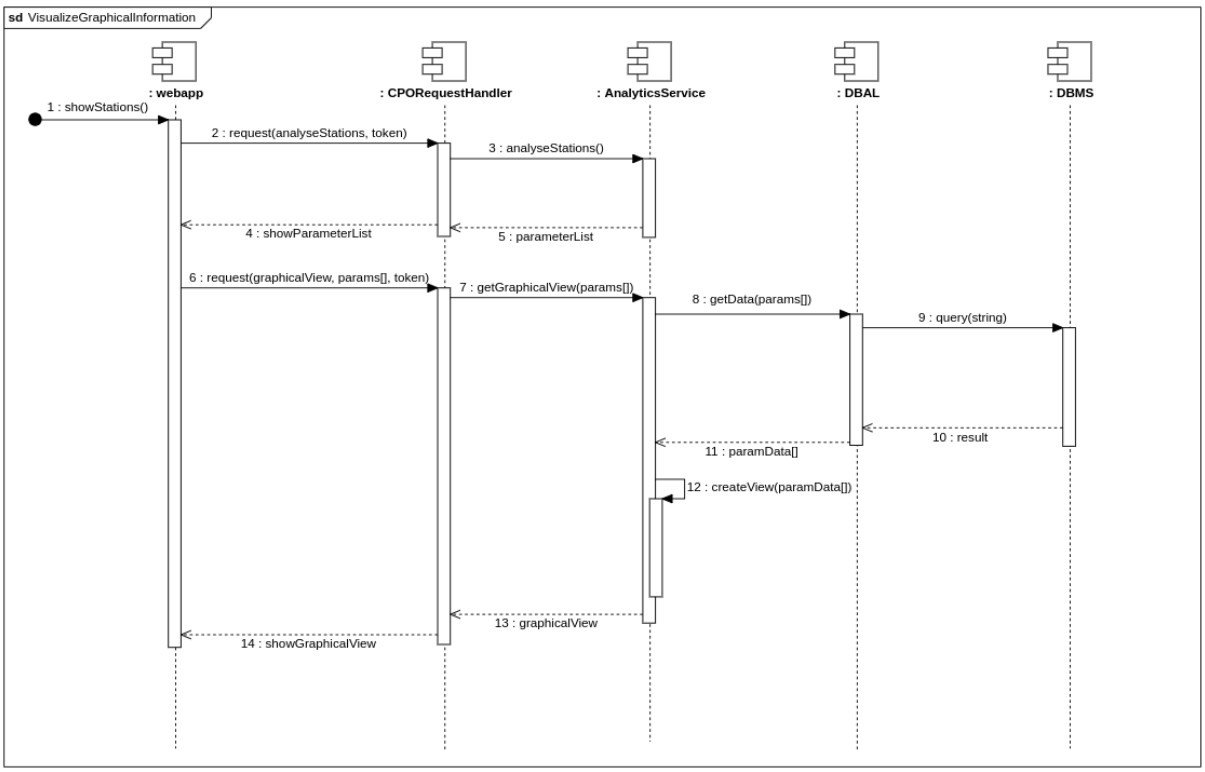

Figure 2.14: Visualize graphical information sequence diagram

The interaction starts when the CPO enters into the section available to analyse the stations. The web application returns a page with a list of parameters that the user can select in order to get a graphical view of the stations progress based on these criteria. Once selected the parameters the user confirms the operation, so he requests the graphical view based on his choices. The request is passed to the AnalyticsService component, which gets the needed data, depending on the desired analyses, and creates the graphical view using the acquired information. Then the graphical view is sent back to the web application, through the CPORequestHandler, and shown to the user.

StoreEnergyInBattery

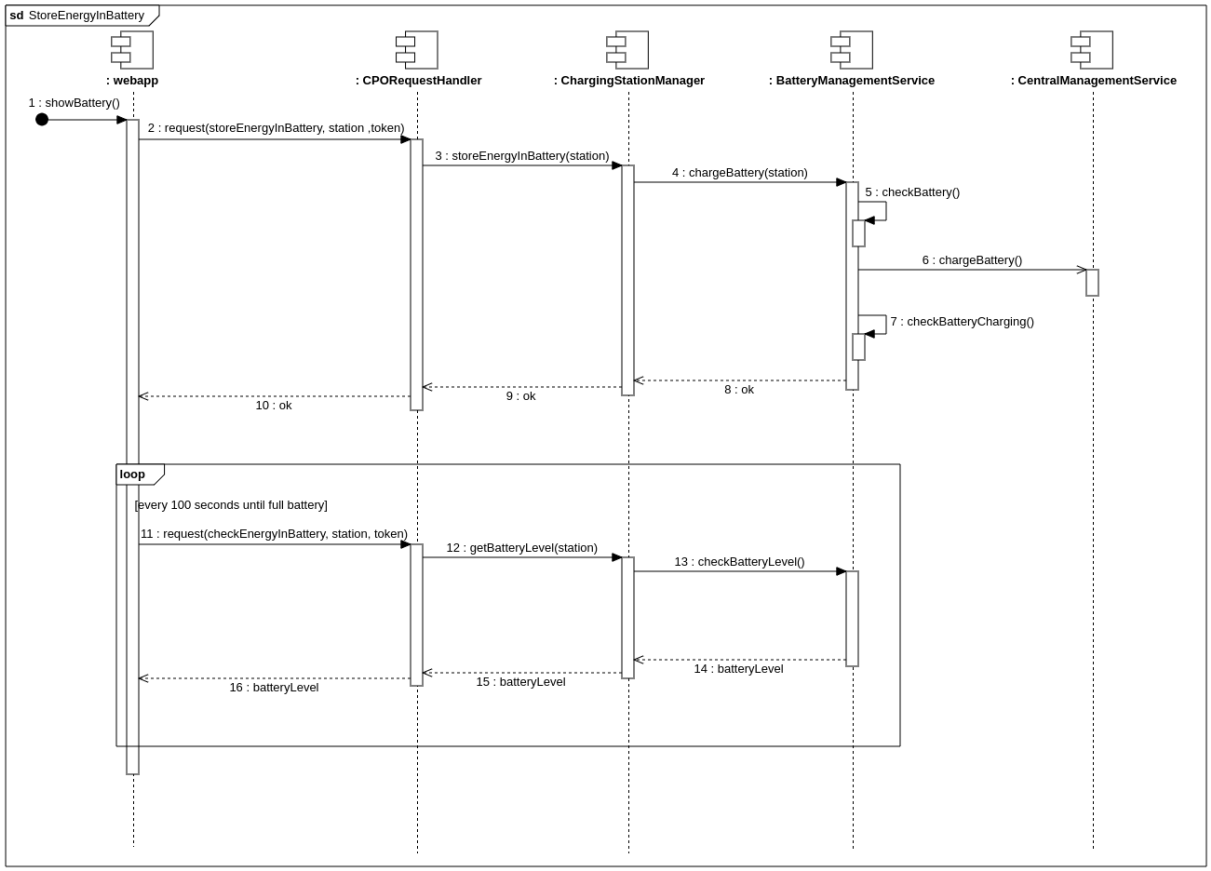

Figure 2.15: Charge battery sequence diagram

Once the CPO decides to store energy in the battery of a certain station that is equipped with it, the CPORequestHandler sends the request to the ChargingStationManager, which is the component that manages the changes of the stations, but it needs to interact with the BatteryManagementService, which checks the battery to see if it is full or can actually be charged and also checks if the battery is functional. If the battery can be charged an asynchronous message is sent to the CentralManagementService, that manages the technical aspects of the charging. The manager of the battery, then, controls if the battery is charging, and if the charging was correctly initiated the request is fulfilled. Finally, in loop, the web application interrogates the BatteryManagementService to know the level of the battery, keeping in this way track of the charging process and showing it to the user.

AddPromotion

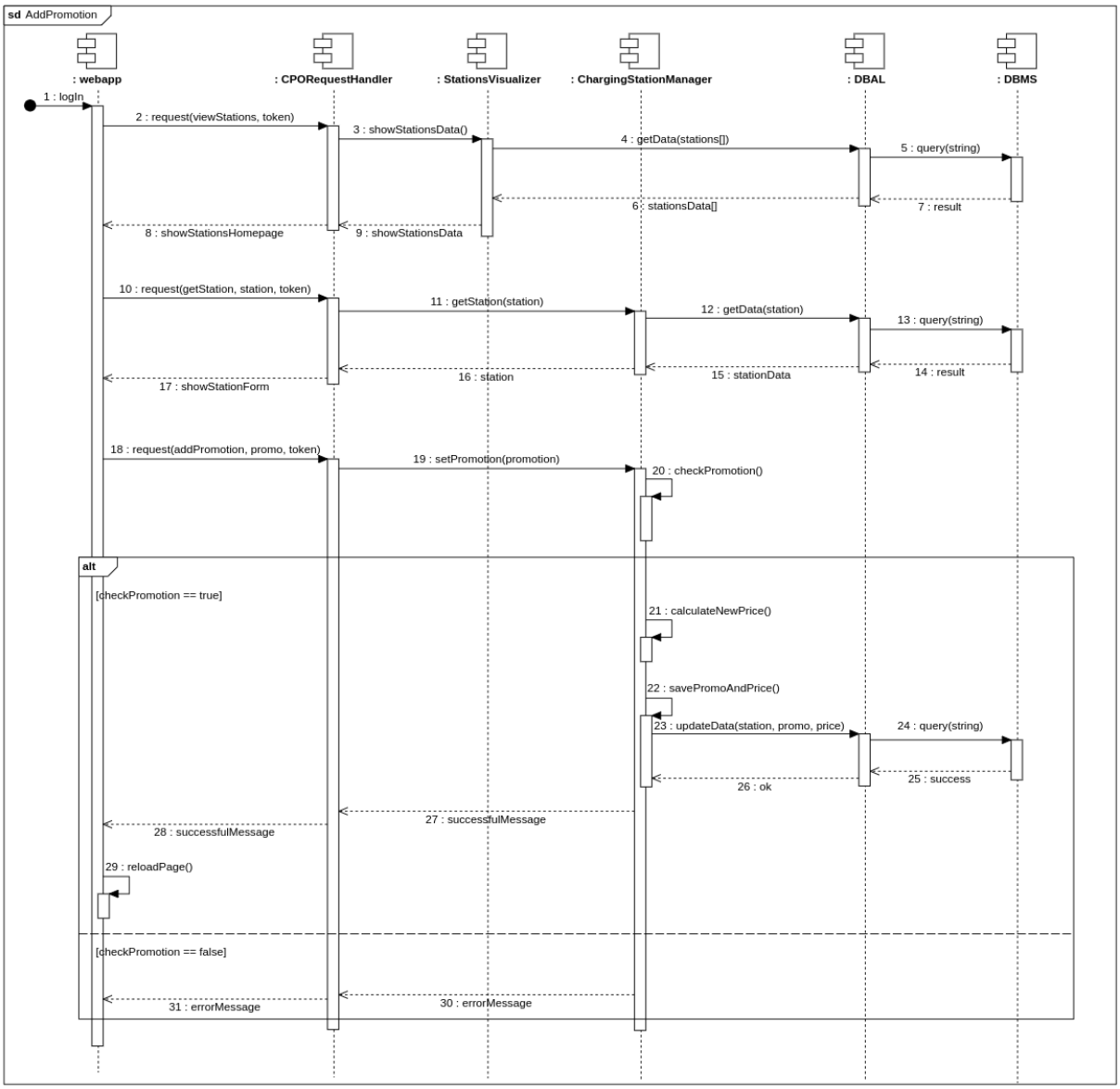

Figure 2.16: Add promotion to charging station sequence diagram

To add a promotion to a charging station the CPO first selects the specific station, and then sets the promotion. The promotion is sent to the ChargingStationManager which checks the promotion, if it is compliant with the stations. If the promotion is acceptable the price is updated accordingly and the promotion and the price are saved on the DB. If the update is successful the page is reloaded. In this sequence diagram we see the main use case and the alternative one that treats the case of inserting an incorrect promotion, because we want to show how in case of error a lot of operations are not needed anymore.

