AY 2022/2023

Politecnico di Milano

DD: Design Document

Alessandro Pignati Federico Sarrocco Alessandro Vacca

Professor
Matteo Camilli

Version 1.1
January 8, 2023

Contents

1 Introduction

eMall (e-Mobility for All) is an easy-to-use application which intent is to help the user to recharge their electric vehicle in order to reduce our carbon footprint. Users need an application whose main intent is to plan the charging process of an electric vehicle, thus reducing the interference and constraints on our daily schedule. Charging Point Operators need an application to manage their charging stations.

1.1 Purpose

The aim of the product is to simplify the process of electric vehicle charging, improving the customers experience. Moreover, the Charging Point Operators processes will be facilitated. The experience will be enhanced because many aspects of the electric vehicle charging will be integrated, and they will be located within a single service.
The purpose of this document is to provide an exhausting explanation about eMall, focusing in particular on the architecture that will be adopted, the modules of the system and their interfaces. Furthermore, a runtime view of the core functionalities of the product are provided, accompanied by some detailed interactions diagrams that show the message exchanging between the components. Finally, there are mentions about the implementation, testing and integration processes.

1.2 Scope

The application provides different actors that will use the application, which are customers and CPOs.

Customers can access the mobile application in order to see the nearby charging stations and to book a charge, selecting the type of socket. Moreover, they can see all the booking's details and unlock the correspondent socket and start the charging. Finally, they can pay directly on the application, if they want. They can also change the vehicle with a new one.

CPOs can access the web application in order to see the managed charging stations and they can set special offers and change the costs of the different sockets. They can manage the charging batteries and set their policies. Finally they see the different DSOs for every station and they can choose if they want to acquire energy from them.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Client-side scripting:** it is performed to generate a code that can run on the client end (browser) without needing the server side processing.

- **Code On Demand:** in distributed computing, it is any technology that sends executable software code from a server computer to a client computer upon request from the client's software.
- **Middleware:** in distributed applications, it represents the software that enables communication and management of data.
- **RESTful:** it's a software architectural style that defines a set of constraints to be used for creating Web services.
- **e-Mobility Service Providers:** The companies that offer the service of charging at different stations.
- **Charging Point Operators:** The companies that manage charging stations (one or more).
- **Distribution System Operators:** The charging station's energy providers.
- **Charging Point Management System:** The single CPO's system to manage all the IT infrastructure.
- **Tier:** it is a row or layer in a series of similarly arranged objects. In computer programming, the parts of a program can be distributed among several tiers, each located in a different computer in a network.
- **Module 1:** this represents the eMSP subsystem explained in the RASD document.
- **Module 2:** this represents the CPMS subsystem explained in the RASD document.

1.3.2 Acronyms

- **eMall:** e-Mobility for all
- **eMSP:** e-Mobility Service Provider
- **CPO:** Charging Point Operator
- **DSO:** Distribution System Operator
- **CPMS:** Charging Point Management System
- **API:** Application Programming Interface, it indicates on demand procedure which supply a specific task.
- **DBMS:** Database Management System.
- **DD:** Design Document
- **HTTPS:** Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP). It is used for secure communication over a computer network, and is widely adopted on the Internet.

- **MVP:** Minimum Viable Product, it is a version of a product with just enough features to be usable by early customers who can then provide feedback for future product development.
- **RASD:** Requirements Analysis and Specification Document
- **S2B:** Software to Be, it is the one designed in this document and not yet implemented.
- **TLS:** Transport Layer Security, it is a protocol which aims primarily to provide privacy and data integrity between two or more communicating computer applications

1.3.3 Abbreviations

- **Gn:** Goal number n
- **Dn:** Domain assumption number n
- **Rn:** Requirement number n

1.4 Revision History

- January 8, 2022: version 1.0, initial release
- January 8, 2022: version 1.1, updated runtime views (latest release)

1.5 Reference Documents

- Requirements Analysis Specification Document (RASD)
- UML official specification: `https://www.omg.org/spec/UML/`

1.6 Document Structure

- **Section 1: Introduction**
  This section offers a brief description of the document that will be presented, with all the definitions, acronyms and abbreviations that will be found reading it.
- **Section 2: Architectural Design**
  This section is addressed to the developer team and offers a more detailed description of the architecture of the system. The first part describes the chosen paradigm and the overall split of the system into several layers. Furthermore, an high-level description of the system is provided, together with a presentation of the modules composing its nodes. Finally, there is a concrete description of the tiers forming the S2B.

- **Section 3: User Interface Design**
  This section is useful for graphical designers of the S2B and contains several mockups of the application, together with some charts useful to understand the correct flow of execution of it. The presented mockups refers to the client-side experience.

- **Section 4: Requirements Traceability**
  This section acts as a bridge between the RASD and DD document, providing a complete mapping of the requirements and goals described in the RASD to the logical modules presented in this document.

- **Section 5: Implementation, Integration and Test Plan**
  The last section is again addressed to the developer team and describes the procedures followed for implementing, testing and integrating the components of our S2B. There will be a detailed description of the core functionalities of it, together with a complete report about how to implement and test them.

2 Architectural Design

2.1 Overview: high-level components and interactions

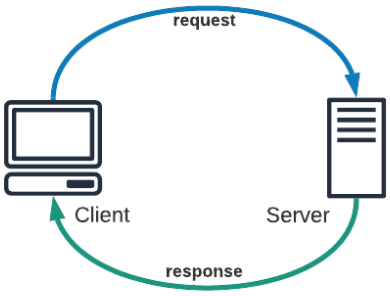


Figure 1: Client-Server paradigm

As figure 1 represents, the system is a distributed application which follows the common known client-server paradigm.

In particular, there are two different types of client-server interactions, because the product has 2 modules that need to fulfill different goals for different actors.

Since Module 1 will feature a mobile application, that will contain an internal database in order to make it less dependent from the server. This aspect makes it a more of a thick client.

Module 2, on the other hand, will feature a *Web Application*, which is by definition a thin client, because of its total dependency from the server. This type of client does not contain the application business logic, but only the presentation layer.

In both cases the server is *fat* and contains all the data management and business logic.

In this section the architecture will be described in an easy way, justifying all the choices for adopted patterns.

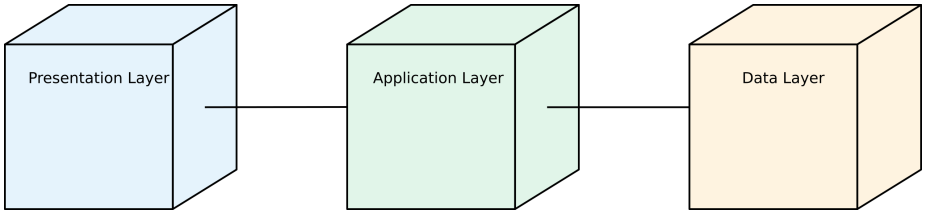


Figure 2: Three layers application

In figure 2 the three S2B layers are shown, which respectively are:

- **Presentation Layer:** it manages the presentation logic and, consequently, all the interactions with the end user. This is also called *rendering layer*.
- **Application (Logic) Layer:** it manages the business functions that the S2B must provide.
- **Data Layer:** it manages the safe storage and the relative access to data.

As shown in the high level representation of figure 3 the S2B is divided into three layers that are physically separated by installing them on different tiers. A tier is a physical (or a set of) machine, each of them with its own computational power.

The application described in this document is composed by four tiers.

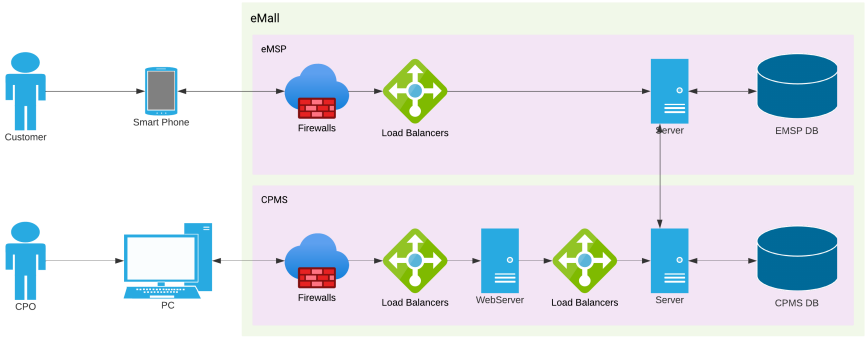


Figure 3: Architecture of the application

2.2 Component view

In this section there is an high-level analysis of the main components and their subcomponents. Main interfaces interactions between components are also provided.

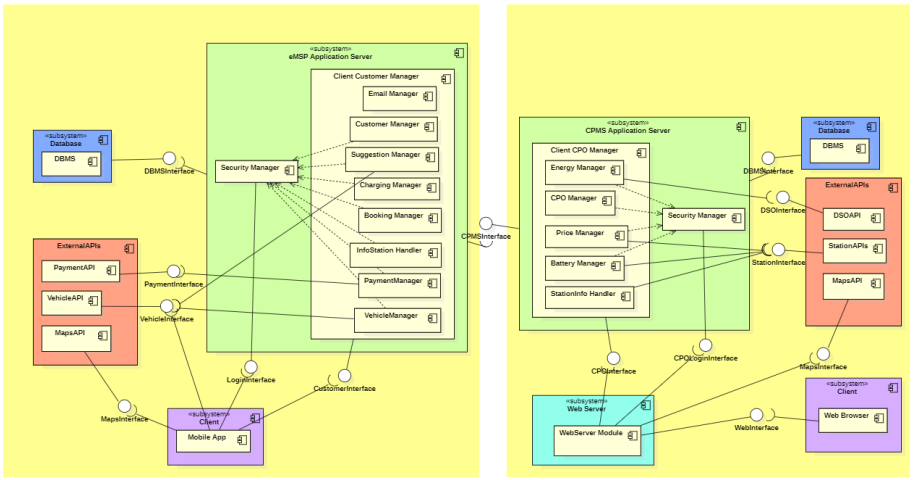


Figure 4: Component Diagram

eMSP's components:

- **Client Customer Manager**
  This module handles all the requests made by the client. At the beginning, when the client is not logged in, the module offers (through the customer manager) a LoginInterface which permits to execute a sign up or sign in operation. Once the client has logged in, the module shows only the CustomerInterface which the customer will exploit.

- **Email Manager**
  This component handles the email notifications, such as the sign up and forgot password ones.

