DD

1 Introduction

1.1 Purpose

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

This document is structured into several sections, including an introduction, architectural design, user interface design, requirements traceability, and the implementation, integration and test plan. The introduction provides an overview of the purpose and scope of the document, as well as definitions of acronyms and abbreviations used throughout the document. It also includes a revision history and a list of reference documents. The architectural design section provides an overview of the high-level components of the system and their interactions, as well as more detailed views of the system's components, deployment, and runtime. It also includes descriptions of specific processes, such as employee login, user registration, and charging, and lists the interfaces between components. The user interface design section describes the user interfaces for the eMSP app and the CPMS, through mockups of all user interactable interfaces. The requirements traceability section defines the requirements for the system and maps them to the relevant components. The final section outlines the plan for implementing, integrating, and testing the system.

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.

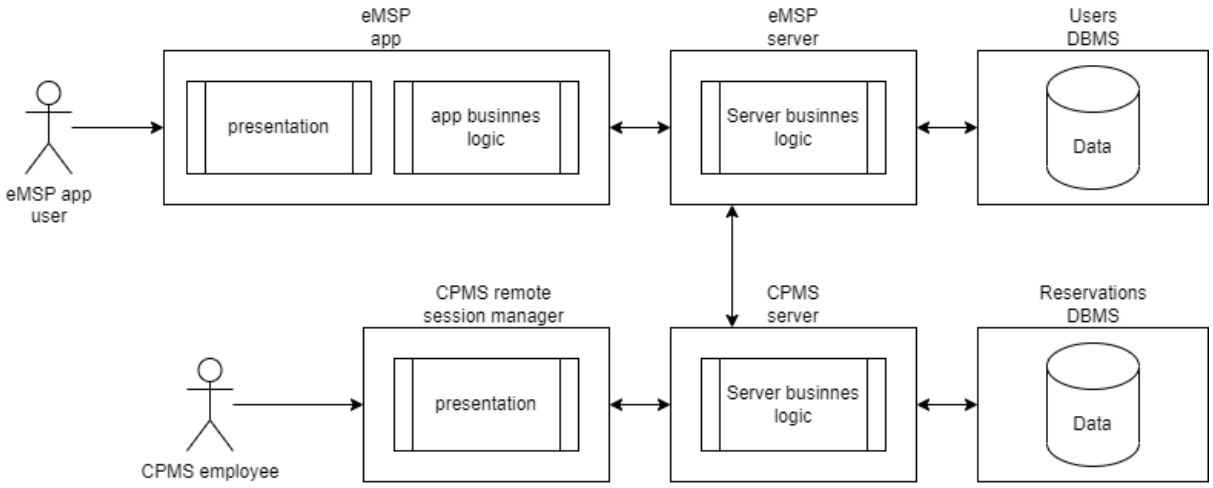


Figure 1: Overview of the system architecture

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.

