DD

1 Introduction

1.1 Purpose

same as rasdd

Electric mobility (e-Mobility) is a way to limit the carbon footprint caused by our urban and sub-urban mobility needs. When using an electric vehicle, knowing where to charge the vehicle and carefully planning the charging process in such a way that it introduces minimal interference and constraints on our daily schedule is of paramount importance.

There are four main entities that need to interact in order to provide the mentioned service:

1. eMSP (e-Mobility Service Providers): an application that links together the final users (owners of electric vehicles) and the charging stations

2. CPOs (Charging Point Operators): they own and manage the charging station
3. DSOs (Distribution System Operators): energy providers

The purpose of this project is to develop e-Mall (e-Mobility For All), a set of applications that:

- will grant the user the possibility to book charges for its vehicles and pay for it, monitoring costs and special offers;
- allows CPOs to handle their own charging areas through CPMS (Charge Point Management System) that manages the charging columns and the energy acquisition for the single charging stations, automatically or manually by employees.

This document outlines the architecture and design for the system, including the various components and their interactions. It also includes mockups of the user interface and a plan for implementing, testing, and integrating the system. Overall, this document serves as a guide for the development process.

1.2 Scope

Our system focuses on the eMSP and CPMS subsystems with all the features listed in the specification document without making the eMSP smarter than it needs to for the end user.

1.3 Definitions, Acronyms, Abbreviations

Definition	Description
Charging column	A device with one ore more standard charging sockets equipped with a NFC tag
Charging station	A group of charging columns displaced in a nearby area owned by a CPO and managed through the CPMS
User	Person interested in using the system
Operator	Instructed personnel that manages a charging station

Acronyms	Description
eMSP	e-Mobility Service Providers application that links together the final users and the charging stations
CPOs	Charging Point Operators owners and managers of the charging station
DSOs	Distribution System Operators energy providers

CPMS	Charge Point Management System manages reservations and energy for charging stations
eMall	Electric Mobility for All
IoT	Internet of Things
NFC	Near Field Communication
CS	Charging station

Abbrevations	Descrpition
RASD	Requirements Analysis and Specification Document
DD	Design Document

1.4 Revision history

1.5 Reference Documents

The specification document “Assignment RDD AY 2022-2023_v3.pdf”

RASD

1.6 Document structure

FARE ALLA FINE!!

Architectural Design

2.1 Overview: High level components and their interaction

The architecture of the eMall system follows for both it’s main components a three-tier architectural pattern but differ in the specific usage.

The eMSP is composed of three levels:

1. eMSP app: This level includes the graphical interface that users use to interact with the application, as well as the logic that allows the application to communicate with the eMSP server and external APIs.

2. eMSP Server level: This level is responsible for communication between the eMSP database and the application. In addition, the server acts as a dispatcher, managing requests from the application and forwarding them to the database or other components of the application as needed. The server of the eMSP also serves as a dispatcher for the server of the CPMS, as we will see later.
3. Database level: This level contains user data, which is used by the application through the server.

The CPMS is composed of three similar levels:

1. Terminal level: This level includes the graphical interface that the CPMS employee uses to interact with the CPMS.
2. Server level: This level is responsible for communication between the database and the terminal. In addition, the server of the CPMS receives requests from the server of the eMSP and forwards them to the CPMS database.
3. Database level: This level contains reservation data for users, which is used by the CPMS through the server.

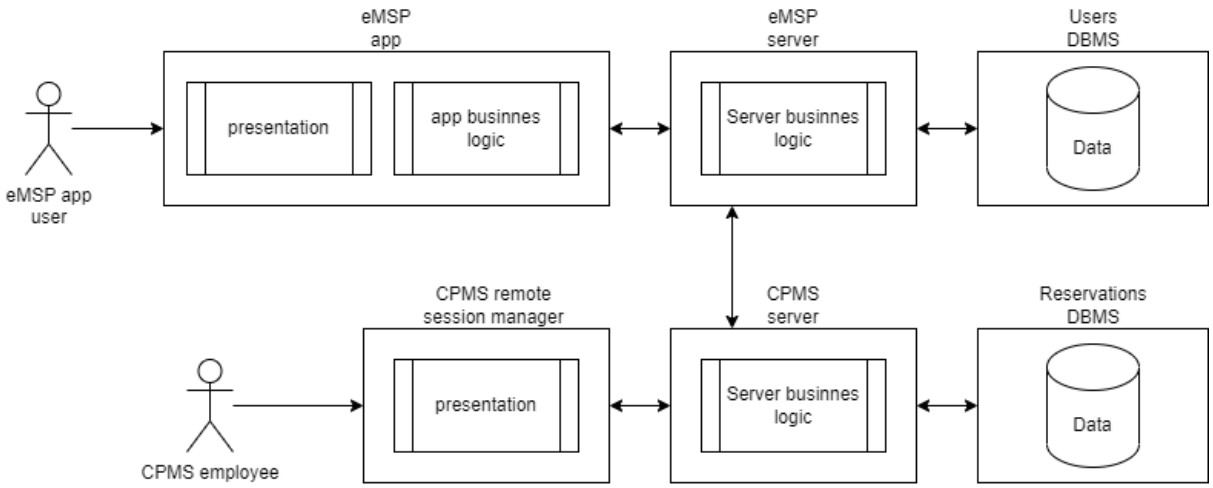


Figure 1: Overview of the system architecture

2.2 Component view

This section of the document presents the component view of the system through a component diagram and accompanying descriptions. The diagram illustrates the three main components of the system and the APIs that they use to communicate with one another.

