POLITECNICO
MILANO 1863

eMall - e-Mobility for All

Design **D**ocument

Riccardo Motta
Pierluigi Negro

January 10, 2023

Version 1.1

Contents

Chapter 1

Introduction

Nowadays, electric vehicles are becoming very popular. The need to reduce pollution to save our planet is compelling, and anything we can do to reduce the effects of climate change is moving towards the use of clean energy sources, and everyone can do their part.

Many people are changing their cars to electric ones, and in many cities charging stations are being installed for these vehicles. The main aim of the eMall project is to limit the carbon footprint of our mobility by making easy and efficient charging electric cars, focusing on planning the charge.

The eMall project also focuses on the energy needed to accomplish charging cars, as it's important to manage it most efficiently. This is done to reduce the cost of the primary resources and to make the service even more competitive and cheap. The eMall project tries to simplify also this aspect of the charge chain.

1.1 Purpose

The purpose of this document is to provide a general, complete, high-level view of the eMSP and CPMS systems, and of their interactions.

This document comprehends:

- The architectural design of the two systems, comprehending various views over components and interfaces of the systems, and connections between them and between external systems. All these components are also viewed dynamically through the use of use-case sequence diagrams.
- A rough idea of what the users will experience in the means of user interfaces.
- The mapping between the systems' components and the requirements identified in the *Requirements Analysis and Specification Document*.
- A plan on how the two systems should be implemented, integrated, and tested.

1.2 Scope

eMSP | Simplify the charging process of electric vehicles Charging an electric vehicle can be a challenging task to do. Nowadays, it's not so easy to find charging stations where to plug the vehicle. So, also looking at the future where this kind of station will be more present in the territory, it's fundamental to have a system like this one to carefully plan the charge of the vehicle, in order to minimize the time spent on this task, so that it doesn't have any impact on drivers' lives. Thanks to this system it's possible to search for a charging station nearby, check prices and sales, and, of course, book a charging slot for the vehicle.

CPMS | Optimize the energy usage and supply To be competitive, it's also important to minimize the costs of the whole infrastructure, paying careful attention to the price of the electricity and to the mix of primary sources from which it was derived. On the market, there are possibly many DSOs, each one with different supply capabilities, prices, and energy mixes. Choosing from which source to buy the energy is crucial, especially if the CPO can store it in batteries at the charging station.

1.3 Goals

More formally, here are formalized all the goals that the system should archive.

1.3.1 eMSP goals

This first part focuses on the goals related to the eMPS.

G1	Allow the user to see all the charging stations
The user can see and look for all the available charging stations with all the important information, like the cost of the charge and any special offers.	

G2	Allow the user to book a charge
Every user can reserve a charge if any slot is available. The user can select a charging station, the starting time, and the duration of the charge. Moreover, s/he selects the type of socket to use.	

G3	Allow the user to start the charge
Every user who has booked can start the charging process during the booked period. This will be done automatically by recognizing the connected vehicle once the socket is plugged in.	

G4	Let the system notify the user when the charge is finished
End users are informed every time one of their vehicles charging in one of the charging stations ends the recharging process by sending them a notification.	

G5	Allow the user to pay for the service
Every user can pay for the obtained charging process.	

1.3.2 CPMS goals

This second part, instead, focuses on the goals related to the CPMS.

G6	Let the system have information about charging stations
Know the position and the characteristics of all the various charging stations, like sockets, batteries, available energy, and occupation.	

G7	Allow for cars to charge
Start the charge of a vehicle, monitoring the process.	

G8	Fetch information from DSO and decide which DSO to use
Acquire the price of the electricity from the providers and decide from which to buy, if more options are available.	

G9	Define energy mix
Decide the mix of electricity to provide to the sockets, deciding whether to buy energy, use the one stored in batteries or buy energy to store in batteries.	

G10	Manage special offers
Create new special offers on prices for charging.	

1.4 Definitions, acronyms and abbreviations

1.4.1 Definitions

Word	Definition
eRoaming	The *eRoaming* is an external service that the system can query in order to obtain information about the various connected eMPSs and CPMSs.
JavaScript	It's a programming language often used client-side in the World Wide Web for making web pages interactive.
QUIC	It's a recent transport layer for the Internet Protocol aimed to replace TCP.

1.4.2 Acronyms

Acronym	Description
API	*Application Programming Interface*: is a set of instructions and interfaces that is standardized in order to facilitate the communication between different pieces of software.
CPMS	*Charge Point Management System*: the CPOs management system that supports the process of charging vehicles and acquiring the energy from the various DSOs.
CPO	*Charging Point Operator*: the name of the operators managing the charging points and providing the actual charging service.
CSS	*Cascading Style Sheets*: it's a style sheet language typically used together with HTML web pages which contains the stylistic information of the page.
DBMS	*DataBase Management System*: it's the system that sits between the physical data structures on disk and the user, allowing to create, read, update and delete data.
DSO	*Distribution System Operator*: the name of the operators that provide the actual energy used during the charging process.
eMSP	*e-Mobility Service Provider*: the name of the providers of the service to the end users, acting as intermediaries between the uses and the backing CPOs.
GDPR	*General Data Protection Regulation*: it's "the toughest privacy and security law in the world" (according to the official website) in force in the European Union.
HTML	*HyperText Markup Language*: it's the standard markup language used for documents designed to be displayed in a web browser.
HTTP	*HyperText Transfer Protocol*: it's one of the application layer protocols used in the Internet Protocol suite for distributing content and information through the network. Moreover, HTTPS represents the Secure version of HTTP (with data encryption), while HTTP3 stands for the third main version of the standard.
IP	*Internet Protocol*: it's the main network layer protocol of the Internet.
JSON	*JavaScript Object Notation*: it's an open format for exchanging data in a human-readable way, with pairs of key-value attributes and arrays.
JWT	*JSON Web Token*: it' a standardized method for securely representing claims between two parties.
OS	*Operating System*: it's the piece of software (of which the kernel is part) that manages hardware and software resources, providing utilities to developers.
REST	*REpresentational State Transfer*: it's a software architectural style and describes an interface for the communication between remote entities (this is also a nice way to describe the web).
SEO	*Search Engine Optimization*: it's the process of improving the quality and quantity of website traffic, from search engines to web pages (in this specific case).
SMTP	*Simple Mail Transfer Protocol*: it's the main protocol for transmitting emails.
SQL	*Structured Query Language*: it's a domain-specific languages used for managing data stored in relational database management systems.
TCP	*Transmission Control Protocol*: it's one of the main transportation layer protocols of the Internet Protocol suite, which provides a reliable (under normal conditions) delivery of information on the Internet.

1.4.3 Abbreviations

Abbreviation	Description
eMall	eMall - e-Mobility for All.

1.5 Revision history

Version 1.0 released on January 8, 2023: original release.

Version 1.1 released on January 10, 2023: added eRoaming handshake functions and few utility functions to the eMSP.

1.6 Reference documents

1. **M. Camilli, E. Di Nitto, M. G. Rossi**; *eMall - e-Mobility for All project* (2022)
2. **R. Motta, P. Negro**; *eMall - e-Mobility for All (Requirements Analysis and Specification Document)* (2022)

1.7 Document structure

Chapter 1: Introduction (page 4) This first part provides a general introduction to the project and describes the main purpose and goals it tries to archive, defining the scope of the project and pointing out some definitions, acronyms, and abbreviations used in this document. Moreover, there are all the revisions of the document with all the references used for writing it.

Chapter 2: Architectural Design (page 8) This second part focuses on the real architecture of the project, providing a general overview of the systems, followed by a precise description of all the designed components and their interactions, pointing also out their interfaces.

Chapter 3: User Interface Design (page 34) This third part provides some mockups of the designed user interface for the systems and their relations.

Chapter 4: Requirements Traceability (page 51) This fourth part, instead, focuses on the requirements of the project and their actual implementation into the system. For each requirement, the components that implement it are pointed out.

Chapter 5: Implementation, Integration and Test Plan (page 54) This fifth part provides an overview of the flow of implementation of the various components of the two systems, also providing a plan for integrating the various components and testing them all.

Chapter 6: Conclusions (page 57) This last part contains the last notes about the project and concludes this document, pointing out the effort that each of the authors of this document spent in order to release it.

Chapter 2

Architectural Design

The following section describes the architecture of the eMSP and the CPMS systems. The descriptions start from a high-level point of view, detailing interactions between all systems at play. We will focus majorly on defining the roles of the systems, to obtain a clear representation of the communication between all participants. After this description, the focus will be on the components needed to obtain the functionalities of the applications and then their deployment and runtime utilization. This section also includes a precise definition of the architectural patterns utilized to deploy all identified components and other design decisions taken in the architectural design process.

2.1 Overview

2.1.1 General context

Generally speaking, two main systems are the focus of this project, which are the eMSP and the CPMS. These two interact with each other and also with other actors:

- The *DSO* is used by the CPMS for querying pieces of information regarding, usually, the price of energy.
- The *eRoaming* is used by the eMSPs for discovering all the available CPOs (with their respective CPMSs) that subscribed to the service.
- The *PaymentService* represents a generic payment service used by both systems (the CPMS also uses it for paying the DSOs for the energy).
- The *users* interact with their respective system.

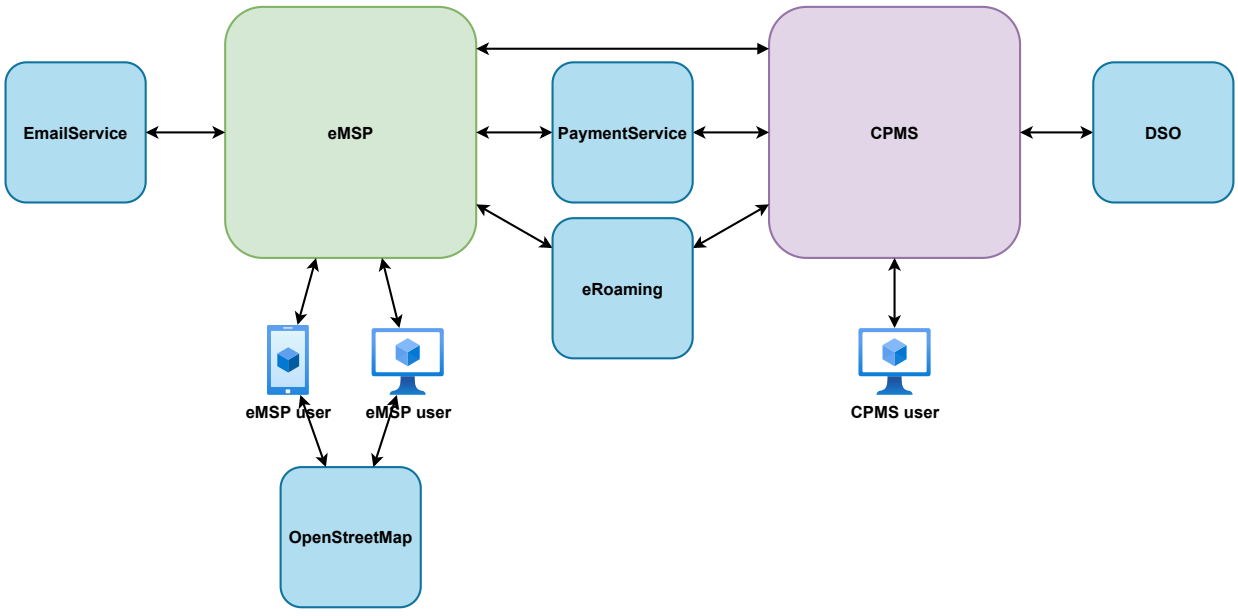

Figure 2.1: general overview of the systems.

2.1.2 eMSP composition diagram

The following diagram presents a more precise description of the eMSP system, in which the most important components of the infrastructure are depicted. These are divided into different groups following the three-tier architecture design pattern, grouping them according to their main purpose. Moreover, the connections of the various components with external entities are pointed out. Please, note that the internal connections between components are not presented here.

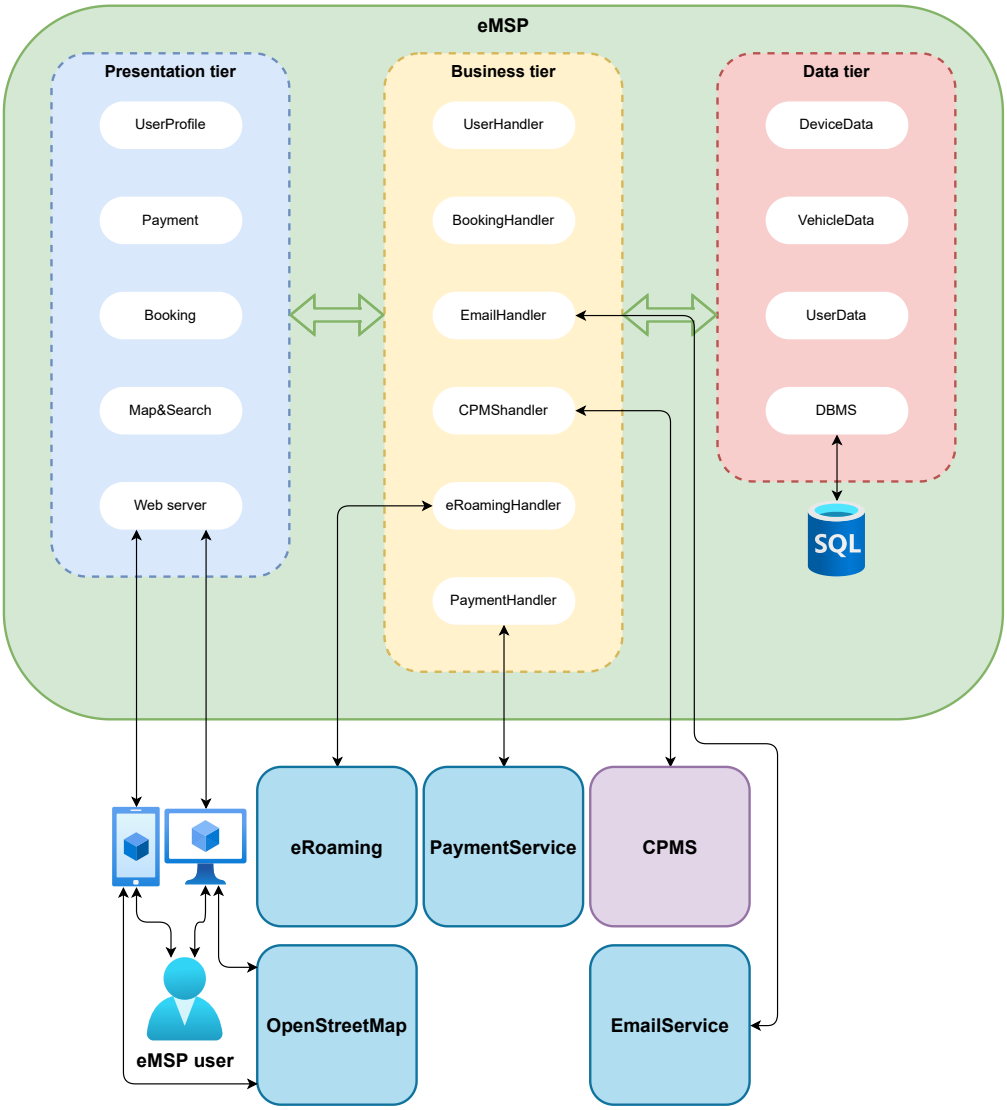

Figure 2.2: general overview of the eMSP.

Presentation tier This tier is the frontend of the system, it's the one that the end user (consumer) uses for interacting with the system. It allows the user to book the charges, look for the stations, pay for the charge...

Business tier This tier is in the middle between the presentation and the data tiers. It manages all the logic of the application but mainly interacts with the CPMS and the other external service providers in order to offer all the required functionalities to the consumer.

Data tier This tier is the one that keeps track of all the user's data. It's in charge of storing, among the others, all the vehicles' certificates, which are some of the most sensible pieces of information in the whole system.

Other components The other components that the system interacts with are the eRoaming service provider, which allows discovery of the various CPOs, the PaymentService for managing transactions, and OpenStreetMap for the map and address translation service.

2.1.3 CPMS composition diagram

A similar diagram to the previous one follows. It presents a more precise description of the CPMS system and its components, following the same three-tier architecture design. The main differences lie in the external components that the CPMS is connected to, as well as the internal main components.

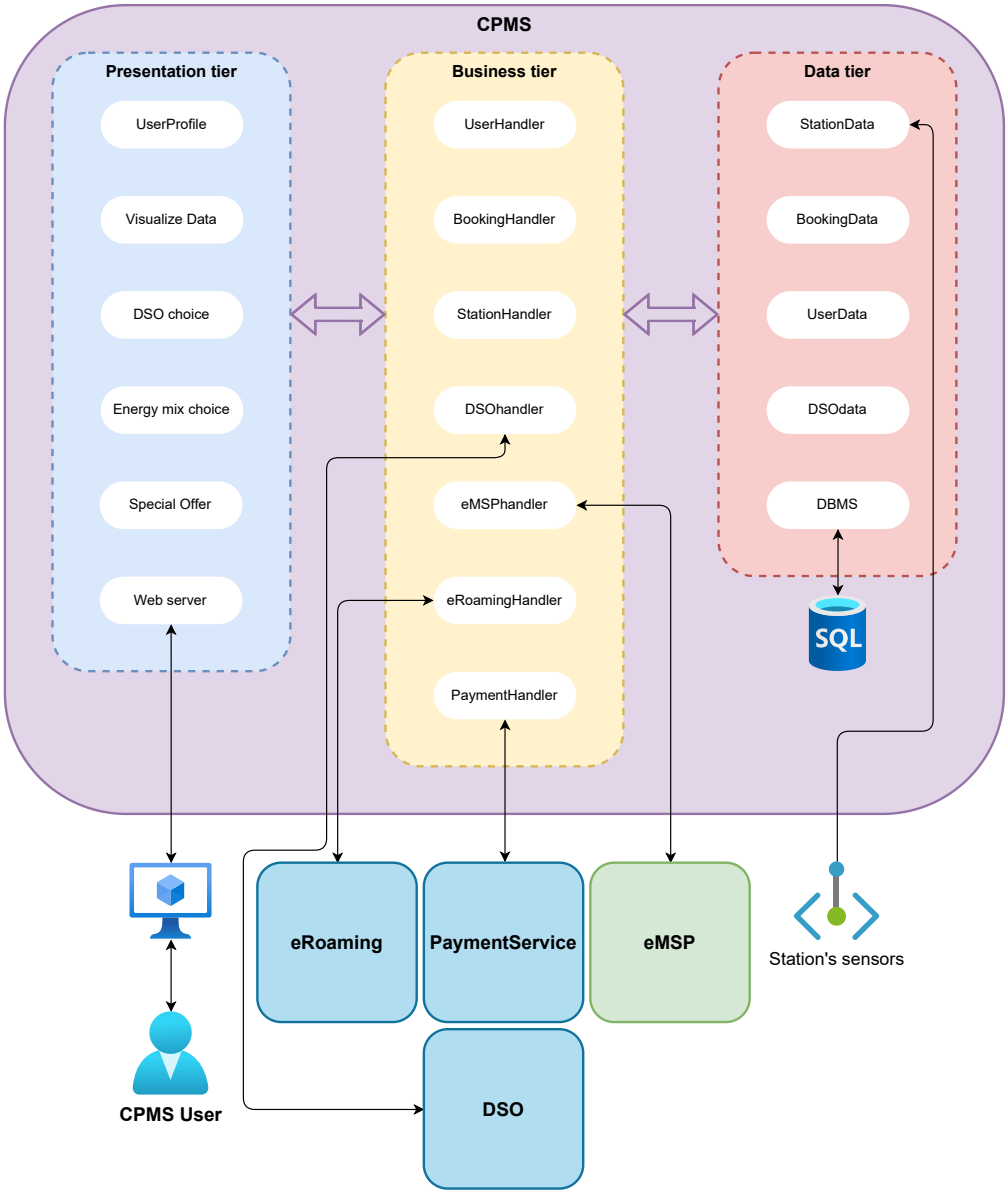

Figure 2.3: general overview of the CPMS.

Presentation tier This tier is the frontend of the system, it's the one that the authorized users use for interacting with the system. It allows visualizing data about a charging station, selecting a DSO, changing the energy mix, and creating a new special offer, as well as actions related to the ones listed.

Business tier This tier is in the middle between the presentation and the data tiers. It manages all the logic of the application but mainly interacts with external service providers to offer all the required functionalities.

Data tier This tier is the one that keeps track of all data used by the system. Particular attention is paid to data provided by the station sensors, and booking information derived from eMSP users.

Other components The other components that the system interacts with are the eRoaming service provider (needed to let eMSPs know about the CPMS), the payment service (to receive payments from end users and to pay the energy from DSOs), the eMSP and the DSOs.

2.2 Component view

This part focuses on the inner architecture of each system, pointing out the connections between all the inner components and the ones with any external component which has to interact with this system. Moreover, all the various interfaces are depicted in this section, even though they are better explained in section 2.4 Components interfaces (page 16) and 2.5 Runtime view (page 18).

For convenience, the two views of the eMSP and the CPMS are divided, to better understand the two very different subsystems of this project, focusing on their duties.

2.2.1 eMSP component view

This component diagram shows the internal structure of the eMSP system and the connections it establishes with the other components. All the internal components are briefly described right after the diagram. Please, note that also the CPMS is treated as an external component since the description of how it internally works is written in the following subsection.

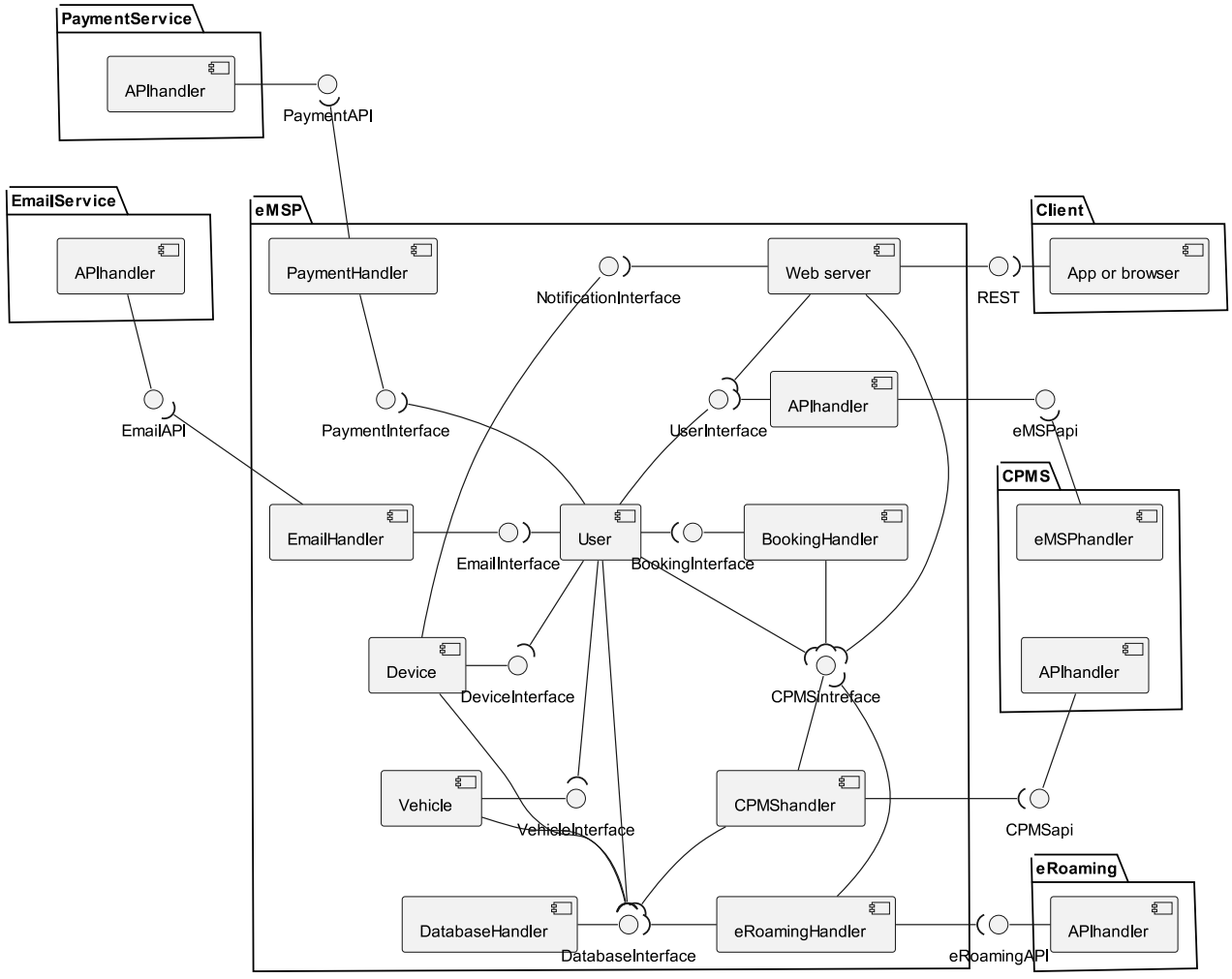

Figure 2.4: connections and interfaces of the eMSP.