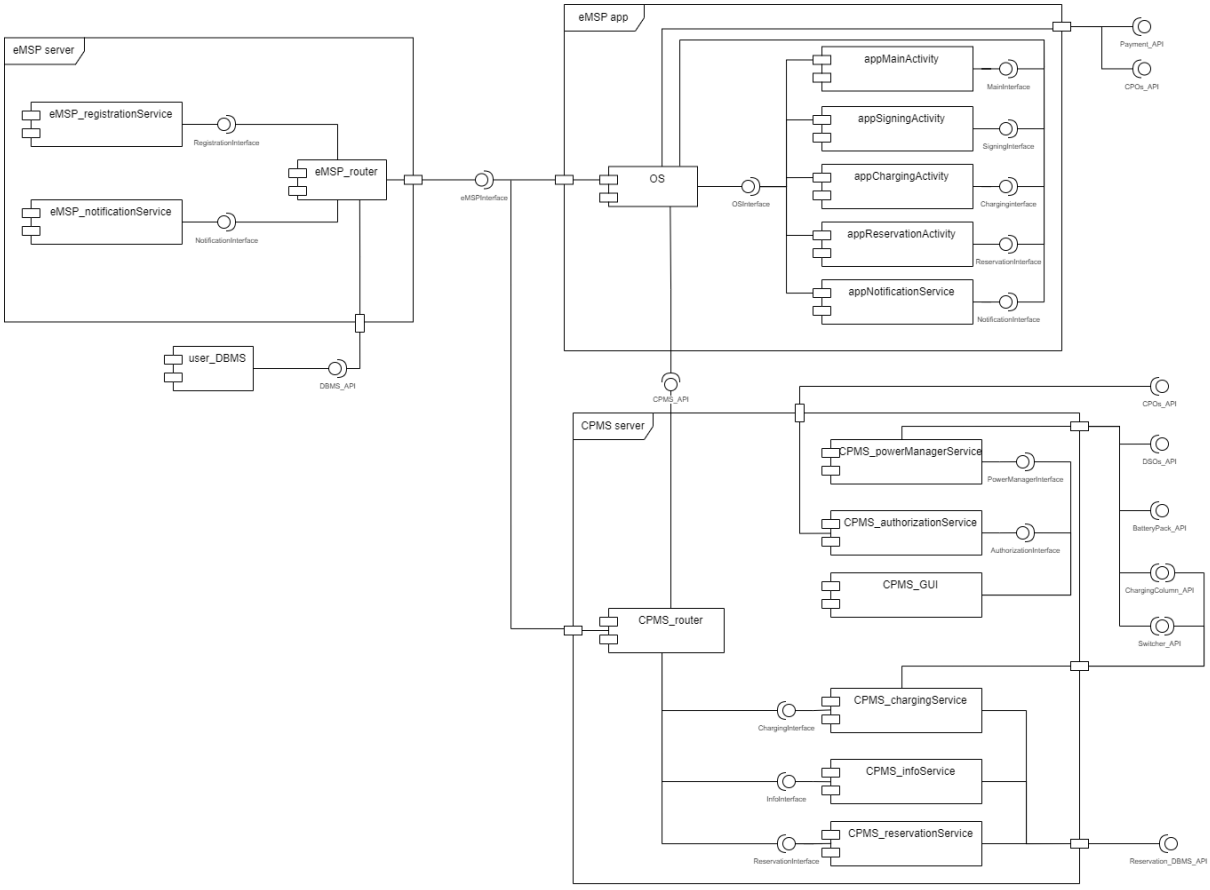


Figure 2: Component diagram of the system

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

eMSP app

- **OS**: is the software (Android or iOS) 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. This module is considered already fully deleloped and not to be implemented.
- **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, that’s means that the service is always running, 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. It is always running.