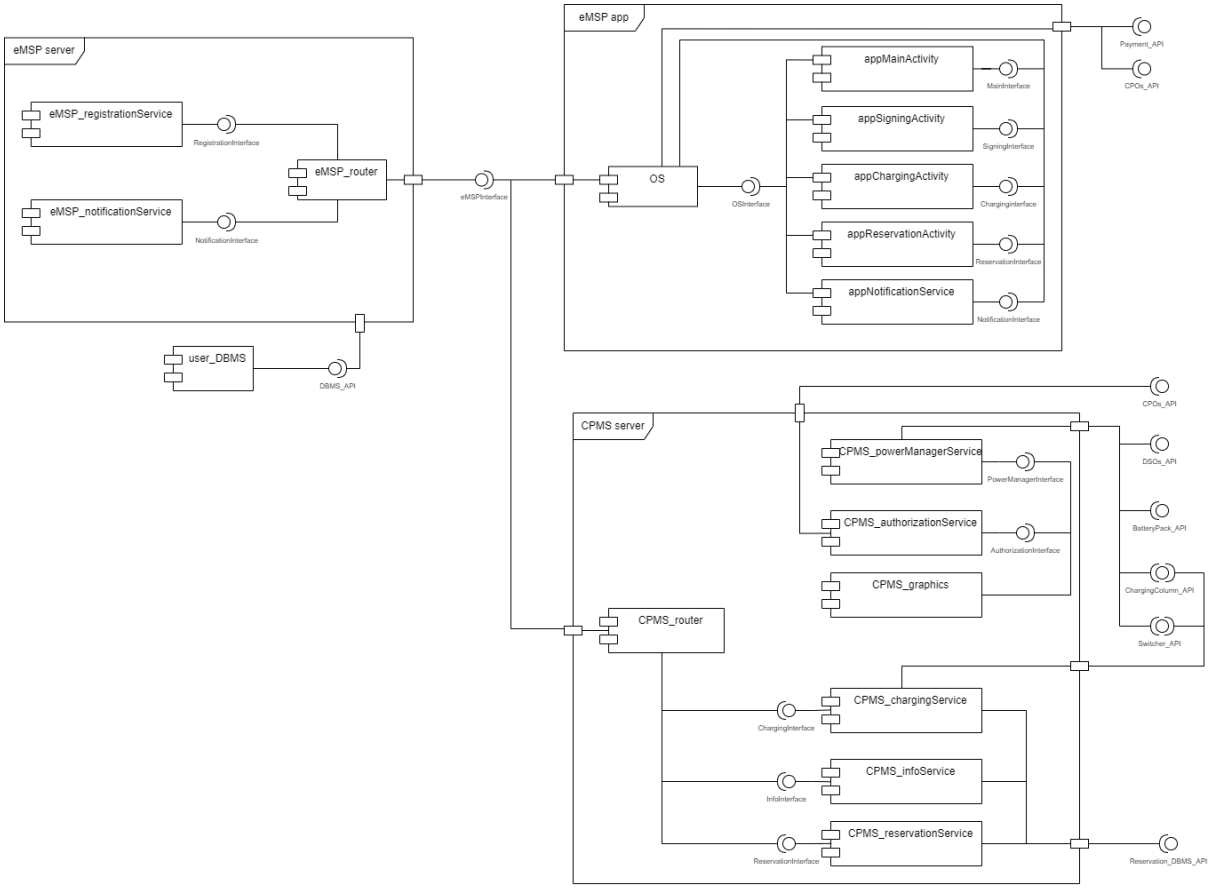


Figure 2: Component diagram of the system

In the following list all present components, divided in the three main systems, are described:

eMSP app

- **OS**: is the software that manages all of the hardware and software resources of a device. It provides a platform for other software to run on top of, and it provides a consistent interface for the user to interact with the system components.
- **appMainActivity**: is the central component of the app that is responsible for displaying the main screen, which typically includes a map with charging stations markers and various interactive elements, as well as managing the startup of other activities within the app. It uses the OS system to access the device's hardware and software resources and to communicate with other apps and the user. The appMainActivity displays the main screen and launches other activities in response to user actions or certain events.
- **appSigningActivity**: is responsible for managing the registration, email confirmation, payment verification, and login process for users as it includes various screens and forms for the user to enter their information and complete

the registration process. It also uses an email confirmation process to verify the user's identity and ensure that they are using a valid email address and includes a payment verification process to ensure that the user can complete any required payment before being allowed to access the app's content. All data about the registered users is saved (and can thus be retrieved) in the user_DBMS through the use the eMSP_API

- appChargingActivity: it is responsible for handling user input, such as reading the NFC token of a charging column or accepting a user-inserted code, and displaying output on the screen. It also communicates with the payment_API to process the payment for the charging by using security features to protect the user's payment information and prevent unauthorized access.
- appReservationActivity: the activity is started from the main activity by passing the charging station handle of interest. It then displays a form for the reservation of a timeframe that the user must fill. When the reservation is sent and confirmed through the CPMS_API the booking fee is processed by using the payment_API
- appNotificationService: it runs in the background and regularly checks for notifications from charging stations for the user. It does so by sending a request to the eMSP server and receiving a response with any available notifications it has for the user.

eMSP server

- eMSP_router: it is responsible for receiving requests from eMSP apps and CPMS servers and routing them to the appropriate internal services for processing, and returning the correct response. It also includes security features to protect the user's information and prevent unauthorized access.
- eMSP_registrationService: it is responsible for handling requests related to user registration and login, adding new user entries to the Users_DBMS, and verifying the correct credentials for login requests.
- eMSP_notificationService: is responsible for storing notifications received from CPMSs in a buffer and delivering them to the intended recipient when requested by the eMSP apps.

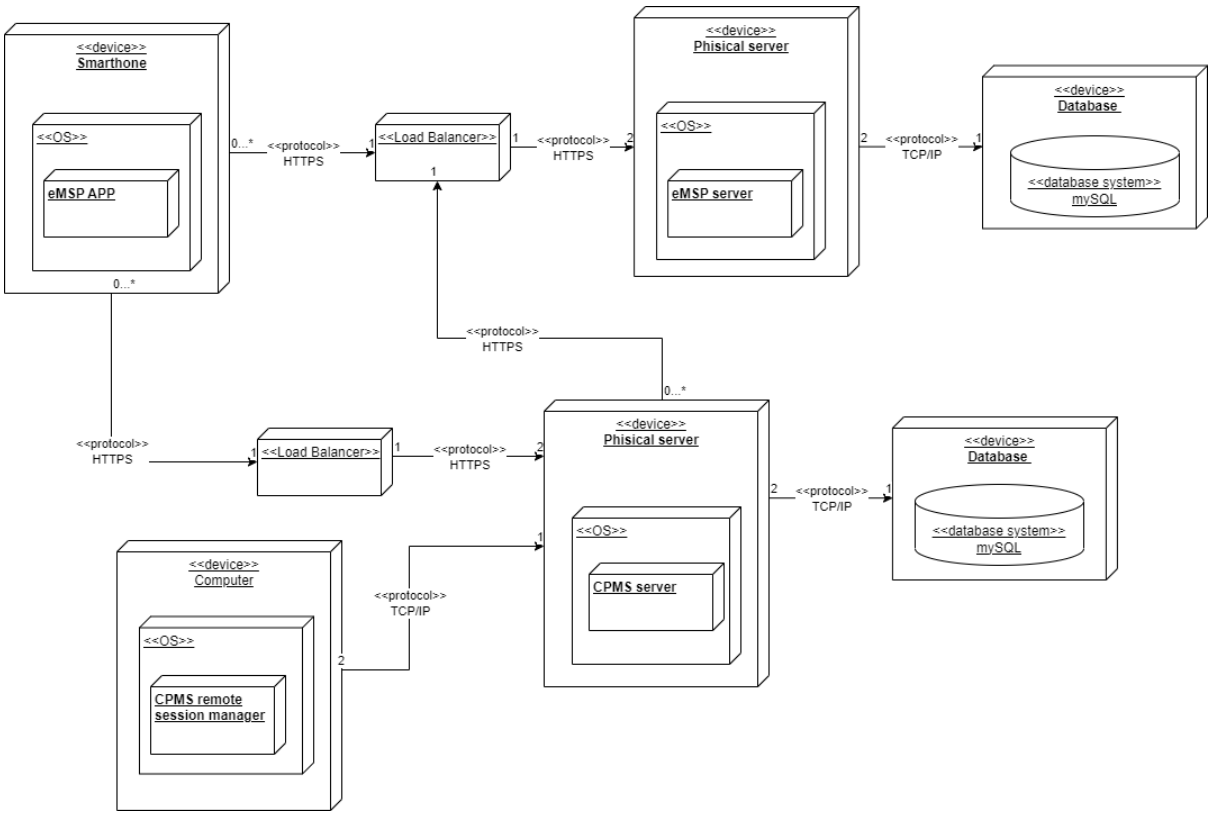
CPMS server

- CPMS_router: software component that is responsible for handling incoming and outgoing requests between eMSP apps and servers. Incoming requests are dispatched in the appropriate internal services for processing and outgoing request are sent to the eMSP servers to receive a response.

