Software Engineering 2 - Prof. Di Nitto Elisabetta
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

eMall - e-Mobility for All

DD
Design Document

January 8, 2023

Giovanni De Lucia (10700658)
Lorenzo Battiston (10618906)
Matteo Salvatore Currò (10940719)

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to give a more detailed view of the eMall - e-Mobility for All - system presented in the RASD, explaining architecture, components and their interaction, processes and algorithms that will satisfy the RASD requirements. Additionally, it includes instructions regarding the implementation, integration and testing plan. This document is intended to be a reference for the implementation of the system, and is aimed toward the developers, testers and project managers.

1.2 Scope

eMall is a system that allows EV - Electric Vehicle - Driver to plan efficiently their charging needs and CPO - Charging Point Operator - to be reached by EV Driver.
In particular, eMall allows EV Driver to search and then book a charge to a CP - Charging Point, at a specific time, pay the charge, start a charge and being notified when the charging process is completed.
The system allows CPO - Charging Point Operator - to smartly manage their charging points, choosing rates and energy sources.
The system consists of two subsystems: eMSP - electric mobility service provider - and CPMSs - charging point management systems. The former offer functionality to drivers, the latter offer management options to operators. One of the focuses of this document is the description of how these entities interact with each other. Instead, it is out of the scope the design of the physical CPs - Charging Points that are assumed to be already implemented and tested.
The eMSP will interact with the CPMS of multiple CPOs
For a more detailed description of the features that the system offers to end users, please refer to the RASD.
The architecture of the S2B is divided into three layers physically separated because installed on different tiers.
These layers are:

- Presentation Layer: it manages the presentation logic and all the interactions with the end users
- Business Logic Layer: it manages the application functions that the S2B provide
- Data Layer: it manages the safe storage and the access to data

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Tier - physical components or servers that make up a system
- Layer - a logical separation of components with related functionality
- S2B - System To Be, is the system we are designing
- EV Driver - Electric Vehicle Driver, people or entities who own an EV car and want to use the system for their charging needs
- EVCP - Electric Vehicle Charging Pool, is a station with multiple CPs
- CP - a synonym of EVSE - is a single charging column with multiple connectors
- Connector - charging socket that can be of different types (e.g. CCS2, Type2)
- Rate - the rate that the CPO decides to set for the CPs it manages. It contains a fixed part for parking and a variable part per kWh. Usually the rates are associated with a certain power (kW)
- OCPP - Open Charge Point Protocol [1] - is a protocol that dictates the communication between CPMS and a controlled CP to achieve smart charging functionalities

[1] OCPP Protocol

1.3.2 Acronyms

eMall	e-Mobility for All
RASD	Requirement Analysis and Specification Document
DD	Design Document
NFR	Non-Functional Requirement
EV	Electric Vehicle
CPO	Charging Point Operator
DSO	Distribution System Operator
CP	Charging Point
EVCP	Electric Vehicle Charging Pool
EVSE	Electric Vehicle Supply Equipment
CPMS	Charging Point Management System
eMSP	Electric Mobility Service Provider
SPA	Single Page Application
DBMS	Database Management System
CDN	Content Delivery Network
PWA	Progressive Web App
ER	Entity Relationship

1.3.3 Abbreviation

R$_x$	x-Functional Requirement

1.4 Revision history

Revised on	Version	Description
8-Jan-2023	1.0	Initial Release of the document
29-Jan-2023	2.0	Added Endpoints and minor database change according to the implementation

1.5 Reference Documents

- Requirement Analysis and Specification Document (referred to as “RASD” in the document)
- Assignment document A.Y. 2022/2023 (”Requirement Engineering and Design Project: goal, schedule and rules”)

1.6 Document Structure

This document is composed of seven sections:

- Introduction: This section provides an overview of the DD, including the scope of the project, definitions of key terms, references to other relevant documents, and an overview of the design

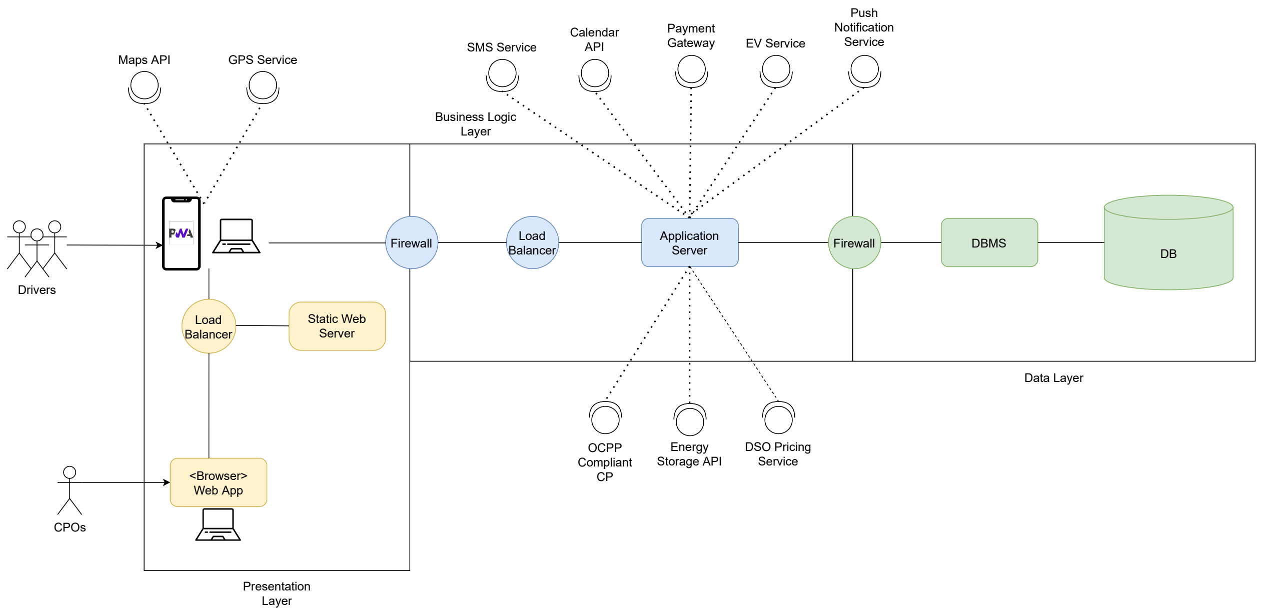
- Architectural Design: This section describes the high-level components and interactions of the system. It also includes a component view, a deployment view, a runtime view, and descriptions of selected architectural styles and patterns
- User Interface Design: This section outlines the design of the user interface (UI) of the system, including user experience (UX) flowcharts
- Requirements Traceability: This section provides a mapping between the requirements specified in the RASD and the components specified in the DD
- Implementation, Integration, and Test Plan: This section outlines the plan for implementing, integrating, and testing the system, including the order in which subsystems and components will be implemented
- Effort Spent: This section provides information on the effort spent on the design process
- References: This section includes a list of any references cited in the DD

2 Architectural Design

The purpose of this section was to present and analyze the architecture of the S2B system in a top-down manner. We first introduced the overall architecture and then provided a diagram of the system's components, focusing on the eMSP and CPMS subcomponents. Next, we used an ER diagram to describe the system's logical data and presented the system's deployment view, including the layers and tiers involved. We also used sequence diagrams to depict important runtime views and class diagrams to analyze the component interfaces. Finally, we discussed the architectural design choices and the reasons behind them.

2.1 Overview

The figure shown below represents a high-level description of the components which make up the System. In this document the presentation layer and the Client (e.g. the Browser) will be referred to as the Frontend, while the Application Layer and the Data Layer will be referred to as the Backend.



A web interface will be used to access the service. A single page application (SPA) will be developed for drivers and CPOs to use the system. An SPA is a good choice for this type of application because it allows for a lot of interaction without the need for frequent page reloads, which can provide a faster and more seamless user experience. The overall architecture of the system is divided into different layers, with the application servers interacting with a database management system and using APIs to retrieve and store data. The application servers are designed to be stateless according to REST standards, and the system includes firewalls to enhance security.

2.2 Component view

In this section we show the components of the S2B and their relationships. The following sections will explain the interaction between interfaces and details on each method of interfaces with REST endpoints, if any.

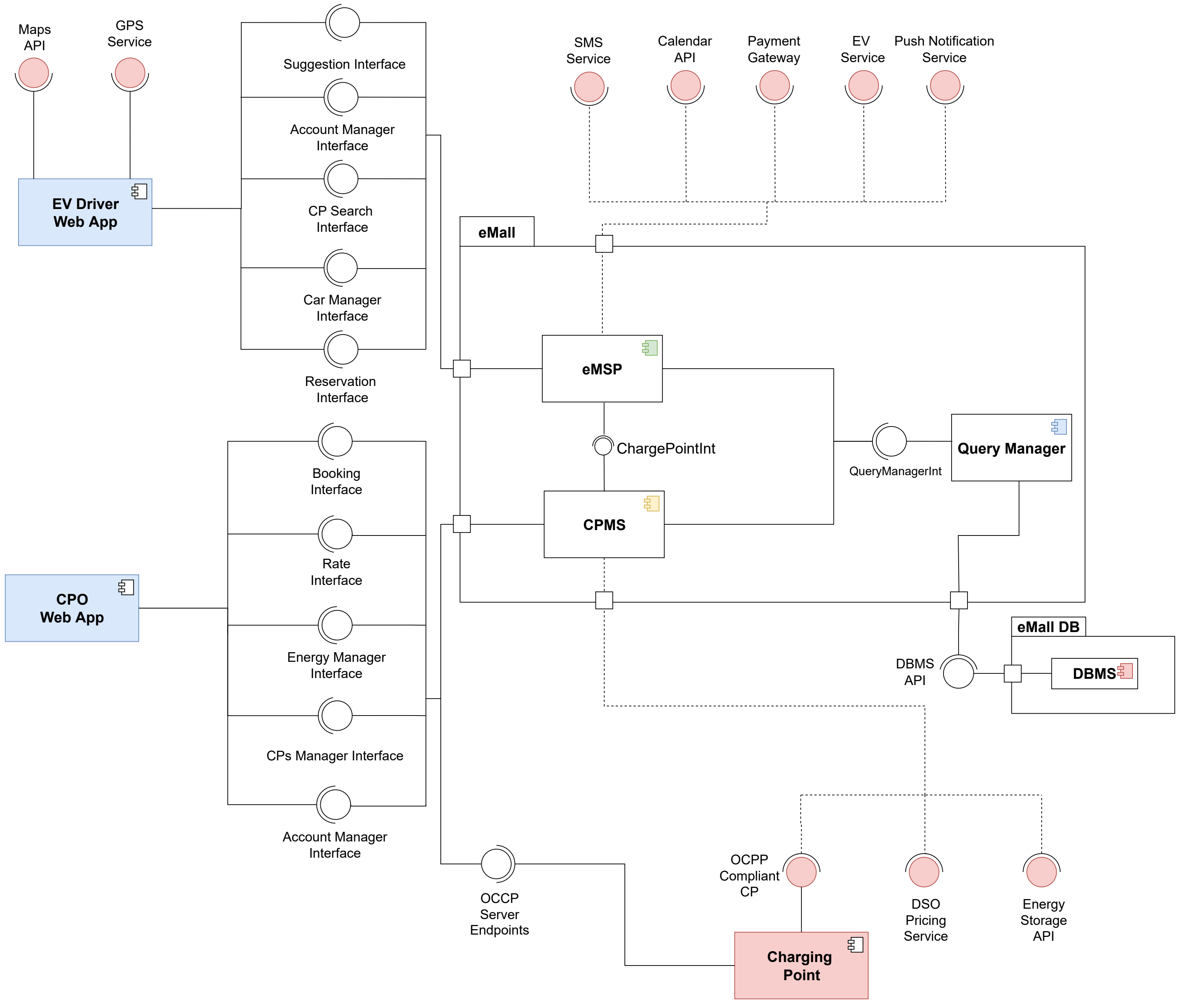


Figure 1: Component Diagram of the eMall System

2.2.1 Query Manager

This component is responsible for communication with a Database Management System (DBMS). It follows the Adapter design pattern, allowing other components to interact with the DBMS without needing to write any SQL code themselves.

2.2.2 eMSP

This component is responsible for interacting with EV drivers. It allows them to view CPs near their location, book and pay for a charge, set the car they are using to receive suggestions based on the battery status, their appointments in the calendar, and any special offers at the EVCP.

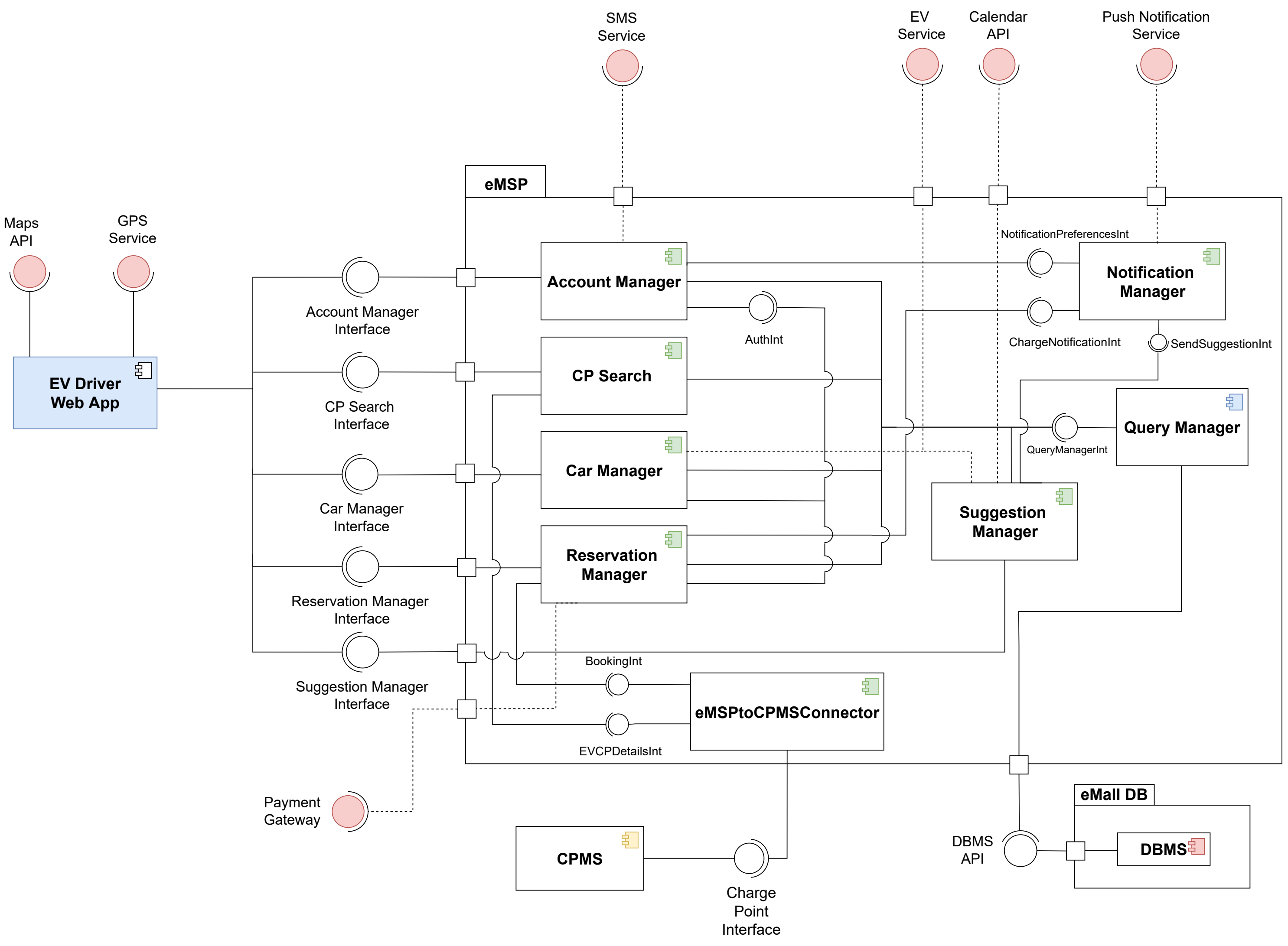


Figure 2: Component Diagram of the eMSP subsystem

eMSP Web App
The EV drivers' application uses client-side rendering of web pages. When a user sends a request through their browser, the eMSP application server responds by sending back HTML and JavaScript code, which the browser downloads and runs. As the user interacts with the page, the JavaScript can make additional requests to the server and update the page dynamically without requiring a full page reload.

eMSP Application Server
The eMSP Application Server is responsible for the business logic to provide the functionality to the application for the EV Drivers and to coordinate the information flow between application layer and data layer. It is composed of several components, each of them used for a specific functionality:

- **eMPS's Account Manager**
  This component handles all the account operations related to the CPO and offers an interface to authenticate the requests in the CPMS application server. It offers functionality to create new account, logging in, setting preferences and verify the authentication of the user at any time. To

create a new account interacts with the external SMS API to make the user receive a code to verify the identity.

- **CP Search**
  This component handles the operations needed to show the CPs in a specified range of km near a location. The location can be described as an address, with coordinates or by geolocating the actual position of the incoming request. It selects the filtered CPs that are then shown in the map as placeholders.
- **Car Manager**
  This component handles the operations related to the status of the car. It offers an interface to select the EV model among a list of all the marketed models and offers an interface to connect the car to the application. It uses this information to show the battery status of the EV, the amount of power during a charging process and to suggest charging plans based on the specific vehicle and status of it.
- **Reservation Manager**
  This component handles all the operations related to the reservations for the EV Drivers. It offers the possibility to create a reservation for a charge and pay for that reservation, see the details of a reservation, start or pause a charge of a particular reservation and see the recap of an already occurred reservation. It permits paying for a reservation using an external Payment Gateway that handles all the payment process and returns the status of the payment.
- **Suggestion Manager**
  This component provides recommendations to drivers. It obtains information from an EV Service that sends an alert when a car's battery is low, and from a Calendar API to identify open slots in the driver's schedule. The component also stores the suggestion in the DB and sends it to the driver through the notification manager's 'SendSuggestion' interface. Drivers can view these recommendations in a separate tab of their application.
- **Notification Manager**
  This component relies on a push notification service to provide other components with the ability to set notification preferences for drivers, send notifications when a charge has ended, and send suggestions. It offers interfaces for these functions to other components.
- **eMSPtoCPMSConnector**
  This component is responsible for implementing a protocol for communication with a specific CPMS. It hides the complexity of this communication by providing a simple interface for other components to use.

eMSP external APIs

- **SMS Service**
  This service is used to send SMS messages to drivers. One example of a service that can be used for this purpose is Twilio.
- **Calendar API**
  This service is used to access the calendars of drivers in order to identify available time slots for charging.
- **Payment Gateway**
  This service is used to handle all payment-related tasks. An example of a service that can be used for this purpose is Stripe.
- **EV Service**
  This service is used to access information about the car if the driver has chosen to personalize their experience by adding information about their car. This service allows the system to access the battery status of the car, which can be used to suggest when the driver should charge their vehicle.
- **Push Notification Service**
  This service is used to send push notifications to users.

2.2.3 CPMS

This component is responsible for interacting with the CPOs. A CPMS is associated with each CPO. The CPMS is responsible for managing CPs, reservations, rates, and special offers. It also controls the source of energy and the source mix for the EVCP.