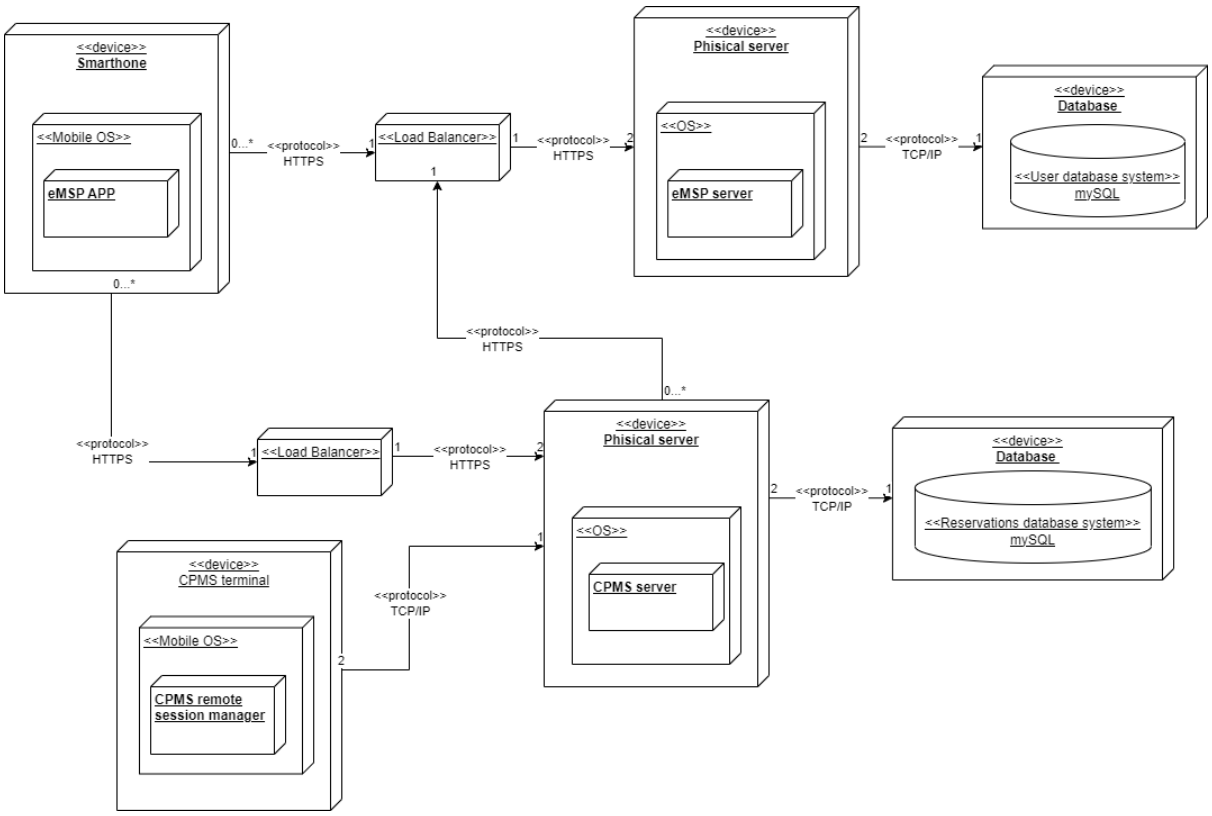
CPMS server

- 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_GUI: 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



- **smartphone:** A mobile device that is designed to allow you to make phone calls, send text messages, and perform a variety of other tasks. In the device runs a system (OS) which is the software that manages all of the device's hardware and software resources. The eMSP app is a software application designed to be used by electric vehicle owners. When an eMSP app is installed on a smartphone, the device becomes a portable tool that users can use to locate and make reservations in charging stations for electric devices, as well as manage their accounts.
- **load balancer:** A load balancer is a device that distributes network or application traffic across a number of servers. A load balancer serves as the single point of contact for clients and distributes incoming requests to one of the available servers. In the case of HTTP load balancers, they specifically handle traffic for HTTP and HTTPS protocols. They can improve the availability and fault tolerance of a network by ensuring that traffic is distributed evenly among servers and by automatically routing traffic away from servers that are experiencing issues.
- **physical server:** physical server computer is a high-performance computer that is designed to host and manage software applications and services for other