- CPMS_powerManagerService: software component responsible for the automatic management of the entire charging station. It does this by using API connections to the DSOs, BatteryPack, Charging Columns, and Switcher. The power manager service is also used indirectly by CPMS employees to override automatic decisions through the graphical interface component.
- CPMS_authorizationService: it is responsible for verifying the credentials of CPO employees. It does this by using the CPOs_API. It also includes security features to protect the user's information and prevent unauthorized access.
- CPMS_graphics: it is responsible for handling the graphical interface that CPO employees use to interact with the charging station. The interface allows the employees to login, view the status of the charging station, and perform manual changes as needed.
- CPMS_chargingService: it is a component that is responsible for managing the charging process for vehicles at a charging station. It receives the token from the eMSP app, verifies the presence of a valid reservation for that user, and enables the charging column sockets through the use of the API. It also monitors the vehicle battery to regulate the current and sends notifications to the user through the eMSP server when the reservation time is up or the vehicle is fully charged.
- CPMS_infoService: it is responsible for managing the response to eMSP apps with information about the charging station. This information may include prices, offers, availability of free time slots, and charging speeds.
- CPMS_reservationService: it is responsible for handling requests to create a reservation, using the Reservations_DBMS_API to insert the reservation into the DBMS, and returning a response to the sender.

2.3 Deployment view

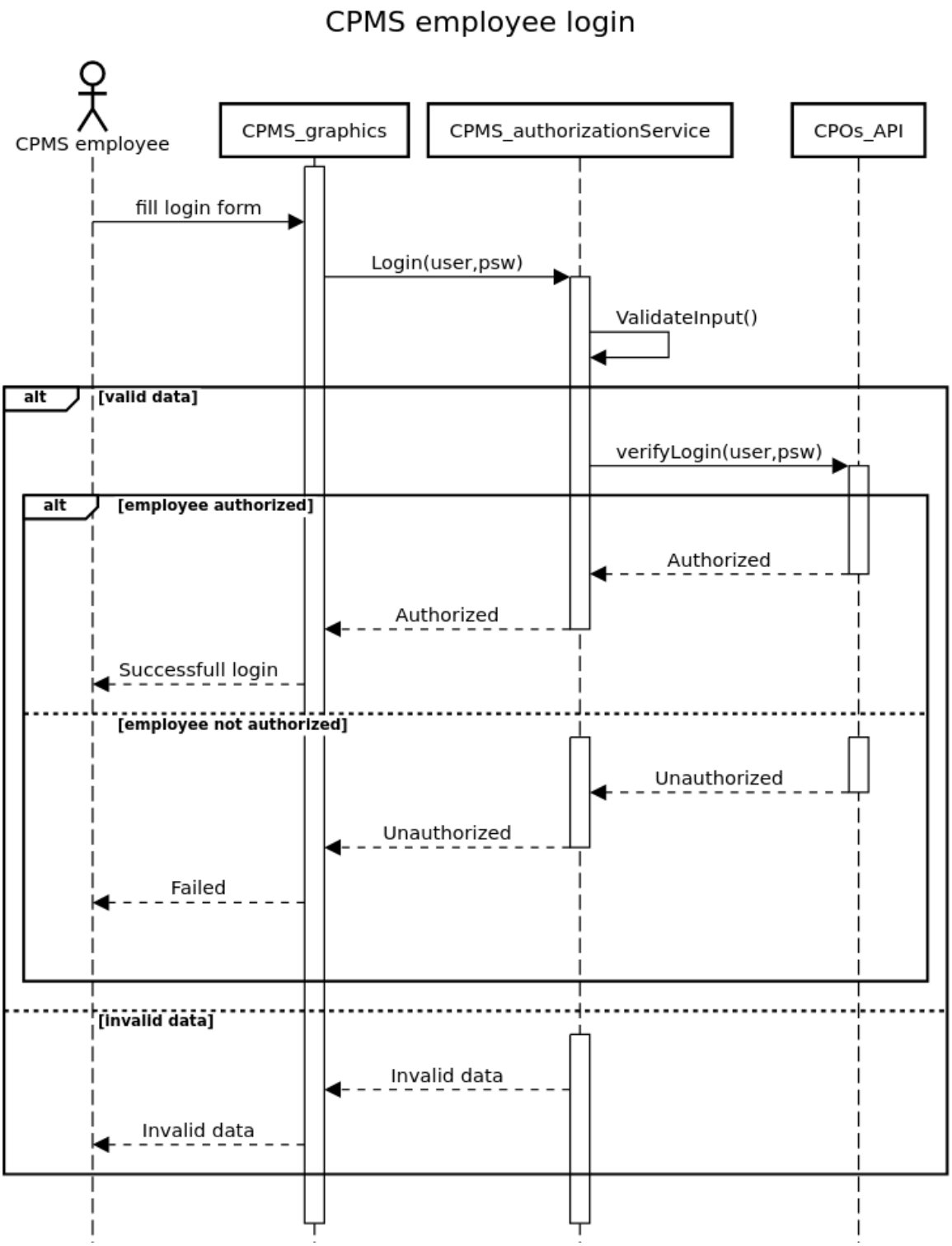
This section provides a deployment diagram that shows how the system will be deployed. It includes information about the environments, tools, and protocols that will be used to build and connect the various parts of the system



NOTE: nell’immagine cambiare DBMS_API con user_DBMS_API

modificare le icone sui sequence diagram: DBMS_APIs, quindi non databases e i rettangolini con componenti