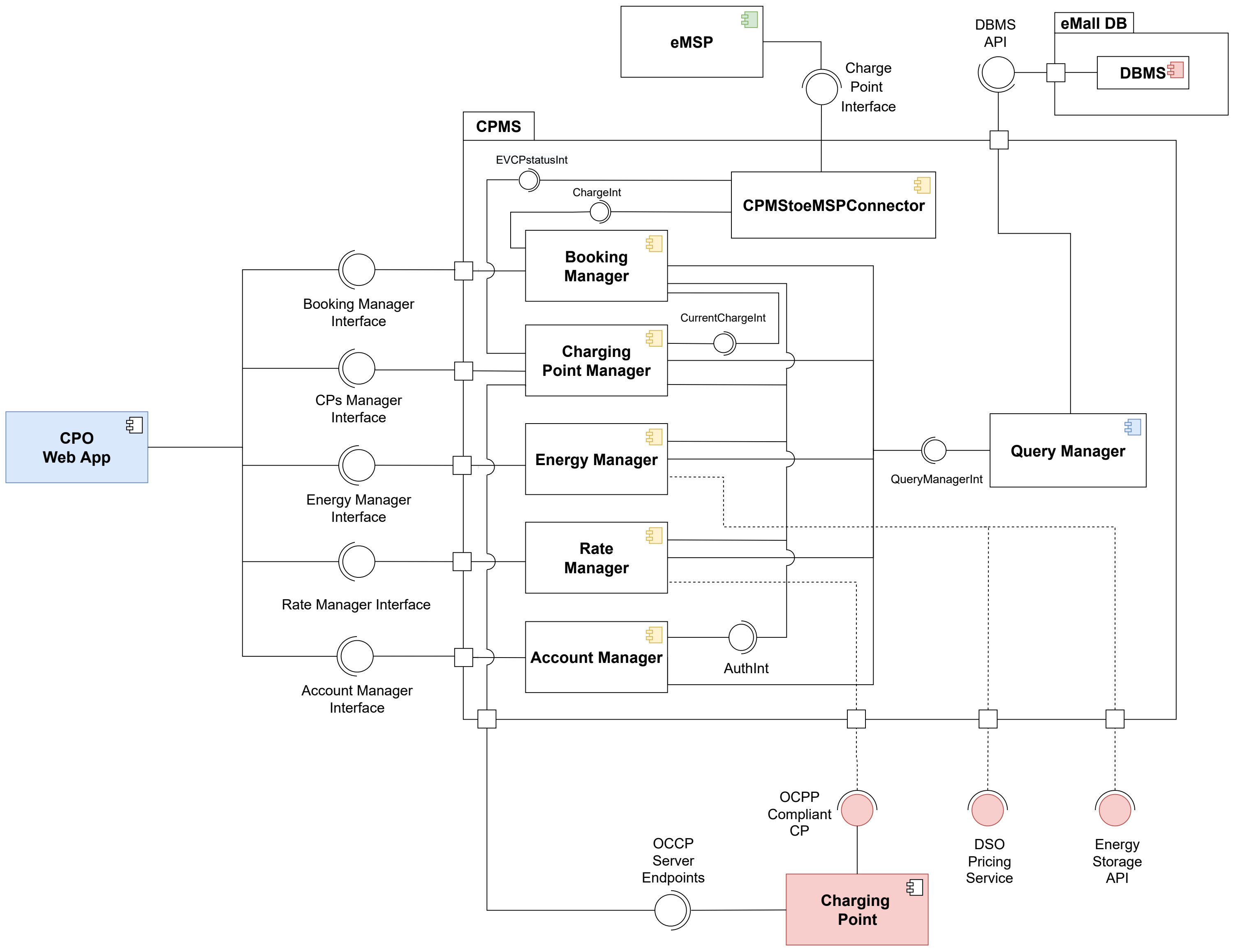


Figure 3: Component Diagram of the CPMS subsystem

CPO Web App
It represents the application dedicated to the different CPOs using the system. This component contains the logic to receive HTTPS requests from the users' browser, forward them to the CPMS Application Server, and generate dynamic pages based on the response from the CPMS Application Server.

CPMS Application Server
The CPMS Application Server is responsible for the business logic to provide the functionality to the application for the CPOs and to coordinate the information flow between application layer and data layer. It is composed of several components, each of them used for a specific functionality:

- **Booking Manager**
  This component handles the operations regarding the reservation of a specific CPO. It offers the possibility to manage the reservations and to control their status.

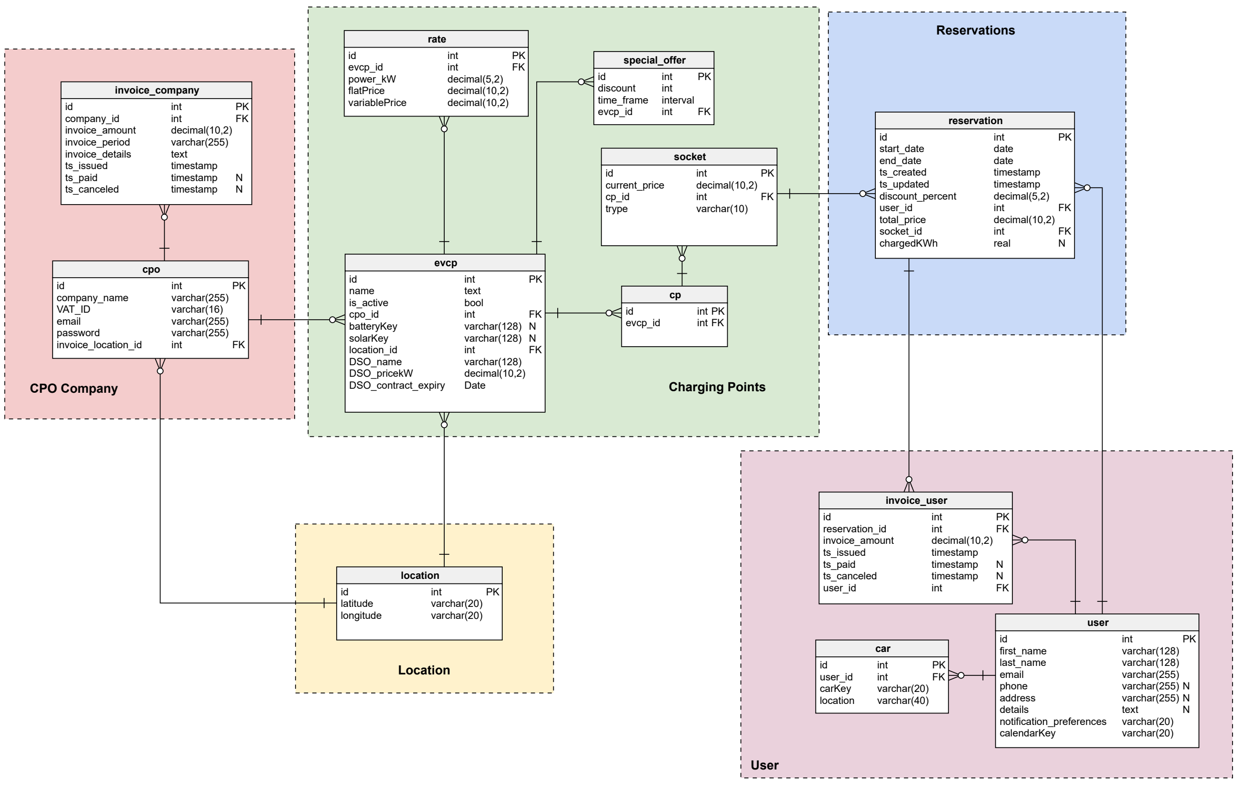
- **Rate Manager**
This component is responsible for the operations regarding the rates of that are associated with the CPs. It offers the possibility to create a new rate in all its details, to add a special rate and specify the duration. The created rates then are associated to a specific CP.
- **Energy Manager**
This component is responsible for the operations regarding the managing of the energy of the different EVCPs. It offers the possibility to verify the energy consumption and production of the different sources, modify and visualize the status of the energy storage system, visualize the availability of energy contracts provided by the DSOs for a specific EVCP and stipulate a contract among the ones available.
- **Charging Points Manager**
This component handles the operations regarding the managing of the CPs. It is an OCPP server for the CPs, offering the required functionalities for adding and removing a CP, for starting and stopping a charge and for receiving status data by the CP. It offers an interface to connect the different CPs through their platform.
- **CPMS's Account Manager**
This component handles all the account operations related to the CPO and offers an interface to authenticate the requests in the CPMS application server. It offers functionality to create new account, logging in and verifying the authentication of the user at any time.
- **CPMStoeMSPConnector**
This component is responsible for receiving commands from an eMSP using a standardized protocol. When a request is received, the component calls specific interfaces of internal components to handle the request.

CPMS external APIs

- **OCPP Compliant CP**
This API is used to interface with the CP that is supposed to implement the OCPP protocol.
- **DSO Pricing Service**
This API is used to retrieve all information about the DSO and to change contracts.
- **Energy Storage API**
This API is used to retrieve information about energy storage, which can be used to select a mix of energy sources.

2.2.4 Logical Description of Data

In this section the ER diagram of the S2B data layer is shown. An ER diagram is a graphical representation of the structure of a database, showing the relationships between entities and their attributes.



2.3 Deployment view

Our system consists of two parts: a static web server and an application server. The static web server will be the entry point for clients to access the SPA, while the application server will provide the necessary APIs for the SPA to work. We have chosen to use two different solutions for these components. The static web server will be hosted on a CDN (Content Delivery Network) to ensure fast response times through its edge location caches and reverse proxies. The application server, which includes both a business logic layer and a data tier, will be hosted on a cloud provider. This offers several advantages compared to traditional in-house hosting, such as:

- Scalability and Flexibility - the ability to add or remove resources such as virtual machines, performance cores, or memory as needed, and the use of load balancing services, allows the application server to adapt to changes in traffic or workload.
- Security - services like live monitoring and firewalls help to protect the application server against data breaches, cyberattacks, and other security threats.
- Cost-efficiency - the pay-as-you-go model of a cloud provider allows to only pay for the resources that are actually used, which can help to lower the overall costs.

These features make a cloud provider an ideal choice for hosting large, high-traffic applications. The chosen cloud provider will need to offer all of these features in order to meet our needs.

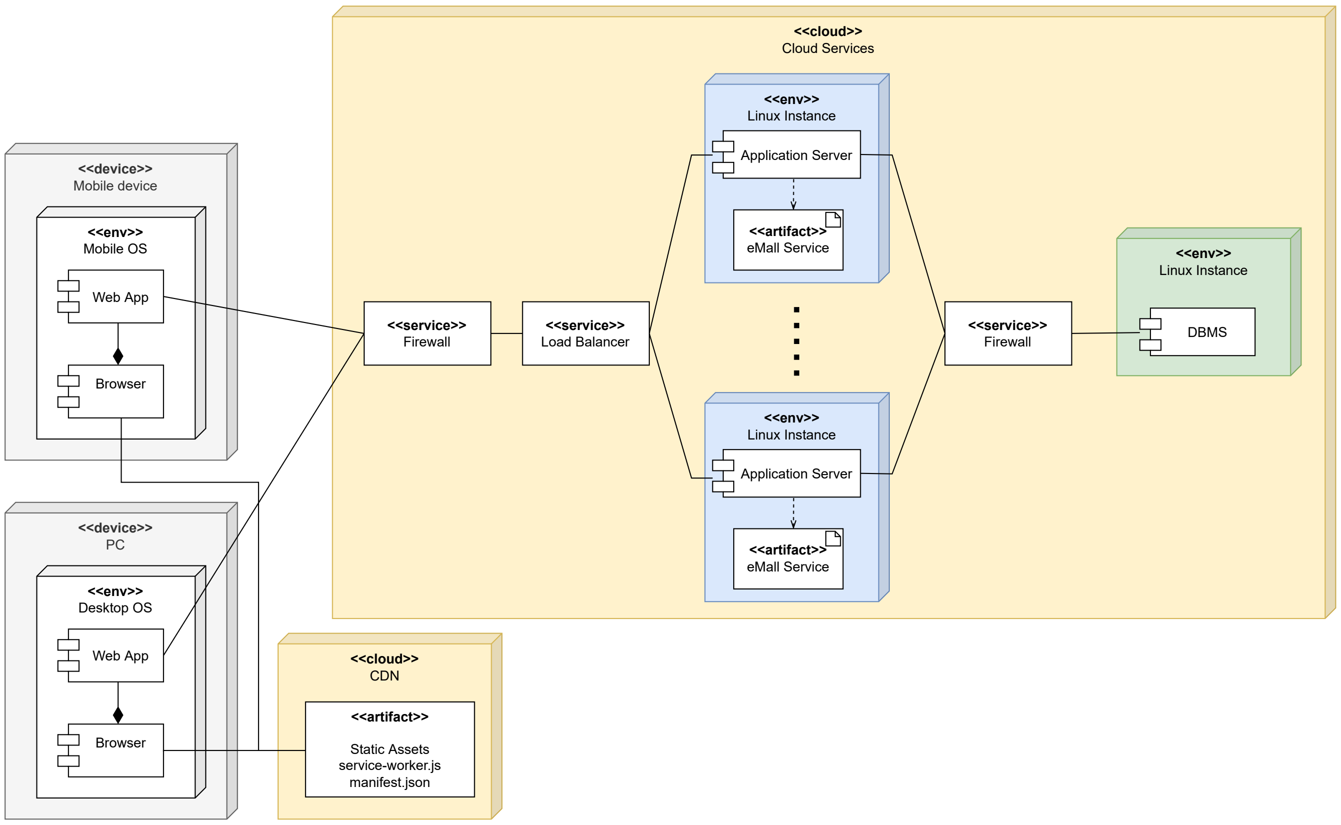


Figure 4: EV Driver Registration process

The deployment diagram offers a more detailed view over the hardware and software resources of the application:

- PC or Mobile Device is any device having a modern browser capable of running the JavaScript based web app.
- The SPA will be hosted by a CDN, which will allow it to be downloaded without affecting the performance of the main application server. The SPA is static and all of its code is run on the client's machine, so there is no need for any logic to be implemented on the CDN side.
- The Cloud Services will host all the business and data logic for the system. It contains:
  - **Firewall** services are used to filter incoming connections to the business and data layers of a

system. They provide an additional layer of security by blocking or allowing traffic based on predetermined rules. This helps to protect the system from unauthorized access or malicious attacks.

– **Load balancer** to distribute incoming traffic across multiple instances of an application in order to optimize resource utilization, improve performance, and ensure high availability. The load balancer helps to ensure that the application can handle a large volume of requests without becoming overloaded or experiencing downtime. It also helps to provide a more stable and reliable experience for users by redirecting traffic to the least busy application instance.
– Multiple copies of the **application**. The various instances can run in parallel and independently to meet the demand for the application. These instances can be created or deleted as needed. Running multiple instances allows the application to handle a high volume of requests without experiencing performance issues, and provides fault tolerance by allowing traffic to be redirected to a different instance if one instance becomes unavailable.
– **Data Instance** which is a data optimized virtual machine containing the DBMS and the database.

2.4 Runtime view

The runtime views describe the interactions between actors, subsystems and interfaces of the system showing the specific method called. The views are divided between eMSP and CPMS.

2.4.1 Token Validation

Users can only search and view details of a CP without first being authenticated and authorized. All other actions that a user (both drivers and CPOs) can perform are dependent on the validation of a token. The token is passed in the API request and the process of verifying it is shown in the sequence diagram below.

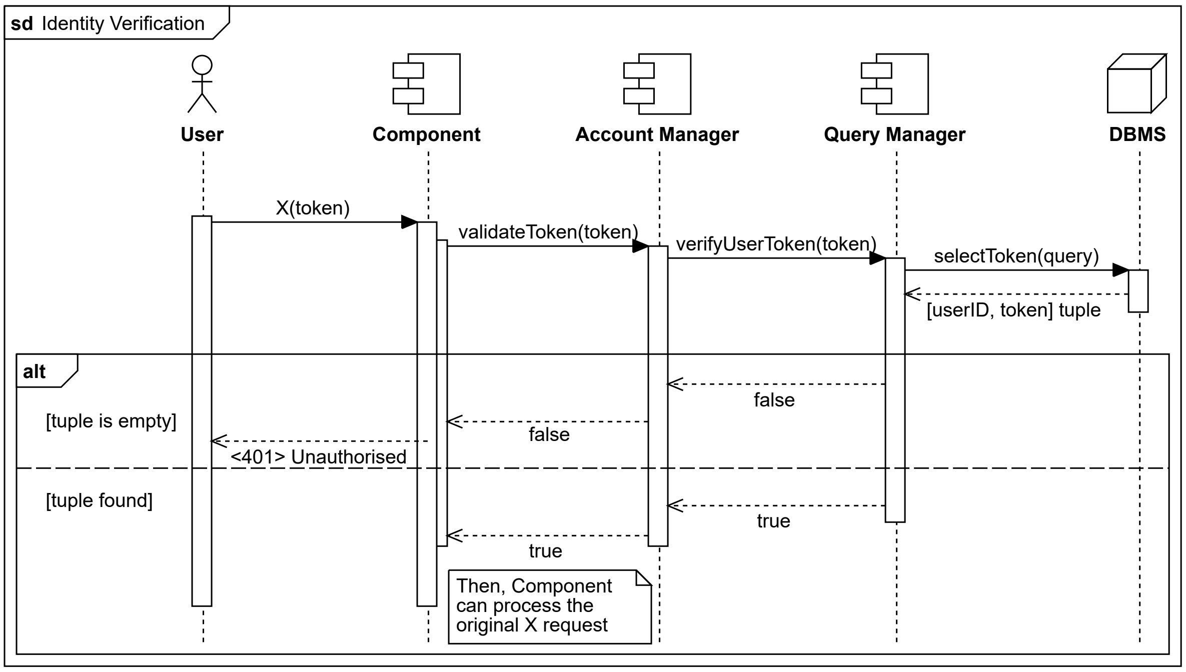


Figure 5: User token validation

2.4.2 eMSP

- **EV Driver Registration** : The following diagram represents the workflow that an EV Driver has to follow to register into eMall.

  When the unregistered user enters the input data the System will check whether the data input is valid. If the input is valid, the request is forwarded to the Account Manager that proceeds to evaluate if there is another user registered with the same phone number. If the user has inserted a phone number not already registered, then the user is created and an sms verification code is sent to the user through the SMS Service. After receiving the correct verification code the user is notified that the registration is completed.

