POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Software Engineering 2
Design Document

Author(s): **Irfan Cela - 10694934**

Mario Cela - 10685242

Alessandro Cogollo - 10571078

Academic Year: 2022-2023

Contents

Contents **i**

1 Introduction **1**
1.1 Purpose . . . 1
1.2 Scope . . . 1
1.3 Definitions, Acronyms, Abbreviations . . . 1
1.4 Reference Documents . . . 1
1.5 Document Structure . . . 1

2 Architectural Design **3**
2.1 Overview . . . 3
2.2 Component View . . . 3
2.2.1 Component diagram . . . 4
2.2.2 Components description . . . 5
2.3 Deployment View . . . 8
2.3.1 Connection to the server . . . 9
2.3.2 Promotions . . . 10
2.3.3 EVD interactions . . . 11
2.3.4 CPOs DBMS . . . 11
2.3.5 Charging stations communication . . . 12
2.3.6 Services . . . 12
2.4 Runtime View . . . 13
2.5 Component Interfaces . . . 30
2.6 Selected Architectural Styles and Patterns . . . 30
2.7 Other Design Decisions . . . 31
2.7.1 Client-Server architecture . . . 31
2.7.2 Thin client . . . 31
2.7.3 Shared databases . . . 31

3 User Interface Design **33**
3.1 General Overview 33
3.2 EVD Interface 33
3.3 CPO Interface 33

4 Requirements Traceability **35**
4.1 Functional Requirements Traceability 35
4.2 Non Functional Requirements Traceability 35

5 Implementation, Integration and Test Plan **37**
5.1 Implementation 37
5.2 Integration 37
5.3 Test Plan 37

6 Effort Spent **39**

7 References **41**
7.1 Paper References 41
7.2 Used Tools 41

List of Figures **43**

List of Tables **45**

1 Introduction

1.1. Purpose

1.2. Scope

1.3. Definitions, Acronyms, Abbreviations

1.4. Reference Documents

1.5. Document Structure

2 | Architectural Design

2.1. Overview

2.2. Component View

The component diagram shows all the identified components. It also describes the relations between the modules, representing the verse of the communication flow and the actors of it. Before showing the diagram, there must be clarifications about it:

- Two client applications are represented, one for both EVD and CPO. They are the same client application. We preferred to double the module in the diagram to facilitate comprehension of how users communicate with the system.
- For the same reasons, there are duplicated interfaces offered by components.
- The diagram shows a macro component called `eMALL` that represents the whole system. Outside the system, they are shown the external services, the DBMSs, and the client application. Communication between modules and `eMALL` happens thanks to interfaces that the system offers or exploits, depending on the component.

After the diagram, it will follow a brief description of each identified component.

2.2.1. Component diagram

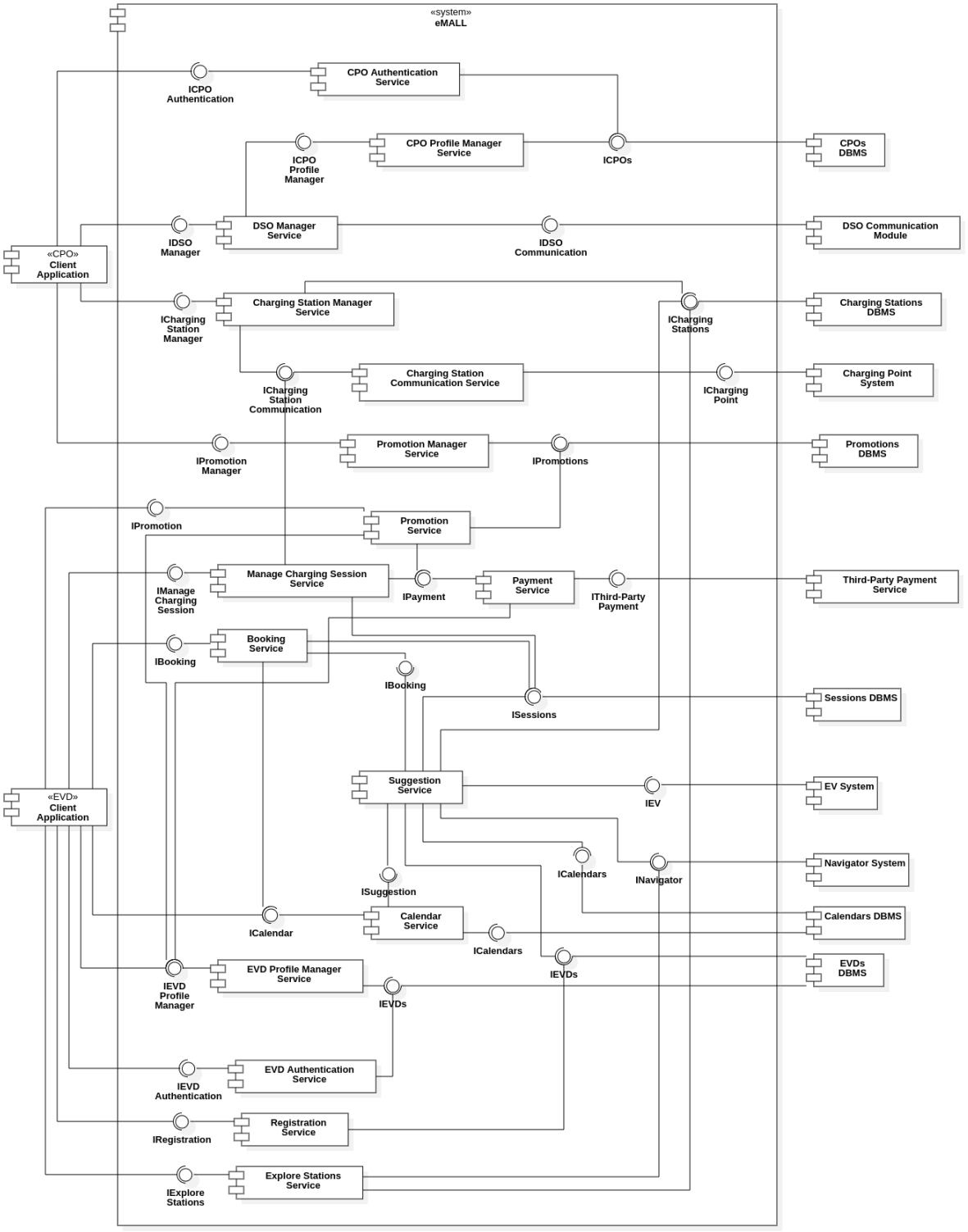

Figure 2.1: Component diagram of the eMALL system.

2.2.2. Components description

The components are:

- **Client Application.** `Client Application` represent the client system used to connect and to communicate with `eMALL`.
- **Registration Service.** `Registration Service` handles the process of the creation of a new account requested by a new EVD user. It communicates with `EVDs DBMS` to save the information about the new created account.
- **EVD Authentication Service.** `EVD Authentication Service` handles the login process requested by an EVD. To do that, it communicates with `EVDs DBMS` to validate the inserted credentials.
- **EVD Profile Manager Service.** `EVD Profile Manager Service` offers the possibility to query the `EVDs DBMS` in order to get requested information. In general, it is the service offered to the user to get or update information of the profile. So, he communicates with `EVDs DBMS`.
- **Explore Stations Service.** `Explore Stations Service` is the service used by the EVDs to navigate into the map of charging station that can be found around user's position. To do that, he needs stations position, so it communicates with `Charging Stations DBMS`. Furthermore, it communicates also with the external service `Navigator System`, used to elaborate positions and to show them into the map.
- **Booking Service.** `Booking Service` is the module used to handle booking requests. It communicates with `Sessions DBMS` to save the information about bookings and with `Calendar Service` to start the process of insertion of a new activity into EVD's calendar representing the booked reservation.
- **Manage Charging Session Service.** `Manage Charging Session Service` handles the charging session process. It allows the EVDs to start and interrupt the session and to pay for the service the user enjoyed. Furthermore, it sends to the EVDs notifications about ongoing session status. The module communicates with `Charging Station Communication Service` to delegate the communication with the charging point where the EVD wants to charge the EV. As said before, the service offers the EVD the interface to make the payment of the session, so it communicates with `Payment Service`. After the payment, the module saves the receipt of the session communicating the `Sessions DBMS` module.

- **Promotion Service.** `Promotion Service` offers the EVDs the possibility to activate special promotions activated by CPOs. To get the list of active promotions, it communicates with `Promotions DBMS`. The module communicates also with `EVD Profile Manager Service` to delegate the save of the activation of a special offer into the `EVDs DBMS`.
- **Calendar Service.** `Calendar Service` is module that manages EVDs calendar. It offers the possibility to visualize the calendar, saved events and their details. It also offers the functionality of activity insertion. To save all this information, the module communicates with `Calendars DBMS`. After a new activity is inserted, `Calendar Service` activates the process offered by `Suggestion Service` in order to create suggestion about when and where the EVD should charge the EV.
- **Suggestion Service.** `Suggestion Service` communicates with different modules to create suggestions. It communicates with:
  - `EVDs DBMS` to get user's EV specifications.
  - `Calendar DBMS` to get EVD's schedule to know precedent events that could affect the research.
  - `EV System` to get current EV status.
  - `Charging Stations DBMS` to get positions of the memorized charging stations.
  - `Navigation System` to calculate the path between the positions defined in the schedule of the EVD, and to identify which charging stations can be suggested to the user.
  - `Sessions DBMS` to get schedules of the charging point, getting in this way their available timeframes, and to see if there are other bookings done by the EVD.
  - `Booking Service` to start the booking process after the EVD confirms the received suggestion.
- **Payment Service.** `Payment Service` offers the possibility to pay for a service the EVD enjoyed. It communicates with `EVD Profile Manager Service` to get EVD's payment methods. Once it has the needed information, it communicates with `Third-Party Payment Service` to make the payment.
- **CPO Authentication Service.** `CPO Authentication Service` handles the login process requested by a CPO. To do that, it communicates with `CPOs DBMS` to validate the inserted credentials.

- **CPO Profile Manager Service.** `CPO Profile Manager Service` offers the possibility to query the `CPOs DBMS` in order to get requested information. In general, it is the service offered to the user to get or update information of the profile. So, he communicates with `CPOs DBMS`.
- **Charging Station Manager Service.** `Charging Station Manager Service` is the service offered to CPOs to manage their charging stations and their charging points. One of the functionalities is to plan a maintenance session for a charging station. To do that, the module communicates with the `Charging Station Communication Service` module. Finally, it communicates with `Charging Stations DBMS` to store the information.
- **Charging Station Communication Service.** `Charging Station Communication Service` is the module that communicates with charging points. Messages are exchanged when an EVD starts or ends the charging session or when the CPO plans a maintenance session.
- **Promotion Manager Service.** `Promotion Manager Service` is used by CPOs to manage their promotions. The module communicates with `Promotions DBMS` to save or update promotions information.
- **DSO Manager Service.** `DSO Manager Service` is the module aimed for the communication with DSOs. To do that, it exchanges messages with the external service `DSO Communication Module`. When a CPO decides to change its electricity provider, the module delegates `CPO Profile Manager Service` to save the information into the `CPOs DBMS`.
- **EVDs DBMS.** `EVDs DBMS` is the system used to save all the information about EVDs, such as credentials, EVs, active promotions.
- **Calendars DBMS.** `Calendars DBMS` is the system used to save all the information about activities inserted by EVDs and to save location and hour of a booked charging session. In this way, it is easily obtainable by the EVD.
- **Sessions DBMS.** `Sessions DBMS` is the system used to save information about bookings specifying data that would be useless for the EVD. It is also used to save the receipts of the charging sessions done by the EVDs.
- **CPOs DBMS.** `CPOs DBMS` is the system used to save all the information about CPOs, such as credentials, company information.
- **Charging Stations DBMS.** `Charging Stations DBMS` is the system used to save

the information about charging stations and charging points.