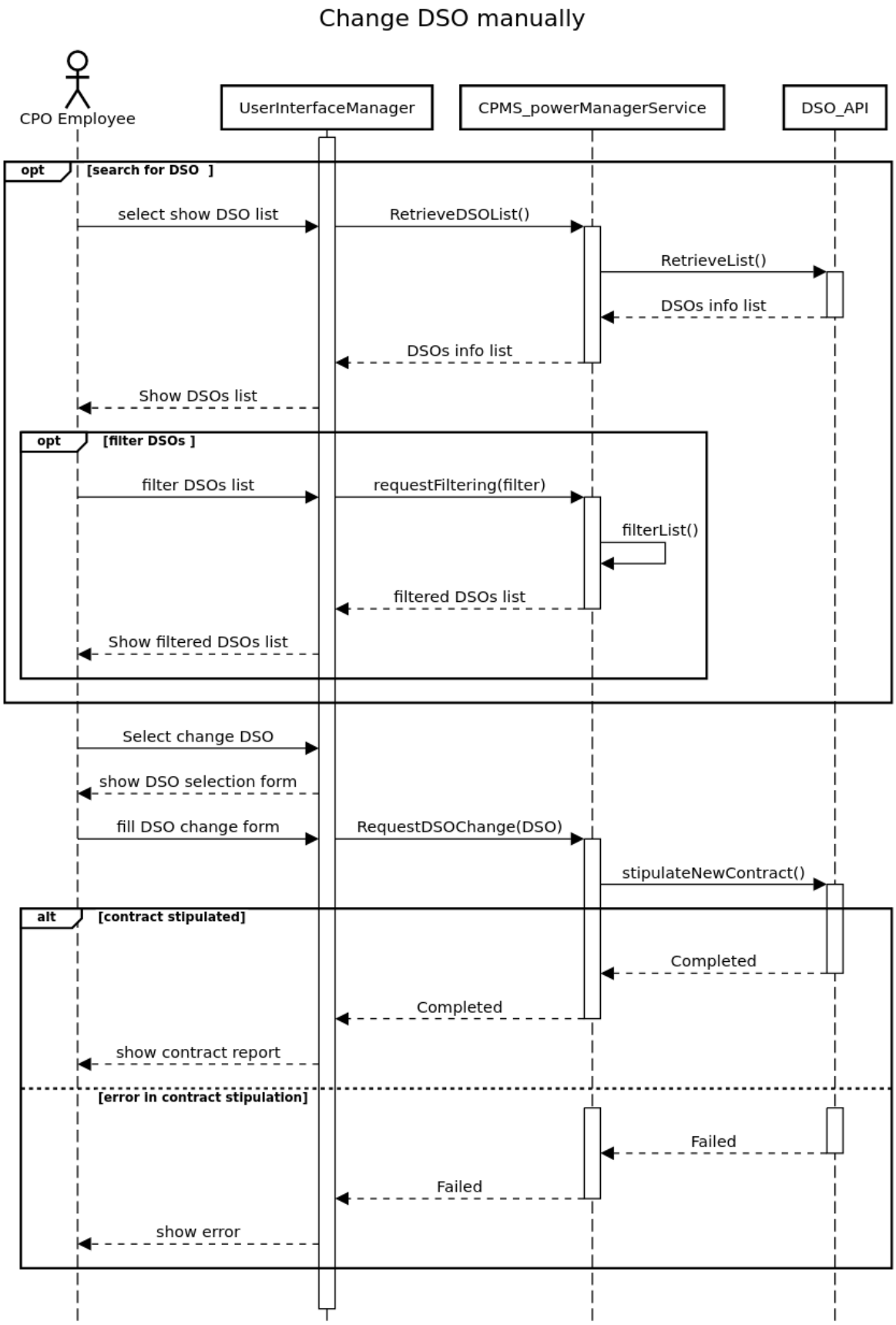
Title: CPMS Employee Login Process



The CPMS employee initiates the login process by entering their login credentials into a form provided by the CPMS graphics module. The login credentials are sent to the CPMS authorization service, which checks that they are in the correct format and verifies the employee's authorization by sending a request to the CPOs API with the login credentials. If the employee is authorized and the login credentials are in the correct format, the login is considered successful and the employee is granted

access to the system. If the employee is not authorized or the login credentials are not in the correct format, the login is considered unsuccessful and the employee is not granted access to the system.

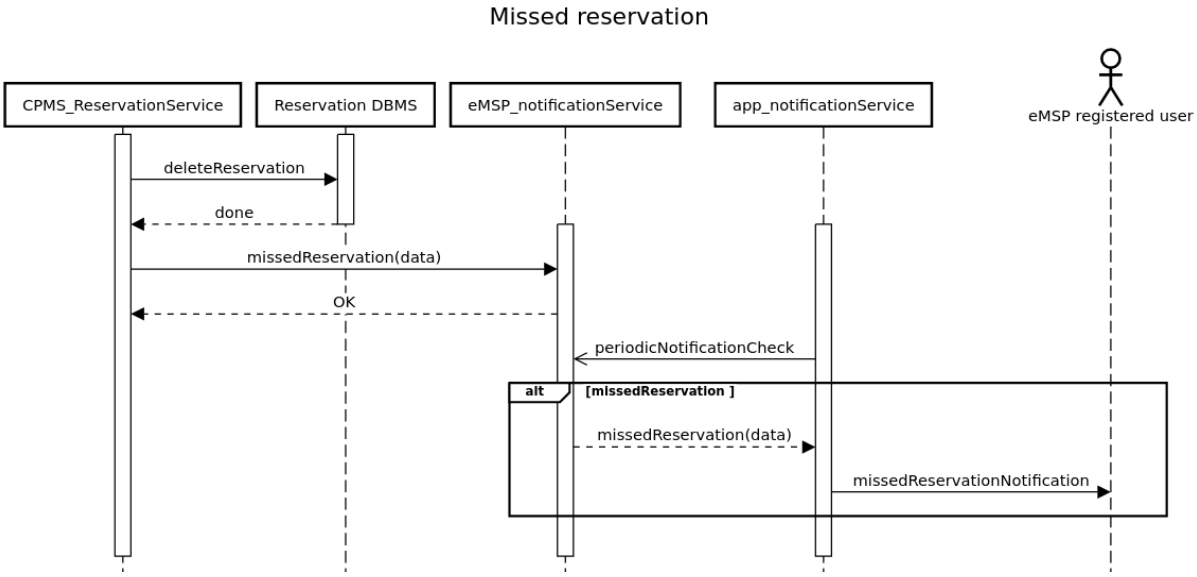
Change DSO Manually



The CPO employee has the option to either view the list of DSOs or filter the list to view a specific subset. If they choose to filter the list, the user interface manager

