POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

DD - Software Engineering 2

Computer Science and Engineering

Authors:
Emilio Corigliano (10627041)
Federico Mandelli (10611353)
Matteo Pignataro (10667498)

Advisor: Prof. Matteo Camilli
Co-advisors: Prof.ssa Elisabetta Di Nitto, Prof. Matteo Giovanni Rossi
Academic Year: 2022-23

Contents

1. Introduction

1.1. Purpose

Due to the recent increase of effort in the battle against climate change, electric vehicles are slowly becoming the new technology for private transport that people use everyday. To sustain this type of strategy, we need to develop a clever and capillary charging system.
e-Mobility for All (eMall) is an e-Mobility Service Provider (eMSP) that aims to help the final users dealing with their charging needs by informing the users about the nearby charging stations, their cost, their environmental friendliness and any special offer that they might have. It will allow the users to book, cancel and pay for a charge and it will notify them when the charging process is terminated.
With the integration of the user's calendar, the system will also suggest the best moment in the schedule to charge the vehicle. To have a fully integrated system, all the Charging Point Operators (CPOs) will have a technological support called Charge Point Management System (CPMS) to interface the service with the physical charging stations and to manage all the energy sources like batteries and Distribution System Operators (DSOs).

1.2. Scope

The document focuses on the design aspects of the eMall and CPMS systems and illustrates the architectural choices behind the implementation of them. Follows a summary about the main design topics of each system:

- **eMall**: It is a 3-tier architecture where the business logic and the data logic are separated. The third tier is the client that needs to be as **FAT** as possible. The whole architectural pattern is the Model View Controller (MVC) due to its flexibility and scalability;
- **CPMS**: It is a client-server 2-tier architecture due to the low user base and all in one system. The architectural pattern is the MVC.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Acronyms

eMall	e-Mobility for All
eMSP	e-Mobility Service Provider
CPO	Charging Point Operator
CPMS	Charge Point Management System
DSO	Distribution System Operator
API	Application Programming Interface
HTTPS	HyperText Transfer Protocol Secure
SPOF	Single Point Of Failure
GDPR	General Data Protection Regulation
SoC	State of Charge
RASD	Requirement Analysis and Specification Document
MVC	Model View Controller
DAO	Data Access Object
IBAN	International Bank Account Number
VPN	Virtual Private Network
PAN	Personal Area Network
UI	User Interface
DBMS	Data Base Managerment System

1.4. Reference documents

- **Requirement Analysis and Specification Document (RASD):** https://github.com/federico-mandelli/CoriglianoMandelliPignataro

1.5. Document structure

The document is divided in seven main sections:

- **Introduction**: After a brief purpose section, it illustrates the summary of main architectural choices and collects all the acronyms and definitions present on the document;
- **Architectural Design**: At first it introduces the major interfaces of the system, and then it illustrates the collection of components, interfaces and their dynamic usage. The document includes also infrastructure details at system level;
- **User Interface Design**: This section includes User Interface (UI) mockups and references the relation between them and the previously mentioned interfaces and components;
- **Requirements traceability**: Maps the requirements collected inside the RASD and the corresponding design elements;
- **Implementation, Integration and Test Plan**: Describes the plans to follow to implement, integrate and test the whole system and its components;
- **Effort Spent**: Summarizes the total hours spent on the document formalization;
- **References**: Summarizes all the reference documents that we used during the description.

2. Architectural design

2.1. Overview

The architecture of the system is composed of two main components: the eMSP and the CPMS.

2.1.1. eMSP overview

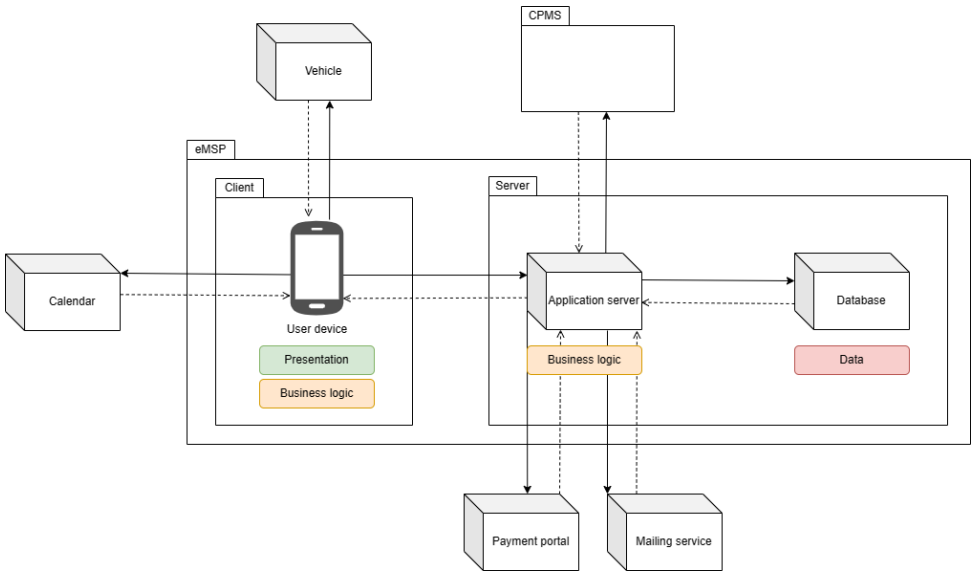

Figure 1: eMSP architectural overview

The eMSP is a **three-tier** architecture with **fat-clients** as seen in Figure 1. This architecture is chosen for different reasons:

- It makes the system more scalable;
- It allows the separation between business logic and data so that we can apply different levels of dependability to different decoupled systems and we can manage how data is accessed in a more granular way;
- A dedicated and optimized infrastructure (Data Base Managerment System (DBMS)) allows to handle a lot of data in the system (such as booked charges, all the infos about CPOs, . . .);
- The database can only be accessible by the middleware, constituting an additional layer of security;
- With fat-clients the number of messages transmitted are fewer and lighter: an initial elaboration can be done on the smart devices of the user without sending a lot of raw data to the remote application server. The local on-the-edge elaboration is not considered as a problem given the computational power of smartphones.

The software pattern applied for this architecture is the **MVC pattern**. This is best suited for our application because we want the components to be as modular, flexible and scalable as possible in order to simplify their distribution.

The following paragraphs will describe the principal components of this pattern and their architectural distribution.

Model The model is the logical representation of persistent data. This is stored in a database system and can only be directly accessed by the server application.

View The view logic is completely delegated to the client application, representing in different ways the data retrieved from the model (e.g. the charging stations sorting based on the user selected preferences).

Controller The controller has to manage all the client requests, interacting with the model, modifying it and returning to the view the changed/retrieved data. For the most part, the business logic is in the server application. Retrieving and processing the vehicle and calendar data(in the case of a request for smart suggestions of the application) is handled by the client application to be as light as possible.

2.1.2. CPMS overview

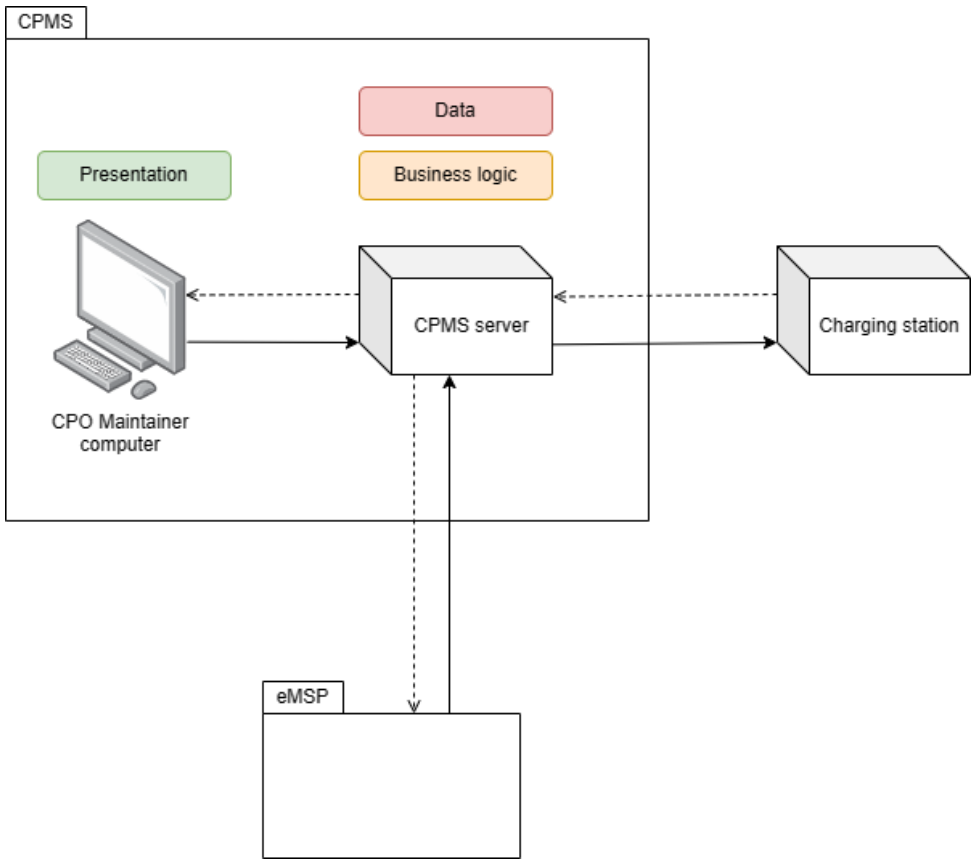

Figure 2: CPMS architectural overview

The CPMS follows the **two-tier** pattern with **thin-clients** as seen in Figure 2. This architecture is chosen for different reasons:

- The system is simpler to implement than the three-tier architecture;

- The system shouldn't handle so much data, thus it isn't necessary to have a dedicated architecture for the data layer;
- Clients are thin because they only have to view infos about charging stations and can send simple requests (for example *use DSO X for charging station Y, set revenue percentage to Z*, etc.).

In this system the MVC pattern is used. In this case, the client has only the view logic and can send simple requests. All the management of the DSOs and activation/deactivation of battery charging processes are handled by the CPMS server. The CPMS server deals also with charging requests from eMSPs, to which it responds with the elaborated data.

2.2. Component view

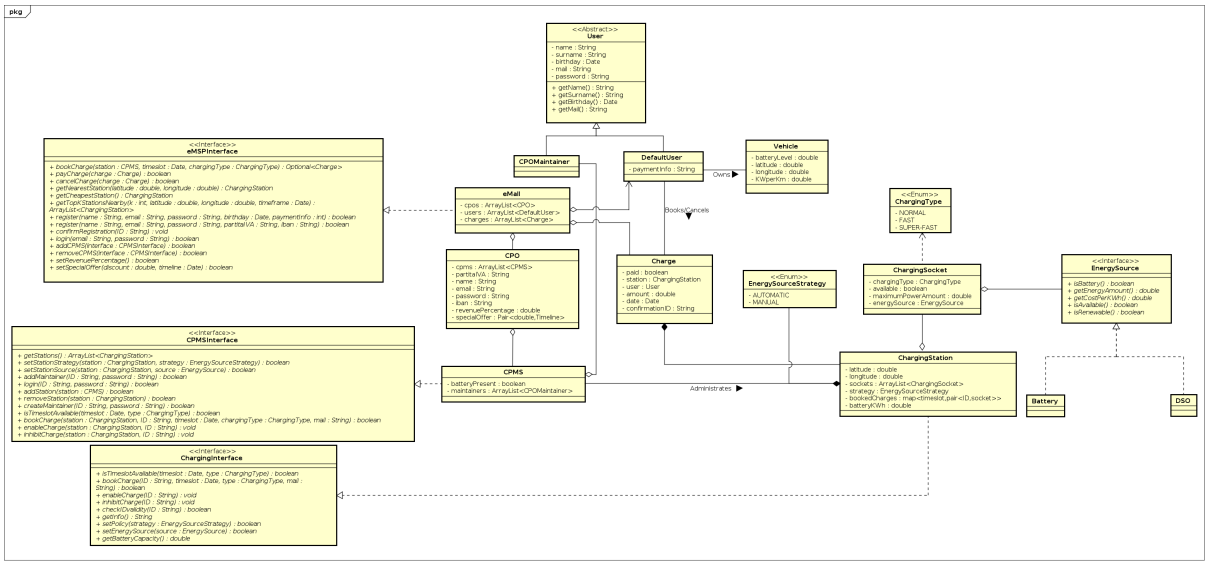

Figure 3: Class diagram

In the class diagram illustrated in Figure 3 a model (not functional) view of the system is represented. The eMall and CPMS interfaces show what the two systems are expected to implement, whereas for the ChargingInterface and EnergySource it is assumed an already existing implementation.

2.2.1. Component diagrams

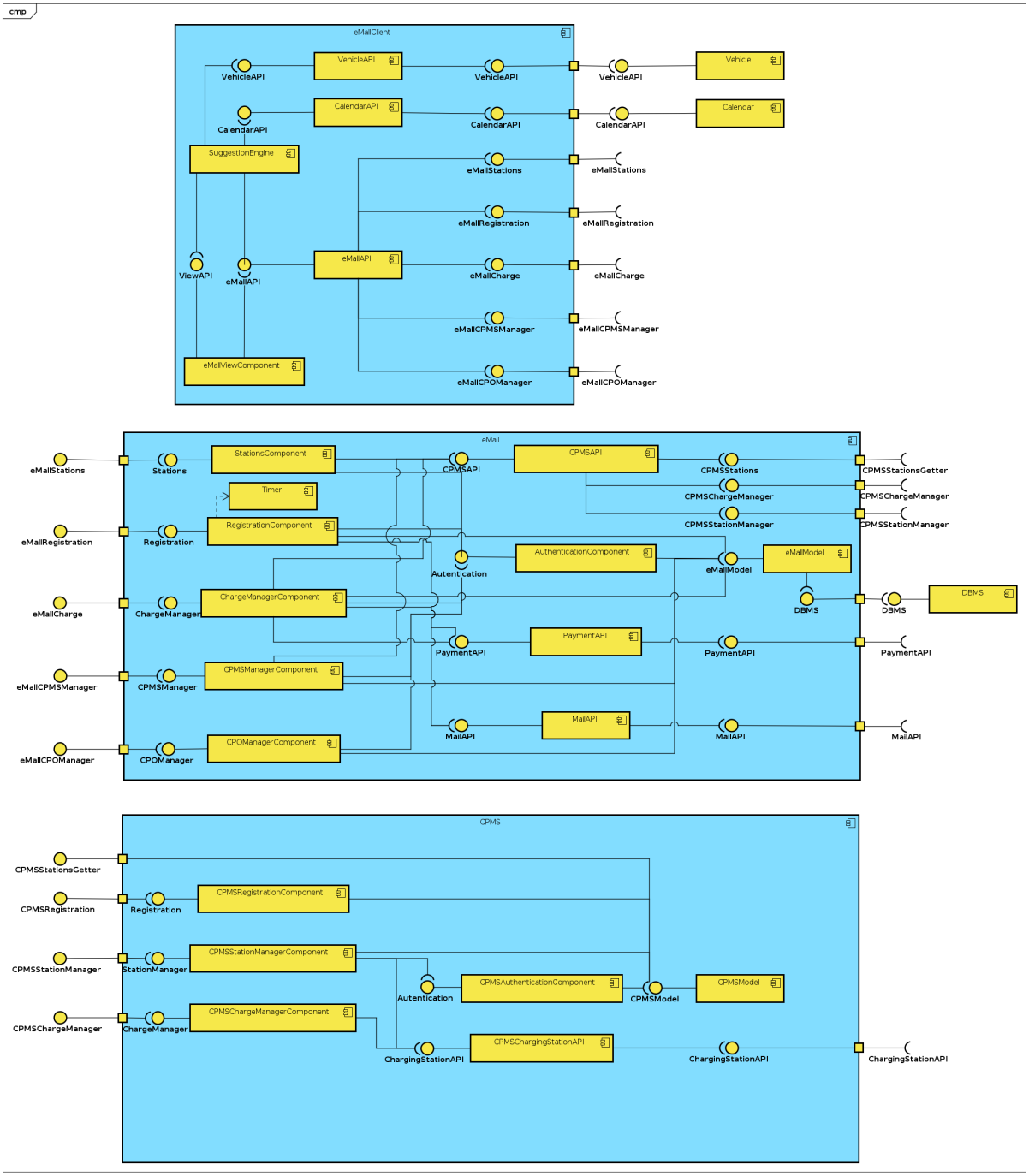

Figure 4: eMall component diagram

The two systems (eMall and CPMS) implement different interfaces and shows the *Controller* and *Model* parts of the MVC pattern.

eMallClient The main components for the eMallClient system are:

CalendarAPI It is the component responsible for interfacing with a general web calendar in order to get the user's appointments;

VehicleAPI It is the component responsible for interfacing with a general electric vehicle in order to get vehicle infos;

SuggestionEngine It is the component responsible for retrieving and parsing the data for a charge suggestion. It is the only active component of the eMall system;

eMallAPI It is the component in charge of remapping the client requests to the eMall interface;

eMallViewComponent It is the component in charge of handling the view logic and interfacing with the user through the UI;

eMall The main components for the eMall system are:

StationsComponent It is the component responsible for handling all the stations research requests (like *getNearestStations*, *getCheapestStation*, *getTopKStationsNearby*);

RegistrationComponent It is the component responsible for handling the registration details along with the parameters checking during the registration phase;

Timer It is the component responsible for handling the registration verification timeout;

ChargeManagerComponent It is the component responsible for handling all the charge booking/payment/cancellation processes.
It interfaces with the payment Application Programming Interface (API) and with the CPMS API;

CPMSManagerComponent It is the component responsible for handling the CPMS operations from the CPOs users;

CPOManagerComponent It is the component responsible for handling the CPOs requests (like *SetRevenuePercentage* and *setSpecialOffer*);

CPMS API It is the component responsible for interfacing the eMall with the CPMS interfaces;

AuthenticationComponent It is the component responsible for authorizing every operation. In this component it is implemented part of the controller (of the MVC pattern) to check if an operation is legit for the account type;

Mail API It is the component responsible for sending the feedback emails to the user every time an important operation is done;

Payment API It is the component responsible for performing financial operations/verifications;

eMall Model It is the core model component. It interfaces the DBMS with the rest of system components.

CPMS The main components for the CPMS system are:

CPMSRegistrationComponent It is the component responsible for handling the registration details of new CPO maintainers;

CPMSStationManagerComponent It is the component responsible for handling all the station managing actions from a CPO maintainer (*setStationStrategy*, *setStationSource*..);

CPMSChargeManagerComponent It is the component responsible for handling all the actions which involve book/enable/inhibit a charge. The eMall is the only component that utilizes this interface;

CPMSAuthenticationComponent It is the component responsible for authorizing every operation. In this component it is implemented part of the controller (of the MVC pattern) to check if an operation is legit;

CPMSChargingStation API It is the component that interfaces the CPMS system with the external Charging Stations;

CPMSModel It is the core component. It stores all the CPMS data and interfaces them with the other system components.

In the component diagrams, there are components (*PaymentAPI*, *CPMSAPI*, *MailAPI*, *CPMSChargingStationAPI*) that could be bypassed, letting the main components interface the external APIs. These components are integrated because they offer the possibility to contain the external interface standards in one single component. This creates the possibility for developers to create an internal interfacing standard different from the external ones.

2.3. Deployment view

The general idea of the deployment view is to show all the main architectural components with the distribution of the artifacts (applications, firewall rule tables, data...).

2.3.1. eMSP

The system has been designed to provide security policies and avoid SPOFs, to support an high availability service. The system's components are:

- Mobile device: It is the device used by users or CPOs to interact with the eMall system. It is provided with internet connection, bluetooth peripheral and the eMall client application installed. This application must be available for both Android and iOS mobile operative systems in order to be compatible with most devices. It uses HyperText Transfer Protocol Secure (HTTPS) to communicate with the server and the calendar. HTTPS provides different security layers and it is compliant with the General Data Protection Regulation (GDPR) law. The device uses bluetooth to create a Personal Area Network (PAN) with the user's vehicle to retrieve its data.
- Application server: The physical server that runs the eMSP application, which receives all the requests from the clients and hosts the business logic. There are two redundant application servers in order to increase reliability and availability avoiding down-times while maintaining the eMall service. Ubuntu is chosen as the server's Operative System because; it is open source and the most used linux distribution according to [The most used linux distro]. Thus support and compatibility is assured

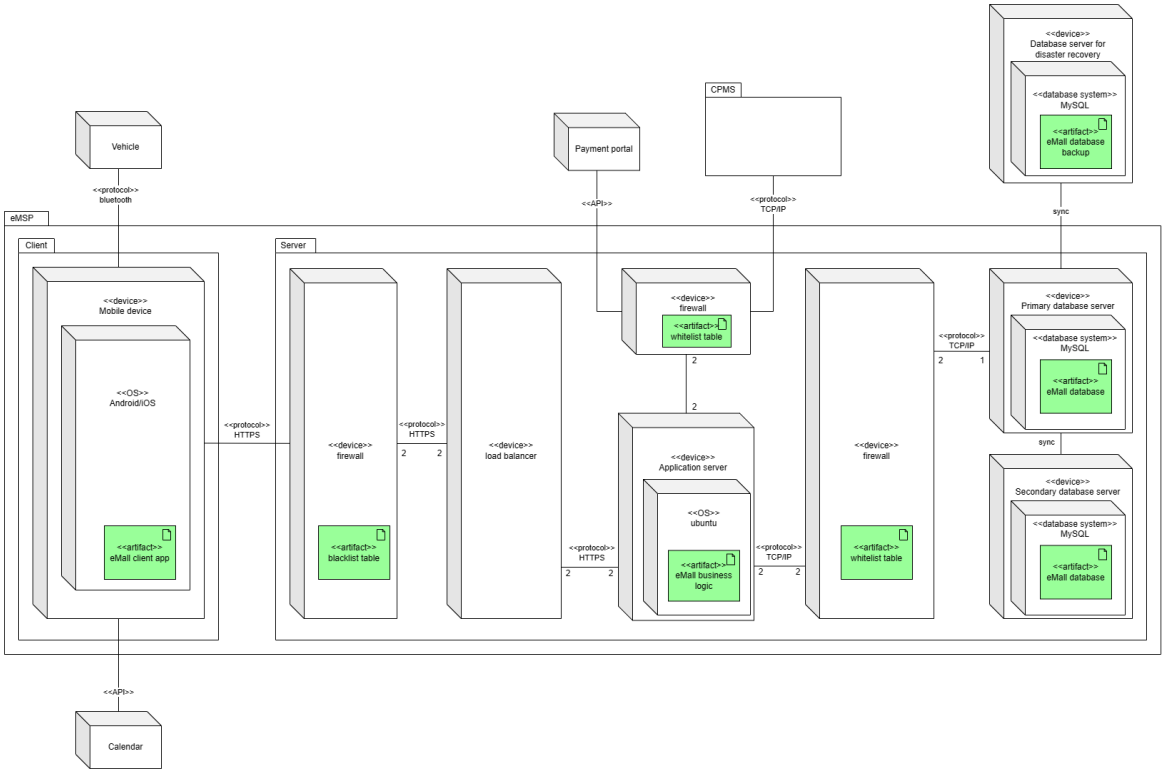

Figure 5: eMall deployment view diagram

by its widespread usage.