APIhandler Manages all the requests coming from any connected CPMS, providing the required data and sending notifications to the user whenever they're needed.

BookingHandler Manages all the bookings. This is the component that, working together with the `CPMShandler`, allows the user to book a charge, edit and delete it. Please, note that this component is not depicted in the class diagram of the *Requirements Analysis and Specification Document*. This is because here, from an implementation point of view, it's better to separate these functionalities from the `User` class.

CPMShandler This is the middleware between the eMPS and the CPMS. It manages all the outgoing requests for the CPMS, like the bookings and any eventual successful payment from the user.

DatabaseHandler The database contains all the data of the system, and together with its DBMS, it provides all the required data to the above modules. The `DatabaseInterface` presented here is just a way to represent the various queries that may be done to the database. As a simplification, in the next sections (2.4 Components interfaces (page 16) and 2.5 Runtime view (page 18)) the `DatabaseInterface` is stated to provide methods for accessing the data. Of course, this is not true in practice since all the queries are written in SQL.

Device This component manages all the information regarding the users' devices. It is also the one that in case the preferred notification method of the user is the notification one, keeps an open channel with that specific device and sends notifications.

eRoamingHandler Manages all the processes of connecting with an eRoaming service, inserting all the information about the found CPMSs (directly taken from the found CPOs) in the database.

PaymentHandler Manages all the financial transactions between the various actors of the system.

User This is the main component with which the user interacts. It is also the one that provides the devices and vehicle data to the other components.

Vehicle This component manages all the information regarding the users' vehicles.

Web server The web server is the frontend of the system for the end user. It statically serves all the page components (it doesn't render anything, it only sends HTML pages, with the required CSS and JavaScript) and replies to the user's HTTP requests through JSON strings.

2.2.2 CPMS component view

APIhandler Manages all the requests coming from any connected eMSP, providing the required data and allowing the eMSP users to book charges.

Booking Handles all the booking of charges, creating and modifying the data stored in the database. It also decides which column to assign to users, communicating the choice by sending a message to the eMSP through the `eMSPhandler`.

ChargingColumn Handles the physical charging of the devices. By communicating with the database it's able to establish whether to start or finish the charge based on the bookings present and the certificate of the connected car. Through its interface, it allows observers to be attached. This can be used to send a notification to the owner of the car when the charge ends, as the column itself cannot communicate to the eMSP's API.

ChargingStation Handles the choice of DSOs and energy mix in the charging station. If "automatic choice" is selected for either "DSO choice" or "EnergyMix", it keeps looping querying useful updates, making decisions, and waiting a fixed amount of time, until the mode is deselected. It adds observers to `EnergySource` and `ChargingColumn` relative to the station, to have quicker access to important information. In particular, it needs to change the energy mix whenever a station battery reaches 100% capacity, and it needs to send a message to the car owner whose car's charge ends.

DatabaseHandler It handles all communication with the database, therefore is used by most components in the CPMS system.

DSOhandler It handles all communication with the DSO's API. It, therefore, is invoked when the DSO choice changes or when new information from a DSO is queried.

eMSPhandler It sends messages to the eMSP by communicating with its API.

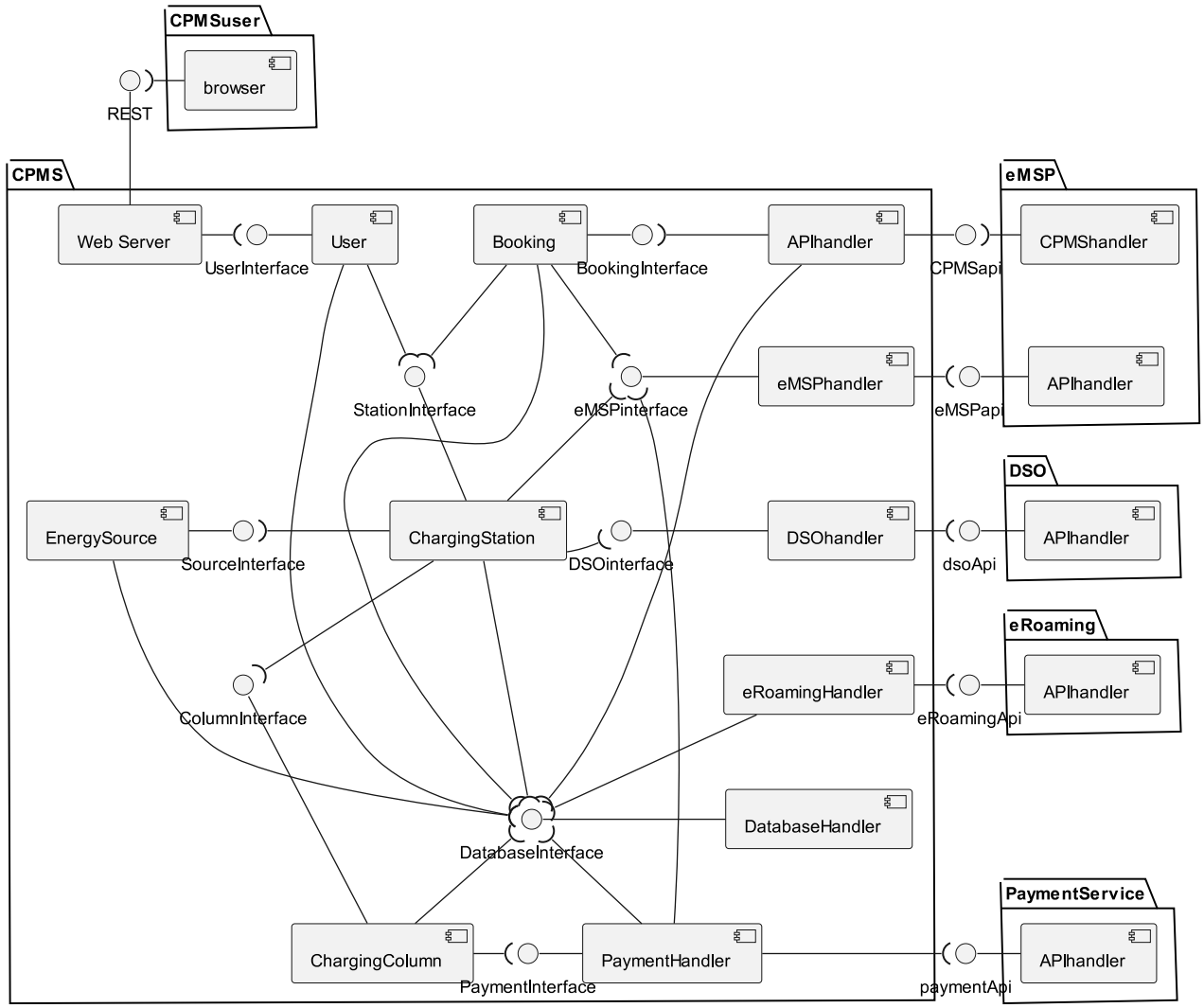

Figure 2.5: connections and interfaces of the CPMS.

Energy Source It represents the various types of sources available in the charging station (batteries, solar panels, and DSO energy). It changes the amount of energy provided by the source. Observers can be attached for various reasons. In batteries, observers can be added to prevent charging over a given threshold or running below a given threshold.

eRoamingHandler It handles connections with the eRoaming.

PaymentHandler It handles connections with the Payment Service, both receiving payments and allowing eMSP users to pay directly at the charging station.

User It handles all actions that the CPMS user can do through the web interface. It's capable to interact with both `ChargingStation` and the database to carry out all actions and to query the required information.

Web Server It's the frontend of the CPMS website. After login, it holds information like the `CPMSuserID` and the session token, used in all internal interactions. It communicates with the user via HTTP requests and responses, sending HTML, CSS, and JavaScript files to create the views, then sending JSON strings to send the data.

2.3 Deployment view

This is the deployment view of the two systems, pointing out the internal architecture of both systems, and the interconnections between themselves and with other external systems (such as the various DSOs and the eRoaming provider).

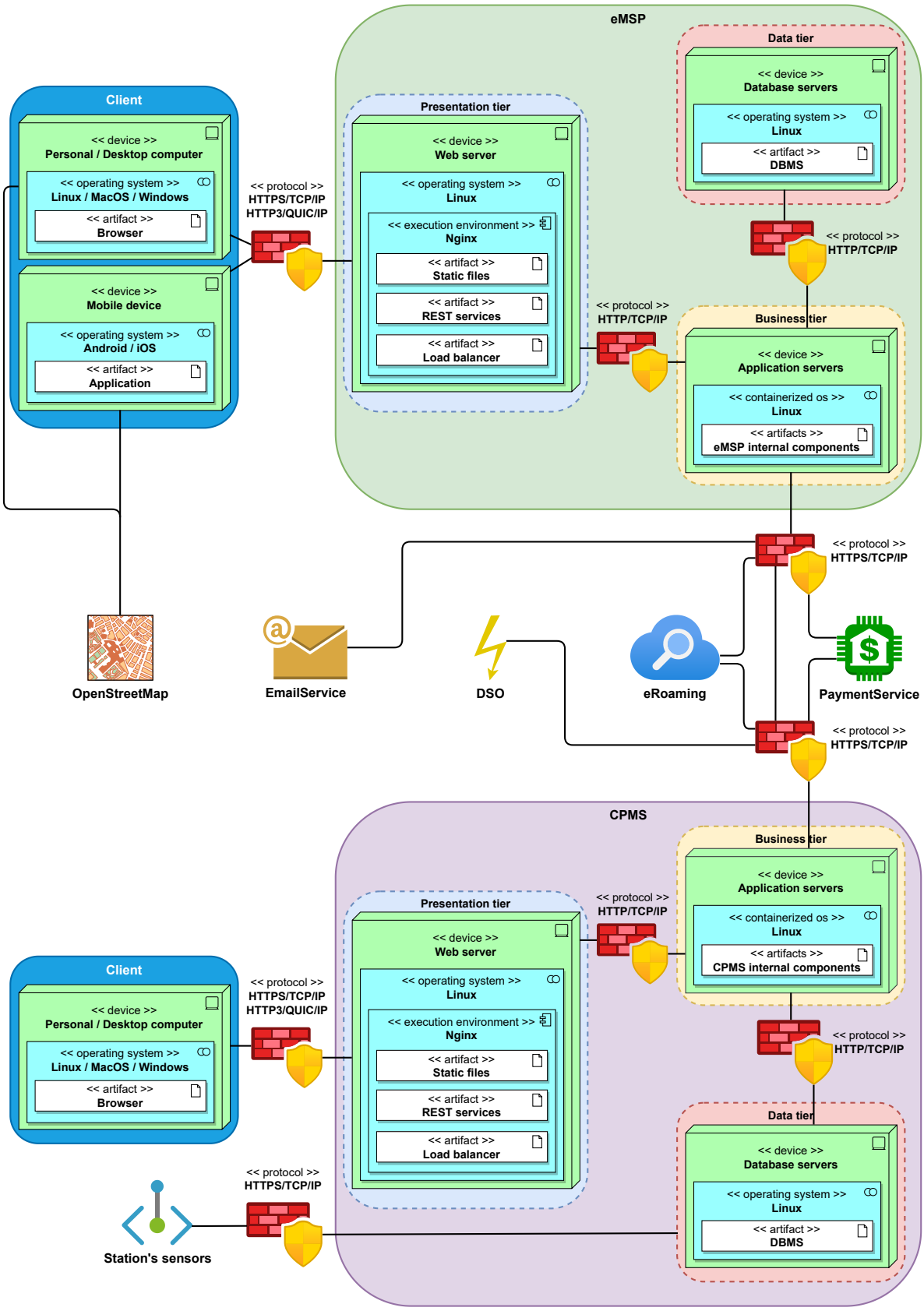

Figure 2.6: deployment view of the systems.

Client Depending on the system, the client can connect to it through a desktop/personal computer or through a mobile device (this one is available only for the eMSP consumer since there exists only for it a mobile application and the website is optimized for being viewed from a mobile device). From them, the clients can interact with the system, being able to do all the actions depicted in the *Requirements Analysis and Specification Document*.

Presentation layer The presentation layer consists of a web server, which acts as a reverse proxy for all the services behind it. In this case, the choice of using NGINX was made because of its event-based nature, and its low memory and CPU footprint. Moreover, it provides a simple configuration for load balancing of the behind components, acting as a reverse proxy (which hides these components from the outside), provides efficient data compression and encryption, which directly provides a higher ranking position in search engines (SEO), and also support live reloads of the configuration in case of updates. NGINX also provides a module for managing JWTs. This is used every time a request is issued from the client in order to verify the authenticity of the incoming requests. If a user doesn't provide a token or the token is invalid, s/he is automatically redirected to the login page.

Business layer The business layer consist of multiple stateless components which interact with each other through the use of sockets, exchanging JSON messages and querying the underlying database. This choice has been taken to provide better scalability through replication.

Data layer The data layer consists of a replicated database with Two-Phase Commit which provides a reliable place where to store all the important data of the customers. This data source is shared between all the components of the above business layer, but it's accessed directly by a single type of component which translates all the requests to SQL and the results back in suitable JSONs.

Firewalls Firewalls are an essential part of the whole design. They provide a way to limit the attack surface of any potential intruder by providing strict access rules. Moreover, there are firewalls between the frontend (presentation layer) and the business layer, and between this last one and the data layer. In order to further enhance security, all those depicted firewalls can be extended with some traffic analyzers which actively or passively react to any potential intrusion.

2.4 Components interfaces

Here are presented the internal interfaces of each subsystem (the arrows denote a "use" relationship). They are used in the sequence diagrams of the following section 2.5 Runtime view (page 18), but some of their calls may be presented in the other system's diagrams. As an example, the eMSP's function `manageNotification (user, message)` taken from the `UserInterface` maps the call of the CPMS's `sendNotification (eMSPuserID, message)` presented in the sequence diagrams.

Attention These diagrams (and also the ones in section 2.5 Runtime view (page 18)) present the interfaces of the components like functions, but, since they communicate through suitable JSONs, they aren't actual functions. It's just a way of modeling more compactly the types of messages and arguments all the various JSONs are built on.

2.4.1 eMSP

This picture depicts the internal interfaces of the eMSP system together with their functions. These allow the user to interact with the system, like booking a charge, and the CPMS to communicate with him/her, like indirectly sending notifications.

Note that all the components don't directly interact with the underlying database, but they query the `DatabaseHandler` through its `DatabaseInterface`, which is the only one to directly access the database.

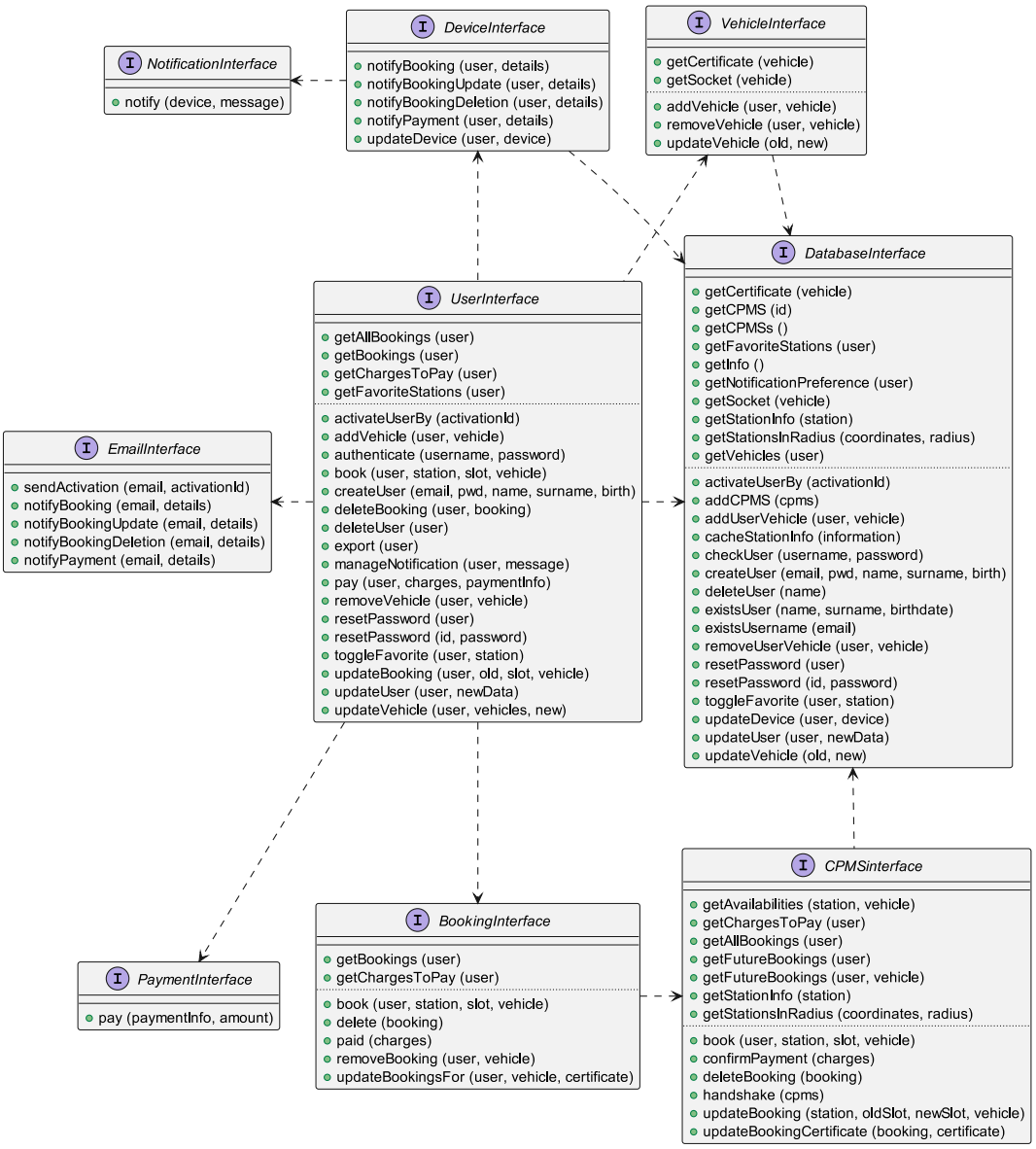

Figure 2.7: internal interfaces of the eMSP system and their functions.

2.4.2 CPMS

Figure 2.8 depicts the internal interfaces of the CPMS system. The functions inside the interfaces cover all functionalities implemented by the CPMS system and/or needed by the eMSP system, and can be improved for future versions of these systems.

Most interfaces in the CPMS systems make use of the `DatabaseInterface`, to write to durable storage all changes happening to the system. This is done to be able to recover quickly in case of failures, in order to guarantee the promised availability. Most functions offered in components' interfaces make use of the same function or a similar function offered by the `DatabaseInterface`, since the `DatabaseHandler` is the only component that directly communicates with the database.

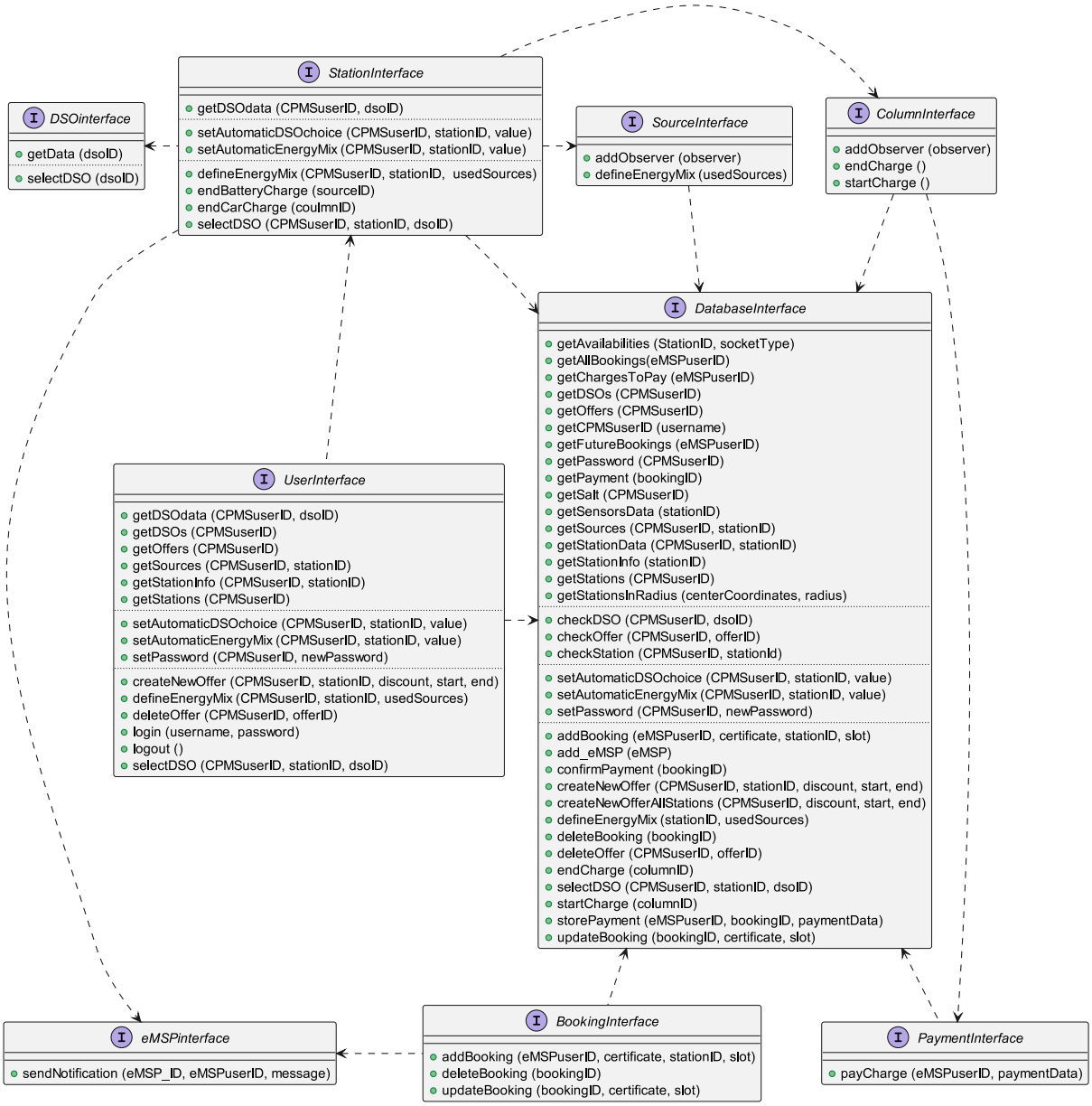

Figure 2.8: internal interfaces of the CPMS system and their functions.

2.5 Runtime view

Here are presented some important actions the user or the system does and how the various components are activated through which function calls in order to perform the required actions.

Every time "consumer" or "user" appears in a UML diagram, it's intended that s/he is interacting with the application or web page that interacts with the system.

For the sake of simplicity, many data checks and the opening of the application or web page, according to the use case, are not illustrated, but they are present in the system. Moreover, some cases are not depicted here:

- *Notification preference*: the user is asked to choose between in-app notifications or email ones after his first login. Furthermore, the user can change this preference at any time under his/her profile.
- *Logout*: simply removes the token from the client.
- *Password change*: it works like most of the password changes implemented nowadays. The first option is that the user clicks on the "Forgot password?" link, receives an email with the link for changing it, and changes the password. The second one is that s/he goes to the user's details page and changes the password directly from there. In both cases, s/he has to log back in on all his/her connected devices.

Also, even if it's not specified in the diagrams, if any component fails, the user is notified and the action is automatically aborted, reverting its state to the original one. The same happens in case of user's network failures (which are "detected" through a timeout).

2.5.1 eMSP

Registration The user registers into the system providing the required data through a dedicated form and his/her email address which is then used for sending the activation link and eventual notifications.