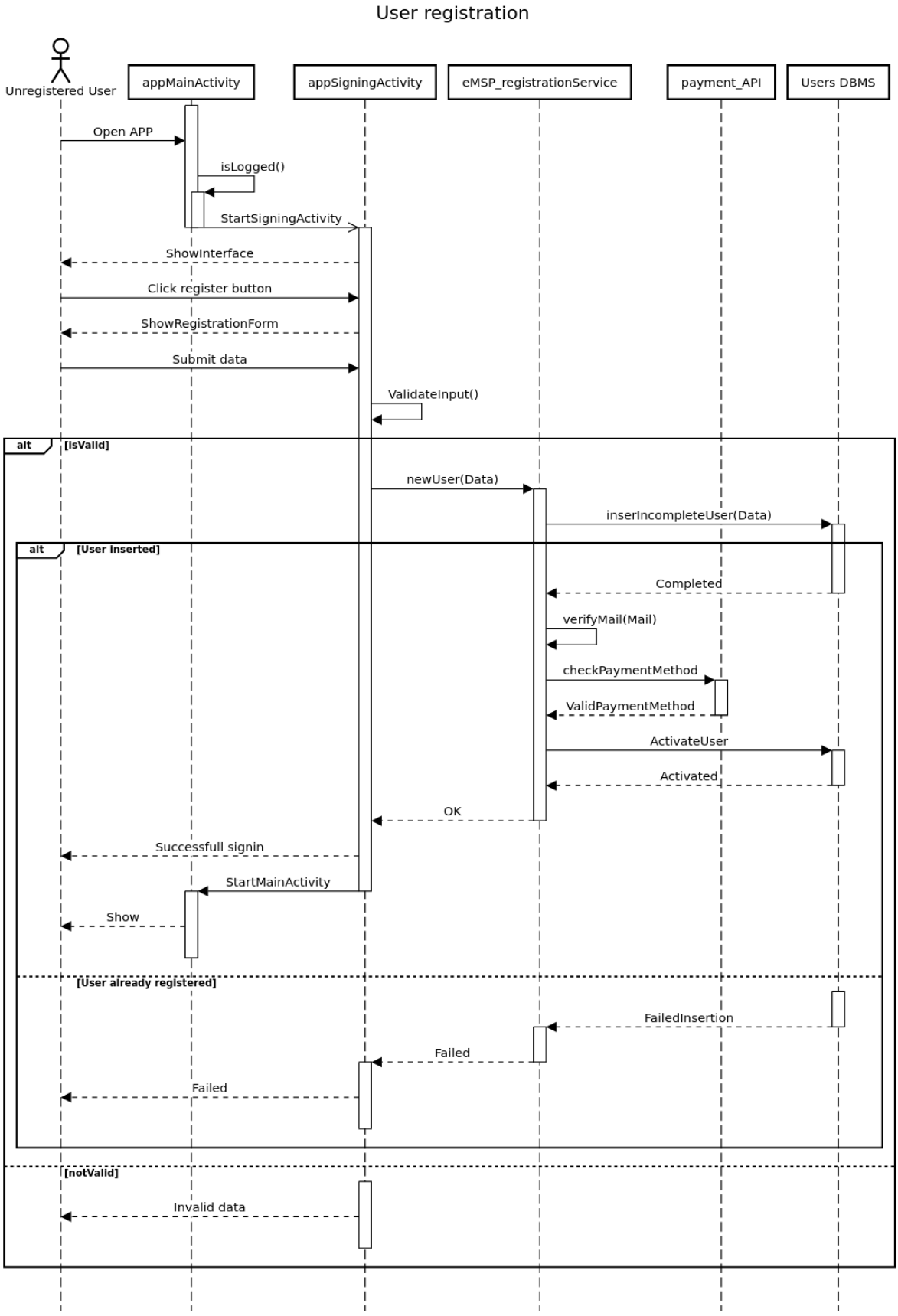
sends a request to the CPMS power manager service to filter the list. The power manager service filters the list and sends the filtered list back to the user interface manager, which displays it to the CPO employee. The CPO employee then selects the option to change the DSO and fills out a form specifying the desired change. This form is sent to the CPMS power manager service, which sends a request to the DSO API to stipulate a new contract with the specified DSO. If the contract is successfully stipulated, a report of the completed contract is displayed to the CPO employee. If there is an error in stipulating the contract, an error message is displayed to the CPO employee.

Missed reservation



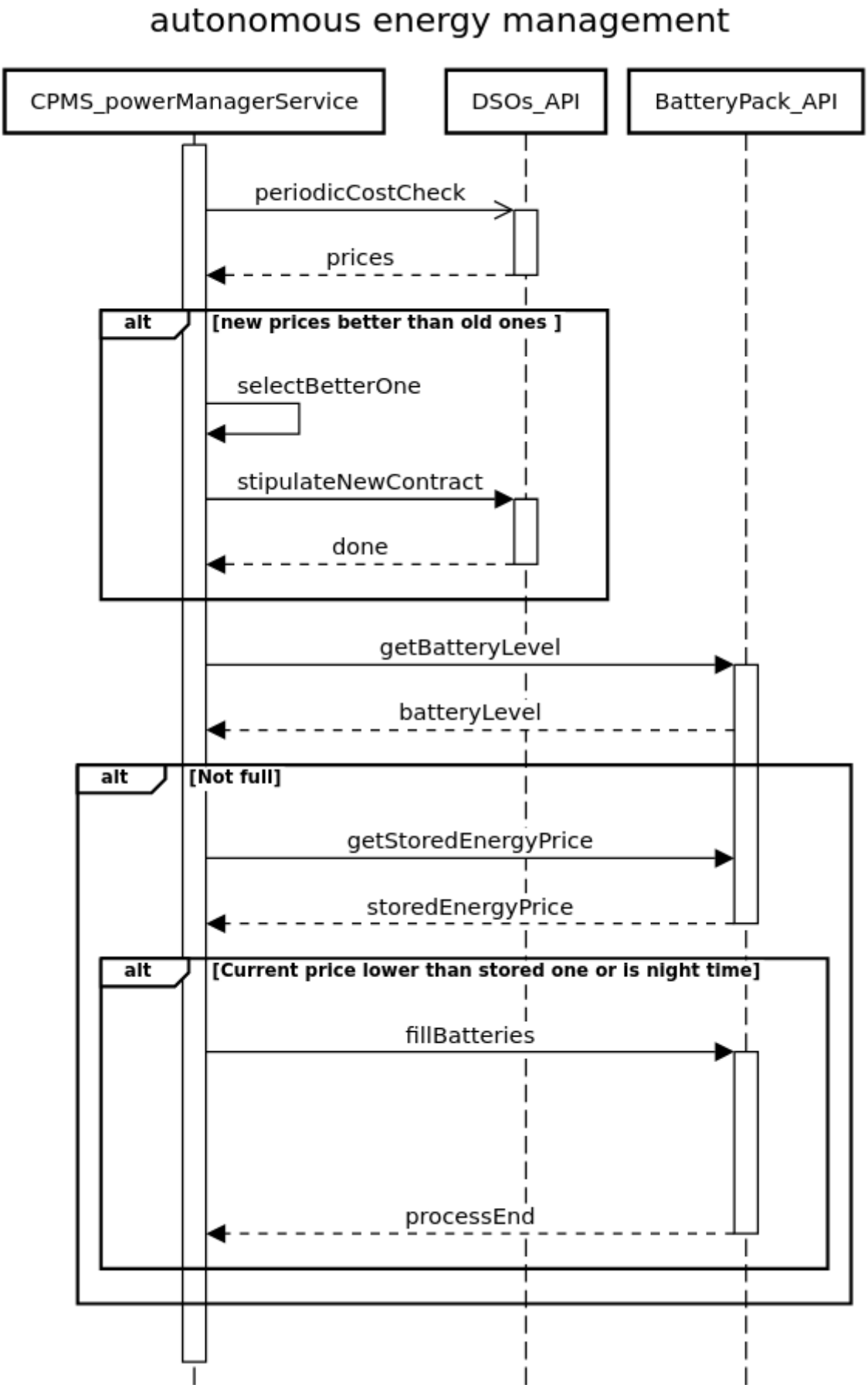
The reservation service sends a request to the reservation DBMS to delete the reservation. The DBMS processes the request and sends a response back to the reservation service indicating that the action has been completed. The reservation DBMS is then deactivated. The eMSP notification service and the app notification service are activated and the reservation service sends a notification to the eMSP notification service indicating that a reservation has been missed. The eMSP notification service sends a response back to the reservation service indicating that the action was successful. The app notification service then sends a request to the eMSP notification service to check for periodic notifications. If the eMSP notification service has a notification of a missed reservation, it sends this notification to the app notification service, which then sends a notification to the eMSP registered user.