UpdateDSO

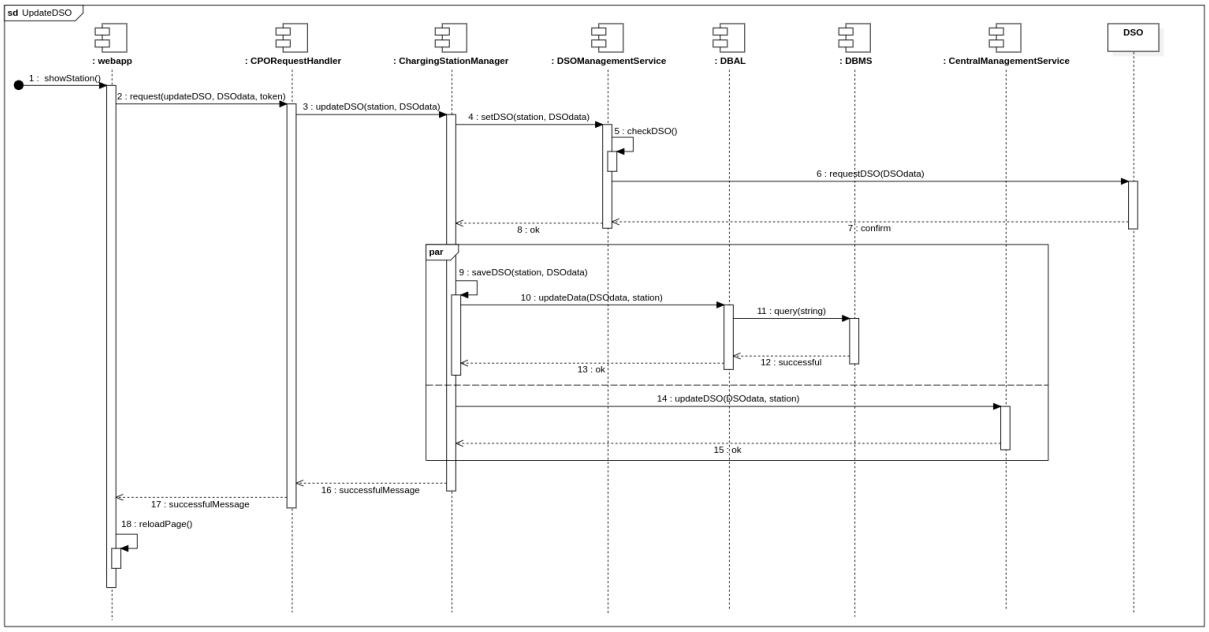

Figure 2.17: Update charging station DSO sequence diagram

Once selected a DSO and the relative data from the form, the station' DSO is updated. The ChargingStationManager sends to the DSOManagementService the station and the selected data of the DSO in order to check the data and send the request to the DSO itself. Once received a confirmation the DSOManagementService completes the update of the DSO and responds to the ChargingStationManager, which acknowledges the success. In parallel this component starts two operations, one that updates the data on the DB passing through the DBAL and DBMS, and another that updates the DSO data on the CentralManagementService. Once everything is updated the user receives a successful message and reloads the page.

2.5. Component interfaces

2.5.1. Common interfaces

In this section we will list the interfaces of the components of out system. Since most of the functions and methods were already described in the previous section, we only give a description of the methods that might not result so straightforward to understand

DBAL

DBAL
+ saveData(data): Boolean
+ getData(object): Object
+ getData(query): Object

The DBAL offers a layer of abstraction from the DBMS. It offers the ability to directly directly save objects with the method `saveData`. Meanwhile for accessing it offers the capability to retrieve data from an object (like a map) or by making a query.

AuthenticationService

Authentication
+ signup(email, password): Boolean
+ authenticate(email, password): Boolean
+ checkValidity(token): Boolean

This component offers the classical interface for signing-up a user, likewise authenticate him and moreover provides the capability of validating the token possessed by the clients.

NotificationService

NotificationFormt
+ notify(receiver, message)
+ sendEmail(email, content)

Notification Service offers the possibility to notify the client application of the user with the method `notify(receiver, message)` and to send emails.

2.5.2. eMSP interfaces

EVDRegistrationService

EVDRegistrationService
+ register(name, surname, email): Boolean
+ setPassword(password): Boolean
+ registrationReady(): Boolean
- verifyUser()
+ sendLink(email)
+ saveUnverifiedUser(user)
+ sendRegistrationSuccess(email)

PaymentService

PaymentService
+ makePayment(paymentProcessor, credentials, amount): Boolean
+ savePayment(credentials, amount, timestamp): Boolean

With the method `makePayment` the PaymentService allows to user to perform a payment. Meanwhile the `savePayment`, saves the payment instance on the DB.

ChargingHistoryService

ChargingHistoryService
+ getChargingHistory(user): ChargingSession[]
+ getChargingHistory(user, filter): ChargingSession[]
+ update(ChargingSession, updatedChargingSession): Boolean
+ delete(ChargingSession): Boolean

EVDProfileManager

EVDProfileManager
+ createProfile(name, surname, email, password)
+ updateUserDetails(user, field, newValue): Boolean
+ addEV(ev): Boolean
+ updateEV(user, ev, field, newValue): Boolean
+ deleteEV(user, ev): Boolean

ChargingSessionService

ChargingSessionService
+ activateChargingPoint(chargingPointCode): Boolean
- openSession()
- closeSession()
- unlockChargingPoint()
+ changeStatus(chargingPoint, value)
+ addToOccupied(chargingPoint, user)
+ getStatus(chargingPoint): ChargingStatus

BookingService

BookingService
+ bookChargingPoint(user, chargingPoint, startTimestamp, endTimestamp): Boolean
- openSession()
- closeSession()
- unlockChargingPoint()
+ makeReservation(chargingPoint, startTimestamp, endTimestamp): Boolean
+ cancelBooking(user, booking): Boolean
+ saveBooking(user, booking)

ChargingStationDataManager

ChargingStationDataManager
+ getStationsData(location, radius): Station[]
+ getStationFromChargingPoint(chargingPoint): Station
+ getStationsChargingPoints(station): ChargingPoint[]
+ collecDataFromCPMS(identifier)
+ updateDataFromCPMS(CPMS)
+ getStatus(chargingPoint): ChargingStatus
- openSession()
- closeSession()

The method `getStationsData(location, radius)` returns a collection of Station data structure containing all the information related to the charging stations that are are in the radius of the location provided as argument.
Moreover, the `collectDataFromCPMS(identifier)` allows the system to collect the data of a new charging station managed by the CPMS identified by the identifier argument, which encodes the information to reach it. Meanwhile the method `updateDataFromCPMS(CPMS)` updated the data of an already known CPMS.

eMci

eMciControlAPI
+ setCode(string)
+ getCode(): String

The `setCode(string)` allows to set the code that the user will use to start a charging session, while `getCode()` can be used by the eMSP to know which code is set for that specific eMci.

EVDRequestHandler

EVDRequestHandler
+ request(service, parameters[])
+ request(service, parameters[], token)

The interface for the EVDRequestHandler is very simple, indeed it is composed of only two methods: one for the operations that do not require authorization and one for those that require it, as can be seen by the token required as argument. The signature tells us that to obtain the services of a component the requester shall indicate the service it want with the arguments to obtain it. This solution allows to implement the decouple the service offered from the component (or micro-service offering it), since will be the EVDRequestHandler to choose which component will respond to the request.

eMma

Notification
`+ notifyUser(content)`

Interfaces implemented to allow the eMSP to notify the user in an asynchronous way.

2.5.3. CPMS interfaces

ReservationService

ReservationService
`+ makeReservation(reservation): Boolean`
`+ checkAvailability(chargingPoint, startTimestamp, endTimestamp)`
`+ cancel(reservation): Boolean`

ChargingStationInformationService

ChargingStationInformationService
`+ getChargingStationInfo(): JSONFile`
`+ getChargingPointsAvailability(): JSONFile`

This components through the method `getChargingStationInfo()` returns a JSON file containing all the information regarding the charging station, including the charging points details. Meanwhile the `getChargingPointsAvailability()` is used to query about the availability of the charging points. This method is included because we preview

that this operation will happen more frequently and consequently this method is computationally less intensive and returns a lighter message.

eMSPRequestHandler

eMSPRequestHandler
+ put(JSONObject[])
+ get(JSONObject[])
+ post(JSONObject[])
+ delete(JSONObject[])
+ patch(JSONObject[])

This components implements the interface described for the OCPI protocol, here we only provide the list of methods this interface requires. For further information about the implementation of the OCPI protocol we reference the OCPI documentation[1].

CentralManagementService

CentralManagementService
+ unlockChargingPoint(chargeCode))
+ monitorChargingPoint(chargingPoint[])
+ chargeBattery()
+ disconnectBattery()
+ updateDSO(DSOData, station)

[1] www.evroaming.org/app/uploads/2021/11/OCPI-2.2.1.pdf

DSOManagementService

DSOManagementService
+ setDSO(station, DSODdata)
+ checkDSO()
+ addNewDSO(DSOdata)
+ refreshAllInfo()
+ refreshDSOInfo(DSO)
+ updateDSO(DSOData, station)

BatteryManagementService

BatteryManagementService
+ chargeBattery(station)
+ stopChargingBattery(station)
+ getBatteryStatus(station): Status
+ getBatteryLevel(station): Percentage
+ setBatery(batteyCapacity)
- checkBattery()
- checkBatteryCharging(): Boolean

The `checkBattery()` is a private method used by the BatteryManagementService to check if the battery is functioning correctly (e.g. it's not broken). Meanwhile `checkBatteryCharging()` check if the battery is charging. **StationsVisualizer**

StationsVisualizer
+ showStationsData(): Stations[]
+ selectStation(id): Station

The `showStationsData()`, returns the collection of the stations managed by he CPO. By using the `selectStation(id)` allows to get the Station from it's identifier.

AnalyticsService

AnalyticsService
+ analyseStations(): Parameter[]
+ getGraphicalView(parameter[]): GraphicalView
+ createView(paramData[]): GraphicalView

ChargingStationManager

ChargingStationManager
+ storeEnergyInBattery(station)
+ setPromotion(promotion)
+ setPrice(chargingPoint, price)
+ setPrice(station, price)
- calculateNewPrice()
- SavePromoAndPrice()
+ updateDSO(station, DSOdata)
+ getDSOdata(): DSO[]
+ addNewDso(DSOdata)
+ storeEnergyInBatttery(station)
+ stopBatteryCharging(station)
+ batteryStatus(station): Status
+ getBatteryLevel(station): BatteryLevel
+ setBattery(station, batteyCapacity)

This is the component which interacts with the webapp and hides the internal implementation of the other components, hence many of its methods are not very simple, only calling homonymous methods of other components.

DSORequestHandler

DSORequestHandler
+ request(service, parameters[], token)

The interface and behaviour of this component is identical to the EVDRequestHandler.

webapp

Notification
`+ notifyUser(content)`

2.6. Selected architectural styles and patterns

A 3-layered Architecture The main architectural choice is to shape the application following the 3-layer architectural model, dividing the application into three main layers, as we can see in the deployment diagram. The three main layers are: the presentation layer, the business logic layer, and the data access layer. Each layer has a specific role and communicates with the other layers through well-defined interfaces.

- **Presentation layer**: This layer is responsible for handling the user interface and user interaction with the application. It includes the mobile app, the web app and the charging point interface
- **Business logic layer**: This layer is responsible for implementing the core functionality and business rules of the application. It includes all the micro-services with the respective code that handles all the functionalities to reach the goals of the system. These modules validate data, communicate with external systems and update the information of the charging stations and of the EVDs, in order to use the system on one hand to manage the stations and on the other hand to take advantage of the service itself, which is transparent and allows the booking and the charging of the EVs as main operations
- **Data access layer**: This layer is responsible for interacting with the database and providing data to the business logic layer as needed. It includes the DBMS, that handles tasks such as querying the database, inserting data, or updating records and this is an important tool to create, delete or update the database in a very fast and efficient way

There are several advantages in using a 3-layer architecture in an application with a mobile app for the users and a web app for the business managers:

- **Separation of concerns and flexibility**: A 3-layer architecture helps to separate the different concerns of the application into different layers, which can make it easier to understand and maintain the codebase. For example, the presentation layer handles tasks related to rendering the UI and handling user input, while the business logic layer handles tasks related to calculating prices and validating data. This separation of concerns can make it easier to understand the different parts of the application and make changes to one part without affecting the others, so this gives more flexibility than a monolithic architecture

- **Modularity**: A 3-layer architecture also promotes modularity, which means that the different layers of the application can be developed and tested independently. This can make it easier to update and maintain the application, and allows to create better implementation and test plans
- **Scalability**: A 3-layer architecture can also be more scalable than a monolithic architecture, as it allows you to add or remove servers or resources as needed to handle an increase in traffic or load. This can be useful if the eMall experiences an increase in the number of users

MVC In the explained 3-layer architecture is embedded the MVC pattern:

- **M - Model**: maps on the business layer and provides all the services to achieve the functionalities of the system and to access the database and acquire the necessary data to correctly provide information to the users
- **V - View**: maps on the presentation layer and manages the interaction between the user and the system, and we have different views for the EVD and the CP
- **C - Controller**: receives the requests of the users and sends them to the interested services, so it is an intermediate component that hides the model from the view and communicates with both parts, creating a modularity of the system and allowing the separation of concerns and more flexibility as previously described in the 3-layered architecture

In the eMall we have as view the eMma, the eMci and the web app interface, which send the user requests to the specific controller, the eMSPRequestHandler and the CPORequestHandler respectively, that forward the request to the specific micro-services of the application layer. Also, from the application layer the components interact with the DBMS through the DBAL and acquire the data needed to process the request and respond to the user, sending the result back to the view passing through the controller.