- Load balancer: balances the load of the requests among the active application servers and it handles server failures. This system provides failover as described in [Redundancy in load balancers].
- Firewall: Each firewall in the server is used to isolate different zones. A first firewall interfaces the external world to the load balancer in order to filter dangerous requests. A blacklist rule table is implemented due to the difficulties with an allowlist. For the internal firewall an allowlist table is implemented to enable only application servers communications with databases. This system provides failover as described in [Redundancy in firewalls].
- Database system: A redundant database system is deployed. The secondary database should always be synchronized with the primary one in order to be ready for an eventual substitution in case of failure as described in [Redundancy in Databases]. A disaster recovery database is also necessary. MySQL DBMS has been chosen because it is one of the most used as stated in [The most used relational DBMS].

2.3.2. CPMS

In the CPMS a Virtual Private Network (VPN) is implemented so that only CPOmaintainers in the same VPN have the ability to perform actions. The only non-VPN connections come from eMSPs. Redundancy is implemented in order to avoid SPOFs. The system components are:

- CPO maintainer computer: This device is connected through a VPN to a VPN server.

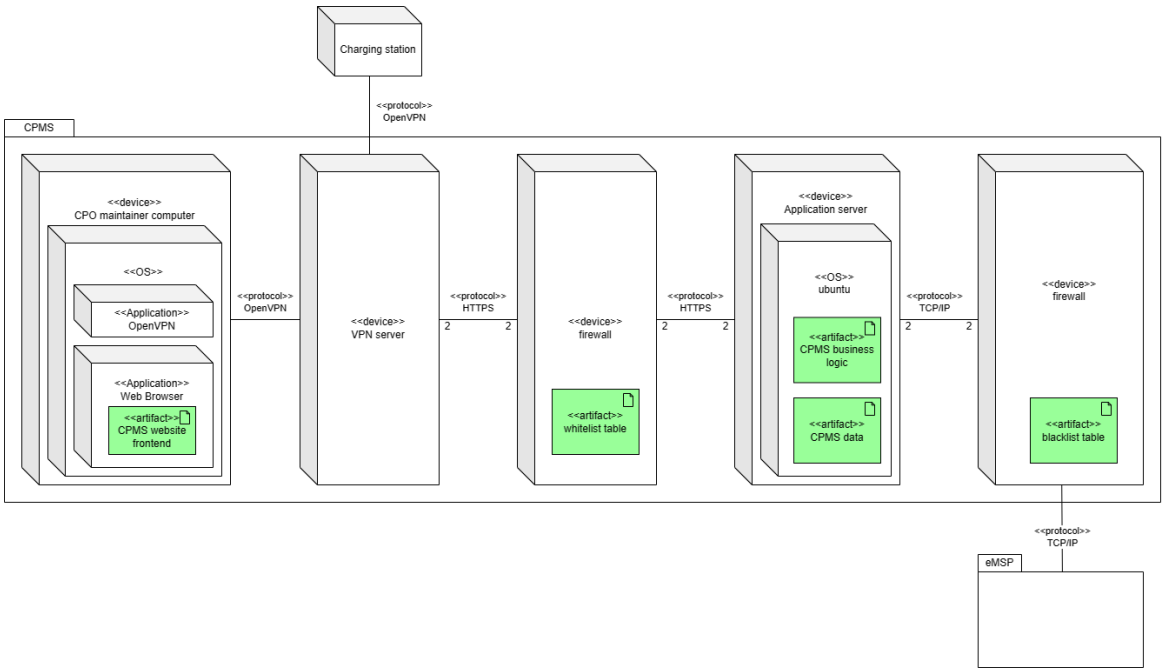

Figure 6: CPMS deployment view diagram

It can connect to the CPMS site with a web browser on top of an Operative System. This is the thin client, the CPMS' website front-end just handles the view and sends requests to the application server.

- VPN server: It allows the CPO maintainer computers, the charging stations and the application server to share the same private network to simplify the communication and improve the overall security. The OpenVPN protocol [OpenVPN official website] has been chosen because it is an open source project. It is considered a secure protocol and it is widely used in many different applications. This system provides failover as described in [Redundancy in VPN servers].
- Application server: This is the component that handles all the requests to the system and redirects them to the charging stations if needed. This system is redundant in order to assure availability during maintenance. Ubuntu is chosen as the server's Operative System because; it is open source and the most used linux distribution according to [The most used linux distro]. Thus support and compatibility is assured by its widespread usage.
- Firewalls: They encapsulate the application server to enforce security policies. The system provides failover as described in [Redundancy in firewalls].