User reservation



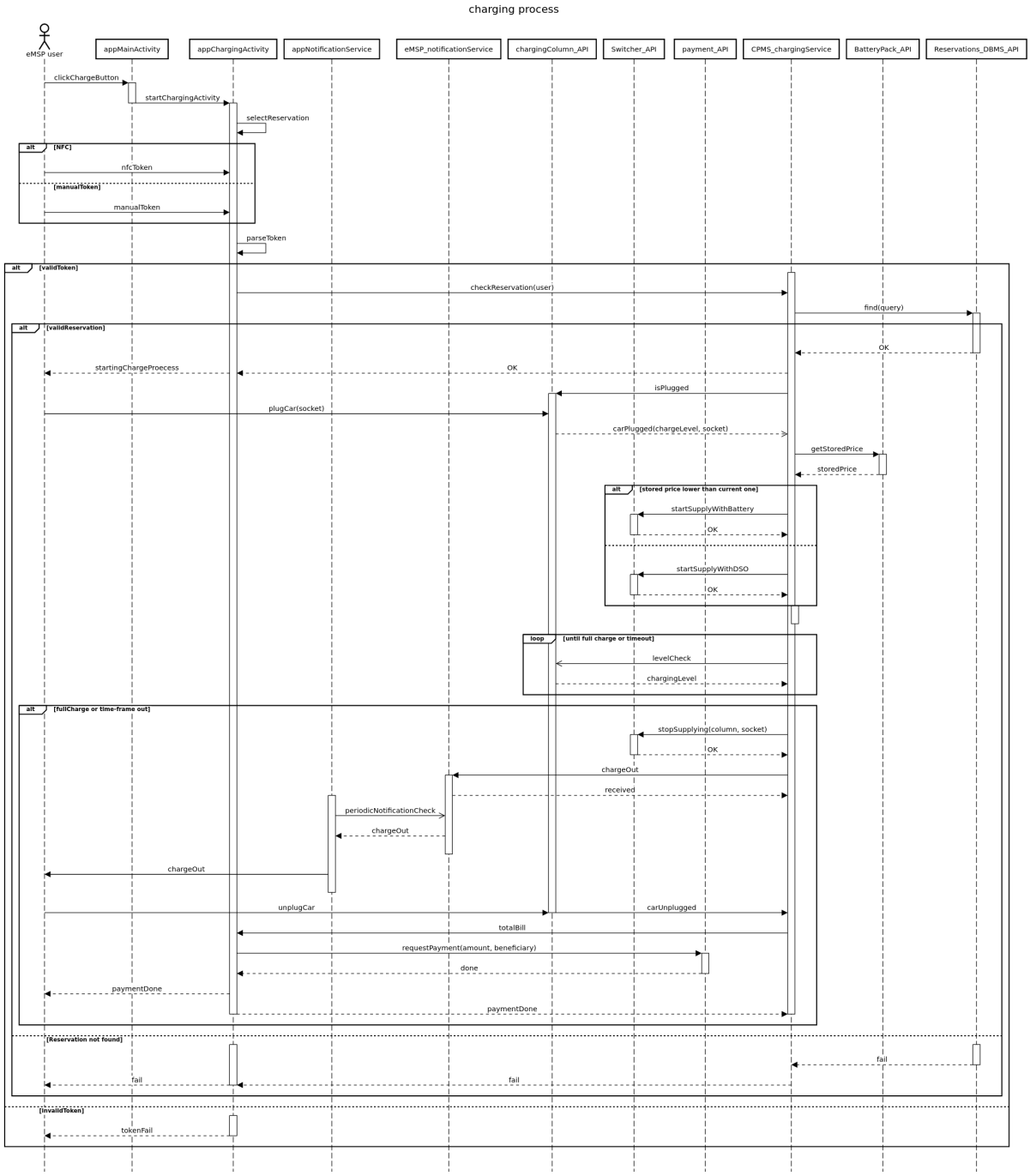
The "User Registration" process begins when the unregistered user opens the app and the app main activity is activated. The app main activity checks if the user is logged in, and if they are not, it starts the app signing activity. The app signing activity displays an interface to the unregistered user, who clicks the "register" button. The app signing activity displays a registration form to the user, who submits their data. The app signing activity checks that the input is valid, and if it is, it sends the data to the eMSP registration service. The registration service sends a request to the user database management system (DBMS) to insert an incomplete user with the submitted data. If the insertion is successful, the registration service verifies the user's email and checks their payment method with the payment API. If the payment method is valid, the registration service sends a request to the user DBMS to activate the user. If the user is successfully activated, the registration process is considered successful and the user is granted access to the app. If the user is already registered or there is an error in the insertion or activation process, the registration process is considered unsuccessful and the user is not granted access to the app. If the input is not valid, the user is prompted to enter valid data.

Autonomous energy management


autonomous energy management
CPMS_powerManagerService
DSOs_API
BatteryPack_API
periodicCostCheck
prices
alt
[new prices better than old ones]
selectBetterOne
stipulateNewContract
done
getBatteryLevel
batteryLevel
alt
[Not full]
getStoredEnergyPrice
storedEnergyPrice
alt
[Current price lower than stored one or is night time]
fillBatteries
processEnd

The power manager service sends a periodic request to the DSOs API to check for updated energy prices. If the prices are updated and are better than the previous ones, the power manager service selects the better option and sends a request to the DSOs API to stipulate a new contract with the selected option. The power manager service then sends a request to the battery pack API to retrieve the current battery level. If the battery is not full, the power manager service retrieves the stored energy price from the battery pack API. If the current price is lower than the stored price or it is night time, the power manager service sends a request to the battery pack API to fill the batteries. The battery pack API processes the request and sends a response back to the power manager service indicating that the action has been completed.