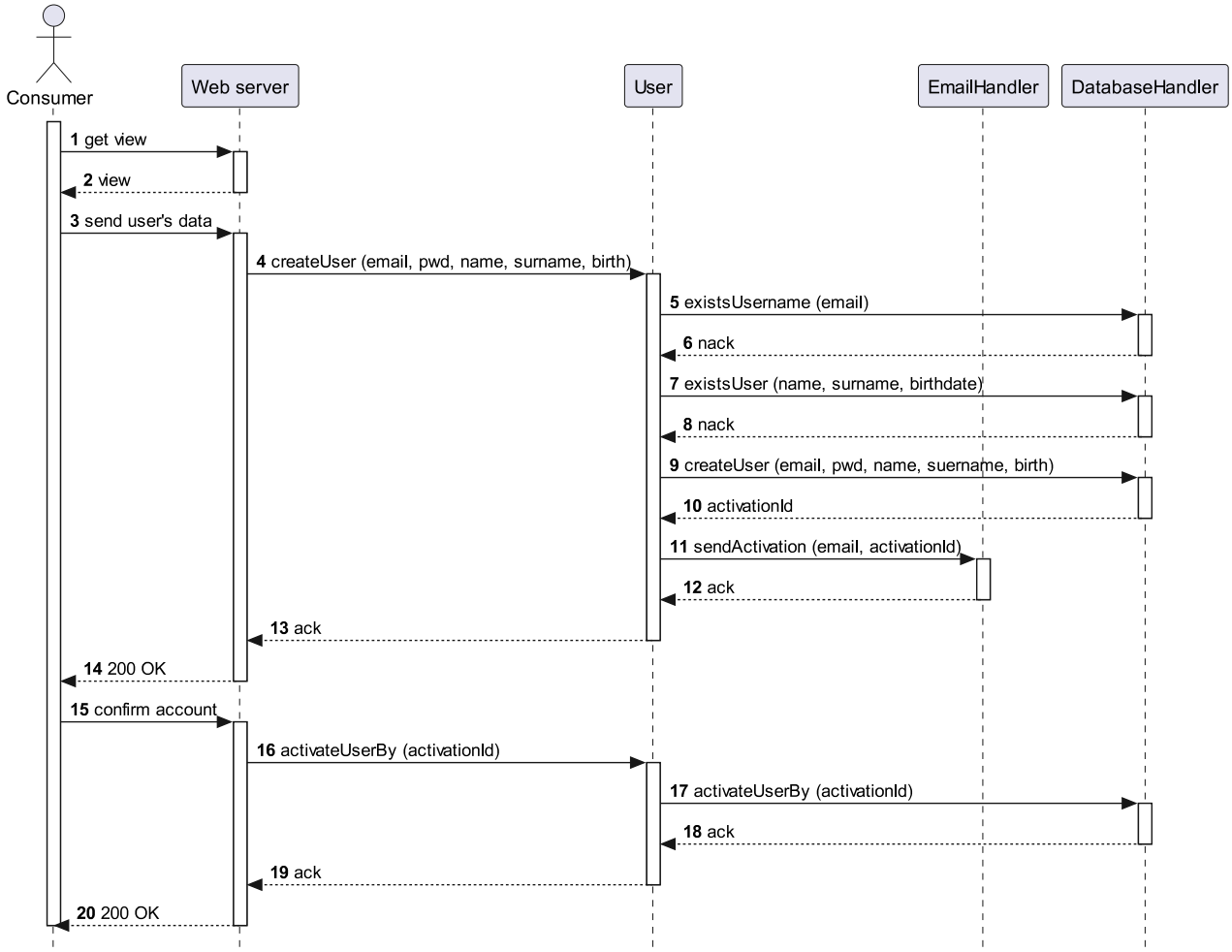

Figure 2.9: registration of the eMSP user (consumer).

Login The user logs into the system, which recognizes him/her and sends back a signed JWT for future actions on the system, like booking charges. This token is properly checked on every request by the web server.

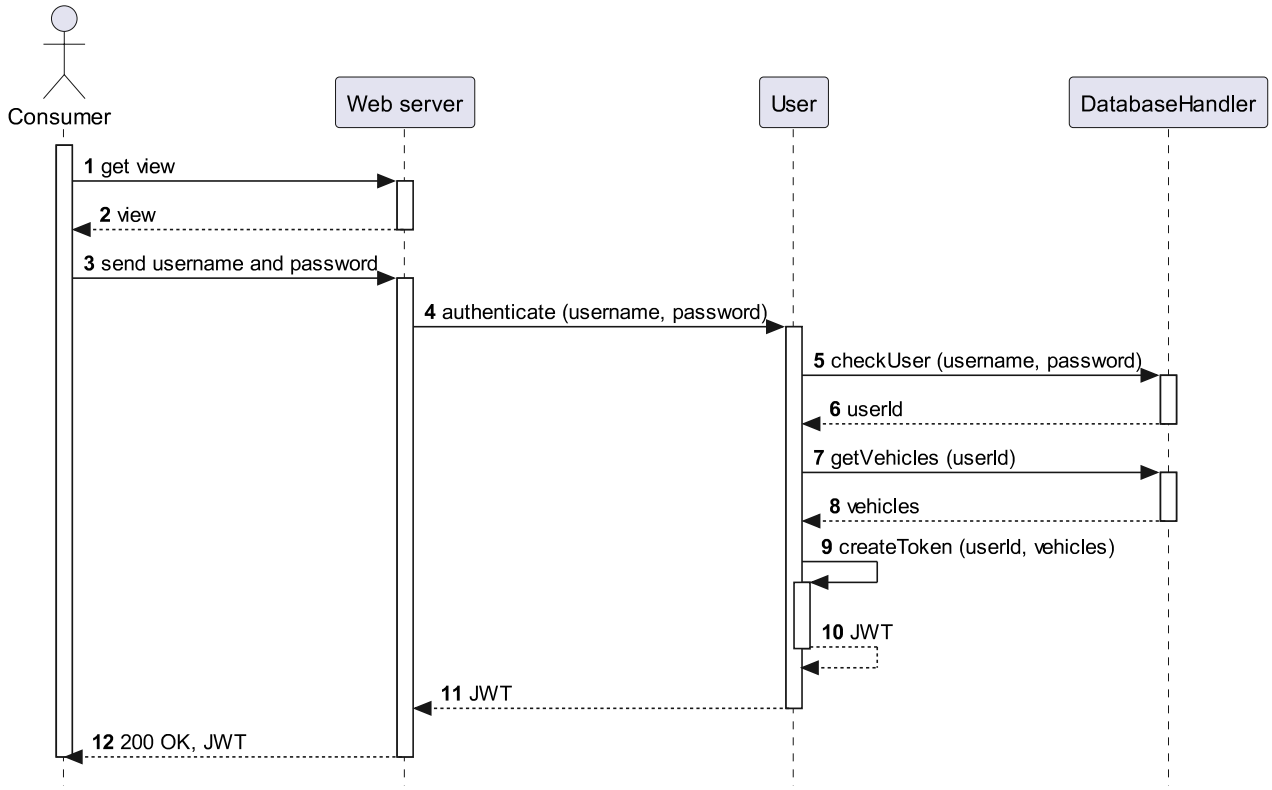

Figure 2.10: login of the eMSP user (consumer).

Mark/unmark a station as favorite The user marks or unmarks any wanted charging station as favorite. The following diagram shows how the stations can be found thanks to this system.

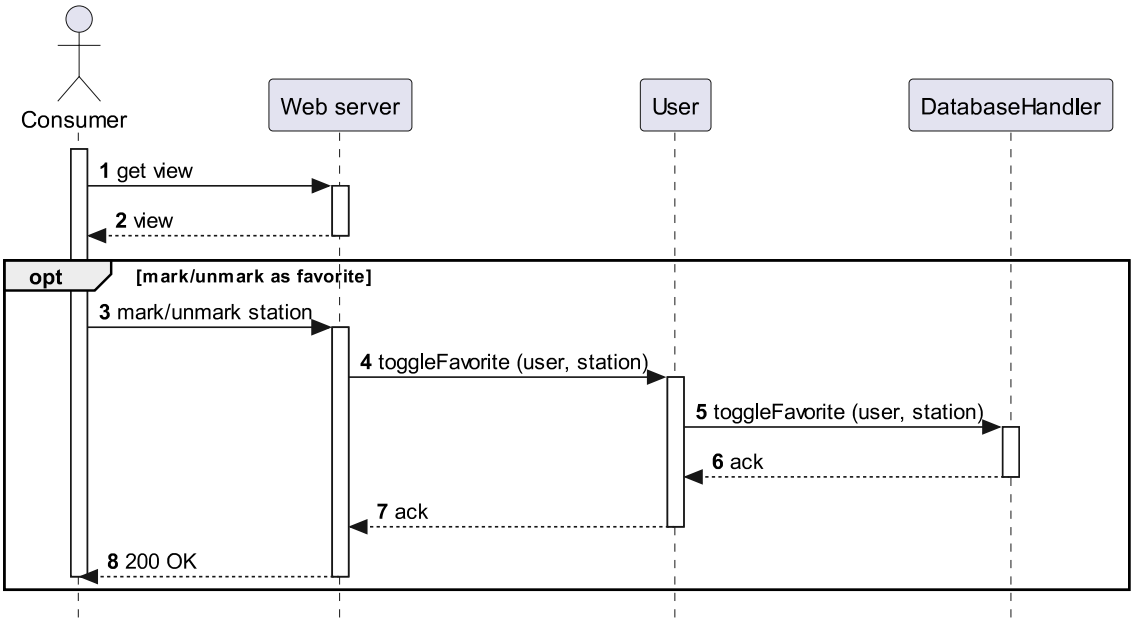

Figure 2.11: the user marks/unmarks a station as "favorite".

Look for nearby stations The user looks for a charging station, either by searching a location or by using a geolocalization service. In any case it's possible to use the map or the list view.

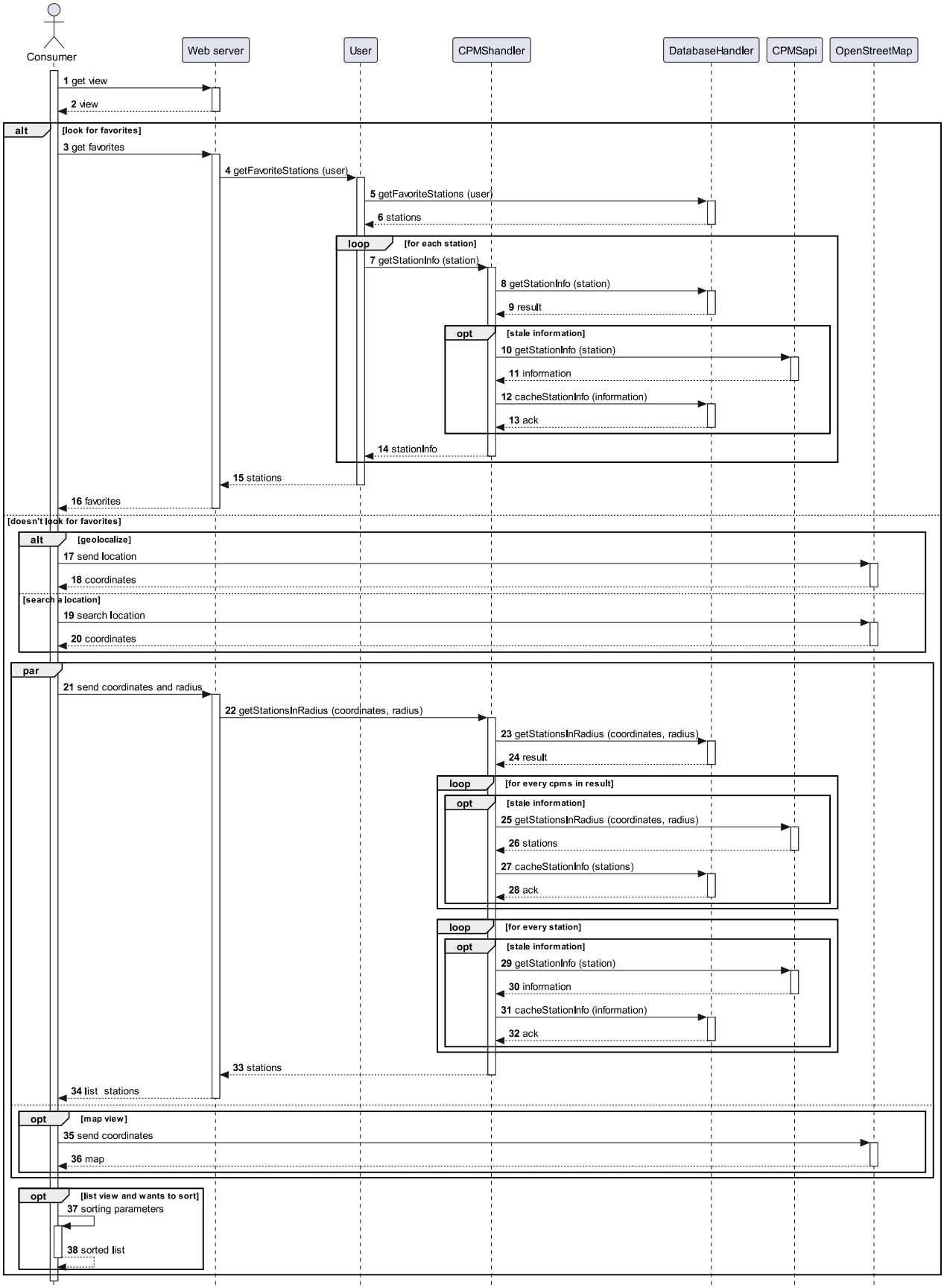

Figure 2.12: the user looks for a charging station.

Book a charge After finding a suitable station (which is depicted in the previous diagram), the user can book a charge by providing the required data to the system.

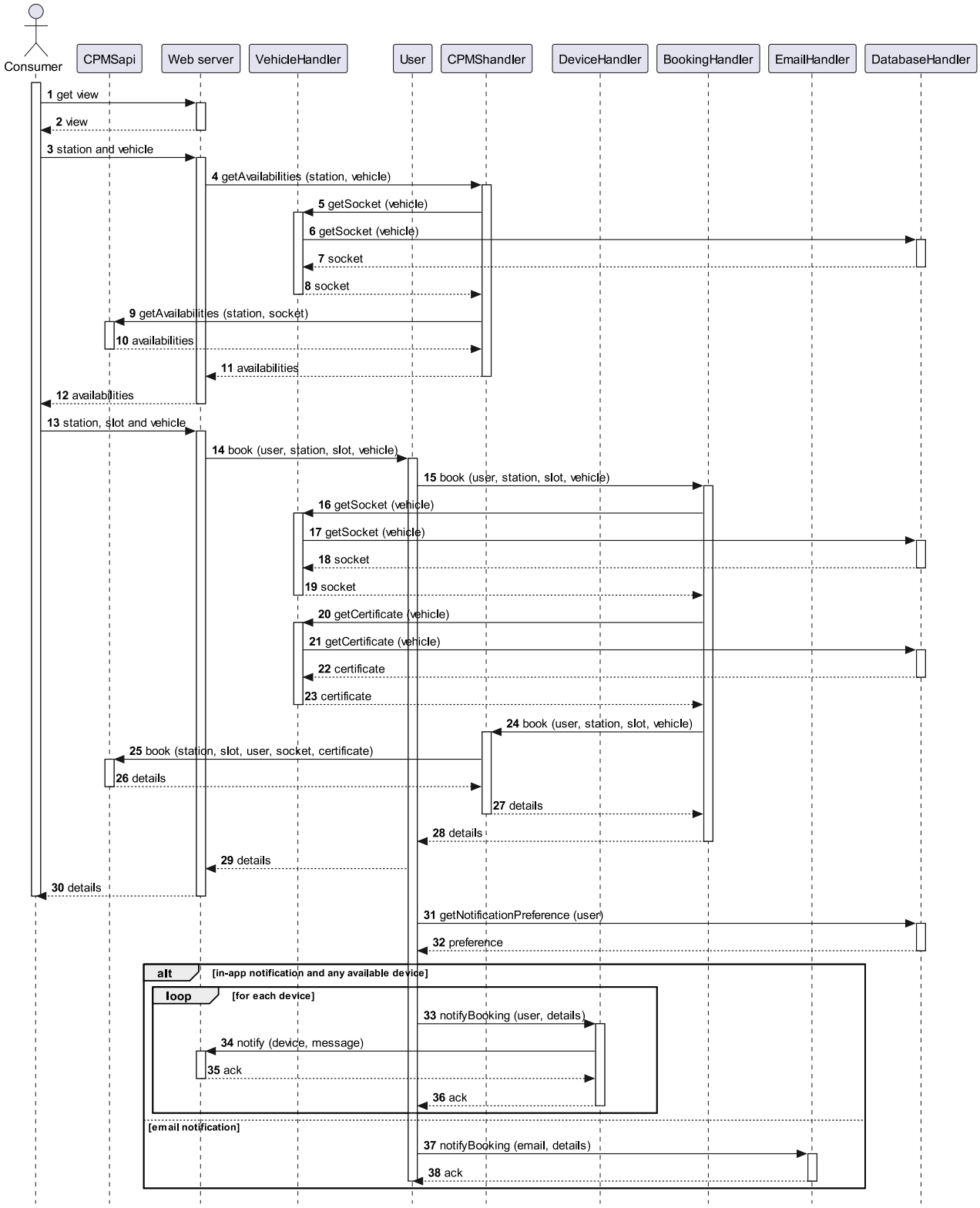

Figure 2.13: the user books a charge.

Edit a reservation If the user needs to, it's possible to modify an already existing booked charge for the same station, providing the required data.

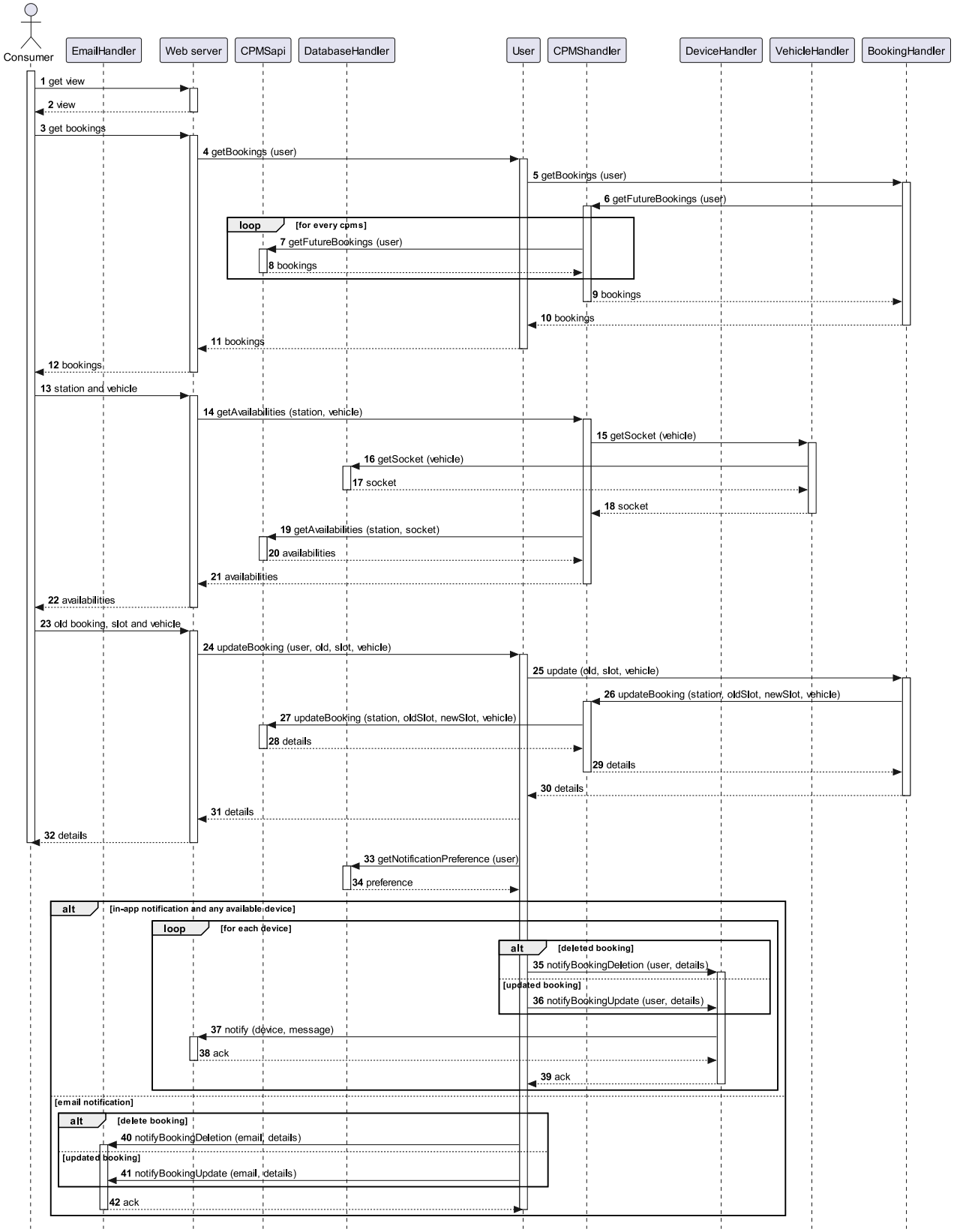

Figure 2.14: the user edits a booked charge.

Delete a reservation If the user needs to, it's possible to delete a previously booked charge unless it's in the past.

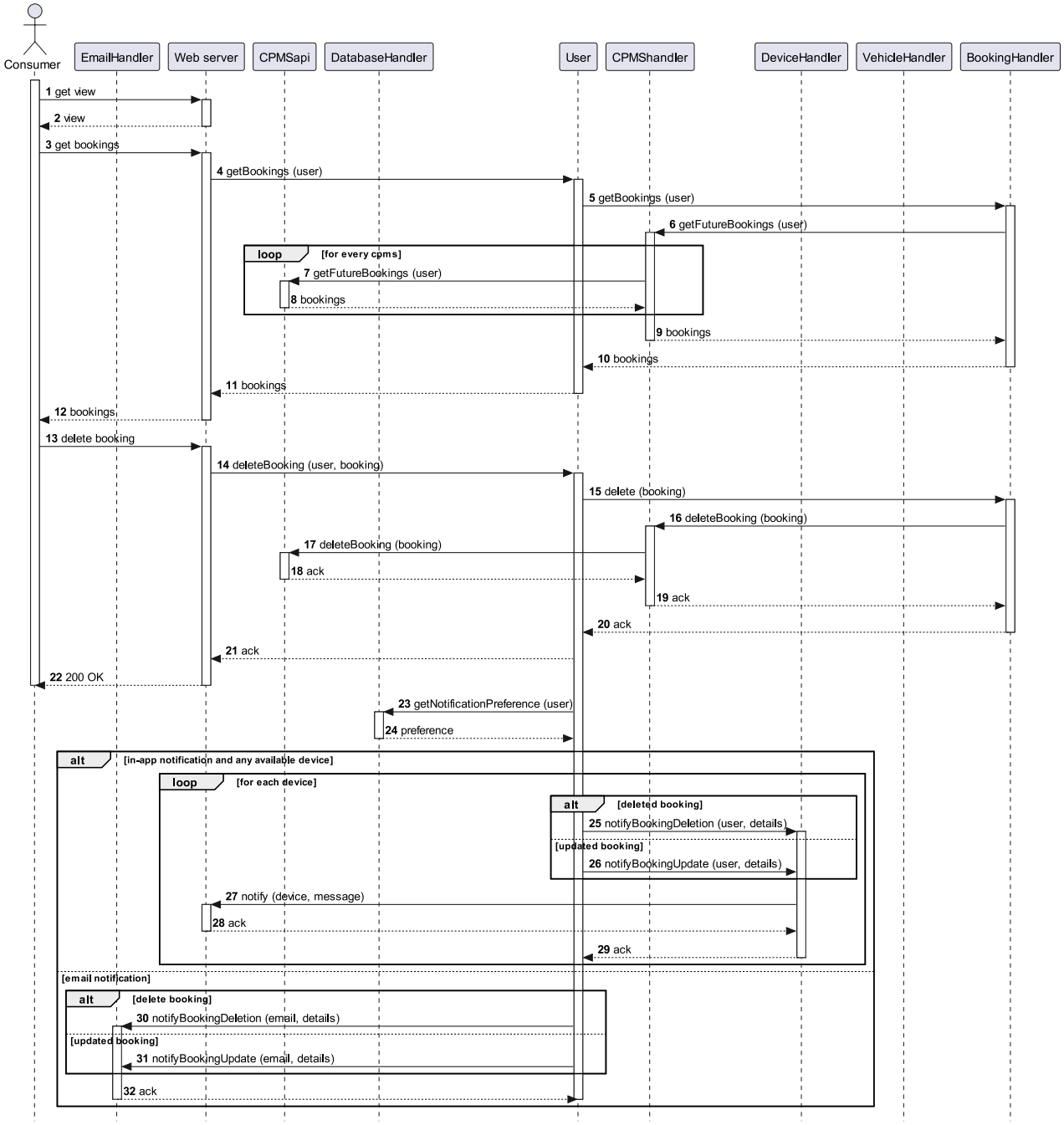

Figure 2.15: the user removes a booked charge.