- **Customer Manager**
  This module contains all the features in order to manage the customer side. In fact, it includes the log in and sign up manager.

- **Suggestion Manager**
  This module manages the push notifications. Based on the information aquired thanks to the Vehicle Manager, it creates suggested bookings and shows all the details about them. It allows also to confirm a suggested booking on the application.

- **Charging Manager**
  This module manages the charging process operations, such as the socket unlocking. It also shows the remaining time to end a charge and the charging state (in charge/ finished).

- **Booking Manager**
  This component manages the booking process. It aims to handle the sockets' booking and shows the details of the different bookings.

- **InfoStation Handler**
  This component handles the information about the charging stations. So it shows information about sockets types and related prices and special offers, sockets availabity and specifications about the charging stations.

- **Payment Manager**
  This component's aim is to handle the customer payment. It uses a PaymentAPI in order to process the task.

- **Vehicle Manager**
  This component's aim is to get the customer's vehicle information, such as the position, the battery percentage and the calendar.

- **Security Manager**
  Finally, this components handles all the security issues of the S2B. In fact, its aim is to authenticate and authorize requests, relying on the token provided from the client (since it should be a REST application). If a request is not authenticated, it takes the user to a login page; otherwise it simply replies with an unauthorized state message.

CPMS's components:

- **Client CPO Manager**
  This module handles all the requests made by the client. At the beginning, when the client is not logged in, the module offers (through the CPO manager) a LoginInterface which permits to execute a sign in operation. Once the client has logged in, the module shows only the CPOInterface which the CPO will exploit.

- **Energy Manager**
  This module's aim is to handle the energy options. It also allows to select a DSO and aquire energy from it. On the other hand allows to set some operations such as the auto-mode.

- **CPO Manager**
  This module contains all the features in order to manage the user side. It includes the log in manager.

- **Price Manager**
  This module in needed to set special offers on socket's types in selected charging stations. Moreover it is used to set socket prices.

- **Battery Manager**
  This component manages the battery in the charging stations. It handles the battery policy change.

- **StationInfo Handler**
  This component handles the information about the charging stations. So it shows information about sockets types and related prices and special offers, sockets availabity and specifications about the charging stations.

- **Security Manager**
  Finally, this components handles all the security issues of the S2B. In fact, its aim is to authenticate and authorize requests, relying on the token provided from the client (since it should be a REST application). If a request is not authenticated, it takes the user to a login page; otherwise it simply replies with an unauthorized state message.

2.3 Deployment view

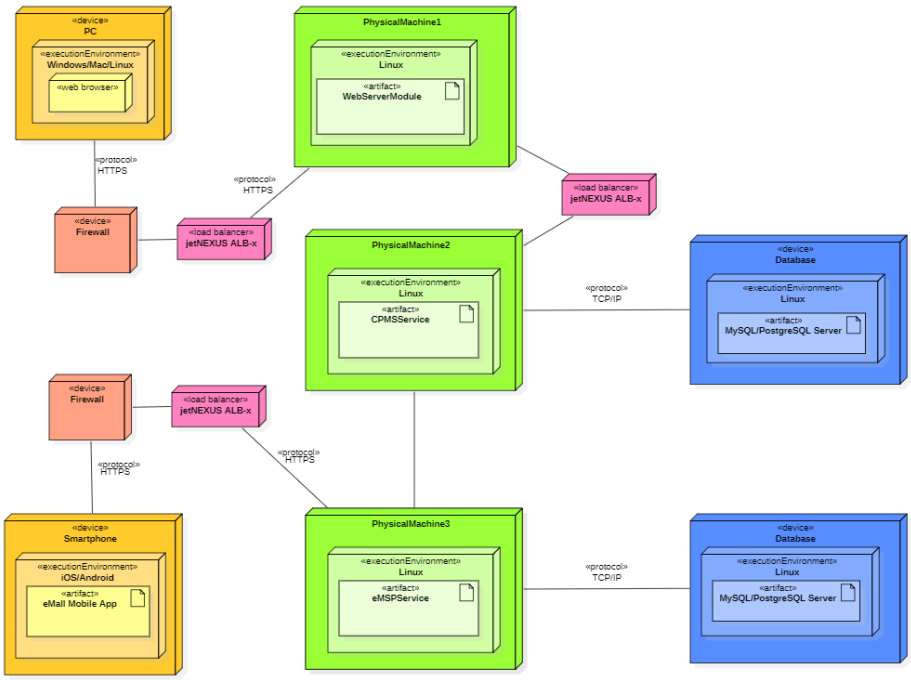


Figure 5: Deployment Diagram

The deployment diagram in figure5 shows the needed components for a correct system behavior and the protocols to communicate. As shown in the above image, firewalls and load balancers manage the data stream from the devices to the servers. First of all, firewalls are in charge of filtering packets received from the Internet. Then, the packets pass through the load balancer, where the workload is distributed among the available resources to increase capacity and

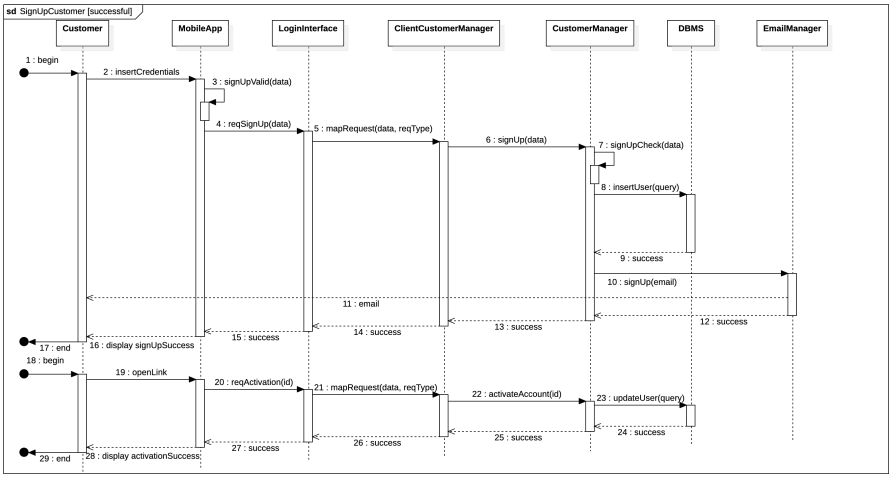
reliability. Each device has its own Operating System where the software runs. The tiers in the image are the following:

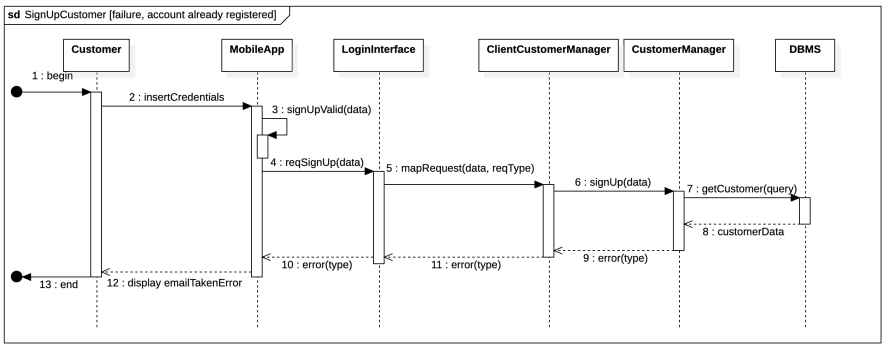
- **Tier 1:** it is the client machine, which can be a computer with a web browser (running, for example, on Windows 10 OS) or the downloadable mobile application (available on both Apple's store and Google's Store).
- **Tier 2:** it includes the replicated web servers, which do not execute any business logic, but simply receive requests from the client, route them to the application servers and serve an HTML file fo the client, which will build the page thanks to client-side scripting. They also append the styling logic of the page (CSS sheets, JS sheets, etc.).
- **Tier 3:** it contains the application servers, which run the core functionalities of the S2B. The whole application layer is mapped into this tier, which communicates to the client tier through APIs, which will be used from the web servers (in case of webapp) and the native application (in case of mobile app download). Furthermore, it communicates to the data tier through the DBMS gateway.
- **Tier 4:** it is composed by the DBMS servers. They store the data and execute actions on it, according to the instruction given by the application servers.

2.4 Runtime view

As stated previously, Module 1 (eMSP subsystem) is a mobile app while Module 2 (CPMS subsystem) is a web app. All the following diagrams represent the runtime view from the mobile apps perspective, in order to increase readability, the web apps perspective isnt shown as it only differs from the other one by using the web servers module before calling the Application Server.
Every interface uses the REST API through HTTP GET or/and POST calls.

2.4.1 Signup Customer



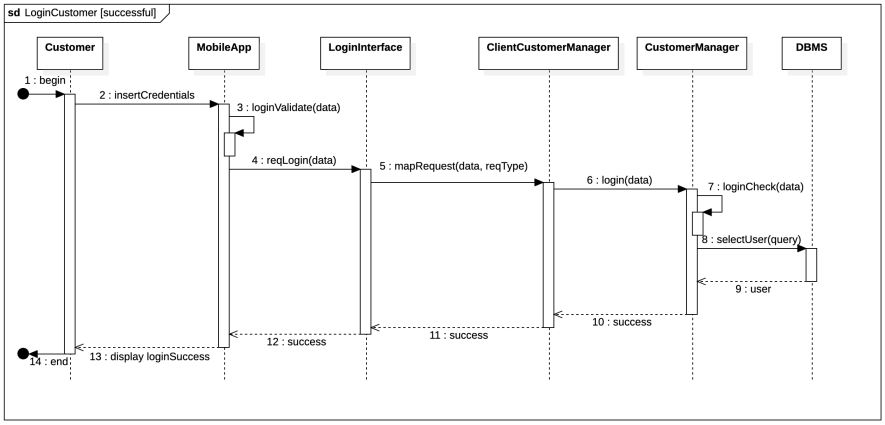


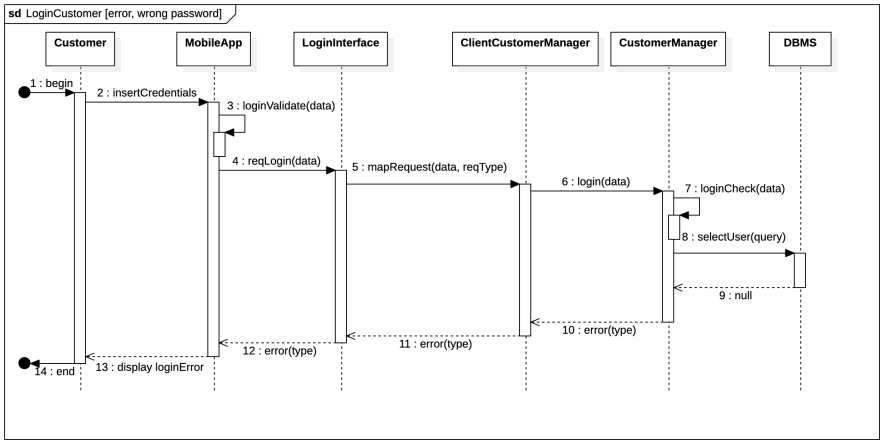
The diagram above represents the process of signing up a Customer. There two possible situations:

- Customer performs a correct registration process;
- Customer performs a wrong registration process (such as the account is already registered or the email in not valid).