- **Promotions DBMS.** `Promotions DBMS` is the system used to save information about promotions that have been activated by the CPOs.
- **EV System.** `EV system` is an external service that gives the system the possibility to retrieve information about EVDs EV.
- **Navigator System.** `Navigator System` is an external service that gives the system the possibility to work on positions and elaborate paths between locations.
- **Third-Party Payment Service.** `Third-Party Payment Service` is an external service used to make payments communicating with banks or payment sites.
- **Charging Point System.** `Charging Point System` is an external service that represent the software running on charging points. It is used in the CPOs interactions to manage their charging points.
- **DSO Communication Module.** `DSO Communication Module` is an external service that gives the system to communicate with the DSOs in order to get their prices and to enable electricity providing.

2.3. Deployment View

The following deployment diagram shows how all the components are distributed into different nodes, highlighting how they communicate with each other.
The next paragraphs will go into details about the reasons behind the design choices.
The deployment diagram is:

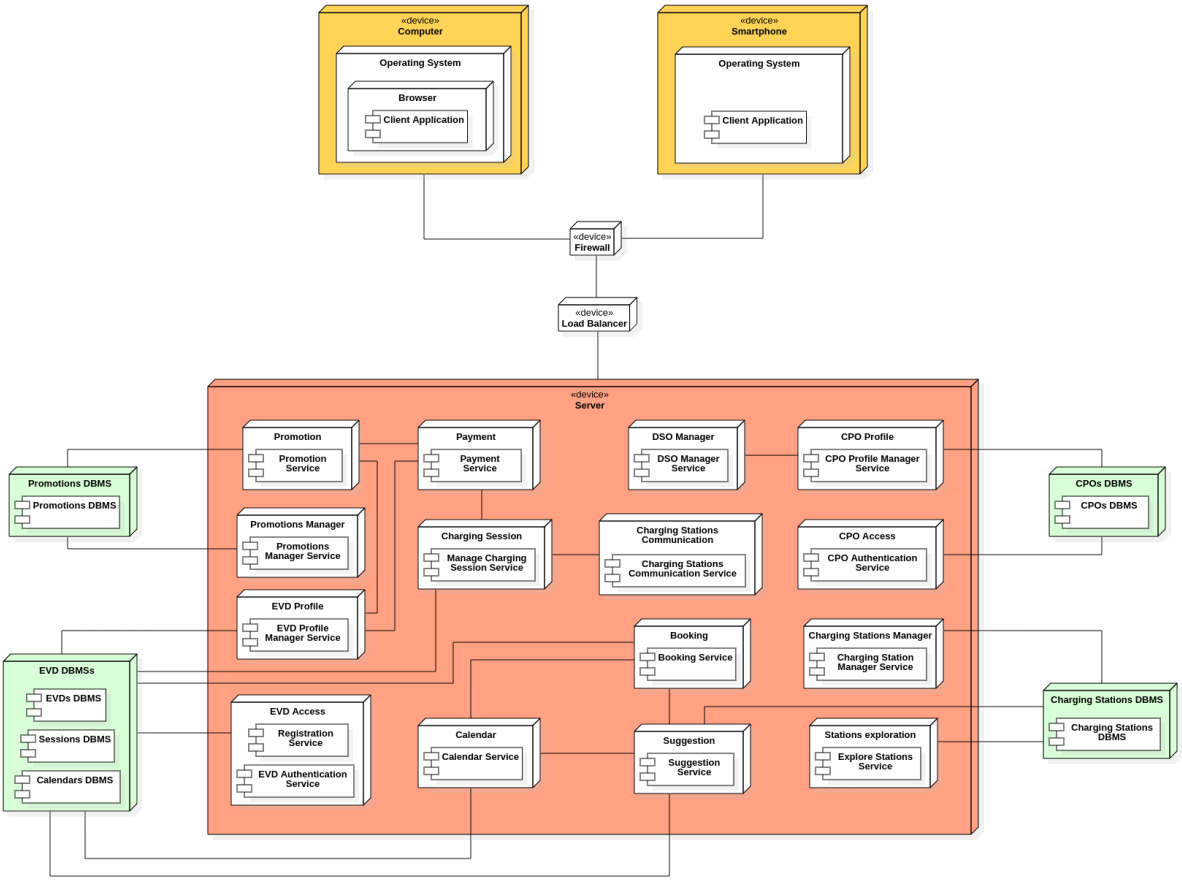

Figure 2.2: Deployment diagram of the eMALL system.

2.3.1. Connection to the server

EVDs and CPOs can access the eMALL system from both PC and smartphones. In the first case, it is necessary to use a browser to load the system's web page. In the second case, the client will use the application after downloading it from the smartphone's store (Android or iOS). When requests are sent to the server, first they pass into the firewall, so to avoid eventual cyberattacks on the system, then they pass into the load balancer, so to optimize resource usage, improve performance, and increase the availability of several services. Requests are now ready to be handled by the eMALL services.
It follows the corresponding part from the deployment diagram:

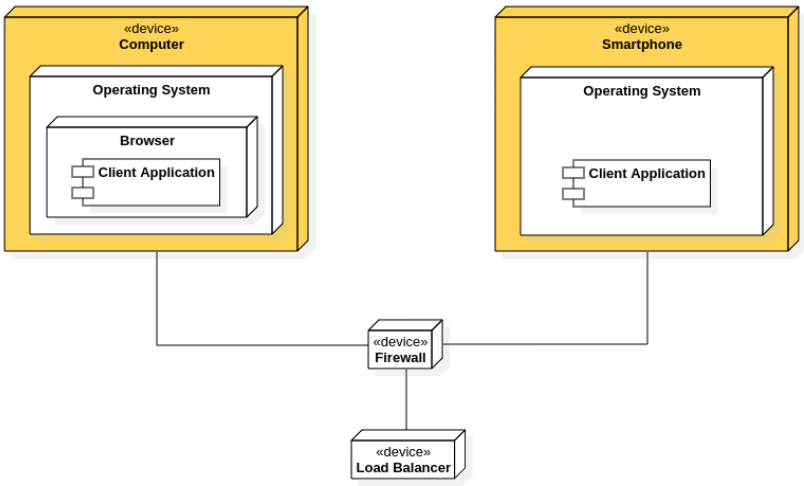

Figure 2.3: Connection to the server diagram.

2.3.2. Promotions

Promotion DBMS is one of the four identified DBMS nodes. The choice of dividing it from other DBMSs relies on the will to better scale the system. In this way, it is easier to guarantee the availability of other services that don't work with promotions, and maintenance sessions are facilitated too. It has not been grouped with other DBMSs into the same node because Promotions DBMS is also used by CPOs, so the system needs to guarantee a high level of scalability to assure business functionalities to the companies. It follows the corresponding part from the deployment diagram:

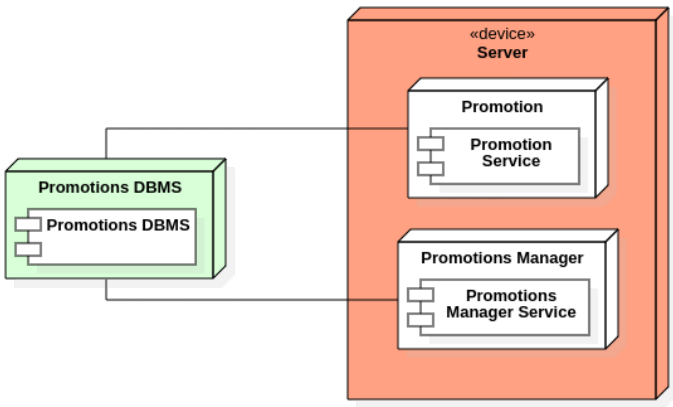

Figure 2.4: Promotions managing diagram.

2.3.3. EVD interactions

This section shows how the system communicates with `EVDs DBMS`. We choose to group `EVDs DBMS`, `Sessions DBMS`, and `Calendars DBMS` into the same node because they all receive requests from the services only in case of interactions with EVDs. Considering that they don't introduce strict time response requirements, it was not necessary to insert new physical nodes for the managing of the DBMSs.
It follows the corresponding part from the deployment diagram:

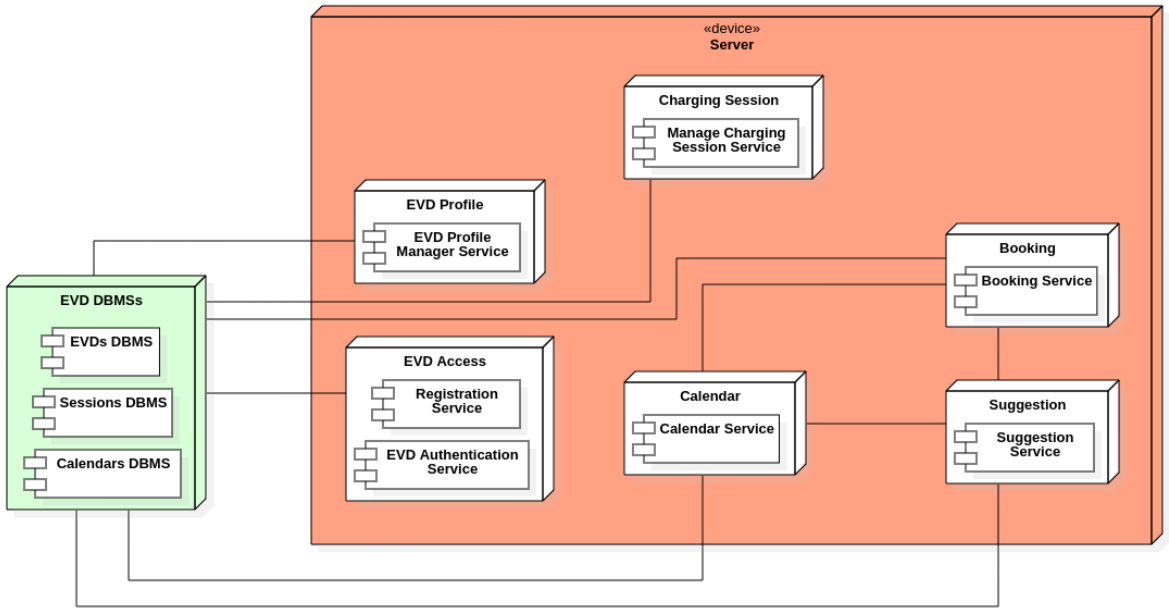

Figure 2.5: EVD interactions diagram.

2.3.4. CPOs DBMS

The components that communicate with the CPOs DBMS are the Authentication Service and the CPO Profile Manager Service. When another service needs to get or to update information about a CPO, the request is handled by the CPO Profile node. The DBMS is used by CPOs, so it is deployed in a single node to better scale the system, and to guarantee business functionalities to the companies.
It follows the corresponding part from the deployment diagram:

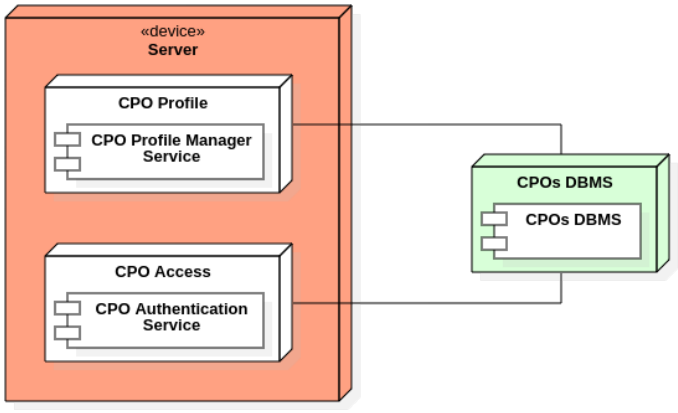

Figure 2.6: CPOs DBMS managing diagram.