Micro-services for the business layer Using the micro-services architecture for the business layer, the application is divided into a set of small, independently deployable services that communicate with each other. Each service is responsible for a specific function or business capability, and it is designed to be highly modular, scalable and resilient.
The eMma and the web app, through the interface of the presentation layer, communicate with the business logic layer via a set of micro-services, each of which is responsible for a specific service and interacts with the others to achieve the requirements of the system to be.

In the component diagrams we show these micro-services and the interaction between these modules. We separated them in the ones necessary for the eMSP server and the ones necessary for the CPMS server, underlying also the common services (as can be seen in the deployment diagram) that both parts of the system need to communicate with, such as the NotificationService, the AuthenticationService and the DBAL.

The micro-services architecture is well-suited to our system because being composed of independent services, if one fails, it will not bring down the entire system. This is perfect for our system because we want some services to be always running independently from the others. For example the CharginSessionService and BookingService are core services that should always be actively running and they do not depend from other secondary services like the EVDRegistrationService or the EVDProfileManager. The same can be said for the CPMS, the components granting access to the services of the charging stations to eMSPs should be independent from those that allow the CPO to manage the charging station, hence even if the CPO can't access the system to change some parameters, like updating the price, for the charging station, the EVDs can still book or start charging sessions. Furthermore, the system benefits from improved scalability. If the number of EVD increases we can increase the instances of the different handlers services (EVDRequestHandlers, eMSPRequestHanlder) to cope with the load increase, or alternatively, we could deploy these services in more powerful machines.

2.7. Other design decisions

Client-side rendering for the presentation layer Implementing the presentation layer with a client-side rendering approach, the web app and the mobile app on the client side are responsible for rendering the user interface. In a client-side rendering architecture, the server typically responds to requests from the client by returning data that the client uses to dynamically generate the UI and this can be done using a front-end JavaScript framework such as React or Angular. With client-side rendering, the initial page load is naturally a bit slow. However, after that, every subsequent page load is very fast, because there is no need to reload the entire UI after every call to the server. The client-side framework manages to update UI with changed data by re-rendering only some particular elements. This architecture can offer several benefits, such as better performance because the server doesn't have to generate the HTML on every request, and a better user experience, without requiring a full page reload. We chose this approach mainly for the better user experience, having a more responsive application, and to manage a large number of clients without overloading the server.

DBAL and DBMS to abstract the interaction with the databases The DBAL is an application programming interface which unifies the communication between a computer application and databases. Database abstraction layers reduce the amount of work by providing a consistent API to the developer and hide the database specifics behind this interface as much as possible, so it allows to abstract from the technology adopted for the DB. This interface, then, passes the requests to the DBMS, which handles the main operations on the specific databases and provides several important functionalities[2]:

- Data Storage Management: creates a database for complex data and manages the data and allows the users to access it and manipulate the data very easily
- Security Management and Multi User Access Control: provide a high level of security measures using various security algorithms to keep the data safe and ensure the data privacy, which is very import in our application which treats sensitive data of the EVDs, such as payment method and personal data, and also treats private business information for the management of the stations. There are certain security rules that ensure what data can be accessed from the database and which user can access it assuring security and privacy. So with the DBMS we can manage the different multi-user databases, shown in the deployment diagram: the EVDs database, the CPOs database and the shared database with the charging stations data. Also, more than one user can access the database at the same time without any problem, because the DBMS makes sure the integrity of the data present in the database is preserved and manages concurrency problems
- Backup and Recovery Management: to keep the data safe and ensure the integrity, the database system also provides the features for backup and recovery management, so if the system fails due to some reason then it recovers the most of the data
- Database Access Language and Application Programming Interface and also Data Transformation and Presentation: provides the support of various query languages to access the databases and manipulate the data, so the three databases artifacts can use different data models and the DBMS allows to access them in the proper way without concerning the application with this interaction. This module also makes a data transformation, so the eMall programmers do not need to worry about the logical and physical representation of the data

[2]The DBMS information are retrieved from `https://www.javatpoint.com/functions-of-dbms`

3 | User Interface Design

3.1. EVD user interface

3.1.1. Log in

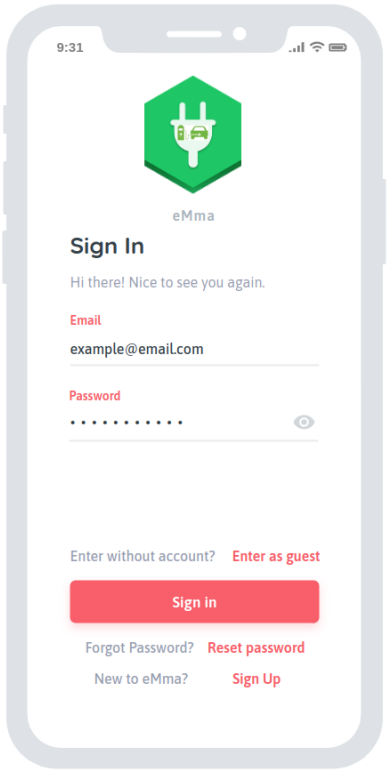

Figure 3.1: UI for the log in

We can see from the UI for the log in that the operation is very simple. To sign up is enough to enter the email and the password used during the registration. The EVD can enter into eMma as a guest, if he doesn't want to sign up, but in that case it will perceive only some limited functionalities of the application. It can also be useful to set a mechanism to manage the case in which the user forgets his password.

3.1.2. Register

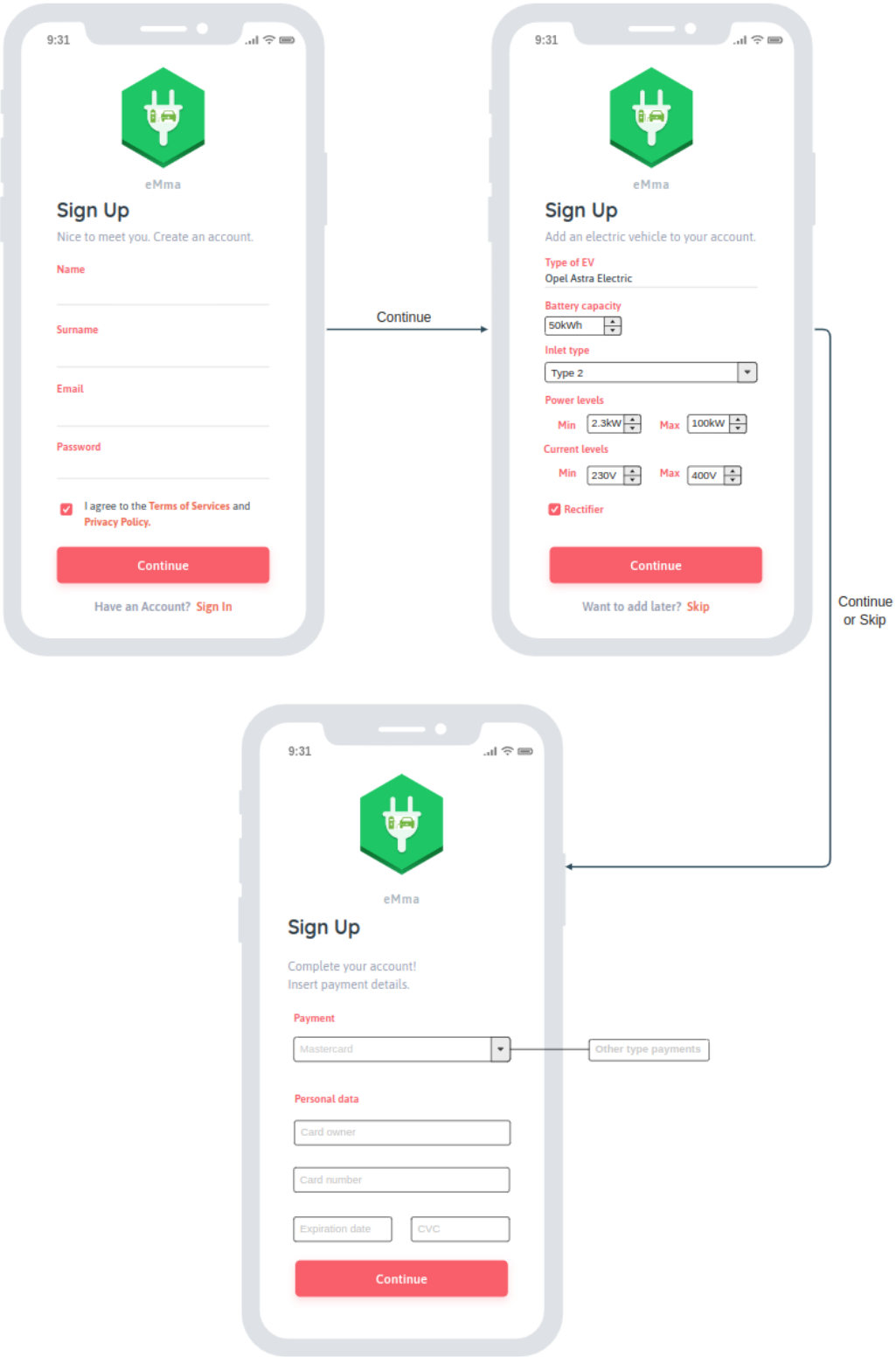

Figure 3.2: UI for registration

The UI for the registration was thought in order to be as user-friendly as possible. First, to sign up is necessary to insert some personal data, such as the name, the surname, and the email and password that will be used for the log in, as previously explained. It is also necessary to accept the 'Term of Service' in order to continue with the registration. The second step asks the user to add some data about his EV and we can see that the insertion is guided by the interface in order to avoid as much as possible data errors. The EVD has to add the type of the EV, the inlet type, the power levels and the current levels, and has to check or not the final box to inform if the car has or not the rectifier. Finally, to complete the sign up process the user is requested to insert the payment details. After registration, from the personal profile the EVD will be able to add new data to his profile, such as new payment methods and new EVs, and he will be able to modify these ones.

3.1.3. Charge now

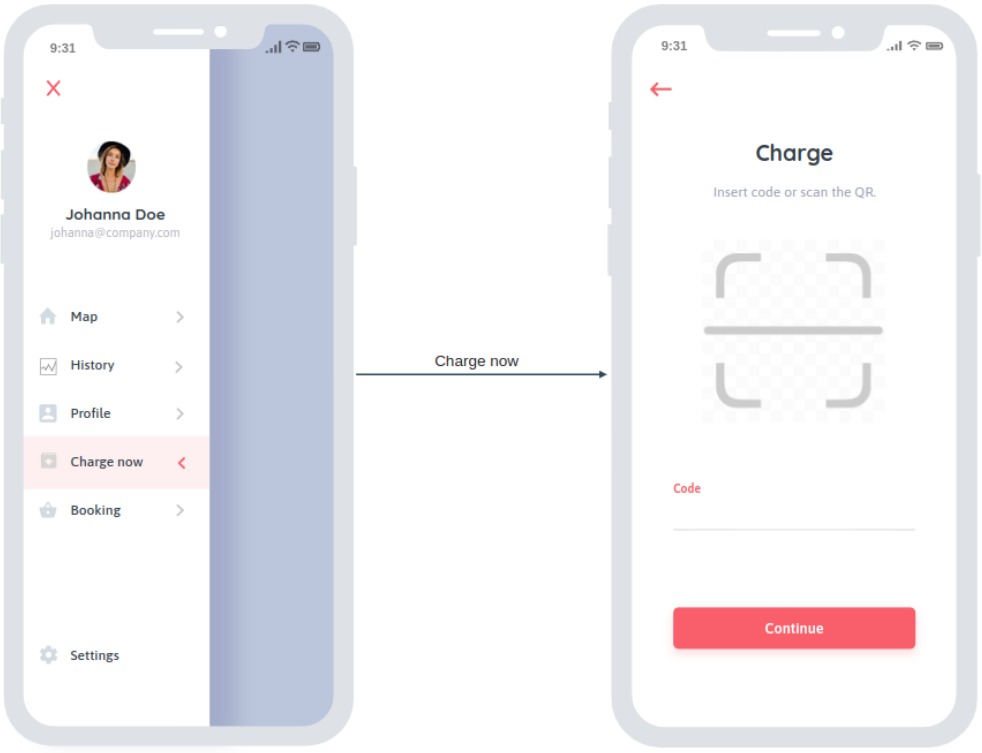

Figure 3.3: UI for starting the charging session

To start the charging process the EVD has to insert the code or scan the QR present on the charging point. This can be started from the menu or from the 'Charge now' button present in the charging station page, as we will see in the following mock-ups.