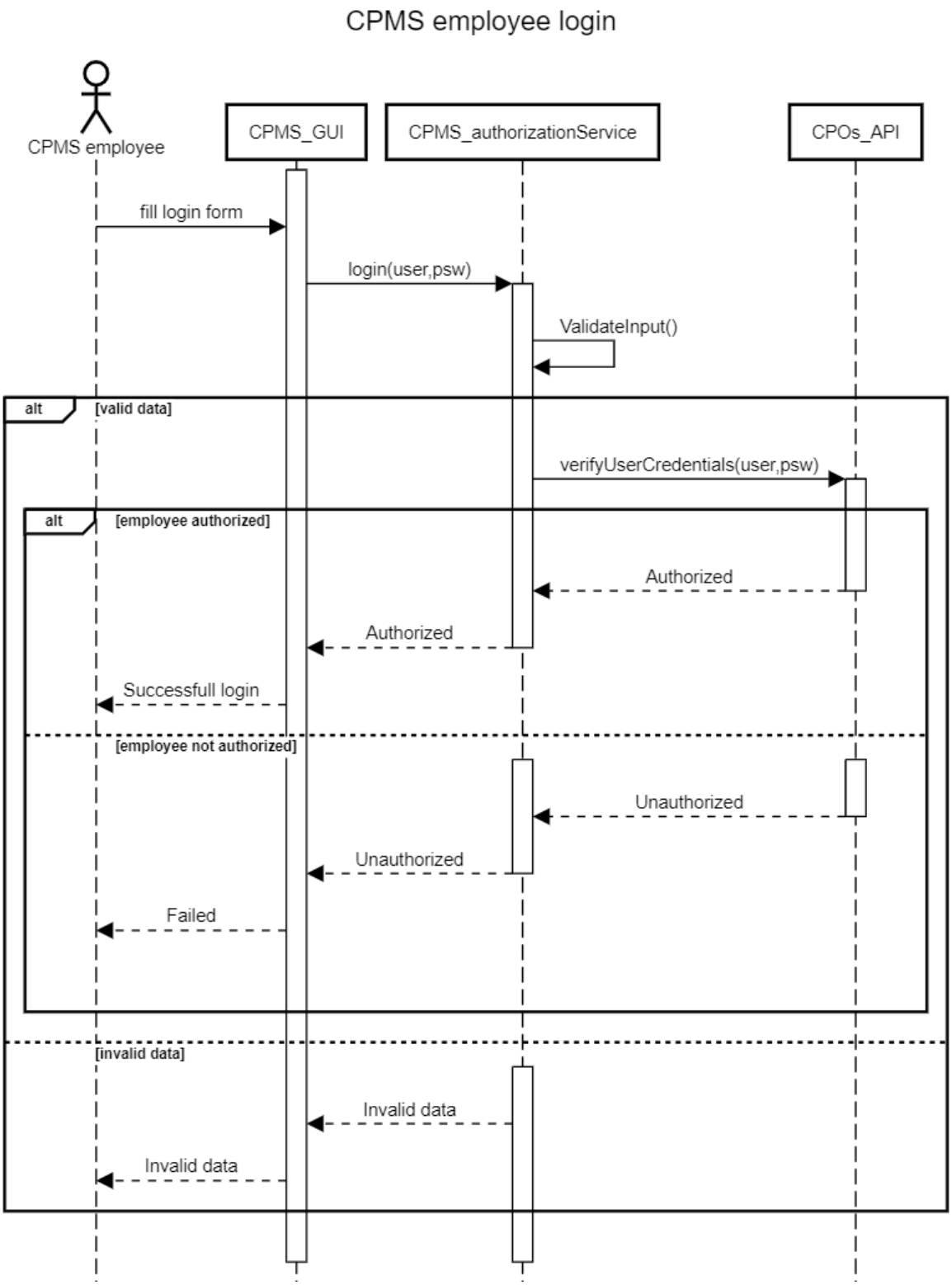
computers and devices on a network. It is typically equipped with a powerful processor, a large amount of memory and storage, and specialized software and hardware to support its role as a server. It is connect to a load balancer and a database, and eventually to a computer. The serverOS is a type of software designed specifically to run and manage the hardware and software resources of a server computer. These operating systems are optimized for running servers and are typically more reliable, stable, and secure than desktop operating systems. The eMSP server application and the CPMS server application run on the server OS, in two distinct physical servers. In order to increase the system reliability, two parallel physical servers will be used for both CPMS and eMSP systems.

- **database**: Hosts the database of the system. There are two main databases running in the eMall system, a eMSP one that will be used to save and manage user accounts and one of the CPMS that contains all past and future reservations for each charging station. Both databases make use of the mySQL DBMS to handle the structure and modification of the contained data.
- **CPMS terminal:** is a computer that has a operating system installed on it that manages the hardware resources of the computer and provides a platform for other software programs to run on. The OS is responsible for controlling the computer's processor, memory, and other hardware components, as well as providing a user interface for interacting with the computer. The CPMS remote session manger is a software application or service that allows users to remotely access and control the CPMS server over a network connection.

2.4 Runtime view

This section includes sequence diagrams that show how the different components of the system work together to perform the main functions. Except for the login and registration processes of both eMSP and CPMS the user is assumed to be already logged in. Also, since all activity calls should be performed by the OS they were simplified by just having activities call other activities. For all systems outgoing and incoming request are handled by routers which have the role of dispatching requests to the correct service, thus these where simplified to just requests reaching the correct component of other systems.