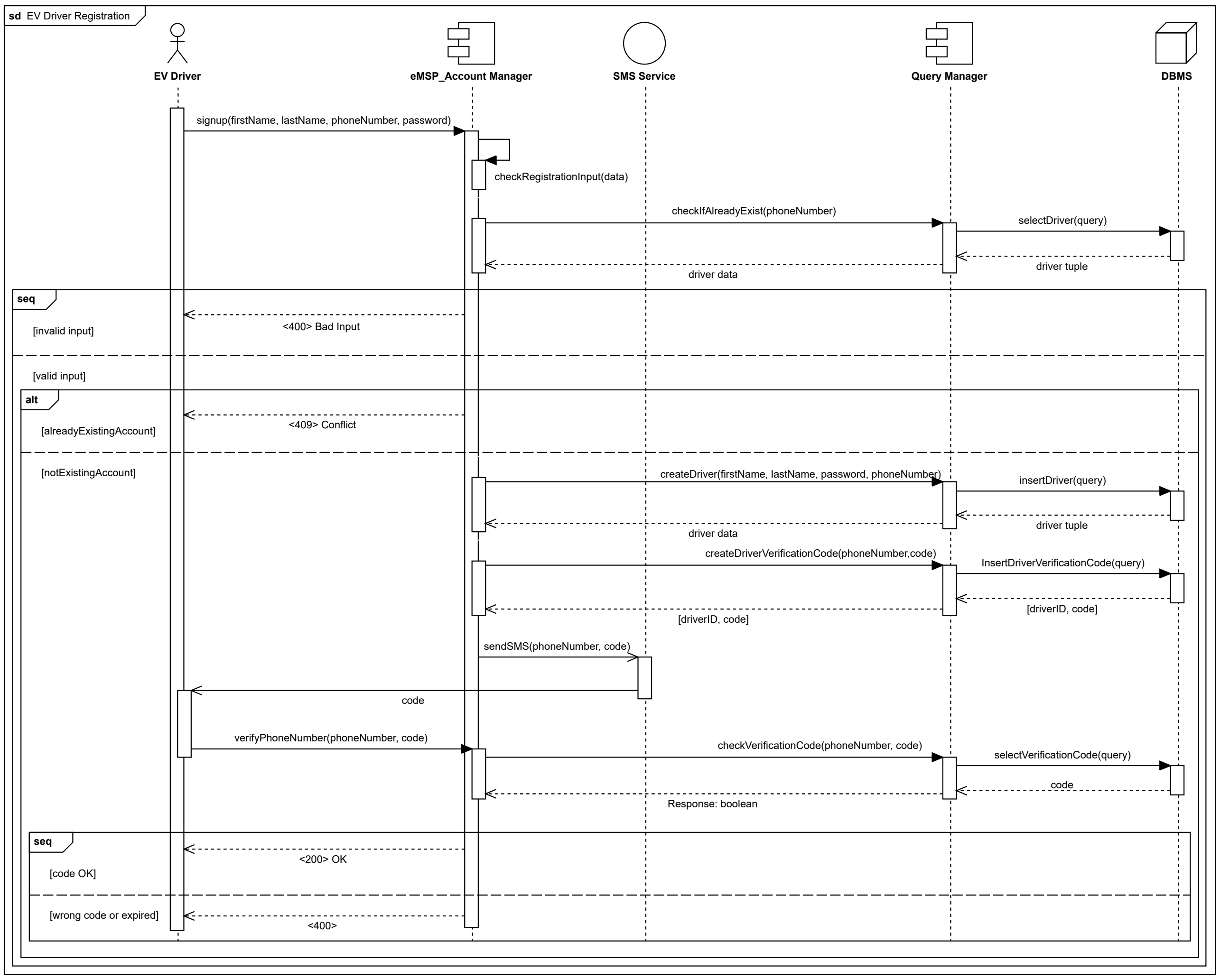

sd EV Driver Registration
EV Driver
eMSP_Account Manager
SMS Service
Query Manager
DBMS
signup(firstName, lastName, phoneNumber, password)
checkRegistrationInput(data)
checkIfAlreadyExist(phoneNumber)
selectDriver(query)
driver tuple
driver data
seq
[invalid input]
<400> Bad Input
[valid input]
alt
[alreadyExistingAccount]
<409> Conflict
[notExistingAccount]
createDriver(firstName, lastName, password, phoneNumber)
insertDriver(query)
driver tuple
driver data
createDriverVerificationCode(phoneNumber,code)
InsertDriverVerificationCode(query)
[driverID, code]
[driverID, code]
sendSMS(phoneNumber, code)
code
verifyPhoneNumber(phoneNumber, code)
checkVerificationCode(phoneNumber, code)
selectVerificationCode(query)
code
Response: boolean
seq
[code OK]
<200> OK
[wrong code or expired]
<400>

Figure 6: EV Driver Registration process

- **EV Driver Log in** : The following diagram represents the workflow that the EV Driver has to follow to log in.

  After submitting the login inputs, the Account Manager verifies the credentials by interacting with the Query Manager. If the input is correct authenticates the user through a token and redirects the EV Driver to the Map Homepage.

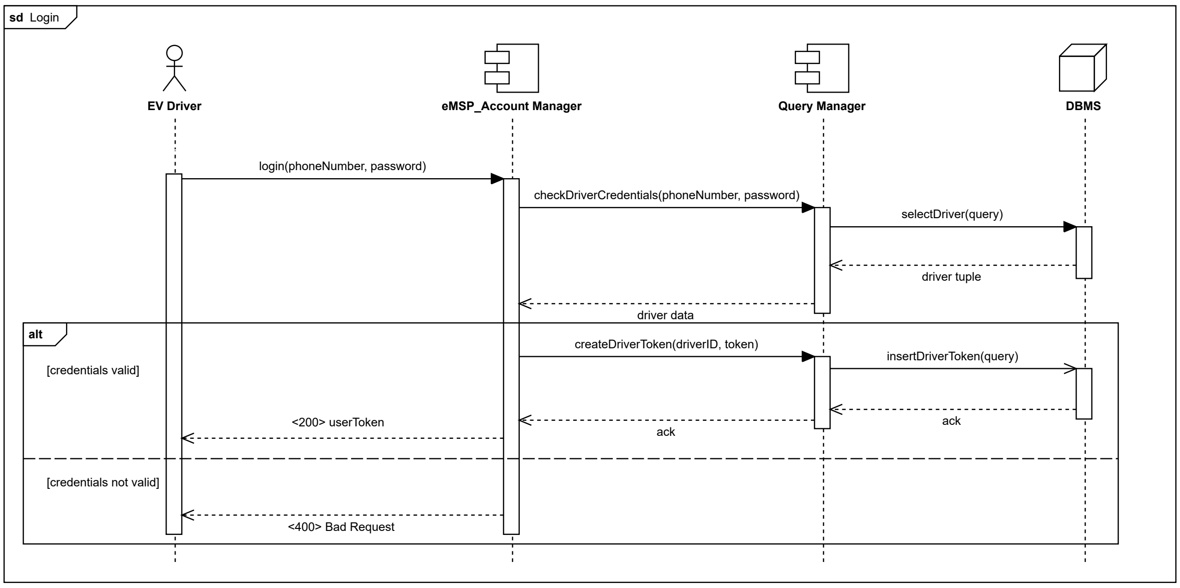


Figure 7: EV Driver Logs in the system

- **EV Driver Search an EVCP** : The following diagram represents the workflow that the EV Driver has to follow to search for CPs in the map.

  By filtering the results by location, connector and/or date sends a request for an update of the page. The CP Search interacts with the Query Manager to get the filtered CPs and responds with the list.

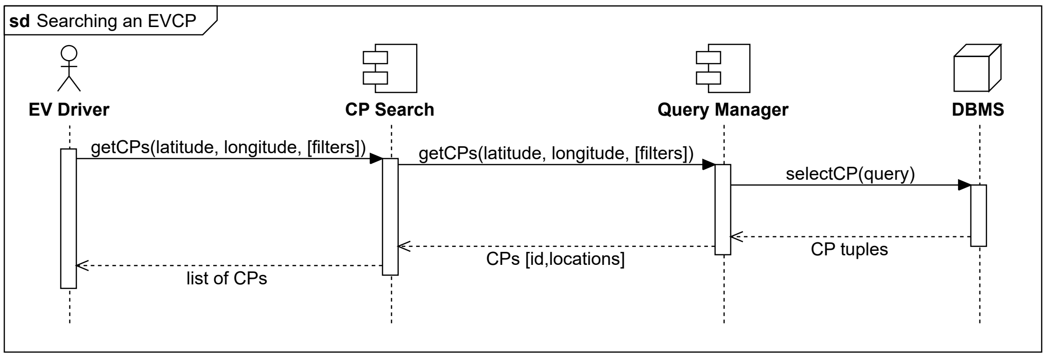


Figure 8: EV Driver searches on the map

- **EV Driver books a charge** : The following diagram represents the workflow when the EV Driver books a charge.

  After the submission of the reservation parameters by the user, the Reservation Manager interacts with the eMSPtoCPMSConnector to contact the specific CPMS to book the charge. The reservations parameters are received by the specific CPMStoeMSPConnector that forwards the payload to the Booking Manager that contacts the Query Manager to verify that the reservations parameters aren't in conflict with others reservations. If the reservation is valid then is added to the database through the Query Manager and the Reservation Manager of the eMSP is notified of the validity of the reservation. The Reservation Manager contacts the Payment Gateway to pre-authorize the payment with the credit card that has been already added and selected by the EV Driver.

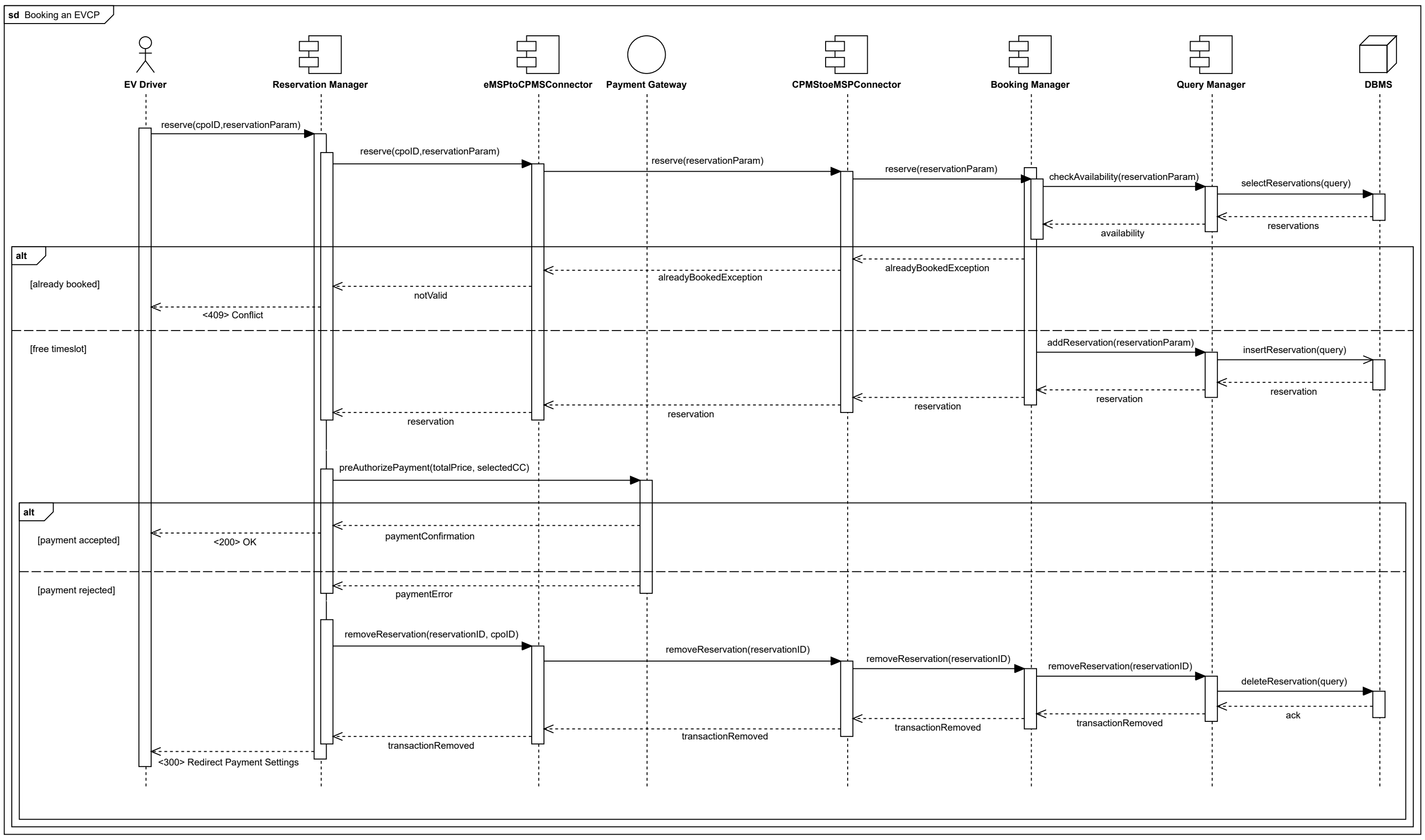


Figure 9: EV Driver books a charge

- **EV Driver starts a charge** : The following diagram represents the workflow when the EV Driver starts a charge.

  The submission of the request contains the reservation identifier of the charge that has to be started and the car details. The Reservation manager interacts with the eMSPtoCPMSConnector to contact the specific CPMS. Through the CPMStoeMSPConnector the Booking Manager receives the request to start a charge. Verifies through the Query Manager that the request corresponds to a reservation that can be started. If so, the Booking Manager contacts the Charging Points Manager to activate the socket of the CP specified in the reservation. The Charging Points Manager contacts the CP through the OCPP Compliant CP interface and starts the charge.

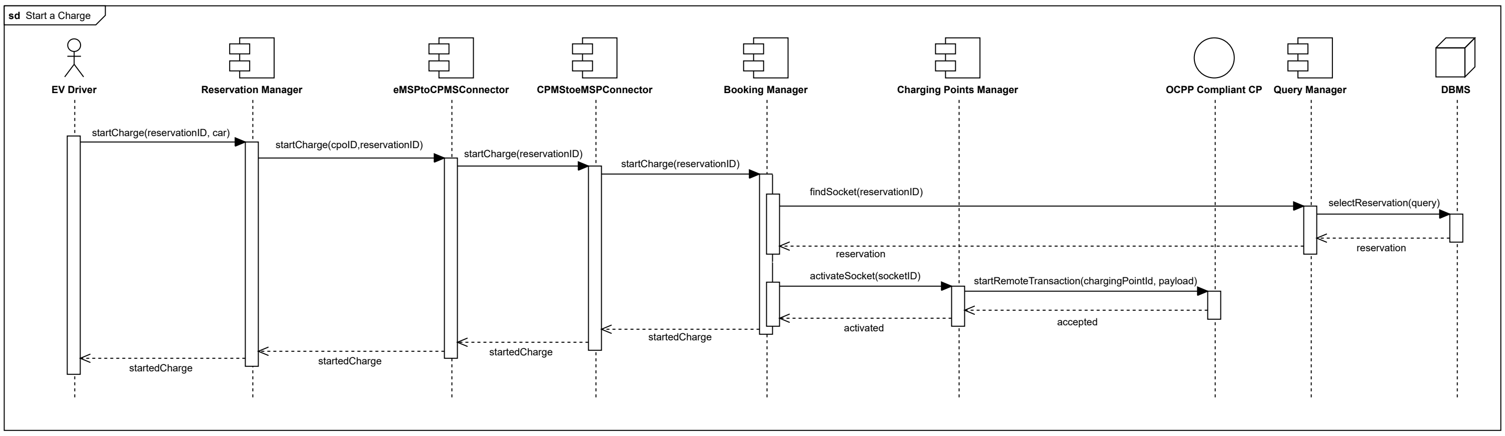


Figure 10: EV Driver starts a charge

- **EV Driver sees reservations** : The following diagram represents the workflow when the EV Driver wants to see the his/her reservations.

  The Reservation Manager is contacted and requires the Query Manager to find all the reservations associated with the EV Driver.

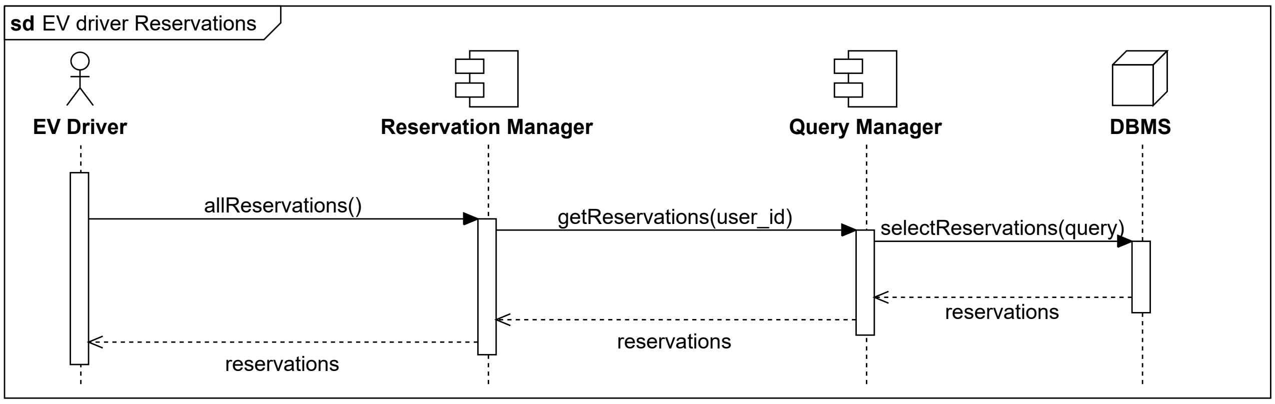


Figure 11: EV Driver sees the reservations

- **EV Driver adds a credit card** : The following diagram represents the workflow when the EV Driver adds a credit card.

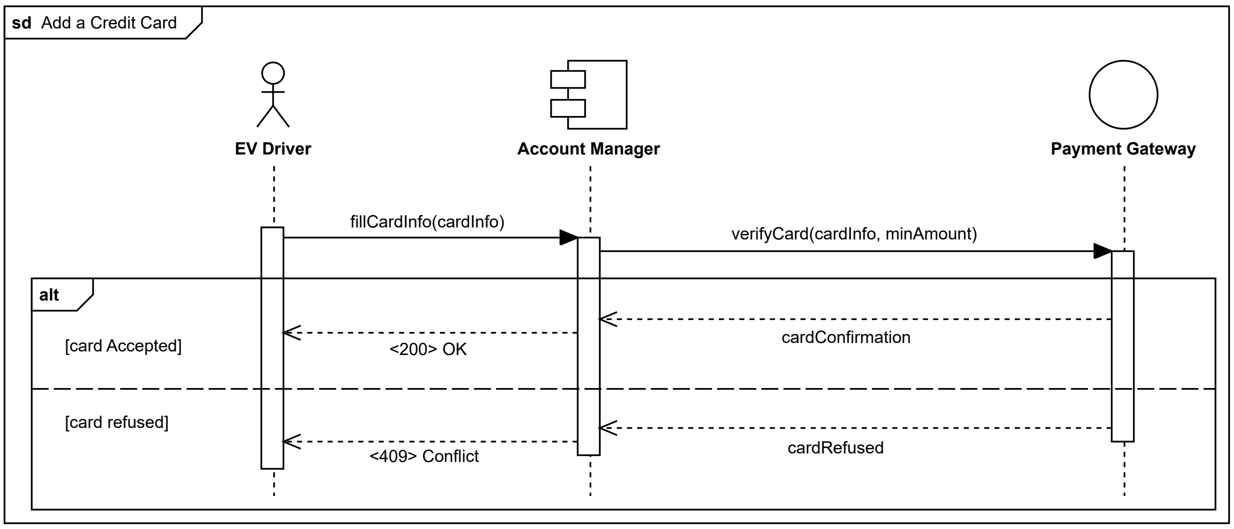


Figure 12: EV Driver adds a credit card information

- **EV Driver adds a car** : The following diagram represents the workflow when the EV Driver adds a car.