3.1.4. Visualize stations

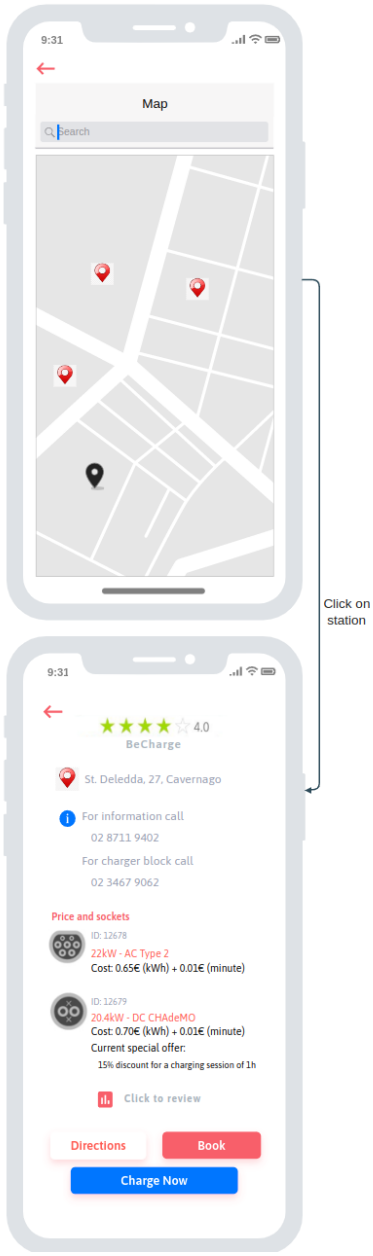

Figure 3.4: UI to visualize stations on the map and their information

The main page of the eMma shows to the user the map and permits to visualize the charging stations nearby or to insert a position in which to search for new stations. Clicking on one of the stations, the application shows the charging station page with the most important information: the rating, the name of the station, the address and the contacts, the available sockets and the respective prices and special offers. Finally, it is possible to leave a review or to see the ones left by other users. From this page is

possible to get the directions to reach the station and it is also possible to start the main operations of the system: the charging session and the booking of a charging point.

3.1.5. Booking charging point

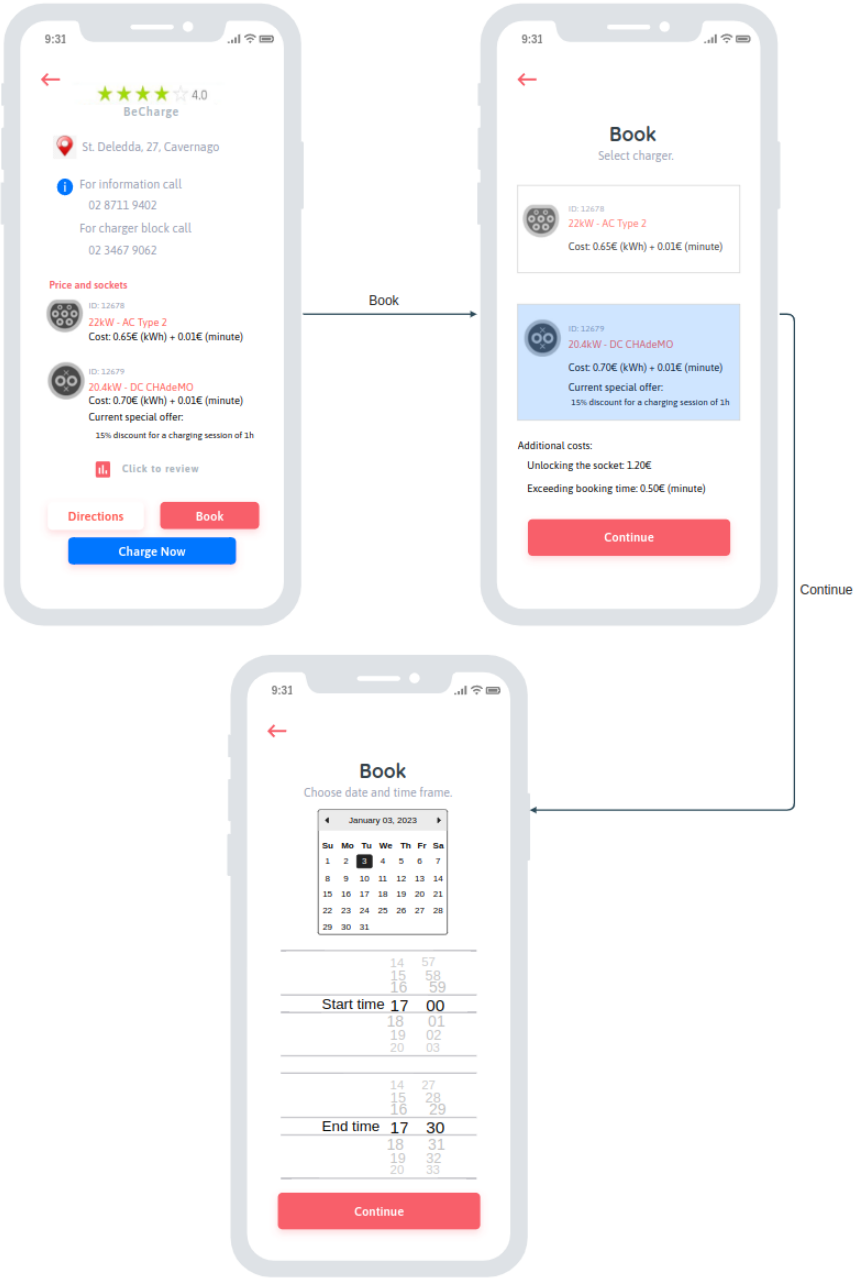

Figure 3.5: UI to book the charging point in a certain time frame

During the booking operation the EVD has to select the charging point he wants to book from the selected station and also to insert the date and the time frame of the booking.

3.1.6. Terminate charging

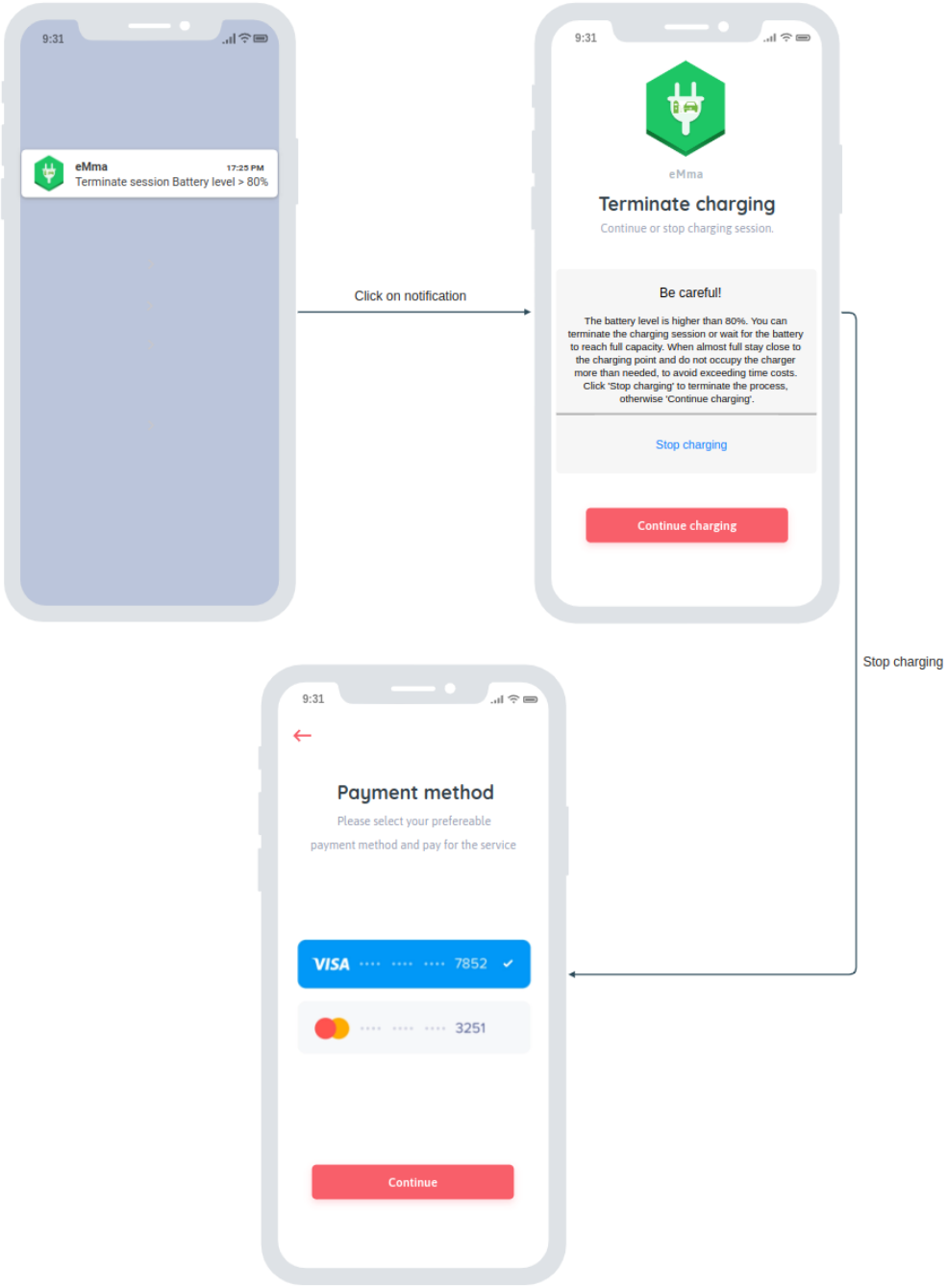

Figure 3.6: UI to terminate the charging session

We can see from this UI, that the eMma sends a notification to the user smartphone when the battery level reaches 80%. The user can decide if he wants to stop the charging session or proceed with the charging until full battery. When the EVD decides to terminate the charging the eMma asks to select the payment method in order to pay for the service.

3.2. CPO user interface

3.2.1. Visualize stations and add a general promotion

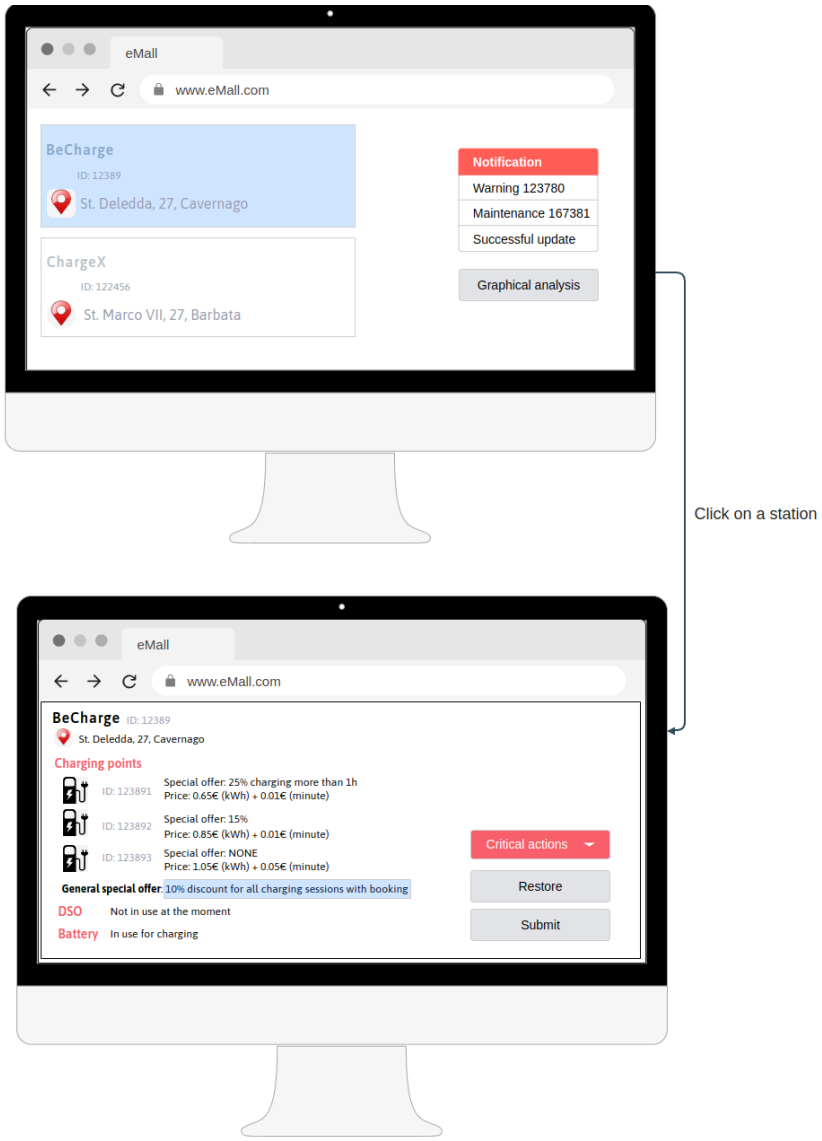

Figure 3.7: UI for visualizing the stations and adding a new promotion

In this minimal example of UI for the CPO we can see how the homepage allows to visualize the stations that the company is managing, and also presents the notifications sent by the system with a different specification and code, showing a precise communication type and the charging station it is related to. From the homepage the CPO can start the graphical analysis of the stations, can solve a notification and he can also select one of the stations to make some changes. In this case the CPO selects a station and the web app

processes the request returning the page with the form related to the specific station. In the form a lot of data can be updated and added to manage the station and in this case the CPO adds a general special offer for the entire station and this will update the prices following the conditions of the promotion. Then the CPO submits the form and waits for a message that informs him of the success of the operation. From the station form is also possible to perform some critical actions, such as deleting the station, deleting a charging point or adding a new charging point.