2.4.1 CPMS Employee Login Process

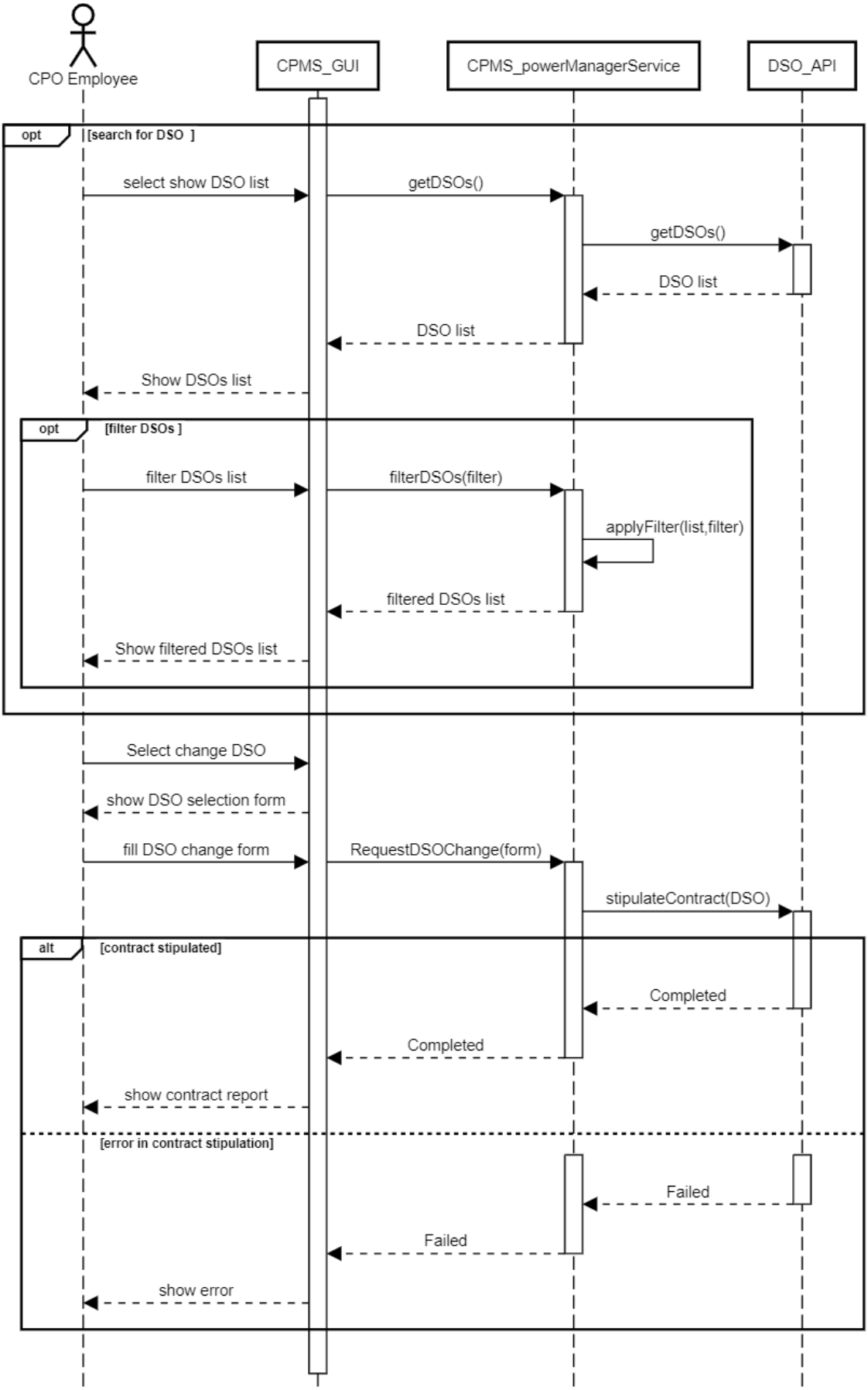


In this component diagram, the process of a CPMS employee logging in to the system is described. The process begins when the employee interacts with the CPMS_GUI by filling out a login form. The CPMS_GUI then sends the login information to the CPMS_authorizationService, which checks the validity of the data. If the data is valid, the CPMS_authorizationService sends a request to the

CPOs_API to verify the employee's login credentials. If the employee is authorized, the CPOs_API sends an "Authorized" message to the CPMS_authorizationService, which in turn sends a message to the CPMS_GUI indicating a successful login. The CPMS_GUI then communicates this success to the employee. If the employee is not authorized, the CPOs_API sends an "Unauthorized" message to the CPMS_authorizationService, which sends an "Unauthorized" message to the CPMS_GUI and the employee. If the original login data was invalid, the CPMS_authorizationService sends an "Invalid data" message to the CPMS_GUI and the employee.