The Customer begins by inserting his registration credentials in the app. Afterwards the app sends the request to the ClientCustomerManager through the LoginInterface, accessing later the CustomerManager.
The CustomerManager checks for the correctness of the received information and if correct it passes it ultimately to the DBMS. Then an email is sent to the Customer by the EmailManager. If the CustomerManager checks fail, the user is sent a specific error message related to his issue.

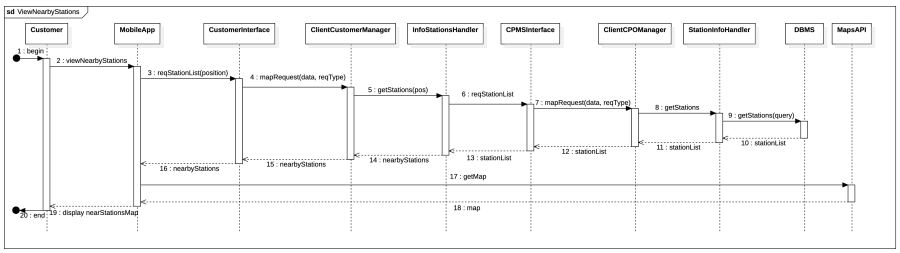
2.4.2 Login Customer





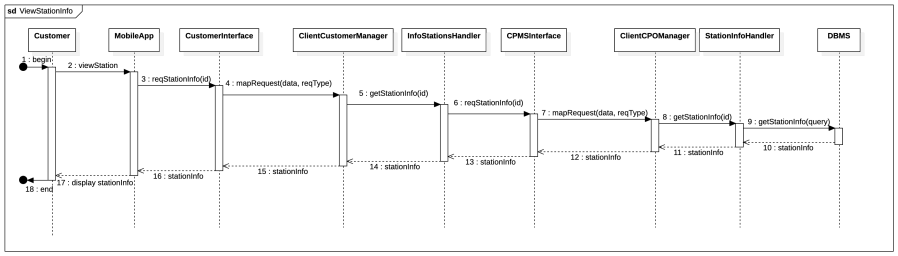
The Customer inputs his login credentials and then the MobileApp sends the request, through the LoginInterface to the ClientCustomerManager, which forwards them to the CustomerManager that checks for their correctness and then interrogates the DBMS and searches for the credentials. If the returned value is null the ClientManager sends a specific error message and the login fails, otherwise a 200 response status code is sent to the MobileApp.

2.4.3 View Nearby Stations



To access this view it is necessary to have performed the login before (2.4.2). The MobileApp will automatically request the list of stations that are nearby the Customer, through the CustomerInterface to the ClientCustomerManager, which forwards them to the InfoStationsHandler. The InfoStationsHandler will then request the list of stations by contacting the CPMS subsystem (Module 2). The CPMS subsystem functions can be accessed by contacting the CPMSInterface. In this case, the ClientCPOManager will forward the request to the StationInfoHandler, which will ultimately access the CPMS' DBMS. The InfoStationsHandler will then filter the far stations and will return to the MobileApp the list of close stations, that will be used in conjunction with the MapsAPI by the app, in order display an interactive map to the customer to the Customer.

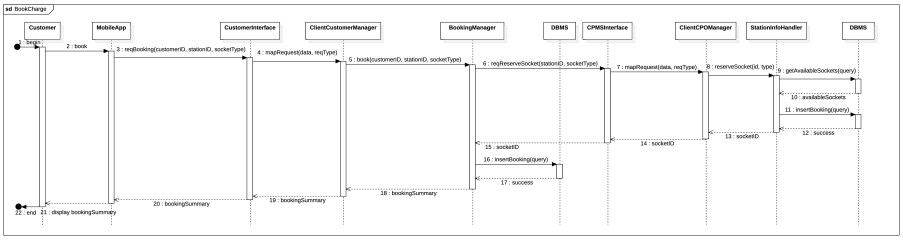
2.4.4 View Information about a Station



To access this view it is necessary to have performed the login before (2.4.2). This view is accessed when the "Book" button from the View Nearby Stations View (2.4.3) is selected. The MobileApp will send the booking request of the selected charging station through the CustomerInterface to the ClientCustomerManager, which forwards them to the InfoStationsHandler. The InfoStationsHandler will then request the list of stations by contacting the CPMS subsystem (Module 2). The CPMS subsystem functions can be accessed by contacting the CPMSInterface. In this case, the ClientCPOManager will forward the request

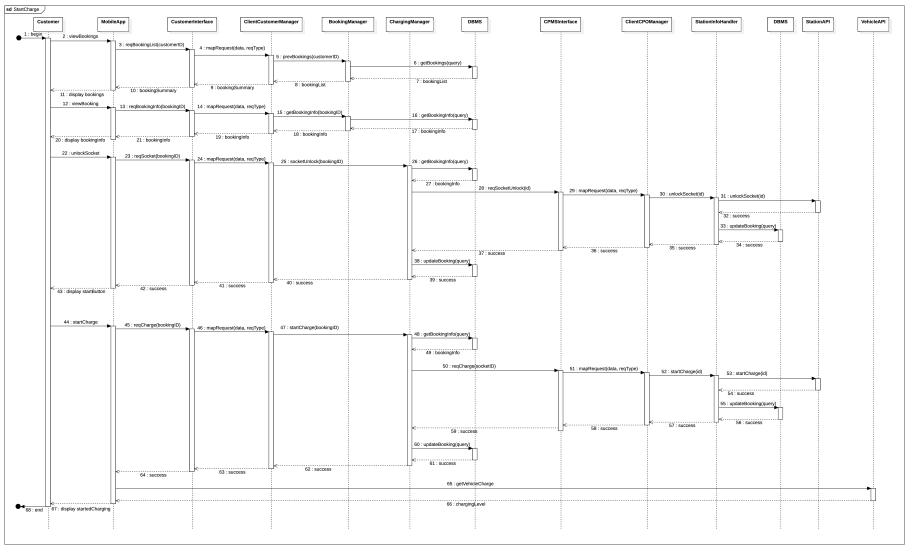
to the StationInfoHandler, which will ultimately access the CPMS' DBMS. The MobileApp will then receive the information of the selected station, that will be displayed to the Customer.

2.4.5 Book a Charge



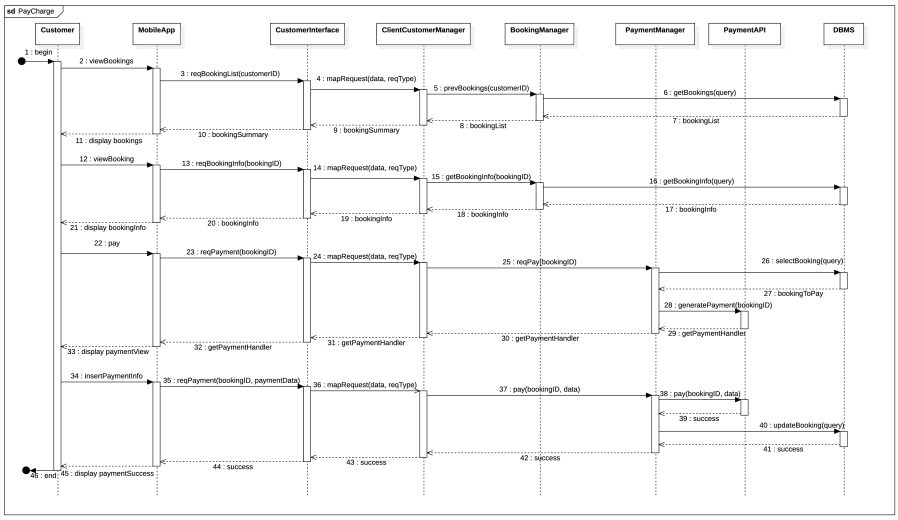
To access this view it is necessary to have performed the login before (2.4.2). This view is accessed when the "Book" button from the View Information about a Station (2.4.3) is selected. The MobileApp will request the booking of the selected type of socket from the current charging station through the CustomerInterface to the ClientCustomerManager, which forwards the request to the BookingManager. The BookingManager will then request the socket by contacting the CPMS subsystem (Module 2). The CPMS subsystem functions can be accessed by contacting the CPMSInterface. In this case, the ClientCPOManager will forward the request to the StationInfoHandler, which will ultimately access the CPMS' DBMS, where a list of the available sockets will be retrieved, and then the socket will be picked and inserted. If successful, the CPMS subsytem will return the 200 status to the BookingManager, that will store all the booking's information on the eMSP's DBMS. The MobileApp will then display a message stating that the charge has started.