3.2.2. Update price

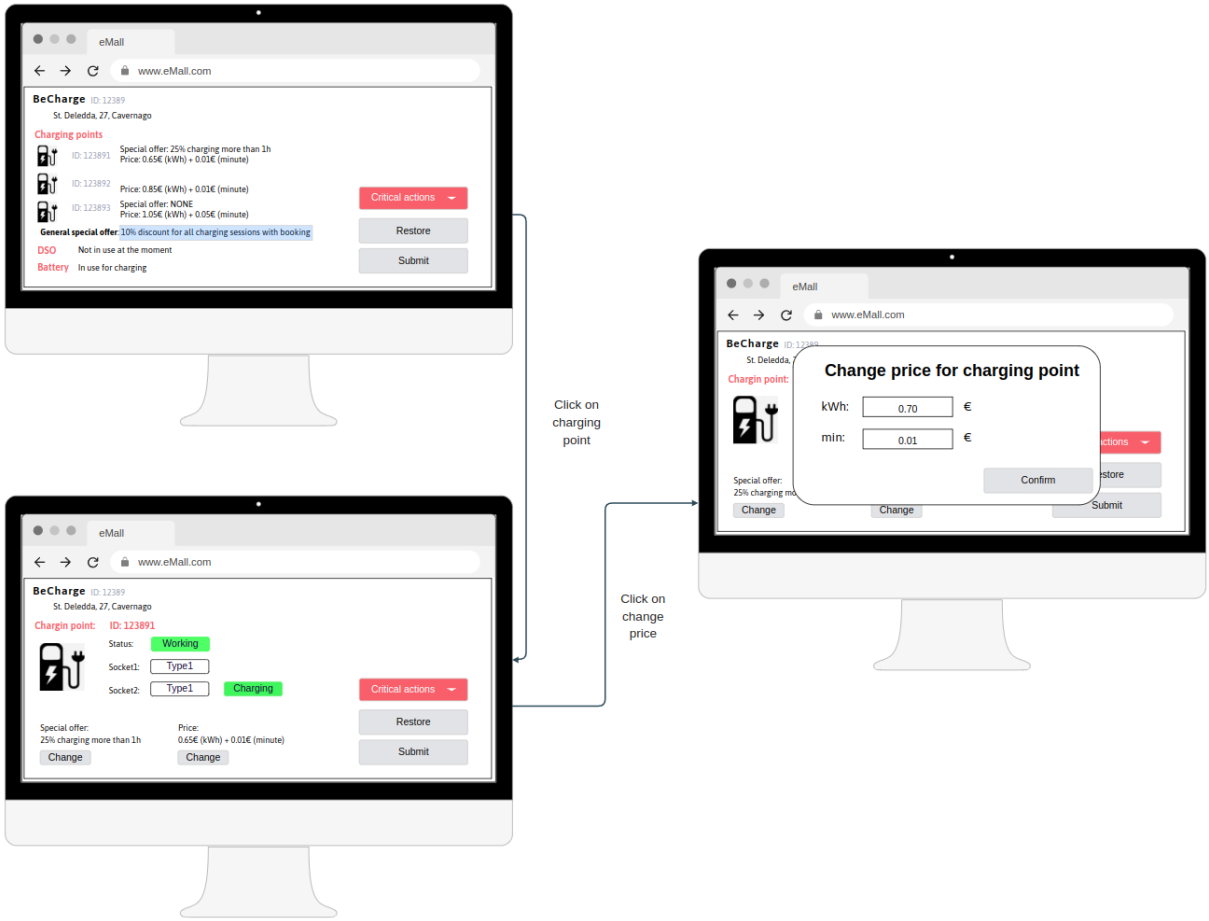

Figure 3.8: UI for updating the price of a charging point

From the homepage of a charging station the CPO can select a charging point to have a better view of its specifics. After clicking on charging point, the webapp loads a new page where the details of the device are shown. Specifically, the page shows the status of the charging point (working, off, not activated, etc.), its sockets with the respective type and

if they are currently used for charging. In the section below the CPO can see the price set for the charging point and any special offer set on it. From this section the CPO, clicks the button to change the price and a popup comes up where he can insert the new price for the charging point. After entering the new price the CPO must confirm his entries by clicking on the Confirm button. Afterwards, the pop-up closes and the page of the charging point is reloaded with the new information. From this page the user can choose whether to confirm the changes, by clicking on the submit button, or undo the changes by clicking the restore button.

3.2.3. Store energy in battery

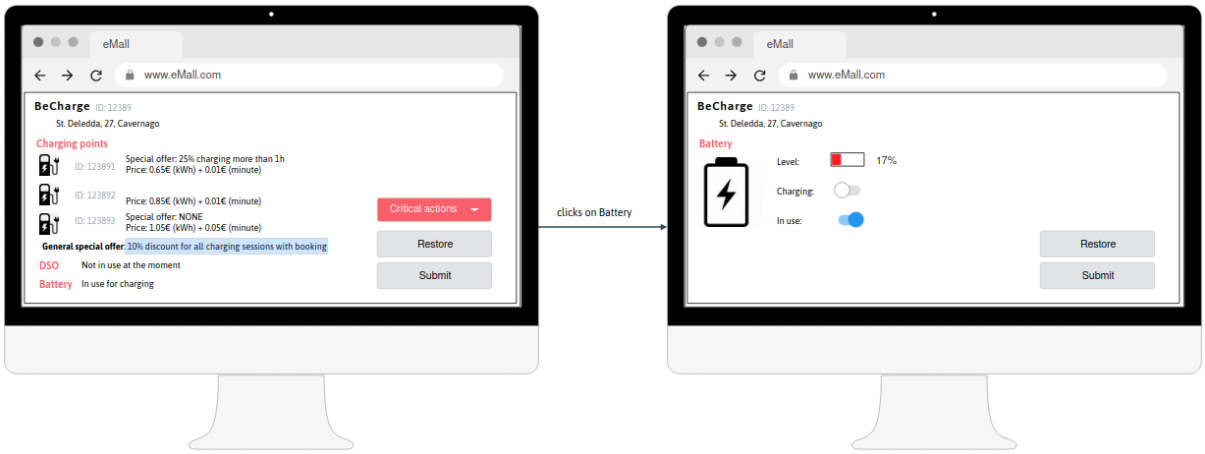

Figure 3.9: UI for storing energy in battery

Same as before, the interaction with the CPO starts from the homepage of a charging station. From here the CPO can select the Battery text (if present), which will load the page for managing the battery of the charging station. This page shows the percentage of charge left in the battery, if the battery is charging and if the battery is currently in use. By clicking on the toggle *Charging* the CPO can connect the battery to the grid and start charging it.

3.2.4. Update DSO

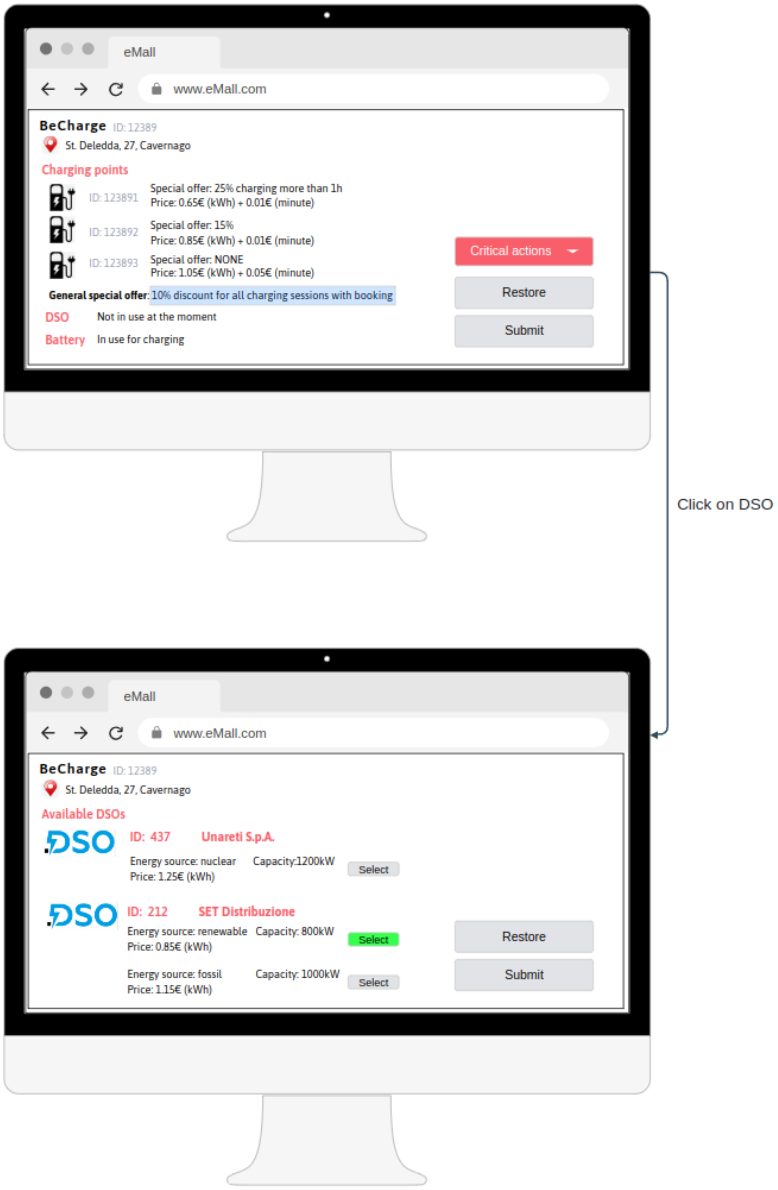

Figure 3.10: UI for updating the DSO of a charging station

Once on the charging station form is possible to select the DSO and update it or add it for the first time. Clicking on the DSO a new form is loaded in which the CPO can choose one from the available DSOs. For each one of them he can see the energy sources with the respective price and capacity. Is enough to select one DSO with the relative energy source and click the 'Submit' button. Then, a request will be sent to the CPMS part of the system, which will communicate through the DSO API with the external chosen DSO to update the contract, and will also save the new information on the DB.

3.2.5. Analyse stations

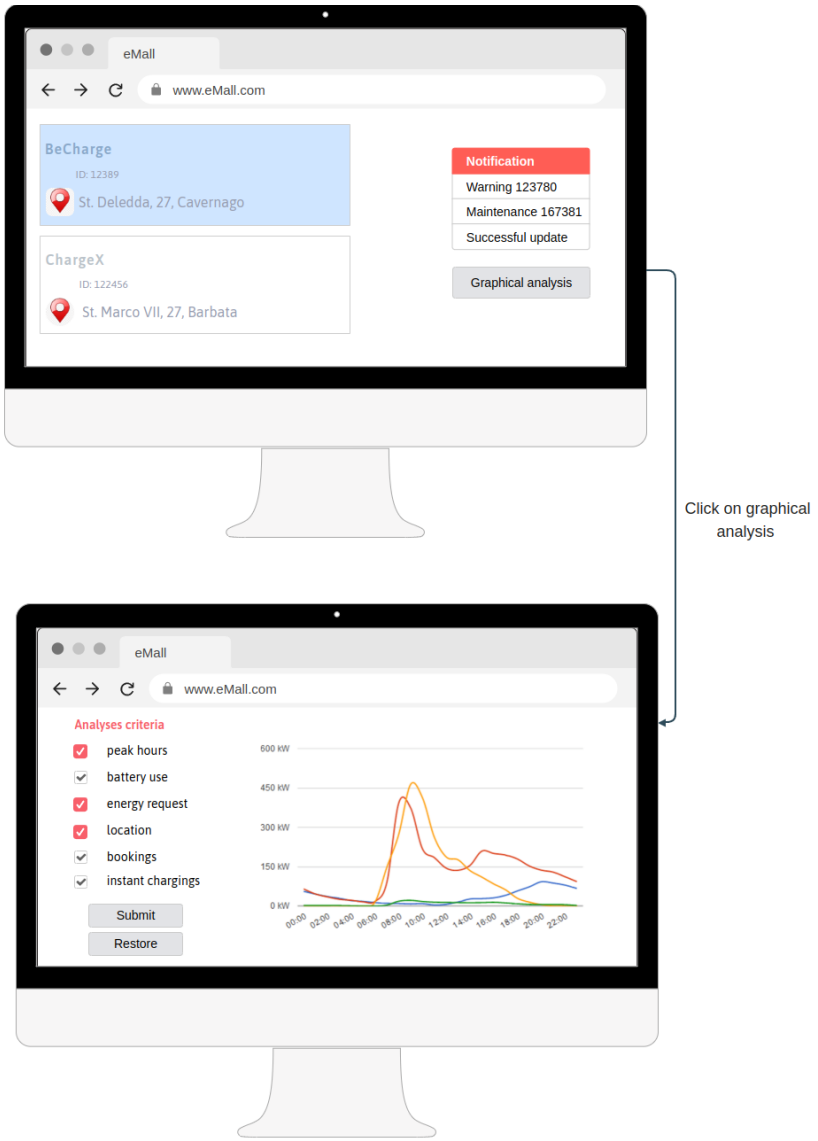

Figure 3.11: UI for analysing the stations

From the homepage, as explained before, the CPO can click the button 'Graphical analysis' and a new page is shown in which he has to select the criteria wanted for the analysis. Once submitted the choice, the eMall takes a few minutes to process the needed data and create a graphical view, that is shown reloading the page. In the reported UI we show an example of a graphical representation of the stations analyzed based on their location and on the energy request in order to identify the peak hours in which the service is used.