Pay for the obtained services After a charge is completed, the user can pay the charge. If more than one charge needs to be paid, they are aggregated into one single payment.

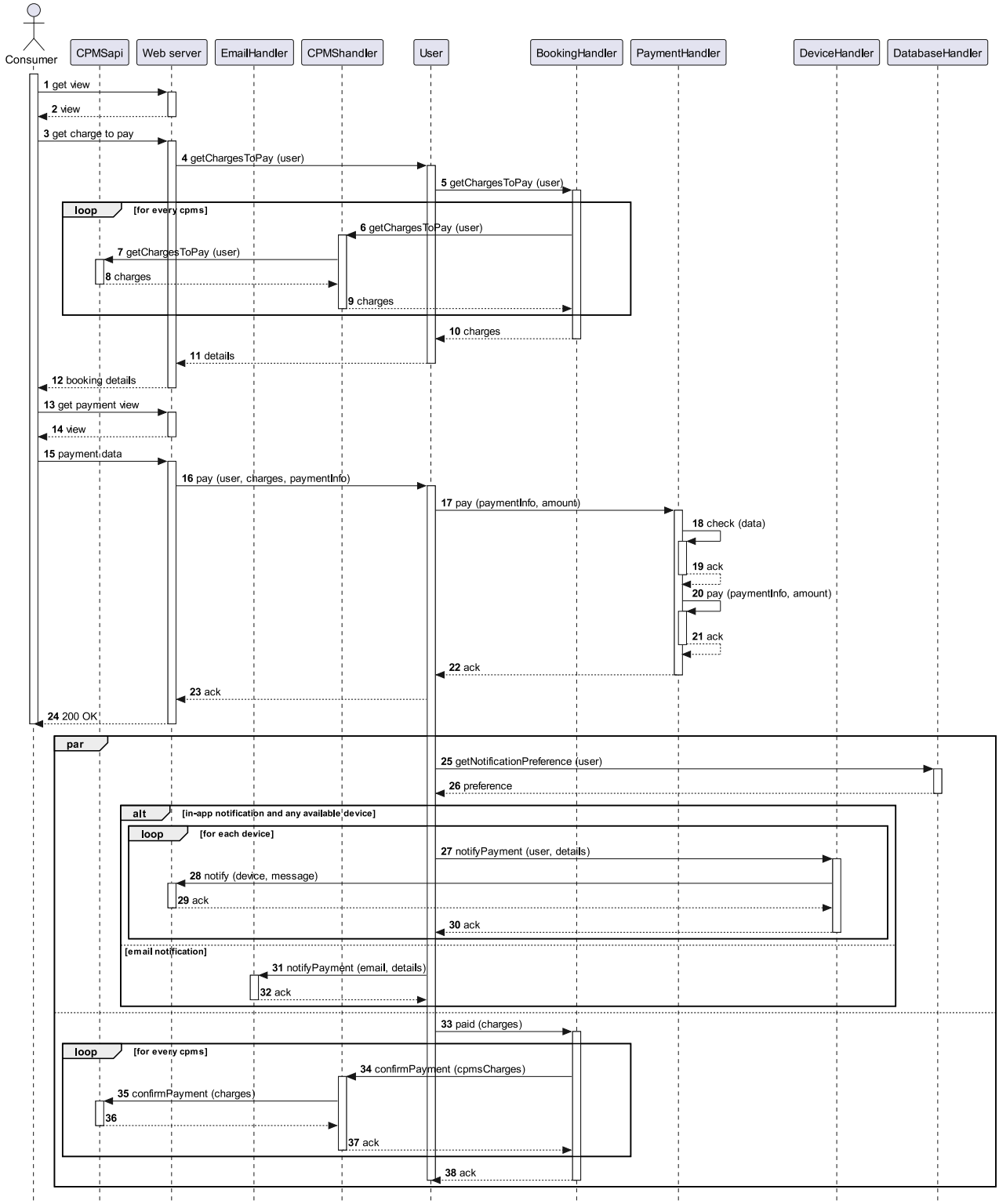

Figure 2.16: the user pays for the charges.

Editing of a vehicle The user can edit at any time his/her vehicles. If also the vehicle's certificate is changed, the system automatically updates all future reservations by telling the CPMSs which is the new certificate for the charge.

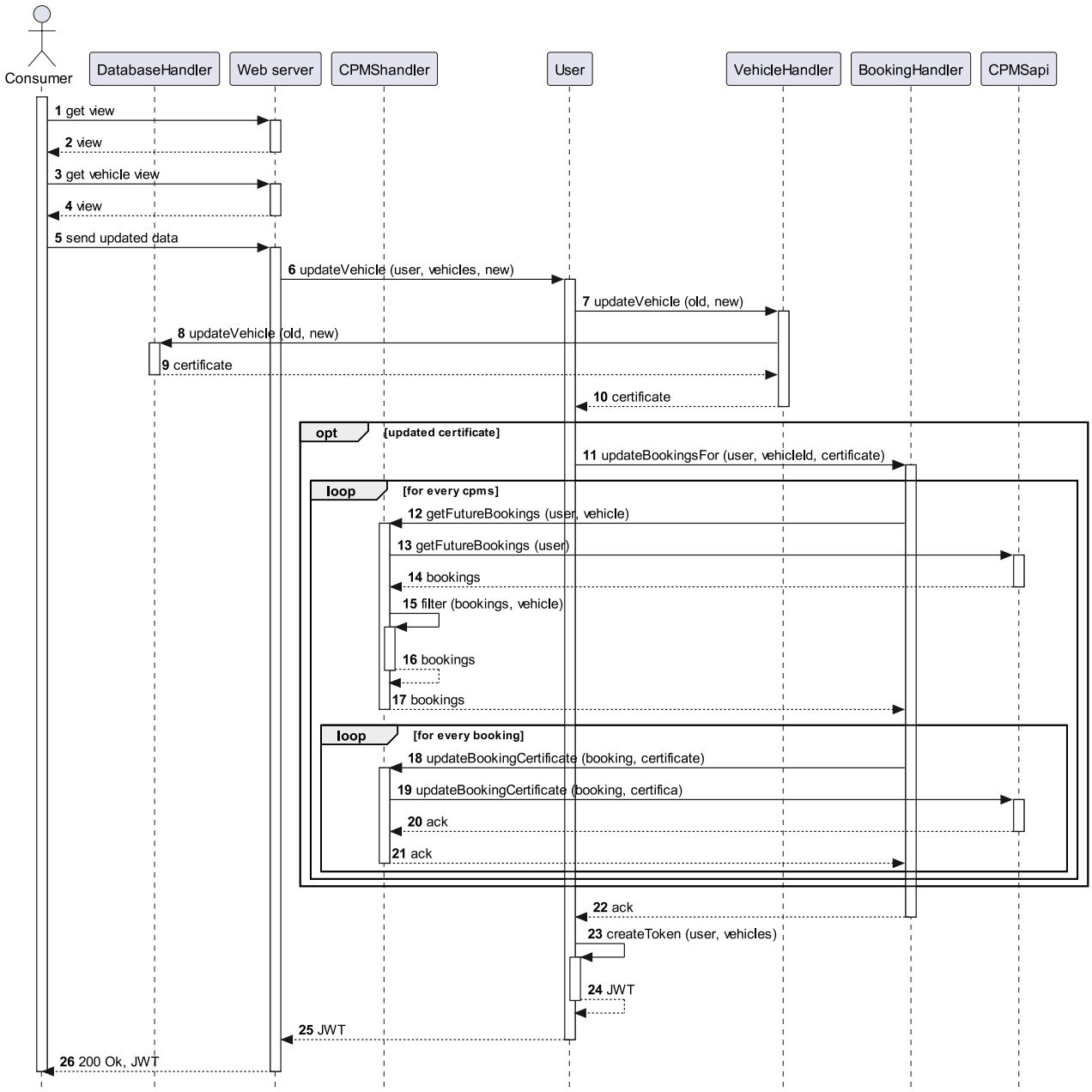

Figure 2.17: the user edits one of his/her vehicles.

Deletion of a vehicle The user can delete at any time his/her vehicles. Every time a vehicle is removed, the system automatically deletes all future reservations for that vehicle on every CPMS.

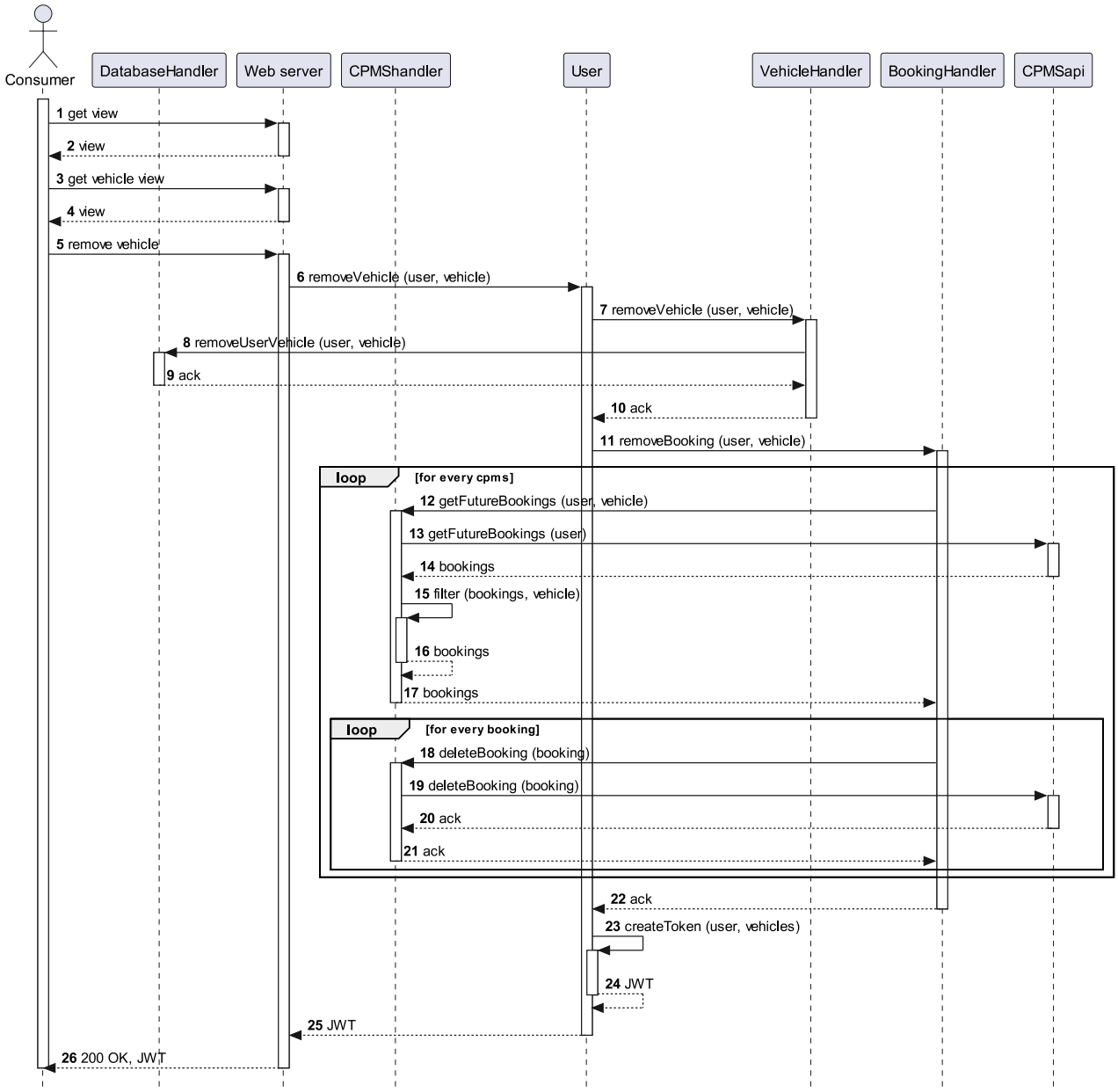

Figure 2.18: the user removes one of his/her vehicles.

Addition of a new vehicle The user adds a new vehicle to the system by providing the required data and its certificate.

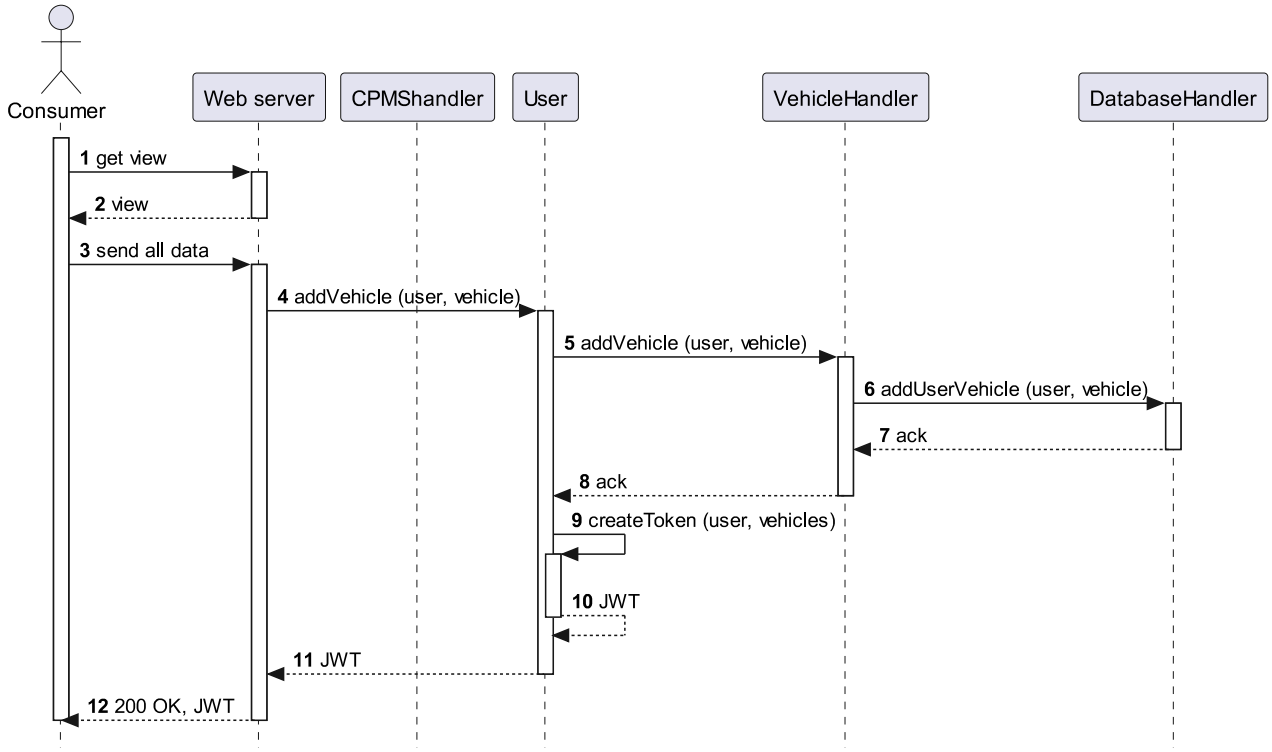

Figure 2.19: the user adds a new vehicle.

Other functions In the 2.4 Components interfaces (page 16) section there are some eMSP functions that are not used in these sequence diagrams, in particular:

- `deleteUser (user)` *from* `UserInterface` *and* `DatabaseInterface`: if the user doesn't have any pending payment, the system removes all his/her data from the system and all the future bookings.
- `export (user)` *from* `UserInterface`: allows the user to export all his/her data collected by the eMSP and the CPMS, according to the GDPR.
- `getAllBookings (user)` *from* `CPMSinterface` *and* `UserInterface`: retrieves all the booking for that user on every connected CPMS.
- `resetPassword (user)` *and* `resetPassword (id, link)` *from* `UserInterface` *and* `DatabaseInterface`: these two methods allow the user to reset his/her access password by proving the email address used for logging in. The system sends the user an email with the link for changing it.
- `updateUser (user, newData)` *from* `UserInterface`: allows the user to update all his/her data, including the password.
- `updateDevice (user, device)` *from* `DeviceInterface` *and* `DatabaseInterface`: even if it's not presented in the diagrams, it's called on every request for keeping updated the list of devices of the user, which are automatically cleaned up after a while if unused.
- `handshake (cpms)` *from* `CPMSinterface` *and* `addCPMS (cpms)` *and* `getInfo ()`[1] *from* `DatabaseInterface`: these functions allow the system to conncet to a newly discovered CPMS and to store it's information into the database.
- `getCPMS (id)` *and* `getCPMSs ()` *from* `DatabaseInterface`: allow retrieving data about a specific CPMS or about every connected one.

[1] This function (`getInfo ()`) allows to retrieve all the eMSP's information that need to be sent to the newly connected CPMS.

2.5.2 CPMS

Login The user logs into the system, which recognizes him/her and sends back a signed JWT for future actions on the system, like monitor stations. This token is properly checked on every request by the web server.
After authentication, the home page view is sent, then asynchronous sub-processes are run to fetch and send information about stations and DSOs. This information is sent in the form of JSON files.
From login until logout (or until a certain inactivity time has elapsed) the `Web Server` process will not end.

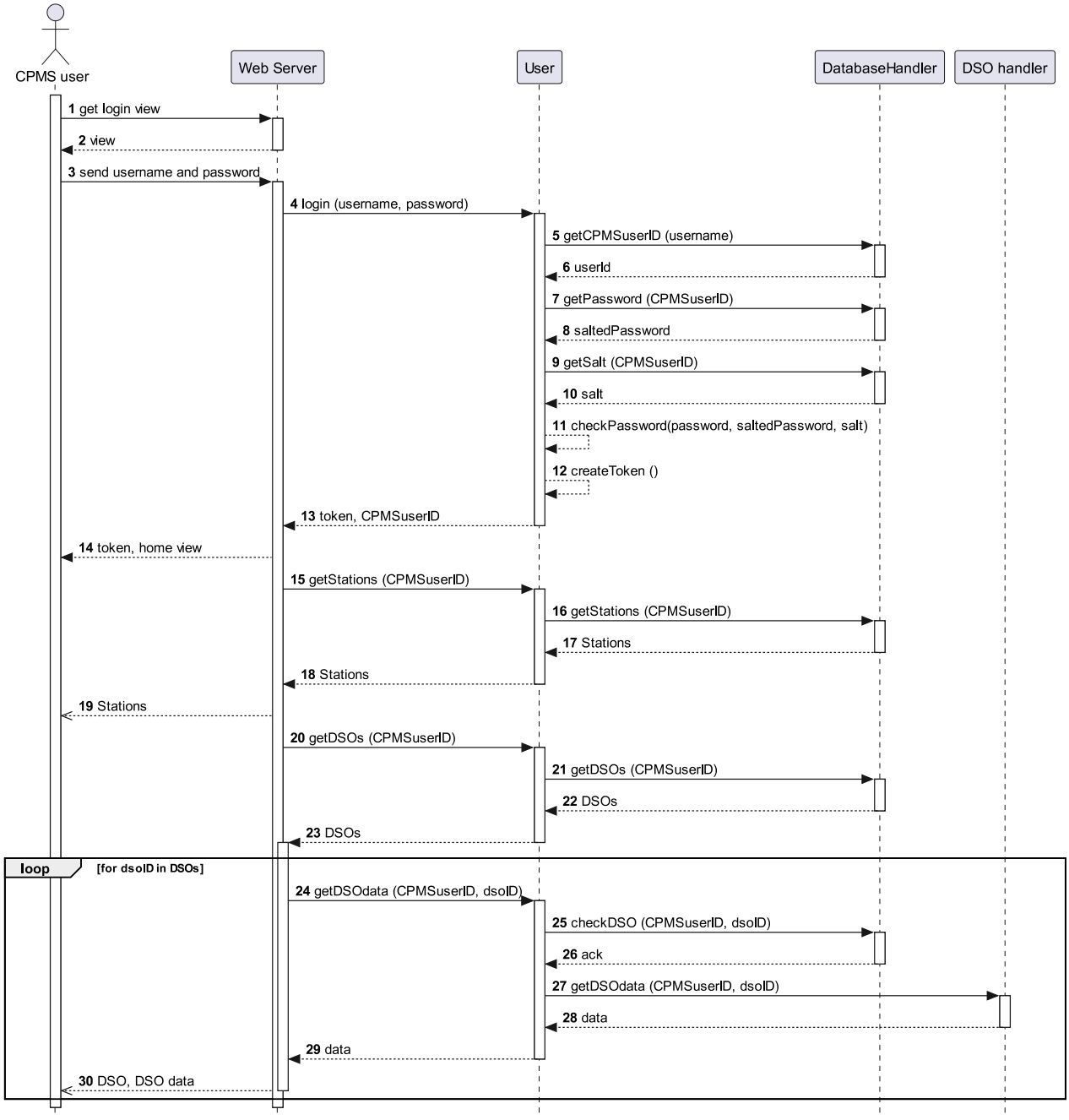

Figure 2.20: the user logs into the CPMS website.

Monitoring a station In the "Station" view users can monitor the situation inside stations. Until they leave the page, the `Web Server` will run an asynchronous periodical loop to fetch new information to send in JSON files.

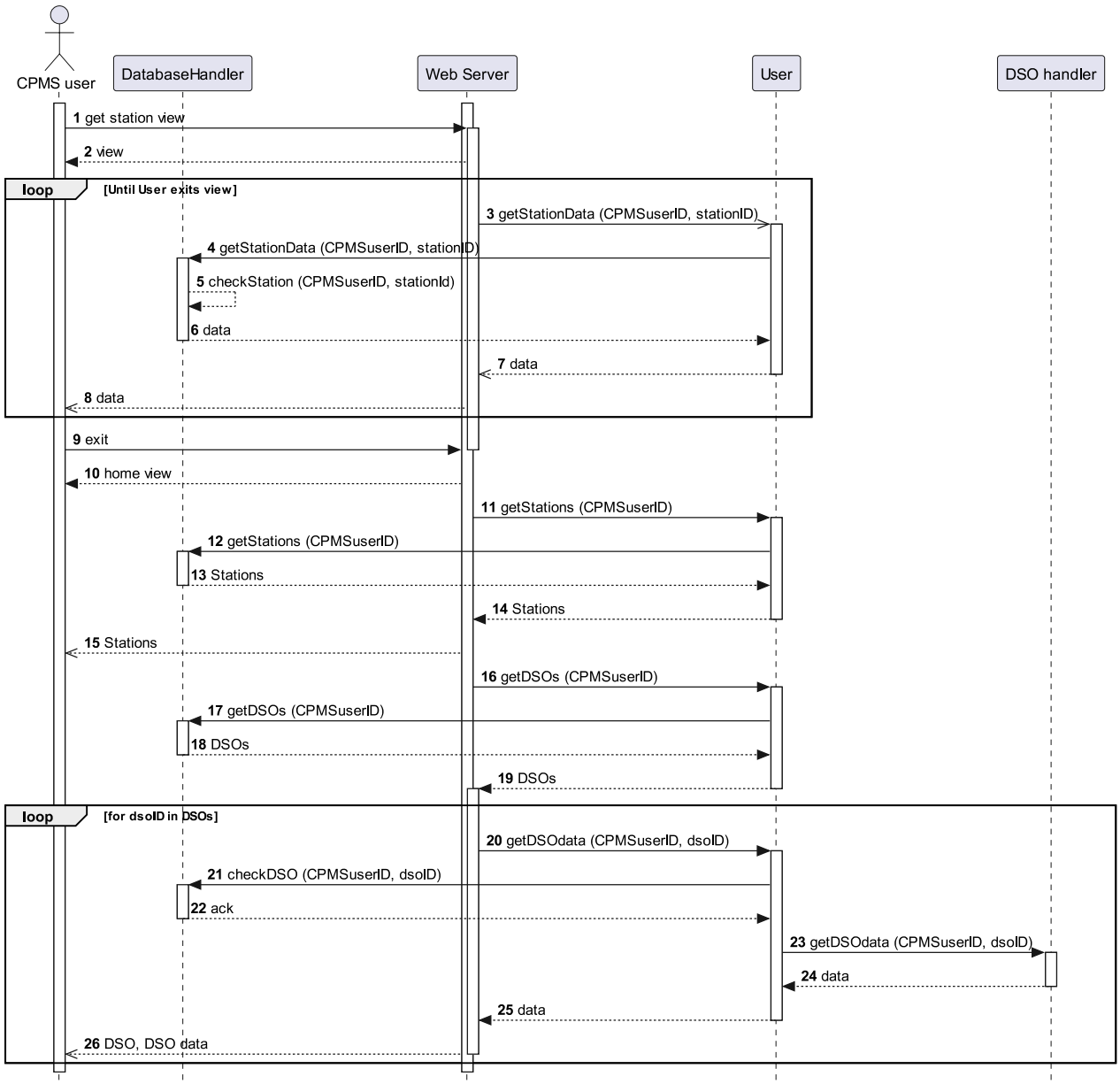

Figure 2.21: the user selects a station to view, then goes back to the home page.