2.4.6 Start Charging



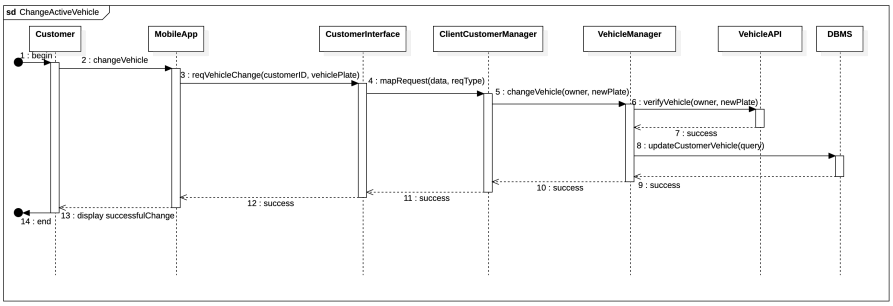
To access this view it is necessary to have performed the login before (2.4.2). The Customer must access the Bookings View. The MobileApp will then request the list of performed booking of the Customer through the CustomerInterface to the ClientCustomerManager, which forwards the request to the BookingManager. Finally, the BookingManager will access the DBMS in order to retrieve the list of previous bookings, that will then be displayed to the Customer. When a specific booking is selected, the App will contact the BookingManger again similarly as before, only this time to request the specific information of a booking, that will then be displayed to the user. Now, the Customer can select the "Unlock socket" button, that will trigger the ChargingManager similarly to the BookingManager previously explained, but then will contact the CPMS subsystem (Module 2) in order to unlock the socket through the StationInfoHandler and the external StationAPI, that will perform the physical socket unlock, and the status will be recorded on both the eMSP and the CPMS DBMS. In the end, the Customer view will be updated, and now the "Start Charging" button will appear only when the physical connection between the vehicle and the socket has been established. When the button is pressed and the requirements are met, the ChargingManager will then request the start of the charging by contacting the CPMS subsystem. The StationInfoHandler, will request the start of the charge to the Station API, and then will ultimately access the CPMS' DBMS to store the state update. The CPMS subsytem will return the specific socketID to the BookingManager, that will store all the booking's information on the eMSP's DBMS. The MobileApp will then receive the summary of the booking, that will be displayed to the Customer.

2.4.7 Pay for the Charging Service



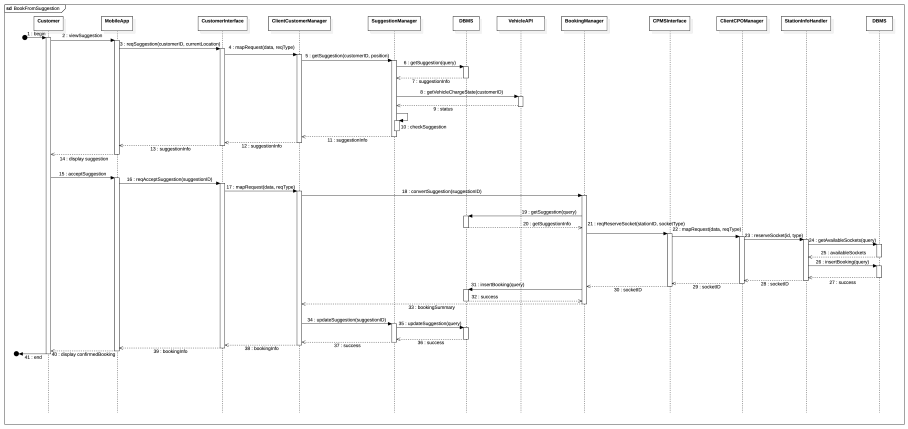
To access this view it is necessary to have performed the login before (2.4.2). The Customer must access the Bookings View and a specific booking in a similarly to Start Charging(2.4.6). The MobileApp will then request the generation of a payment ticked through the CustomerInterface to the ClientCustomerManager, which forwards the request to the PaymentManager that will access the PaymentAPI. The Customer will be presented with a web page inside the app where he can pay with whatever method the payment processor offer. It is important to say that this web template will be provided and mantained by the external payment service. After the Customer has input his information, the MobileApp will contact the PaymentManager again, that will forward the information to the PaymentAPI for processing. If correct, the 200 OK status code is returned.

2.5 Change active vehicle for suggestions



To access this view it is necessary to have performed the login before (2.4.2). The MobileApp, after the Customer has input his new license plate, will contact the VehicleManager through the CustomerInterface and the ClientCustomerManager. The VehicleManager will then verify the ownership and the plate itself through the external VehicleAPI. If successful, the updated is stored on the DBMS and the 200 OK status code is returned to the MobileApp.

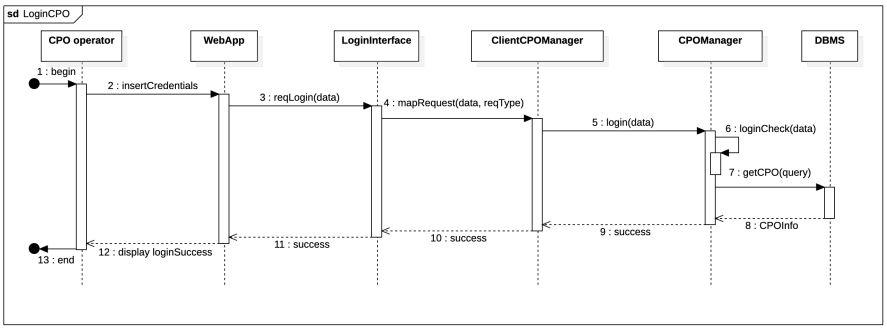
2.5.1 Book a charge from a suggestion



To access this view it is necessary to have performed the login before (2.4.2). When a Customer clicks on a received push notification, the MobileApp will contact the SuggestionManager that will retrieve the suggestion from the DBMS, and then the vehicle charge state will be retrieved from the CPMS. The SuggestionManager will then check in the suggestion is still valid. If it is still valid, the

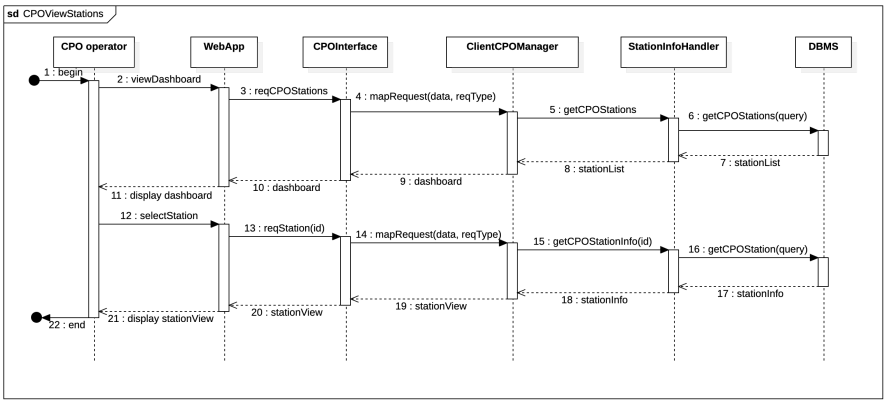
suggestionInfo will be sent to the MobileApp and displayed to the user. If the Customer accepts the suggestion, the BookingManager will be contacted, and it will retrieve all the suggestion info from the DBMS, and then will perform the booking of a charge in the same way as the Book a Charge view (2.4.5).

2.5.2 Login CPO operator



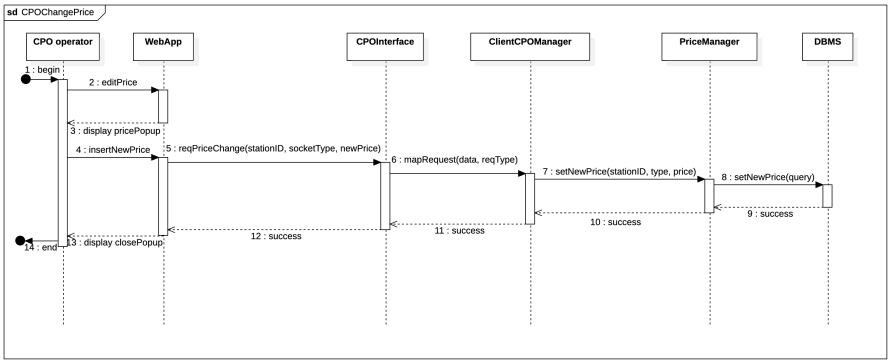
The CPO operator inputs his login credentials and then the WebApp sends the request, through the CPOLoginInterface to the ClientCPOManager, which forwards them to the CPOManager that checks for their correctness and then interrogates the DBMS and searches for the credentials. If the returned value is null the ClientManager sends a specific error message and the login fails, otherwise a 200 response status code is sent to the WebApp.

2.5.3 View CPO dashboard



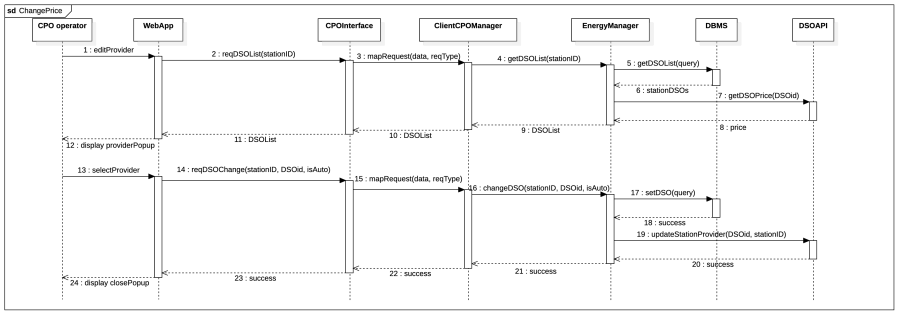
To access this view it is necessary to have performed the login before (2.5.2). After the CPO operator login, the WebApp automatically request the dashboard where all the CPO's station are present. The WebApp contacts the ClientCPOManager through the CPOInterface. Then the StationInfoHandler is accessed, that in turn queries the DBMS to retrieve the list of charging stations associated with the CPO. The CPO main dashboard is then shown to the operator. If an operator wants to select the single station's dashboard, he can click the "modify" button that triggers the request from the WebApp to the StationInfoHandler, in the same way as before. The CPO operator can then view the station's current status and settings.

2.5.4 Modify a station current charging price



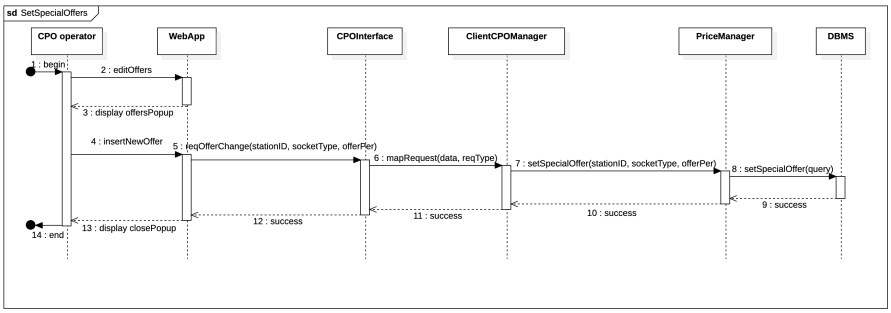
To access this view it is necessary to have performed the login before (2.5.2). The CPO operator must access the single station dashboard. In order to edit the station's current charging price, the "Update price" button must be selected. Then the WebApp will show a popup where the operator can enter the new price. When a new price is entered, the WebApp contacts the ClientCPOManager through the CPOInterface, and finally the PriceManager is accessed. The PriceManager will store the station's new price in the CPMS' DBMS. The WebApp will be sent a 200 OK status code that will automatically close the popup.