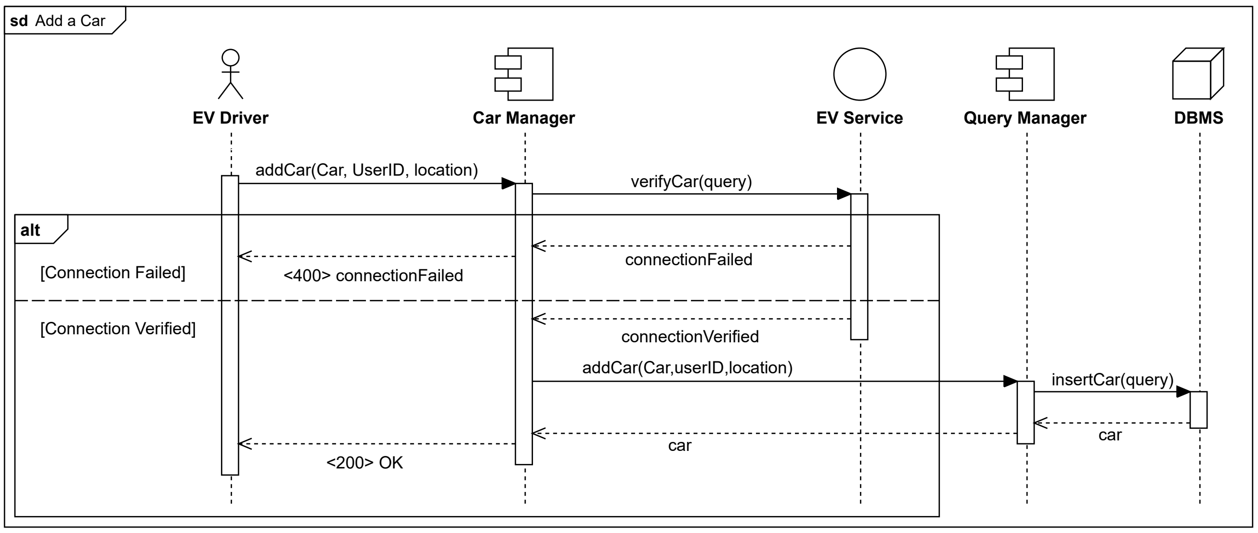


Figure 13: EV Driver adds a car

- **Suggestion** : The following diagram represents the workflow when a Suggestion is sent to the EV Driver.

  After the discharge of the car, the interface EV Service sends "lowBatteryEvent". The Suggestion Manager receives carID such that can obtain through the Query Manager, from the DB, the corresponding userID. The Suggestion Manager, thanks to the attribute calendar "key" of the user, interacts with the Calendar API and receives "FreeSpots" (those time intervals in the user Calendar that are free). The Suggestion Manager contacts the Query Manager with "getTopKSuggestions" through which obtains the top K suggestions according to the ranking algorithm implemented by the system. The Suggestion Manager contacts Notification Manager through which contacts the Query Manager to find the notification preferences. If the notification preferences of the user are such that he accepts suggestion, the Notification Manager pushes a notification through the Push Notification Service.

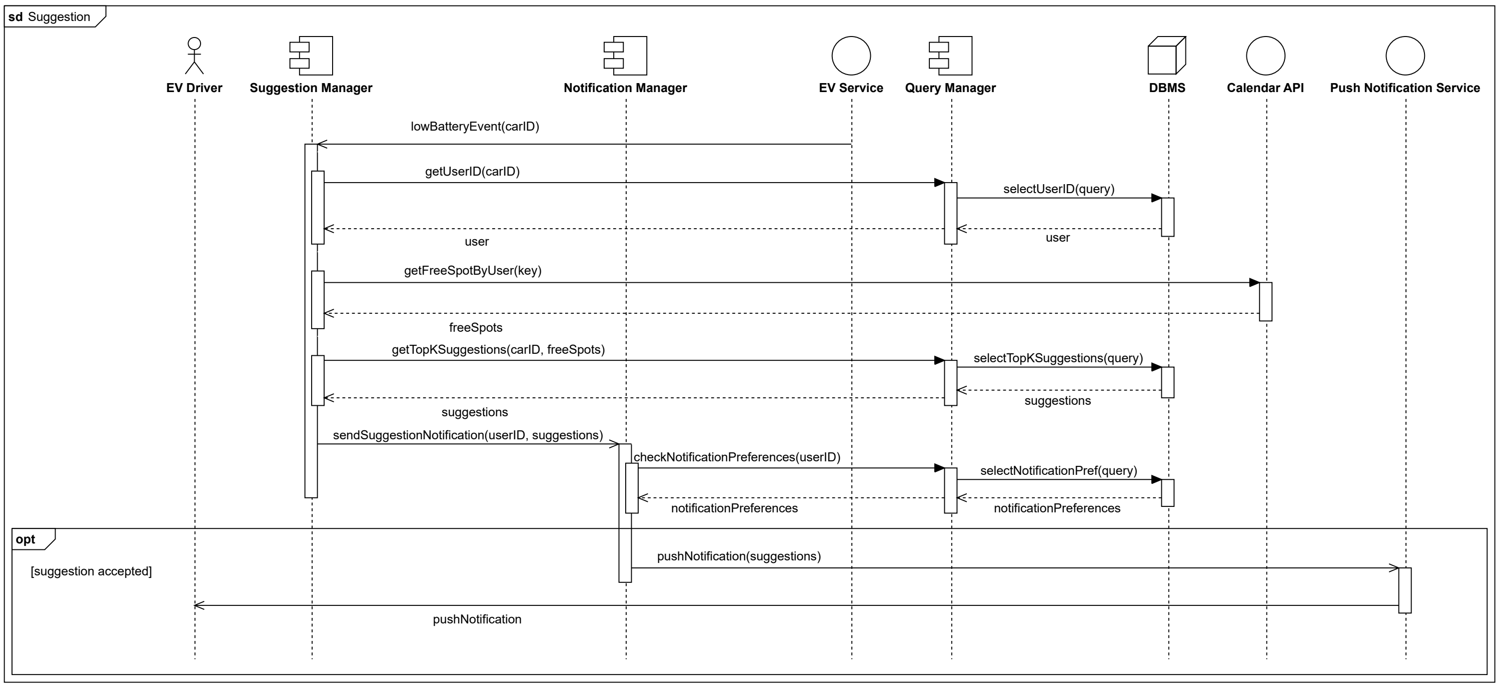


Figure 14: Suggestion

- **Current Charge Info**: The following diagram represents the workflow when the EV Driver wants to see details of the current charge.

  The Reservation Manager receives a request from a driver for reservation details and uses the connector to cpms to contact the specific CPMS of the CPO that owns the charging point where the reservation is taking place. The CPMS then sends a request for meter values to the charging point via the OCPP protocol and receives a response with the requested information. This information is then passed back through the system and displayed to the driver through the web app.

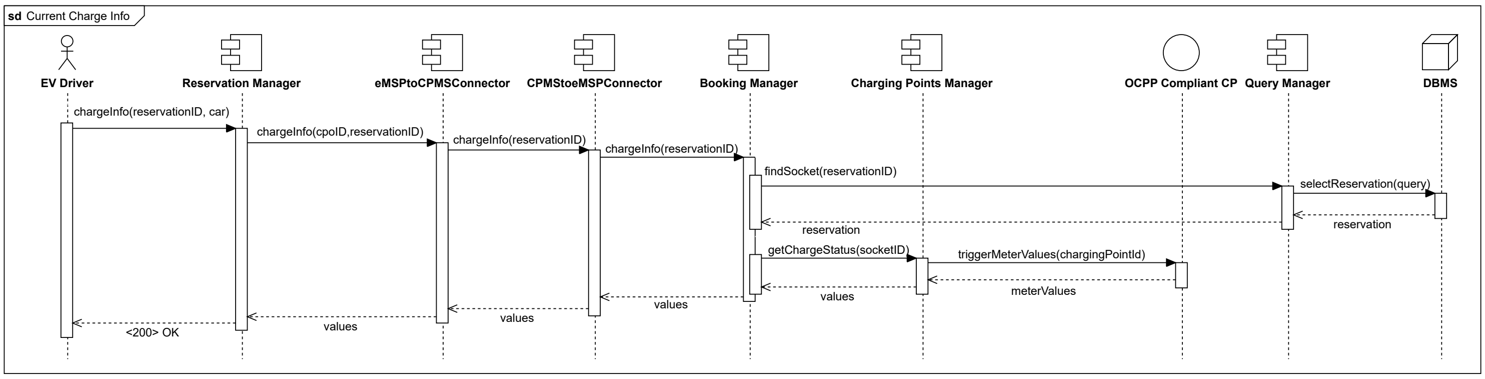


Figure 15: EV sees the current charge information

2.4.3 CPMS

- **CPO sees the reservations** : The following diagram represents the workflow when the CPO wants to see the his/her reservations.

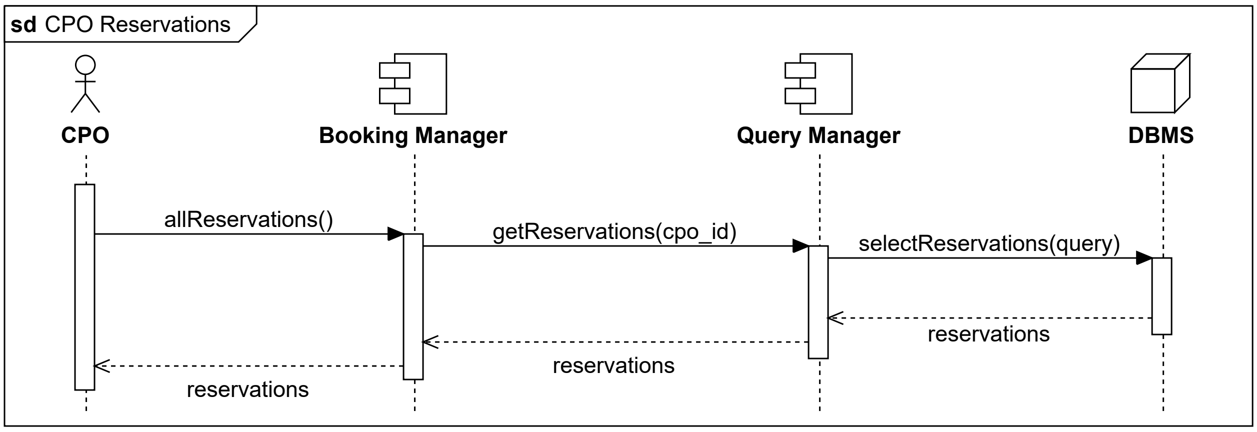


Figure 16: CPO sees the reservations

- **CPO adds a Charging Point** : The following diagram represents the workflow when the CPO adds a charging point.

  The request contains the information about the CP unique identifier, the EVCP in which the CP is going to be added, and the rate associated with. The Charging Points Manager interacts with the Query Manager to add the information. When the user, through the CP manufacturer's portal connects the CP to the CPMS, then the CP is connected and is able to receive OCPP messages from the CPMS of the CPO.

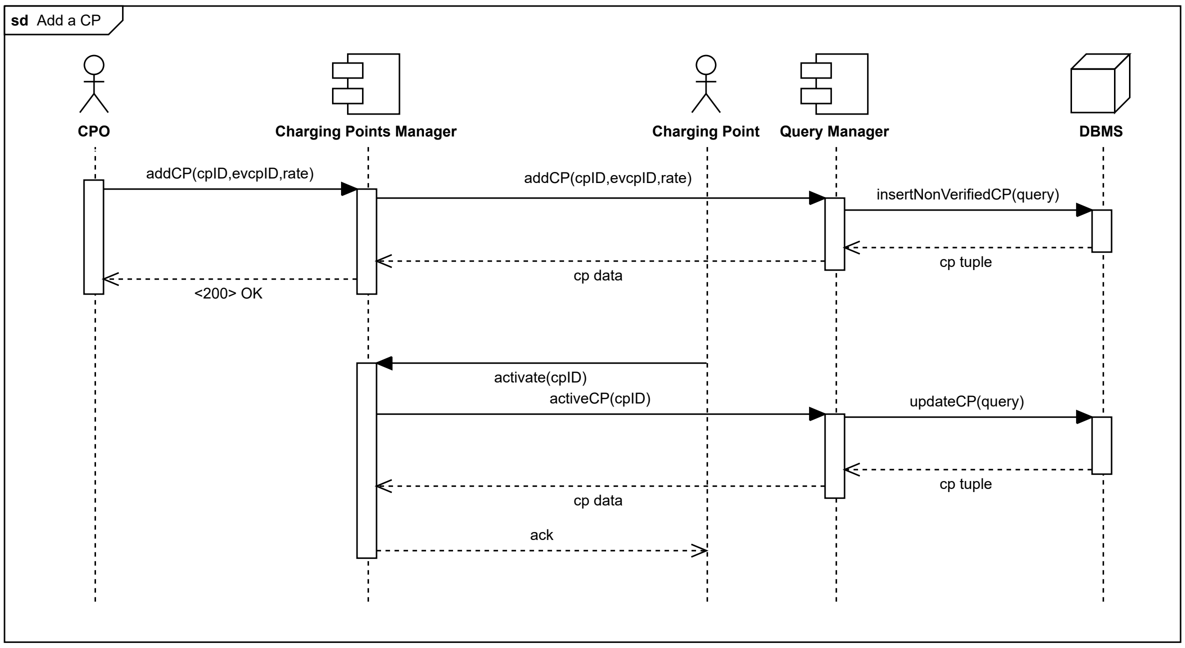


Figure 17: CPO adds a Charging Point

- **CPO adds a rate** : The following diagram represents the workflow when the CPO adds a rate.

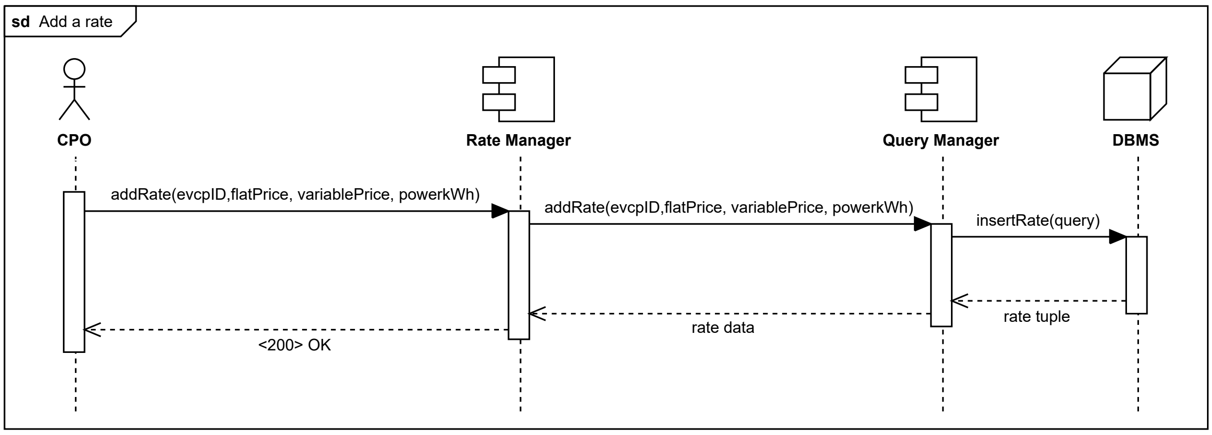

sd Add a rate
CPO
Rate Manager
Query Manager
DBMS
addRate(evcpID,flatPrice, variablePrice, powerkWh)
addRate(evcpID,flatPrice, variablePrice, powerkWh)
insertRate(query)
rate tuple
rate data
<200> OK

Figure 18: CPO adds a rate

- **CPO sees the energy storage system capacity** : The following diagram represents the workflow when the CPO wants to see the energy storage system capacity.

  The request contains the specific EVCP identifier of where the energy storage system is located. The Energy Manager requires to the database the unique key to access to the API of the Energy Storage. Then asks the Energy Storage API the information about the status.

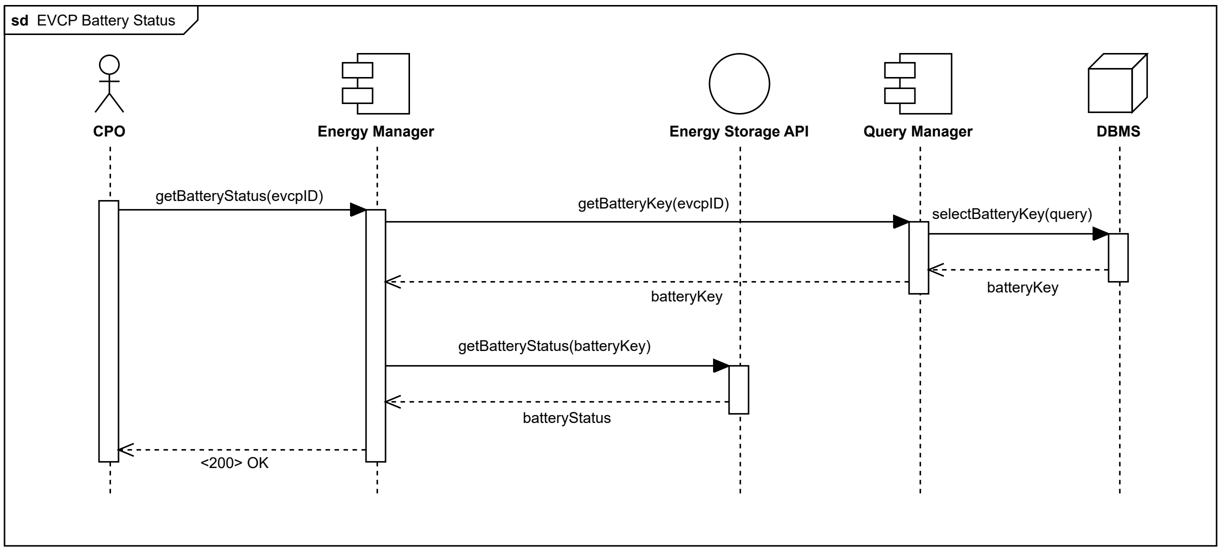


Figure 19: CPO sees the energy storage system capacity

- **CPO sees the status of its Charging Points** : The following diagram represents the workflow when the CPO wants to see the charging points status of an evcp.

  The request contains the specific EVCP identifier of where the CPs are located. The Charging Points manager asks the Query Manager for a list of CPs of the specific EVCP and the contacts each one of the CP from the list with the OCPP Compliant CP interface.