2.4. Runtime view

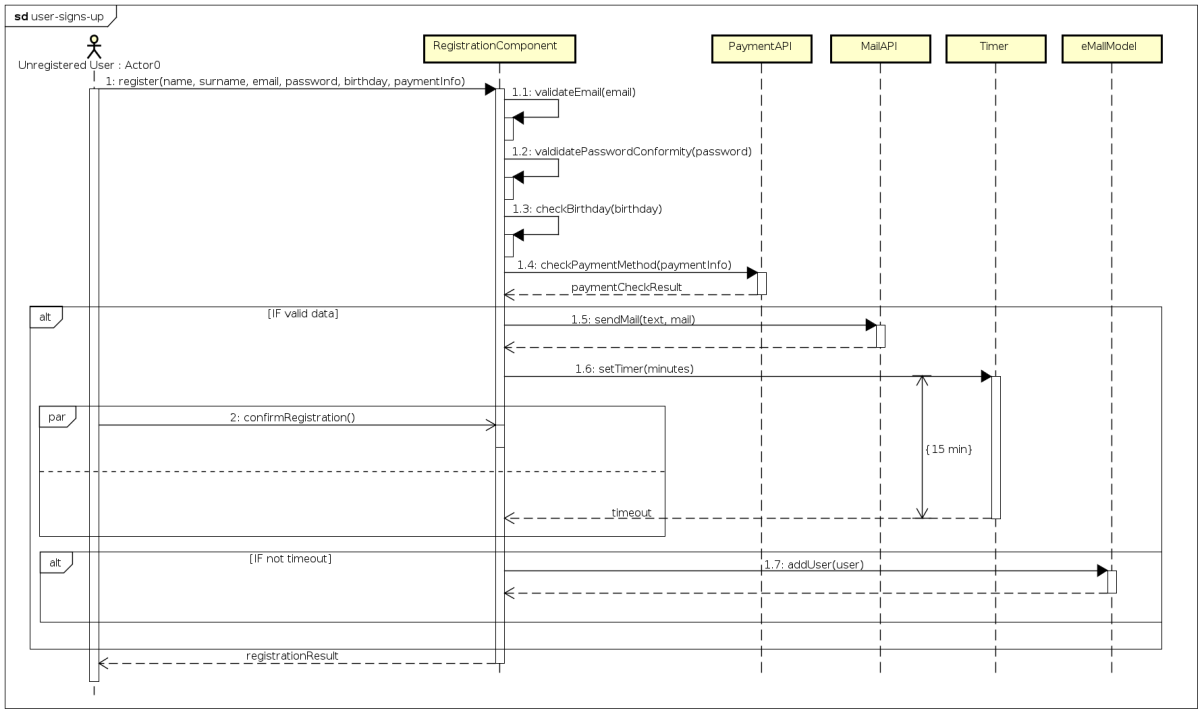

Figure 7: User signs up

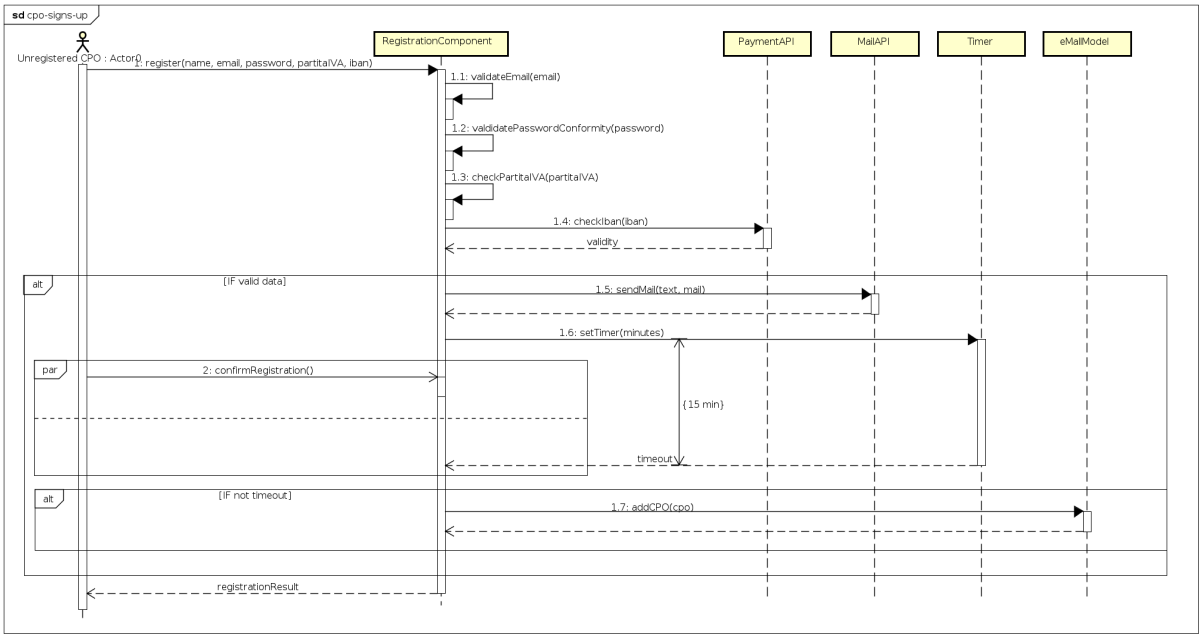

Figure 8: CPO signs up

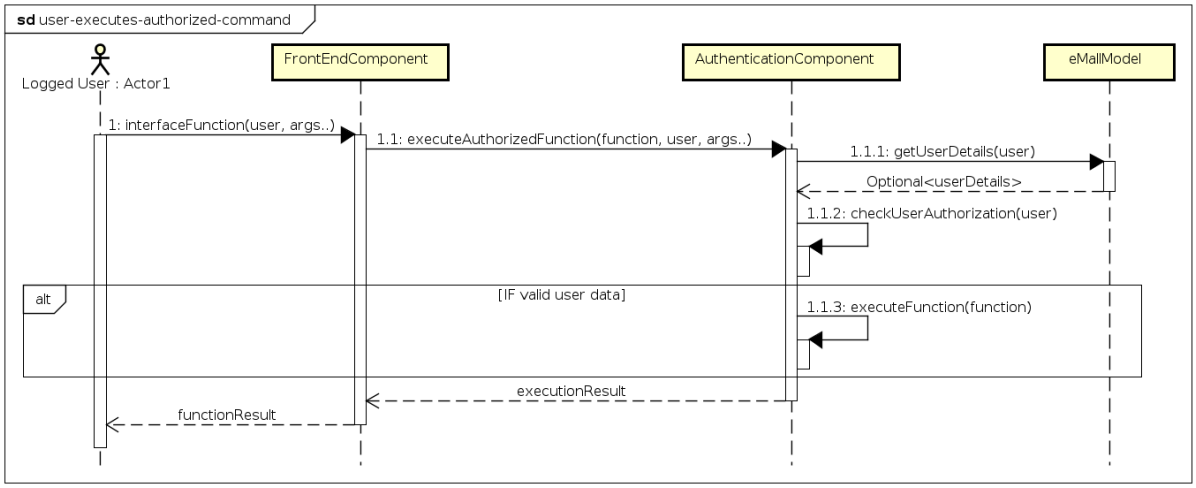

Figure 9: User executes authorized command

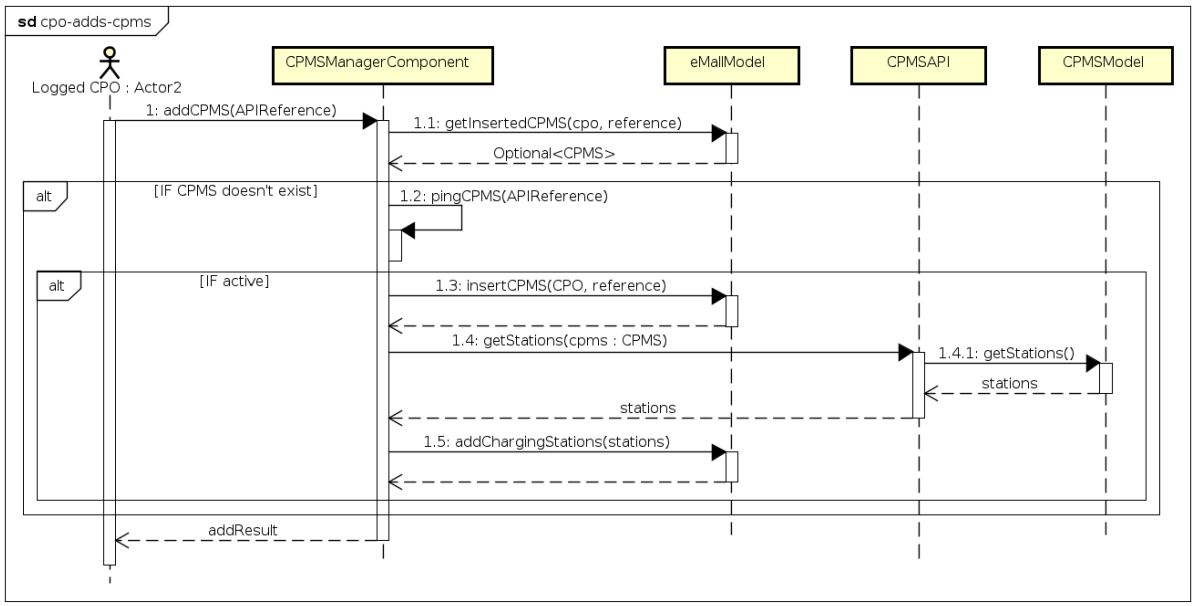

Figure 10: CPO adds CPMS into eMall

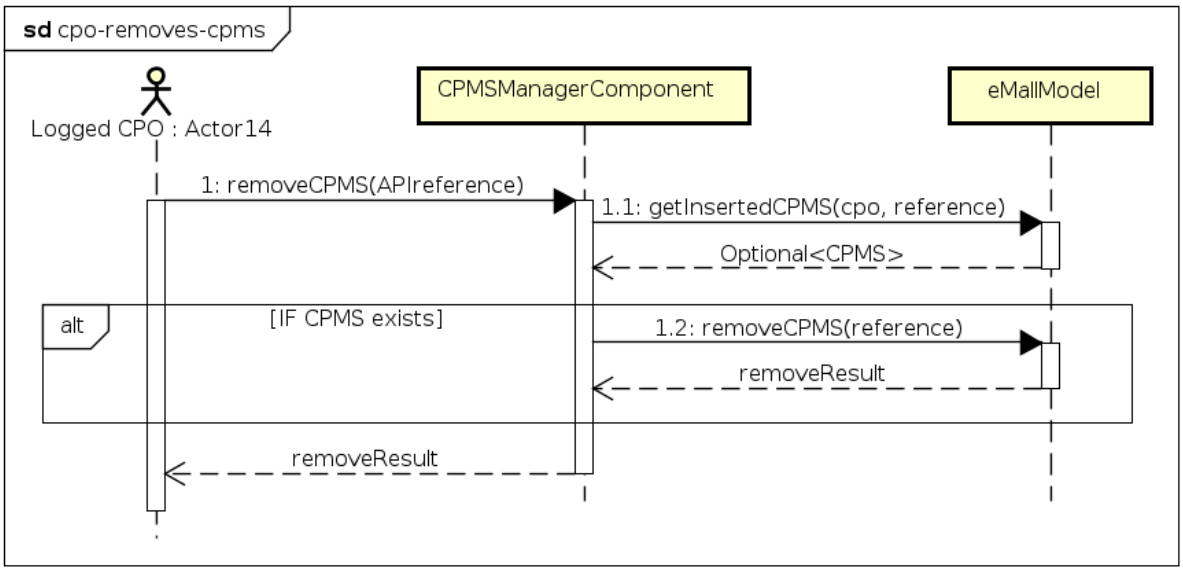

Figure 11: CPO removes CPMS from eMall

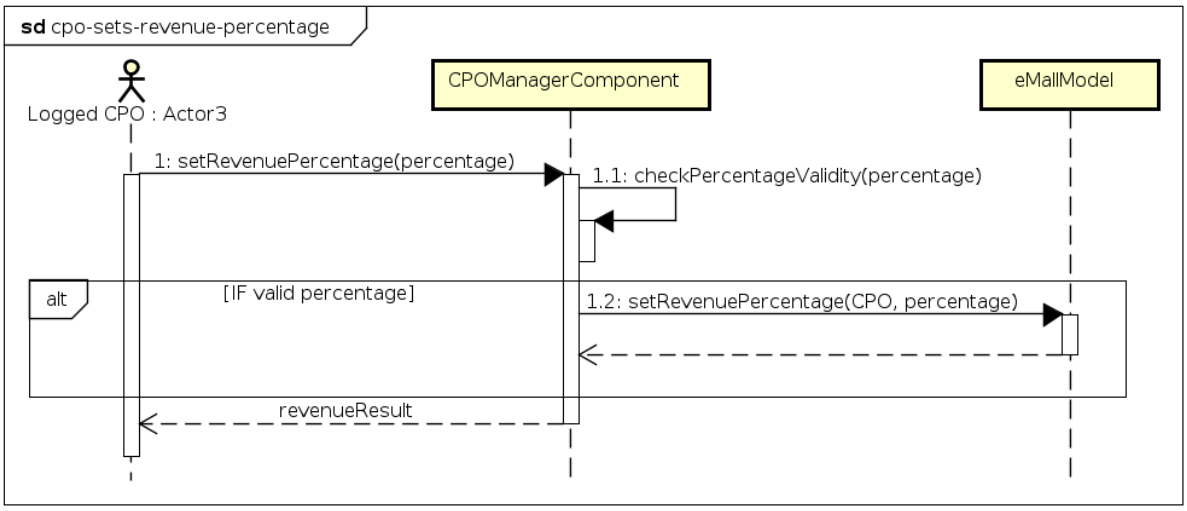

Figure 12: CPO sets the revenue percentage

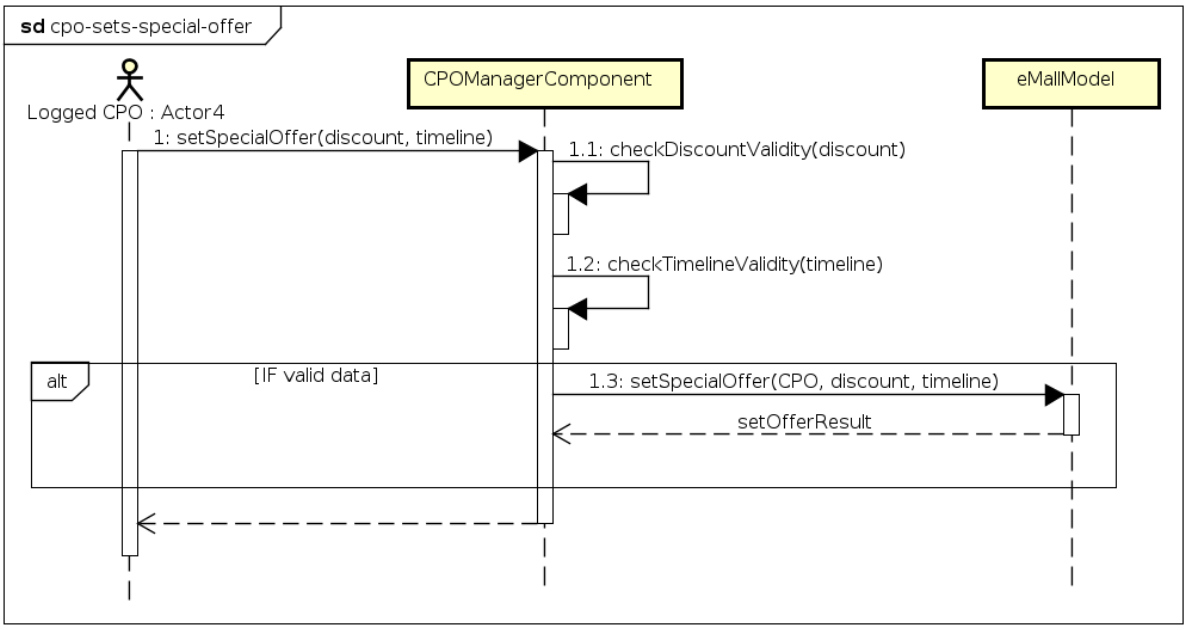

Figure 13: CPO sets a special offer

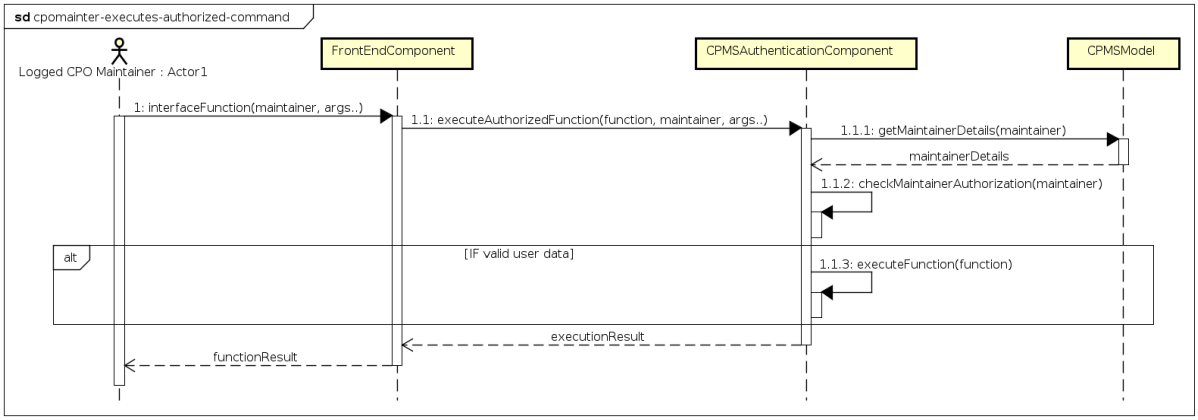

Figure 14: CPOmaintainer executes authorized command

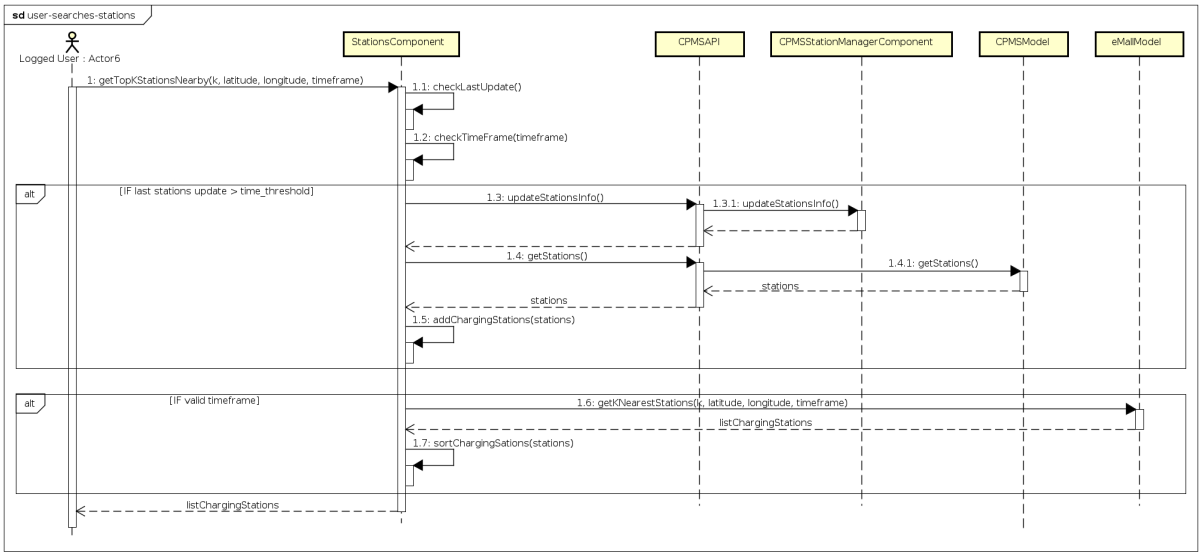

Figure 15: Get the nearby charging stations

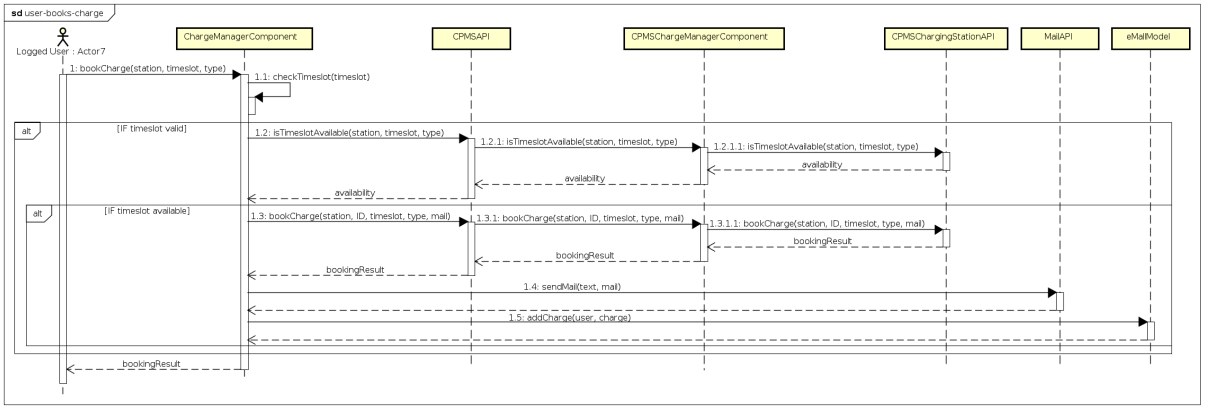

Figure 16: Book a charge sequence

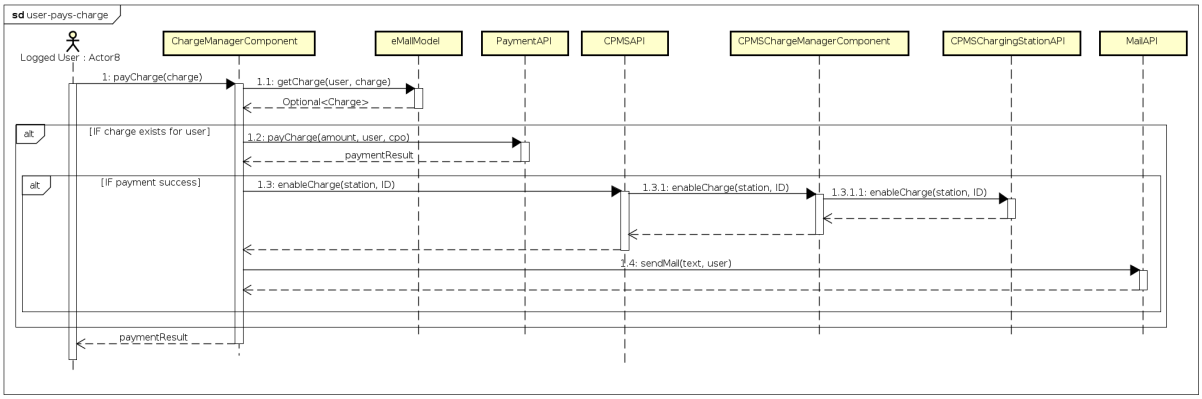

Figure 17: Pay a charge sequence

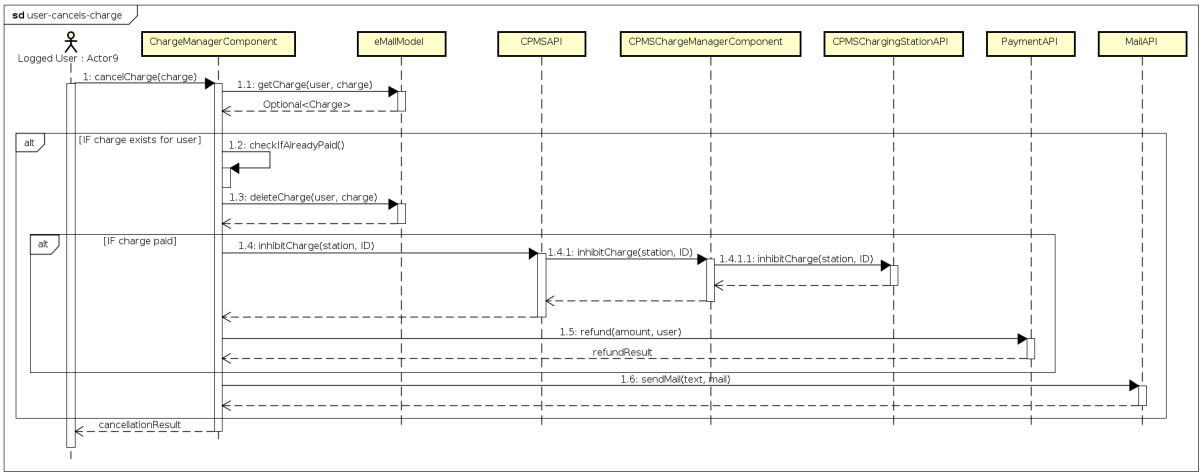

Figure 18: Cancel a charge sequence

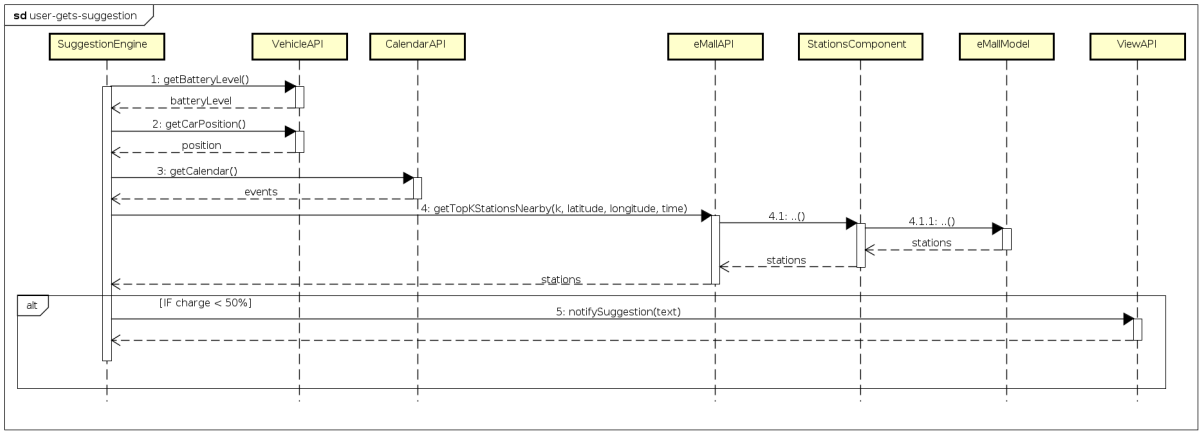

Figure 19: Get a suggestion sequence

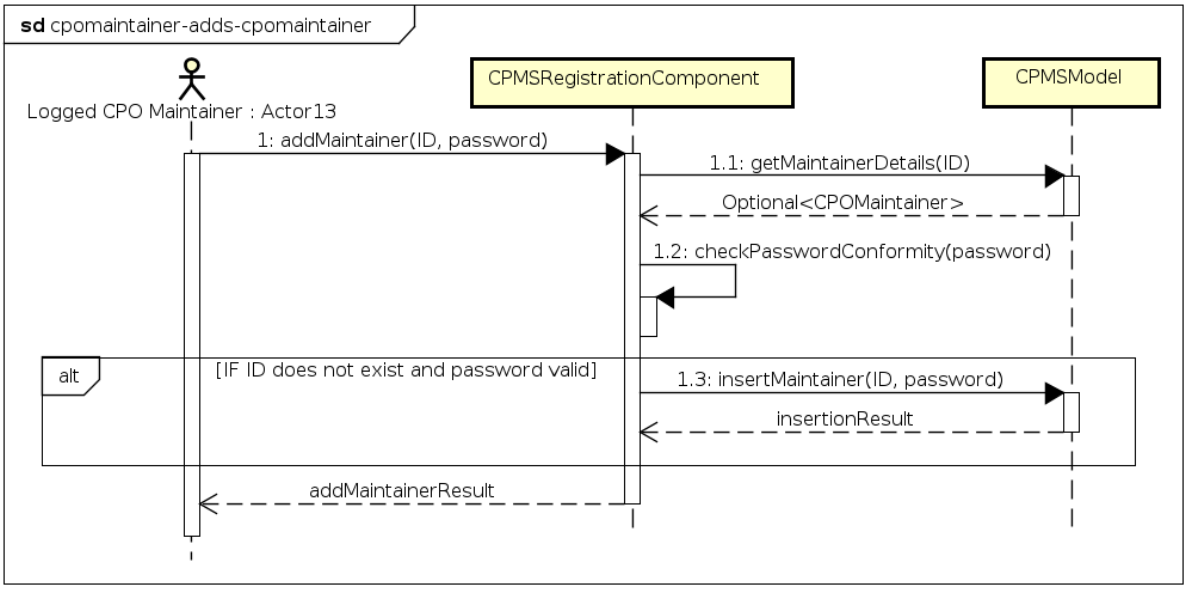

Figure 20: CPOmaintainer adds CPOmaintainer to CPMS

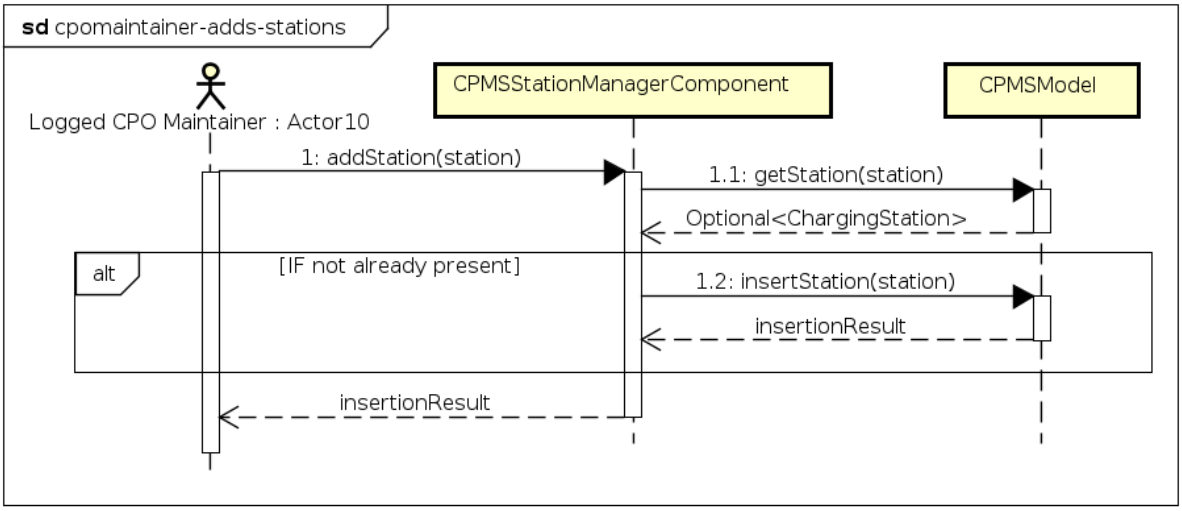

Figure 21: CPOmaintainer adds stations to CPMS

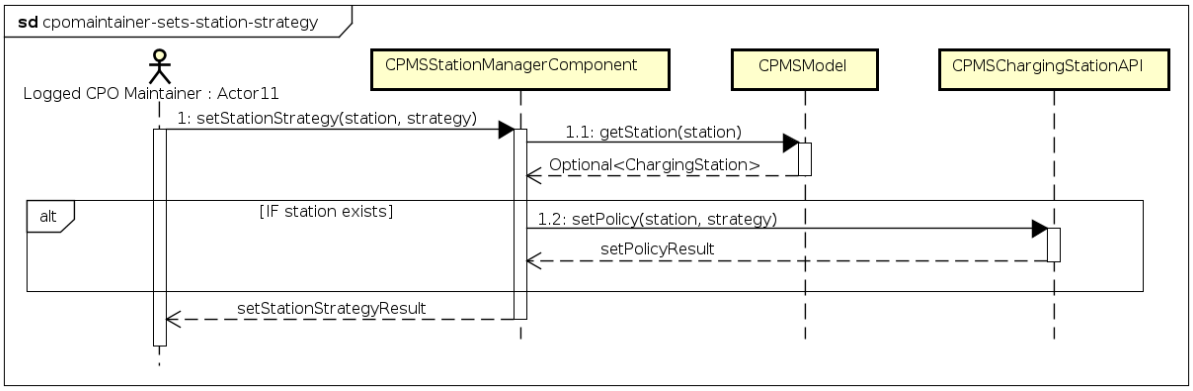

Figure 22: CPOmaintainer sets station strategy in CPMS

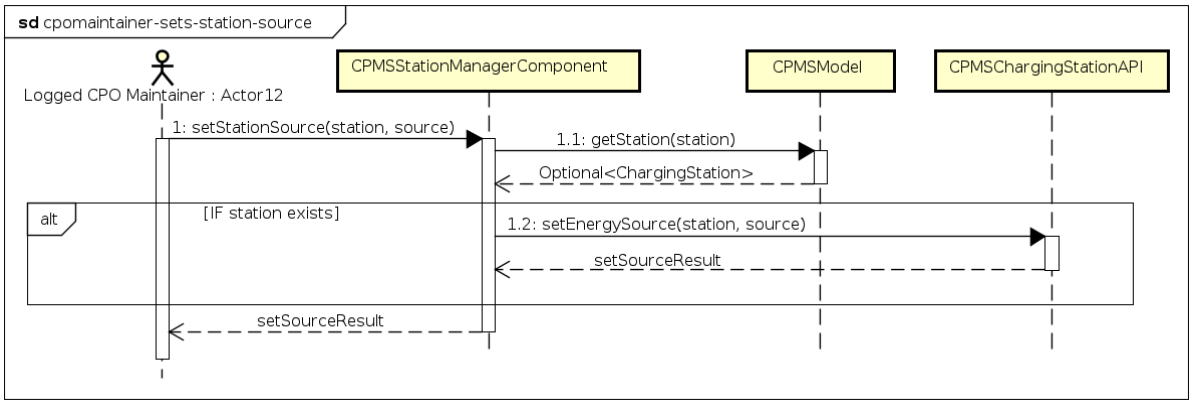

Figure 23: CPOmaintainer sets station source in CPMS

There are some particular design decisions that need to be clarified:

- **Login sequences:** In Figure 9 and Figure 14 it is illustrated the way the system authorizes CPOs and users. It filters authorized commands using functions (like [The command pattern]). The component that filters the requests is the AuthorizationComponent (present in both eMall and CPMS systems).
  When the client utilizes the interface, automatically it has to send also his credentials; these are then verified by the AuthorizationComponent and, if correct it executes the corresponding API function. This pattern allows the system to decentralize the authorization verification from the API code and allows every function that needs this data to have access to the client's account information;
- **Asynchronous messages:** During the inter-system communications (like eMall -> CPMS or CPMS -> ChargingStation) it would be useful to implement asynchronous communications to avoid blocking situations. However in the sequence diagrams there aren't any because a better solution (compromise between having a completely asynchronous communication and having feedback from the interface) is to implement a timeout timer to avoid a deadlock. This information is not shown in the sequence diagrams due to its verbosity of notation.