2.5.5 Modify a station current energy provider (DSO)



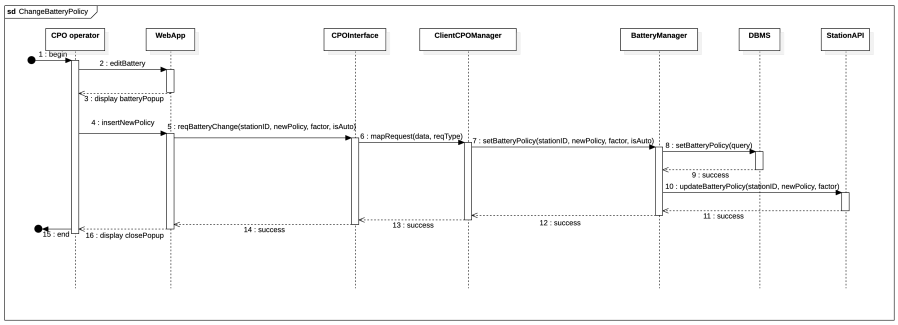
To access this view it is necessary to have performed the login before (2.5.2). The CPO operator must access the single station dashboard. In order to edit the station's current energy provider (DSO), the "Update provider" button must be selected. Then the WebApp will contact the ClientCPOManager through the CPOInterface, and finally the EnergyManager is accessed. The EnergyManager will query the DBMS in order to retrieve the station's DSO list. The module will then query the external DSOAPI to get their current energy price. Then the DSO list will be sent to the WebApp, that will show a popup with all the available DSOs with their current prices.Finally, the operator can select either the automatic option or a manual DSO to acquire energy from. When a new option is selected, the WebApp contacts the EnergyManager again in the same way, but this time it updates the current station provider in the DBMS, and then it contacts the external DSOAPI to propagate the change. The WebApp is returned a 200 OK status code that automatically closes the popup.

2.5.6 Modify a station special offers



To access this view it is necessary to have performed the login before (2.5.2). The CPO operator must access the single station dashboard. In order to edit the station's current special offers, the "Update special offer" button must be selected. Every special offer is bound to a type of charging socket. After the button is selected, the WebApp will show a popup where the operator can enter the new special offer for the type of socket. When a new offer is entered, the WebApp contacts the ClientCPOManager through the CPOInterface, and finally the PriceManager is accessed. The PriceManager will store the station's new offer in the CPMS' DBMS. The WebApp will be sent a 200 OK status code that will automatically close the popup.

2.5.7 Modify a station battery policy



To access this view it is necessary to have performed the login before (2.5.2). The CPO operator must access the single station dashboard. Another requirement lies in the fact that the station has to have a battery installed. In order to edit the station's current battery policy, the "Update battery policy" button must be selected. After the button is selected, the WebApp will show a popup where the operator can enter the new battery policy of the station. Finally, the operator can select either the automatic option or a manual battery policy. When a new option is selected, the WebApp contacts the ClientCPOManager through the CPOInterface. The next step is to access the BatteryManger that will update the battery policy on the DBMS and then apply the policy to the station by contacting the external StationAPI. The WebApp is returned a 200 OK status code that automatically closes the popup.

2.6 Component interface

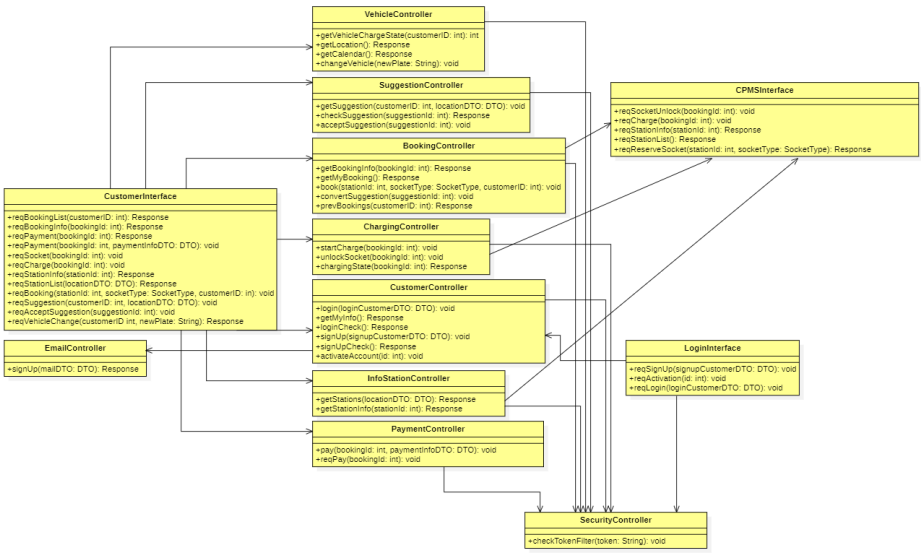


Figure 6: Component Interface 1

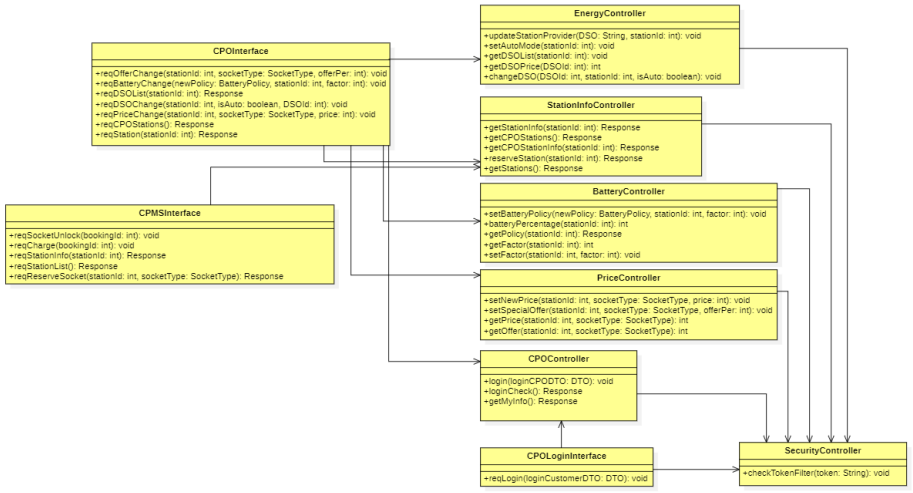


Figure 7: Component Interface 2

eMSP interfaces description:

- **CustomerInterface**: this is the main interface to exchange data between the mobile-application and the eMSP.
- **LoginInterface**: this interface is used to handle the login, sign-up, and account activation of the customer. It's used to comunicate between the mobile-app and the eMSP.
- **SecurityController**: eMSP's internal interface, used to manage the authorization of the user operations (operations of the CustomerInterface and LoginInterface). It's used by each component to validate the request token.
- **VehicleController**: eMSP's internal interface, used to handle the request and data exchanged with the VehicleApi. It offers all the methods related with the user-vehicle.
- **SuggestionController**: eMSP's internal interface, used to comunicate with the SuggestionHandler, it exposes methods to find candidate suggestions, to accept suggestions.
- **BookingController**: eMSP's internal interface, used to manage customer's bookings. It exposes methods to create a new booking, get info about a booking, get all booking (olds and current).
- **ChargingController**: eMSP's internal interface, used to manage the vehicle charging process. For example the startCharge method is used to start a booked charge.
- **CustomerController**: eMSP's internal interface, it is linked with the LoginInterface. It is used to internally perform login-signup operations.
- **InfoStationController**: eMSP's internal interface, it is used to retrive station informations. The method getStationInfo for example returns all the info about a station: socket availability, charging price
- **PaymentController**: eMSP's internal interface, it is used to manage the payment phase. The reqPay method is used to send a payment request. The pay method is used to communicate the payment informations and pay.

CPMS interfaces description:

- **CPMSInterface**: this is the main interface to exchange data between the eMSP and the CPMS. It can be retrived a list of stations, or info about a single station. It can be requested to unlock a socket or request a reservation for a booking.
- **CPOInterface**: this is the main interface to exchange data between the CPMS and the CPO-WebServer. It expose all the functionalities to manage a station, for example to set special offers, set charging price...

- **CPOLoginInterface**: this is the interface to handle CPO's login. It is an interface between the CPMS and the CPO-WebServer.
- **EnergyController**: CPMS's internal interface, it is used to manage the DSO providers. To retrive the DSO prices. To set the acquisition of energy to automatic mode.
- **StationInfoController**: CPMS's internal interface, mainly used to retrive stations' information. It is also used to make a reservation for a socket.
- **BatteryController**: CPMS's internal interface, it is used to control the station's batteries. It exposes methods to set the battery policy (mantain, charge or discharge), set discharging factor, or retrive information.
- **PriceController**: CPMS's internal interface, it is used to set and retrive charging prices and to set special offers.
- **CPOController**: CPMS's internal interface, it is linked with the CPOLoginInterface. It is used to internally perform login operations.
- **SecurityController**: CPMS's internal interface, used to manage the authorization of the user operations (operations of the CustomerInterface and LoginInterface). It's used by each component to validate the request token.

2.7 Selected architectural styles and patterns

2.7.1 Four-tiered architecture

We chose this architecture for many reasons:

- **Flexibility:** Once the interfaces of the S2B are defined, then the interior logic is independent from outside. Single components can therefore be implemented in parallel. And can be modified without affecting the system.
- **Scalability:** An application divided on several tiers guarantees that the approach of scaling the architecture is adopted only for the most critical components. The result obtained maximizes the performance but also minimizes the costs.
- **Load Distribution:** the presence of several application servers, preceded by a load balancer, guarantees an acceptable division of requests. Otherwise, the presence of a single node means that node can become overrequested, sending the entire system down.

2.7.2 RESTful Architecture

The restful application will be adopted both on web and mobile side. This architecture is based on the stateless principle, in which the server does not contain any information about the state of client, that is managed directly on client side.