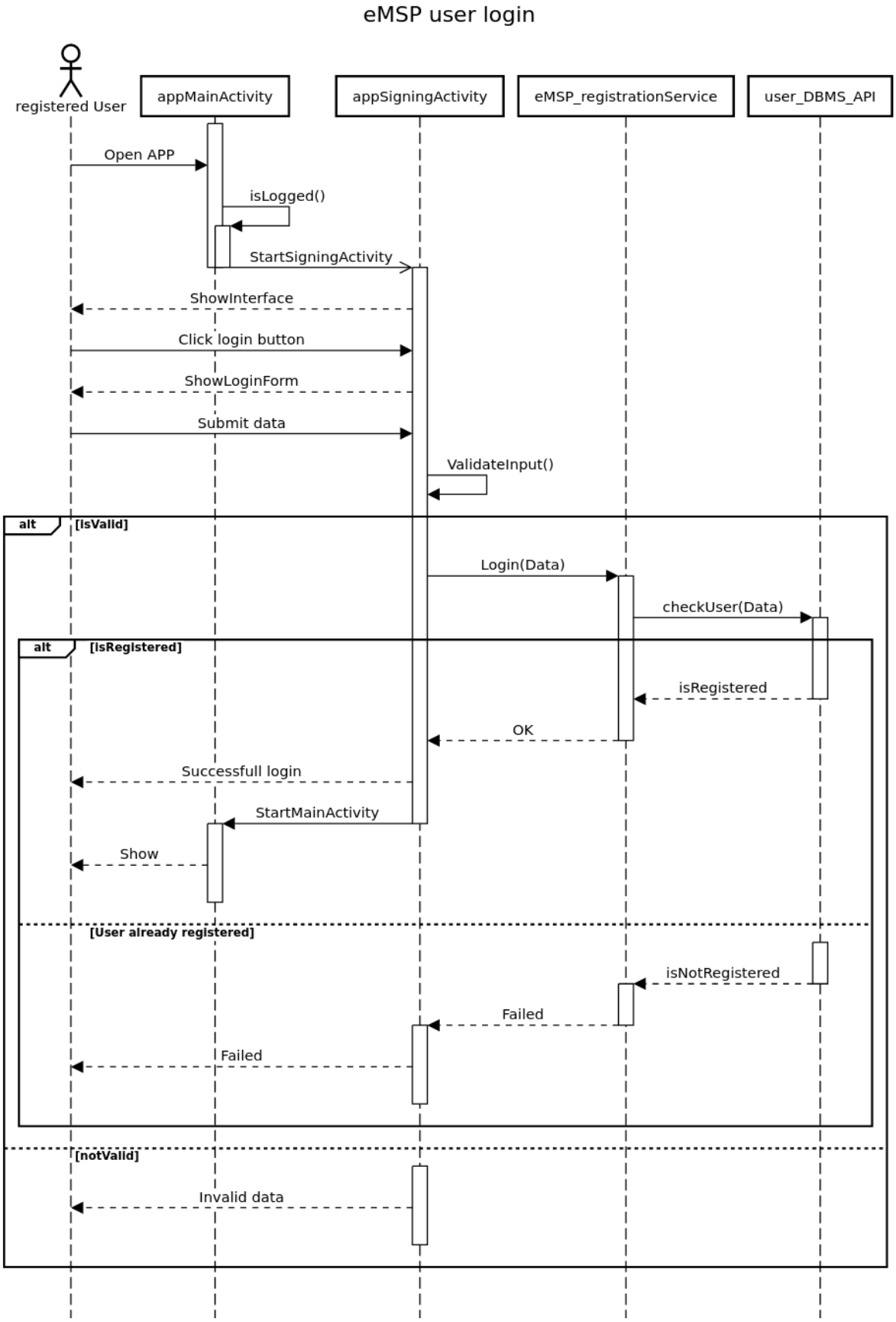
Charging process



The "charging process" flow describes the steps taken when an eMSP user initiates a charging session through the app. The app main activity is opened, and the user clicks on the charge button which opens the app charging activity. The user then selects their reservation method (either NFC or manual input) and enters the necessary information. The app validates the reservation and sends a request to the CPMS charging service to check the reservation. If the reservation is valid, the charging process begins by checking if the car is plugged in and to which socket. The CPMS charging service then determines the most cost-effective power source (either the battery pack or DSO) and starts the charging process. The process

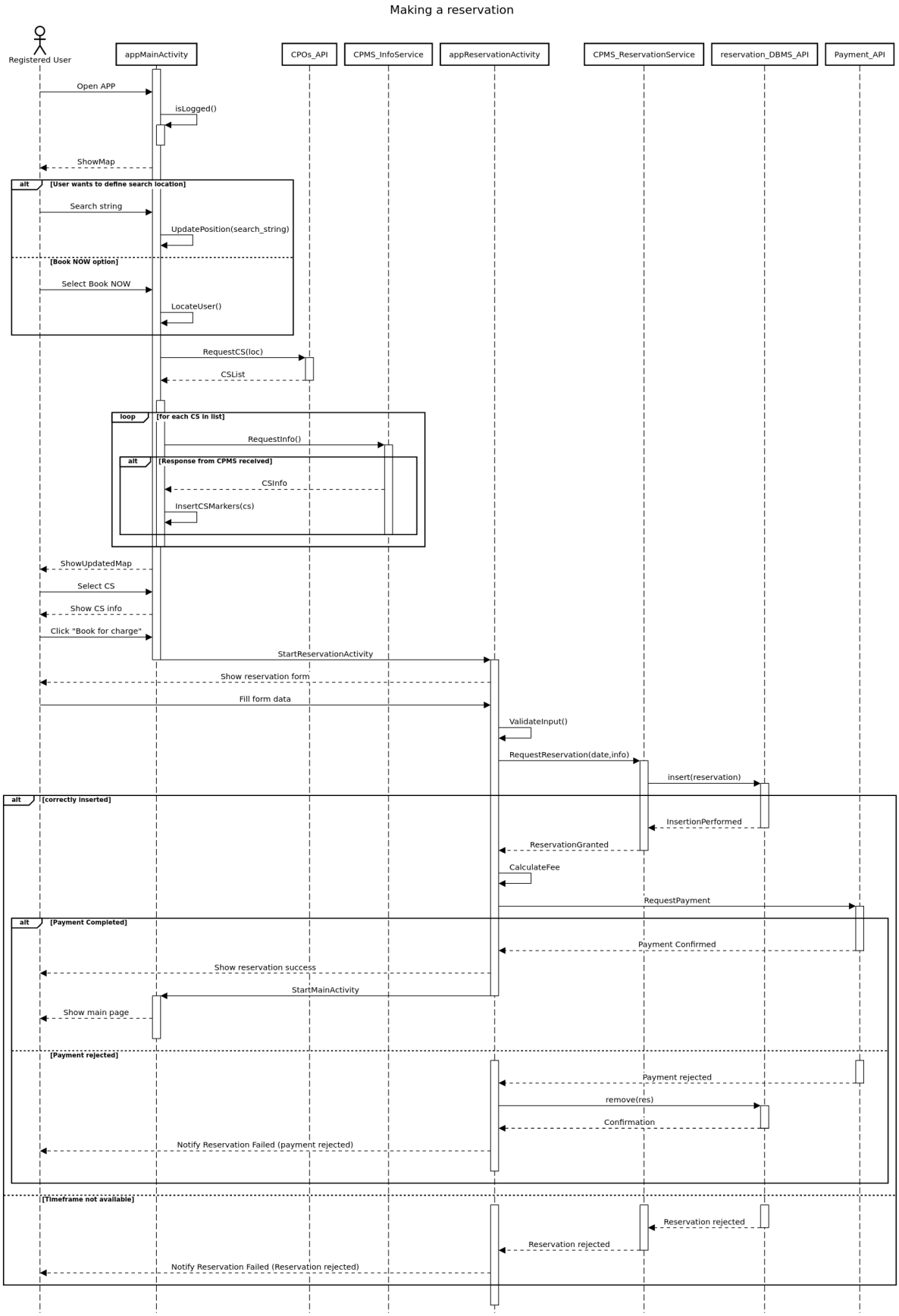
continues until the car is fully charged or the time frame for the reservation has been reached. If the car becomes fully charged or the reservation time has ended, the charging process is stopped and the user is notified. The user then unplugs their car, and the total cost for the charging session is calculated and displayed in the app charging activity.

eMSP login


eMSP user login
registered User
appMainActivity
appSigningActivity
eMSP_registrationService
user_DBMS_API
Open APP
isLogged()
StartSigningActivity
ShowInterface
Click login button
ShowLoginForm
Submit data
ValidateInput()
alt
[isValid]
Login(Data)
checkUser(Data)
alt
[isRegistered]
isRegistered
OK
Successfull login
StartMainActivity
Show
[User already registered]
isNotRegistered
Failed
Failed
[notValid]
Invalid data