2.5. Component interfaces

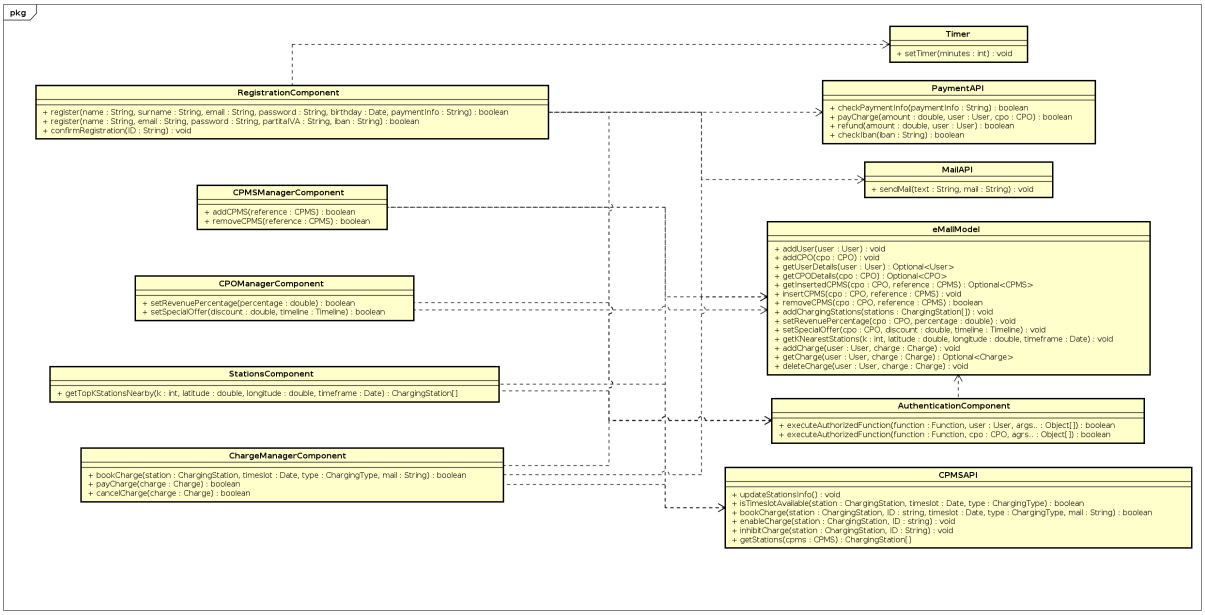

Figure 24: eMall components interface

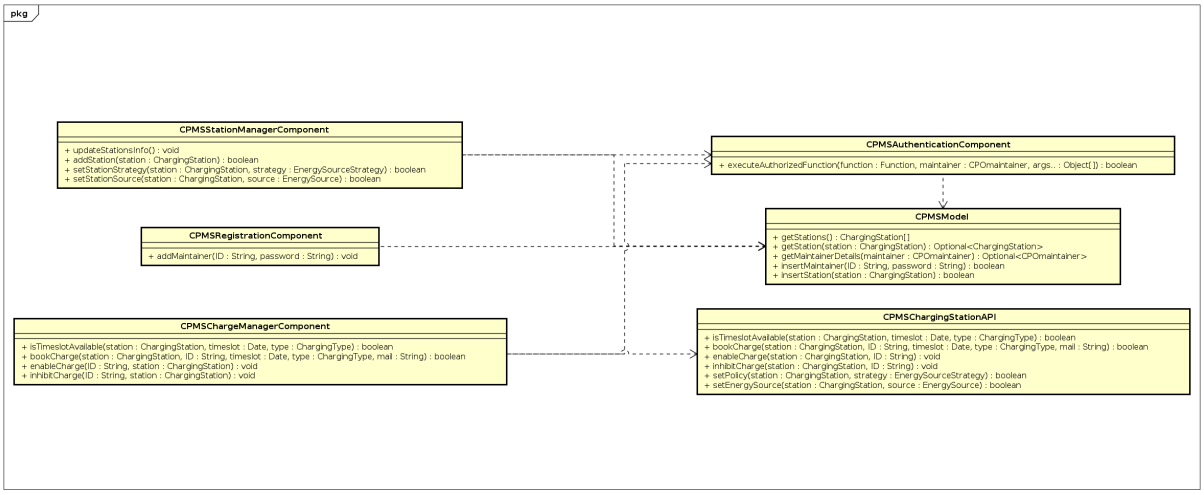

Figure 25: CPMS components interface

The eMall client interface diagram is not present due to the absence of concrete exposed interfaces. Here follows a brief clarification about the client components' interactions:

- **Vehicle and Calendar APIs:** allow the local system to interface itself with the car and the user's calendar if any. They both serve the SuggestionEngine component which is a thread that continues checking the need of a recharge;
- **eMall API:** allows the client to interact with the business logic layer of the system. It is interrogated by the suggestion engine and the view component;
- **eMallViewComponent:** interacts with the suggestion engine and the eMall API to visualize the application (it is the view in MVC pattern).

The main aspect that makes this subsystem a fat client is that the SuggestionEngine continues interrogating the eMall, the vehicle and the calendar APIs in order to proactively suggest a charge to the user.

2.6. Selected architectural styles and patterns

- Bridge pattern [The bridge pattern]: It consists in decoupling completely two systems by having two interfaces. In Figure 26 there is a simplification of the architecture in order to highlight the usage of this pattern. Instead of having a complex system with the composition of an eMSP and a CPMS we exploit their interfaces in order to implement them in a decoupled way. The eMSP implementation (i.e. eMall) interacts with a CPMS interface. This pattern allows an implementation-independent interface.

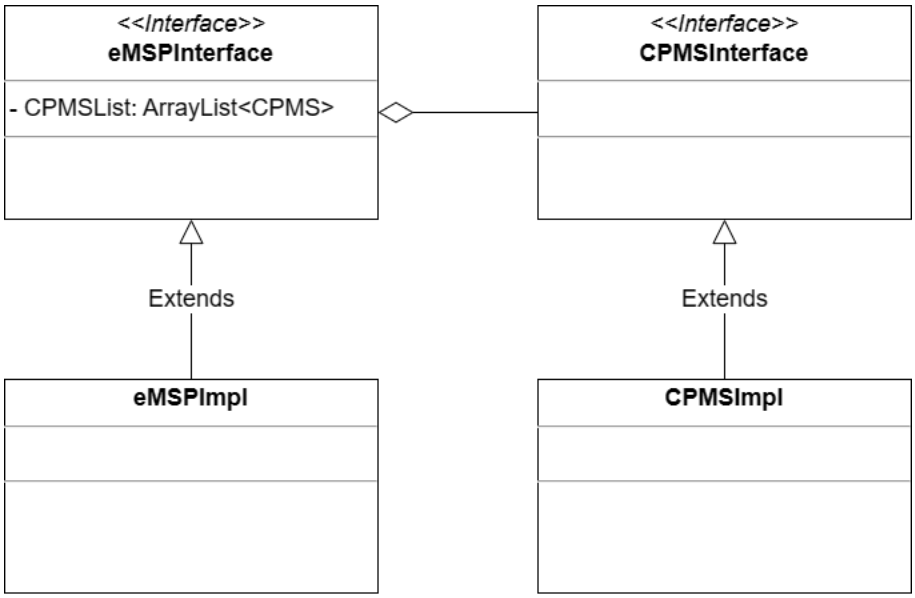

Figure 26: Bridge pattern representation in the system

2.6.1. eMSP architectural styles and patterns

- Model View Controller pattern [The MVC pattern]: It consists in grouping the system in three clusters:
  - the model, with all the persistent data of the system;
  - the view, with the logic to present the model;
  - the controller, with the business logic for elaborating requests from the view and manipulating the model.

  The controller component is then divided into two classes: the first, distributed on the client, elaborates the data retrieved from the calendar and the vehicle in order to avoid cumbersome requests to the server; the second, distributed on the application server, handles the requests made by the client and interacts with the model.
- The three-tier architecture [The multi-tier pattern]: It increases the modularity and

manageability of the system by dividing it into presentation, application, and data tiers. This allows for improving a single tier without affecting the others.

- The Data Access Object (DAO) pattern [The DAO pattern]: It maps relational database tables directly to objects in the application. This decouples the application and data layers further.

2.6.2. CPMS architectural styles and patterns

- The Model View Controller pattern [The MVC pattern]: is used to decouple the view logic from the rest of the system, simplify code organization and implementation, and provide a software layer to handle access to the model and prevent the presentation layer from directly accessing the model.
- The two-tier architecture [The multi-tier pattern] is used because the CPMS does not need to handle as much traffic as an eMSP system. As a result, it is simpler and only has two tiers: a thin-client with the presentation layer, which handles the view logic, and an application server.

2.7. Other design decisions

2.7.1. SPOF avoidance

Both the eMSP and CPMS systems have been designed with a focus on avoiding SPOFs. As eMall is a large service that competes with other similar services, this quality can also be a competitive advantage. In addition, with many potential users accessing the service at the same time, a downtime would result in a significant loss of money and damage to the public image. The cost of having and maintaining a redundant system is justified by the increased availability of the system. These decisions can be seen in Figures Figure 5 and Figure 6.

Firewalls, load balancers, VPN servers For firewalls [Redundancy in firewalls], load balancers [Redundancy in load balancers], and VPN servers [Redundancy in VPN servers], the general pattern is to have a floating IP [Floating IP] mapped to the exposed service, and two physical redundant systems that are directly connected to send and receive a "heartbeat" signal indicating that the system is up and running. If one system does not receive the heartbeat from the other, the floating IP linked to the service is redirected to the active system.

Application server For the application server, we use the load balancers to use both application servers (if available) or just the active one if the other is unavailable.

Databases The database system is designed to withstand temporary failures of a database server and to be able to recover from disasters. A secondary database server is synchronized in real-time to take over if the primary database server goes down. A third database server is located in a different location from the other two to enable a disaster-recovery plan; this server is synchronized once a day.

3. User interfaces

In this section mockups of the user interfaces are presented; an app for the final Users, an app for the CPOs and a site for the CPOmaintainers.

3.1. User

The user is whoever wants to use the service to book and manage charges trough the app. The mockups only represent the idea of the app, the actual design could differ based on the OS on which the app is implemented (different operative systems could provide different gestures/features to enhance the users experience).
It is assumed that the user has already installed the correct version of the app compatible with his/her operative system.
The user can close every popup by clicking outside of it (in the grey area).

3.1.1. Login

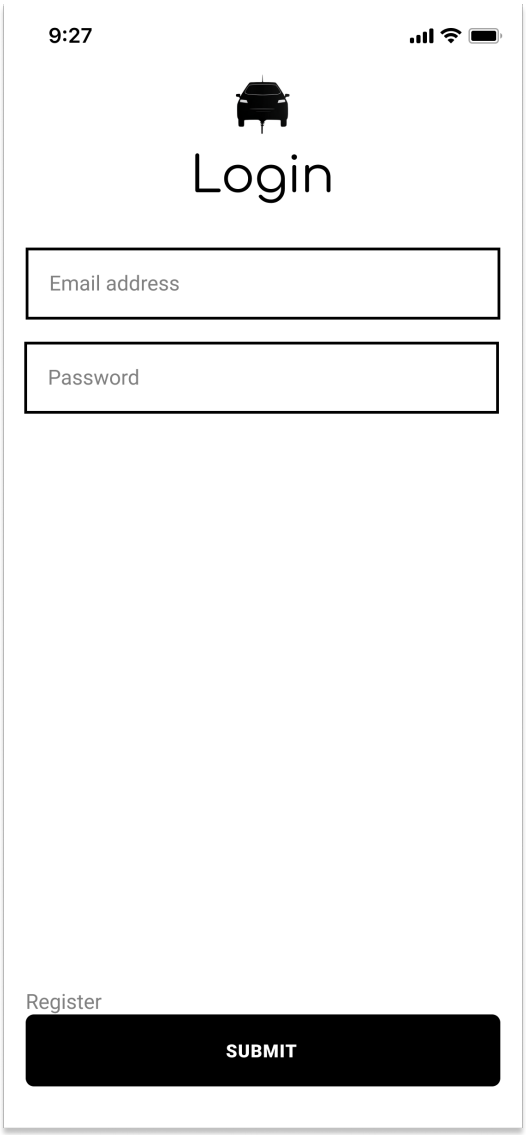

Figure 27: User Logs in

The first time the user opens the app he/her is prompted to log in by inserting email and password in the corresponding field and pressing the submit button. If the information provided are correct the search station page is displayed; otherwise the failed login page is shown.
The user can press the register button to open the register page.

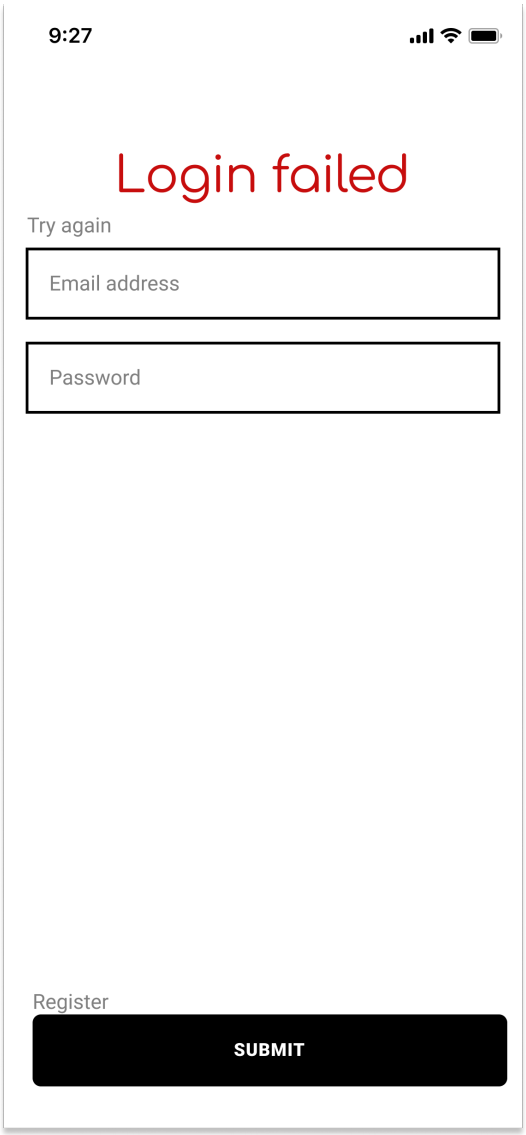

Figure 28: Wrong Credential

If the user has inserted the wrong email or password he/she can retry it in this page, otherwise the user can press the Sign Up button to open the register page.

3.1.2. Register

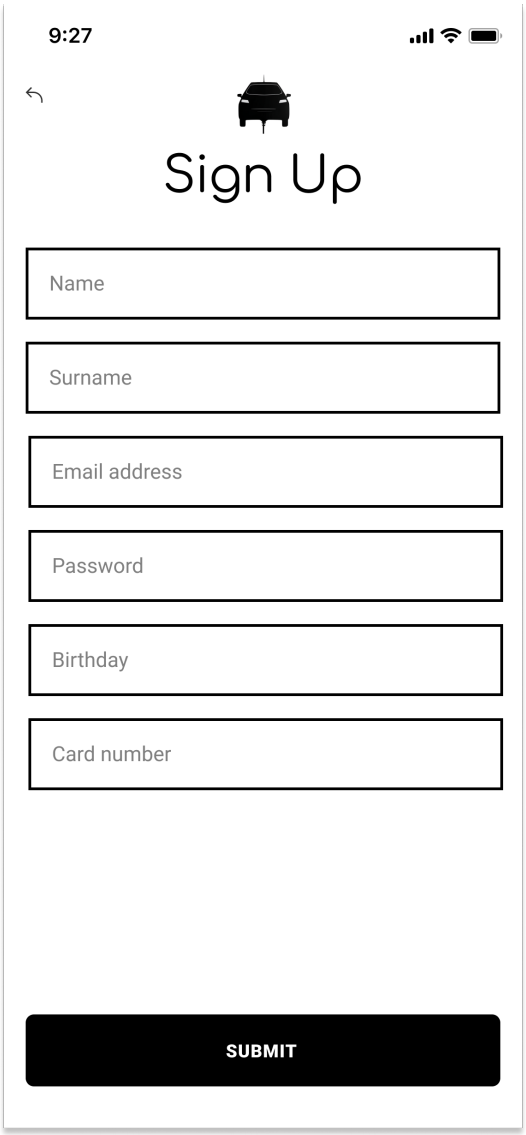

Figure 29: User Register

If the user has not already registered himself/herself he/her can do it here by inserting the right data in the corresponding fields and pressing the submit button. An email will be sent to the user with a link within, clicking the link will open the app displaying the confirmation page completing the registration procedure.
If the user wants to return to the login page he can do so by pressing the backward arrow in the top left corner.

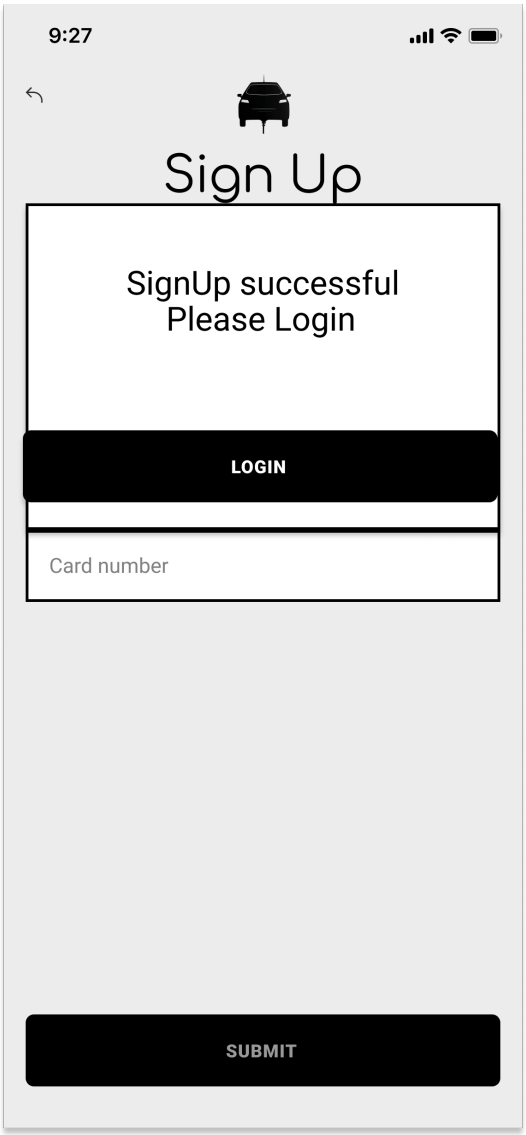

Figure 30: User Finish Registering

Once this page is displayed the user has to log in by clicking the log in button which opens the login page.

3.1.3. Search a Station

At any screen shown in this section the user can press the car logo at the top to load the manage charges page

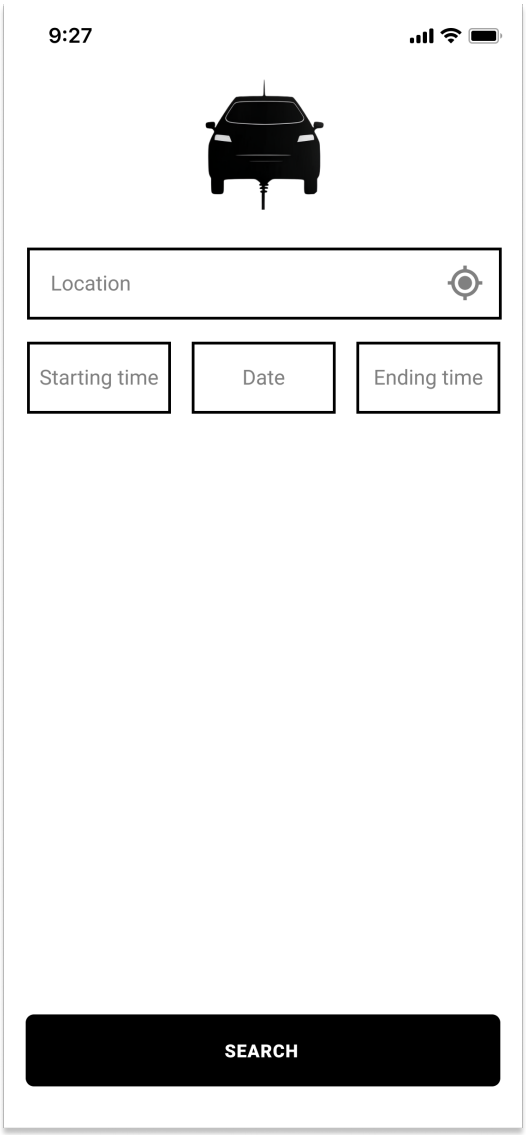

Figure 31: User Search for Stations

In this section the user can search for stations by inserting the location, the starting/ending time and the date for the charge. By pressing the location icon the user can automatically use his current location. Once the data are inserted the user can press Search to advance to the results page.

Figure 32: Search results

Here the user is shown the results of the search, using the slider on top the user can change the radius of search from the given position, at the right of the slider the selected distance is shown; at the top right the filter icon is displayed, if pressed the filter popup is showed. A list of stations with their details is displayed (station name, location, speed, distance from selected location, whether it uses renewable energy sources and price of the charge), clicking on a station opens the station details page. Pressing the Search button allows the user to make another search trough the search station page.

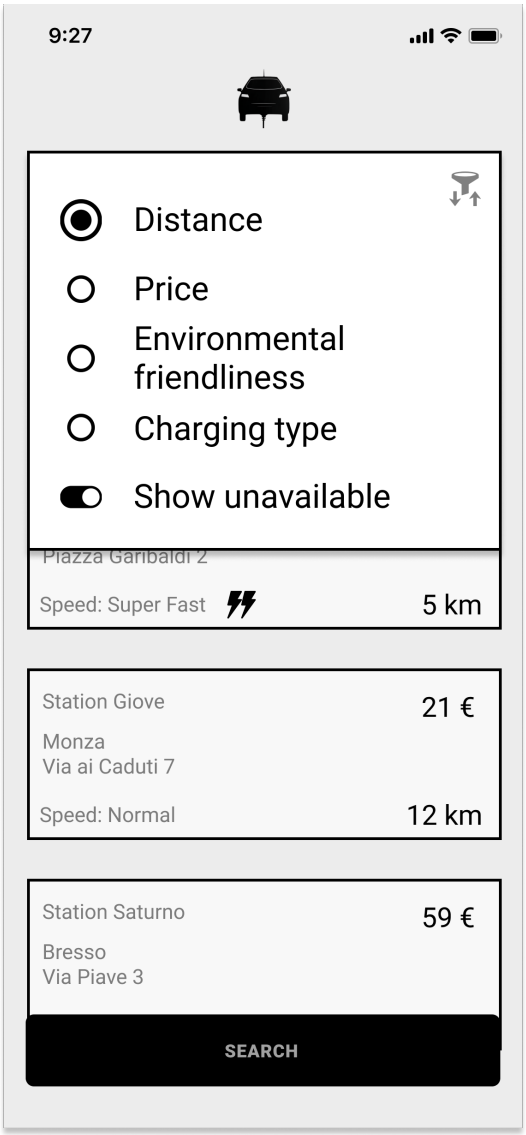

Figure 33: User Filters the Results

In this popup the user can select whether to sort the results by distance, price, environmental friendliness or charging type via a radio button, and to show or to not unavailable stations trough a toggle switch.