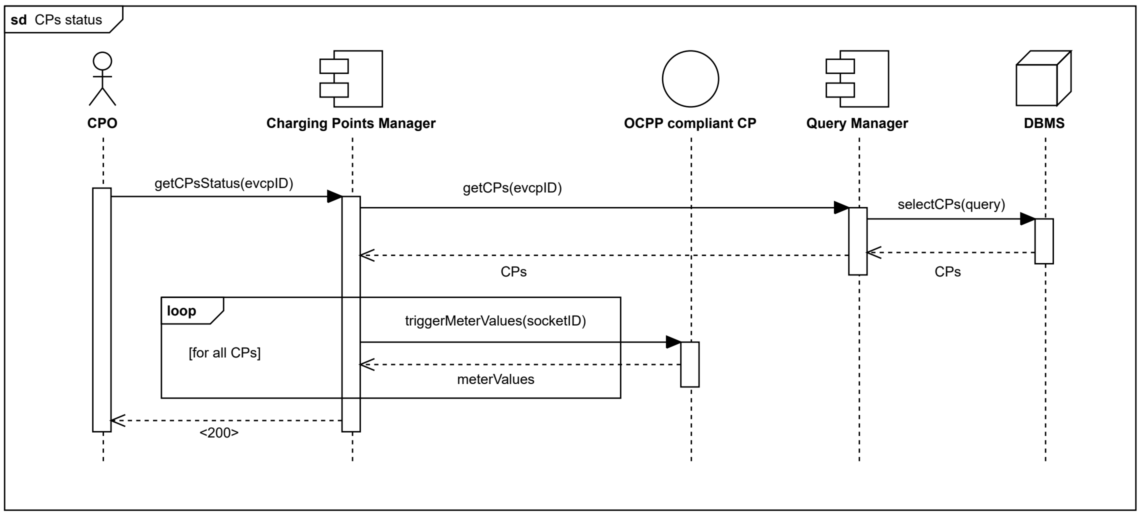


Figure 20: CPO sees the status of its Charging Points

- **CPO adds a special offer** : The following diagram represents the workflow when the CPO adds a special offer.

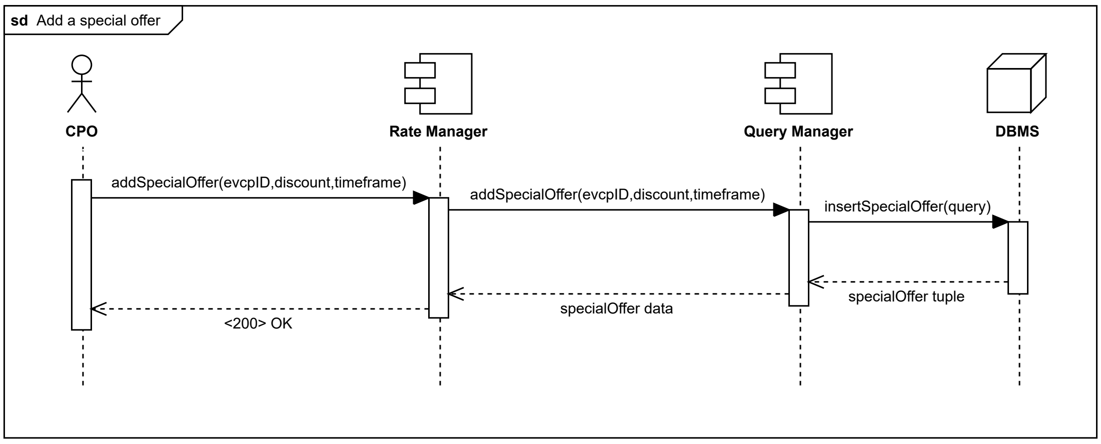

sd Add a special offer
CPO
Rate Manager
Query Manager
DBMS
addSpecialOffer(evcpID,discount,timeframe)
addSpecialOffer(evcpID,discount,timeframe)
insertSpecialOffer(query)
specialOffer tuple
specialOffer data
<200> OK

Figure 21: CPO adds a special offer

- **Choose energy Mix** : The following diagram represents the workflow when the CPO chooses the energy mix.

  The CPO changes energy mix interacting with the Energy Manager. The Energy Manager interacts with the Query Manager obtaining the battery and solar keys. Then, the Energy Manager with this information and the mix attribute contacts the Energy Storage API and obtains the new mix.

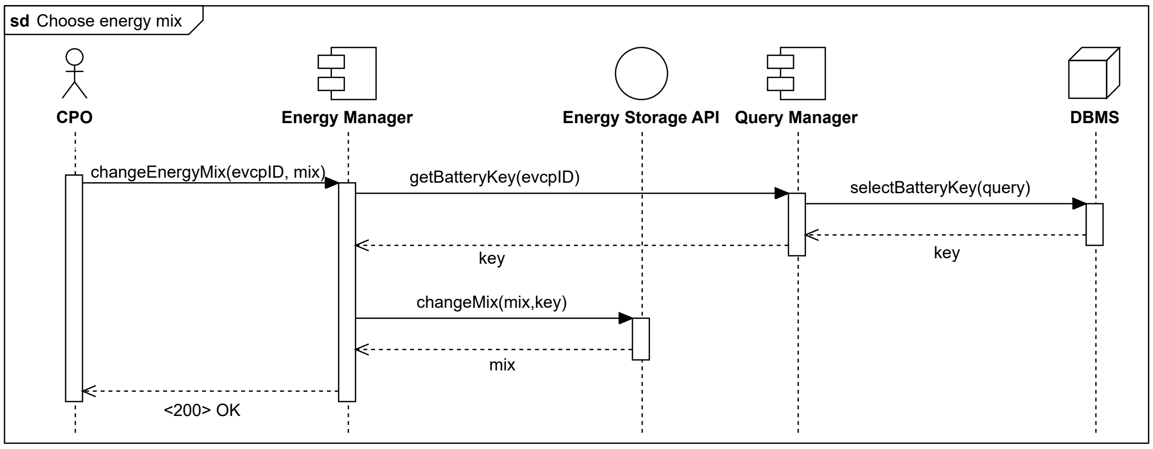


Figure 22: CPO chooses energy Mix

- **Choose DSO** : The following diagram represents the workflow when the CPO chooses DSO.

  The CPO chooses DSO interacting with the Energy Manager. The Energy Manager interacts with the Query Manager obtaining the Commitment Date of the actual contract. If the Commitment date is not reached yet, the Energy Manager responds that the request is not valid. Otherwise, the Energy Manager interacts with the Energy Storage API obtaining a contract. Then the Energy Manager sends an ACK to the CPO and interacts with the Query Manager to store the new contract.

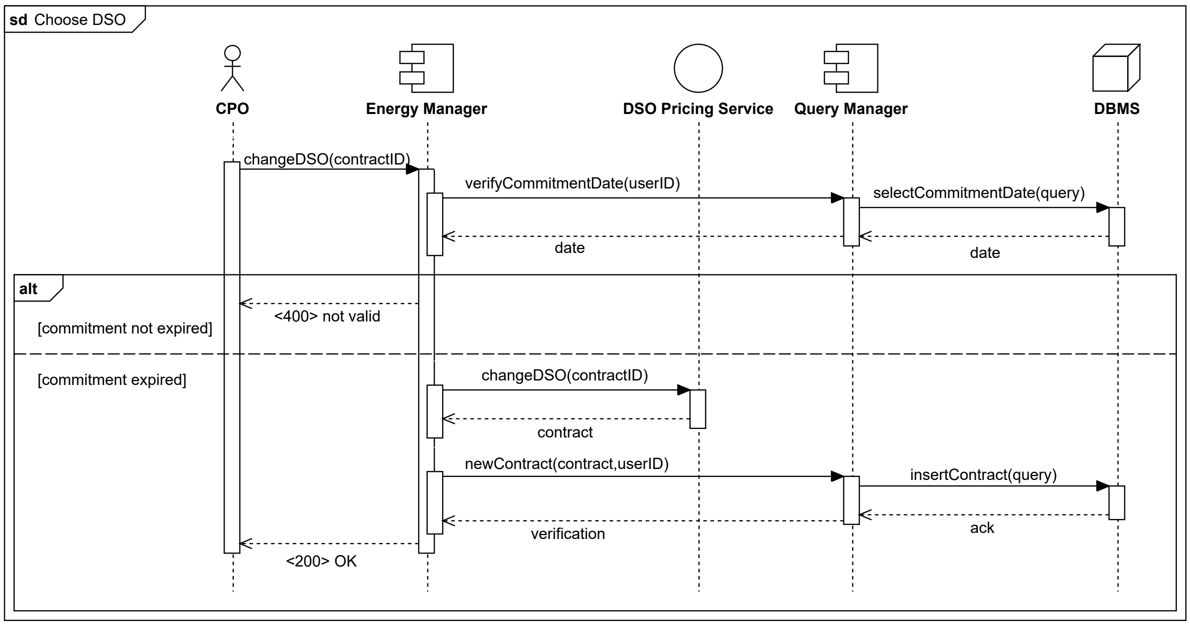


Figure 23: CPO chooses a DSO to acquire energy from

2.5 Component interfaces

In this section, we use a class diagram to illustrate the component interfaces. The methods are presented as clearly as possible, and we expect their functionality to be easily understandable.

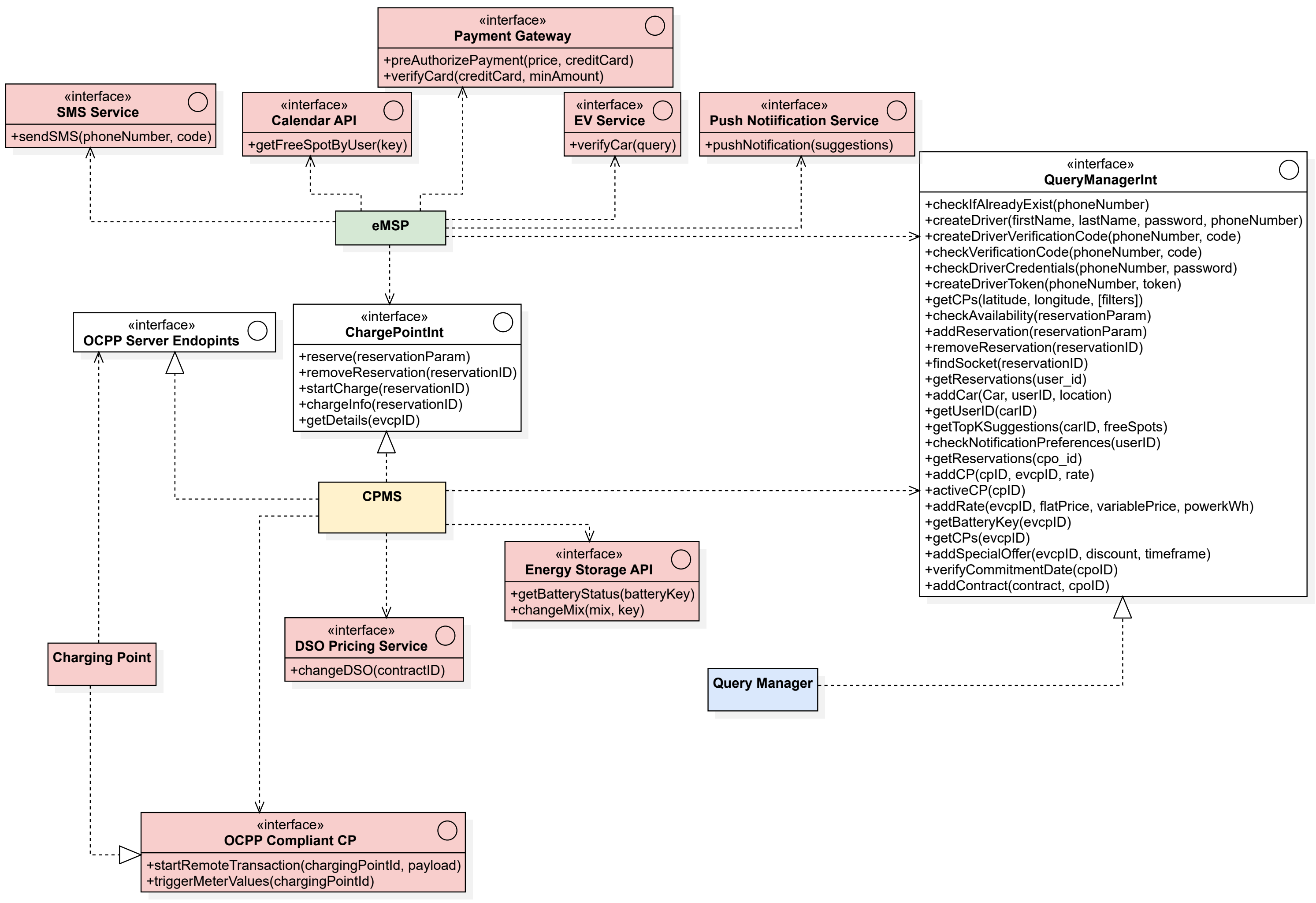

«interface»
Payment Gateway
+preAuthorizePayment(price, creditCard)
+verifyCard(creditCard, minAmount)
«interface»
SMS Service
+sendSMS(phoneNumber, code)
«interface»
Calendar API
+getFreeSpotByUser(key)
«interface»
EV Service
+verifyCar(query)
«interface»
Push Notiification Service
+pushNotification(suggestions)
«interface»
QueryManagerInt
+checkIfAlreadyExist(phoneNumber)
+createDriver(firstName, lastName, password, phoneNumber)
+createDriverVerificationCode(phoneNumber, code)
+checkVerificationCode(phoneNumber, code)
+checkDriverCredentials(phoneNumber, password)
+createDriverToken(phoneNumber, token)
+getCPs(latitude, longitude, [filters])
+checkAvailability(reservationParam)
+addReservation(reservationParam)
+removeReservation(reservationID)
+findSocket(reservationID)
+getReservations(user_id)
+addCar(Car, userID, location)
+getUserID(carID)
+getTopKSuggestions(carID, freeSpots)
+checkNotificationPreferences(userID)
+getReservations(cpo_id)
+addCP(cpID, evcpID, rate)
+activeCP(cpID)
+addRate(evcpID, flatPrice, variablePrice, powerkWh)
+getBatteryKey(evcpID)
+getCPs(evcpID)
+addSpecialOffer(evcpID, discount, timeframe)
+verifyCommitmentDate(cpoID)
+addContract(contract, cpoID)
eMSP
«interface»
OCPP Server Endopints
«interface»
ChargePointInt
+reserve(reservationParam)
+removeReservation(reservationID)
+startCharge(reservationID)
+chargeInfo(reservationID)
+getDetails(evcpID)
CPMS
«interface»
Energy Storage API
+getBatteryStatus(batteryKey)
+changeMix(mix, key)
«interface»
DSO Pricing Service
+changeDSO(contractID)
Charging Point
Query Manager
«interface»
OCPP Compliant CP
+startRemoteTransaction(chargingPointId, payload)
+triggerMeterValues(chargingPointId)