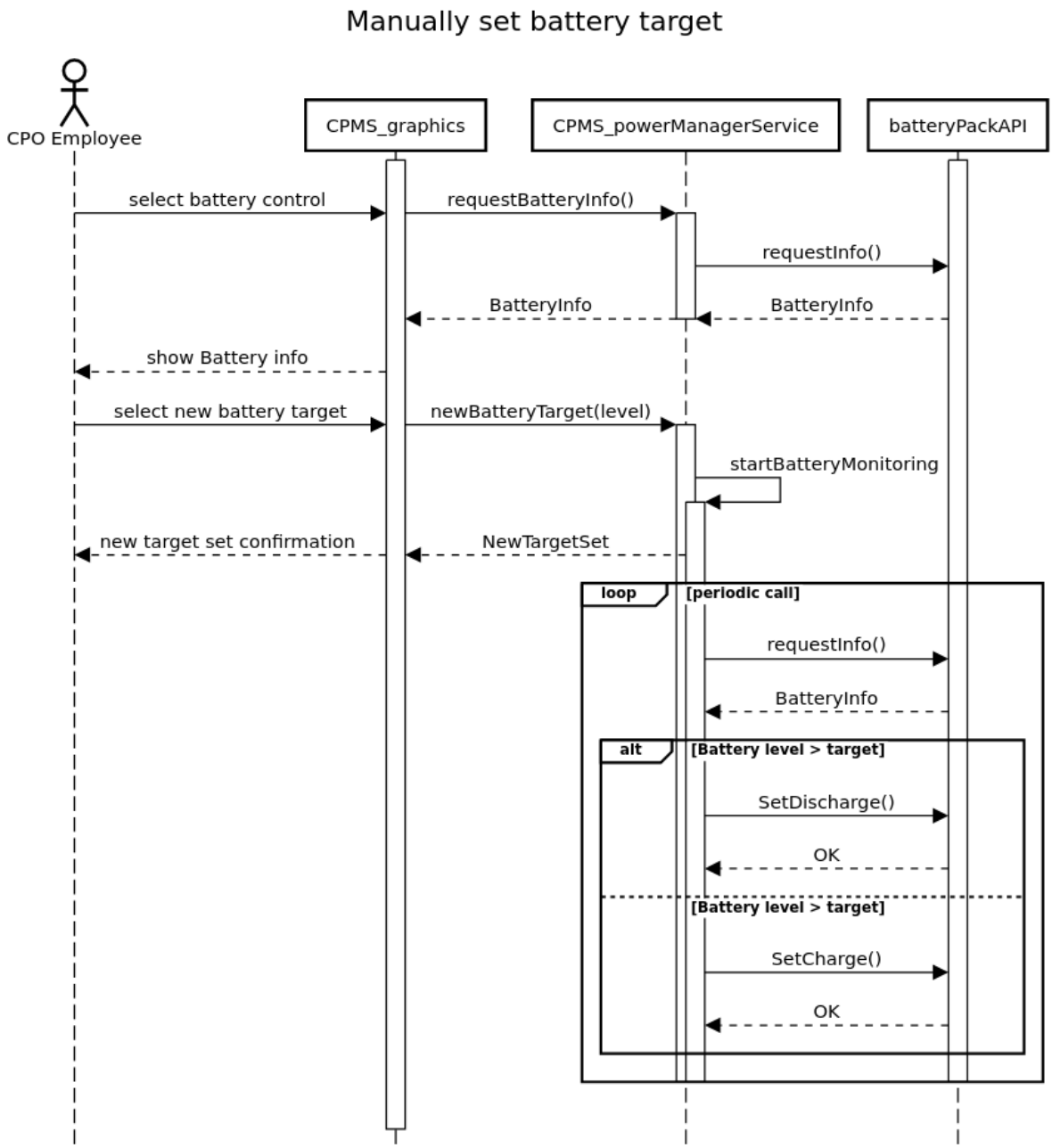
In this activity a user log in the eMSP app. The app first checks if the user is already logged in, and if not, it directs the user to the app signing activity where they can click the login button. The user then enters their login data, which the app checks for validity. If the data is valid, the app sends a request to the eMSP registration service to log the user in. The registration service checks the user database to see if the user is registered. If the user is registered, the login process is successful and the user is directed to the main activity of the app. If the user is not registered or the login data is invalid, the login process fails and the user is notified.

Make a reservation


Making a reservation
Registered User
appMainActivity
CPOs_API
CPMS_InfoService
appReservationActivity
CPMS_ReservationService
reservation_DBMS_API
Payment_API
Open APP
isLogged()
ShowMap
alt
[User wants to define search location]
Search string
UpdatePosition(search_string)
[Book NOW option]
Select Book NOW
LocateUser()
RequestCS(loc)
CSList
loop
[for each CS in list]
RequestInfo()
alt
[Response from CPMS received]
CSInfo
InsertCSMarkers(cs)
ShowUpdatedMap
Select CS
Show CS info
Click "Book for charge"
StartReservationActivity
Show reservation form
Fill form data
ValidateInput()
RequestReservation(date,info)
insert(reservation)
alt
[correctly inserted]
InsertionPerformed
ReservationGranted
CalculateFee
RequestPayment
alt
[Payment Completed]
Payment Confirmed
Show reservation success
StartMainActivity
Show main page
[Payment rejected]
Payment rejected
remove(res)
Confirmation
Notify Reservation Failed (payment rejected)
[Timeframe not available]
Reservation rejected
Reservation rejected
Notify Reservation Failed (Reservation rejected)

This is a flow diagram that describes the process of making a reservation. The diagram begins with the user opening the app and being shown a map of nearby CSs. The user then has the option to either define their own location or use the "Book NOW" option to automatically locate themselves. The app sends a request to a Charging Point Operators (CPO) API to retrieve a list of nearby CSs. This list is then used to request more information on each CS through the Charging Point Management System (CPMS) InfoService. The app then updates the map with markers for each CS and displays it to the user. When the user selects a specific CS, they are shown its information and have the option to "Book for charge". This triggers the app to start a reservation activity and prompts the user to fill out a reservation form. The form data is validated, and a request is made to the CPMS ReservationService to create a reservation. If the reservation is successfully inserted into the reservation database, the app calculates the fee and requests payment. If the payment is completed, the reservation is considered successful and the user is shown a confirmation. If the payment is rejected or the reservation timeframe is not available, the reservation is cancelled and the user is notified of the failure.

Change battery target



This is a sequence diagram that describes the process of a CPO employee manually setting a target battery level for a battery pack. It begins when the CPO employee selects the battery control in the CPMS graphics interface. The CPMS power manager service is then activated, and it requests battery information from the battery pack API. The battery pack API responds with the battery information, which is then shown to the CPO employee through the CPMS graphics interface. The CPO employee can then select a new battery target level, which is passed to the CPMS power manager service. The service then starts monitoring the battery level, and periodically requests updates from the battery pack API. Depending on the current battery level in relation to the target level, the power manager service will either instruct the battery pack API to charge or discharge the battery.