3.2. Select time frame

At any screen shown in this section the user can press the car logo at the top to load the manage charges page

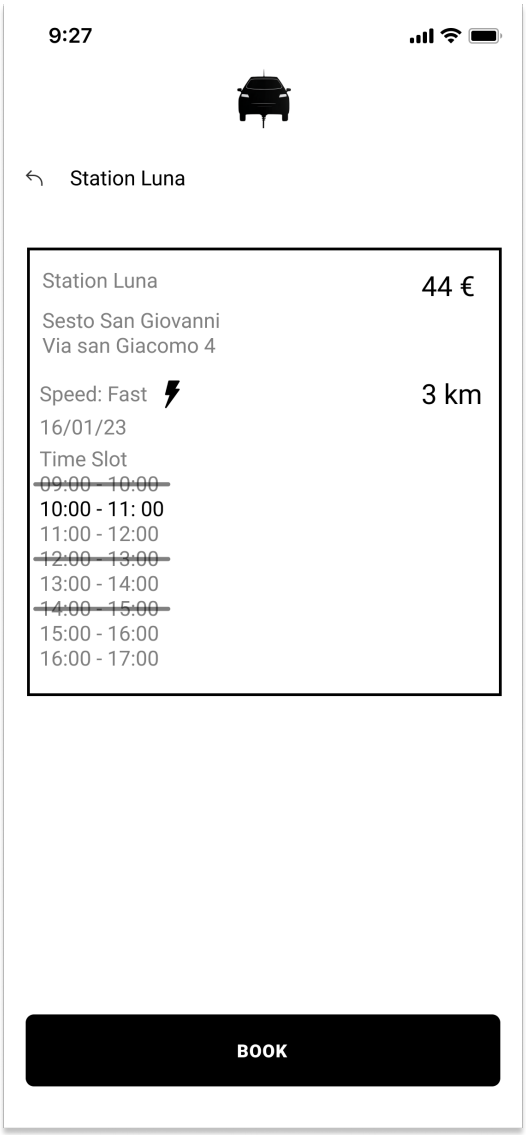

Figure 34: User Checks Station Details

Here the user can select a time frame. At the center of the screen the station details and a list of possible time frames are shown; the user can select the desired time frame by tapping on it. The selected time frame is bold while unavailable time frames are crossed. Once the user has selected the time frame he/her can proceed to the booking by clicking the book button opening the confirmation popup. The user can return to the search station page by clicking the back arrow in the top left corner.

3.2.1. Book a Station

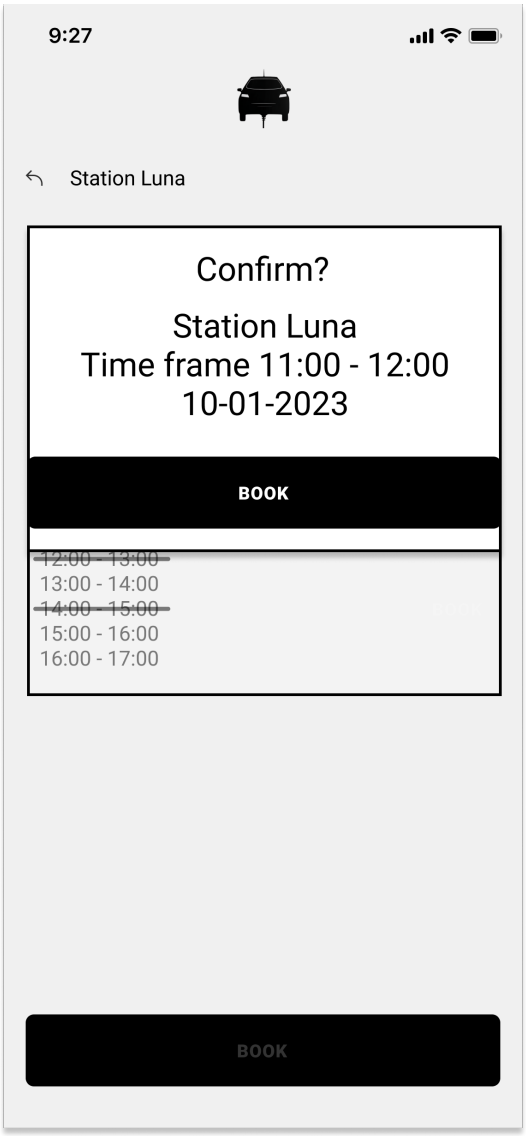

Figure 35: User Confirms the Booking

In this popup: station name, the selected date and time frame are displayed. The user can confirm the booking by pressing the Book button opening the confirmation popup.

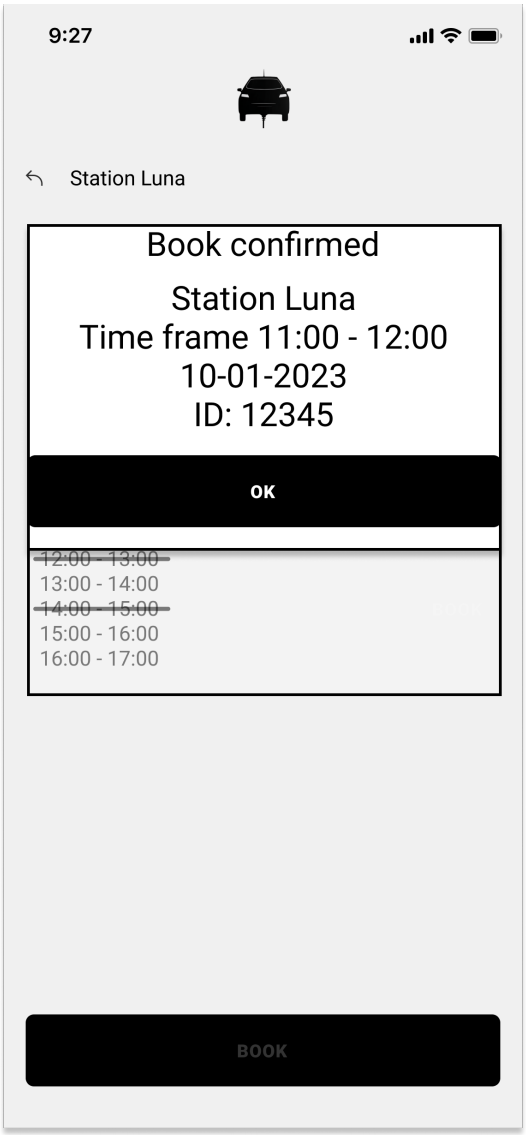

Figure 36: User Reads Confirmation Popup

In this popup the booking information are displayed confirming the user that the booking was successful. Pressing the ok button opens the search station page.

3.2.2. Checks Booked Stations

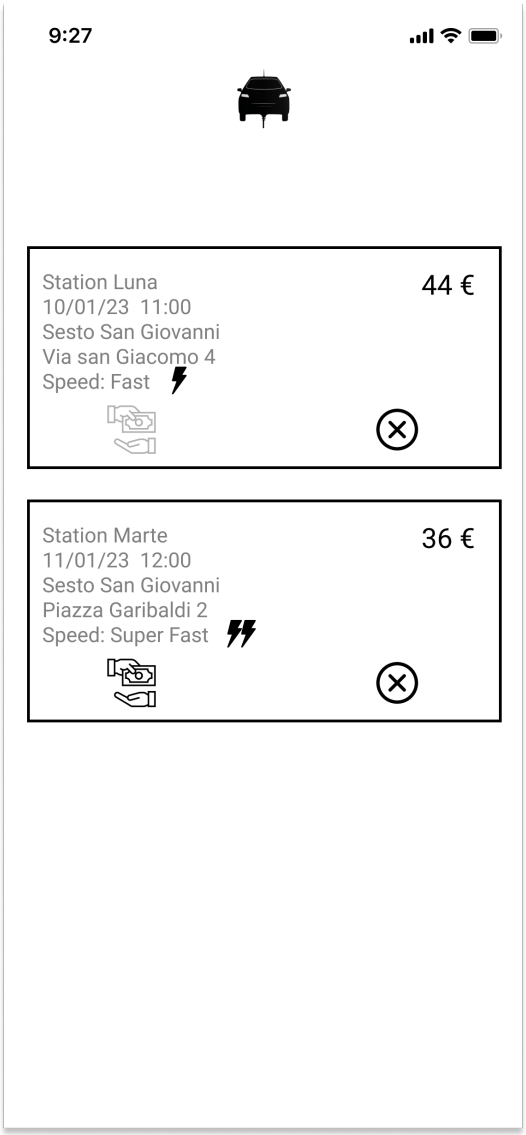

Figure 37: User Manages Charges

In this section the user can see a list of his/her booked charges; for each booking the name, date, time frame and charge speed are displayed; the user can also pay for the charge by pressing the pay button opening the pay popup and delete a charge by pressing the delete button opening the delete popup. The charge has been already paid (cannot be paid again) if the pay button is greyed out.
By pressing the car logo the user can switch his/her view to the search page.

3.2.3. Pay a Charge

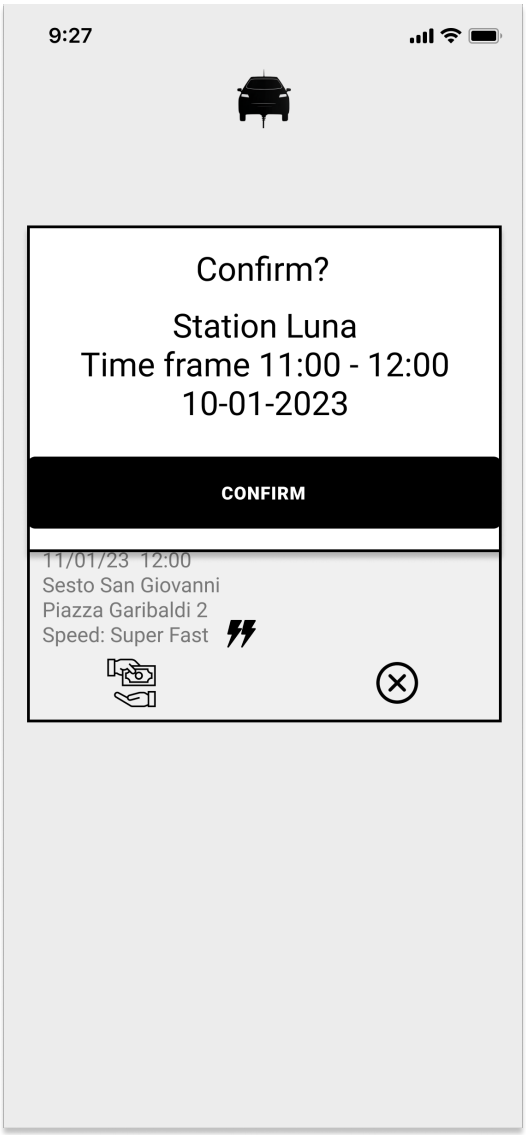

Figure 38: User Pays a Charge

In this section the charge details are displayed and the user can pay the charge by pressing the confirm button.

3.2.4. Cancel a Charge

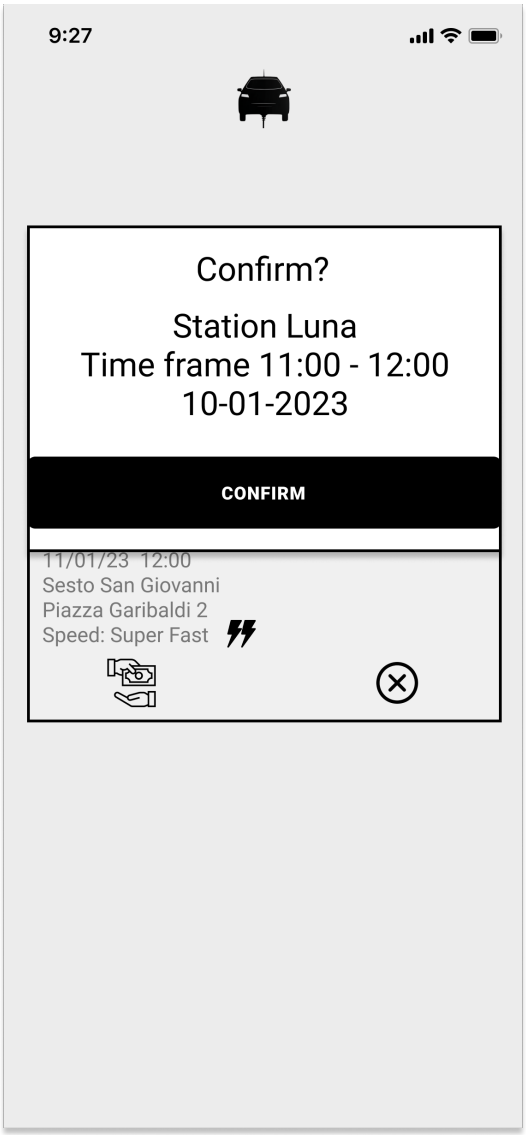

Figure 39: User Cancels a Charge

In this section the charge details are displayed and the user can pay the charge by pressing the confirm button.

3.2.5. Suggest a charge

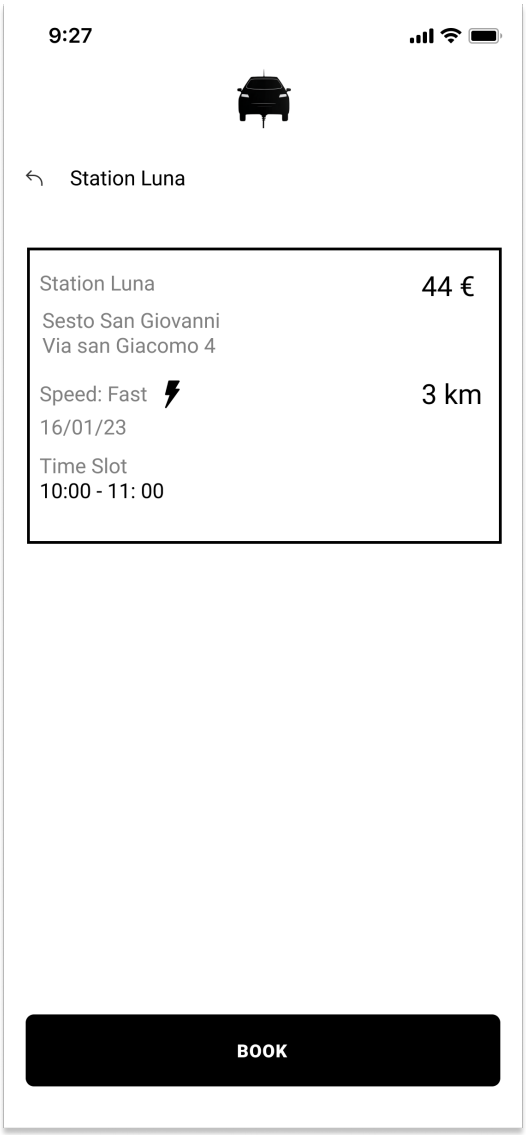

Figure 40: User Gets Suggested Charge

When the system thinks that a charge is needed/possible a notification to the user is sent. If the user opens it this page is displayed; here the info about the charge are shown and the booking can be confirmed by pressing the book button loading the booking popup.
The user can return to the search page by pressing the backward arrow in the top left corner.

3.3. CPO

This section show the interface for the CPO. The CPO can do everything trough an app. The app is different than the one used by the user and it is assumed to be already installed on a device.

3.3.1. Login

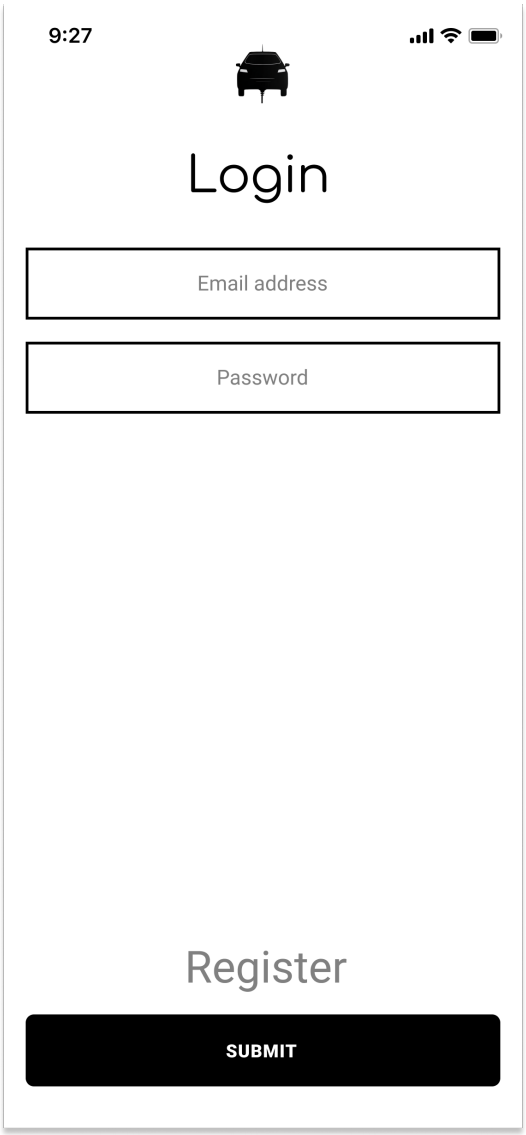

Figure 41: CPO Login

Once the CPO opens the app this page is displayed. Here the CPO can log into the system inserting the email and password provided during the registration. When the user presses the submit button, if the information provided are correct, the homepage is loaded, otherwise the failed login page is shown.
If the CPO is not registered it can do it by pressing the Register button which loads the register page.

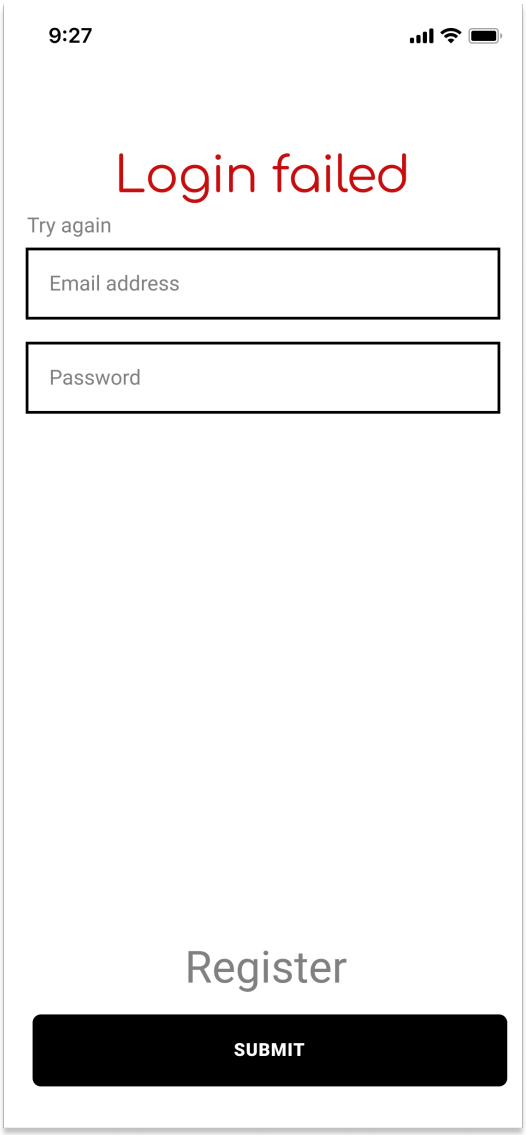

Figure 42: Wrong Credential

If the CPO has inserted the wrong email or password he/she can retry to log in this page, otherwise the CPO can press the Register button to open the register page.