2.3.5. Charging stations communication

Suggestion and Stations exploration nodes read from the DBMS to get the position of the stations that will be elaborated or shown to the user. Charging Station Manager Service can also write or update the instances of the DBMS. The DBMS is used by CPOs, so it is deployed in a single node to better scale the system, and to guarantee business functionalities to the companies.
It follows the corresponding part from the deployment diagram:

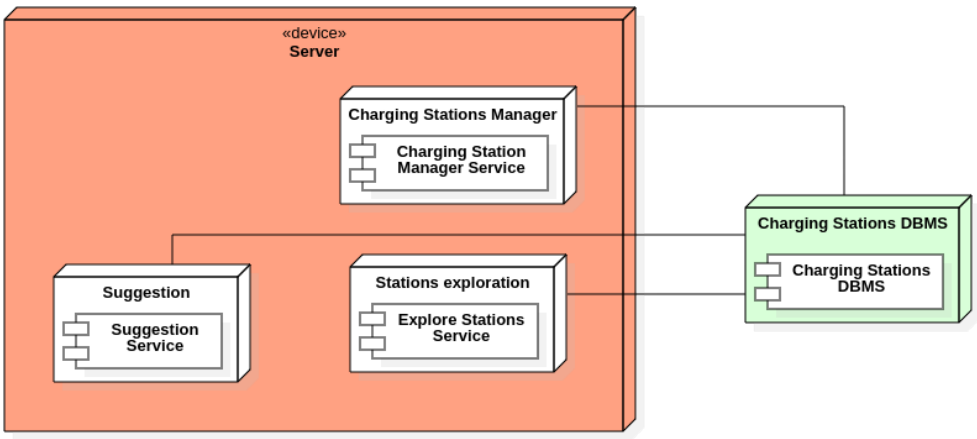

Figure 2.7: Charging stations communication diagram.

2.3.6. Services

The section shows all the identified nodes in which services run.
Decisions have been made giving particular attention to the concepts of loose coupling and

high cohesion. As explained in Sam Newman's *Building Microservices*, they are defined as follows:

- **Loose coupling.** *When services are loosely coupled, a change to one service should not require a change to another. The whole point of a microservice is being able to make a change to one service and deploy it, without needing to change any other part of the system.*
- **High Cohesion.** *We want related behavior to sit together, and unrelated behavior to sit elsewhere. Making changes in lots of different places is slower, and deploying lots of services at once is risky, both of which we want to avoid.*

The following image wants also to highlight the relations between different services, which are direct consequences of the communication interfaces previously shown in the component diagram.

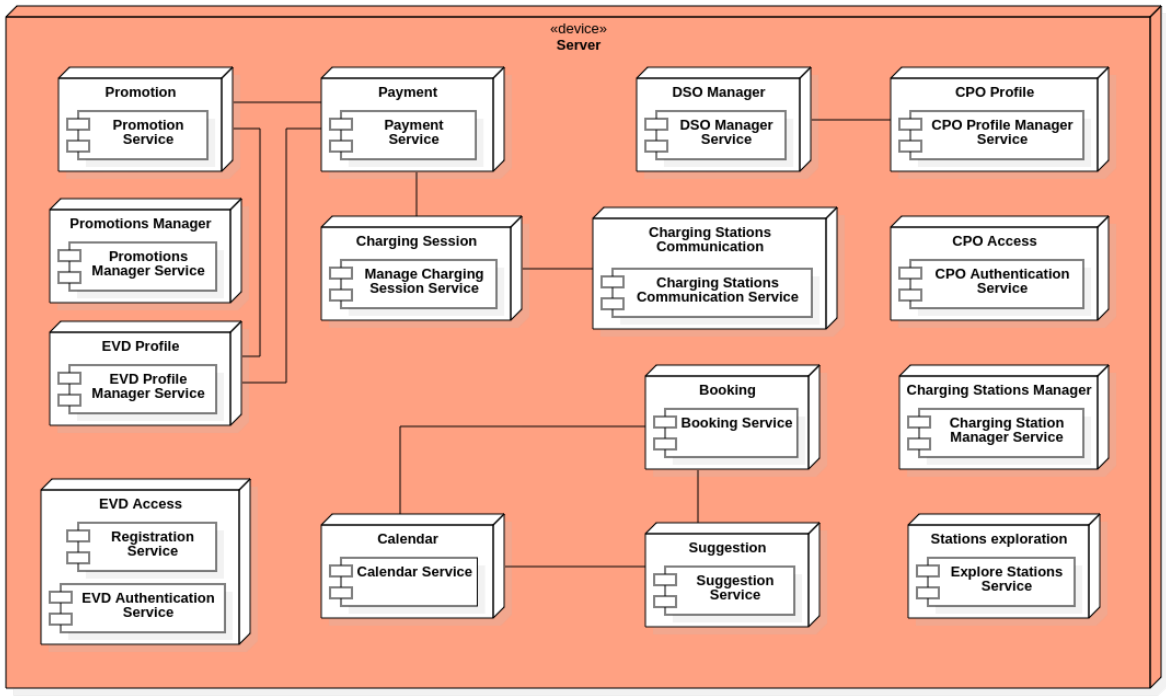

Figure 2.8: Server services diagram.

2.4. Runtime View

Here we present the dynamic of our system through sequence diagrams. We have found the components that communicate with each other to form our system, so now we explain their behaviors.

First, we will present eMALL actions from the point of view of the EVD user, i.e., logging in, booking a charge, managing his profile, etc. Then, we will present eMALL actions from the point of view of the CPO user, actions that are much more business functionality. In this section, we hadn't presented all the RASD document use cases because we have decided to focus on the critical part of the system functionalities.

Registered EVD Logs In When the EVD - `Client Application` - wants to log in to eMALL, he calls the "`logIn`" function from the `EVD Authentication Service` component. That component requests the client to display the login form by running the "`displayLogInForm`" function, and then it waits for the email and password from the `Client Application`.

When the EVD sends this information, the Authentication Service asks the `EVDs DBMS` to validate it. Finished processing the login, the last component returns an ACK message to the `EVD Authentication Service`, which sends a confirmation or an error to the client.

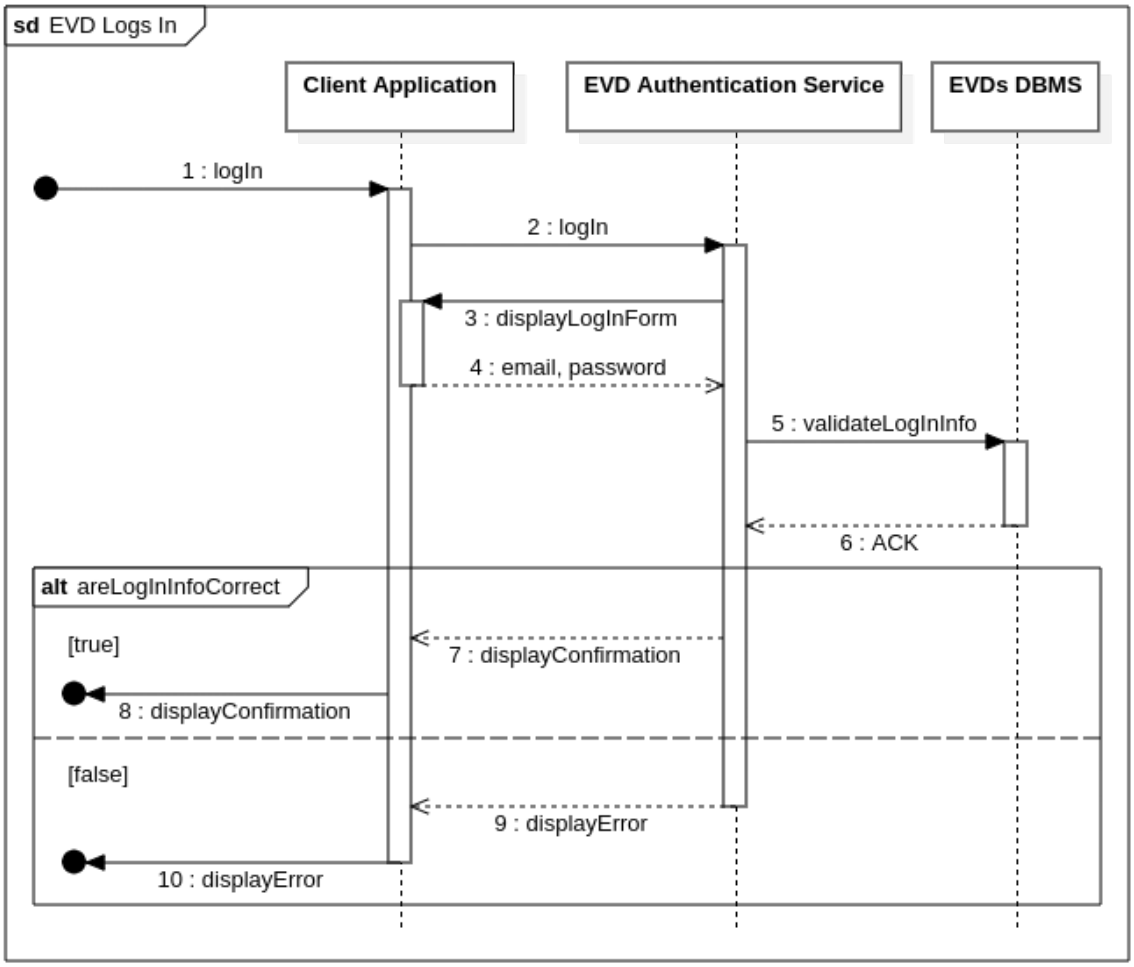

Figure 2.9: Registered EVD logs in sequence diagram

Registered EVD adds an EV When the EVD is on his profile page and wants to add a new vehicle to his parking lot, he runs the "`addVehicle`" function through the `EVD Profile Manager Service` component.

At first, it asks back to the client to insert the EV information to save in his profile, and then it waits for him. After the user has sent the information, the component also asks for the nickname to give to the EV. Sending the nickname, the Profile Manager saves the new information by passing them to the `EVDs DBMS`. When the DBMS has sent back the ACK message, the `EVD Profile Manager Service` component returns the outcome of the process to the EVD.