4 | Requirements traceability

In this chapter we create a mapping between the components defined before and the requirements defined in the RASD, in order to clarify the utility of the components to achieve the desired functionalities of the system to be.

4.1. Requirements

For the DD to be self-contained we report here the table with the requirements defined in the RASD.

Requirement	**Description**
R1	The system shall allow an unregistered EVD to register an account
R2	The system shall allow a registered EVD to insert data about his EVs
R3	The system shall allow a registered EVD to update EV's details
R4	The system shall allow a registered EVD to add new EVs
R5	The system shall allow a registered EVD to delete an EV
R6	The system shall allow a registered EVD to insert personal data (name, surname, email, payment details)
R7	The system shall allow a registered EVD to update personal data
R8	The system shall allow an EVD to view the charging stations on the map
R9	The system shall allow an EVD to view relevant data about the charging stations
R10	The system shall allow an EVD to view relevant data about a specific charging point
R11	The system shall allow an EVD to view the prices of the charging points of the stations
R12	The system shall allow an EVD to view the special offers of the charging points of the stations
R13	The system shall allow a registered EVD to review a charging station

R14	The system shall allow an EVD to share his location through the GPS module of his mobile phone
R15	The system shall allow an EVD to choose the area in which to visualize the charging stations on the map, if different from the actual GPS location
R16	The system by default must show on the homepage the charging stations on the map territory, centered on the EVD's location
R17	The system shall allow a registered EVD to book a charge from an available charging point in a charging station
R18	The system shall allow a registered EVD to choose the time-frame (date and time) in which to book a charge
R19	The system shall allow a registered EVD to cancel a charging point booking
R20	The system shall allow a registered EVD to visualize his profile data
R21	The system shall allow a registered EVD to view the charging history
R22	The system shall allow a registered EVD to choose a visualization criteria for the charging history
R23	The system shall allow an EVD to start a charging session
R24	The system shall allow a registered EVD to insert the charging point code, in order to start a charging session
R25	The system shall allow an EVD to choose the payment method to use in order to pay for the obtained service
R26	The system must allow the registered EVD to log in
R27	The system must allow the CPO to log in with the company credentials
R28	The system must allow the EVD to access the mobile application, eMma, with or without an account
R29	The system must store the charging history: previous charges and bookings related to the specific EV
R30	The system must collect electric energy data from the DSOs
R31	The system must collect the data about the charging stations from the CPOs
R32	The system must notify the EVD with a specific message, that clarifies the problem, if an error occurs (login error, update error, payment error, ecc.)

R33	The system must notify the EVD with a success message if the operation terminated without errors (successful registration, successful profile modification, ecc.)
R34	The system must ask for EVD's consent to use the location information given by the GPS module
R35	The system must ask the EVD, during registration, to agree to the terms of service
R36	The system must check for the correctness of the data inserted by the EVD (login details, charging point code, ecc.)
R37	The system must show only the free charging points to book
R38	The system must show for each available charging point the relevant information (type of charging, type of socket, charging speed, price, special offers, etc.)
R39	The system must show a summary of the successful booking operation
R40	The system must change the status of the charging point during charging
R41	The system must unlock the charging point if the code is correct
R42	The system shall allow the CPO to view the charging stations
R43	The system shall allow the CPO to view any notification regarding the charging stations
R44	The system shall allow the CPO to update the details of a charging station
R45	The system shall allow the CPO to delete a charging station
R46	The system shall allow the CPO to add a charging station
R47	The system shall allow the CPO to delete a charging point from a charging station
R48	The system shall allow the CPO to add a charging point to a charging station
R49	The system shall allow the CPO to set the price of the charging point
R50	The system shall allow the CPO to set a special offer for the charging point
R51	The system must check the correctness of the data inserted by the CPO
R52	The system must store the data of the charging stations
R53	The system must notify the CPO with a specific message if an error occurs during an operation

R54	The system must notify the CPO with a success message if the operation terminates without errors
R55	The system shall allow the CPO to select a charging station
R56	The system shall allow the CPO to view the DSO's updated prices for the energy sources
R57	The system shall allow the CPO to view the DSO's special offers for the energy sources
R58	The system shall allow the CPO to change the DSO of a charging station
R59	The system shall allow the CPO to choose the DSO's energy source for the charging station
R60	The system shall allow the CPO to set a special offer for the charging station
R61	The system shall allow the CPO to select some criteria to graphically visualize aspects of the charging stations
R62	The system must show a graphical representation of some aspects of the charging stations
R63	The system shall allow the registered EVD to view his upcoming bookings
R64	The system shall allow the CPO to select a notification to solve it
R65	The system shall allow the CPO to decide if for a charging station wants to store energy from the DSO in the battery of the station
R66	The system shall allow the CPO to decide if for a charging station wants to use the battery or acquire energy from the DSO
R67	The system shall allow an EVD to charge the battery until full capacity
R68	The system must communicate with external APIs to complete the payment

Table 4.1: Requirements

4.2. EVD requirements traceability

For the EVD requirements traceability we use the following acronyms for the components of the mapping:

- eMma
- eMci
- ERH - EVDRequestHandler
- ERS - EVDRegistrationService
- CHS - ChargingHistoryService
- PS - PaymentService
- CSS - ChargingSessionService
- CSDM - ChargingStationDataManager
- BS - BookingService
- EPM - EVDProfileManager
- NS - NotificationService
- AS - AuthenticationService
- AL/DB - Database Abstraction Layer/Database Management System

Component from the CPMS[1]

- CMS - CentralManagementService
- RS - ReservationService
- CSIS - ChargingStationInformationService

[1]If an external CPMS is used, then wherever these component are cited in the matrix the external CPMS can be used instead

R	eMma	eMci	ERH	ERS	CHS	PS	CSS	CSDM	BS	EPM	NS	AS	AL/DB	CMS	RS	CSIS
R1	x		x	x								x				
R2	x		x							x						
R3	x		x							x			x			
R4	x		x							x			x			
R5	x		x							x			x			
R6	x		x							x			x			
R7	x		x							x			x			
R8	x		x					x					x			x
R9	x		x					x					x			x
R10	x		x					x					x			x
R11	x		x					x					x			x
R12	x		x					x					x			x
R13	x		x					x					x			
R14	x															
R15	x															
R16	x															
R17	x		x					x	x				x		x	
R18	x		x						x				x		x	
R19	x		x						x				x		x	
R20	x		x							x			x			
R21	x		x		x								x			
R22	x		x		x								x			
R23	x		x				x						x	x		

R	eMma	eMci	ERH	ERS	CHS	PS	CSS	CSDM	BS	EPM	NS	AS	AL/DB	CMS	RS	CSIS
R24	x	x														
R25	x		x			x										
R26	x		x									x				
R28	x															
R29					x		x		x				x			
R31								x					x			x
R32	x			x	x	x	x	x	x	x	x					
R33	x		x	x	x	x	x	x	x	x	x					
R34	x															
R35	x															
R36	x			x		x	x	x	x	x			x			
R37	x		x					x					x			x
R38	x		x					x					x			x
R39	x															
R40							x	x					x			x
R41														x		

Table 4.2: EVD's requirements traceability matrix

EVD requirements traceability explanation

Let us start the description of the traceability matrix for the EVD requirements by pointing out that the component EVDRequestHandler is required every time the user makes a request to the system, so it is present in most of requirements. The same can be said for eMma, since it is responsible for interacting with the user. Hence, in the following paragraphs we will omit describing these component in each requirements they collaborate on satisfying. Moreover we have considered the eMSPRequestHandler since all the request to the CPMS components pass through it, this implies that every time a CPMS component maps to a requirement there is also the eMSPRequestHandler.

R1: The system shall allow an unregistered EVD to register an account

- eMma
- EVDRequestHandler
- EVDRegistrationService
- AuthenticationService

To allow an EVD to register we need eMma for handling the user interaction and view, the EVDRequestHandler, as said before, to receive the requests from eMma and forward them to the right components. The EVDRegistrationService is the main micro-service which handles most of the registration process and uses the AuthenticationService to validate the user credentials and handles them in a secure manner.

The requirements concerned with the user management of his personal data (R2-R7) are all solved by the EVDProfileManager. After the user interacts with eMma and selects the modification he wants to do, the request is forwarded to the eMSP. Here is analyzed by the EVDRequestHandler which assigns it to the EVDProfileManager. Obviously the data must be stored and retrieved from a DB, hence the presence of the DBAL and DBMS.

R2: The system shall allow a registered EVD to insert data about his EVs

- eMma
- EVDRequestHandler
- EVDProfileManager
- Database Abstraction Layer/Database Management System

R3: The system shall allow a registered EVD to update EV's details

- eMma
- EVDRequestHandler
- EVDProfileManager
- Database Abstraction Layer/Database Management System

R4: The system shall allow a registered EVD to add new EVs

- eMma
- EVDRequestHandler
- EVDProfileManager
- Database Abstraction Layer/Database Management System

R5: The system shall allow a registered EVD to delete an EV

- eMma
- EVDRequestHandler
- EVDProfileManager
- Database Abstraction Layer/Database Management System

R6: The system shall allow a registered EVD to insert personal data (name, surname, email, payment details)

- eMma
- EVDRequestHandler
- EVDProfileManager
- Database Abstraction Layer/Database Management System

R7: The system shall allow a registered EVD to update personal data

- eMma
- EVDRequestHandler
- EVDProfileManager
- Database Abstraction Layer/Database Management System

To provide EVDs with information about the charging stations (such as price, promotions, type of charging point) (R8-R13) we envisioned the component ChargingStationDataManager to handle this operation. The component will periodically query information about charging stations from their respective CPMS and store them in the DBMS. When the user wants to know information about charging stations, the ChargingStationManager shall retrieve the appropriate information from the DB and send them back to the user as a reply. From the perspective of the CPMS, the component responsible for granting this information is he ChargingStationInformationService

R8: The system shall allow an EVD to view the charging stations on the map

- eMma
- EVDRequestHandler
- ChargingStationDataManager
- ChargingStationInformationService
- Database Abstraction Layer/Database Management System

R9: The system shall allow an EVD to view relevant data about the charging stations

- eMma
- EVDRequestHandler
- ChargingStationDataManager
- ChargingStationInformationService
- Database Abstraction Layer/Database Management System

R10: The system shall allow an EVD to view relevant data about a specific charging point

- eMma
- EVDRequestHandler
- ChargingStationDataManager
- ChargingStationInformationService
- Database Abstraction Layer/Database Management System

R11: The system shall allow an EVD to view the prices of the charging points of the stations

- eMma
- EVDRequestHandler
- ChargingStationDataManager
- ChargingStationInformationService
- Database Abstraction Layer/Database Management System

R12: The system shall allow an EVD to view the special offers of the charging points of the stations

- eMma
- EVDRequestHandler
- ChargingStationDataManager
- ChargingStationInformationService
- Database Abstraction Layer/Database Management System

R13: The system shall allow a registered EVD to review a charging station

- eMma
- EVDRequestHandler
- ChargingStationDataManager
- Database Abstraction Layer/Database Management System

R14: The system shall allow an EVD to share his location through the GPS module of his mobile phone

- eMma

To allow the user to share his location through GPS module, the eMma component, which is an application running on the user smartphone, will interact with the GPS module on the user smartphone to get the location data.

R15: The system shall allow an EVD to choose the area in which to visualize the charging stations on the map, if different from the actual GPS location

- eMma

This feature will be offered by eMma, the user can select a different area from his actual location and eMma will make the request to the eMSP server accordingly.

R16: The system by default must show on the homepage the charging stations on the map territory, centered on the EVD's location

- eMma

As we already explained in the mapping of the previous requirement, eMma will interact with the GPS module of the EVD's smartphone to provide the user with location based services. If the user doesn't change any settings, eMma will, by default, show the charging station nearby to the EVD's location.