An useful property of this architecture is the *code on demand* one, which permits sending some code snippets from the server to the client, and then make the client executing them locally (usually in the web browser). This behavior guarantees less computational load on the server and also a dynamic attitude of the service.

The application is then intended to be developed through *client side scripting*, which means that all requests and update of the page are made on client side. This behavior also improves the user experience, and prevent refreshing the page each time an action is made.

2.7.3 Model View Controller (MVC)

Model–View–Controller (usually known as MVC) is a software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user.

These three components are:

- **Model:** the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.
- **View:** any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
- **Controller:** accepts input and converts it to commands for the model or view.

2.8 Other design decisions

2.8.1 Scale-Out

This method consists of cloning the nodes in which we expect to have a bottleneck in order to increase the general system scalability.

This choice leads to a higher deployment effort but also to a lower hardware upgrade cost when the limits are reached. In conclusion, the scale-out is a preferable road.

Once split, the system requires a load balancer in order to correctly redirects the incoming requests to the node with the lowest workload.

2.8.2 Thin and thick client and fat server

The thin client will be the web application.
This architecture consists of keeping as low information as possible on client side. It means that the business logic resides only on server side.
The minimum requirement of this choice is a stable connection between the parts; otherwise the application would not work as expected.
Of course, the main advantage of choosing this implementation style is that the client machine is not required to have an high computational power.

Instead, in the case of mobile application, the best choice is to save useful information on a local database, in order to avoid continuous requests to the server (less computational load) and also to keep information even when the Internet connection is not available. In this second case it is said to be a thick client.

2.8.3 Automatic-DSO-Energy-Source selection - Algorithm

When the CPMS system has the automatic DSO feature active, an algorithm to automatically choose a DSO is used. The choice of the energy source is based on the minimum price available (with the assumption that energy availability is not an issue). The DSO that offers the minimum price is choosen. Every 30min a scan for the minimum price DSO is performed and if a better DSO is find, the energy source is changed.

2.8.4 Automatic-Battery-Mode selection - Algorithm

When the CPMS system has the automatic battery mode active, an algorithm to automatically managed the battery operation mode (Discharge, Mantain, Recharge) is used. When the battery is in discharge mode, it means that the battery is discharging and the energy is flowing from the battery to the vehicles, when the battery is in mantain mode it is not used. The automatic battery mode uses the policy in figure 8. Once activated the automatic mode the battery state is mantain. Every time the battery state of charge (SoC) is less than 20 % it goes to the recharge state. The transition between mantain and discharge states are based on the station socket occupation: if there are many customers (>50 % of sockets occupied) the battery is set to discharge state, if insted the occupation becomes lower than 40 % (debouncing factor) the battery is deactivated.

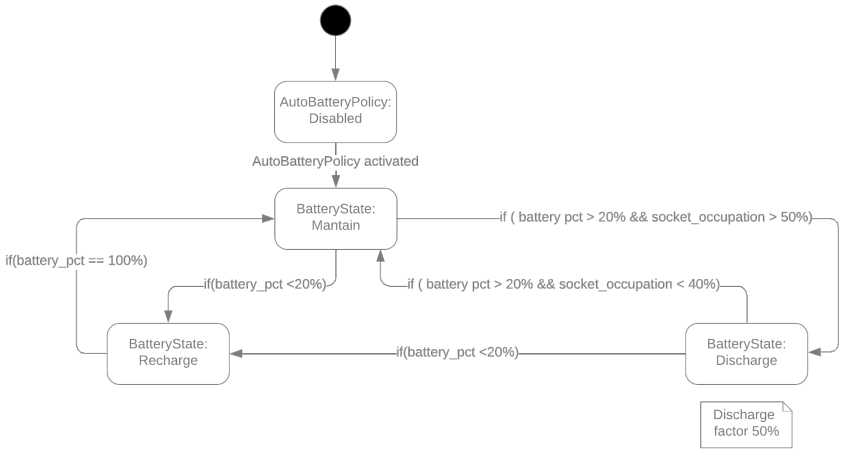


Figure 8: Automatic battery policy state machine

2.8.5 Send-suggestions - Algorithm

The eMSP sends booking suggestions to the customer's mobile-app. Suggestions are based on customer position, car's SoC, calendar. A basic algorithm will send suggestions only when the customer has at least the next 3 hours free, the battery SoC must be under the 60%. The application will not "spam" noification, so a debouncing time of 2h is set between one suggestion and another. The suggested charging station is simply the nearest to the customer's location.

3 User interface design

The goal of this section is to show the design of the main screens of the Customer-app and CPO-administration-app. It's described the flow of screens, according to the main functionalities for which the application was intended.

3.1 Customer Mobile App

3.1.1 Sign-Up

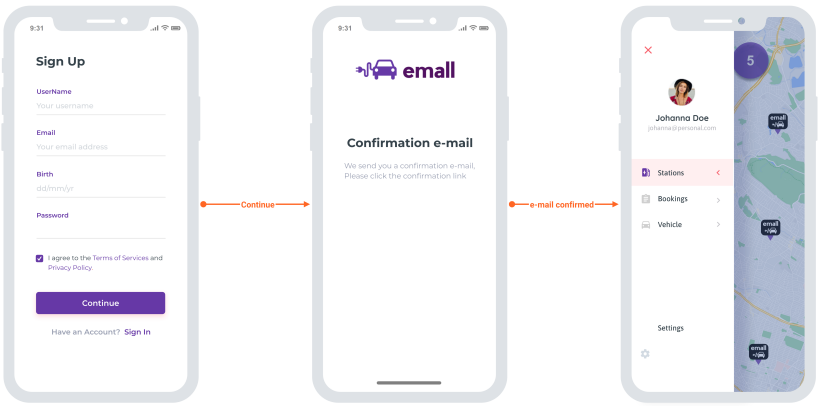


Figure 9: Sign Up screens

These mobile-app's sceens represent the graphical flow during the Sign-up procedure. Once user-data have been inserted and the continue button is pressed the mobile app sends a request to signUp to the eMSP. All the communication between the app and the eMSP use the eMSP's LoginInterface. After that the user is requested to press the confirmation link sent in the mail, and after a successful confirmation the user is redirected to the mobile-app's main page.

3.1.2 Log-In

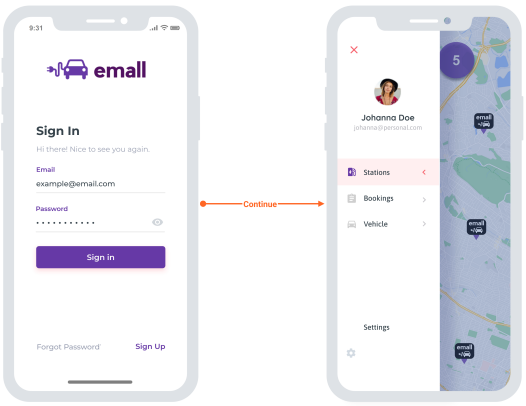


Figure 10: Log-in screens

These mobile-app's sceens represent the graphical flow during the Log-in procedure. The customer inserts his personal data and then presses Sign-in. This will call, using the eMSP's LoginInterface, the signIn methon on the application server.

3.1.3 Stations view

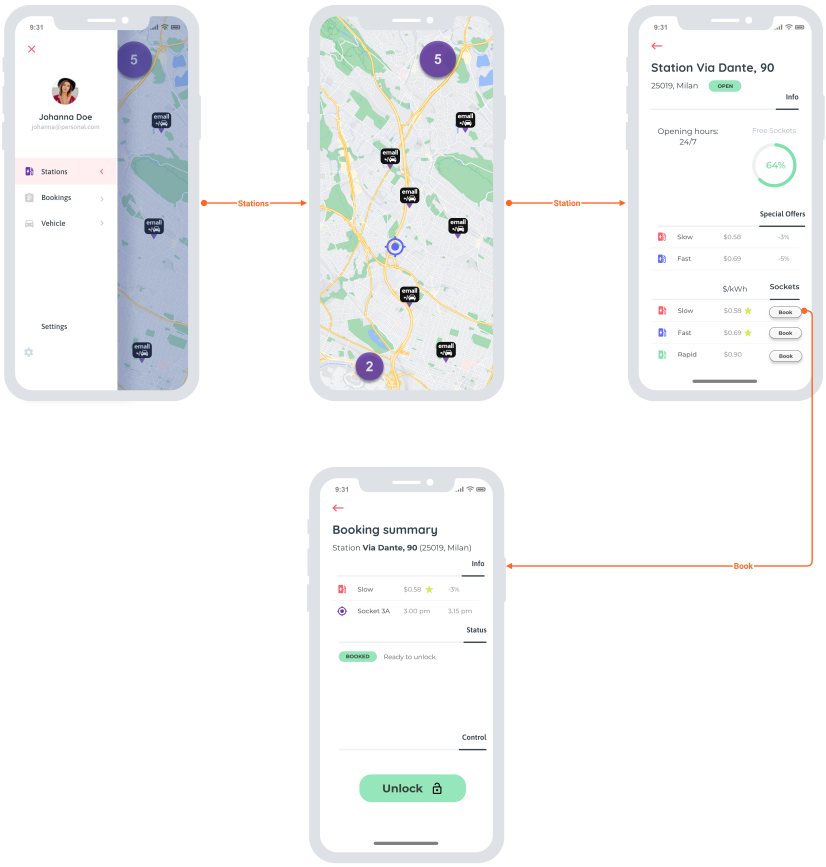


Figure 11: View of the nearby stations and booking screen

These mobile-app's sceens represent the graphical flow during stations searching. If the customer clicks on a station a request to the eMSP to get the station info is sent. If the user wants to book a charge he pushes the book button. The book button sends a request to the eMSP to book a socket.