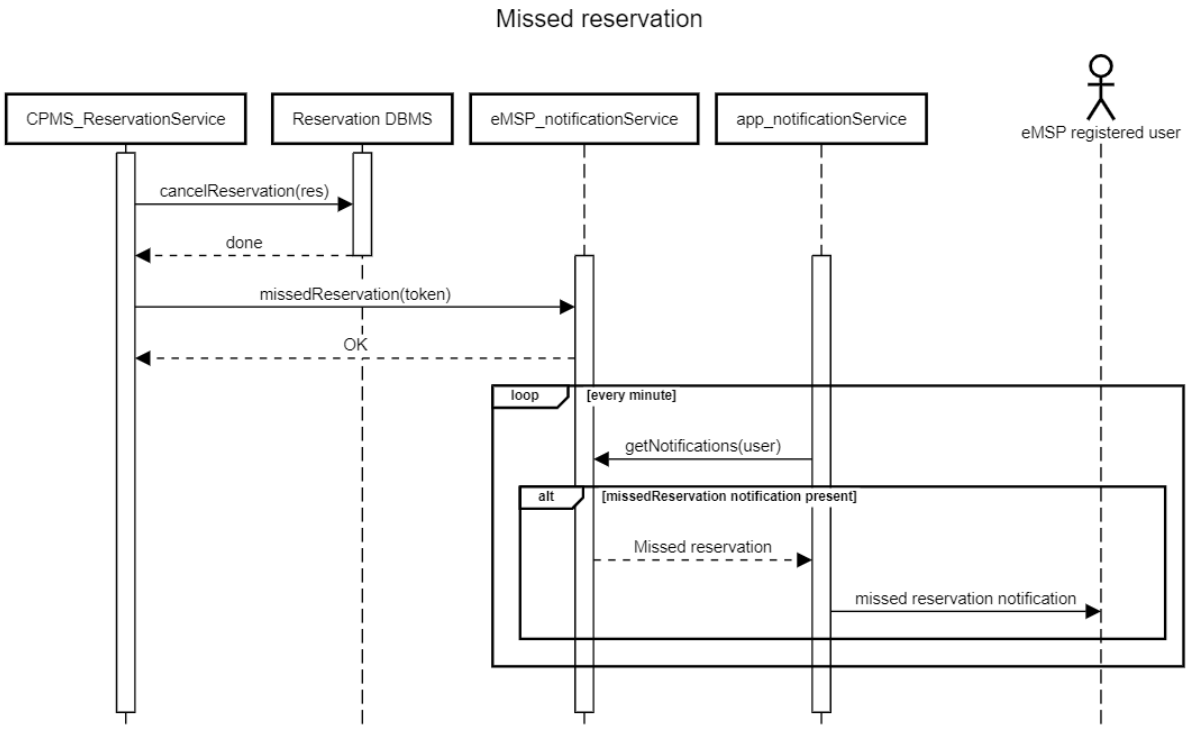
2.4.2 Change DSO Manually

Change DSO manually



In this component diagram, the process of a CPO employee manually changing the DSO for the power management system is described. The process begins when the CPO employee selects the option to show the list of DSOs in the CPMS_GUI. The CPMS_GUI sends a request to the CPMS_powerManagerService to retrieve the list of DSOs, which in turn sends a request to the DSO_API to retrieve the list. The DSO_API sends the list of DSOs back to the CPMS_powerManagerService, which sends the list to the CPMS_GUI. The CPMS_GUI displays the list of DSOs to the CPO employee. The CPO employee has the option to filter the list of DSOs, in which case the CPMS_GUI sends a request to the CPMS_powerManagerService to apply the filter to the list. The CPMS_powerManagerService applies the filter and returns the filtered list to the CPMS_GUI, which displays the list to the CPO employee. When the CPO employee selects to change the DSO, the CPMS_GUI displays a form for the CPO employee to fill out. Once the form is filled out, the CPMS_GUI sends a request to the CPMS_powerManagerService to initiate the DSO change, providing the form as input. The CPMS_powerManagerService sends a request to the DSO_API to stipulate the contract for the new DSO. If the contract is successfully stipulated, the DSO_API sends a message indicating completion to the CPMS_powerManagerService, which sends a message to the CPMS_GUI and the CPO employee indicating completion. If there is an error in stipulating the contract, the DSO_API sends a message indicating failure to the CPMS_powerManagerService, which sends a message to the CPMS_GUI and the CPO employee indicating failure.