R17: The system shall allow a registered EVD to book a charge from an available charging point in a charging station

- eMma
- EVDRequestHandler
- ChargingStationDataManager
- BookingService
- Database Abstraction Layer/Database Management System
- ReservationService

To book a charging session in a charging point from a specific station, the EVD has to insert the appropriate information in the eMma terminal. The request will then be handled by the BookingService component of the eMSP, which after a preliminary consistency check of the information the user has inserted with the ones the system has saved locally, it will retrieve the information necessary to communicate with the appropriate CPMS and forward it the reservation request. On the CPMS side, the request will be handled by the ReservationService, which after his consistency check will confirm the reservation or deny it.

R18: The system shall allow a registered EVD to choose the time-frame (date and time) in which to book a charge

- eMma
- EVDRequestHandler
- BookingService

- Database Abstraction Layer/Database Management System
- ReservationService

This functionality is granted by the component we previously discussed in the mapping of requirement R18. The user has to just insert the time-frame from the eMma terminal, then the request will be handled by these components.

R19: The system shall allow a registered EVD to cancel a charging point booking

- eMma
- EVDRequestHandler
- BookingService
- Database Abstraction Layer/Database Management System
- ReservationService

The components which collaborate to offer this functionality are still the ones mentioned in the two previous mappings. The flow of interaction is totally similar to the one seen for R17.

R20: The system shall allow a registered EVD to visualize his profile data

- eMma
- EVDRequestHandler
- EVDProfileManager
- Database Abstraction Layer/Database Management System

The component responsible to show the user his profile data is mainly the eMma. If eMma hasn't stored locally the user profile data, it must request them to the eMSP. The request will be handled by the EVDProfileManager which will retrieve the information from the DB.

R21: The system shall allow a registered EVD to view the charging history

- eMma
- EVDRequestHandler

- ChargingHistoryService
- Database Abstraction Layer/Database Management System

This interaction is totally similar to the previous one, except for the fact that for this specific case the component responsible for retrieving and delivering the data in the eMSP is the ChargingHistoryService.

R22: The system shall allow a registered EVD to choose a visualization criteria for the charging history

- eMma
- EVDRequestHandler
- ChargingHistoryService
- Database Abstraction Layer/Database Management System

As this operation involves the manipulation of information about the charging history the component responsible for handling this request is obviously the ChargingHistoryService.

To start a charging session the user needs to fetch the code shown by the eMci terminal on the charging point (R24) and insert it in the eMma mobile application, this code uniquely identifies the charging point. The request is then passed to the ChargingSessionService, which will check the correctness of the code before forwarding the request to the corresponding CPMS. Here, the request is handled by the CentralManagementService, which will unlock the charging point and enable the flow of electricity if the data from the eMSP are correct.

R23: The system shall allow an EVD to start a charging session

- eMma
- EVDRequestHandler
- ChargingSessionService
- Database Abstraction Layer/Database Management System
- CentralManagementService

R24: The system shall allow a registered EVD to insert the charging point code, in order to start a charging session

- eMma
- eMci

R25: The system shall allow an EVD to choose the payment method to use in order to pay for the obtained service

- eMma
- EVDRequestHandler
- PaymentService

The user can select a payment method, from one of those supported by the system, directly from the eMma. The eMma will require the EVD to enter all the necessary details to process the payment request. All the data will then passed to PaymentService microservices which will interact with the specific payment processor indicate by the EVD, through the exposed API to complete the payment process.

R26: The system must allow the registered EVD to log in

- eMma
- EVDRequestHandler
- AuthenticationService

To login into the system the EVD must insert his credentials in eMma. This credentials will then be validated by the AuthenticationService, which will return a token to be used for the request that need authorization.

R28: The system must allow the EVD to access the mobile application, eMma, with or without an account

- eMma

The eMma shall be built in a such way that some of the functionalities can be accessed even without logging in the system.

R29: The system must store the charging history: previous charges and bookings related to the specific EV

- ChargingHistoryService
- ChargingSessionService

- BookingService
- Database Abstraction Layer/Database Management System

To have a charging history of bookings and charging sessions each of the components responsible for handling the charging sessions and bookings, ChargingSessionService and BookingService respectively, shall save the data of the session or booking on the DB. Then the Charging history component will be responsible for processing this data in the right format.

R31: The system must collect the data about the charging stations from the CPOs

- ChargingStationDataManager
- Database Abstraction Layer/Database Management System

The task of collecting the data from the charging stations managed by different CPOs is entrusted to the ChargingStationDataManager. This component periodically queries the CPMS, that knows about the information of the managed charging stations.

Whenever an error occurs during the request of the service the micro-service handling that request must return an error message containing the specifics of the error. Afterwards eMma will be responsible of showing the error message to the user. On the other hand (R33), if the operations terminates successfully, then, eMma shall notify the user about the correct conclusion of the operation. If the component needs to send a notification to user, even if the user is waiting for a reply, it will be used the NotificationService component.

R32: The system must notify the EVD with a specific message, that clarifies the problem, if an error occurs (login error, update error, payment error, ecc.)

- eMma
- EVDRegistrationService
- ChargingHistoryService
- PaymentService
- ChargingSessionService

- ChargingStationDataManager
- BookingService
- EVDProfileManager
- NotificationService

R33: The system must notify the EVD with a success message if the operation terminated without errors (successful registration, successful profile modification, ecc.)

- eMma
- EVDRequestHandler
- EVDRegistrationService
- ChargingHistoryService
- PaymentService
- ChargingSessionService
- ChargingStationDataManager
- BookingService
- EVDProfileManager
- NotificationService

The following requirements (R34-R35) are satisfied by eMma, which must not allow the user to perform any request before giving consent.

R34: The system must ask for EVD's consent to use the location information given by the GPS module

- eMma

R35: The system must ask the EVD, during registration, to agree to the terms of service

- eMma

R36: The system must check for the correctness of the data inserted by the EVD (login details, charging point code, etc.)

- eMma

- EVDRegistrationService
- PaymentService
- ChargingSessionService
- ChargingStationDataManager
- BookingService
- EVDProfileManager

Each component shall check the correctness of the provided data before executing the request.

R37: The system must show only the free charging points to book

- eMma
- EVDRequestHandler
- ChargingStationDataManager
- Database Abstraction Layer/Database Management System

The information about free charging points is managed by the ChargingStationDataManager, which will return to eMma this information in an appropriate data structure. eMma is responsible to show to the user this information.

R38: The system must show for each available charging point the relevant information (type of charging, type of socket, charging speed, price, special offers, etc.)

- eMma
- EVDRequestHandler
- ChargingStationDataManager
- Database Abstraction Layer/Database Management System

As we said before, the component responsible for managing the data about the charging stations and charging points is the ChargingStationDataManager. This component shall save the data about the charging point in the DB, from which it can retrieve in the future and send back to eMma when it requests them.

R39: The system must show a summary of the successful booking operation

- eMma

When eMma receives the booking confirmation from the BookingService, it must show a summary of the booking.

R40: The system must change the status of the charging point during charging

- ChargingSessionService
- ChargingStationDataManager
- ChargingStationInformationService
- Database Abstraction Layer/Database Management System

When the user successfully unlocks the charging point, the ChargingSessionService must communicate this event to the ChargingStationDataManager, which will update it's internal information by setting the corresponding charging point as occupied. The CPMS must also execute this operation so to display to other eMSPs that the charging point is occupied. The component responsible for this in the CPMS is the ChargingStationInformationService.

R41: The system must unlock the charging point if the code is correct

- CentralManagementService

When the CentralManagementService (component of the CPMS) receives a request for unlocking a charging point with a correct code it must abide by the request enabling the flow of electricity for the chosen charging point.

4.3. CPO requirements traceability

For the CPO requirements traceability we use the following acronyms for the components of the mapping:

- ANS - AnalyticsService
- SV - StationsVisualizer
- CSM - ChargingStationManager
- BMS - BatteryManagementService
- DMS - DSOManagementService
- NS - NotificationService
- AS - AuthenticationService
- AL/DB - Database Abstraction Layer/Database Management System

R	AS	ANS	SV	CSM	BMS	DMS	NS	AL/DB
R27	X							X
R30				X		X		X
R42			X					X
R43							X	X
R44			X	X				X
R45			X	X				X
R46			X	X		X		X
R47			X	X				X
R48			X	X				X
R49			X	X				X
R50			X	X				X
R51				X				X
R52				X				X
R53				X			X	
R54				X			X	
R55			X	X				X
R56			X	X		X		X
R57			X	X		X		X
R58			X	X		X		X

R59			X	X		X		X
R60			X	X				X
R61		X						X
R62		X						X
R64			X	X			X	X
R65				X	X			X
R66				X	X			X

Table 4.3: CPO requirements traceability matrix

In the previous table the CPORequestHandler component is missing from the representation, because all the CPO requirements need this component to be full-field. Every request passes through this component, which sends it to the specific service that implements the demanded functionality. This micro-service mainly communicates with the ChargingStationManager, that invokes in order to satisfy the client requests.

CPO's requirement traceability explanation

In the final part of this section we report all the requirements with the respective components in order for the reader to have a more clear view of their relation and we also add a minimal explanation to clarify this connection.As before, the CPORequestHandler will not be listed in the following parts, to avoid redundancy, because it always present. Also the web app is part of all the interactions, being the presentation layer with which the CPO interacts.

R27: The system must allow the CPO to log in with the company credentials

- AuthenticationService
- DBAL and DBMS

To log in the CPO inserts the company credentials (email and password), already present on the CPOsDB, and the web app passes the authentication request to the CPORequestHandler, which communicates with the AuthenticationService and receives a token, if the credentials are valid. The token will be used to interact with the system in order to forward authorized requests.

R42: The system shall allow the CPO to view the charging stations

- StationsVisualizer
- DBAL and DBMS

After the log in, the web app sends a request to visualize the charging stations the CPO has to manage, and the request arrives to the StationsVisualizer component, which interacts with the DB, using the abstraction layer, to retrieve the minimal data about the stations that the web app needs to show on the homepage.

R43: The system shall allow the CPO to view any notification regarding the charging stations

- NotificationService
- DBAL and DBMS

After the log in, there is also a request to the NotificationService, which acquires any notification the system has automatically produced, thus, the CPO can manage to solve possible warnings.

R44: The system shall allow the CPO to update the details of a charging station

- StationsVisualizer
- ChargingStationManager
- DBAL and DBMS

After visualizing the stations, the CPO can select one of them and change some parameters, updating the details through the ChargingStationManager functionalities. In mainly all the interactions with the ChargingStationManager the CPO starts from visualizing the stations on the homepage; and to complete the requests the system has to access the DB and get some data or insert some data, through the use of the DBAL and the DBMS. We will not mention again all these details in the following explanations, since the interaction is always the same.

R55: The system shall allow the CPO to select a charging station

- StationsVisualizer
- ChargingStationManager
- DBAL and DBMS

As already explained before, after the log in, the CPO can visualize the stations and select one of them getting the form of the station with the respective information, in virtue of the interaction with the ChargingStationManager and the persistent data.

From R45 to R50, plus R60, we need almost the same components. To delete or add a charging station or a charging point we need to interact with the ChargingStationManager. Also in adding a charging station is necessary to communicate with the DSOManagementService in order to set the DSO from which to acquire energy, as explained before. The rest of the details that we need to add or update, such as the price and the special offers are handled by the ChargingStationManager itself, which interacts with the DB.

R45: The system shall allow the CPO to delete a charging station

- StationsVisualizer
- ChargingStationManager
- DBAL and DBMS

R46: The system shall allow the CPO to add a charging station

- StationsVisualizer
- ChargingStationManager
- DSOManagementService
- DBAL and DBMS

R47: The system shall allow the CPO to delete a charging point from a charging station

- StationsVisualizer
- ChargingStationManager
- DBAL and DBMS

R48: The system shall allow the CPO to add a charging point to a charging station

- StationsVisualizer
- ChargingStationManager
- DBAL and DBMS

R49: The system shall allow the CPO to set the price of the charging point

- StationsVisualizer
- ChargingStationManager
- DBAL and DBMS

R50: The system shall allow the CPO to set a special offer for the charging point

- StationsVisualizer
- ChargingStationManager
- DBAL and DBMS

R60: The system shall allow the CPO to set a special offer for the charging station

- StationsVisualizer
- ChargingStationManager
- DBAL and DBMS

R51: The system must check the correctness of the data inserted by the CPO

- ChargingStationManager
- DBAL and DBMS

Every time the CPO inserts new data the ChargingStationManager checks their correctness, before invoking other components.

R52: The system must store the data of the charging stations

- ChargingStationManager
- DBAL and DBMS

The ChargingStationManager is the main component that allows to manage the stations and their data, so every time there is an update or any kind of action that changes the stations data, this component interacts with the DBAL to store the new data on the DB or to update the previous stored information.