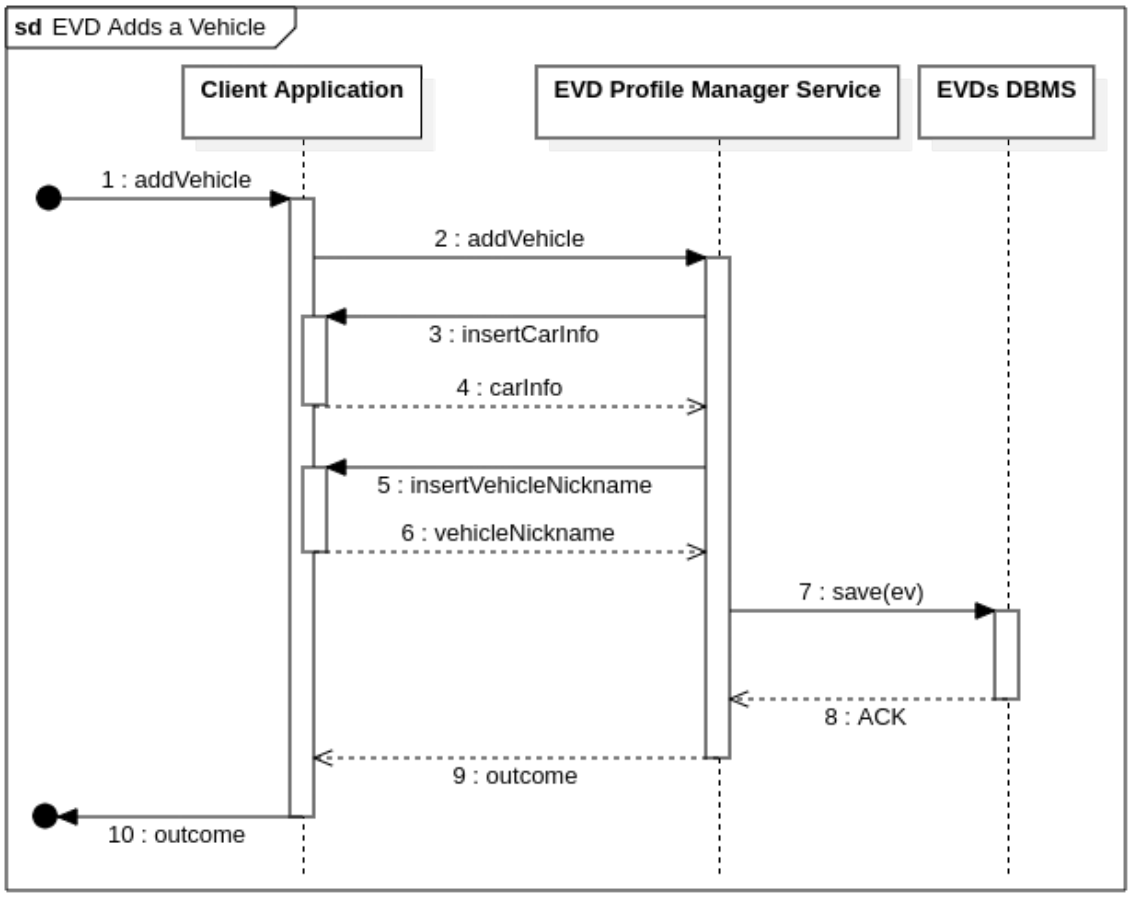

Figure 2.10: Registered EVD adds an EV sequence diagram

Registered EVD books a charge In eMALL, the EVD can also book a charge in a charging station. In the sequence diagram, we suppose the `Client Application` knows the charging stations he wants to book. Then, the user calls the "`bookCharge`" function of the `Booking Service` component, passing the given charging station.

The component needs to know the available timeframes for the charging station, so it waits for the `Sessions DBMS` component to process the information. When it receives the list of timeframes, the `Booking Service` asks the user to choose one among them. Knowing that more than one EVD uses the system, then one timeframe that is at first available might be unavailable when eMALL proceeds with the booking. To avoid it, the `Booking Service` asks for the `Sessions DBMS` to check the availability of the chosen timeframe and to lock it if so by calling the "`isTimeframeAvailable`" function.

Once the chosen timeframe is locked, the component asks the EVD to confirm the booking - "EVD Confirms the Booking" sequence diagram. In the end, the system sends the

outcome of the process to the `EVD Client Application`.

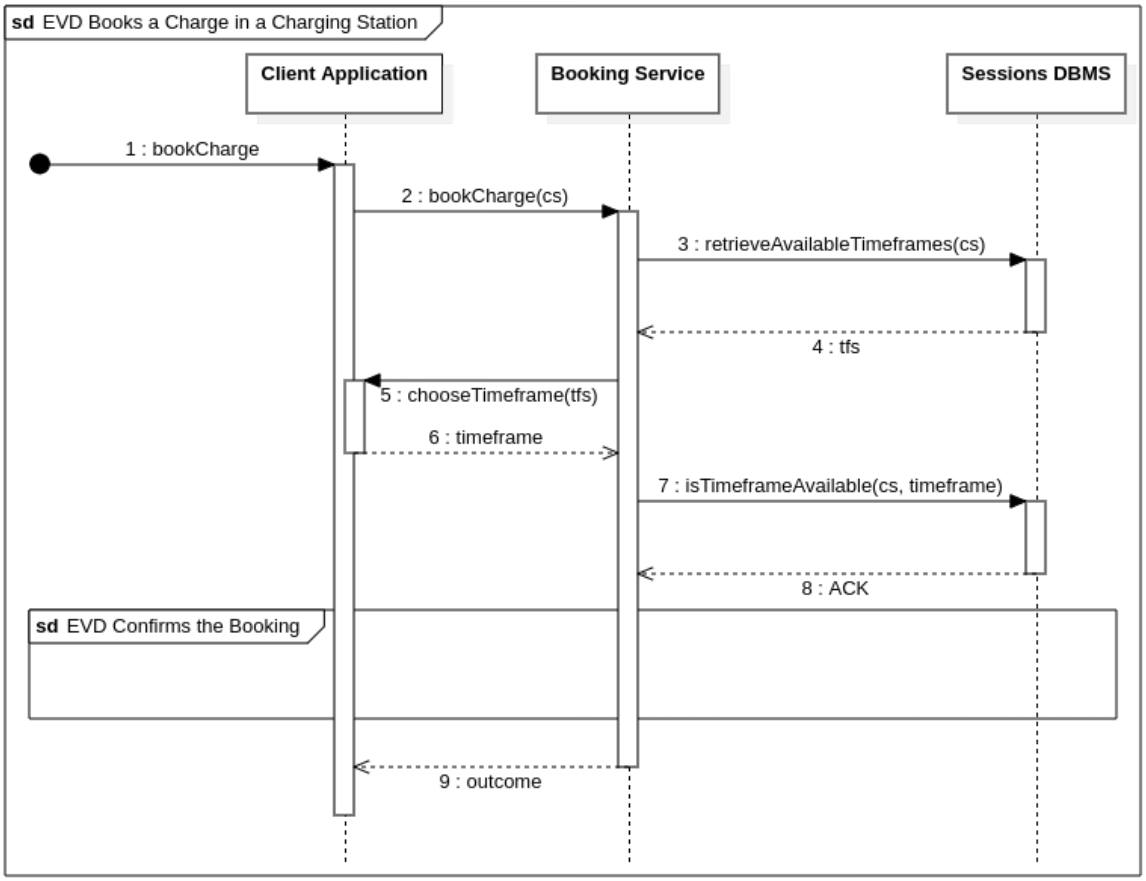

Figure 2.11: Registered EVD books a charge sequence diagram

Registered EVD confirms the booking Our system always asks the client to confirm a booking before saving it, so the `Booking Service` component receives the client, the charging station where he wants to book, and the timeframe to reserve for the client. At first, it asks the EVD to confirm the booking and waits for his reply. Given the confirmation, the `Booking Service` runs the "`addSession`" function of `Sessions DBMS` to add the new booking to the database and orders to the `Calendar Service` to add it to the calendar. After the `Calendar Service` has run the "`saveActivity`" function of `Calendars DBMS`, `Booking Service` returns the outcome to the caller.

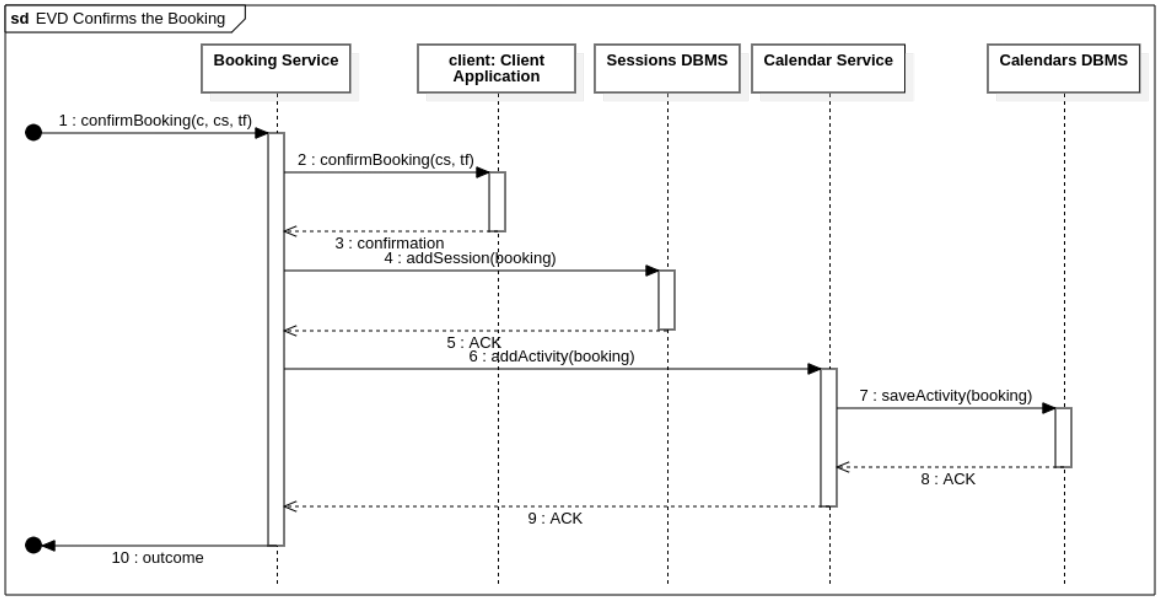

Figure 2.12: Registered EVD confirms the booking sequence diagram

Registered EVD consults a specific promotion that can be redeemed An EVD might want to redeem an available promotion in eMALL through his page. He has to call the "`redeemPromotion`" function from the `Promotion Service` component, which retrieves all the promotions by running the get function of the list of promotions from the `Promotions DBMS`. When the `Promotion Service` has received the list, it asks the user to select which promotion he wants to redeem and then asks for a confirmation for its activation.

If the user has decided to proceed with the activation, `Promotion Service` initializes the payment by calling the "`pay`" function from the `Payment Service` component - "EVD Makes a Payment" sequence diagram. When the payment successfully ends, the activation process runs through the `EVD Profile Manager Service` component by calling the "`activate`" function that receives the paid promotion and the client in input. The Profile Manager calls the counterpart "`activate`" function from the `EVDs DBMS`.