3.1.4 Bookings

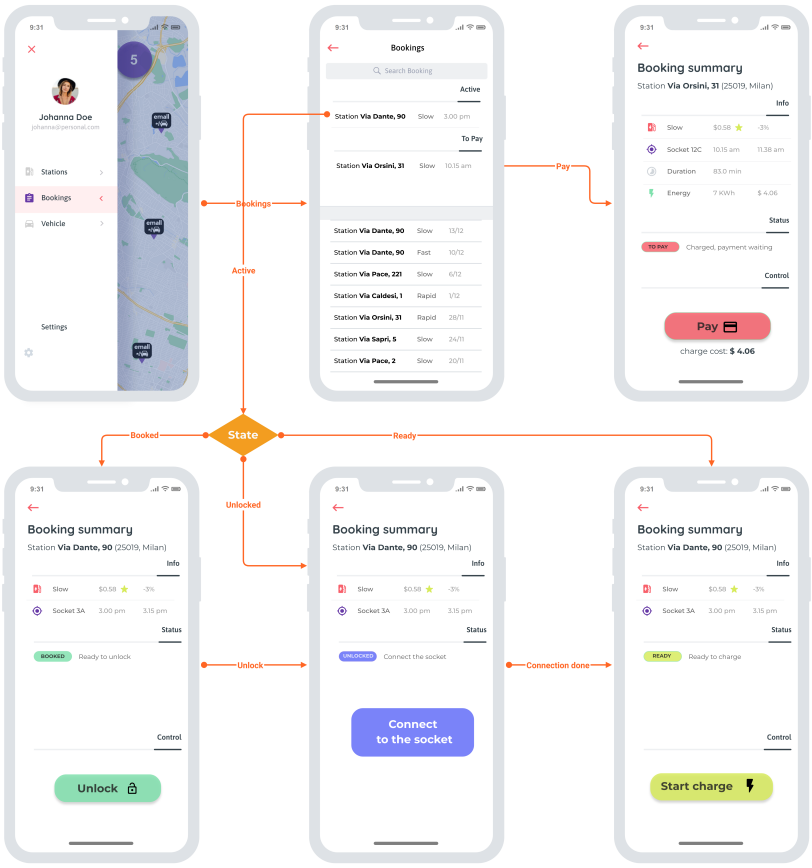


Figure 12: Booking screens

Using the booking button the customer can see all his bookings (old and current booking). The customer clicking on a specific booking on the list can perform several actions. The customer can pay a booking. The customer can unlock the socket pushing the unlock button, after having unlocked the socket he is taken to the screen "connect to the socket". Finally he can push the start charge button. All the requests and responses from and to the eMSP and the mobile-app uses the eMSP's customerInterface.

3.1.5 Vehicle

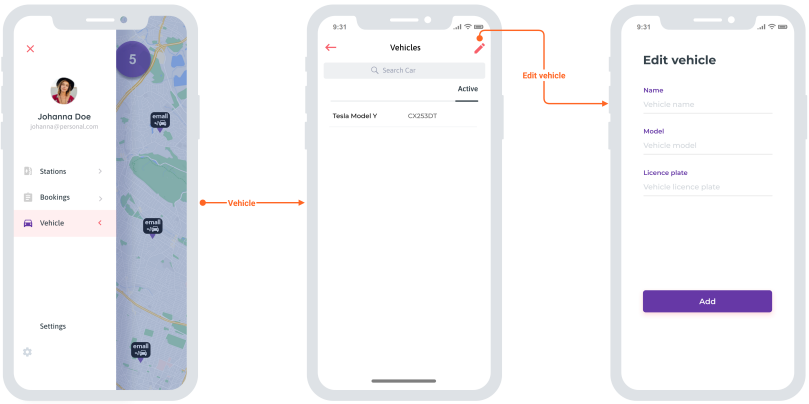


Figure 13: Vehicle change screens

These mobile-app's sceens represent the graphical flow during the default vehicle change. If a user wants to change its vehicle, he must enter the new informations and click the button. All the requests and responses from and to the eMSP and the mobile-app uses the eMSP's customerInterface.

3.1.6 Suggestion

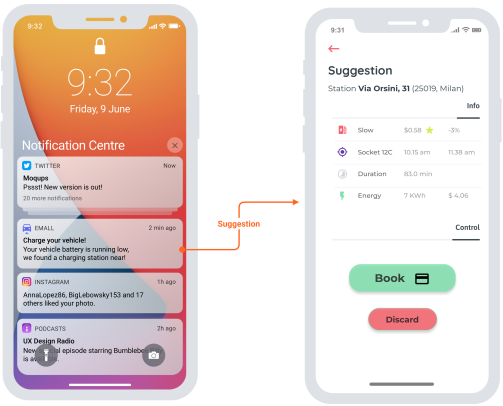


Figure 14: Push notification, suggestion to charge your vehicle

If the eMSP finds a suggestion, a push notification is sent to the app, suggesting to book a charge.

3.2 CPO Administration Web-App

3.2.1 CPO login

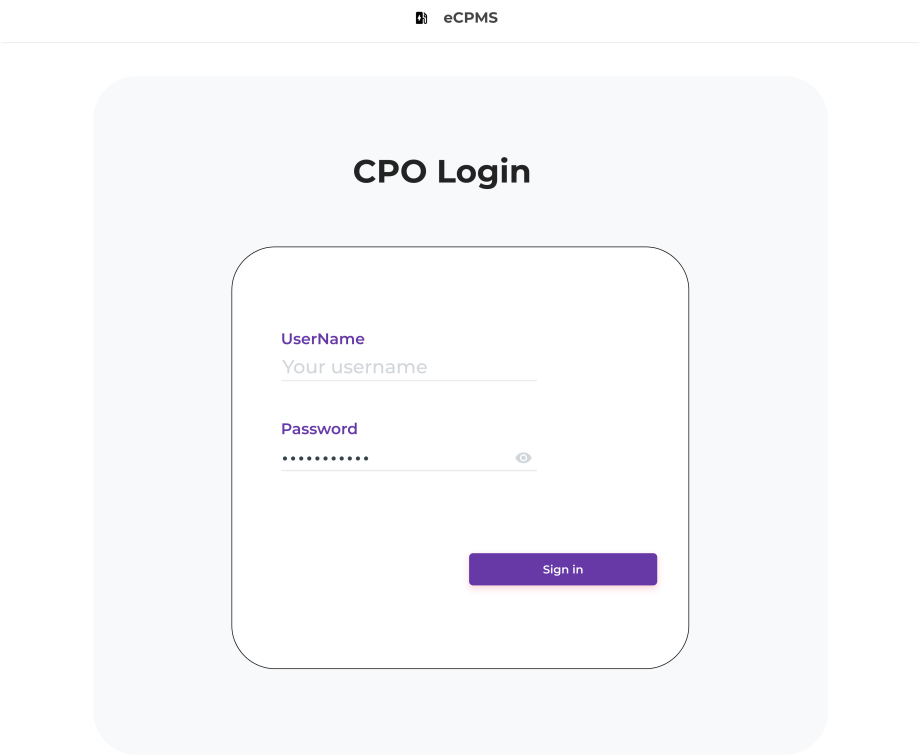


Figure 15: CPO login screen

With this screen the CPO can log in to the CPMS system. The requests and responses to retrive and set data use the CPMS's CPOLoginInterface

3.2.2 Stations

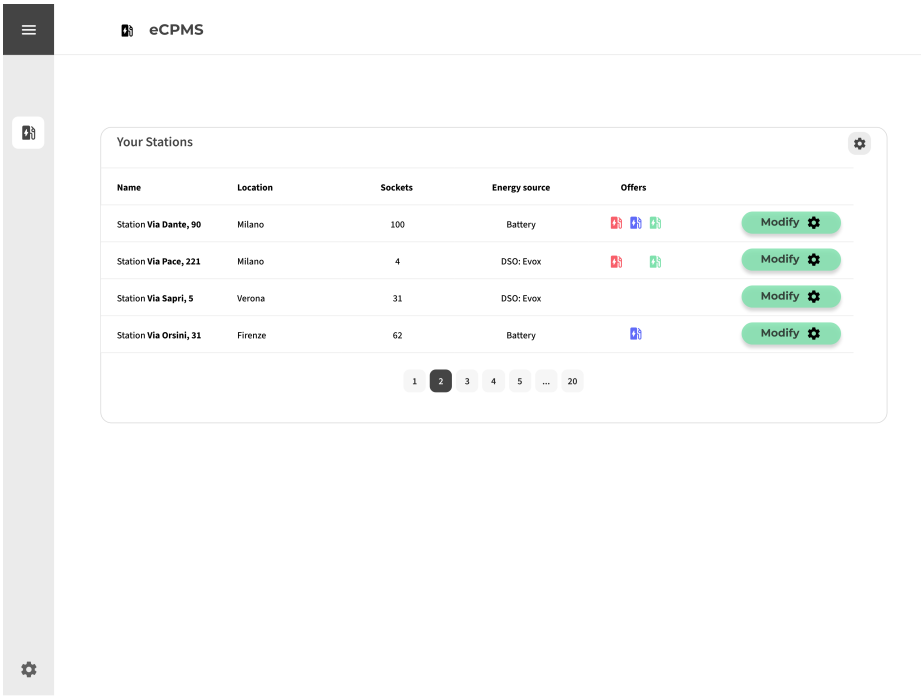


Figure 16: Dashboard with a list CPO's stations

This screen is a dashboard with all the stations of a CPO, basic information are shown. To edit the configuration (price, battery policy,...) of a specific station the CPO user have to press the Modify Button. The requests and response to retrive and set data use the CPMS's CPOInterface

3.2.3 Station-Dashboard

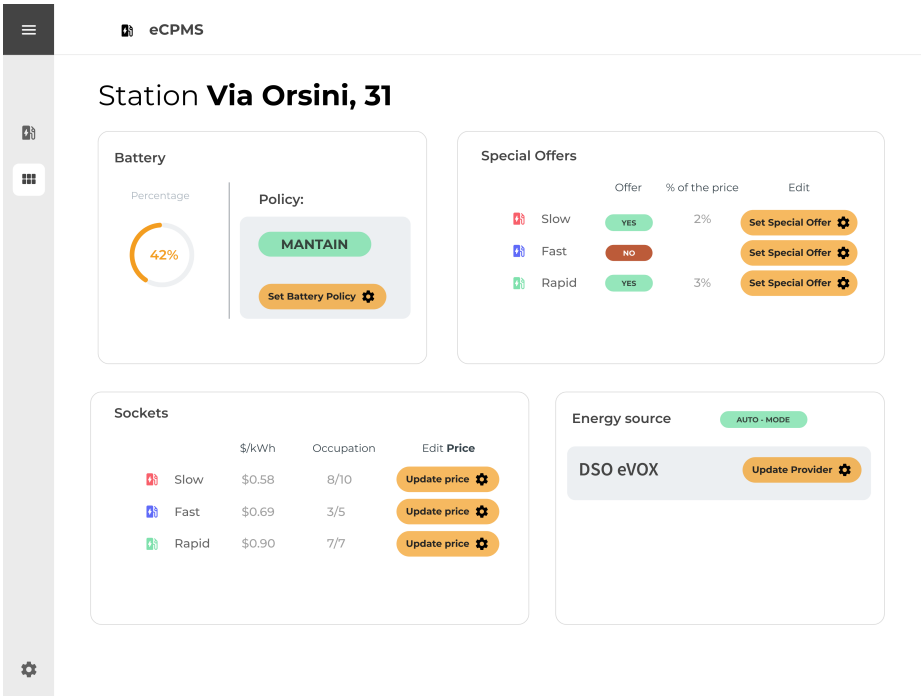


Figure 17: Dashboard to manage a single charging station

This screen is a dashboard used to modify station specific parameters. It's made up of 4 blocks. Orange buttons are used to modify the settings, so for example to update the DSO provider, set special offers... Once an orange button is pressed a pop-up appears to modify the parameters. The requests and response to retrive and set data use the CPMS's CPOInterface

4 Requirements traceability

This section will shown the traceability between requirements and modules described in component diagram.