3.3.2. Register

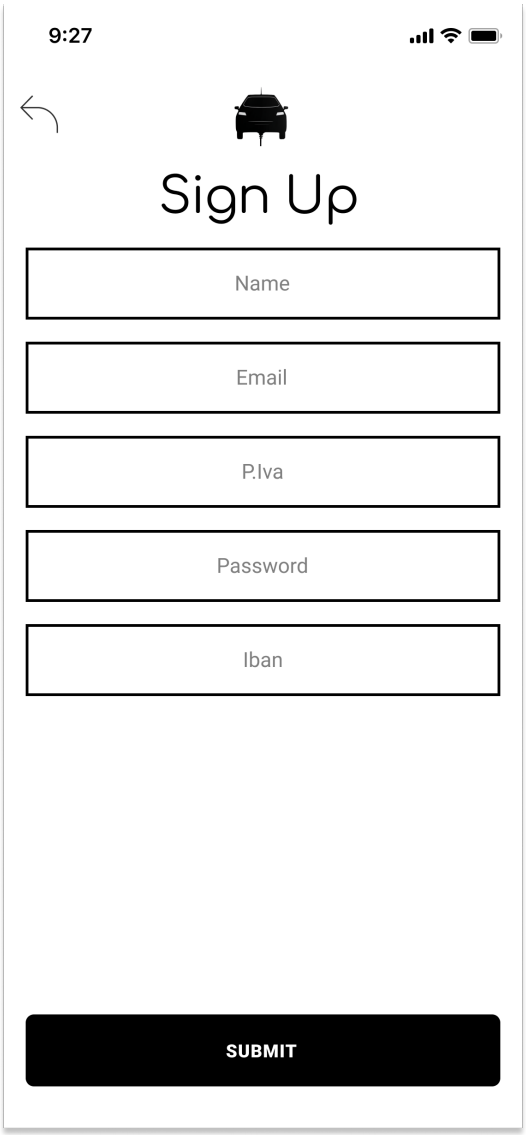

Figure 43: CPO Registers

In this page the CPO can register itself to the service by providing the asked information and pressing the Submit button. An email will be sent to the CPO with a link within, clicking the link will open the app displaying the confirmation page completing the registration procedure.
If the CPO wants to return to the login page he/she can do so by pressing the backward arrow in the top left corner.

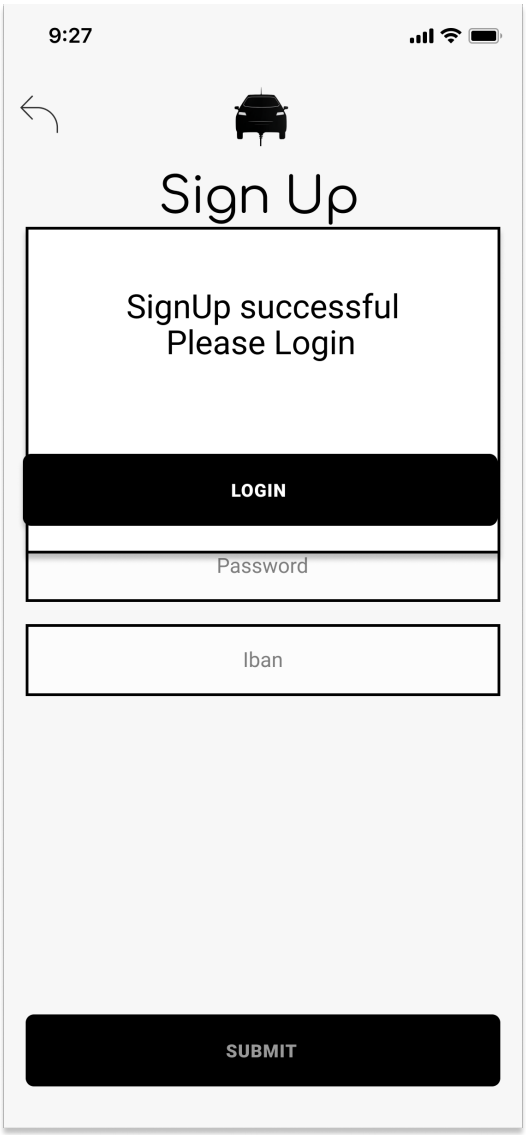

Figure 44: CPO Finishes Registering

Once this page is displayed the CPO has to log in by clicking the log in button which opens the login page.

3.3.3. Home Page

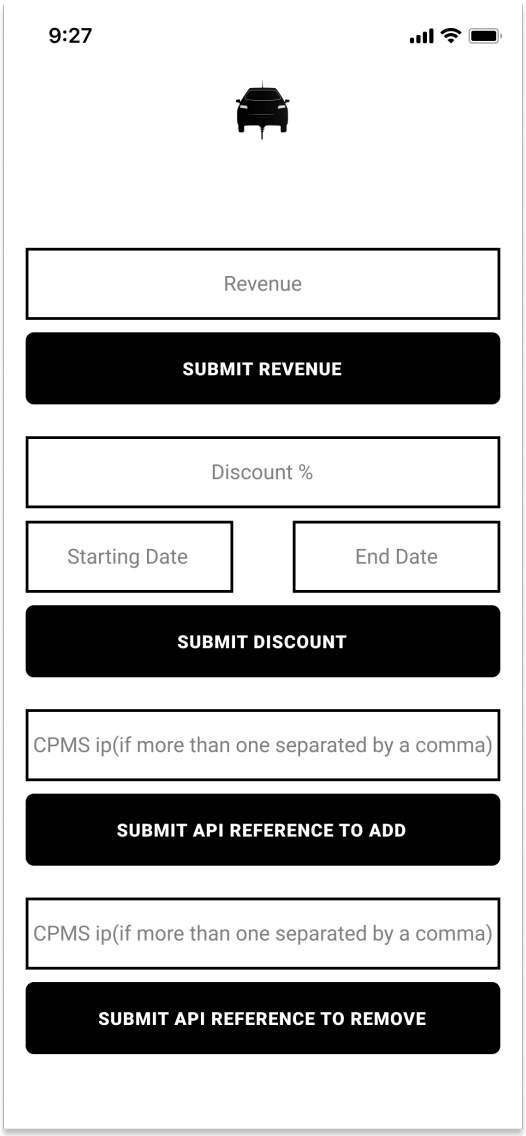

Figure 45: CPO accesses the site functions

In this page the CPO can change the revenue, create discounts and add/remove CPMS reference. To change the Revenue the CPO has to insert the new revenue percentage in the field and press the submit revenue button. The revenue will be updated and the page refreshed.
To create new discounts the CPO has to insert the percentage, the time frame of the discount and press the submit discount button. The discount will be updated and the page refreshed.
To add new CPMSs the CPO has to insert the IP to the CPMS API (if more than one separated by a comma) and press the submit API reference to add button, the API reference will be updated and the page refreshed.
To remove a CPMS the CPO has to insert the IP to the CPMS API (if more than one separated by a comma) and press the submit API reference to remove button, the CPMS will be removed and the page refreshed.

3.4. CPOMaintainer

This are the interfaces for the maintainers to access their CPMSs, it is assumed that the maintainer is already connected to the VPN trough a pc and an internet connection.

3.4.1. Login

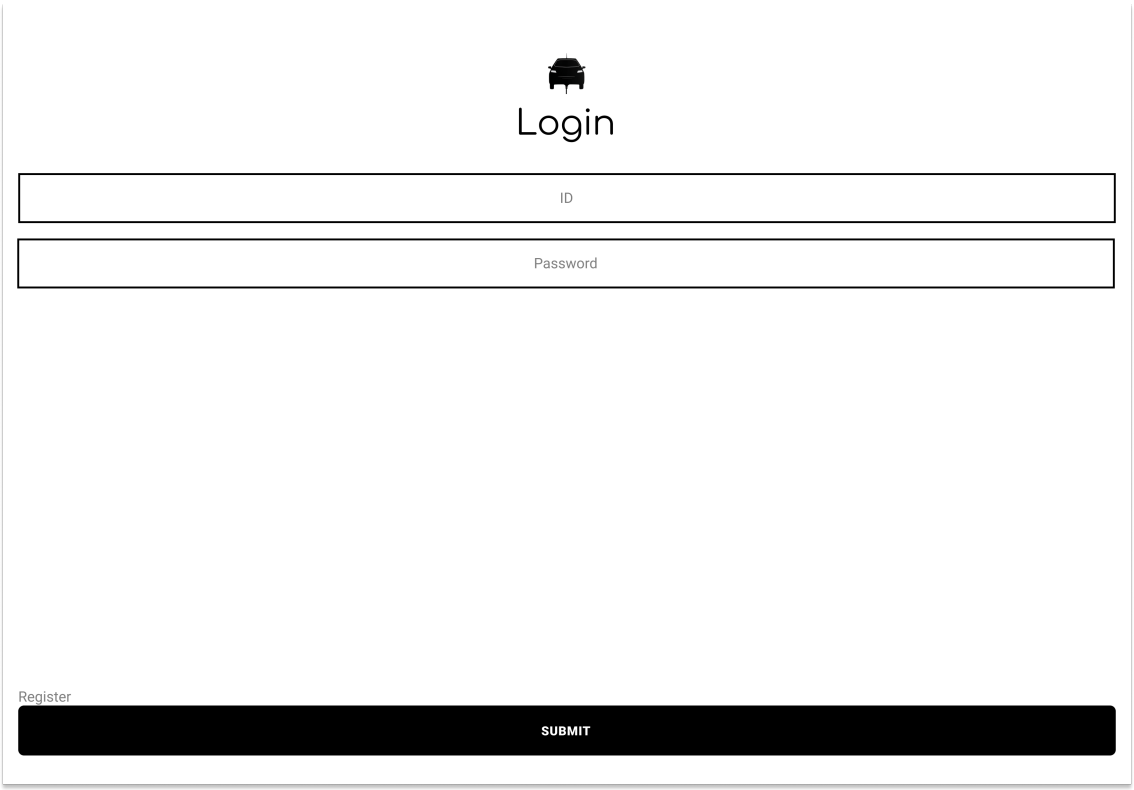

Figure 46: CPOmaintainer logs In

Here the CPOMaintainer can access the CPMS by inserting the correct ID and password and pressing the submit button. If the data is valid the homepage is displayed.

3.4.2. Homepage

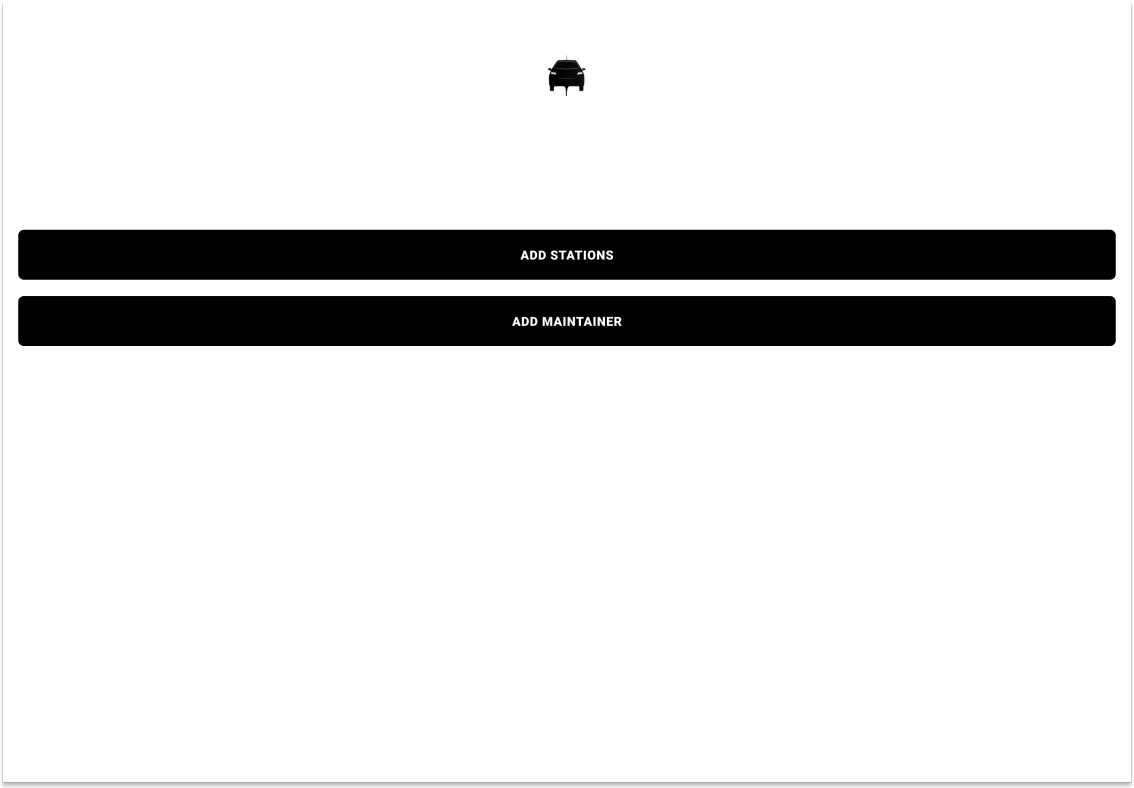

Figure 47: CPOmaintainer accesses the site functions

Here the maintainer can select the function to operate by either clicking the add station button (opening the add station page) or clicking the add maintainer button (opening the add maintainer page)

3.4.3. Station Management

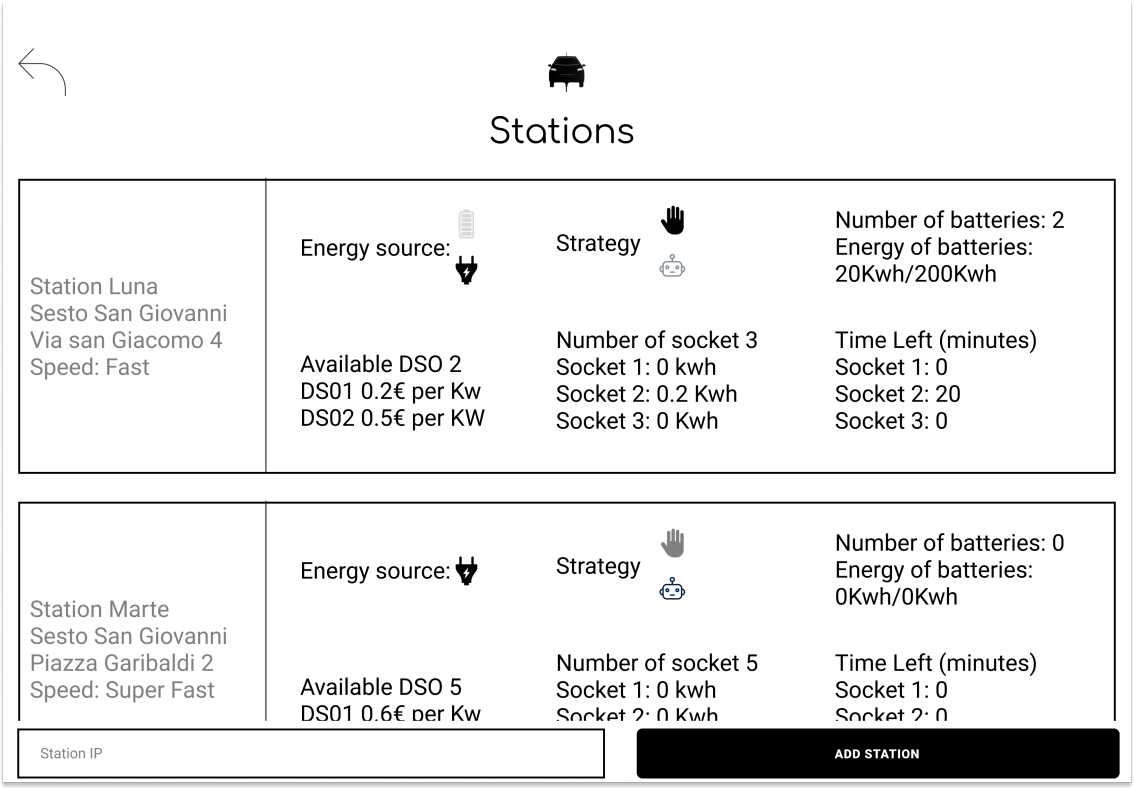

Figure 48: CPOmaintainer manages stations

In this page all the information regarding the maintained stations are displayed. On the left the station location and name; on the right: the battery capacity and their quantity, the available DSOs and their prices, the number of sockets, and for each active socket the current electricity usage and the time remaining for the charge.
The maintainer can change the station's energy source by clicking the battery or plug icon (meaning battery or grid power is getting used). In the same fashion he/she can select the strategy by clicking either the hand or robot icon; the first representing manual while the second automatic. The selected icon is bold while the other is greyed out. If no battery is available only the plug icon is displayed.
In this page it is also possible to add a new station to the system by inserting its IP in the "station IP" field and pressing the add station button; the station will be added and the page refreshed displaying the new station and its data.
The user can return to the homepage by pressing the backward arrow in the top left corner.

3.4.4. Add Maintainer

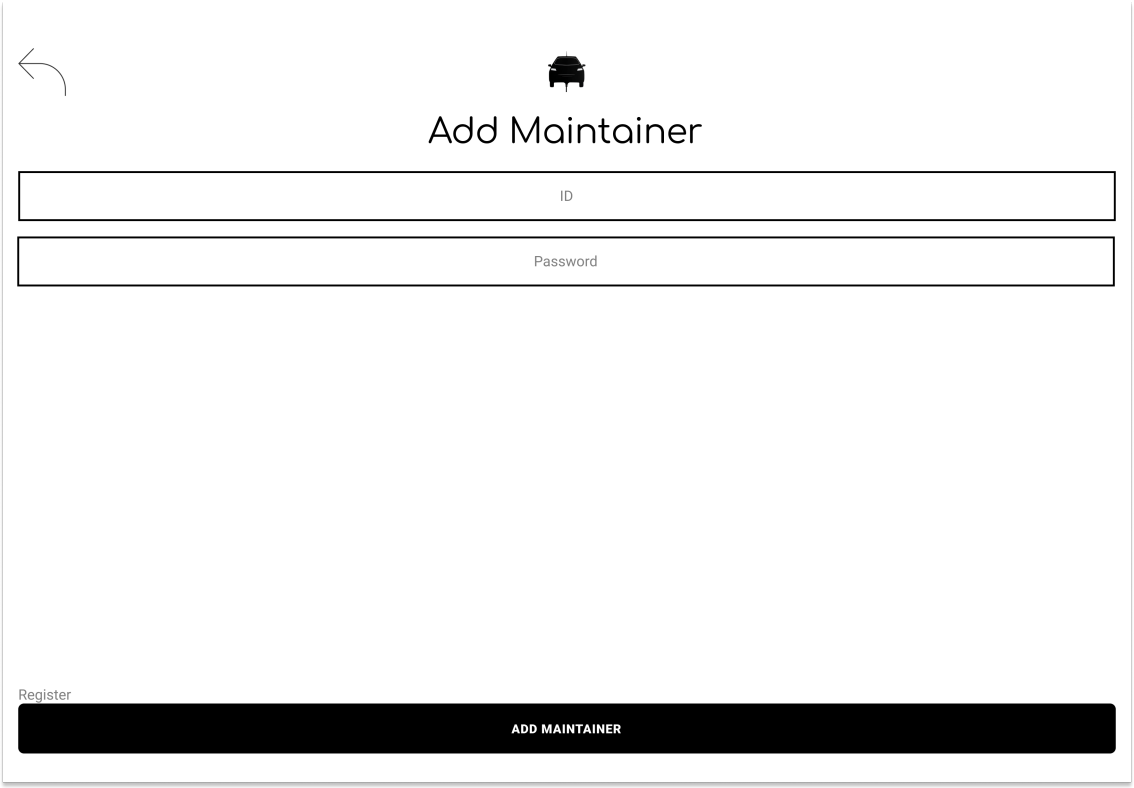

Figure 49: CPOmaintainer adds new CPOmaintainers

In this section the maintainer can add other maintainers to the CPMS by simply inserting the credentials(id and password) and pressing the add maintainer button, adding the new maintainer and loading the homepage.
The user can return to the homepage by pressing the backward arrow in the top left corner.

3.5. Scenario UI

The following table relate the different sequence diagrams with the corresponding user interfaces

Sequence diagrams	Implemented by figure number
user-signs-up	29, 30
cpo-signs-up	43, 44
cpo-adds-cpms	45
cpo-removes-cpms	45
cpo-sets-revenue-percentage	45
cpo-sets-special-offer	45
user-executes-authorized-command	27, 28, 41, 42
user-searches-stations	31, 32, 33, 34
user-books-charge	34, 35, 36
user-pays-charge	37, 38
user-cancels-charge	37, 39
user-gets-suggestion	40
cpomaintainer-executes-authorized-command	46
cpomaintainer-adds-cpomaintainer	47, 49
cpomaintainer-adds-stations	47, 48
cpomaintainer-sets-station-strategy	47, 48
cpomaintainer-sets-station-source	47, 48

Table 1: Linking table among sequences and user interfaces

4. Requirements traceability

R1	RegistrationComponent, Payment API, Mail API, Timer, eMall Model
R2	AuthenticationComponent, eMall Model
R3	AuthenticationComponent, eMall Model, StationsComponent, CPMS API, CPMSModel
R4	AuthenticationComponent, eMall Model ChargeManagerComponent, CPMS API, CPMSChargeManagerComponent, CPMSChargingStation API
R5	AuthenticationComponent, eMall Model, ChargeManagerComponent, CPMS API, CPMSChargeManagerComponent, CPMSChargingStation API
R6	AuthenticationComponent, eMall Model, ChargeManagerComponent, Mail API, Payment API, CPMS API, CPMSChargeManagerComponent, CPMSChargingStation API
R7	AuthenticationComponent, eMall Model, ChargeManagerComponent, Mail API, Payment API, CPMSChargeManagerComponent, CPMSChargingStation API
R8	AuthenticationComponent, eMall Model, StationsComponent, CPMS API, CPMSModel
R9	AuthenticationComponent eMall Model CalendarAPI SuggestionEngine
R10	AuthenticationComponent eMall Model CalendarAPI VehicleAPI eMallAPI SuggestionEngine
R11	AuthenticationComponent, eMall Model, ChargeManagerComponent, CPMS API, CPMSChargeManagerComponent, CPMSChargingStation API
R12	CPMSManagerComponent, CPOManagerComponent, CPMS API, eMall Model
R13	RegistrationComponent, Payment API, Mail API, Timer, eMall Model
R14	AuthenticationComponent, eMall Model, CPMSManagerComponent, StationsComponent, CPMS API, CPMSModel
R15	RegistrationComponent, Payment API
R16	AuthenticationComponent, eMall Model, CPOManagerComponent
R17	AuthenticationComponent, eMall Model, CPOManagerComponent
R18	CPMS API, CPMSChargeManagerComponent, CPMSModel
R19	CPMSAuthenticationComponent, CPMSModel
R20	CPMSAuthenticationComponent, CPMSModel, CPMSRegistrationComponent, CPMSStationManagerComponent, CPMSChargingStation API
R21	CPMSAuthenticationComponent, CPMSModel, CPMSStationManagerComponent

Table 2: Requirements traceability table

R1 *The eMSP shall allow the users to register, providing name, surname, birthday, email, password, payment method*: The unregistered user uses RegistrationComponent in order to issue a registration. Then the system will use the Payment API in order to confirm that the payment method is valid. After the verification of parameters, a mail through the Mail API is sent and a timer with the Timer component is set in order to issue a deadline for the confirmation of registration. When the confirmation of registration is received, the user will be added to the eMall Model.