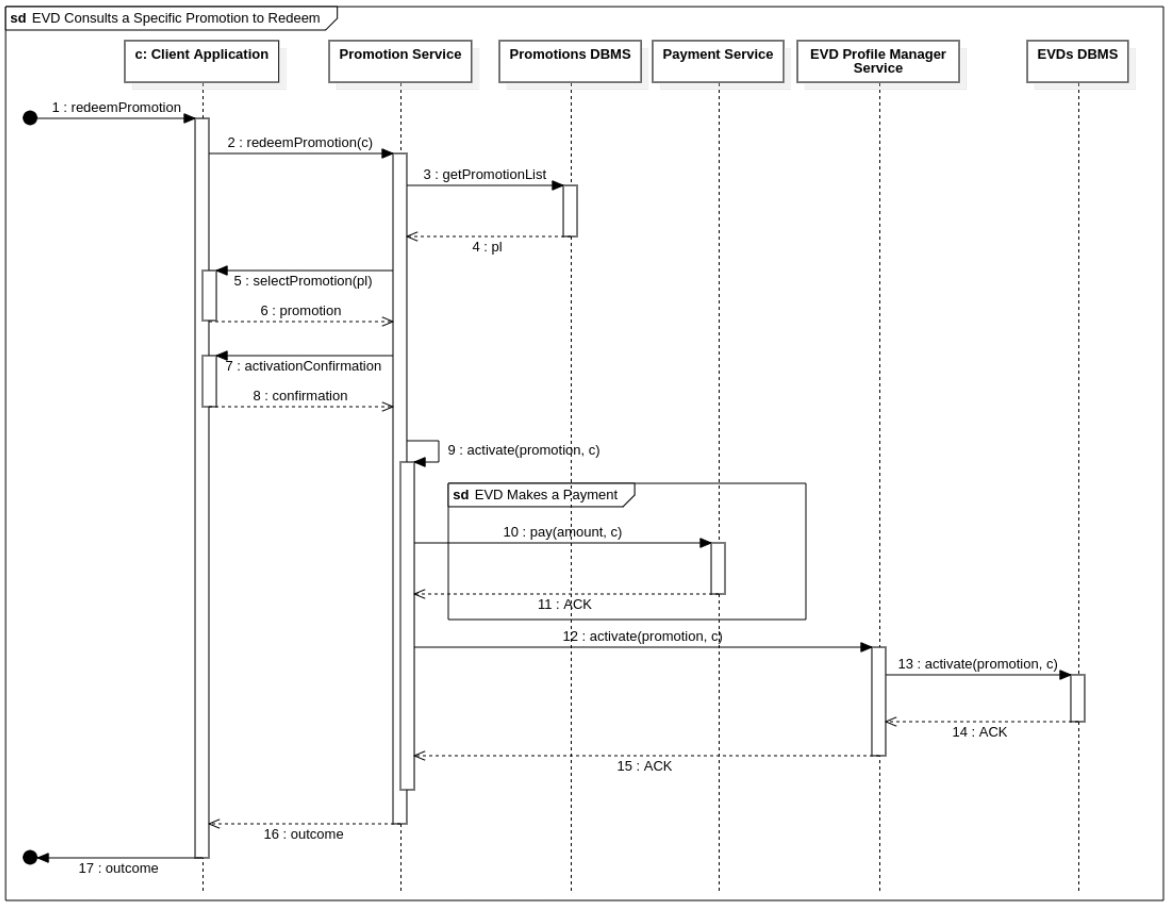

Figure 2.13: Registered EVD consults a specific promotion that can be redeemed sequence diagram

Registered EVD charges his EV Now we explain how a charging process evolves in eMALL. At first, we need the EVD to call the function "`startChargeEV`" from the `Manage Charging Session Service` component to verify that the current user has booked a charging session. The service calls so the "`validateClient`" from the `Sessions DBMS` because it is the only one that can retrieve this information.

The timeframe is not necessary because the system checks at the current time. If the EVD has booked, the service requests to unlock the charging point to the `Charging Station Communication Service`, which delegates the process to the `Charging Point System`. Once the charging point is unlocked, the `Manage Charging Session Service` asks the user to connect his EV and confirm the start of the charging process.

When the EVD confirms, the service asks the `Charging Session Communication Service` to start charging, which delegates to the `Charging Point System`. Then, we have a

loop sequence where the user asks indirectly for information from the `Charging Point System` to see the status of his EV's battery. When the EVD decides to stop the charging process, he communicates it to the `Manage Charging Session Service`, which delegates the decision to the `Charging Session Communication Service`, which orders to stop charging to the `Charging Point System`. Finally, the `Manage Charging Session Service` initializes the payment - EVD Makes a Payment sequence diagram - and calls the "`save`" function from the `Sessions DBMS` to save the EVD session.

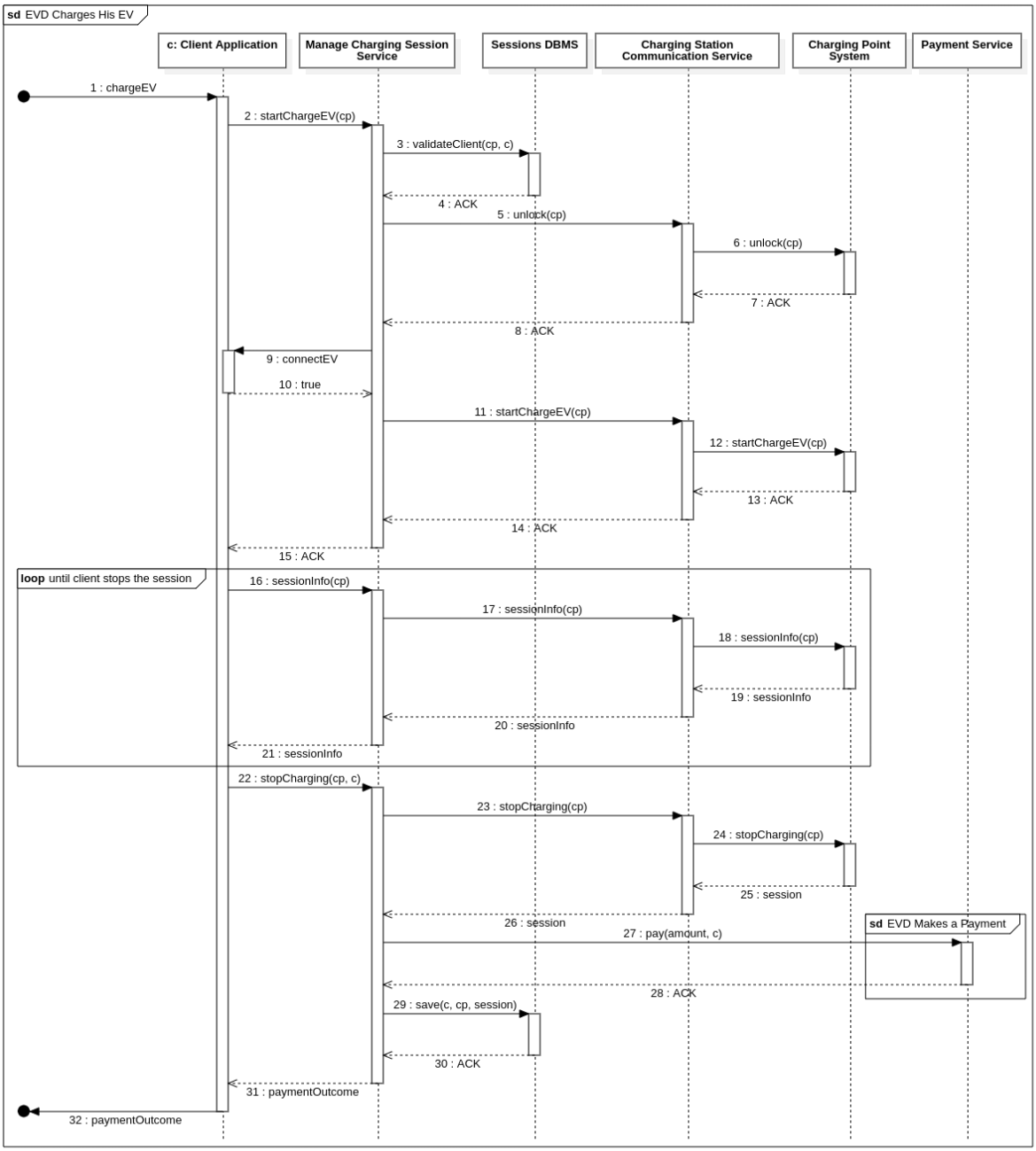

Figure 2.14: Registered EVD charges his EV sequence diagram

Registered EVD makes a payment When an EVD has to pay, the `Payment Service` receives a call with the amount to pay from a client and the client himself in input. The service asks the `EVD Profile Manager Service` to select a payment method for the given EVD. The Profile Manager asks the `EVDs DBMS` for the user's payment methods by calling the "`getPaymentMethods`" function and returns the obtained list to the caller. Finally, the `Payment Service` runs the "`pay`" function from the `Third-party Payment Service` with the amount to pay in input.

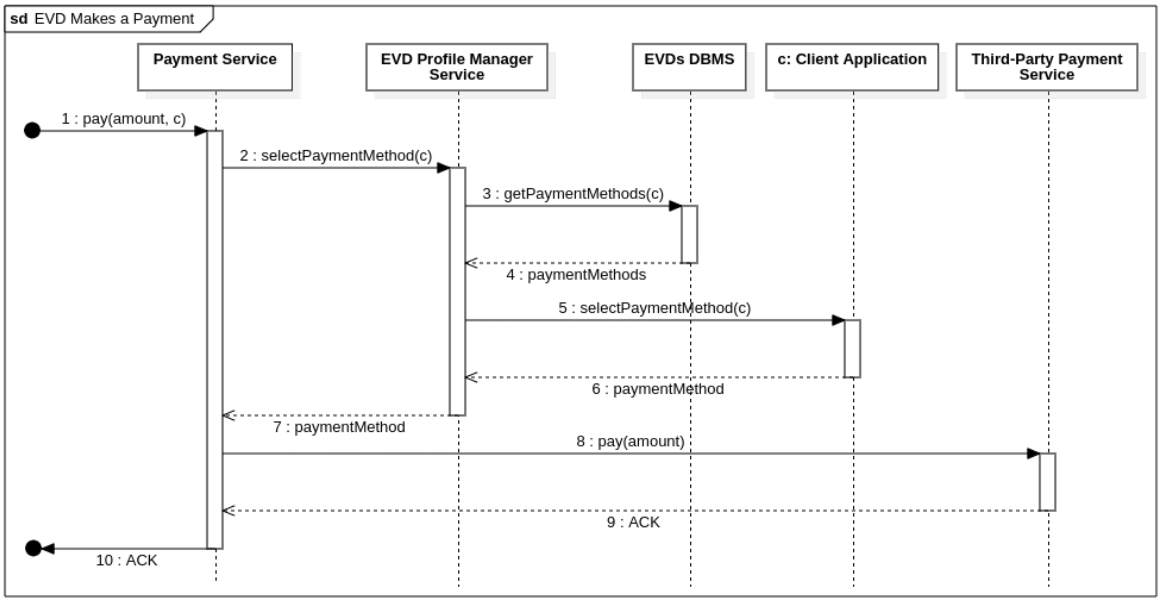

Figure 2.15: Registered EVD makes a payment sequence diagram

Registered EVD adds a new activity into the calendar and receives suggestions about charging schedule When the EVD wants to add a new activity to his calendar, he first opens the calendar section by calling the "`openCalendar`" function from the `Calendar Service` component, which delegates the call to `Calendars DBMS`.

Once the user has received the calendar, he can insert the new activity by requesting a form to the `Calendar Service` through the "`insertNewActivity`" function call. When the user fills out the form, he sends it to the `Calendar Service`, which calls the "`validateActivity`" function from the `Calendars DBMS` and then the "`saveActivity`" function from the same DBMS component.

Finally, the service initializes asynchronously the function "`findBestSchedule`," which computes the best path in reaching the different appointments of the user depending on the battery status of the EV, its capacity, and energy costs. The call is asynchronous

because the service is not essential for the EVD. If the `Suggestion Service` component is down, eMALL has to work without it and allow the users to add new activities to their calendars.