Activating automatic energy mix From a "Station" view, the user selects "Automatic Mix choice", then decides to log out. Since "automatic mix" is activated, the `ChargingStation` process enters a loop until this option is deactivated. In this loop, it periodically fetches information from sensors, decides what the best energy mix is, then waits some time.

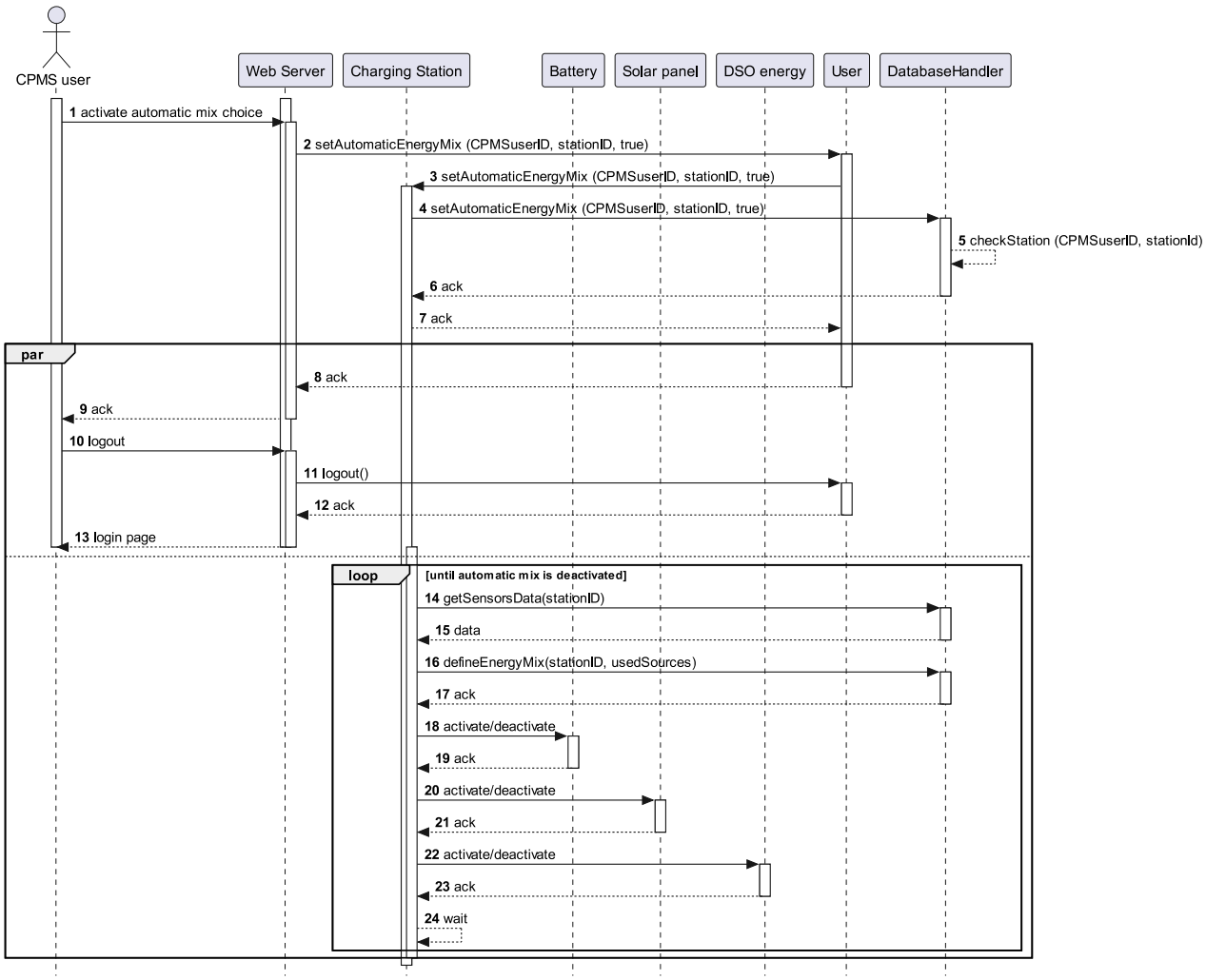

Figure 2.22: the user activates automatic mix choice, then logs out of the website.

DSO choice From a "Station" view, the user selects "Automatic DSO choice", then selects another DSO, ending the automatic choice. Since "automatic DSO choice" is activated, the `ChargingStation` process enters a loop until this option is deactivated. In this loop, it periodically fetches information from DSOs, decides what the best price is, selects that DSO, then waits some time.

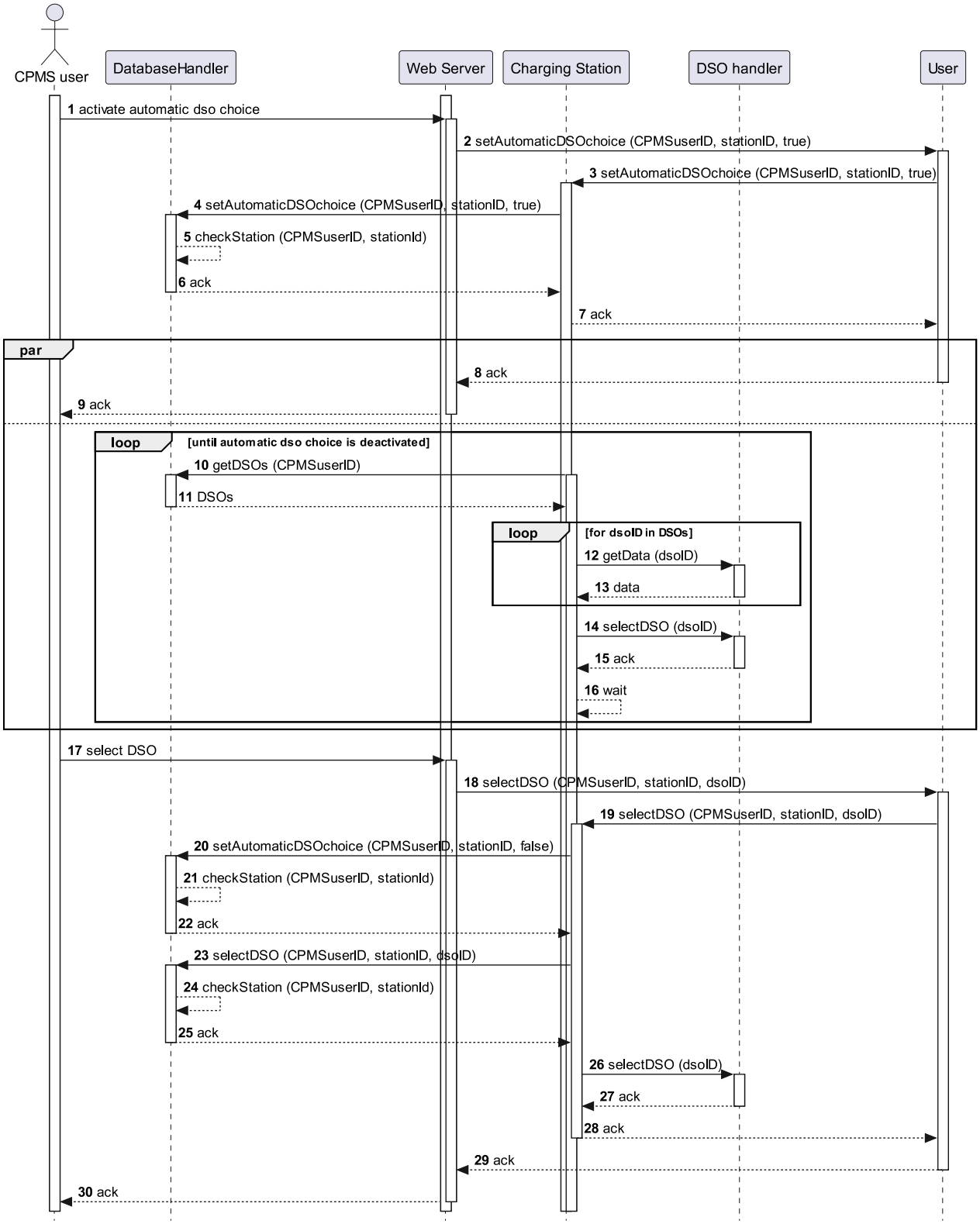

Figure 2.23: the user activates automatic DSO choice, then deactivates it by selecting a specific DSO.

Create a new special offer The user enters the "Offer" view and creates a new special offer.

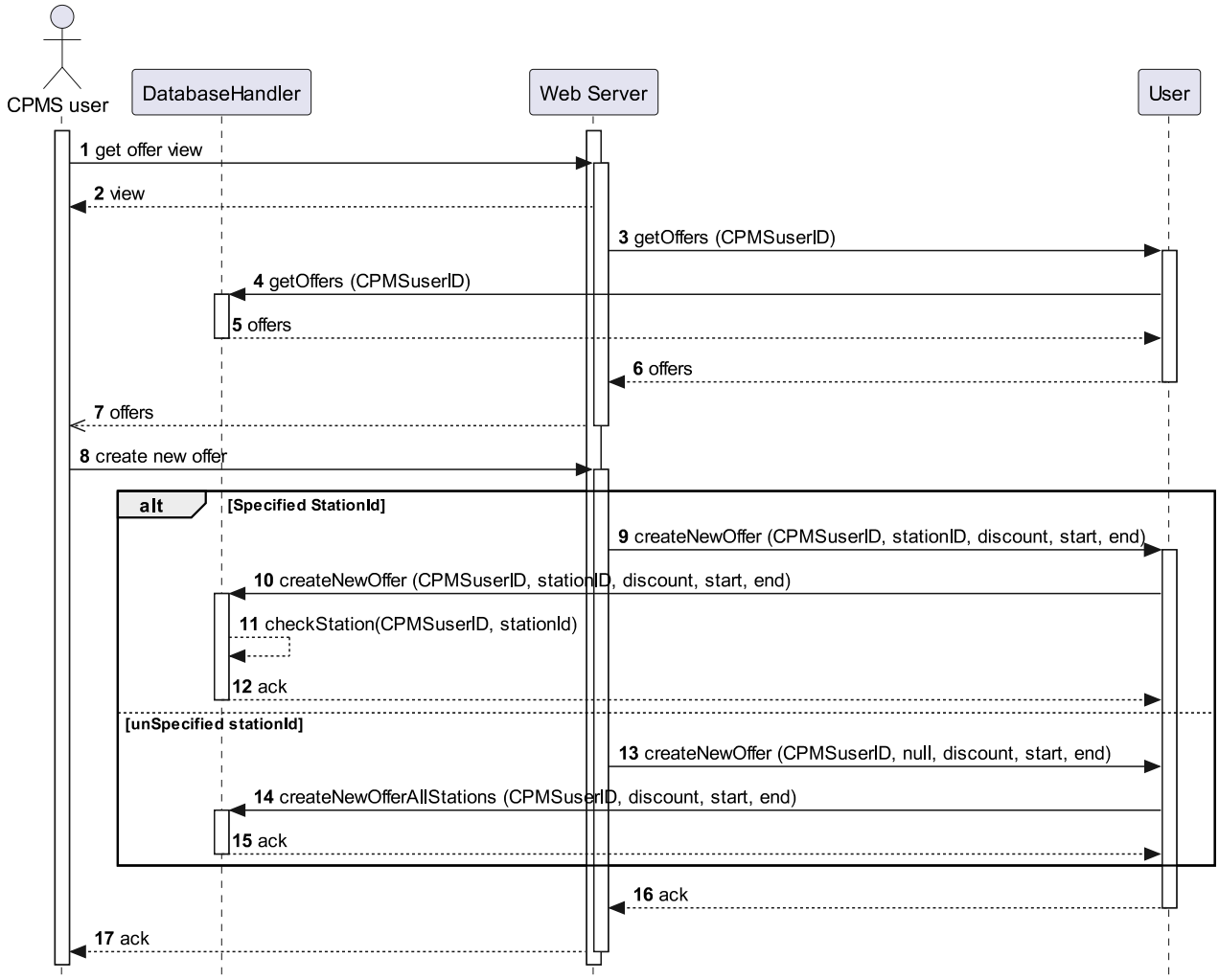

Figure 2.24: the user creates a new special offer.

Other functions The CPMS system allows for other actions to occur, but they are not depicted here because they are fairly basic, and their diagram can be better described with words. Every notification to the eMSP is sent through the `sendNotification (eMSP_ID, eMSPuserID, message)` function of the `eMSPinterface`. eMSP users are allowed to pay the charge at the charging station, therefore `ChargingColumn` can access the `PaymentInterface` in order to use the function `payCharge (eMSPuserID, paymentData)`. eMSP add, delete, update and query bookings through the use of the various "Booking" functions of the `BookingInterface` and the `DatabaseInterface`. eMSPs are added to the database though the function `add_eMSP (eMSP)` called by the eRoamingHandler.

2.6 Selected architectural styles and patterns

Here are presented all the design patterns that have been chosen for building up the two systems, which are very similar in this aspect.

2.6.1 Three-tier architecture

Both the eMSP and the CPMS present a three-tier architecture. This is because it helps to divide the local functions of every component into three different "classes" (presentation, business and data) which have different duties. The presentation layer is in charge of directly answering the user's requests, activating the required components in the backend, and collecting the final answer. The data layer is in charge of keeping organized all the data of the system. Finally, the business is the logical core of the system, indeed it provides the needed answers to the frontend by indirectly analyzing the requests from the client and picking and processing the required data from the data layer.

2.6.2 Microservice architecture

Both the eMSP and the CPMS present a microservice architecture. This design decision has been taken because it allows the system to be more resilient, to be able to scale better (e.g. duplicating the bottleneck components) and it can also be developed, updated, and tested by different teams, allowing a faster startup.

2.6.3 Containerized architecture

Both the eMSP and the CPMS present a containerized business layer. For providing even stronger security and resilience to failures, all those components are containerized, which means that all run on a different "virtual" environment, without the knowledge of the whole system on which they are running. Thanks to the use of a Kubernetes service, it's possible to orchestrate multiple machines in order to proactively react to the traffic to the service and easily update components. This can be done thanks to the volatile nature of the containers, which can be brought up and down and updated in a really easy way.

2.6.4 Database-centric architecture

Both the eMSP and the CPMS present a database-centric architecture. This implies that every component relies on the database for obtaining data and processing information. In case of failures of components, they can be restarted without any loss since all the information is stored in the database, which, itself, is a replicated piece of software.

2.6.5 RESTful architecture

Both the eMSP and the CPMS present a RESTful architecture. This is especially true in the client-server relation since the client only asks the server for HTML, CSS and JavaScript files for correctly visualizing pages (which is not required in the case of the eMSP's mobile application) and then queries the server through standard HTTP requests for the required data, providing a JWT at every request for being recognized.

2.6.6 Observer design pattern

Components in the CPMS system can implement the observer pattern: `ChargingColumn` and `EnergySource` offer through their interfaces the `addObserver (observer)` function, `ChargingStation` can then add observers to them. These observers are used to prevent in-station batteries from passing a certain charge level or to send a notification when the user's car's charge ends.

Chapter 3

User Interface Design

This third chapter is about the frontend interfaces the users will use for accessing and interacting with the two systems. For every system, there is a brief presentation of all the main interfaces they are composed of, and then, thanks to the use of flow diagrams, the navigation flow is presented.

These represent only the interfaces used by the end users of the two systems, the pages for the system administrators are not presented here, since their views are similar to these or consist only of a list of employees with the possibility to add or remove them (which is especially true in the CPMS).

Moreover, these interfaces are presented only in day/light mode for better readability, but in real-case scenarios, they adapt to the user's (or browser's) preferred theme, so they come also with a night/dark mode.

3.1 eMSP

The eMSP is the system that interacts with the end user of the whole system: the consumer. In this section, the only presented mockups are the ones of the mobile application. The ones available from the browser are the same, with the only difference being that in the case of a landscape design (which is more common on desktop/personal computers), the elements of the interfaces are organized in a slightly different way to better adapt to the display.

3.1.1 Interfaces design

Here are presented all the mockups of the pages of the mobile application of the eMSP. The mobile application is available on all the main application stores for mobile phones available nowadays, like *App Store* (Apple), *Google Play Store* (Google), and *AppGallery* (Huawei). The browser version is slightly different since it's usually used on a landscape device.

Login and signup When the user first opens the application or the personal area of the eMSP website, he is prompted to log into the application, but if the user doesn't still have an account, there is the possibility to go to the signup page (or registration page) where he is asked to insert all of his data. The interface also offers the possibility to display (or to hide) the password and to display a date picker to easily insert the birthdate.

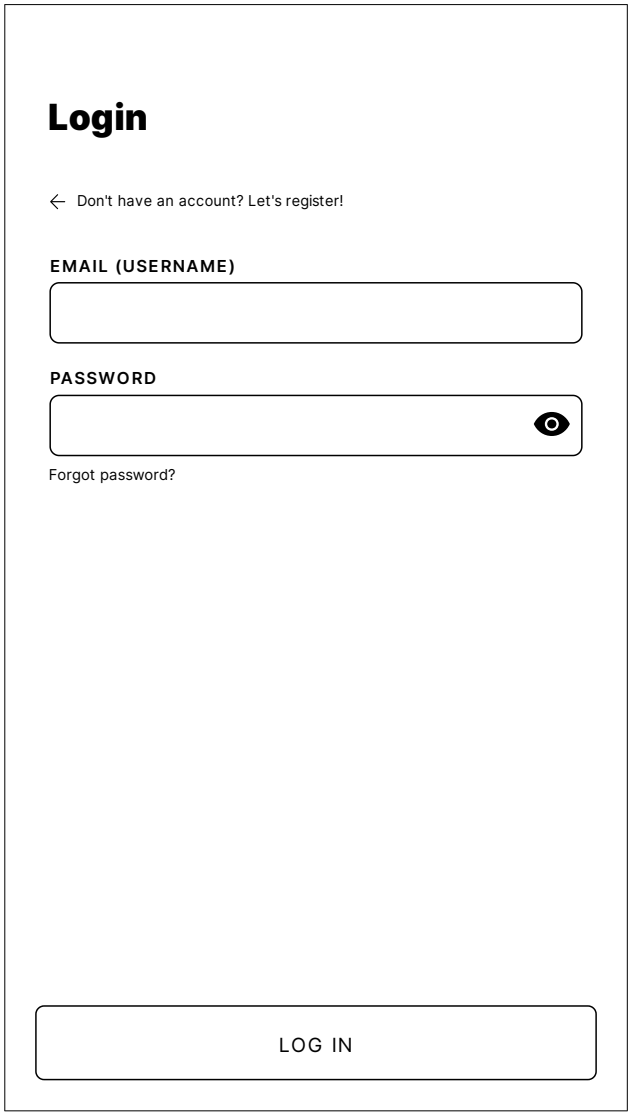

Figure 3.1: login page.

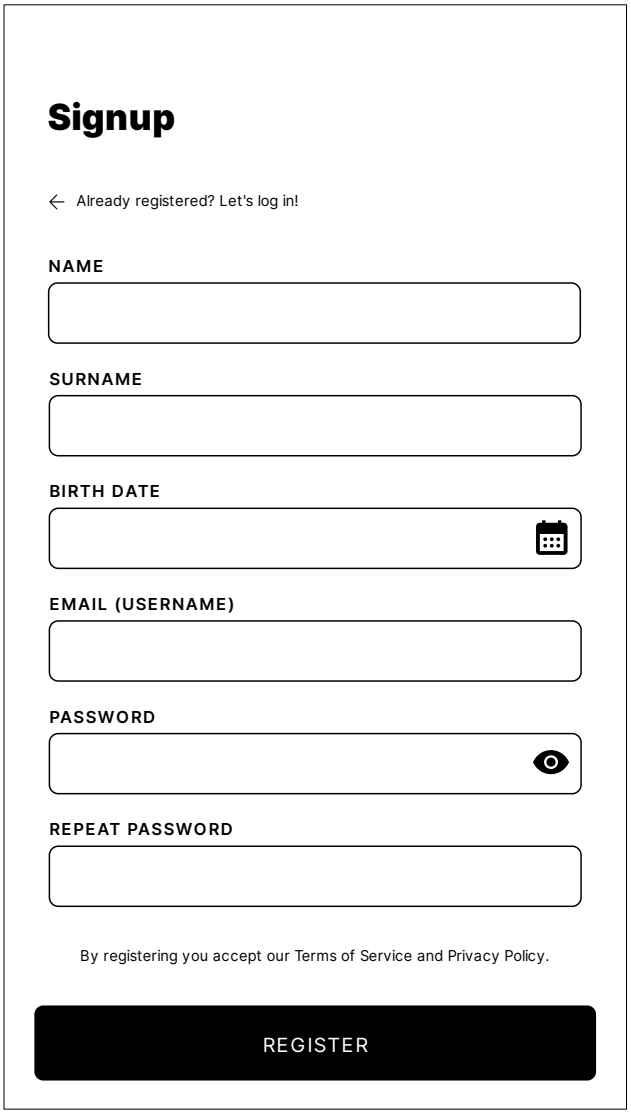

Figure 3.2: signup (registration) page.

Post registration actions After the uses registers to the application, the system sends him/her an email with a link for activating the account. Once the user clicks on it, s/he is sent to the login page where a popup tells the user that his/her registration ended up successfully. After closing the banner, the user can log into the application, and since it's the first login, the home page is displayed and s/he is asked whether to use any logged-in device for sending notifications or the associated email address.

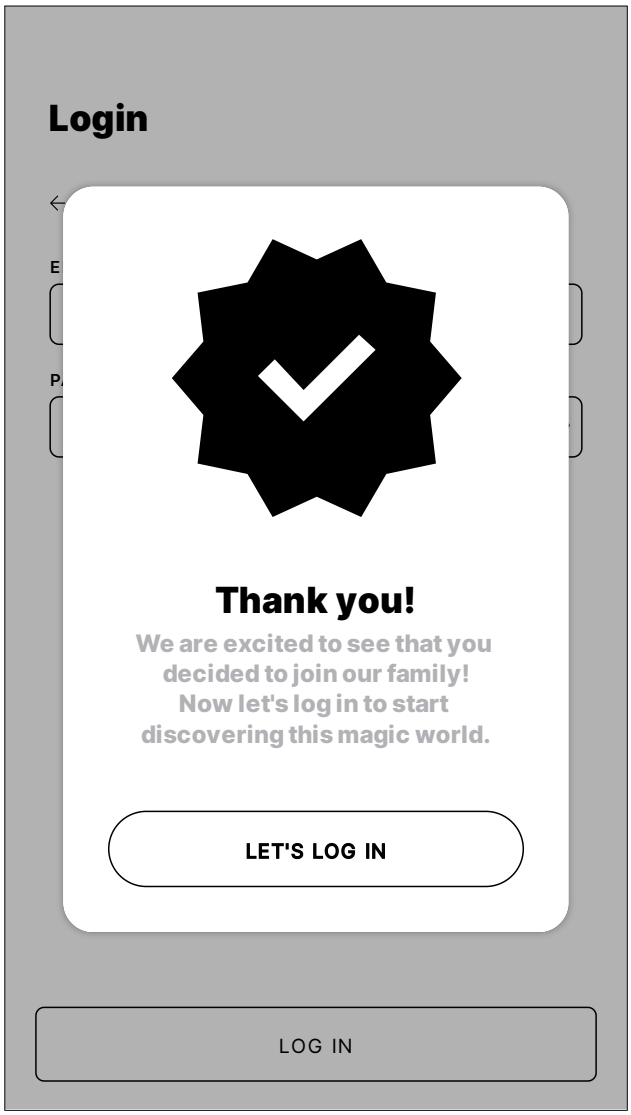

Figure 3.3: successful registration message.