R2 *The eMSP shall allow the user to login with email and password*: A user, in order to log in,

communicates the credentials to the AuthenticationComponent. This component will query the eMall Model in order to verify the credentials.

R3 *The eMSP shall provide information about a selected station such as types of available sockets, price for the charge, location, available time slots, parameter on environmental friendliness*: After the user is logged in (see **R2**), the user interacts with the StationsComponent in order to retrieve through the CPMS API the charging station information contained in the CPMSModel.

R4 *The eMSP shall reserve a socket in the right charging station for a user who booked a charge through the application*: After the user is logged in (see **R2**), in order to book a charge, he interacts with the ChargeManagerComponent which will pass through the CPMS API to interact with the CPMSChargeManagerComponent which will inform the charging station through the CPMSChargingStation API. The charging station will respond to the CPMSChargingStation API with its state in the selected time frame. If the operation succeeds, the slot is booked for that user.

R5 *The eMSP shall allow only one user to book a socket in a particular time slot, so no booking collisions shall occur*: After the user is logged in (see **R2**), in order to book a charge, he interacts with the ChargeManagerComponent which will pass through the CPMS API to interact with the CPMSChargeManagerComponent which will inform the charging station through the CPMSChargingStation API. The charging station will respond to the CPMSChargingStation API if the charging station selected is busy or not for that type of charge in the time frame selected. If the operation succeeds, is assured that nobody else can book that charging socket for that time slot.

R6 *The eMSP shall allow the user to pay for a booked charge*: After the user is logged in (see **R2**), and after successfully booking a charge the user can issue a payment for the booked charge. So, through the ChargeManagerComponent, the application interfaces with the Payment API to pay for the charge. If the payment is successful, then the application will communicate to the relative charging station (through the CPMS API and the CPMSChargingStation API) to activate the charge booked. After this, the system will also send a confirmation mail through the Mail API.

R7 *The eMSP shall refund the user when a charge is canceled*: After the user is logged in (see **R2**), and after successfully booking a charge the user can issue a deletion for the booked charge. So, through the ChargeManagerComponent and the CPMS API, the application interacts with the CPMSChargeManagerComponent in order to cancel a charge. This will check for the charge existence and its owner. The deletion will go on only if the user issuing the deletion is the owner of the charge. After these checks the charge can be deleted informing the charging station through the CPMSChargingStation API and the refund can be made (using the Payment API) only if the user has already payed for it. After this, the system will also send a confirmation mail through the Mail API.

R8 *The eMSP shall allow the user to see nearby*[1] *charging stations ordered by distance, price or environmental friendliness*: After the user is logged in (see **R2**), he can request infos of nearby charging stations through the StationsComponent. This will query the CPMSModel thanks to the CPMS API. The client application can change the retrieved data sorting in order to present it as ordered by distance, price or environmental

[1] This parameter may be set by the user

friendliness.

R9 *The eMSP shall be able to connect to a web calendar, retrieve information about the appointments and parse them*: After the user is logged in (see **R2**), the user can set up the connection to his calendar (through the CalendarAPI) in order to retrieve the user's appointments. The SuggestionEngine will be able to parse this data.

R10 *The eMSP shall be able to use the information about the appointments, the charging stations and the vehicle in order to proactively suggest to the user when and where to charge the vehicle*: After the user is logged in (see **R2**), the user can set up the connection to his calendar (through the CalendarAPI) and his vehicle (through the VehicleAPI) in the client application. When the setting for proactive suggestions is enabled, the eMSP client application will use its SuggestionEngine in order to interact with various components. With the CalendarAPI it retrieves the user's appointments. With the VehicleAPI it gets the vehicle data (i.e. battery State of Charge (SoC), vehicle position, ...). With the eMall server through the eMallAPI it retrieves data about charging stations in the eMall Model. With all this data, the SuggestionEngine will be able to elaborate a suggested time frame and location of the next charge.

R11 *The eMSP shall notify the user when the charging process is finished*: When a user books a charge, in the request generated by the ChargeManagerComponent there are all the infos of the charge, along with the user's email. This will be sent in all the forwarding requests through the CPMS API and CPMSChargeManagerComponent in order to arrive to the CPMSChargingStation API which will communicate the booking to the charging station. When the charge finishes, the charging station shows on its display the finished charge message and sends an email to the user.

R12 *The eMSP shall aggregate different CPOs*: An eMSP can interact with different CPOs thanks to the interface implemented by all the supported CPMSs. A list of all the registered CPOs is available in the eMall Model. The support for these CPMSs can be added through the CPMSManagerComponent while the revenue percentage and special offers can be set through the CPOManagerComponent. Finally, the CPMS API permits to interact with the CPMS.

R13 *The eMSP shall allow a CPO to register, providing name, email, password, International Bank Account Number (IBAN) and Partita IVA*: The unregistered CPO uses the RegistrationComponent in order to issue a registration. Then the system will confirm that the parameters are valid, in particular will use the Payment API in order to validate the IBAN. After the parameters verification, a mail through the Mail API is sent and a timer with the Timer component is set in order to issue a deadline for the confirmation of registration. When the CPO confirms its registration, it will be added to the eMall Model.

R14 *the eMSP shall allow to add to an already registered CPO a CPMS, providing its API reference*: After the CPO is logged in (see **R2**), he can insert a new CPMS in the eMall Model thanks to the CPMSManagerComponent. After verifying that the new CPMS is valid and active, the CPMSManagerComponent uses the CPMS API in order to retrieve all the charging stations contained in the relative CPMSModel and add them to the eMall Model.

R15 *The eMSP shall verify the correctness of the identification data for the CPOs*: During the registration phase of the CPO through the RegistrationComponent, the RegistrationComponent

verifies all the parameters and uses the Payment API in order to verify the IBAN.

R16 *The eMSP shall allow the CPO to set the wanted revenue percentage*: After the CPO is logged in (see **R2**), he can set the wanted revenue percentage on the eMall Model through the CPOManagerComponent.

R17 *The eMSP shall allow the CPO to set special offers*: After the CPO is logged in (see **R2**), he can add a wanted special offer on the eMall Model through the CPOManagerComponent.

R18 *The CPMS shall be reachable by eMSPs in order to perform or cancel a booking, or query the system*: The eMSP can use the CPMS API in order to communicate to the CPMS an operation. The CPMSChargeManagerComponent manages all the booking and canceling of charges while the CPMSModel can be accessed in order to retrieve data about the charging stations.

R19 *The CPMS shall allow the CPOmaintainer to access to the system*: When a CPO maintainer issues an operation, before executing it, his/her authentication is verified by the CPMSAuthenticationComponent. He/She can access the system providing an identification number and a password that will be validated by the CPMSAuthenticationComponent which will query the CPMSModel.

R20 *The CPMS shall allow the CPOmaintainer to modify the information about their systems, such as adding/removing charging stations, set stations sources and create/remove maintainers*: Once the CPO maintainer is authenticated (see **R19**), he/she can execute different operations such as adding other CPO maintainers (through the CPMSRegistrationComponent) or manage the charging stations (through the CPMSStationManagerComponent and the CPMSChargingStation API).

R21 *The CPMS shall allow the CPOmaintainer to choose manual or automatic mode*: Once the CPO maintainer is authenticated (see **R19**), he/she can manage the charging stations with the manual or automatic modes through the CPMSStationManagerComponent.

5. Implementation, Integration and Test plan

5.1. Implementation

The implementation phase should follow the component diagram. eMall and the CPMS could be implemented at the same time by different teams. The two components are fully independent and if the interfaces are respected the development of one should not interfere with the other. This is permitted thanks to the usage of the bridge pattern (see subsection 2.6).
In this section eMall and CPMS will be analyzed separately following a bottom up strategy. External components (API) are considered already finished and tested, furthermore they will not be discussed in this section.

5.1.1. eMall

The implementation should be divided in 3 layers to be developed in the following order

1. Model: The eMallModel should be the first component to be developed being the core of the system. The divide and conquer pattern should be used as much as possible to allow the whole team to work at the same time.
2. Middle components: the second wave of development should focus on the: CPMSAPI, AuthenticationComponent, PaymentAPI and MailAPI. This components are independent from each other and should be developed at the same time compatibly with the team size.
3. Leaf components: finally the last components can be implemented at the same time compatibly with the team size.

Once the logic of the system is completed the apps (eMallClient: the view in MVC pattern) (for users and CPOs) should be developed.

5.1.2. CPMS

Similarly to the eMall the CPMS should be implemented in 3 parts

1. Model: The CPMSModel should be the first component to be developed being the core of the system. The divide and conquer pattern should be used as much as possible to allow the whole team to work at the same time.
2. Middle components: the second wave of development should focus on the: CPMSAuthenticationComponent, CPMSChargingStationAPI. This components are independent from each others and should be developed at the same time compatibly with the team size.
3. Leaf components: finally the last components can be implemented at the same time compatibly with the team size.

Once the logic of the system is completed the website (for CPOMaintainers) should be developed.

5.2. Integration and Test Plan

The components should be integrated and unit tested. Each component inside the same layer can be tested autonomously giving the single teams the flexibility needed to solve problems that arose during testing. The unit testing shall be done using a driver to simulate the rest of the system. The single units of the same abstraction level are fully testable independently, thus a parallel integration and test campaign can be sustained depending on the team's flexibility. A white box testing for each unit is mandatory to cover as much code and branches as possible. A good level of path coverage is above 80%. At the end of the unit tests a series of black box tests must be done to confirm that the system behaves like expected. [Black and white box testing]