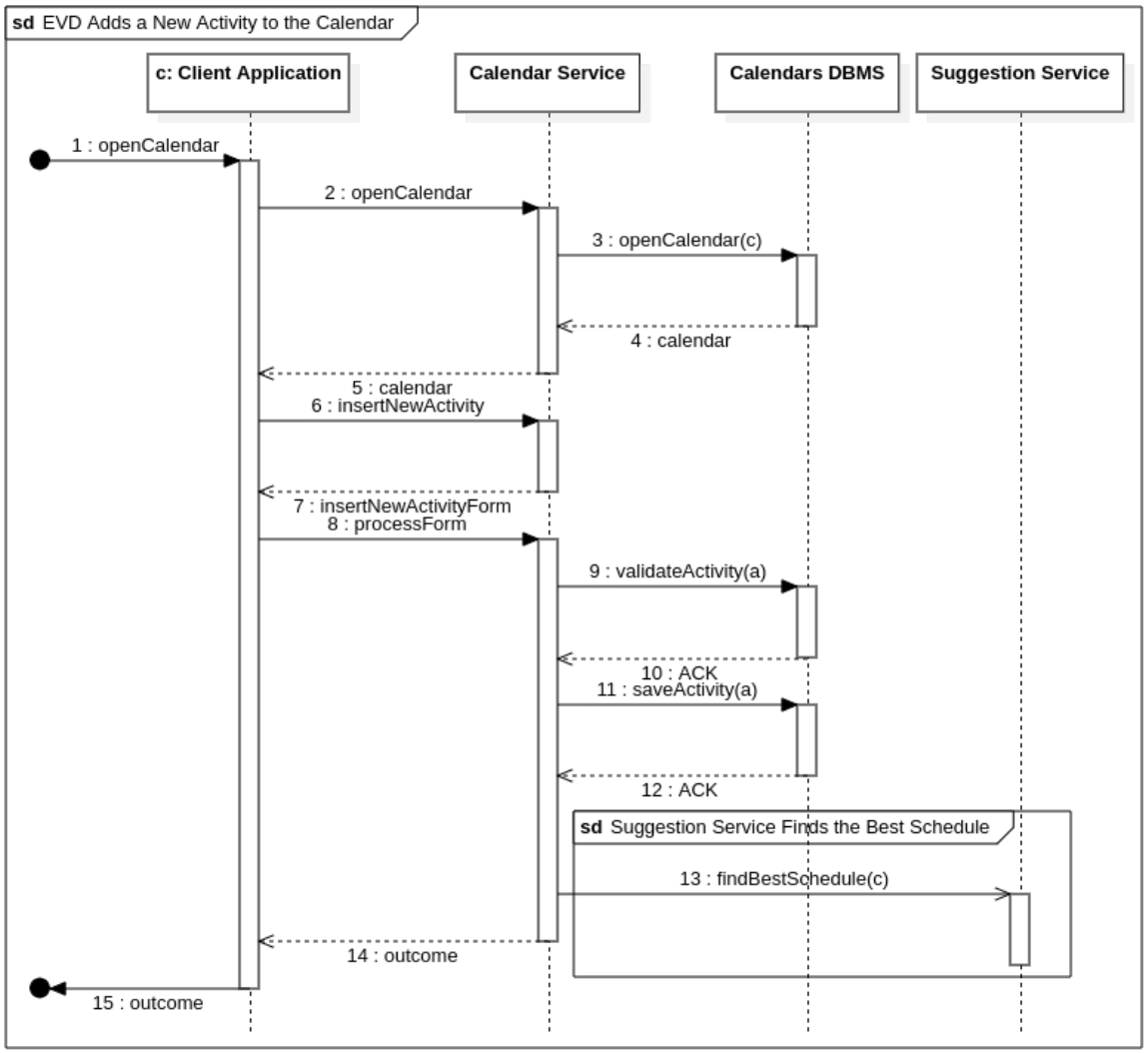

Figure 2.16: Registered EVD adds a new activity into the calendar and receives suggestions about charging schedule sequence diagram

CPO logs in In eMALL, when a CPO wants to log in, he calls the function "`logInCPO`" from the `CPO Authentication Service` component, which calls back the "`insertInfo`" function from the `Client Application` to retrieve the essential information for the CPO identification, i.e., ID, email, and password. Once the CPO has sent the information, the Authentication Service runs the "`validateAccount`" function from the `CPOs DBMS`. If the CPO operator has inserted the correct credentials, he will enter his homepage.

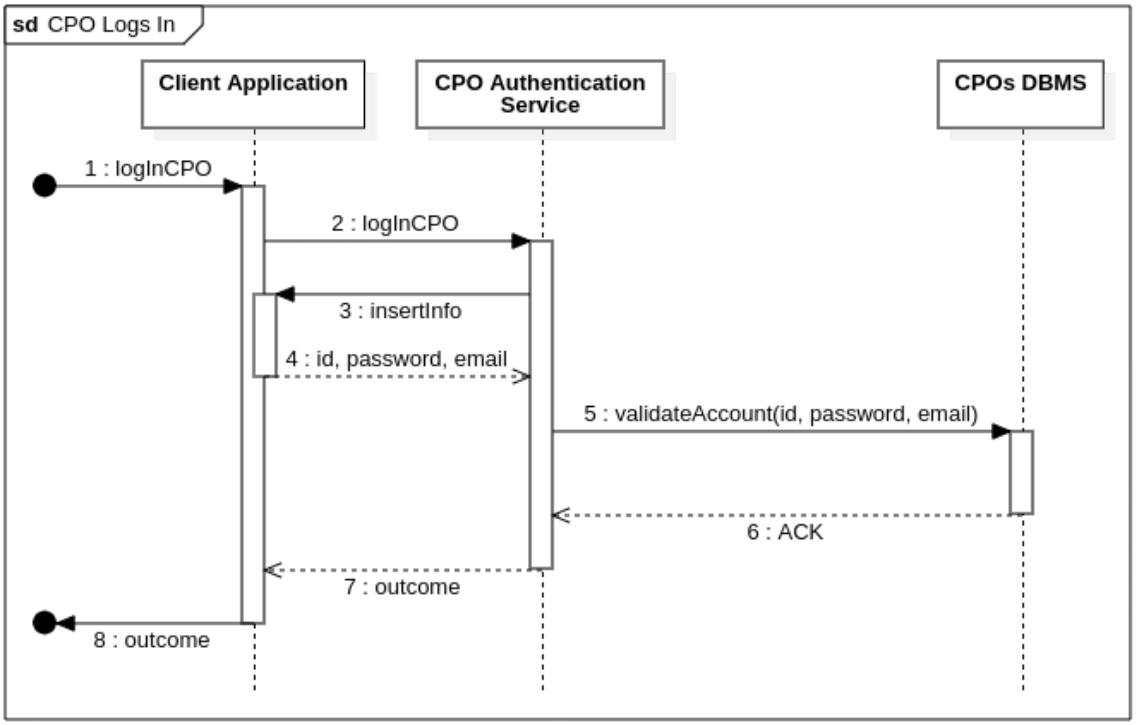

Figure 2.17: CPO logs in sequence diagram

CPO sets a fee From his homepage, a CPO might want to change the fees of his charging stations. When he has decided which charging station, he calls the "`setFee`" function from the `Charging Station Manager Service` component, which asks the user to insert the new fee to set to the given charging station. Once the CPO has specified the new fee, the Charging Station Manager runs the "`updateFee`" function from the `Charging Stations DBMS`, which replaces the old fee value with the new one in input.

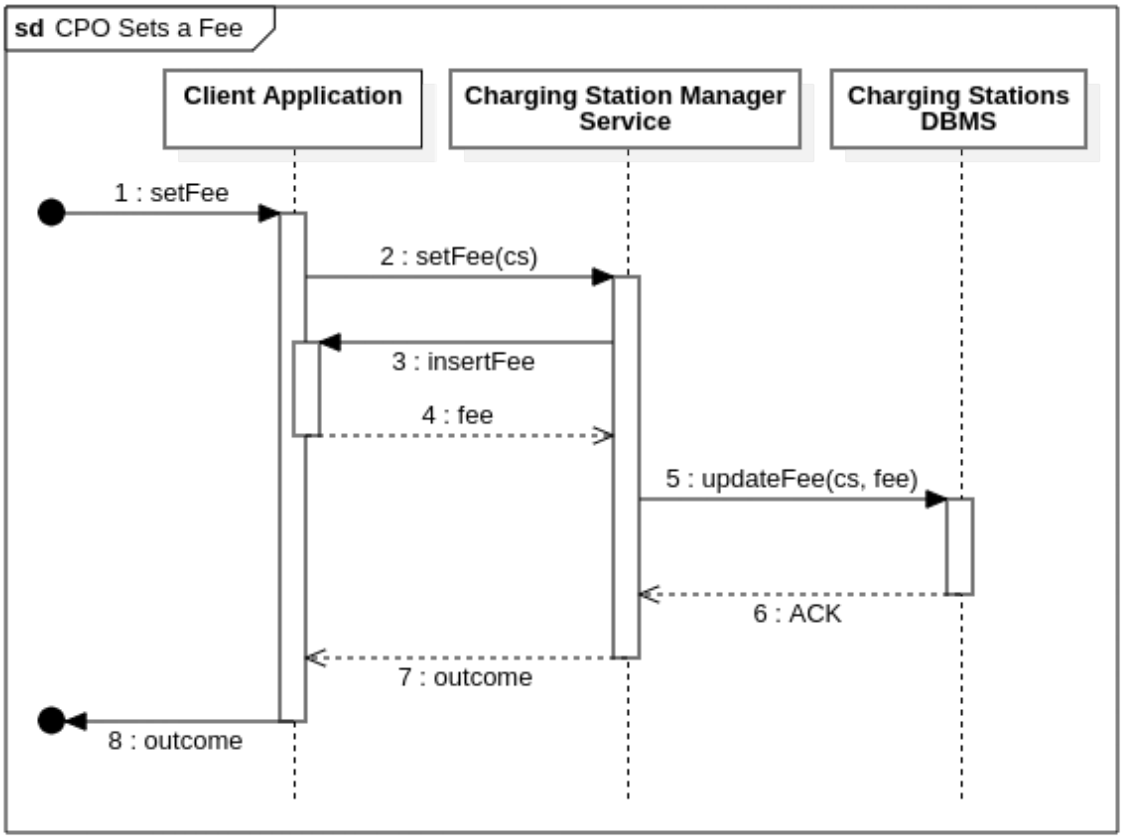

Figure 2.18: CPO sets a fee sequence diagram

CPO adds a charging station From his homepage, a CPO can add new charging stations to his profile, so he calls the "`addChargingStation`" function from the `Charging Station Manager Service` component. The service asks the `Client Application` to insert the location of the new charging station, its status, the charging costs, and the charging points with their information. For each piece of information, the service waits for the CPO to reply before proceeding with the new request. Once the service has received all the information, it passes them to the validate function called from the `Charging Stations DBMS`.