Figure 24: Class Diagram with interfaces of the eMall System

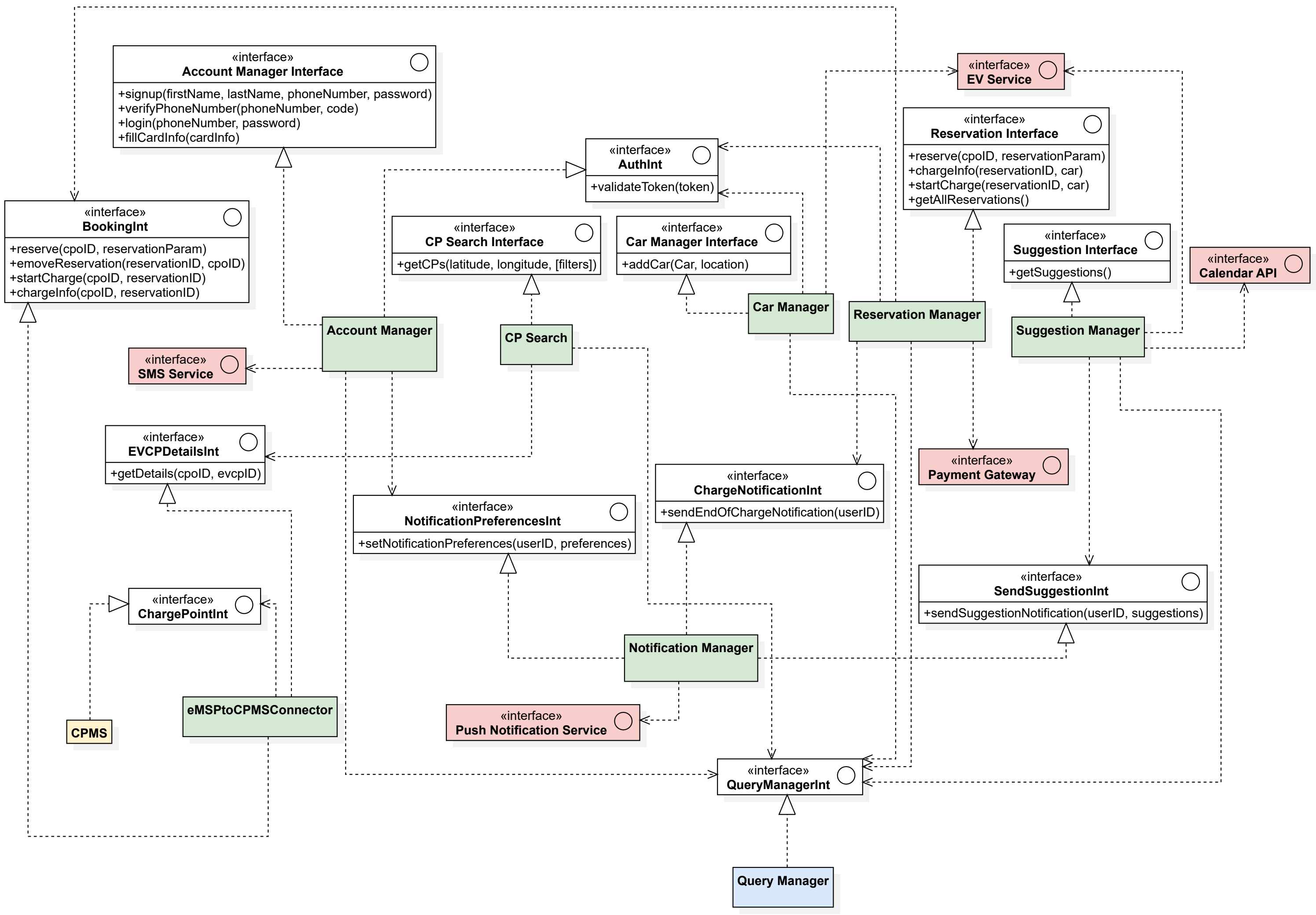


Figure 25: Class Diagram with interfaces of the eMSP System

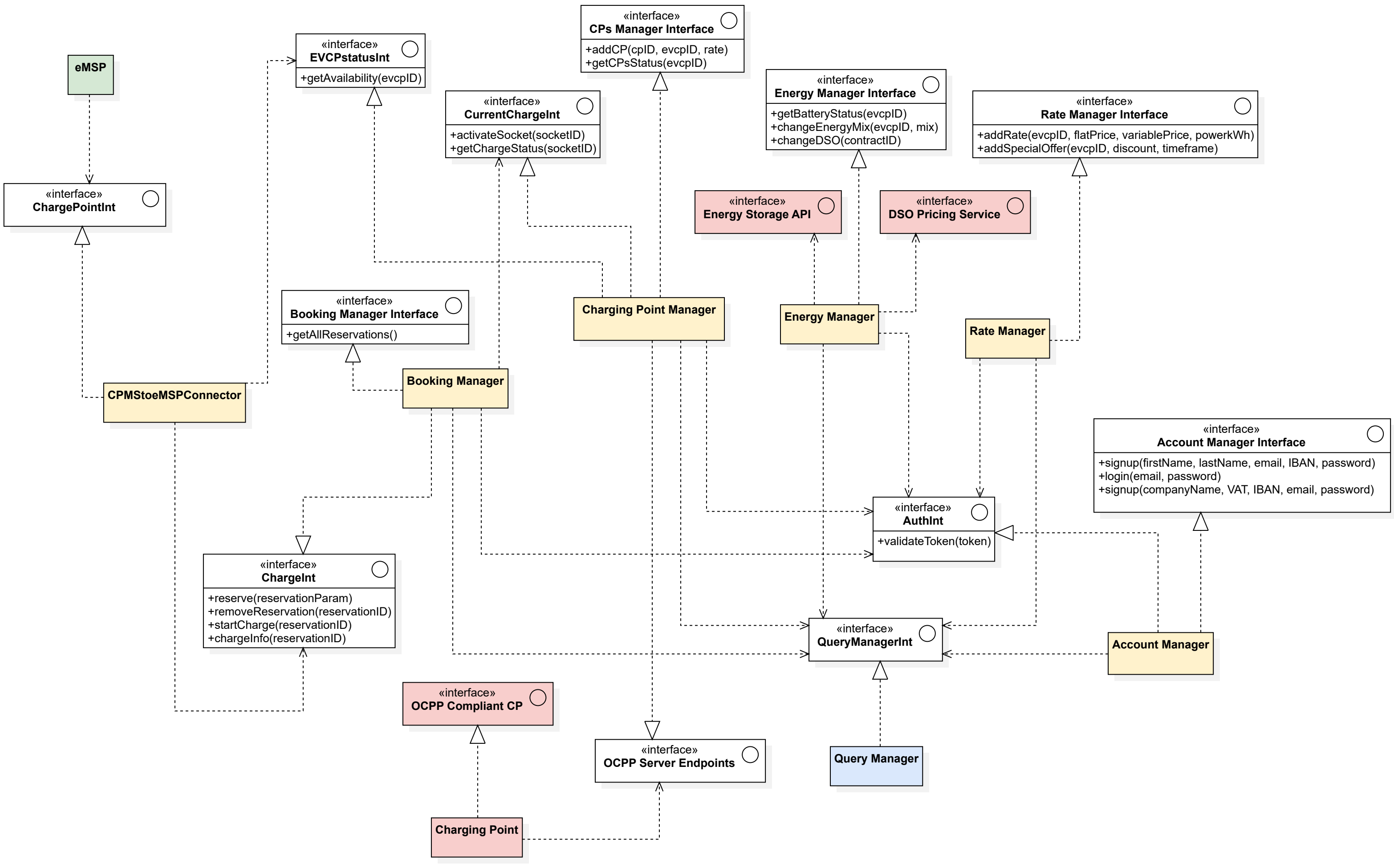

eMSP
«interface» ChargePointInt
«interface» EVCPstatusInt
+getAvailability(evcpID)
«interface» CPs Manager Interface
+addCP(cpID, evcpID, rate)
+getCPsStatus(evcpID)
«interface» CurrentChargeInt
+activateSocket(socketID)
+getChargeStatus(socketID)
«interface» Energy Manager Interface
+getBatteryStatus(evcpID)
+changeEnergyMix(evcpID, mix)
+changeDSO(contractID)
«interface» Rate Manager Interface
+addRate(evcpID, flatPrice, variablePrice, powerkWh)
+addSpecialOffer(evcpID, discount, timeframe)
«interface» Energy Storage API
«interface» DSO Pricing Service
«interface» Booking Manager Interface
+getAllReservations()
Charging Point Manager
Energy Manager
Rate Manager
CPMStoeMSPConnector
Booking Manager
«interface» Account Manager Interface
+signup(firstName, lastName, email, IBAN, password)
+login(email, password)
+signup(companyName, VAT, IBAN, email, password)
«interface» AuthInt
+validateToken(token)
«interface» ChargeInt
+reserve(reservationParam)
+removeReservation(reservationID)
+startCharge(reservationID)
+chargeInfo(reservationID)
«interface» QueryManagerInt
Account Manager
«interface» OCPP Compliant CP
«interface» OCPP Server Endpoints
Query Manager
Charging Point

Figure 26: Class Diagram with interfaces of the CPMS System

2.5.1 API endpoints

In this section the API endpoints are presented. The focus is on the method used, on the parameters required and on the response.

Endpoint: api/cpo/user/signup

Method: POST

Request Body

- companyName: String
- email: String
- password: String

Response

- 200
  - message: String (”A verification code is sent via mail”)
  - id: number
- 400
  - error: String (”eMail not valid”, ”Password must contain at least 8 character (at least one uppercase), at least 1 special symbol and at least 1 number”)

- 409
  - error: String (”An account with the same email already exist, retry”)

Endpoint: api/cpo/user/code

Method: POST

Request Body

- cpoID: number
- code: String (6-digit number)

Response

- 200
  - message: String (”Code inserted is valid, user verified”)
- 400
  - error: String (”Code must be a six-digit number”, ”Wrong code, retry”)
  - id: number (if error is ”Wrong code, retry”)
- 409
  - error: String (”Intrusion”)
- 410
  - error: String (”Too late, the code is expired. Register a new account.”)

Endpoint: api/cpo/user/login

Method: POST

Request Body

- email: String
- password: String

Response

- 200
  - message: String (”Cookie Token has been set”)
- 400
  - error: String (”eMail not valid”, ”Password must contain at least 8 character (at least one uppercase), at least 1 special symbol and at least 1 number”)
- 401
  - error: String (”Email or Password are wrong”)
- 410
  - error: String (”User not verified. Register a new account.”)

Endpoint: api/cpo/book/:evcpID

Method: GET

URL Parameters

- evcpID: number

Response

- 200
  - reservations: Array of Objects
    - id: number
    - evcpID: number
    - driverID: number
    - start: String (date)
    - end: String (date)
- 401
  - error: String (”Unauthorized”)

Endpoint: api/cpo/book/

Method: GET

Response

- 200
  - aggregated: Array of Objects
    - evcpID: number
    - date: String (date)
    - profit: number
- 401
  - error: String (”Unauthorized”)

Endpoint: api/cpo/cp/

Method: GET

Response

- 200
  - EVCPs: Array of Objects
- 401
  - error: String (”Unauthorized”)

Endpoint: api/cpo/cp/:evcpID

Method: GET

URL Parameters

- evcpID: number

Response

- 200
  - evcpDetails: Array of Objects
- 401
  - error: String (”Unauthorized”)

Endpoint: api/cpo/cp/

Method: POST

Request Body

- name: String
- latitude: number
- longitude: number
- address: String

Response

- 200
  - message: String ("EVCP added")
- 401
  - error: String ("Unauthorized")
- 409
  - error: String ("An EVCP in the same spot already exists")

Endpoint: api/cpo/cp/:evcpID

Method: POST

URL Parameters

- evcpID: number

Response

- 200
  - message: String ("CP added")
- 400
  - error: String ("EVCP not found")
- 401
  - error: String ("Unauthorized")

Endpoint: api/cpo/cp/:cpID

Method: POST

URL Parameters

- cpID: number

Request Body

- power: number
- type: String

Response

- 200
  - message: String ("Socket added")
- 400

  - error: String (”CP not found”)
- 401
  - error: String (”Unauthorized”)

Endpoint: api/cpo/energy/:evcpID

Method: GET

URL Parameters

- evcpID: number

Response

- 200
  - dso: Object (Name, contract, price)
- 401
  - error: String (”Unauthorized”)

Endpoint: api/cpo/energy/dso/:evcpID

Method: GET

URL Parameters

- evcpID: number

Response

- 200
  - availableDSOs: Array of Objects
- 401
  - error: String (”Unauthorized”)

Endpoint: api/cpo/energy/dso/:evcpID

Method: POST

URL Parameters

- evcpID: number

Request Body

- dsoID: number

Response

- 200
  - message: String (”DSO updated”)
- 400
  - error: String (”DSO not available”)
- 401
  - error: String (”Unauthorized”)

Endpoint: api/cpo/rate/:evcpID

Method: POST

URL Parameters

- evcpID: number

Request Body

- typeName: String
- flatPrice: number
- variablePrice: number

Response

- 200
  - message: String (”Rate added”)
- 400
  - error: String (”EVCP not found”)
- 401
  - error: String (”Unauthorized”)

Endpoint: api/cpo/rate/:evcpID

Method: GET

URL Parameters

- evcpID: number

Response

- 200
  - rates: Array of Objects
- 401
  - error: String (”Unauthorized”)

Endpoint: api/driver/user/signup

Method: POST

Request Body

- firstName: String
- lastName: String
- phoneNumber: String
- password: String

Response

- 200
  - message: String (”A verification code is sent via SMS”)
  - id: number
- 400

  - error: String (”Phone number not valid”,”Password must contain at least 8 character (at least one uppercase), at least 1 special symbol and at least 1 number”)
- 409
  - error: String (”An account with the same phone number already exist, retry”)

Endpoint: api/driver/user/code

Method: POST

Request Body

- driverID: number
- code: number (6-digit code)

Response

- 200
  - message: String (”A verification code is sent via SMS”)
  - id: number
- 400
  - error: String (”Code must be a six-digit number”,”Wrong code, retry”)
  - id: number (if error is ”Wrong code, retry”)
- 410
  - error: String (”Too late, the code is expired. Register a new account”)

Endpoint: api/driver/user/login

Method: POST

Request Body

- phoneNumber: String
- password: String

Response

- 200
  - message: String (”Cookie Token is set”)
- 400
  - error: String (”Phone number not valid”)

  à
- 401
  - error: String (”Phone number or Password are wrong”)
- 410
  - error: String (”User not verified. Register a new account”)

Endpoint: api/driver/user/notification

Method: PATCH

Request Body

- notificationToken: String

Response

- 200
  - message: String ("Notification Token Set Correctly")
- 401
  - error: String ("Unauthorized")

Endpoint: api/driver/search

Method: GET

URL Query Parameters

- latitude: number
- longitude: number
- filters: Array of Objects

Response

- 200
  - EVCPs: Array of Objects

Endpoint: api/driver/search/:evcpID

Method: GET

URL Parameters

- evcpID: number

Response

- 200
  - evcpDetails: Array of Objects

Endpoint: api/driver/reserve

Method: GET

Response

- 200
  - listOfReservations: Array of Objects
- 401
  - error: String ("Unauthorized")

Endpoint: api/driver/reserve/slot/:evcpID

Method: GET

URL Parameters

- evcpID: number

URL Query Parameters

- socketType: String
- power: number
- date: String

Response

- 200
  - availableSlots: Array of Objects
- 401
  - error: String ("Unauthorized")

Endpoint: api/driver/reserve/duration/:evcpID

Method: GET

URL Parameters

- evcpID: number

URL Query Parameters

- socketType: String
- power: number
- date: String

Response

- 200
  - maxDurations: Array of Objects
- 401
  - error: String ("Unauthorized")

Endpoint: api/driver/reserve/:evcpID

Method: POST

URL Parameters

- evcpID: number

Request Body

- socketType: String
- power: number
- timeFrom: String (date)
- timeTo: String (date)

Response

- 200
  - message: String ("The reservation has been accepted")
- 401
  - error: String ("Unauthorized")

- 409
  - error: String (”Conflict”)

Endpoint: api/driver/reserve/start/:reservationID

Method: POST

URL Parameters

- reservationID: number

Response

- 200
  - message: String (”The charging has been started”)
- 401
  - error: String (”Unauthorized”)
- 404
  - error: String (”Charging not started, error”)

Endpoint: api/driver/car/:carID

Method: GET

URL Parameters

- carID: number

Response

- 200
  - chargeDetails: Array of Objects
- 401
  - error: String (”Unauthorized”)

2.6 Architectural Styles and patterns

- **Three layers and Four Tier**
  We decided to use a three-layer, four-tier architecture for our software system because it offers several benefits. First, it allows us to divide the system into distinct layers (presentation, business logic, data access) and tiers (client, static web server, application server, DBMS), which makes the system more modular and easier to modify or maintain. It also enables us to reuse code and components across different layers and tiers. Additionally, the separation of the layers and tiers allows us to scale the system more easily by distributing the workload across multiple servers. Finally, the architecture enables the use of load balancing and caching techniques to improve performance.
- **RESTful APIs**
  We chose to use a REST API as the backend for the frontend application because it offers several benefits. The API is based on standard web technologies, making it easy to consume and integrate with other systems. It is stateless, which simplifies the development. Additionally, the REST API is scalable, which can improve the overall performance of the system.
- **Adapter Pattern**
  The Query Manager component acts as an adapter between the business logic and the DBMS services. It provides a simplified interface for accessing the DBMS, while hiding the underlying complexity and exposing only a limited set of high-level functions.

2.7 Other design decisions

2.7.1 PWA

The decision to build the web application as a PWA for drivers was made in order to improve the offline functionality, performance, and app-like experience of the application, as well as making it easier to install. This design choice enables the web application to work offline or with a low-quality network connection, using service workers to store assets and data locally. The PWA also improves performance by reducing the need for network requests and loading faster due to the use of service workers. Finally, the PWA can be easily installed on the user's device, similar to a native mobile app, making it convenient for drivers to access the application.

2.7.2 Scale-out

The decision to implement a scale out design in the software was taken to enhance its ability to scale, its availability, and its performance. This design approach enables the system to expand its capacity to cope with increased demand by adding more resources, such as servers or machines, as needed. This can be more cost-effective than upgrading individual components and avoids the need for downtime. The scale out design also improves the system's reliability by providing redundant resources that can take over if any component fails. In addition, the design allows for flexibility, as resources can be added or removed as required. Lastly, the scale out design improves the system's performance by allowing workloads to be processed in parallel rather than sequentially.

2.7.3 Relational Database

We selected a relational database for our system design because it is effective at storing structured data, enforcing data integrity, and providing fast query performance. It can also scale to handle large amounts of data and support many concurrent users. The database allows us to store and retrieve information efficiently, while also ensuring that the data is accurate and consistent.

2.7.4 The separation of eMSP and CPMSs

The decision to design eMSP and CPMSs as independent entities is driven by the need to maintain a consistent interface between them that follows a standard protocol. While this may initially lead to the duplication of some common components, it is necessary because the systems have distinct roles. As the operating environment is subject to change, it is possible that future standard protocols will require eMSP and CPMS to follow a single protocol for searching and booking CPs. By designing them as independent entities, it will be easier to incorporate any new protocol and enable eMSP to access CPMSs even from CPOs not directly subscribed to the system.

3 User Interface Design

In the RASD, mockups for both the CPO and driver web app have already been created. We have included UX flowcharts in this document because they are useful for visualizing the user experience and providing a clear overview of the interactions and steps involved in using the web app. They help to ensure that the web app is easy to use and intuitive for users.