- **R1:** The system allows users to sign-up.
  - **Customer Manager**: in order to sign up.
  - **Email Manager**: in order to sign up.
- **R2:** The system allows customers to log-in.
  - **Customer Manager**: in order to sign in.
  - **Email Manager**: in order to deal with forgot passwords.
- **R3:** The system allows customers to view nearby station.
  - **InfoStation Handler**: it simply returns all stations nearby.
- **R4:** The system allows customer to select a charging station to view its information.
  - **InfoStation Handler**
- **R5:** The system shows available socket for each charging station.
  - **InfoStation Manager**
- **R6:** The system allows customer to book an available socket in a charging station.
  - **Booking Manager**: in order to make a booking
- **R7:** The system allows customers to view personal bookings.
  - **Booking Manager**
- **R8:** The system allows customers to select a specific booking.
  - **Booking Manager**
- **R9:** The system allows to start a charge for a booking when the vehicle is connected and ready to charge.
  - **Charging Manager**
- **R10:** The system shows to the user the predicted charging time when the charge is started.
  - **Charging Manager**: this module display the time left to end the charge
- **R11:** The system show the binary charging state (in charge/finished).

  - **Charging Manager**

- **R12:** The system allows customers to pay a booked charge.
  - **Payment Manager**
- **R13:** The system periodically searches possible nearby station to charge based on the customer car's battery SoC (<50%), customer's location, customer's calendar.
  - **Suggestion Manager**: in order to create possible notifications.
  - **Vehicle Manager**: in order to retrieve vehicle's informations.
- **R14:** The system sends a push notification if a possible nearby station is found and no suggestions were sent in the previous 1h.
  - **Suggestion Manager**: in order to create possible notifications.
  - **StationInfo Handler**: in order to retrieve informations about nearby stations to display in the notification.
- **R15:** The system allows CPO to log-in.
  - **CPO Manager**: in order to sign in.
- **R16:** The system allows CPO to view all their the charging stations
  - **StationInfo Handler**
- **R17:** The system allows CPO to view information about a single charging station
  - **StationInfo Handler**
- **R18:** The system allows CPO to select energy acquisition battery policy (auto, charge, discharge) for a selected charging stations, if the station has batteries
  - **Battery Manager**: in order to manage batteries e set policies
- **R19:** The system allows CPO to select an energy provider (DSO) or set the selection to auto-mode
  - **Energy Manager**: select DSO and the energy actions
- **R20:** The system allows CPO to set the charging prices for a selected charging station
  - **Price Manager**
- **R21:** The system allows CPO to set a special offer for a selected charging station and a specific charging type (slow, fast, rapid)

– **Price Manager**

Also the system attributes are guaranteed by the design choices explained in this document; more precisely:

- **Easy usability**: guaranteed through a very simple, minimal and intuitive user interface. Since the main target of the application is the customer-side, the experience is designed to be very simple. In fact, there are only a few buttons with clear and precise functionalities. On the other hand the CPO-side interface is very easy to use and has a user-friendly interface
- **Reliability and Availability**: accomplished through a replication of the running application in different clones, following the *scale-out* method. The physical nodes of the system would then work in parallel, avoiding system downtimes due to a failure. In fact, when a clone breaks down, there are others in parallel which can supply the requested service, but with some performance issues. The scope is to obtain at least 97% of availability for each tier (the total system would then have 97% of availability).
- **Security**: guaranteed through an encrypted communication between client and server. Connections moves on HTTPS protocol.
- **Cross Platform**: firstly obtained through a development of a web interface, which can be accessed from any type of connected device with a browser installed. The mobile application, instead, will be available only for iOS and Android customers and will be downloadable from App Store and Google Play.
- **Maintainability and Modularity:** the first is accomplished through the second, because the application will be divided in some small parts (called modules) which interacts each other to provide the requested service.

5 Implementation, integration and test plan

The eMall will be naturally divided as follows:

- eMsp: Application Server
- eMsp: Mobile App
- CPO: Application Server
- CPO: WebServer

The implementation logic will be Bottom up. In this way the integration testing will follow the implementation and less code will be required as stubs are not necessary.
All these macro-components can be developed in a parallel manner, but the main focus will be on the Application servers for both the CPO and the eMSP, as they are the most difficult and complex parts.
The building and the integration will be strongly related, so the integration and testing plans will strictly follow the implementation's one.
In the following table a mapping between functionalities, importance for the customer and implementation difficulty is carried on, so that we can better plan the module implementation order:

Functionality	**Module**	**Importance for customer**	**Difficulty**
Sign Up and Login	eMSP	High	Low
View the nearby charging stations	eMSP-CPMS	High	Medium
Charge booking	eMSP-CPMS	Medium	High
Remotely unlocking a socket	eMSP-CPMS	Medium	Medium
Remotely starting a charge	eMSP-CPMS	Medium	Medium
Notify of a finished charge	eMSP-CPMS	Low	Medium
Pay for the service	eMSP	Medium	High
Change active vehicle for the suggestions	eMSP	Low	High
Select the charging stations energy provider	CPMS	High	High
Select a battery policy for a charging station	CPMS	High	High
Select the charging stations charging cost	CPMS	High	High

Table 1: functionalities importance - difficulty mapping

As the importance of functionalities for the customer is mainly Medium-High, and considering that most of the functionalities are necessary for the correct working of the application, the implementation order will not follow the "Importance for the customer" factor. The implementation order will follow the

bottom up structure of the modules. Firstly the CPMS module will be developed, and then the eMSP as the eMSP strongly depends on the CPMS.

All the modules rely on the DBMSs, so this component must be the first one to be implemented. The following picture describes the implemetnation schedule of the main components of the CPMS and eMSP application servers:

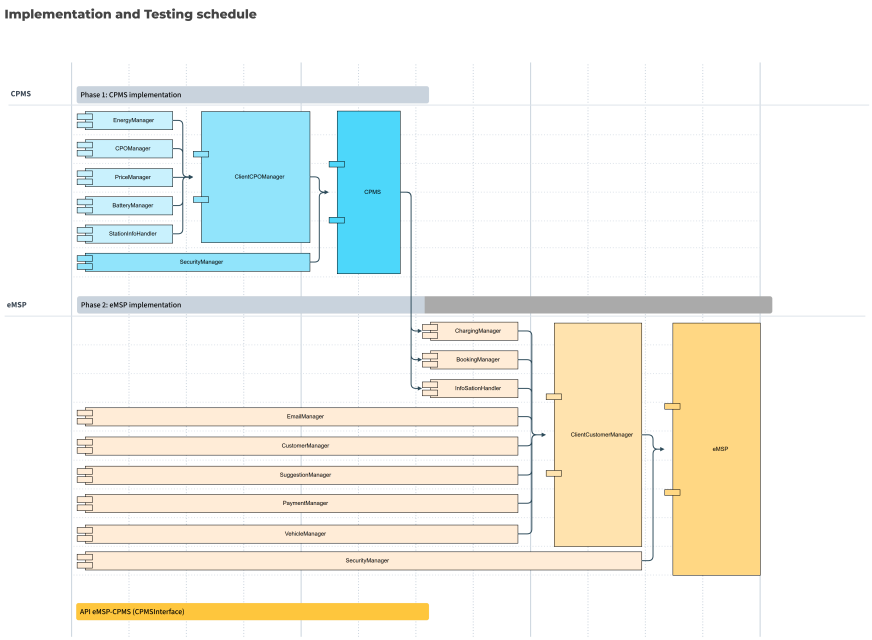


Figure 18: Implementation schedule of the main components of the CPMS and eMSP application servers, arrows represent precedence operator

The first subsystem to be developed is the CPMS, it's internal modules can be implemented in parallel. For each one of them unit testing must be carried out, the integration testing is done with the DBMS and the ExternalAPIs. After that, is composed the ClientCPOManager and the CPMS Application server, for both of them as soon as they are implemented unit testing is made, while integration testing will be done when the eMSP and or the WebServer will be implemented.

In the mean-while components of the eMSP Application server that are not related with the CPMS can be implemented and tested, both doing unit testing and integration testing with the DBMS, the PaymentAPI and the VehicleAPI.

The development of the communication API between CPMS and eMSP (CPMSInterface) can be carried out in parallel, it must be completed in order to develop and test the remaining modules of the eMSP.

Once the CPMS and the CPMSInterface are ready, the remaining components of the eMSP Application server are implemented. For them unit testing and integration testing are carried out. The integration testing uses the CPMSInterface to verify the correct operating functionalities.

When the two application server are fully implemented integration testing between them is done, a driver will test using their interfaces, their functionalities.

The eMSP's mobile app and the CPO's WebServer can be developed in parallel to the application servers.
Once the CPO's WebServer is ready,it is performed integration testing with the CPMS's Application Server to verify the overall CPMS system functionality..
Once the eMSP's mobile app is ready it is performed integration testing with the eMSP's Application Server to verify the overall eMSP system functionality.
If both the CPO's and the eMSP's subsytems are fully working the overall system is functioning properly.

It's not specified each specific unit tests, but the coverage must be of at least 80% of code lines.

6 Effort spent

Student	Time for S.1	S.2	Time for S.3	Time for S.4	Time for S.5
Alessandro Pignati	2h	14h	4h	1h	2h
Federico Sarrocco	3h	13h	4h	2h	3h
Alessandro Vacca	3h	11h	4h	2h	2h