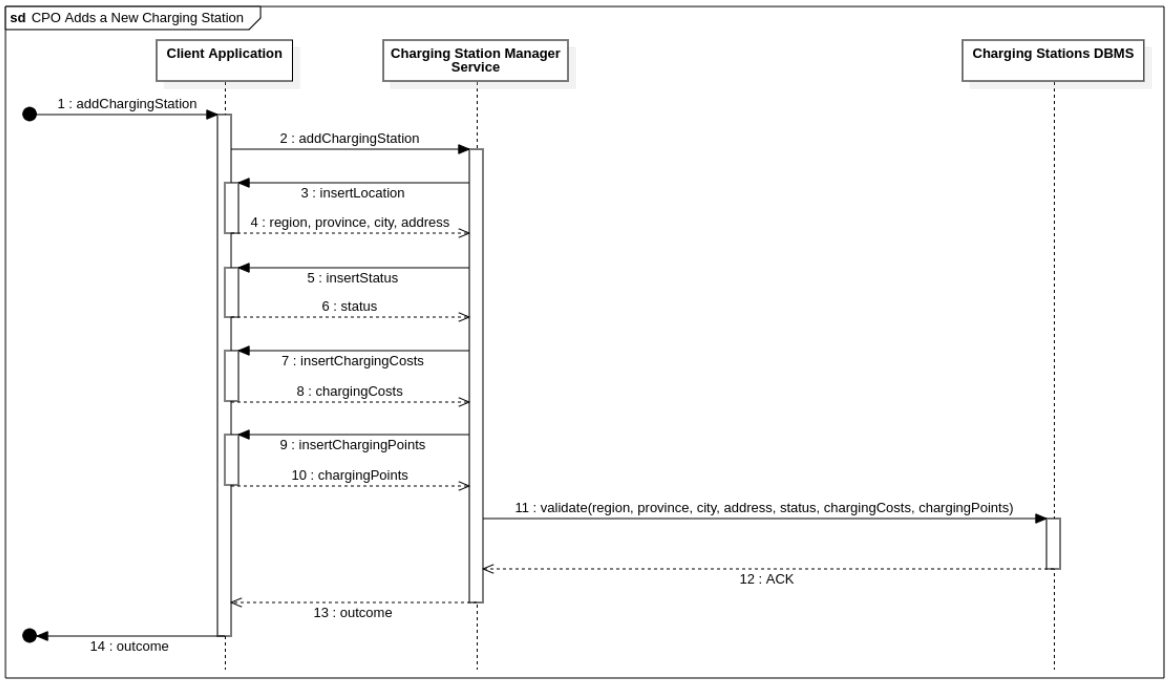

Figure 2.19: CPO adds a charging station sequence diagram

CPO adds a charging point A CPO can also add new charging points to an existing charging station. When he chooses this operation, he calls the "`addChargingPoint`" function from the `Charging Station Manager Service` component, which asks the `Client Application` to enter the charging station where the new charging point has to be added and then the information of the charging point itself. Once the service has received all the information, it passes them to the validate function called from the `Charging Stations DBMS`.

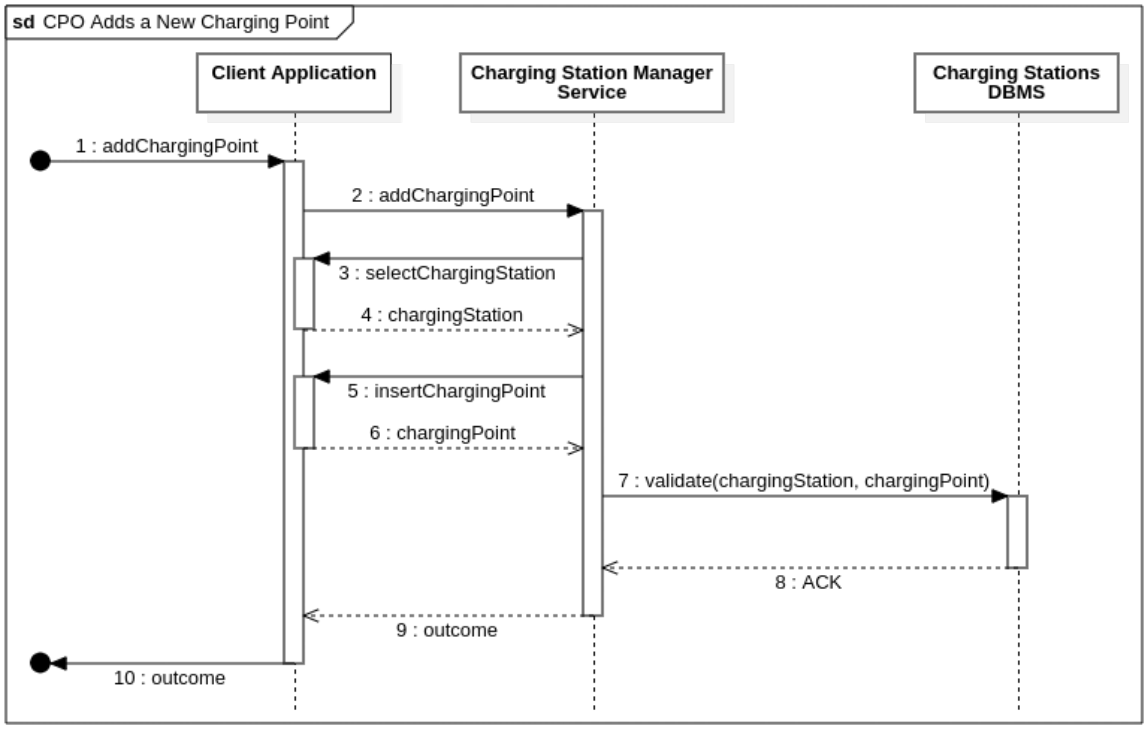

Figure 2.20: CPO adds a charging point sequence diagram

CPO changes the metadata of a charging point When a CPO wants to edit the metadata of charging points, he calls the "`updateMetadataChargingPoint`" from the `Charging Station Manager Service` component with the given charging station in input. At first, the service asks the `Client Application` to select the charging point to edit, so it calls the "`editMetadataChargingPoint`" function from the `Client Application` that allows the CPO to edit the charging point information. Once the user has filled out the form, the function returns the new charging point to the Charging Station Manager, which calls the update function with the charging station, the old charging point, and the up-to-date charging point from the `Charging Stations DBMS`.

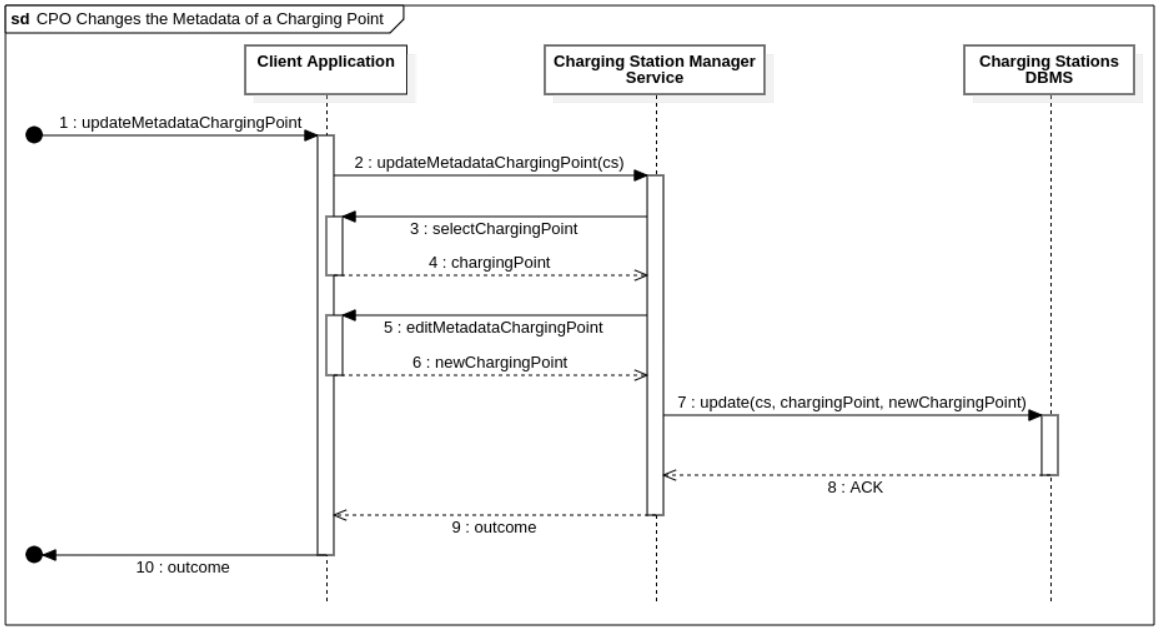

Figure 2.21: CPO changes the metadata of a charging point sequence diagram

CPO activates a promotion The CPO can define promotions for the EVD. At first, the CPO calls the "`activatePromotion`" function from the `Promotion Manager Service` component, which calls back the "`definePromotionFeatures`" from the `Client Application`. The function, as the name suggests, allows the CPO to define the features of the new promotion. The form is sent to the caller when filled out. The Promotion Manager processes the result and, if it's correct, calls the function "`save`" with the promotion in input from the `Promotions DBMS`. If the store is successful, the service calls the "`initialize`" function to make the promotion available to EVDs.

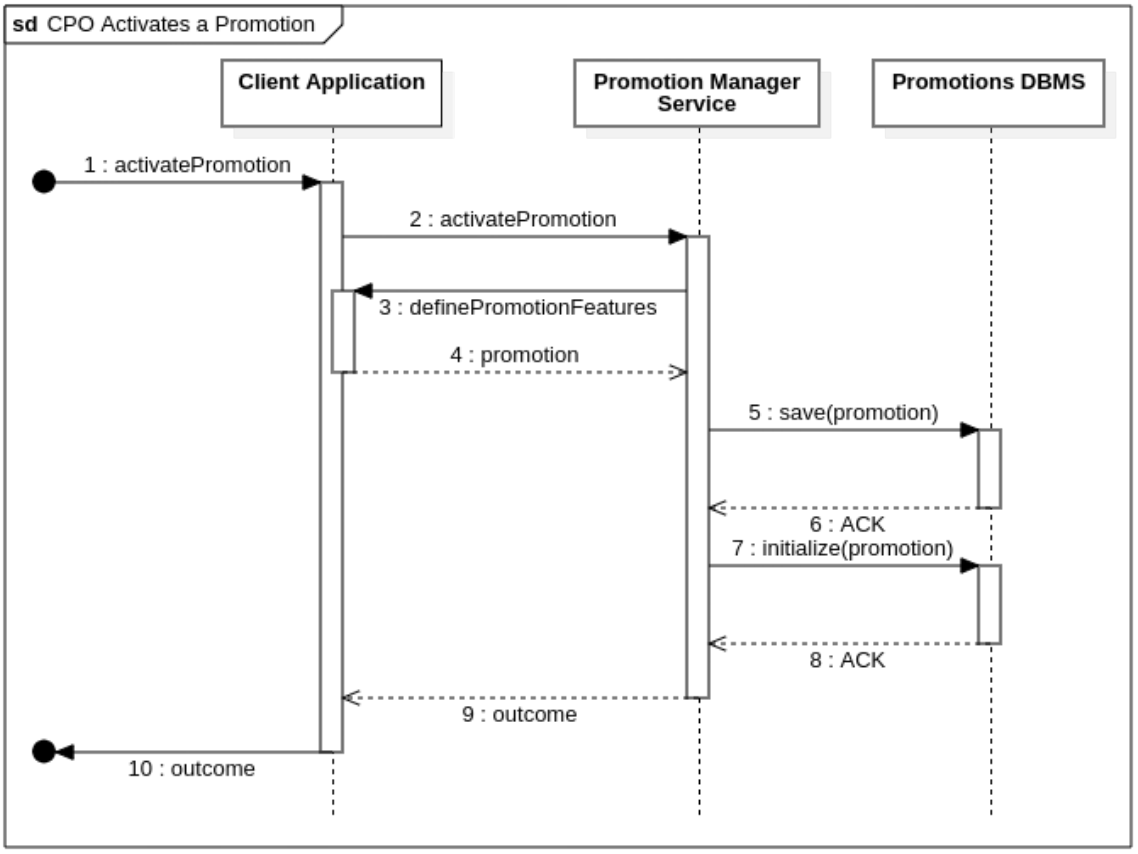

Figure 2.22: CPO activates a promotion sequence diagram

CPO plans a maintenance session for a charging station A CPO can plan maintenance sessions for his charging station, so here we describe the sequence diagram of the planning process. At first, the CPO calls the "`planMaintenanceSession`" function from the `Charging Station Manager Service` component with the charging station in input. The service asks the user to fill out the form for the date and hour of the planned maintenance. Once the CPO returns the form, the Charging Station Manager calls the "`planMaintenance`" function from the `Charging Station Communication Service`, which delegates the job to the `Charging Point System`. The function notifies the `Charging Point System` that the charging points of a given charging station will be unavailable on a particular date. When the Charging Station Manager receives the success notification of the alert, it saves the information into the `Charging Stations DBMS` by calling the "`savePlannedMaintenance`" function with the charging station, date, and hour in input.