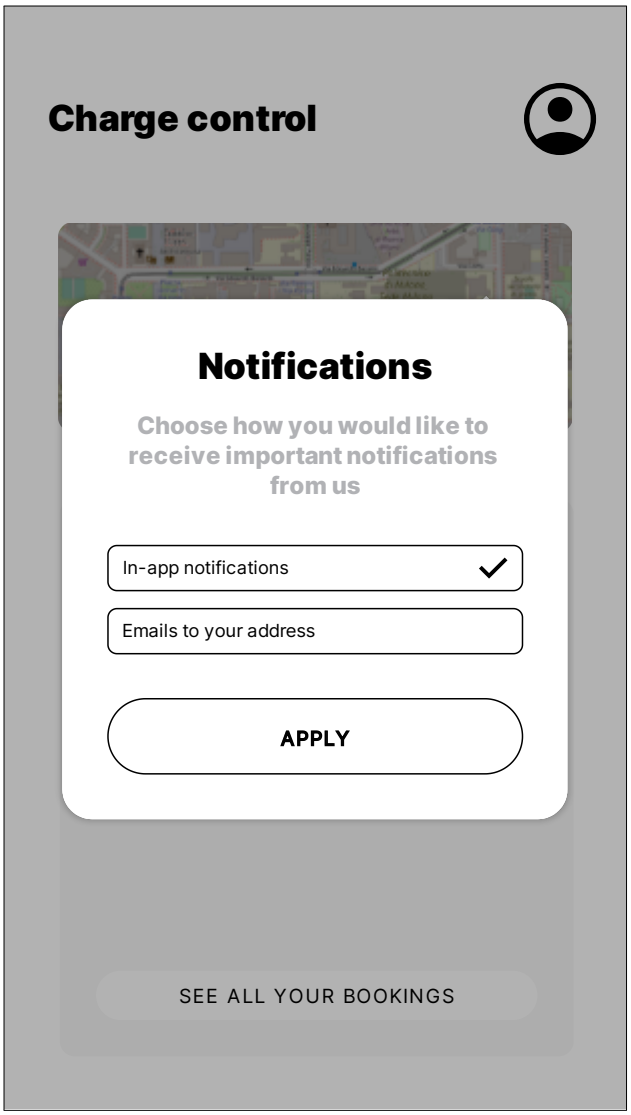

Figure 3.4: first choice of the notification method.

Home page The home page is the center of the application. From here the user can reach his profile with all the details, the map for booking charges, the history of his charges, and the future ones (which also appear on the home page, as shown in Figure 3.6), and, if any, a button for paying all the unpaid charges.

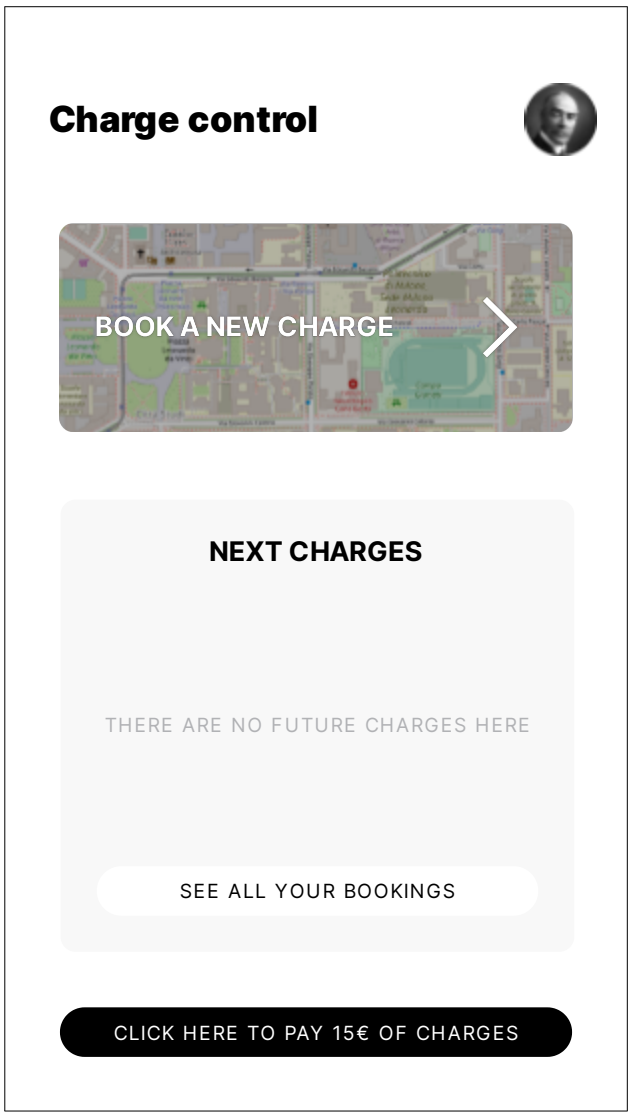

Figure 3.5: home page with no bookings.

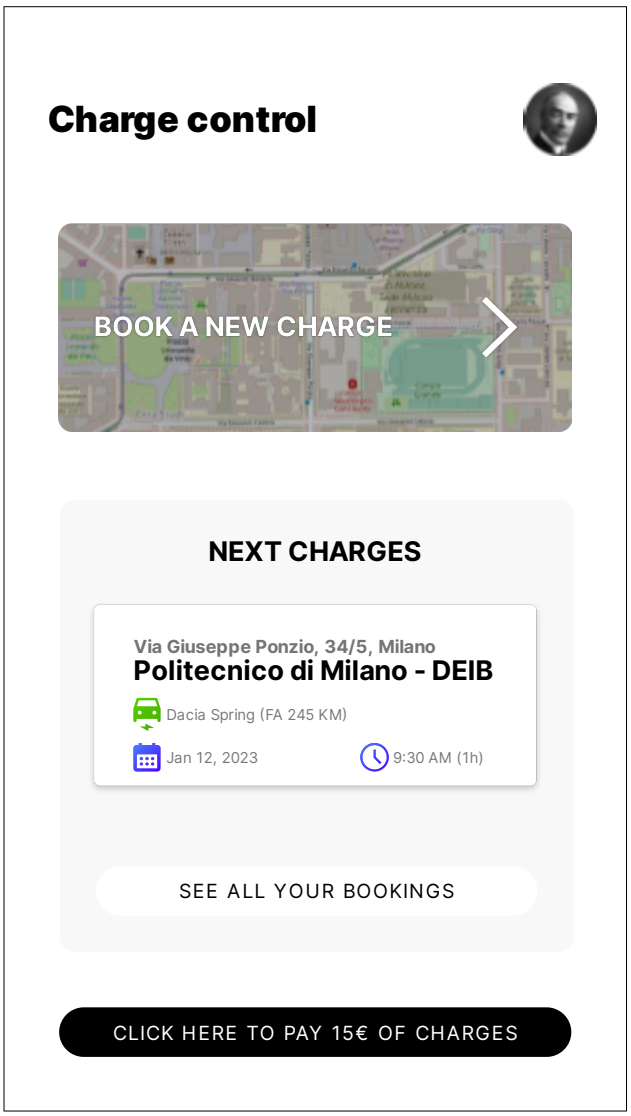

Figure 3.6: home page with bookings.

User's profile From this page, the user can see all the data s/he uploaded to the system and can update, download, and delete it. Moreover, from this point, the user can log out from the system, change the preferred notification method and reach his/her vehicles page.

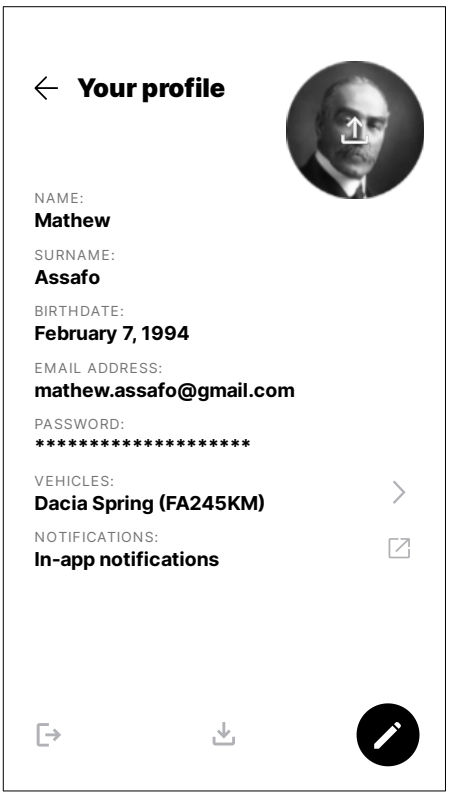

Figure 3.7: profile page with all the details.

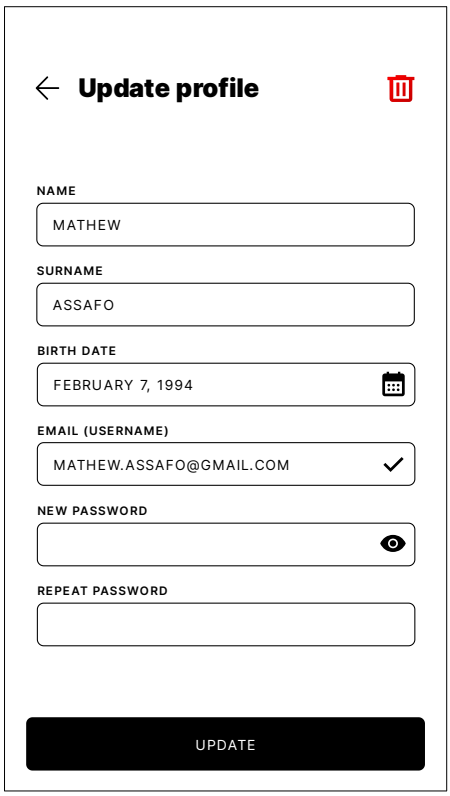

Figure 3.8: page for profile update.

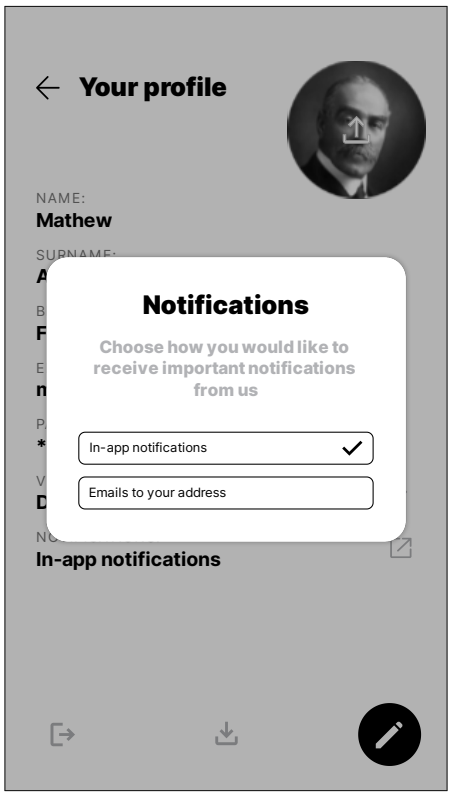

Figure 3.9: notification preference change on user's profile.

Stations lookup: map view The map view allows the user to see all the charging stations around thanks to the use of OpenStreetMap. The user can activate and deactivate the localization functionality, search for a specific location, look at his/her list of favorite stations or move to the list view.

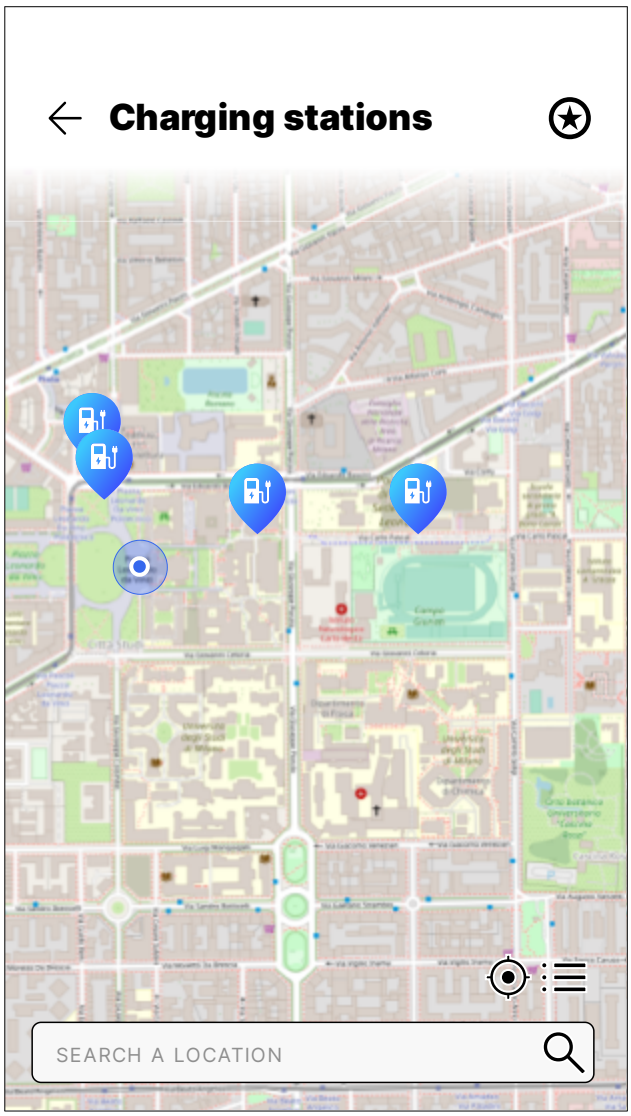

Figure 3.10: map view.

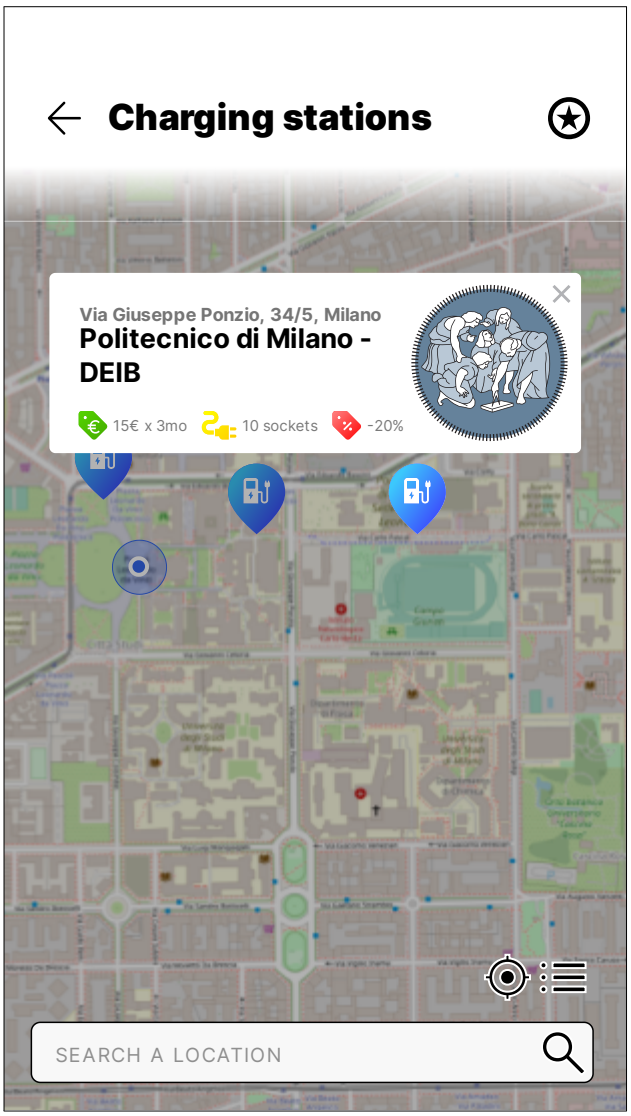

Figure 3.11: map view when a station is selected.

Stations lookup: list view and favorites The list view acts like the map view, but instead of presenting the charging stations in the map, it shows them in a list form which allows the user to sort them according to various parameters[1], to go back to the map view or to see the list of favorite stations. Moreover, it allows the user to mark or unmark any station as favorite.

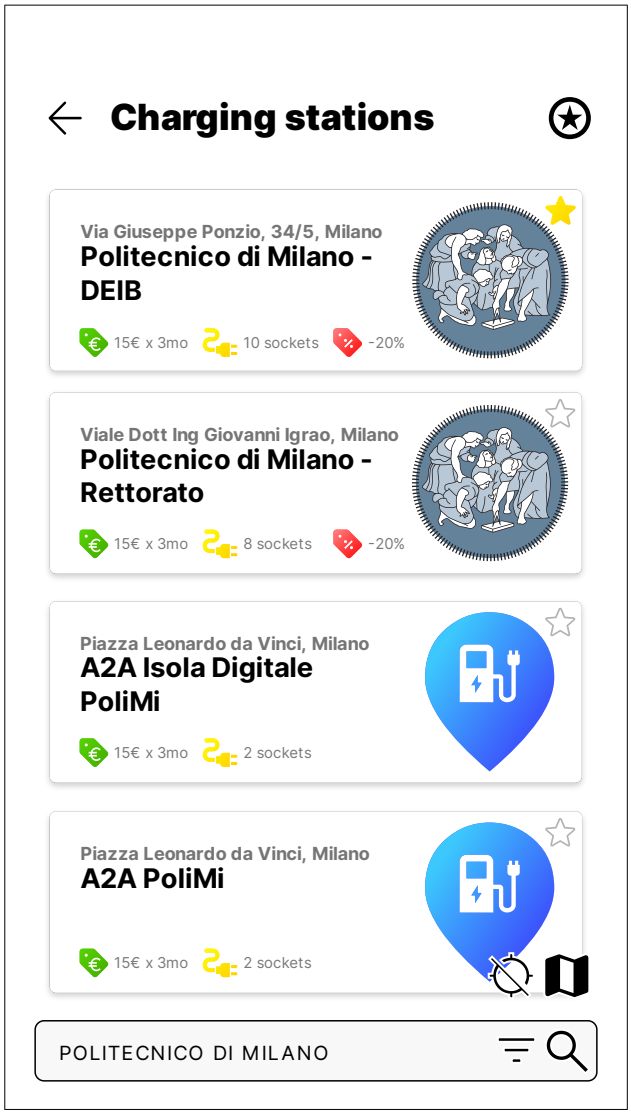

Figure 3.12: list view.

Figure 3.13: favorites view.

[1]The popup used for selecting the sorting parameters is the classical one of each system, so it's not represented here, but it includes the distance from the selected point, the price (with the presence of any eventual discount) and the availability, both in ascending and descending order.

Book a charge Once the user has selected from the map or list view a station, s/he can open its page from which he can see all the data of that station and can book a charge (and also can toggle the "favorite" state). Once the user clicks on the booking button, a popup appears giving the possibility to book a slot for a specific date and time for one of his/her vehicles.

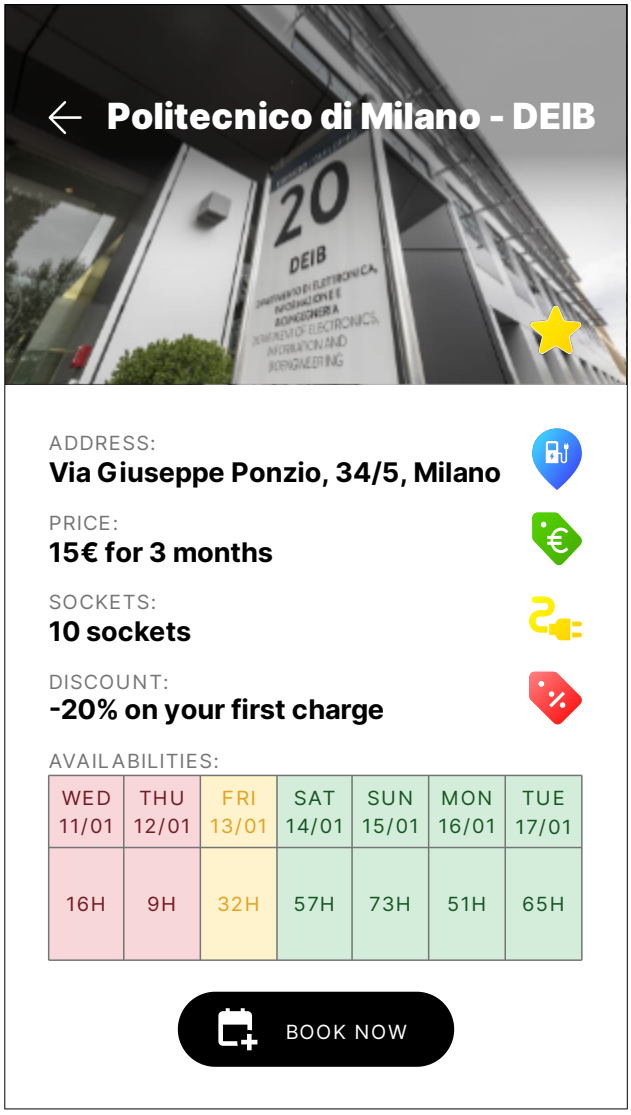

Figure 3.14: charging station's page.

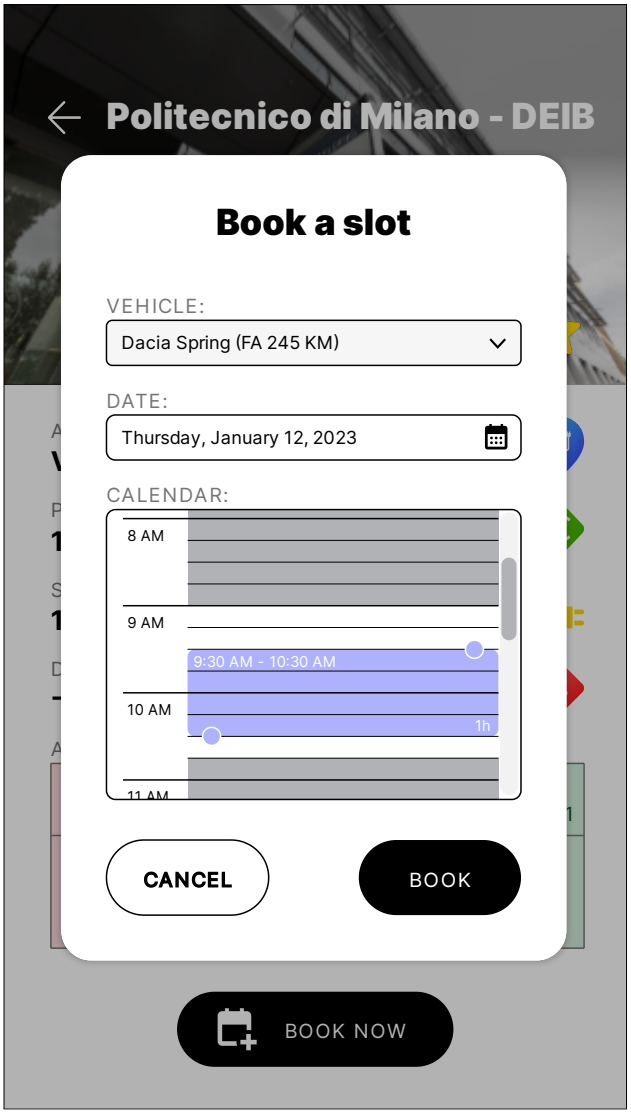

Figure 3.15: station's page with booking popup.

Bookings views This view can be reached from the home page and lists all the future and past bookings of the user. Any future (or current) charge can be opened on a dedicated page in order to show all the details (including the assigned slot if present) and can be edited or deleted. Moreover, the red past bookings are the ones that still have to be paid, and for doing that there is an apposite button on the end of the page.

Figure 3.16: the page with all the bookings.

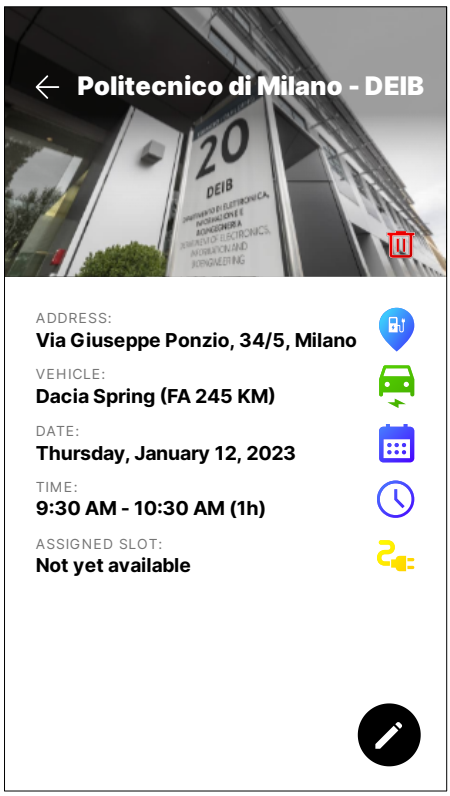

Figure 3.17: a booking page with all the details.