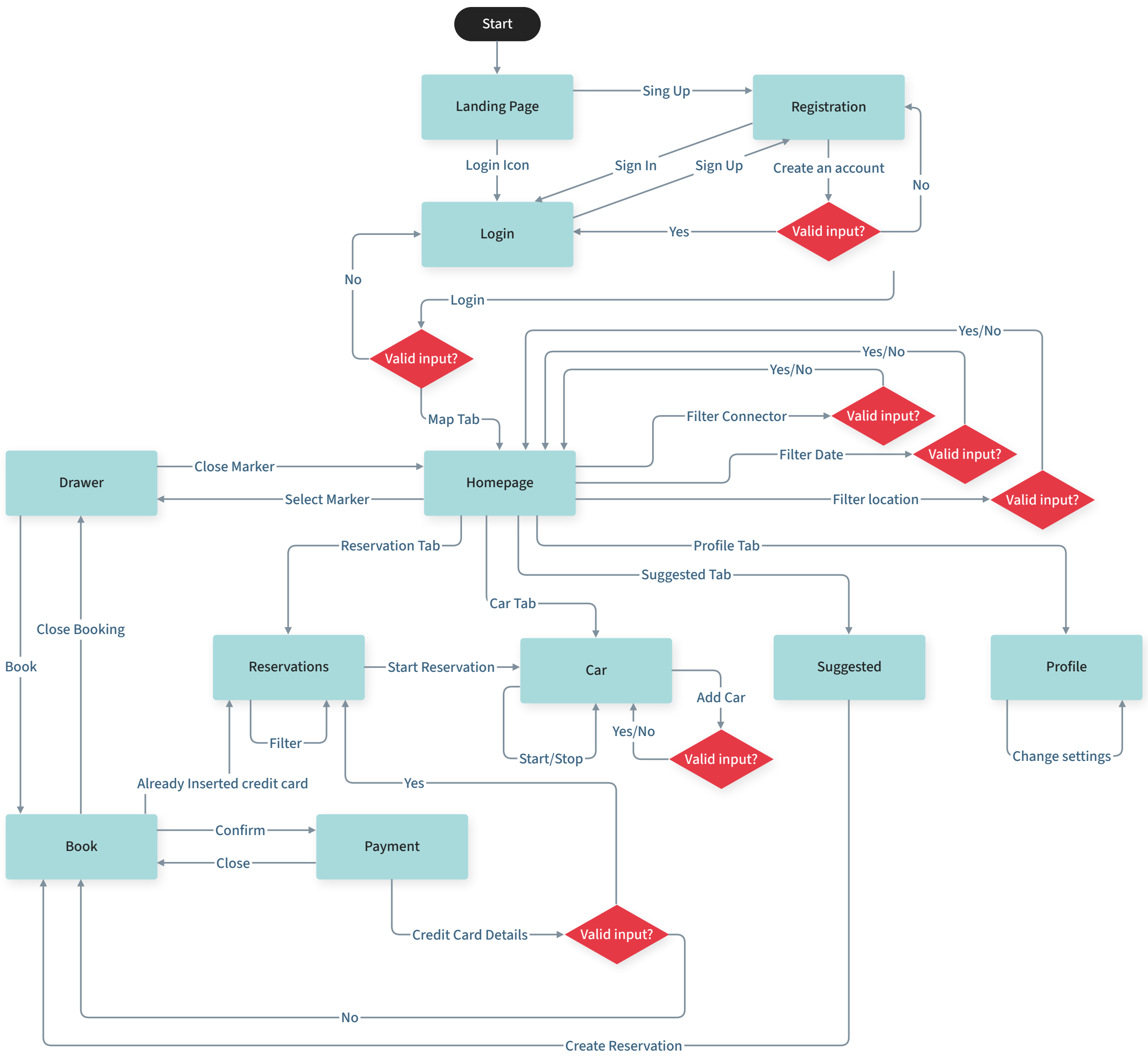


Figure 27: EVdriver

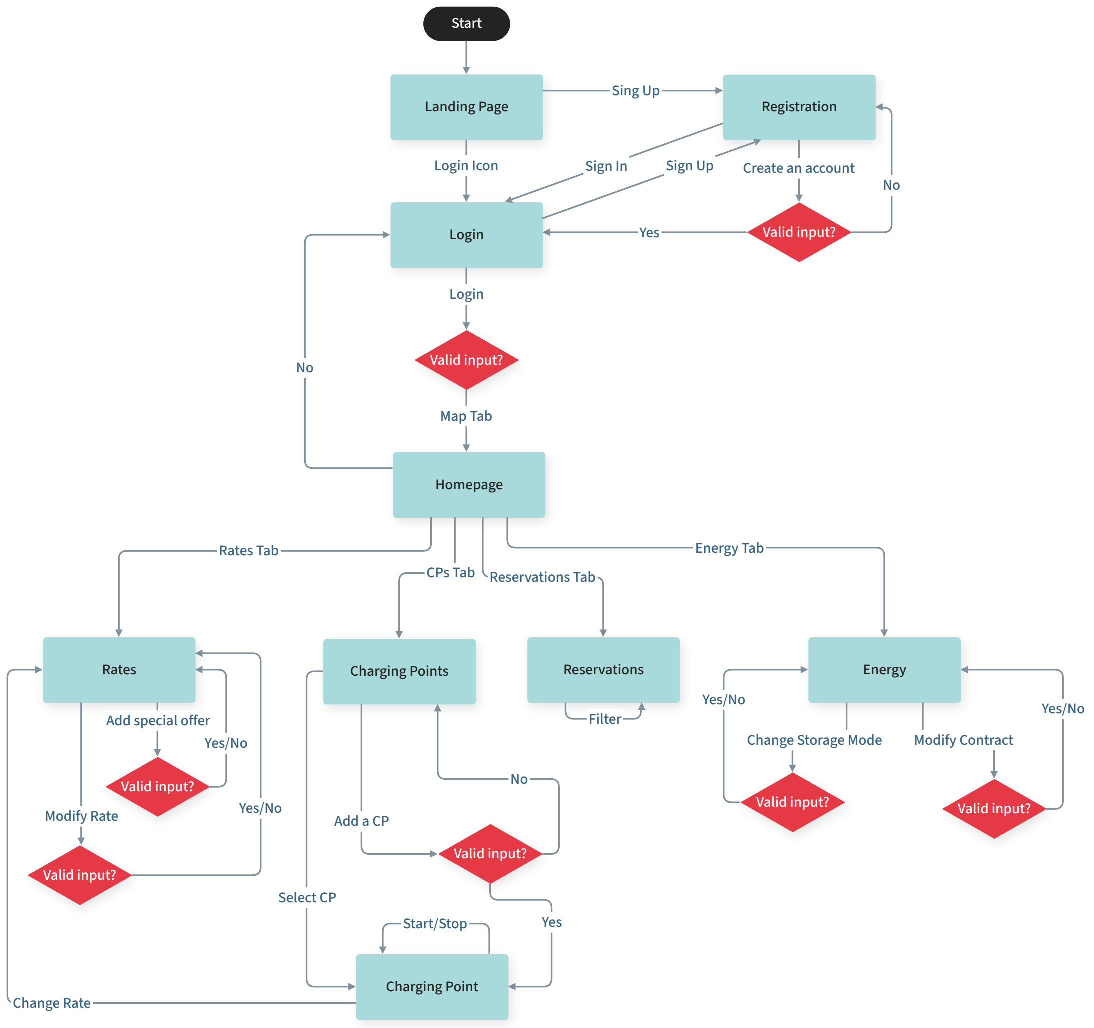


Figure 28: Charging Point Operator

4 Requirements traceability

In this section, it is provided the mapping between the requirements defined in the RASD document and the components defined in the DD document. For each requirement is used the component "Query Manager" to interact with the DB.

R1 The system must allow unregistered CPO to register an account

CPMS's Account Manager In order to register

R2 The system must allow registered CPO to login

CPMS's Account Manager In order to authenticate

R3 The system must allow authenticated CPOs making a special offer on their CPs prices

CPMS's Account Manager In order to authenticate

Rate Manager In order to add a special rate

R4 The system must allow authenticated CPOs monitoring the charging process to infer when the battery is full

CPMS's Account Manager In order to authenticate

Charging Points Manager In order to monitor the charging process and infer

R5 The system must allow authenticated CPOs retrieving the amount of energy available in their EVCPs batteries

CPMS's Account Manager In order to authenticate

Energy Manager In order to retrieve the amount of energy available in their EVCPs batteries

R6 The system must allow authenticated CPOs retrieving the number of vehicle being charged in their EVCPs and for each vehicle the amount of absorbed power

CPMS's Account Manager In order to authenticate

Charging Points Manager In order to retrieve the number of vehicle being charged in their EVCPs and for each vehicle the amount of absorbed power

R7 The system must allow authenticated CPOs retrieving the remaining charge time for each connected vehicle

CPMS's Account Manager In order to authenticate

Charging Points Manager In order to retrieve the remaining charge time for each connected vehicle

R8 The system must allow authenticated CPOs retrieving details on active and historical reservations on their EVCPs

CPMS's Account Manager In order to authenticate

Booking Manager In order to retrieve details on active and historical reservations on their EVCPs

R9 The system must allow authenticated CPOs acquiring information from the DSOs about the current price of energy

CPMS's Account Manager In order to authenticate

Energy Manager In order to acquire information from the DSOs about the current price of energy

R10 The system must allow authenticated CPOs deciding from which DSO to acquire energy from

CPMS's Account Manager In order to authenticate

Energy Manager In order to decide from which DSO to acquire energy from

R11 The system must dynamically decide where to get energy for charging (electrical grid, battery or a mixture)

CPMS's Account Manager In order to authenticate

Energy Manager In order to decide where to get energy for charging (electrical grid, battery or a mixture)

R12 The system must allow authenticated CPOs statically deciding where to get energy for charging (electrical grid, battery or a mixture)

CPMS's Account Manager In order to authenticate

Energy Manager In order to statically decide where to get energy for charging (electrical grid, battery or a mixture)

R13 The system must allow authenticated CPOs adding, modifying and deleting CPs

CPMS's Account Manager In order to authenticate

Charging Points Manager In order to add, modify and delete a CP

R14 The system must allow authenticated CPOs changing availability status of their CPs

CPMS's Account Manager In order to authenticate

Charging Points Manager In order to change availability status of their CPs

R15 The system must allow unregistered users to register an account

eMPS's Account Manager In order to register

R16 The system must allow registered users to login

eMPS's Account Manager In order to authenticate

R17 The system must allow authenticated users to personalize their experience by providing information of their EV

eMPS's Account Manager In order to authenticate

Car Manager In order to provide information of their EV

R18 The system must allow users to search for CPs in the map

CP Search In order to search for CPs in the map

R19 The system must show to the users CPs nearby their current position

CP Search In order to geolocating the current position and search for nearby CPs in the map

R20 The system must allow retrieving details on a given CP regarding connector types supported and cost of the charge

CP Search In order to get the information about the CPs

eMSPtoCPMSConnector In order to communicate to a specific CPMS

CPMStoeMSPConnector In order to respond to the eMPS

Charging Points Manager In order to retrieve data of the CPs

R21 The system must allow authenticated user to book a CP for a certain time interval

eMPS's Account Manager In order to authenticate

Reservation Manager In order to book a CP for a certain time interval

eMSPtoCPMSConnector In order to communicate to a specific CPMS

CPMStoeMSPConnector In order to respond to the eMPS

Booking Manager In order to let booking the requested CP for the requested time interval

R22 The system must allow booking a CP if and only if it is free for the specified time interval

Reservation Manager In order to book a CP for a certain time interval

eMSPtoCPMSConnector In order to communicate to a specific CPMS

CPMStoeMSPConnector In order to respond to the eMPS

Booking Manager In order to let booking the requested CP for the requested time interval

R23 The system must notify users when the charging shift is about to start

eMPS's Account Manager In order to authenticate

Notification Manager In order to notify

R24 The system must allow authenticated users to start the charge

eMPS's Account Manager In order to authenticate

Reservation Manager In order to start the reserved charge

eMSPtoCPMSConnector In order to communicate to a specific CPMS

CPMStoeMSPConnector In order to respond to the eMPS

Booking Manager In order to verify that reservation can be started

Charging Points Manager In order to start the charge

R25 The system must suggest users when and where to charge based on daily schedule, special offers and availability

eMPS's Account Manager In order to authenticate

Suggestion Manager find the best suggestion according to daily schedule, special offer and availability

R26 The system must allow authenticated users to monitor the charging status

eMPS's Account Manager In order to authenticate

Reservation Manager In order to see details of the reserved charge

eMSPtoCPMSConnector In order to communicate to a specific CPMS

CPMStoeMSPConnector In order to respond to the eMPS

Booking Manager In order to identify the correspondent socket CP of the charge

Charging Points Manager In order to see details of the charge

R27 The system must notify authenticated users when the charging process is completed

eMPS's Account Manager In order to authenticate

Notification Manager In order to notify

R27 The system must notify authenticated users when the charging process is completed

eMPS's Account Manager In order to authenticate

Notification Manager In order to notify

R28 The system must allow authenticated users to pay for the charge

eMPS's Account Manager In order to authenticate

Notification Manager In order to notify the reservation manager that the charge has been completed

Reservation Manager In order to notify the payment Gateway with the information of the total cost of the charge

R29 The system must allow authenticated users to delete a reservation

eMPS's Account Manager In order to authenticate

Reservation Manager In order to delete a reserved charge for the user

eMSPtoCPMSConnector In order to communicate to a specific CPMS

CPMStoeMSPConnector In order to respond to the eMPS

Booking Manager In order to delete the reservation from the DB and notify the payment Gateway to refund the pre authorized payment

R30 The system must allow authenticated users to view historical reservations

eMPS's Account Manager In order to authenticate

Reservation Manager In order to view historical reservation

NFR traceability

NFR are also guaranteed thanks to the design choices made in this document, in particular:

- **Easy usability**
  The user interface of the application is designed to be easy to use and intuitive, with a focus on providing a simple experience for drivers, the primary target of the application. This ensures that the application is user-friendly.
- **Availability and Reliability**
  The S2B can achieve high levels of availability through the use of the scale-out method, which involves creating copies of the running application, known as clones. These clones run on separate physical nodes and can work in parallel to provide the requested service, even if one of the clones experiences a failure. The goal is to achieve at least 99.5% availability for each tier, resulting in an overall system availability of 99.5%.
- **Easy maintainability**
  The S2B is designed to be modular. This can help improve the maintainability of the system by making it easier to identify and fix issues, update and modify individual components, test the system, and use reusable components.

5 Implementation, integration and testing plan

This section will be dedicated to discuss how the subcomponents' implementation is planned, to present the integration plan and then to explain the tests dedicated to evaluate if the System behaviours as expected.

5.1 Development Process and Approach

The application is composed of three layers (client, business and data) and these layers can be implemented in parallel and integrated. The tiers can be unit tested independently and after the integration they can be tested at the end as the whole System. The whole System can be implemented, integrated and tested exploiting bottom-up approach. The decision to use this approach derives from the possibility to incrementally develop the system by proceeding implementing in parallel the different tiers. The components are tested and integrated with other components to evaluate the dependencies between components of the same subsystem.

5.1.1 Frontend

It consists of the PWA and web application for the EV Drivers and the web application for CPOs that can be developed and tested independently since are decoupled and doesn't communicate directly. Both the two components rely on REST API to interact with the business logic and can be unit tested by mocking the REST APIs.

5.1.2 Backend

It consists of business logic and the data logic. The application server of the business logic is composed by the Query Manager and two important components, the eMSP and the CPMS that need to communicate each other. The developers can be work separately on eMSP and CPMS and integrate and test at the end of their development the two components to evaluate the dependency and the interactions between them. These whole business logic can be tested independently from the client logic, speeding up the development process. In the following two sections will be analyzed the complexity and the importance for the subcomponents of the eMSP and the CPMS.

- **eMSP**
  In the following two sections will be analyzed the complexity and the importance for the subcomponents of the eMSP.

Component	**Importance**	**Complexity**
Account Manager	High	Medium
CP Search Manager	High	Medium
Car Manager	Medium	Medium
Reservation Manager	High	High
Suggestion Manager	Medium	Medium
Notification Manager	Medium	Medium
eMSPtoCPMSConnector	High	High
Query Manager	High	Low

- **CPMS**
  In the following two sections will be analyzed the complexity and the importance for the subcomponents of the CPMS.

Component	**Importance**	**Complexity**
Account Manager	High	Medium
Rate Manager	Medium	Low
Energy Manager	Medium	Medium
Booking Manager	High	High
Charging Points Manager	High	High
Query Manager	High	Low

5.1.3 External components

All the external APIs are provided by third parties and are supposed to be reliable and conform to their specification.

5.2 Implementation Plan

The implementation of the Application Server will be divided between eMSP and CPMS a can be done in parallel.

5.2.1 eMSP

1. **Query Manager**: It is the first component to be implemented since all the other component relies on This on to interact with the Database. It is the only component in which is necessary to implement the queries to the database.
2. **Notification Manager**: Then component can be implemented, it offers two interfaces to two components that will later be developed.
3. **eMSP's account Manager**: This is the component that permits to register or log in a user. It has an authorization interface that is used by the other components to verify at each request that the user has the permissions to do the requested operations.
4. **eMSPtoCPMSConnector**: The development of this module is particularly critic because is the component dedicated to the communication with the CPMS. The dialogue between the two can be simulated during the implementation and tested after the integration.
5. **Charging Points Manager, Reservation Manager, Suggestion Manager, Car Manager**: These components can be implemented in parallel since there are no dependencies between them.

5.2.2 CPMS

1. **Query Manager**: It is the first component to be implemented since all the other component relies on This on to interact with the Database. It is the only component in which is necessary to implement the queries to the database.
2. **CPMS's account Manager**: This is the component that permits to register or log in a user. It has an authorization interface that is used by the other components to verify at each request that the user has the permissions to do the requested operations.
3. **Rate Manager**: This component is necessary to enable the creation of new rates to associate with the CPs.
4. **Charging Points Manager**: The dialogue between the CPMS and the CPs is done through this Charging Points Manager that needs to be implemented compliant with the OCPP protocol used by the CPs.
5. **Booking Manager**: The Reservation Manager is the component that manages all the reservation done through the eMSP to the specific CPMS. This component is used to access a reservation and can call the Charging Points Manager to do operations on CPs of a specific reservation.

6. **CPMStoeMSPConnector**: The development of this module is particularly critic because is the component dedicated to the communication with the eMSP. The dialogue between the two can be simulated during the implementation and tested after the integration.
7. **Energy Manager**: This component has a medium complexity because it interacts with the DSO Pricing Service and Energy Storage API.

5.3 Integration Plan

The section describes the integration plan of the different components and subcomponents of the eMAll System. The graphs below show the dependencies of a component or subcomponent on another component or subcomponent. Each subcomponent before being integrated with others subcomponents to form a component need to be unit tested and after the integration the integrated component will be tested. The testing can be done in parallel between eMSP and CPMS.

5.3.1 eMSP

The first components to be tested together are the Query Manager and the DBMS as they are the components necessary for all the others when handling data.

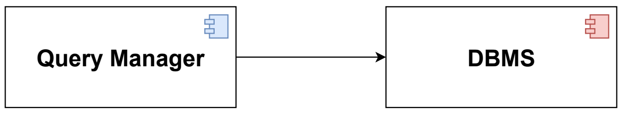


Figure 29: Data Subsystem

Before implementing the Account Manager it is preferred to develop the Notification Manager.

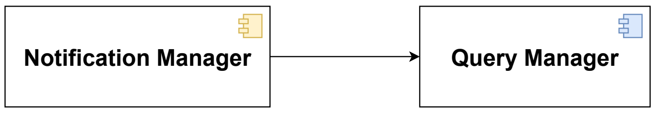


Figure 30: Notification Subsystem

Then it can be implemented the Account Manager that provides the account managing functions and the authorization interface to verify the permissions to do operations with the other components. It handles personal user's data and for this reason needs to be tested carefully.

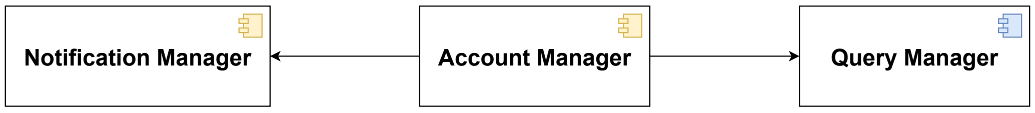


Figure 31: Account Subsystem

The eMSPtoCPMSConnector can be implemented after the Account Manager, a stub of the CPMS can be implemented at this point to simulate the response of the CPMS on the other side of the communication. This stub is an exception to the bottom-up approach, but enables the possibility to develop separately the eMSP and the CPMS.

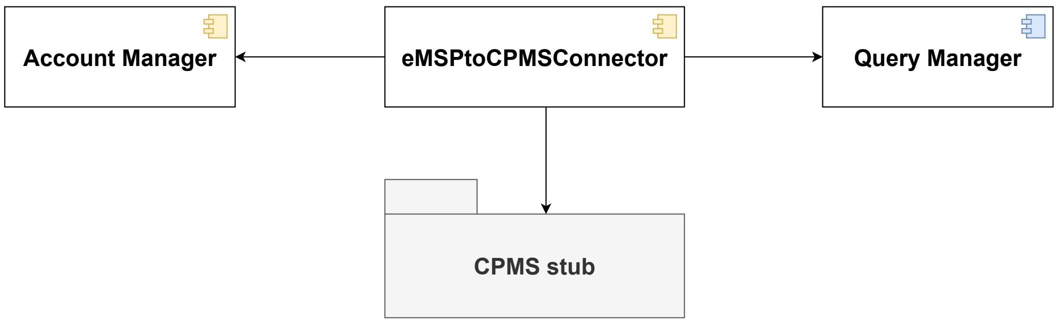


Figure 32: Connector Subsystem

Then the following subsystems could be developed in any order or in parallel because they are decoupled. All the components require the eMSPtoCPMSConnector to simulate and test efficiently the implementation. The develop can start from the most complex tasks such as the Charging Point Manager and the Reservation Manager. Then the Suggestion Manager and the Car manager can be implemented.