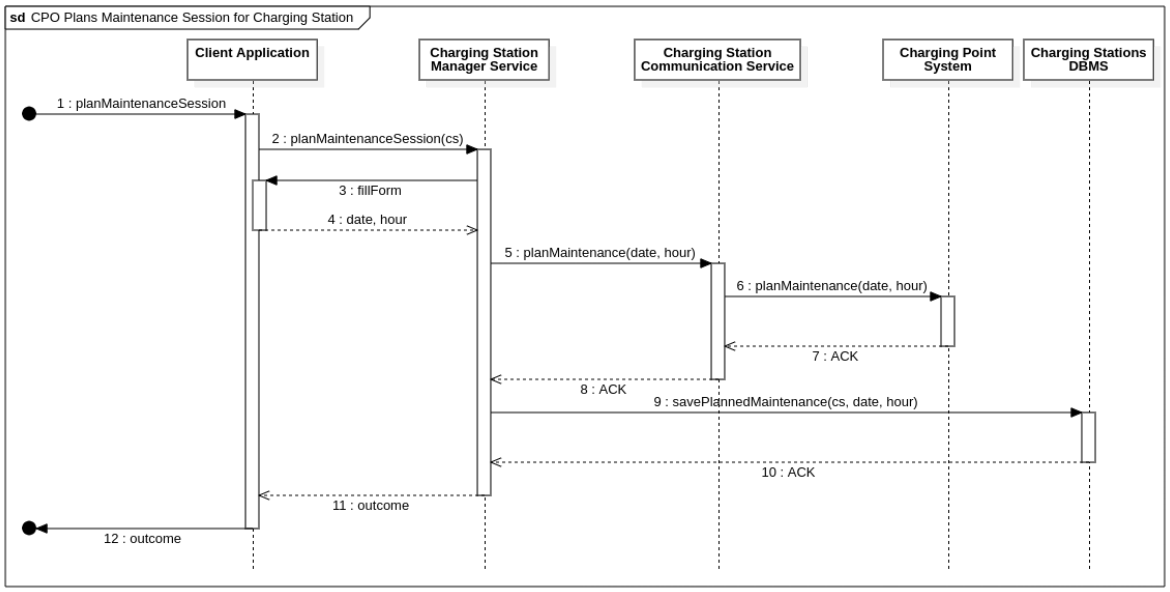

Figure 2.23: CPO plans a maintenance session for a charging station sequence diagram

CPO decides the DSO from which acquire energy A CPO can edit his energy provider - DSO - from his homepage. At first, he calls the "`setDSOAcquireEnergy`" function from the `DSO Manager Service` with the CPO's identifier in input. The DSO Manager orders the `DSO Communication Module` to return the list of all the DSOs available in eMALL. When the list is ready, the DSO Manager asks the client to select the DSO from the list, and then it runs the "`activateDSO`" function of the `DSO Communication Module`, which alerts the DSO that a new CPO will acquire energy from it. After the alert, the service calls the "`update`" function from the `CPO Profile Manager Service`, which delegates to the counterpart "`update`" function of the `CPOs DBMS` to store the new relationship. The process ends by notifying the `Client Application` about the successful ending of the operation.

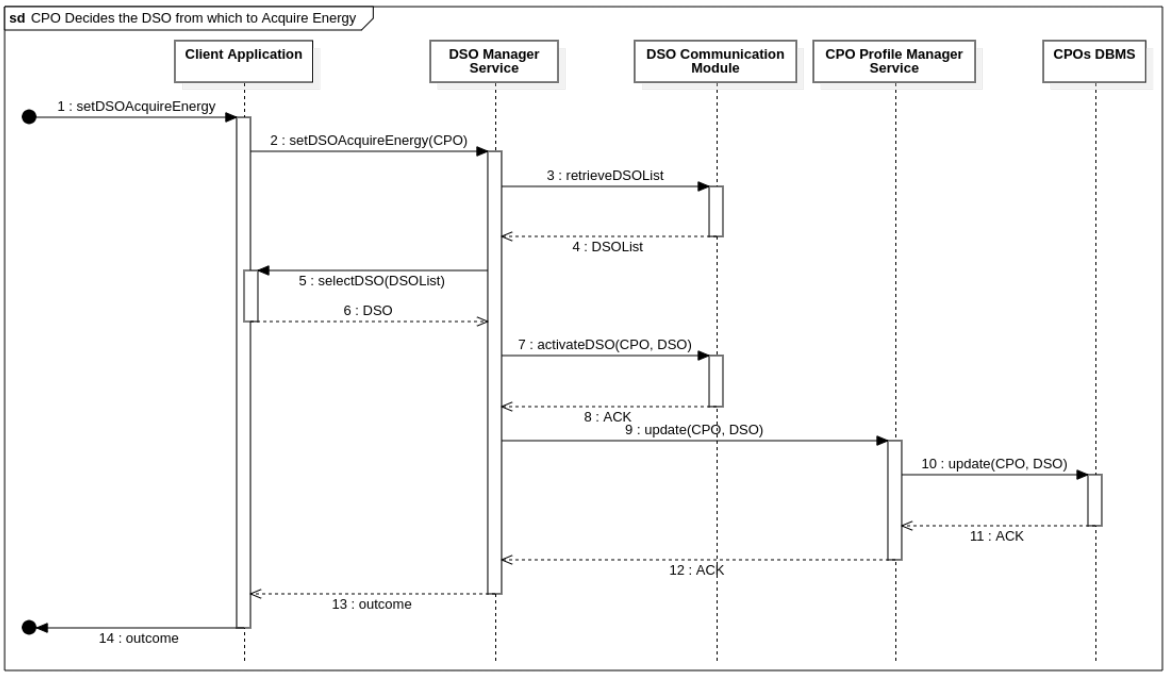

Figure 2.24: CPO decides the DSO from which acquire energy sequence diagram

2.5. Component Interfaces

2.6. Selected Architectural Styles and Patterns

The eMALL system offers functionalities to both EVDs and CPOs. The second ones represent companies that use the system to manage their business goals. For this reason, we choose to adopt a microservices architecture.
The following list describes the key benefits of the choice:

- **Technology heterogeneity.** With a system composed of multiple, collaborating services, we can decide to use different technologies inside each one. This allows to pick the right tool for each job. As shown in the previous sections, it is shown a wide variety of functionalities. So, it is good for the system to use different programming languages and tools depending on the characteristics of the module in question.
- **Scalability.** Microservices are designed to be independently deployable and scalable.
- **Availability.** If a service fails, the rest of system keeps running. For this reason, is easier to maintain modules that present problems still guaranteeing all the other functionalities of the system.

- **Ease of deployment.** With microservices, it is possible to make a change to a single service and deploy it independently of the rest of the system. If problems occur, they can be identified, isolated, and corrected quicker, without the need of halting the whole system.

2.7. Other Design Decisions

2.7.1. Client-Server architecture

The `eMALL` system adopts a 3-tier client-server architecture. So, the system is divided into three main components: the client, the server, and the databases. The client represents the front-end user interface, and it is the tool used by users to communicate with the system. The server is the back-end platform, which receives users' requests, elaborates answers, and stores data. This design decision has different key benefits: the server can handle several requests simultaneously. Additionally, it is easier to guarantee the security and integrity of the stored data, given that the databases are separated from the business logic and users. Finally, it facilitates maintenance and updates of the parts of the system, given that client and server can be handled independently.

2.7.2. Thin client

The mobile app and the web page that will be used by users act as a client, sending requests to the server and receiving responses, while the server handles the heavy lifting of processing and storing data. Given the microservices architecture we adopted, the client will communicate to defined offered services according to its needs. So, the client will be lightweight and easy to use since it doesn't need to store and process large amounts of data. Another benefit the system enjoys from the thin client design decision is an increase in the system's security since all the information is processed and stored on the server rather than on the client device.

2.7.3. Shared databases

We decided to introduce databases that will be shared by several services, as already shown in the deployment view section. The choice relies on several services working on the same kind of data. The possibility of having only one database shared entirely by services could have introduced strong dependencies between modules. In the same way, introducing one database for each service and replicating information where needed would have been more expensive. Considering the system's requirements, sharing databases between modules is,

instead, a good point of balance.

3 | User Interface Design

3.1. General Overview

3.2. EVD Interface

3.3. CPO Interface

4 | Requirements Traceability

4.1. Functional Requirements Traceability

4.2. Non Functional Requirements Traceability

5 | Implementation, Integration and Test Plan

5.1. Implementation

5.2. Integration

5.3. Test Plan

6 | Effort Spent

Member of group	Effort spent	
Cela Irfan	Introduction	$0h$
	Architectural Design	$20h$
	User Interface Design	$0h$
	Requirements Traceability	$0h$
	Implementation, Integration and Test Plan	$0h$
	Reasoning	$2h$
Cela Mario	Introduction	h
	Architectural Design	h
	User Interface Design	h
	Requirements Traceability	h
	Implementation, Integration and Test Plan	h
	Reasoning	h
Cogollo Alessandro	Introduction	h
	Architectural Design	h
	User Interface Design	h
	Requirements Traceability	h
	Implementation, Integration and Test Plan	h
	Reasoning	h

Table 6.1: Effort spent by each member of the group.

7 | References

7.1. Paper References

- The specification document Assignment RDD AY 2022–2023.pdf

7.2. Used Tools

- GitHub for project versioning
- StarUML for UML diagrams
- Notion for reasoning and notes
- IntelliJ as LaTeX editor

List of Figures

2.1 Component diagram of the eMALL system. 4
2.2 Deployment diagram of the eMALL system. 9
2.3 Connection to the server diagram. 10
2.4 Promotions managing diagram. 10
2.5 EVD interactions diagram. 11
2.6 CPOs DBMS managing diagram. 12
2.7 Charging stations communication diagram. 12
2.8 Server services diagram. 13
2.9 Registered EVD logs in sequence diagram 15
2.10 Registered EVD adds an EV sequence diagram 16
2.11 Registered EVD books a charge sequence diagram 17
2.12 Registered EVD confirms the booking sequence diagram 18
2.13 Registered EVD consults a specific promotion that can be redeemed sequence diagram 19
2.14 Registered EVD charges his EV sequence diagram 20
2.15 Registered EVD makes a payment sequence diagram 21
2.16 Registered EVD adds a new activity into the calendar and receives suggestions about charging schedule sequence diagram 22
2.17 CPO logs in sequence diagram 23
2.18 CPO sets a fee sequence diagram 24
2.19 CPO adds a charging station sequence diagram 25
2.20 CPO adds a charging point sequence diagram 26
2.21 CPO changes the metadata of a charging point sequence diagram 27
2.22 CPO activates a promotion sequence diagram 28
2.23 CPO plans a maintenance session for a charging station sequence diagram 29
2.24 CPO decides the DSO from which acquire energy sequence diagram 30

List of Tables

6.1 Effort spent by each member of the group. 39