To notify the user of a successful operation or of an error (R53 and R54) in some situations the ChargingStationManager has to interact with the NotificationService, especially if the notification is spontaneous, deriving from the system that monitors the stations and the energy flow. In other situations, such as sending only a confirmation message, the ChargingStationManager can complete the operation by itself.

R53: The system must notify the CPO with a specific message if an error occurs during an operation

- ChargingStationManager
- NotificationService

R54: The system must notify the CPO with a success message if the operation terminates without errors

- ChargingStationManager
- NotificationService

R64: The system shall allow the CPO to select a notification to solve it

- StationsVisualizer
- ChargingStationManager
- NotificationService
- DBAL and DBMS

Once selected a notification, the NotificationService interacts with the ChargingStationManager and opens the form of the station associated to the warning, thus the CPO can solve the problem.

All the operations which involve data regarding the DSO (R56-R57-R58-R59) are passed from the ChargingStationManager to the DSOManagementService, that is a micro-service focused on the DSO parameters, so it can check more in detail if the CPO's updates are valid and also can show the renewed information of the DSOs. Thus, the CPO can visualize the new prices and special offers of the DSOs and can change from which one to acquire energy and from which energy source, in order to improve the business profit.

R56: The system shall allow the CPO to view the DSO's updated prices for the energy sources

- StationsVisualizer
- ChargingStationManager

- DSOManagementService
- DBAL and DBMS

R57: The system shall allow the CPO to view the DSO's special offers for the energy sources

- StationsVisualizer
- ChargingStationManager
- DSOManagementService
- DBAL and DBMS

R58: The system shall allow the CPO to change the DSO of a charging station

- StationsVisualizer
- ChargingStationManager
- DSOManagementService
- DBAL and DBMS

R59: The system shall allow the CPO to choose the DSO's energy source for the charging station

- StationsVisualizer
- ChargingStationManager
- DSOManagementService
- DBAL and DBMS

R30: The system must collect electric energy data from the DSOs

- ChargingStationManager
- DSOManagementService
- DBAL and DBMS

To collect energy from the DSOs is necessary to send an outside request through the DSO API, and this is a task carried out by the DSOManagementService, which interacts with the ChargingStationManager to save on the DB the data of the used DSO. Thus, the ChargingStationManager has to interact with the persistent data of the stations and keep track of the DSO from which the specific central system is collecting electric energy.

The following two requirements allow to the CPO to analyse the stations, and the microservice assigned to this task is the AnalyticsService. This component interacts with the database in order to create a graphical view of the stations, based on some criteria chosen by the CPO. It acquires the persistent data and processes it, thus, the CPO can track some behaviours and trends in the business operations.

R61: The system shall allow the CPO to select some criteria to graphically visualize aspects of the charging stations

- AnalyticsService
- DBAL and DBMS

R62: The system must show a graphical representation of some aspects of the charging stations

- AnalyticsService
- DBAL and DBMS

Finally, the CPMS server of the eMall also allows to the CPO to manage the battery of the station, if it is present. For this activity is necessary to interact with the BatteryManagementService, which checks if the battery is functional and keeps track of the battery level. Also there is a communication with the CentralManagementService of the charging station, that we didn't show in the CPO requirement traceability, but we mention here to clarify the interaction shown in the sequence diagram. The BatteryManagementService needs to send a message to the CentralManagementService to communicate with the battery of the station and physically start the charging of the battery or switch to battery use. We also report the ChargingStationManager, the DBAL and DBMS in the achievement of these requirements, since the ChargingStationManager is the one that invokes the BatteryManagementService and saves the changes on the DB.

R65: The system shall allow the CPO to decide if for a charging station wants to store energy from the DSO in the battery of the station

- ChargingStationManager
- BatteryManagementService
- CentralManagementService
- DBAL and DBMS

R66: The system shall allow the CPO to decide if for a charging station wants to use the battery or acquire energy from the DSO

- ChargingStationManager
- BatteryManagementService
- CentralManagementService
- DBAL and DBMS

5 | Implementation, integration and test plan

For the implementation of the system we opt for an iterative approach. Accordingly, the system will be released to the customer not in its entirety with all the functionalities, but the first releases will consist only of some parts of the system that can be used only by some category of the users and will have only a subset of functionalities. Given that the system is composed by two main subsystems, which must be interoperable with other subsystems of the same kind, we decided that it would be best to build one of the subsystems first and then deliver this subsystem so part of the stakeholders can start to work with the system without having to wait for all the eMall to be completed. Of course if the development team is sufficiently large, the two subsystems can be build in parallel. We decided that the sub-system that should be built first is the CPMS.

5.1. Implementation, integration and test plan for the CPMS part of the eMall

First release The first release of this subsystem consist on delivering a system with the bare essential to allow different eMSP to start a charging session for their clients and for the CPO to set the price for his services. To build this part of the system we have chosen a bottom up approach, and consequently the testing will be done in this fashion. Firstly, the effort of the whole development team shall be focused on building the DBAL to build it as fast as possible, since it will provide access to the DBMS, which is a fundamental component of the system. Naturally, this component will be thoroughly tested, so a driver to test this component shall be built.
In the next iteration, the following components shall be implemented and unit-tested separately:

- CentralManagementService
- DSOManagementService

- ChargingStationInformationService

After unit testing each of this components, the team responsible for the ChargingStationInformationService shall integrated it with the DBAL, to test if the system is well integrated. Meanwhile for the components CentralManagementService and DSOManagementService we can't do a real integration testing because these components use externals APIs, hence the integration testing for this part will consist in verifying if the interfaces provided by the external components respond as intended.

In the next iteration the following components shall be implemented and tested:

- StationsVisualizer
- ChargingStationManager
- eMSPRequestHandler

We'd like to point out that in this iteration for ChargingStationManger the part which communicates with the BatteryManagementSystem will be excluded from the implementation and testing, since it will be left to be developed in a further release. This is also true for the eMSPRequestHandler, which will exclude the part of the component that interacts with the ReservationService, since it will be implemented in the future. After the unit testing are conducted, we shall proceed to test the integration of the eMSPRequestHandler with the ChargingStationInformationService and the integration of the ChargingStationManager component with the DSOManagementService and CentralManagementService components. StationsVisualizer will be not integrated in this iteration since it is a bottom module.

The next iteration, and the last before the release, will require the implementation of:

- a minimal webapp to allow the user to interact with the system
- CPORequestHandler
- NotificationService

As we did before the CPORequestHandler will be implemented only to use the functionalities of the components, or part of them, built in the previous iterations. Meanwhile, for the webapp the focus will be on building only the bare essentials to grant the CPO to interact with the CPMS. After the components pass the unit testing then we will proceed to integrate this two last components. After that we will proceed to integrate test the webapp, DSORequestHandler with the AuthenticationService, which is component that

we will develop ourselves but we will use an already built external framework. With this last step we have built a functioning system ready to be released, but with only a subset of all functionalities. The system will be whole only with the second release.

Second release The second release aims to integrate the system with the micro-services responsible for providing the remaining functionalities demanded of the CPMS, namely allowing an eMSP to reserve charging sessions for its clients, provide the CPO with a data visualization tools, and creating a component to manage the battery of the charging station if present. All this can be achieved in one iteration and specifically the following components need to be built:

- BatteryManagementService
- extend ChargingStationManager to be able to interact with BatteryManagementService
- AnalyticsService
- extend DSORequestHandler to be able to handle the request relative to the AnalyticsService and BatteryManagementService
- conclude webapp iterface
- ReservationService
- extend eMSPRequestHandler in order to be able to call ReservationService

This task can be achieved by three teams working in parallel; the first team should focus on implementing the BatteryManagementService and AnalyticsService components, while extendended ChargingStationManager and DSORequestHandler to support the newly added components. The second team shall terminate the interface of the webapp and lastly, the third team, will be responsible of implementing the ReservationService and extended the eMSPRequestHandler. As we already did before, each component shall be first unit tested, and only after passing these test the team shall move to test the integration of the newly added component with the ones extended to interact with them. Lastly the team shall perform test the whole system to validate the new additions and modification to the system.

After this new addition the CPMS subsystem has all the functionalities demanded of him, and can fully interact with the eMSPs of different vendors.

5.2. Implementation, integration and test plan for the eMSP part of the eMall

First release We proceed once more with a bottom-up approach and implementing in parallel more than one service. For the eMSP part of the system we first implement the micro-services that provide the functionality of charging the EV and since this operation can be made by both a registered and unregistered EVD we do not have to concern ourselves with the registration module. So, initially, for the charging operation to be functional we implement:

- ChargingSessionService
- ChargingStationDataManager
- PaymentService

After unit testing we test each of this component individually using some drivers, and before integrating them for further testing, we first implement the EVDRequestHandler, the controller that manages the requests of this service. The team can be divided in such way that in parallel all these micro-services are implemented, and then subjected to unit testing. Afterwards, they are integrated and tested using a driver that interacts with the EVDRequestHandler, which passes the data to the other three components, testing the functionality as a whole, so also their interaction to reach the requirement; and moreover we can notice that in this integration we need the DBAL, implemented before, to interact with the DB trough the DBMS and acquire the necessary data.

As second step, we implement:

- EVDRegistrationService
- a minimal part of the eMma necessary for the charging and the registration
- eMci

Following the same approach as before, we first test these new implementations on their own, and then we perform an integration to test the complex behaviour of the components and their interactions. We integrate the new elements with the one tested in the prevoius step and with the shared components of the eMall, which were already implemented and tested when developing the CPMS part of the system: NotificationService and AuthenticationService. If all the tests are passed the system can be deployed, having a first minimal release of the eMma, the eMci and the eMSP application server, allowing

to the EVD to charge the EV, after registering and logging in the eMma or simply using the system as a guest.

Second release The second release aims to provide the remaining main features of the eMma and of the eMSP, in order to allow the EVD to book a charging point and to manage its profile data, adding new information or deleting and updating the ones used during registration. To achieve these requirements is necessary to implement the following micro-services:

- BookingService
- EVDProfileManager
- complete the eMma implementation

These components are implemented in parallel, tested singularly with unit tests and then tested using a driver that simulates the above invocations. The components are also integrated and tested in order to verify that their interaction works properly. After a certain set of test, before releasing the new functionalities, is also performed a system test, integrating these new micro-services with the first release and checking the main sequences of events that the EVD could initiate. Once completed the system testing the eMma is ready to be released with all its main functionalities.

In the previous parts of this document we also talked about another component, the ChargingHistoryService, but this micro-service achieves some requirements that are not essential for the offered services. So based on the time and the resources this component can be developed in a later moment, having another release of the system. The eMall has to be maintained and if possible not only pure testing should be conducted, but also some validation, receiving feedback from the stakeholders in order to update the system to meet their needs as much as possible, satisfying also new requirements and improving the non-functional requirements.

6 | Effort spent

Activity	Time spent
Organization	5h
Introduction	3h
Architectural design	22h
User interface design	8h
Requirements traceability	6h
Implementation, integration and test plan	4h
Revision	4h
Total time spent	52h

Table 6.1: The time Bianca Savoiu has spent working on this project

Activity	Time spent
Organization	5h
Introduction	0.5h
Architectural design	35h
User interface design	1h
Requirements traceability	3h
Implementation, integration and test plan	3.5h
Revision	2h
Total time spent	50h

Table 6.2: The time Fabio Lusha has spent working on this project

7 | References

Fowler, M. (2003). *UML Distilled*. Addison-Wesley.

Larman, C. (2005). *Applying UML and Patterns*. Prentice Hall PTR.

List of Figures

2.1 High level description of the components and their interactions 7
2.2 Component view of the system 9
2.3 eMSP composite structure 11
2.4 CPMS composite structure 14
2.5 Deployment view of the system 16
2.6 EVD registration sequence diagram 20
2.7 EVD Login sequence diagram 21
2.8 Authorized request sequence diagram 22
2.9 Charging point booking sequence diagram 23
2.10 Charging point reservation sequence diagram 24
2.11 Visualize stations sequence diagram 24
2.12 Start Charging session sequence diagram 25
2.13 Unlock charging point sequence diagram 26
2.14 Visualize graphical information sequence diagram 27
2.15 Charge battery sequence diagram 28
2.16 Add promotion to charging station sequence diagram 29
2.17 Update charging station DSO sequence diagram 30

3.1 UI for the log in 45
3.2 UI for registration 46
3.3 UI for starting the charging session 47
3.4 UI to visualize stations on the map and their information 48
3.5 UI to book the charging point in a certain time frame 49
3.6 UI to terminate the charging session 50
3.7 UI for visualizing the stations and adding a new promotion 51
3.8 UI for updating the price of a charging point 52
3.9 UI for storing energy in battery 53
3.10 UI for updating the DSO of a charging station 54
3.11 UI for analysing the stations 55

List of Tables

4.1 Requirements 60
4.2 EVD's requirements traceability matrix 63
4.3 CPO requirements traceability matrix 77

6.1 The time Bianca Savoiu has spent working on this project 91
6.2 The time Fabio Lusha has spent working on this project 91