5.2.1. eMall

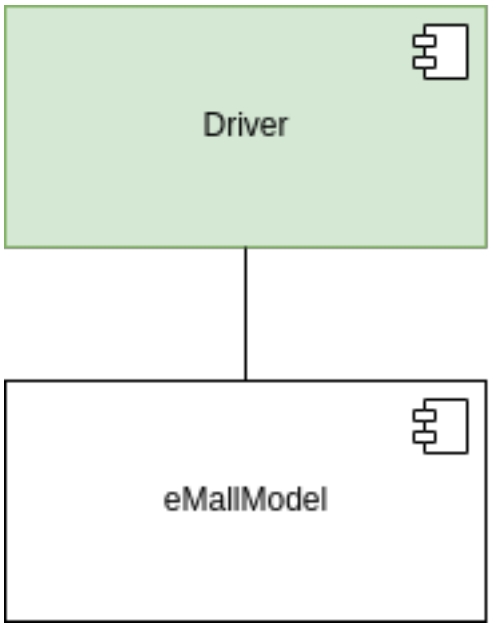

Figure 50: eMall model testing

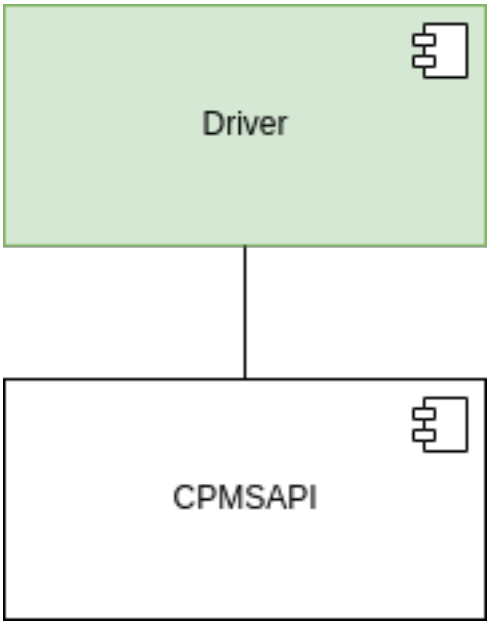

Figure 51: CPMSAPI testing

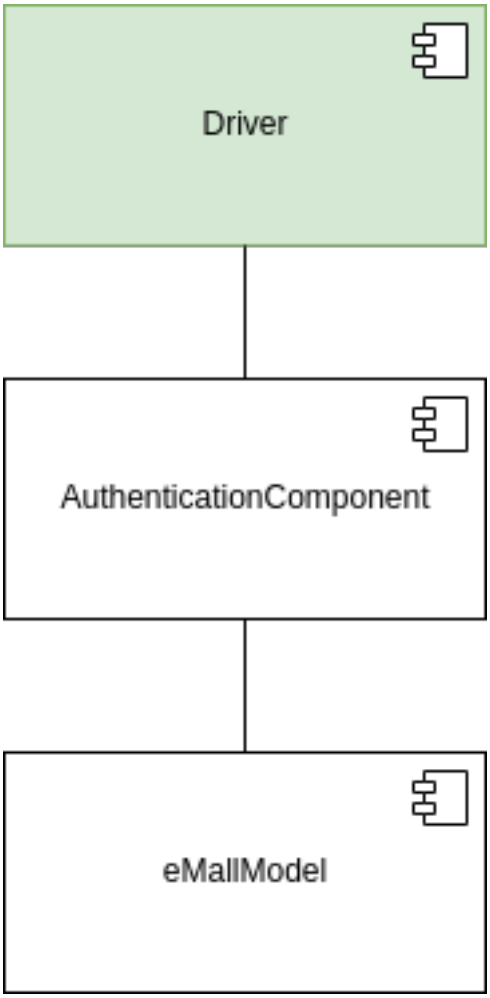

Figure 52: AuthenticationComponent testing

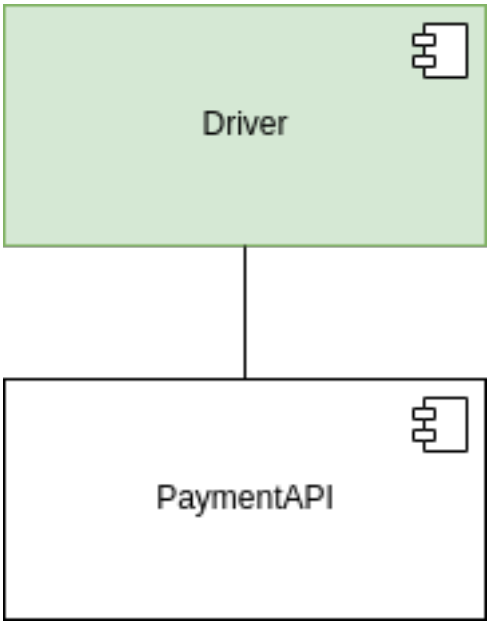

Figure 53: Payment API testing

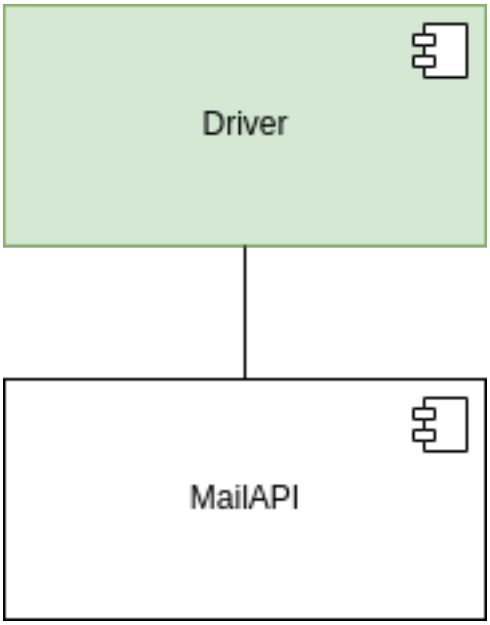

Figure 54: MailAPI testing

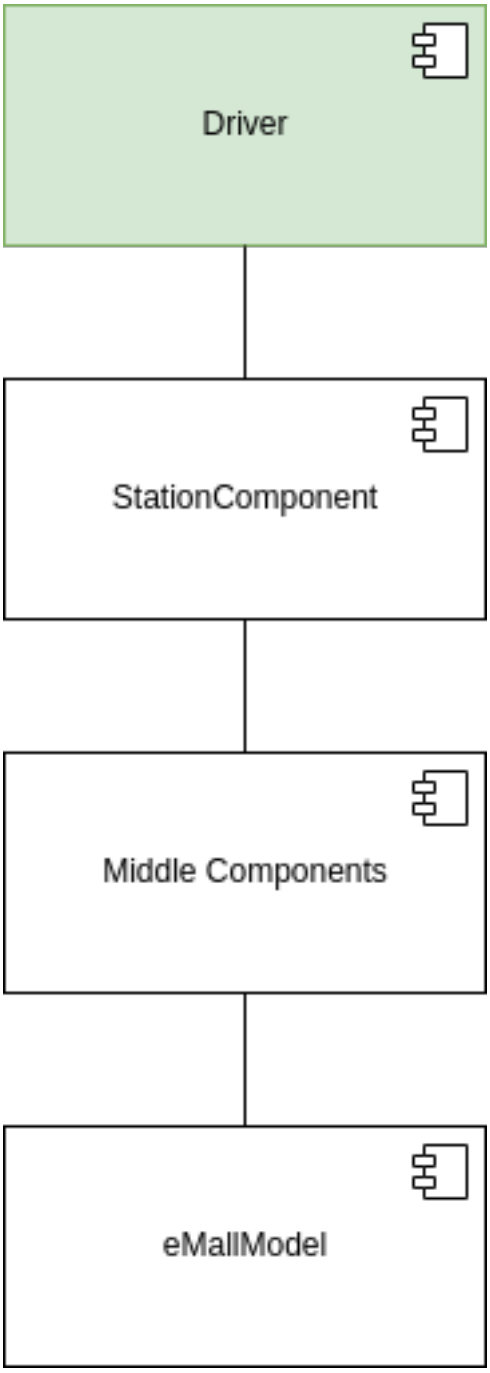

Figure 55: StationComponent testing

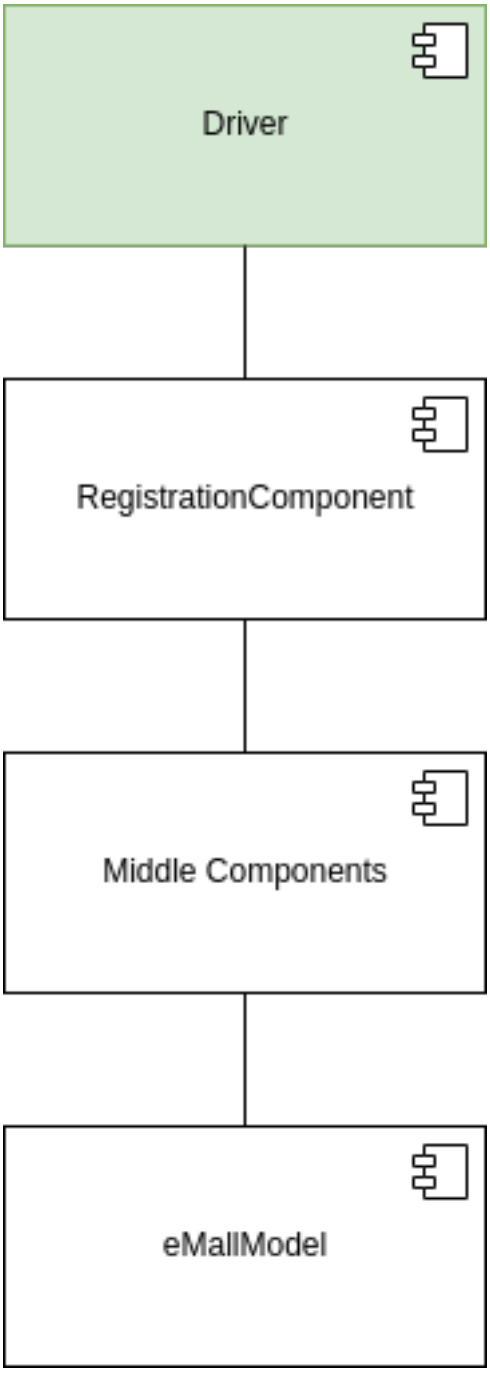

Figure 56: RegistrationComponent testing

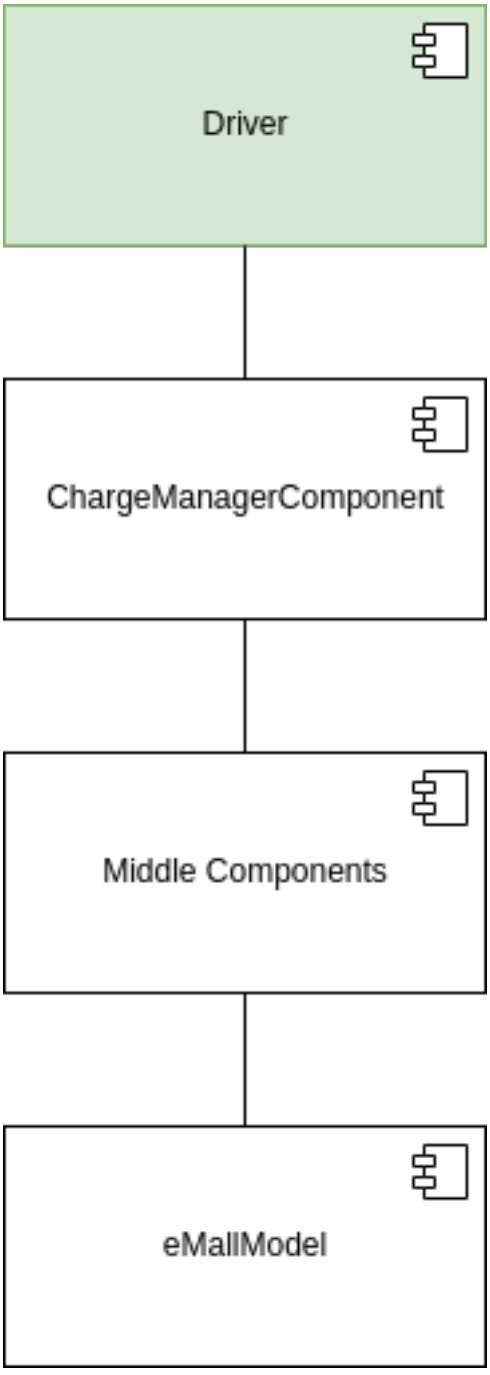

Figure 57: ChargeManagerComponent testing

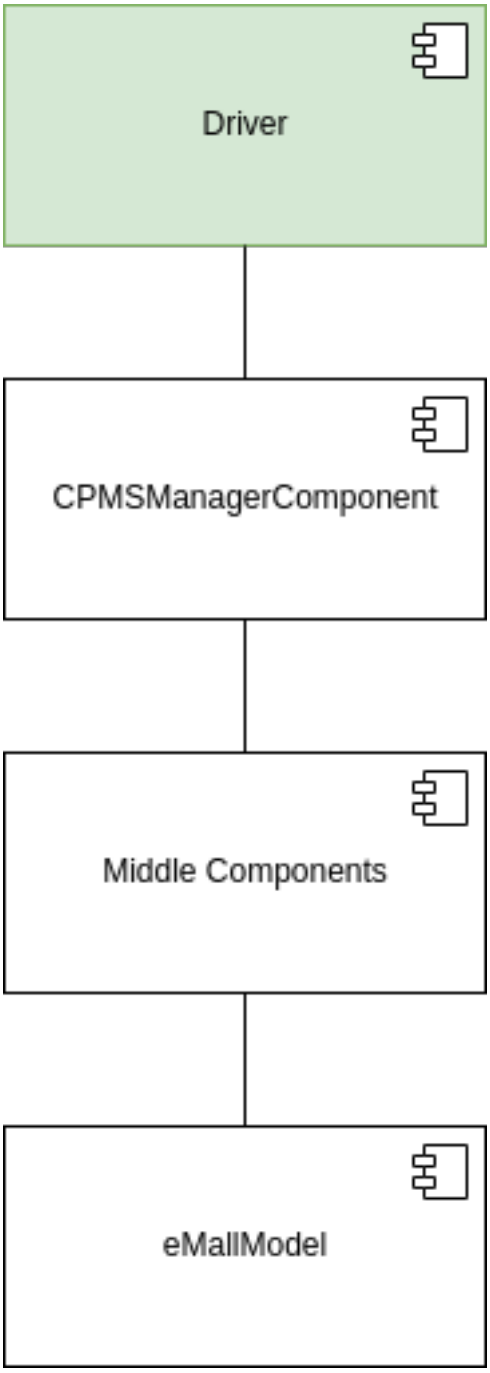

Figure 58: CPMSManagerComponent testing

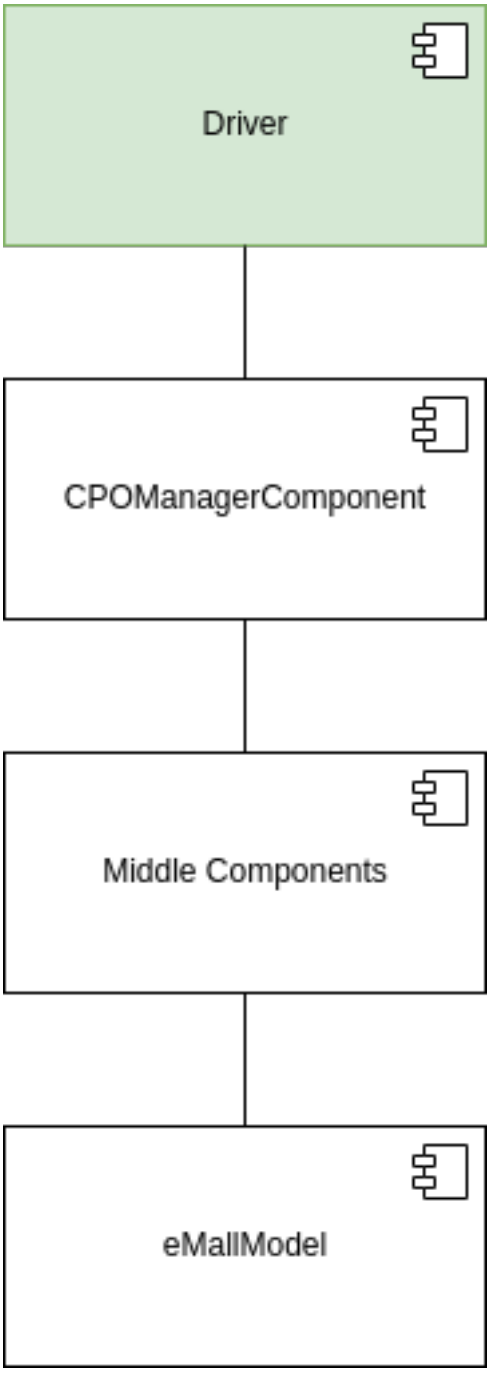

Figure 59: CPOManagerComponent testing

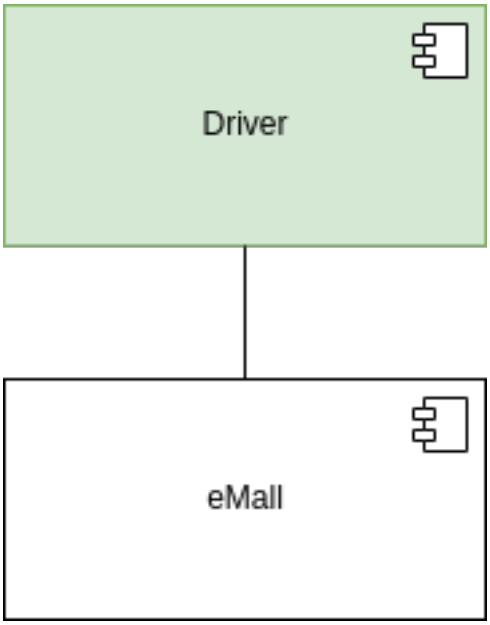

Figure 60: eMall testing

5.2.2. CPMS

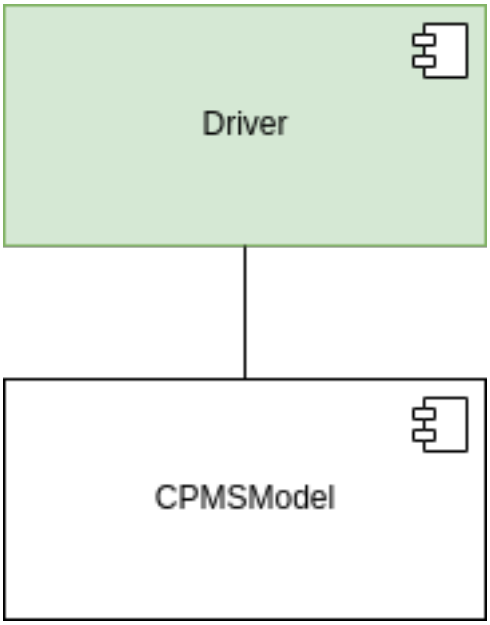

Figure 61: CPMSModel testing

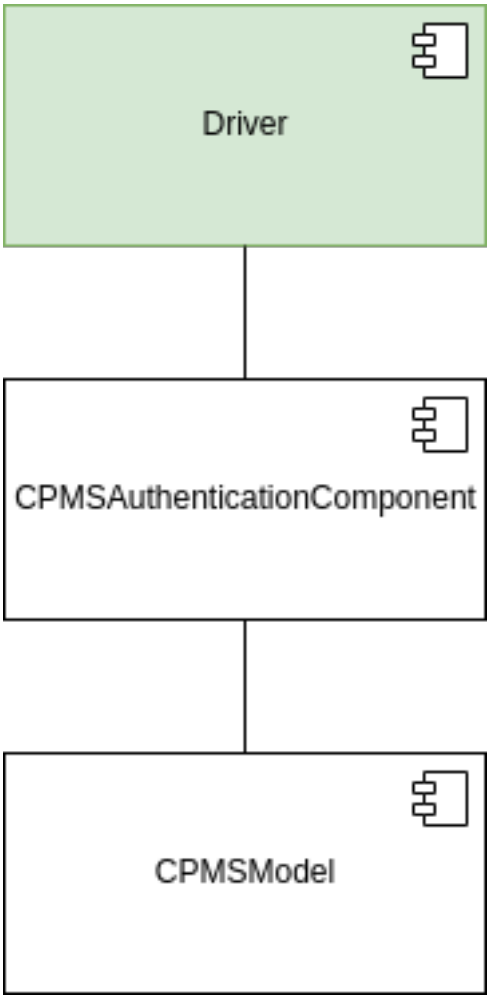

Figure 62: CPMSAuthenticationComponent testing

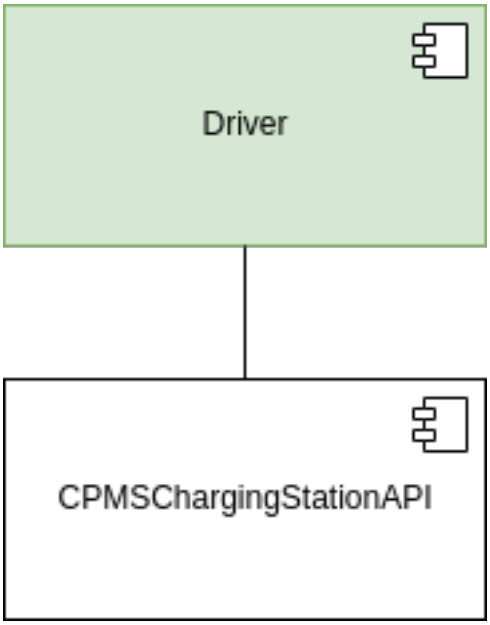

Figure 63: CPMSChargingStation testing

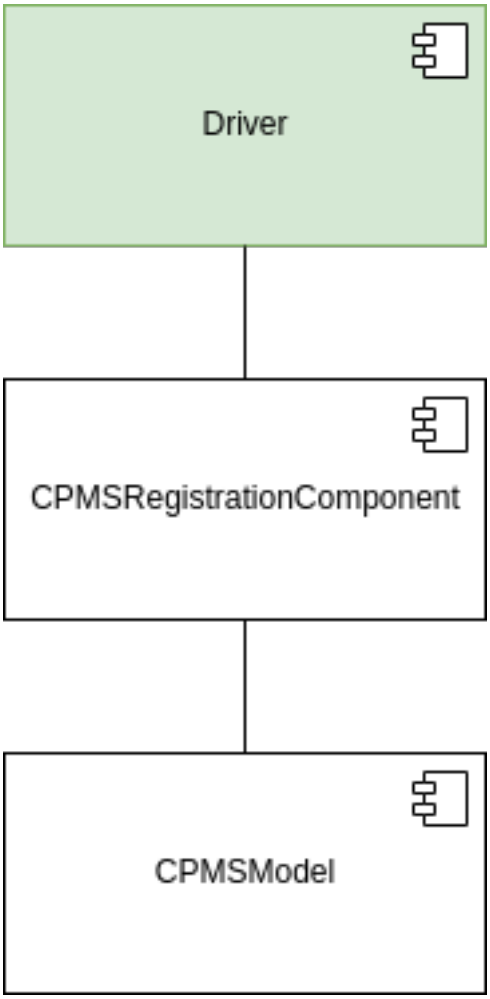

Figure 64: CPMSRegistrationComponent testing

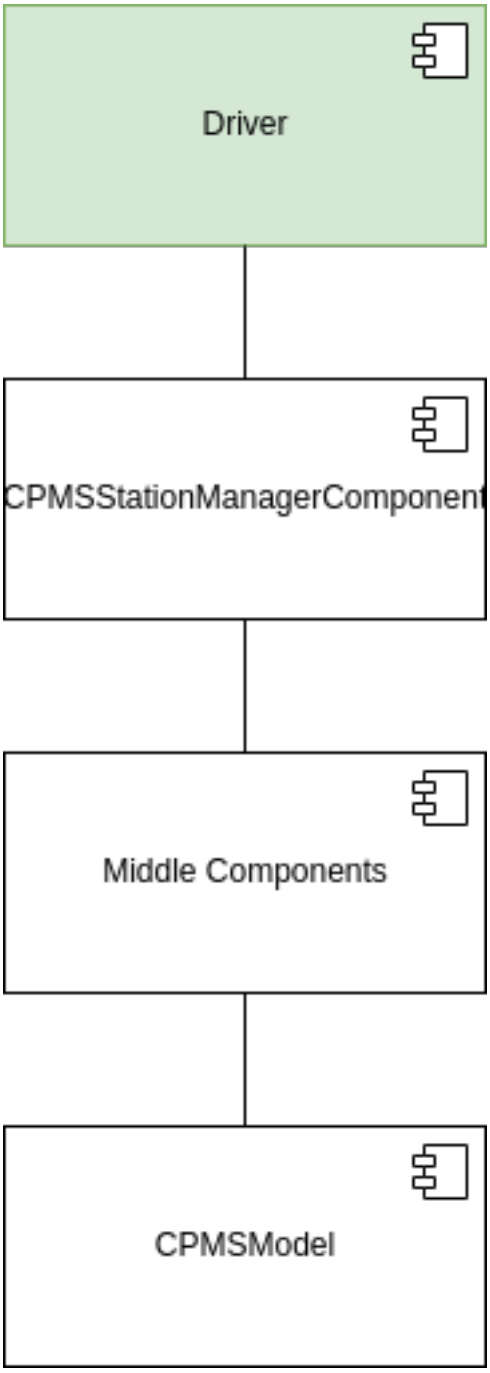

Figure 65: CPMSStationManagerComponent testing

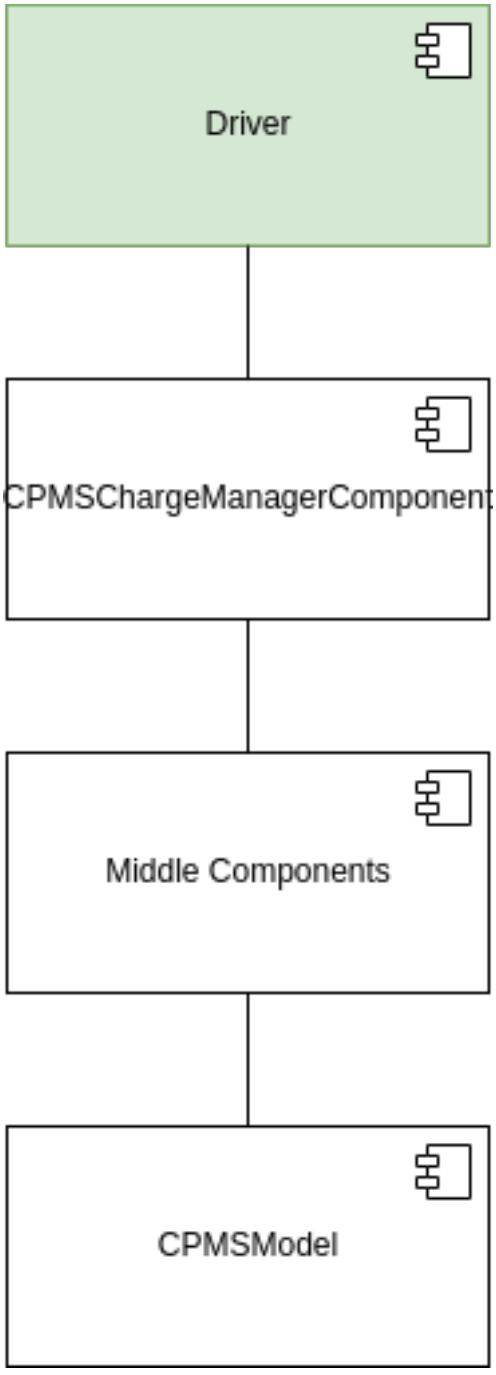

Figure 66: CPMSChargeManagerComponent testing

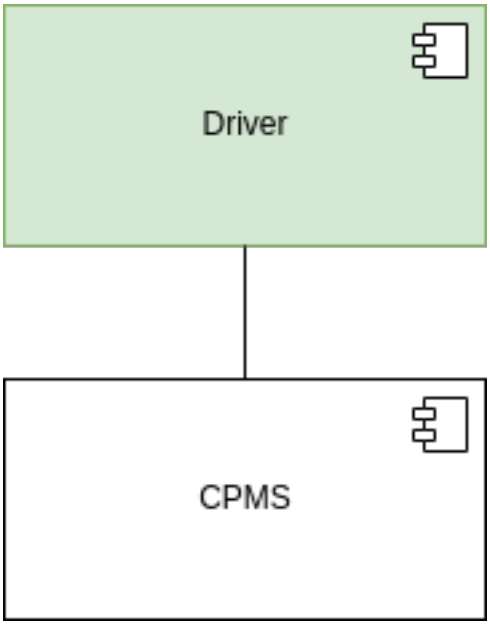

Figure 67: CPMS testing

6. Effort spent

- 24/12/2022: 15:20 - 16:20 Matteo
- 26/12/2022: 16:20 - 16:50 Matteo
- 27/12/2022: 22:00 - 23:15 Federico, Emilio, Matteo
- 28/12/2022: 16:30 - 18:00 Emilio
- 28/12/2022: 18:00 - 18:30 Matteo
- 29/12/2022: 11:30 - 13:30 Emilio
- 29/12/2022: 14:45 - 15:45 Emilio
- 29/12/2022: 16:00 - 17:30 Matteo
- 31/12/2022: 10:15 - 12:30 Emilio
- 31/12/2022: 16:00 - 17:40 Matteo
- 01/12/2023: 19:00 - 21:00 Matteo
- 02/01/2023: 12:15 - 14:00 Emilio
- 02/01/2023: 16:00 - 18:00 Emilio
- 02/01/2023: 22:00 - 23:30 Matteo
- 03/01/2023: 10:00 - 18:00 Federico
- 03/01/2023: 11:00 - 14:00 Emilio
- 03/01/2023: 16:15 - 17:45 Federico, Matteo, Emilio
- 03/01/2023: 21:30 - 22:20 Matteo
- 04/01/2023: 17:00 - 19:00 Emilio
- 04/01/2023: 21:00 - 22:30 Matteo
- 05/01/2023: 10:00 - 18:00 Federico
- 05/01/2023: 18:00 - 19:45 Matteo
- 06/01/2023: 10:00 - 18:00 Federico
- 06/01/2023: 11:30 - 14:00 Emilio
- 06/01/2023: 19:00 - 20:10 Matteo
- 07/01/2023: 11:30 - 14:00 Emilio
- 07/01/2023: 16:00 - 18:30 Emilio
- 07/01/2023: 09:00 - 12:00 Federico
- 07/01/2023: 14:00 - 19:30 Federico
- 07/01/2023: 20:00 - 22:30 Federico
- 07/01/2023: 20:45 - 24:00 Emilio
- 07/01/2023: 09:00 - 10:00 Matteo
- 07/01/2023: 15:00 - 23:00 Matteo
- 08/01/2023: 00:30 - 02:00 Emilio, Federico
- 08/01/2023: 09:30 - 20:30 Emilio, Federico, Matteo

References

[Redundancy in load balancers] https://www.digitalocean.com/community/tutorials/what-is-load-balancing

[Redundancy in firewalls] https://www.checkpoint.com/cyber-hub/network-security/what-is-firewall/high-availability-ha-firewall/

[Redundancy in VPN servers] https://docs.aws.amazon.com/vpn/latest/s2svpn/vpn-redundant-connection.html

[Redundancy in Databases] https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls

[Floating IP] https://www.digitalocean.com/blog/floating-ips-start-architecting-your-applications-for-high-availability

[The most used linux distro] https://truelist.co/blog/linux-statistics/

[The most used relational DBMS] https://www.c-sharpcorner.com/article/what-are-the-most-popular-relational-databases/

[OpenVPN official website] https://openvpn.net/

[The bridge pattern] https://refactoring.guru/design-patterns/bridge

[The DAO pattern] https://en.wikipedia.org/wiki/Data_access_object

[The MVC pattern] https://it.wikipedia.org/wiki/Model-view-controller

[The multi-tier pattern] https://en.wikipedia.org/wiki/Multitier_architecture

[The command pattern] https://it.wikipedia.org/wiki/Command_pattern

[Black and white box testing] https://www.geeksforgeeks.org/differences-between-black-box-testing-vs-white-box-testing/