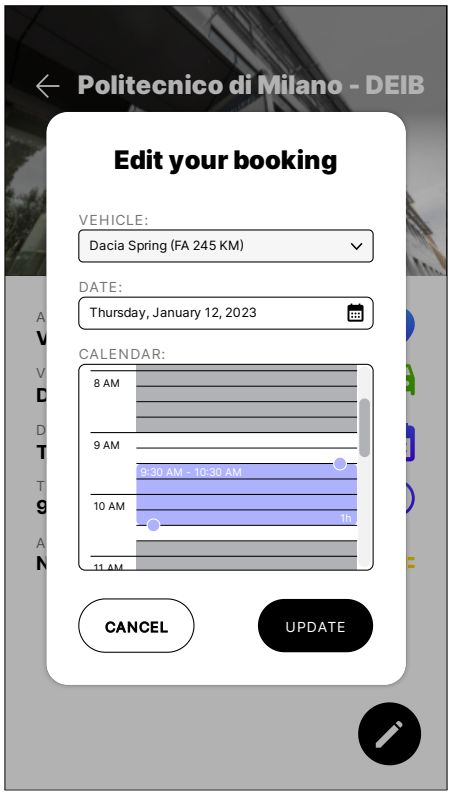

Figure 3.18: the editing popup of the booking.

Vehicles The user can add a new vehicle, update it or delete it at any time. If the change interests some future charge (in general, any non-started charge), the system acts as a consequence as stated in chapter 2 Architectural Design (page 8), for example deleting the booking or updating the certificate for the charge.

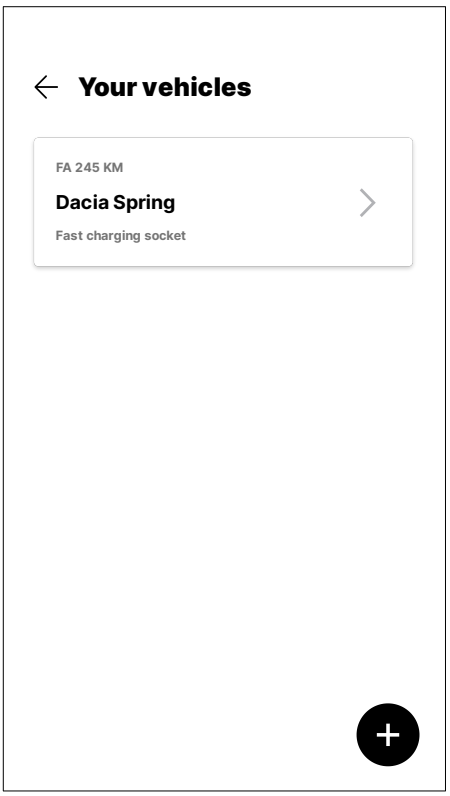

Figure 3.19: the page with all the vehicles.

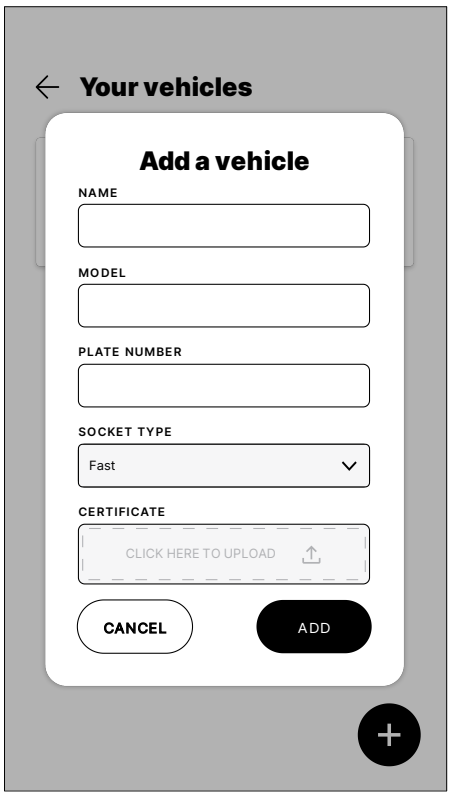

Figure 3.20: the popup for adding a new vehicle.

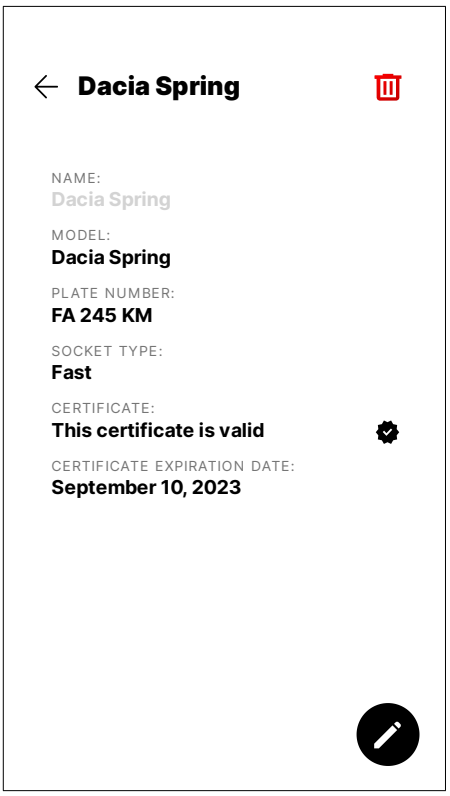

Figure 3.21: the details of a vehicle.

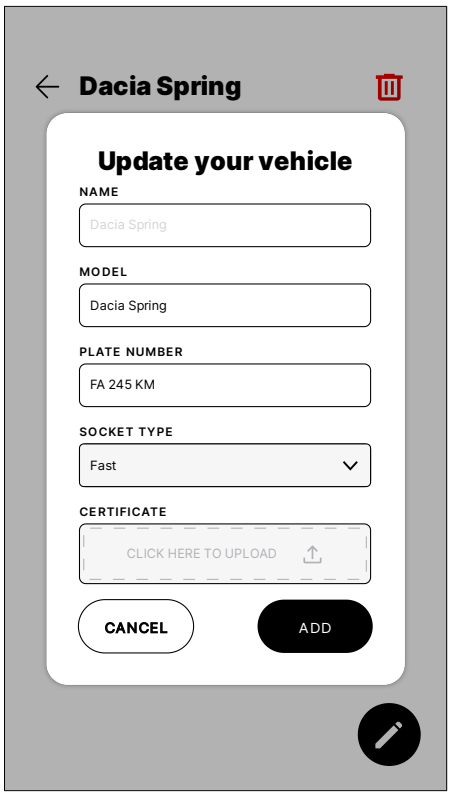

Figure 3.22: the popup for editing a vehicle.

Payment Before the user pays from the application the charges, the system shows this page allowing him/her to choose the method s/he prefers for paying. The system supports multiple transaction managers like the ones shown in this interface, but it can be expanded in the future to support more.

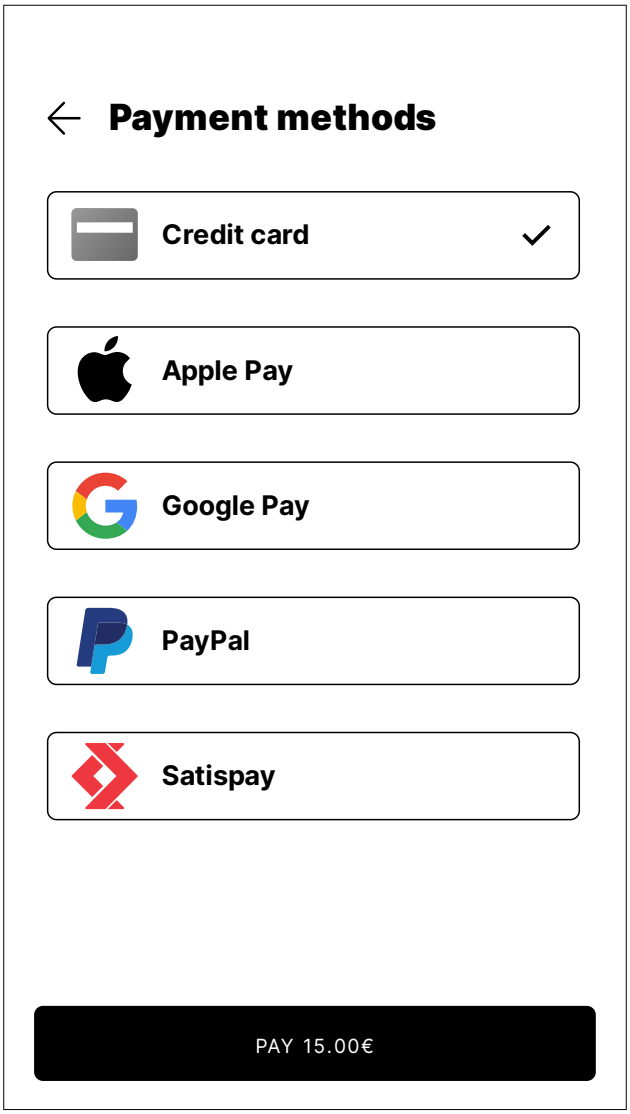

Figure 3.23: the choice of the payment method.

3.1.2 Pages connections

This diagram shows how all the presented interfaces are connected. The white circle at the top left of the diagram presents the starting point, which can be either the opening of the application or the opening of the private area on the eMSP's website.

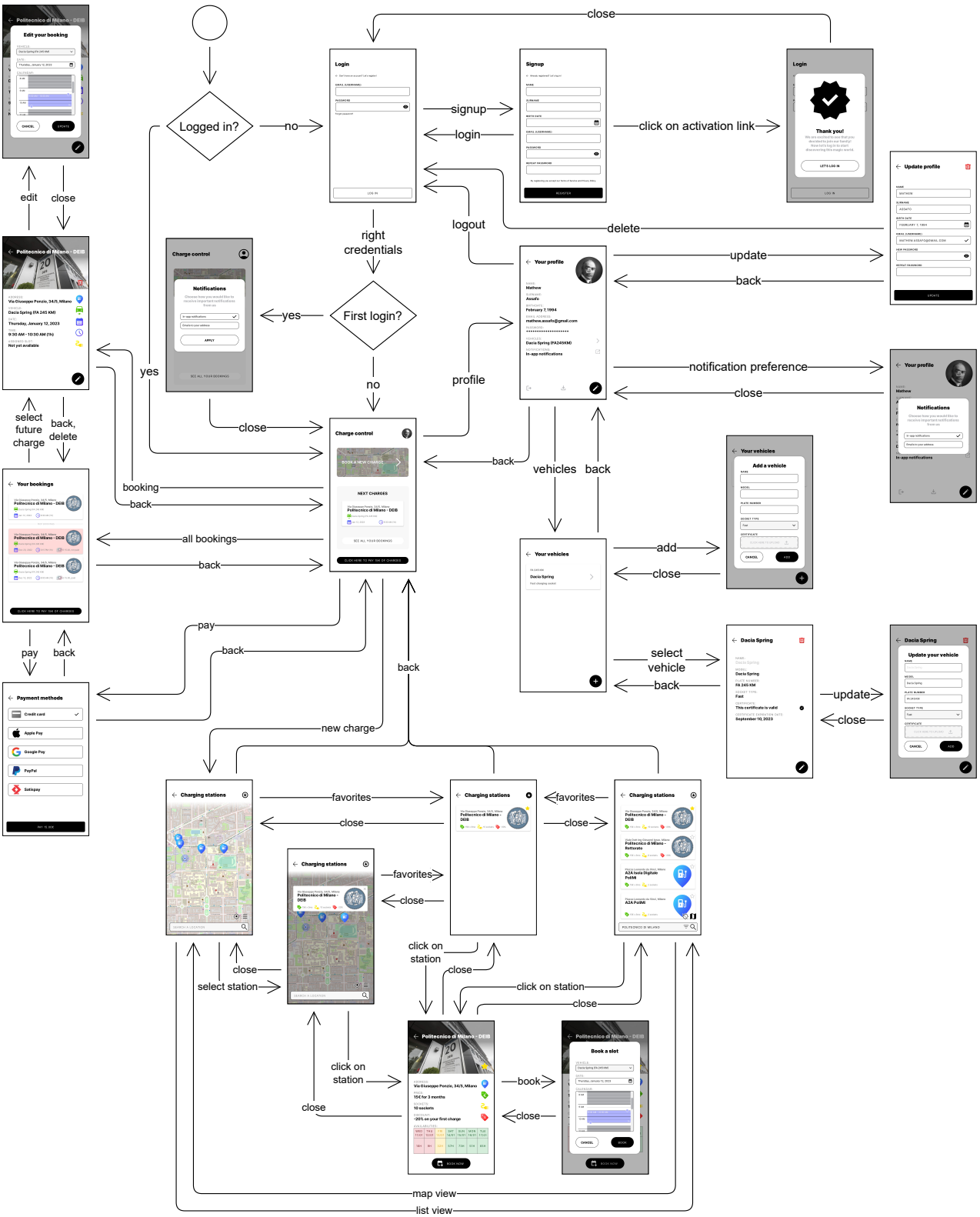

Figure 3.24: the connections between the various interfaces.

3.2 CPMS

Interactions with the CPMS system are conducted by authorized CPMS users. These users are expected to interact with the CPMS to carry out occasional tasks, as the main system is thought to be automatized. Nonetheless, the CPMS system offers a website capable of allowing all reasonable actions to the users, with an immediate and fairly easy-to-use interface. In this section, mockups of the website are presented to resemble what the actual site could look like.

3.2.1 Interfaces design

To represent the website for the CPMS system, instances of a *Firefox* browser from a *macOS*-looking operating system are shown, but any browser on any operating system would work, as long as it's running on a working computer. Viewing the website from a mobile phone would be strongly discouraged.

Login In order to access the website, users must first login through the login page. The standard username and password are the only requirements, but the system can be expanded to include the need for a second factor for authentication.

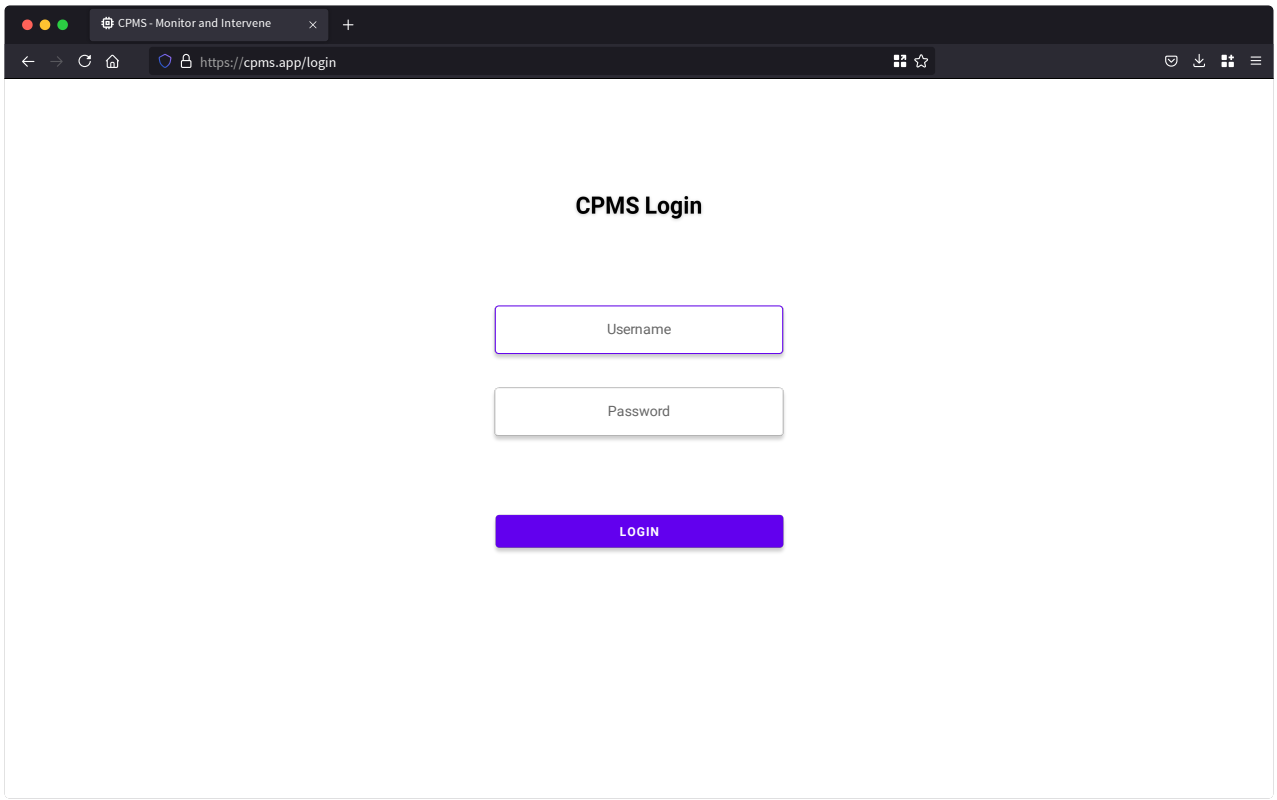

Figure 3.25: CPMS login page.

Home After login, the web server sends the user the home page, then sends JSON files containing data on DSOs and stations managed by the user. The home page comprehends a brief statistic tab on a selected station for a selected metric, the central tab needed to access the station page of a selected station, and a brief statistic tab on all DSOs available to any of the stations. A button to go to the special offer page sits underneath the station tab. In the top left corner, a menu tab can be clicked to navigate the site, mainly to return to the home page or to go to the offers page. In the top right corner, the user tab can be clicked to see the user information, to change the password, or to log out of the website. In the bottom left corner, buttons are present to help navigate the website, for general information, and to send emails or to call for support. All the tabs and buttons in the corners are also present on the other pages of the website.

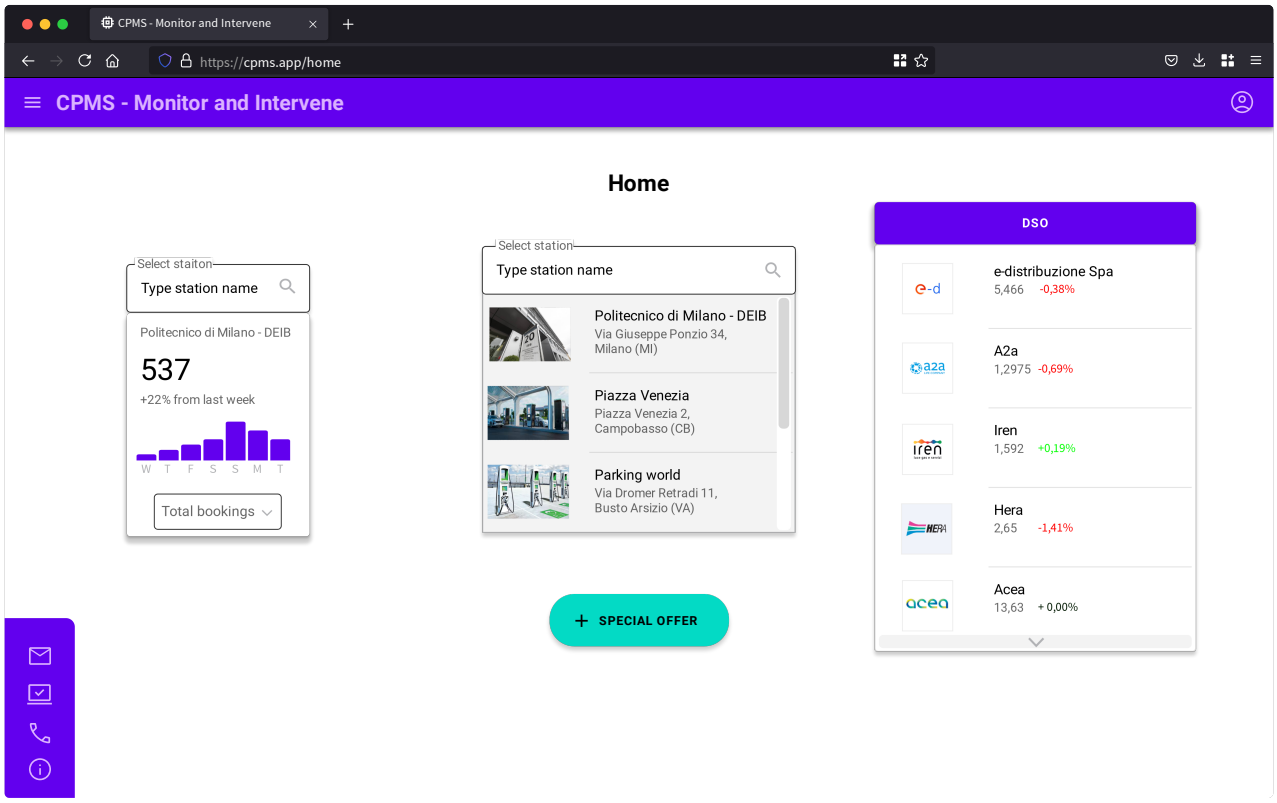

Figure 3.26: CPMS home page.

Station By searching (and then clicking on) a charging station, the web server sends the station page, then send data on the sources and DSO available, along with data on the state of the charging columns. On the station page, there are three main tabs. These are periodically updated with new data by the web server.

The tab in the middle informs the user of the various charging columns present in the charging station, along with information on their socket types and if they are free or occupied at the time of viewing.

The left tab is the DSO tab: if the automatic choice is selected (by pressing the button on top), this tab only informs of the DSO available to the station and the prices they offer, along with the currently selected DSO. Otherwise, the user is able to manually and statically select the preferred DSO by clicking on the button on the side of it.

The right tab is dedicated to the energy mix. By clicking on the bottom on top the user activates or deactivated the automatic mix choice. If the mix choice is set to manual, users can select if a source can be used or not, and if batteries are present, if these are to be charged or not (batteries cannot be charged while they are in use). The system will still check if illegal states are being reached by the manual choice of an energy mix, and take countermeasures consequently. For example, if not enough power can be obtained by the current setup, and the DSO energy is not being used, the system will automatically set the DSO energy to be used, and will communicate this choice to the user with the next periodic update of the page. Information on the charge level of the batteries and the power obtained from each source is also displayed in this tab.

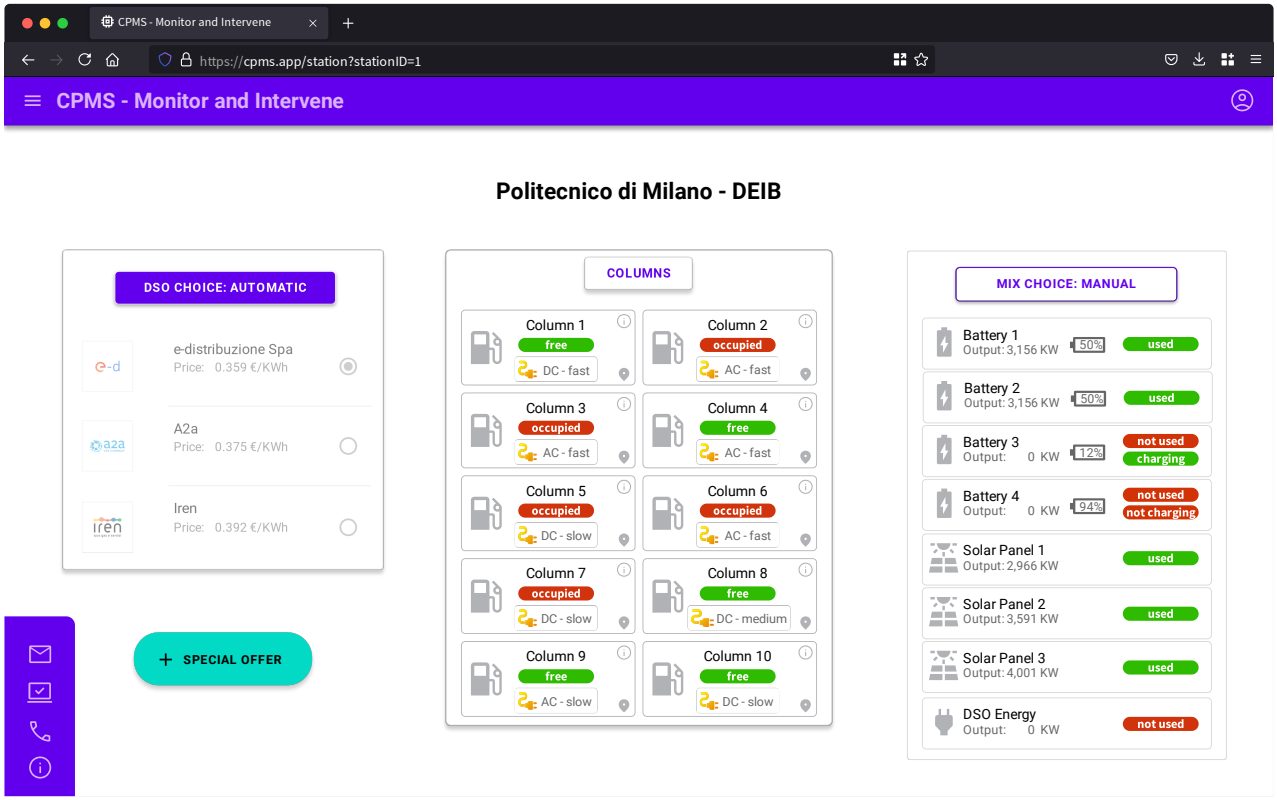

Figure 3.27: Charging station page.