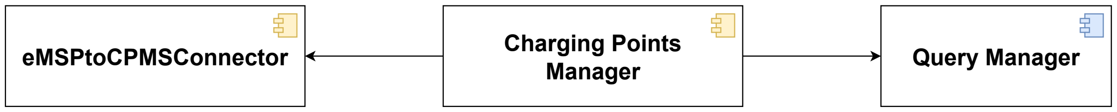


Figure 33: Charging Points Subsystem

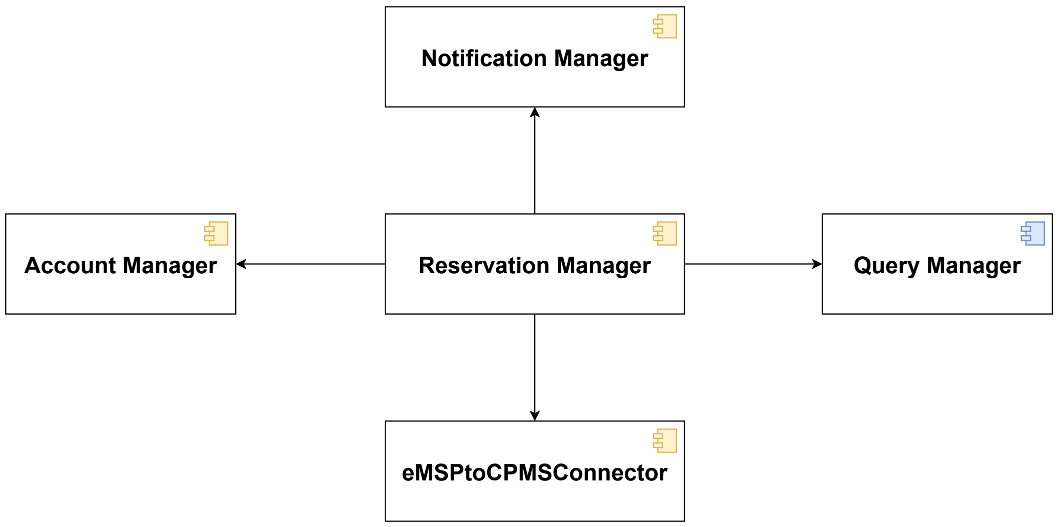


Figure 34: Reservation Subsystem

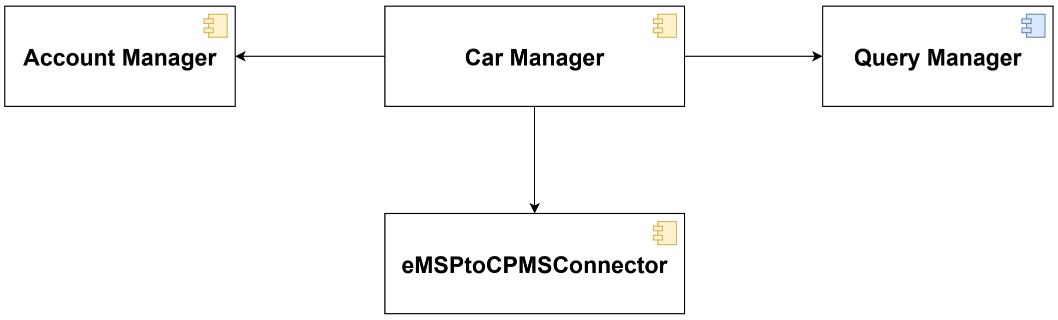


Figure 35: Car Subsystem

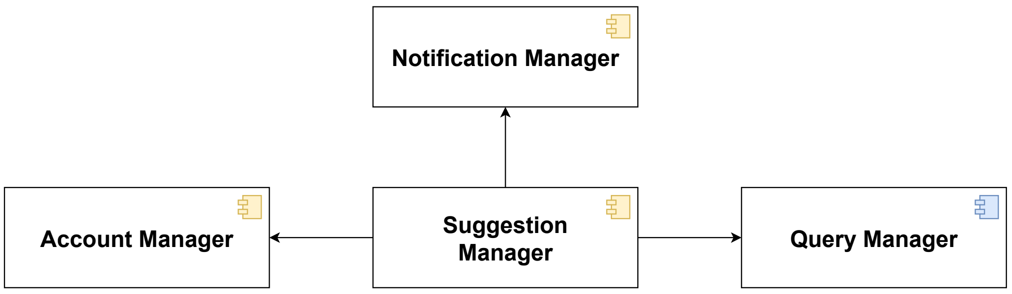


Figure 36: Suggestion Subsystem

After having implemented and tested all the subsystems, the client and the eMSP can be integrated and tested together.

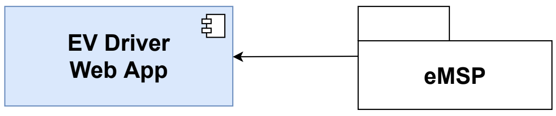


Figure 37: Client Application Integration

The final result of the integration of the eMSP Web Application and the eMSP is the following:

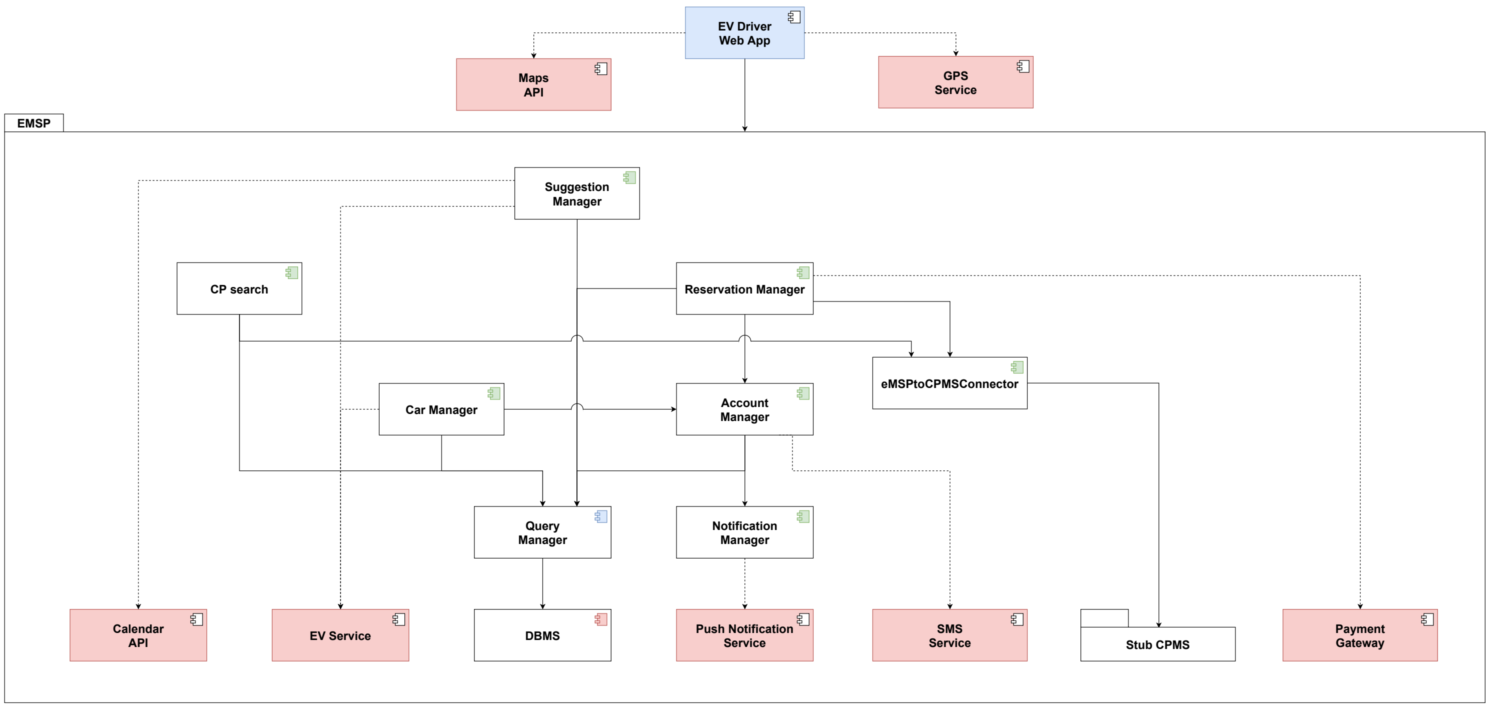


Figure 38: eMSP overview

5.3.2 CPMS

The first components to be tested together are the Query Manager and the DBMS as they are the components necessary for all the others when handling data.

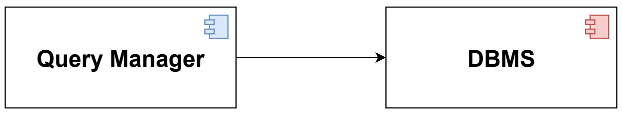


Figure 39: Data Subsystem

Then it can be implemented the Account Manager that provides the account managing functions and the authorization interface to verify the permissions to do operations with the other components. It handles personal CPO's data and for this reason needs to be tested carefully.

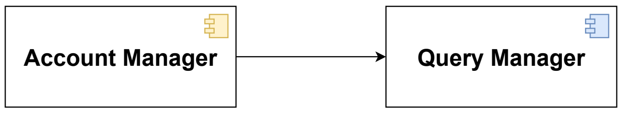


Figure 40: Account Subsystem

The Rate Manager can be implemented to provide the functions to create new rates that can also be special.

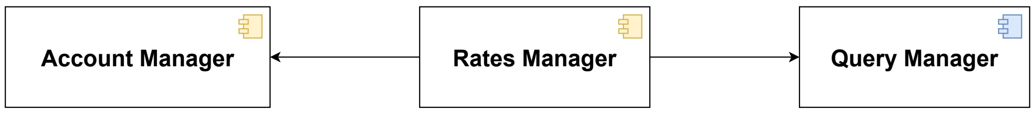


Figure 41: Rates Subsystem

Then can be implemented the Charging Points Manager. This component has an interface to the booking manager and uses the OCPP compliant CP to do manage the CP using the standard OCPP protocol.

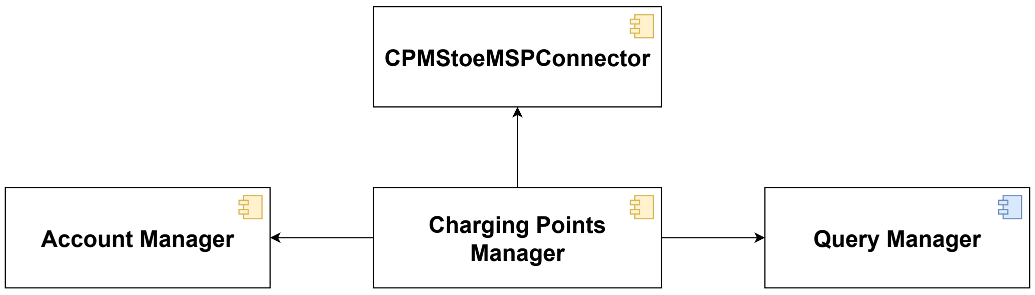


Figure 42: Charging Points Subsystem

After the implementation of the Charging Points Manager the Booking Manager can be implemented because depends on the interface offered by the former. The usage of the interface is clarified in the second chapter.

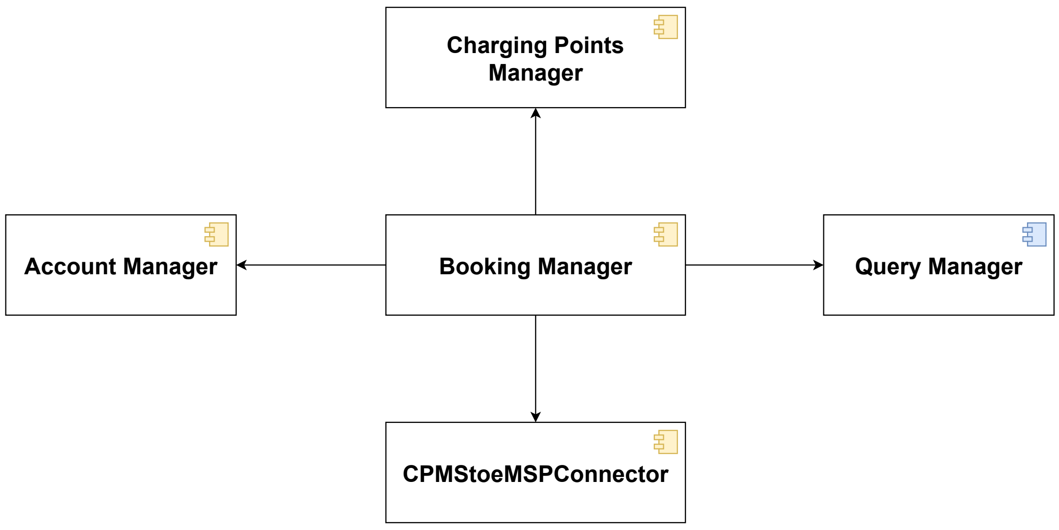


Figure 43: Booking Subsystem

The CPMStoeMSPConnector can be implemented after the Booking and Charging Points Managers. A stub of the eMSP can be implemented at this point to simulate the requests coming from the eMSP on the other side of the communication. This stab is an exception to the bottom-up approach, but enables the possibility to develop separately the eMSP and the CPMS.

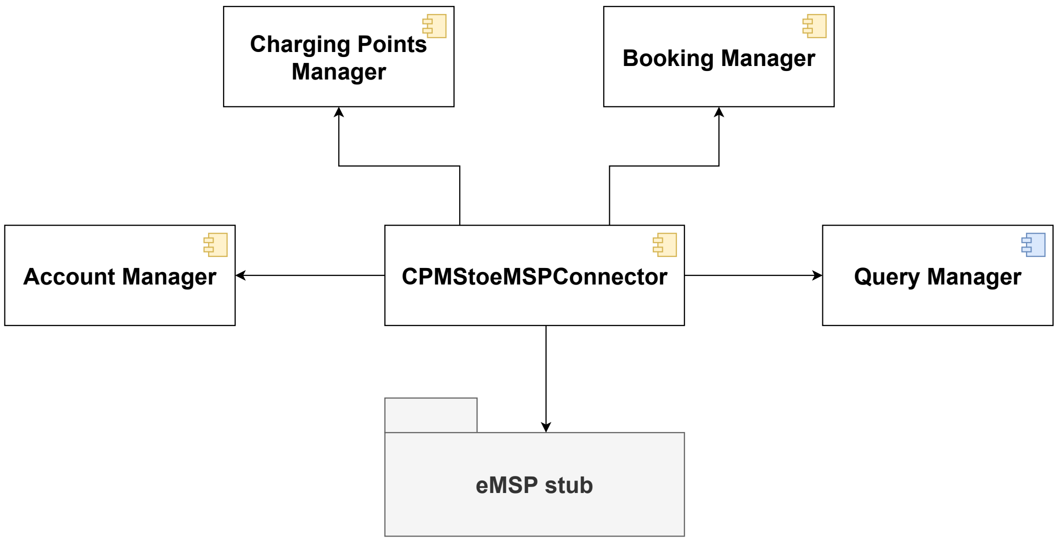


Figure 44: Connector Subsystem

Then can be implemented the Energy Manager that doesn't depend on the others components. The final result of the integration of the CPMS Web Application and the CPMS is the following:

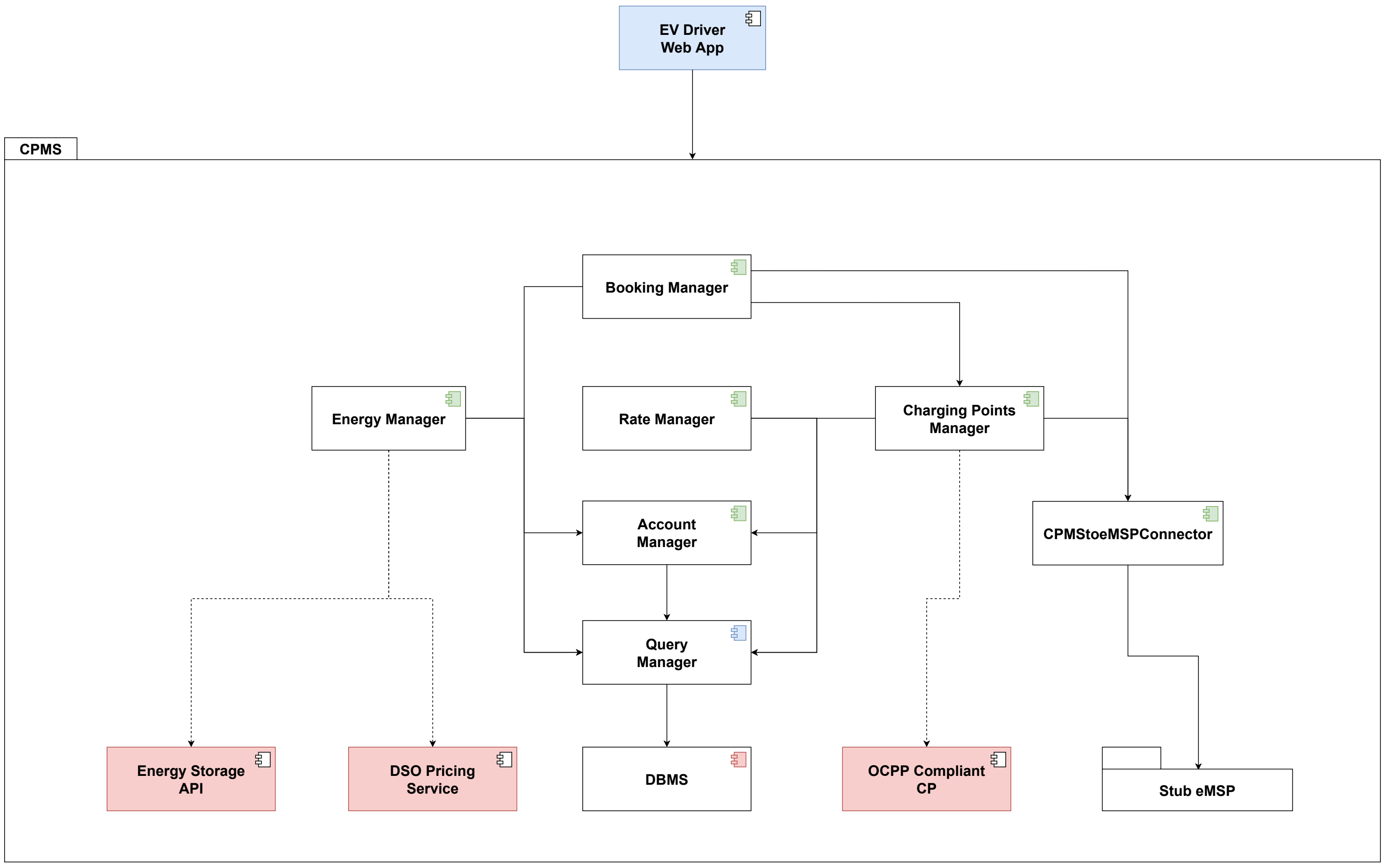


Figure 45: CPMS Subsystem

5.4 System Testing

Eventually, when the integration of the components have taken place take place the testing of the entire System. The aim is to verify the functional and non-functional requirements and must take place in a testing environment that is as close as possible to the production environment. The eMall System will undergo the following test:

- **Functional testing**: The system is tested against the functional requirements and specification specified on the RASD document.
- **Performance testing**: Check wether a software remains functional with increase demand and various environment conditions.

6 Effort Spent

Pair working

DD Structure analysis, overview architecture brainstorming	**4h**
Component Diagram	**5h**
Runtime View	**8h**
Database Entity Relationship Diagram	**2h**
Final Review	**2.5h**

Giovanni

Introduction section	**1h**
Deployment View	**1.5h**
Review: ER Diagram	**1.5h**
NFR Traceability	**1h**
Component Interfaces: diagram and description	**2.5h**
Design Choices description	**0.5h**

Matteo

Requirements traceability	**5h**
Component Diagram	**1h**

Lorenzo

User Interface Design	**3h**
Development process and approach	**1.5h**
Implementation Plan	**1h**
Integration Plan	**4h**
System Testing	**0.5h**

7 References

- PWA
  - https://web.dev/progressive-web-apps/
  - https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps
- Push Notification
  - https://web.dev/notifications/
  - https://firebase.google.com/docs/cloud-messaging
- Three Tier Architecture
  - https://www.ibm.com/topics/three-tier-architecture
  - AWS Three Tier Architecture Docs
- RESTful API
  - https://restfulapi.net/
  - https://restcookbook.com/
- OCPP
  - https://www.openchargealliance.org/protocols/ocpp-201/
- Scale-out
  - https://learn.microsoft.com/en-us/azure/architecture/guide/design-principles/scale-out
- SMS, email Service: Twilio
  - https://www.twilio.com/
- Payment Gateway: Stripe
  - https://stripe.com/docs/development