Special offer In the special offer page, users can see all special offers present, and decide if they want to delete them or create a new special offer. A new special offer is created by clicking on the blue button with the plus sign. A pop-up appears where information on the new special offer is asked: the user has to select a start date (from the current day forward), an end date (that comes after the start date), one charging station affected (or all stations if that's the case) and the percentage amount of the discount. By clicking on "Create" the new special offer is created, by clicking on the "x" mark the whole creation is aborted.

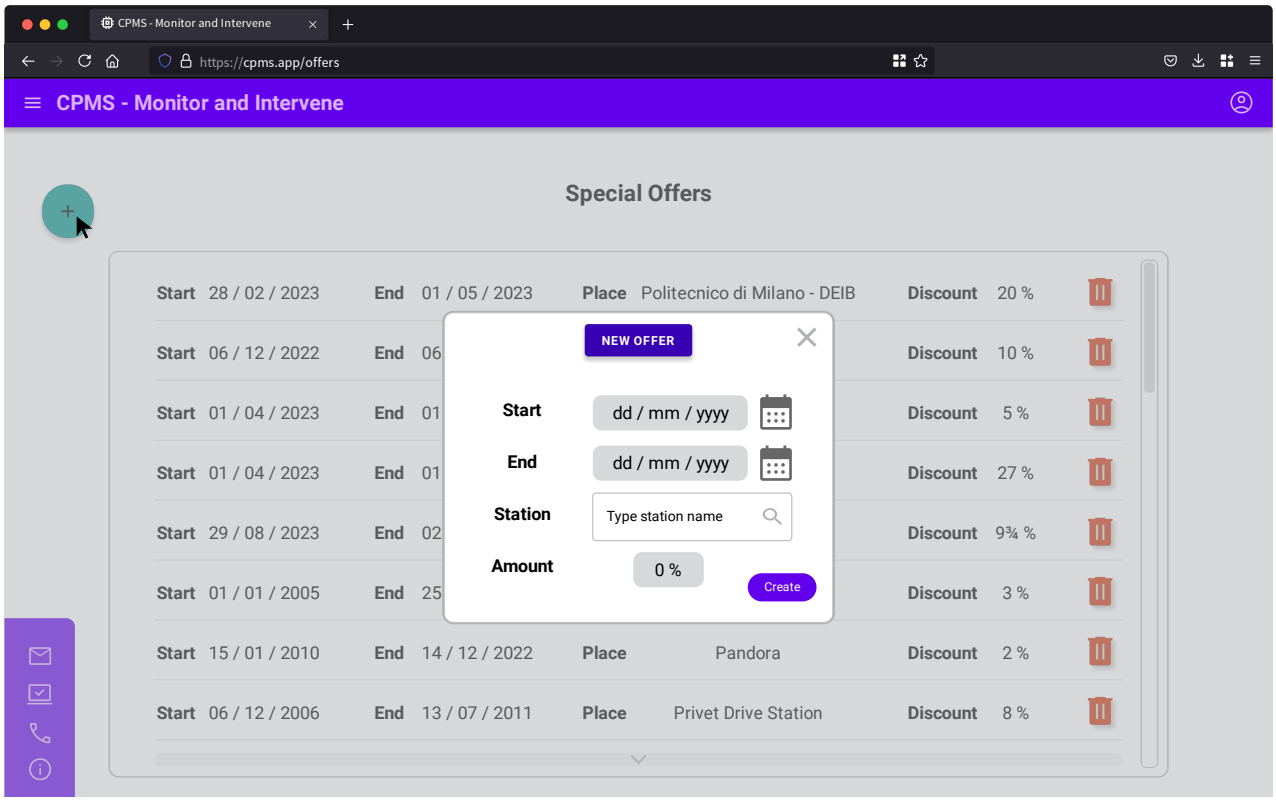

Figure 3.28: Charging station page.

3.2.2 Pages connections

This diagram shows how all the presented interfaces are connected. The white circle at the top left of the diagram presents the user opening the web page.

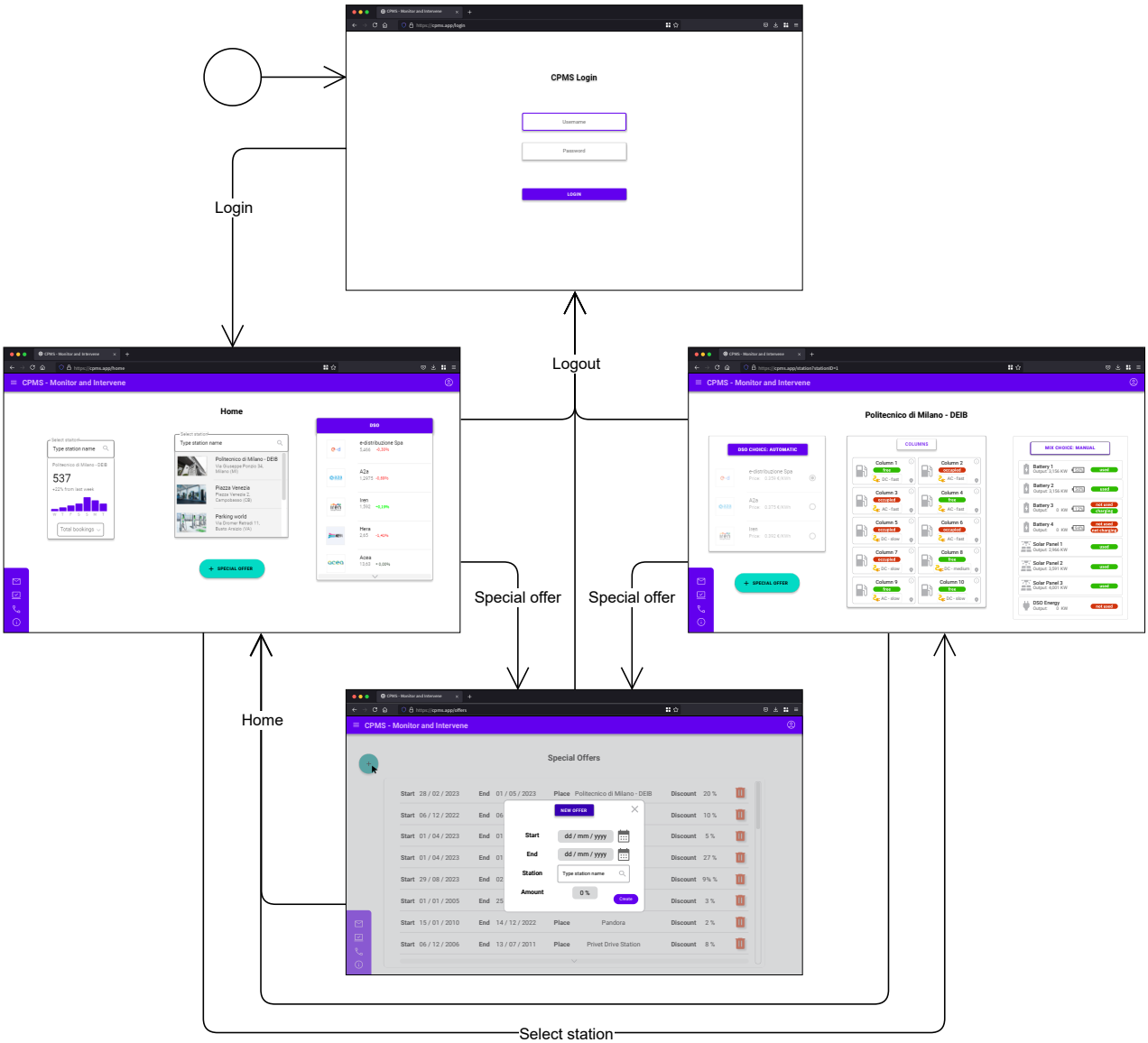

Figure 3.29: the connections between the various interfaces.

Chapter 4

Requirements Traceability

It's important to check that all the requirements pointed out into the *Requirements Analysis and Specification Document* of the eMall project are actually satisfied into the design of the system. That's why in this section every requirement is mapped into one or more components of the system.

4.1 Requirements

For reference purposes, here are reported all the requirements for the systems, divided into *eMSP requirements* and *CPMS requirements*.

4.1.1 eMSP requirements

Identifier	Description
R1	The system must allow the user to sign-up.
R2	The system must check the sign-up data.
R3	The system must send the user an email for activating the account after a successful sign-up.
R4	The system must discard any account activation request that arrives after 24 hours from the sign-up procedure.
R5	The system must consider any signed-up user who hasn't already activated the account as an unregistered user.
R6	The system must notify the user of the sign-up result, sending an email if it's successful.
R7	The system must allow the user to log in.
R8	The system must check the login data.
R9	The system must allow the user to change his/her password after logging in.
R10	The system must allow any registered user to reset his/her password by sending an email to the specified address.
R11	The system must discard any password change 24 hours after the request.
R12	The system must discard any duplicated usage of the same link for password change after a successful one.
R13	The system must allow the user to logout.
R14	The system must allow the user to set the preferred notification method.
R15	The system must allow the user to add a new vehicle.
R16	The system must allow the user to remove an associated vehicle, only if there is at least one.
R17	The system must allow the user to edit any associated vehicle's details.
R18	The system must allow the user to look for nearby stations.
R19	The system must allow the user to sort the nearby stations according to distance, price, and availability.
R20	The system must allow the user to mark and unmark stations as "favorite".
R21	The system must allow the user to book a charge in a charging station if any slot is available.
R22	The system must ask the user the vehicle s/he wants to charge every time s/he books a charge.

Identifier	Description
R23	The system must notify the user of any successful booked charge.
R24	The system must allow the user to add any booked charge to the calendar.
R25	The system must allow the user to show all the booked charges, both past, and future (there might be limits).
R26	The system must allow the user to edit or delete a booked charge before the end of the booked slot.
R27	The system must notify the user of the charging slot to use before the charge.
R28	The system must notify the user when the charge is finished.
R29	The system must allow the user to pay for the obtained service.
R30	The system must notify the user after every payment attempt (both in case of success and failure).
R31	The system must periodically contact an eRoaming service for obtaining the list of available CPOs (and respective CPMSs).
R32	The system must be able to connect to any CPMS that offers standard APIs.
R33	The system must be able to communicate with any connected CPMS through standard APIs.
R34	The system must allow any registered user to do everything specified, except signing up.
R35	The system must allow any unregistered user to only sign up and log in.
R36	The system must send the notification only to the user's authenticated devices if the notification preference is by in-app notification.
R37	The system must send emails instead of in-app notifications if that's the user's preferred method or if there is no available user's device.

4.1.2 CPMS requirements

Identifier	Description
R38	The system must check all input data correctness.
R39	The system must allow the user to log in.
R40	The system must allow the user to change his/her password after logging in.
R41	The system must allow any registered user to reset his/her password by sending an email to the specified address.
R42	The system must discard any password change 24 hours after the request.
R43	The system must discard any duplicated usage of the same link for password change after a successful one.
R44	The system must allow the user to logout.
R45	The system must provide an API function to get information about all bookings present for any charging station.
R46	The system must provide an API function to add/delete a booking for a charging point in a charging station.
R47	The system must provide an API function to get information about all charging points and respective sockets in a charging station.
R48	The system must provide an API function to get information about prices relative to a charging station and socket types.
R49	The system must start charging a car when it is first plugged in if it corresponds to the car for which a charge was booked in the corresponding time and place (same charging point/socket as well).
R50	The system must stop charging a car once it recognizes that the car battery has reached full capacity.
R51	The system must stop charging a car if the booking corresponding to that car ends and the car is still plugged in.
R52	The system must automatically select the DSO that offers the best energy price if the "automatic DSO choice" is selected.

Identifier	Description
R53	If a battery was charging and reaches a full charge, the system must stop charging the battery.
R54	The system must automatically change the energy mix if the "automatic mix choice" is selected.
R55	The system must automatically notify the eMSP of the assignment of a charging column.
R56	The system must automatically notify the eMSP of the end of the car charge.
R57	The system must periodically update the database with new data coming from sensors present in charging stations.
R58	The system must allow the CPMS user to view all bookings present for any charging station.
R59	The system must allow the CPMS user to view all charging points, relative socket types, and if they are occupied or not, for any charging station.
R60	The system must allow the CPMS user to view the batteries' charge levels if present.
R61	The system must allow the CPMS user to create and delete special charge offers.
R62	The system must allow the CPMS user to view all DSO available.
R63	The system must allow the CPMS user to view all DSO prices.
R64	The system must allow the CPMS user to change the energy mix.
R65	The system must allow the CPMS user to start recharging batteries if they are present.
R66	The system must allow the CPMS user to activate and deactivate "automatic DSO choice".
R67	The system must allow the CPMS user to activate and deactivate "automatic mix choice".
R68	The system must allow the CPMS user to modify the automatic DSO payment method.

4.2 Mapping with components

Goal	Requirements	eMSP components	CPMS components
G1	R1-R14, R18-R20, R31-R35	`CPMShandler`, `DatabaseHandler`, `eRoamingHandler`, `User`, `Web server`	`APIhandler`, `DatabaseHandler`, `eRoamingHandler`
G2	R1-R17, R21-R27, R33-R37	`BookingHandler`, `CPMShandler`, `DatabaseHandler`, `Device`, `EmailHandler`, `User`, `Vehicle`, `Web server`	`APIhandler`, `Booking`, `DatabaseHandler`
G3	R15-R17, R33-R35	`DatabaseHandler`, `User`, `Vehicle`, `Web server`	`DatabaseHandler`, `ChargingColumn`, `ChargingStation`
G4	R1-R17, R28, R33-R37, R55, R56	`APIhandler`, `DatabaseHandler`, `Device`, `EmailHandler`, `User`	`DatabaseHandler`, `ChargingStation`, `eMSPhandler`
G5	R1-R17, R29, R30, R34-R37	`CPMShandler`, `DatabaseHandler`, `Device`, `EmailHandler`, `PaymentHandler`, `User`, `Web server`	`APIhandler`, `ChargingColumn`, `DatabaseHandler`, `eMSPhandler`, `PaymentHandler`
G6	R21, R25-R27, R45-R48, R57-R60	`CPMShandler`	`APIhandler`, `ChargingColumn`, `ChargingStation`, `DatabaseHandler`
G7	R45, R49-R51, R55, R56	None	`ChargingColumn`, `ChargingStation`, `DatabaseHandler`
G8	R38-R44, R52, R62, R63, R66, R68	None	`ChargingStation`, `DatabaseHandler`, `DSOhandler`, `User`, `Web server`
G9	R38-R44, R53, R54, R64, R65, R67	None	`ChargingStation`, `DatabaseHandler`, `EnergySource`, `User`, `Web server`
G10	R38-R44, R48, R61, R63	None	`DatabaseHandler`, `SpecialOffer`, `User`, `Web server`

Chapter 5

Implementation, Integration and Test Plan

5.1 Introduction

After having described all the components of the system and their interconnections, this chapter describes how these should be implemented, integrated, and tested in order to have two fully working systems: the eMSP's and the CPMS's.

In general, the method followed for building up both of the systems is to first start with a bottom-up approach to create the component that directly interacts with the database, to have a fully working data retrieval and management. Then a thread approach is used to develop all the other components in parallel (maybe even across multiple teams), dividing the work by functionality. Meanwhile, it's possible to test the correct functioning of every part of the system and to provide some betas of the software to the stakeholders[1], to let them check that all the required functionalities have been implemented according to their needs and desires.

5.1.1 Terminology

Development strategies Just to be even more clear, here we provide a brief description of some strategies nominated in this section. There are references to the components interfaces of the two systems that can be found in 2.4 Components interfaces (page 16).

- *Bottom-up approach*: the bottom-up approach consists in starting building the system from its foundations, which is made of the components in the leaves of a "uses" hierarchy. Starting from there, the strategy goes up to the root of the hierarchy.
- *Thread approach*: the thread approach focuses more on developing specific functionalities across multiple components, thus developing a single piece of all the touched components.

Testing strategies Testing software is not trivial and there are multiple methods for doing it. Every strategy has its own goals and here are reported the ones nominated in this chapter.

- *Load testing*: this kind of test helps to identify eventual memory leaks, buffer overflows, and in general any problem related to memory usage. Even though the choice of the language (or languages) to use is not enforced here, we suggest using the Rust language for avoiding these kinds of errors.
- *Performance testing*: its main purpose is to identify any bottleneck of the application, therefore affecting response times and general throughput.
- *Stress testing*: the main purpose of a stress test is to check if the system is able to gracefully adapt and recover after some failures.
- *Unit testing*: this testing is done in order to check whether the functions of the system (or groups of them) do what they are supposed to.

[1] The *stakeholders* are the ones who own shares in an enterprise.

5.2 eMSP

First, here we provide the description of how the eMSP should be implemented, integrated, and tested. The flow follows the general description provided above in the introduction.

5.2.1 Implementation plan

Step 1 (bottom-up) As stated in the introduction, the implementation should start with a bottom-up approach, by first implementing the `DatabaseHandler`, which is the component in charge of providing all the data to the above layers of the business layer and to do some processing with them. During this phase, it's also suggested to start developing the internal functions of the `PaymentHandler` and the `EmailHandler` (in this last case, like the methods for connecting to the SMTP server and general methods for building up and sending the emails).

Step 2 (thread) After having fully developed the `DatabaseHandler`, the rest of the components should be developed in parallel following a thread approach, first focusing on the main functionalities that the eMSP should provide, and then moving to the other ones (like the password reset, the account activation procedure...). Together with this, the NGINX configuration should be created and updated in order to allow it to redirect the traffic accordingly to the various components.

Step 3 (frontend) While developing all the components presented in the previous step, the frontend can also be developed, possibly following the development of the various functionalities, but it's not mandatory. In case of problems, it's possible to easily change the frontend JavaScript querying and displaying functions.

5.2.2 Integration plan

The integration plan roughly follows the implementation plan presented above.

Components of Step 1 (bottom-up) These few components should be integrated with the other ones following the developed threads.

Components of Step 2 (thread) Since the main part of the development is done following a thread-based approach, the integration of many of the components is done directly while developing them. Integrating the various threads, instead, should be done whenever any thread is completed.

Components of Step 3 (frontend) The frontend should be integrated with the underlying components as soon as the functionalities have been developed, in order to also provide some betas of the software to the stakeholders.

5.2.3 Test plan

The test plan for the system is slightly more complex than what was done in with the previous plans.

For every component or thread, after it has been developed, a unit test should be performed in order to immediately spot any possible bug in the implementation.

Moreover, for any completed thread, some other tests should be performed in order to discover some other more subtle bugs in the software and to possibly optimize the performance of the whole system:

- *Performance testing*: this is done for spotting the bottlenecks.
- *Load testing*: this is done to identify memory leaks, overflows or other memory-related problems.
- *Stress testing*: its purpose is instead to recover gracefully after failures.

Once any beta version of the software is available, it's possible to perform some acceptance testing which should be conducted by different people, maybe also the stakeholders, for testing the usability of the system.

5.3 CPMS

We'll now describe how the CPMS system should be implemented, integrated, and tested. The flow follows the general description provided above in the introduction.

5.3.1 Implementation plan

Step 1 (bottom-up) As stated in the introduction, the implementation should start with a bottom-up approach, by first implementing the `DatabaseHandler`, which is the component in charge of providing all the data to the above layers of the business layer and to do some processing with them. After the `DatabaseHandler` is completed, The `ChargingStation`, `ChargingColumn`, and `EnergySource` should be implemented, and possibly be connected to their physical counterparts or to similar components.

Step 2 (thread) After having fully developed the components mentioned in step one, the rest of the components should be developed in parallel following a thread approach, first focusing on the main functionalities that the CPMS should provide, like the booking mechanism. After those, the focus should be on the components that interact with, or offer, external interfaces. Together with this, the NGINX configuration should be created and updated in order to allow it to redirect the traffic accordingly to the various components.

Step 3 (frontend) While developing all the components presented in the previous steps, the frontend can also be developed, possibly following the development of the various functionalities, but it's not mandatory. In case of problems, it's possible to easily change the frontend JavaScript querying and displaying functions.

5.3.2 Integration plan

The integration plan roughly follows the eMSP implementation plan.

Components of Step 1 (bottom-up) The components implemented in the bottom-up approach should all be integrated as they are developed, making use of mockup physical counterparts to ensure their correct functioning.

Components of Step 2 (thread) Since the main part of the development is done following a thread-based approach, the integration of many of the components is done directly while developing them. Integrating the various threads, instead, should be done whenever any thread is completed.

Components of Step 3 (frontend) The frontend should be integrated with the underlying components as soon as the functionalities have been developed, in order to also provide some betas of the software to the stakeholders.

5.3.3 Test plan

The test plan for the CPMS system should roughly follow the same test plan for the eMSP system.

For every component or thread, after it has been developed, a unit test should be performed in order to immediately spot any possible bug in the implementation.

Tests for the `ChargingColumn`, the `ChargingStation`, and the `EnergySource` should be conducted on their physical counterparts as well as on the components.
Moreover, for any completed thread, the same tests specified for the eMSP system should be performed in order to discover some other more subtle bugs in the software and to possibly optimize the performance of the whole system:

- *Performance testing*: this is done for spotting the bottlenecks.
- *Load testing*: this is done to identify memory leaks, overflows, or other memory-related problems.
- *Stress testing*: its purpose is instead to recover gracefully after failures.

Once any beta version of the software is available, it's possible to perform some acceptance testing which should be conducted by different people, maybe also the stakeholders, for testing the usability of the system. Particular attention should be paid to how the physical elements behave when the system is in place, and tests should be done while consulting specialists in the electronic components to ensure the success of the system.

Chapter 6

Conclusions

6.1 Final thoughts

This document was redacted following the guidelines for the project (*eMall - e-Mobility for All project*). Discrepancies between our document and the guidelines are to be considered our hypothesis on how the two systems should behave or be modeled.

Also, these systems have been designed having in mind possible future expansions. Thus, anyone who wants to expand this project is welcome and encouraged to do so. For example, one possibility is to set a limit after which the user is no more allowed to book charges until s/he decides to pay the previous ones.

6.2 Credits

For writing this document, we used different pieces of software and materials from the Internet. This is a rather comprehensive list of them:

- *LaTeX* (with packages) and *Visual Studio Code* (with plugins) for writing the document.
- *Git*, *GitHub* and *Notion* for keeping things organized.
- *draw.io* and *PlantUML* for creating the diagrams.
- *Google Fonts* (icon library), *Inkscape*, *OpenStreetMap*, *Penpot* (with designs from ApeWTF and Google) and *Wikipedia* for drawing the mockups.

6.3 Effort

Task	Riccardo Motta	Pierluigi Negro
Writing chapter 1	3h	2h
Writing chapter 2	4h	4h
Drawing overview, composition diagrams, component views and deployment view	6h	7h
Drawing runtime views and components interfaces	4h	8h
Writing chapter 3	2h	1h
Designing the mockups	20h	18h
Creating the pages connections diagrams	2h	1h
Mapping components and requirements and writing chapter 4	2h	2h
Writing chapter 5	3h	1h
Other related work	5h	4h
Updating document for Version 1.1	0.5h	0.5h
Total	**51.5h**	**48.5h**