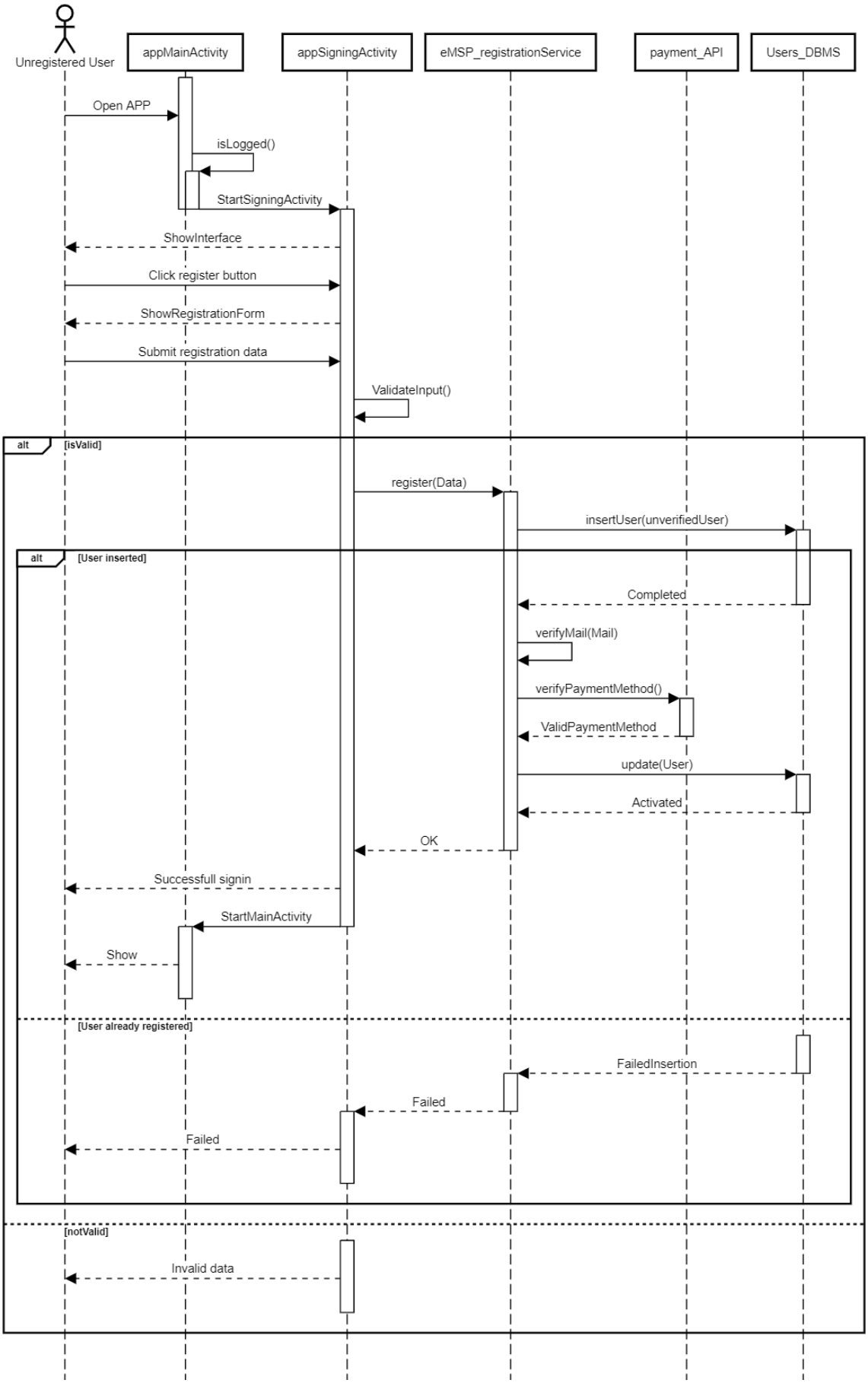
2.4.3 Missed reservation



In this component diagram, the process of handling a missed reservation is described. The process involves the CPMS_ReservationService interacting with the Reservation DBMS to cancel the reservation, and the eMSP_notificationService and app_notificationService interacting to send a notification about the missed reservation to the eMSP registered user. The process begins when the CPMS_ReservationService sends a request to the Reservation DBMS to cancel the reservation. The Reservation DBMS cancels the reservation and sends a message indicating completion to the CPMS_ReservationService. The CPMS_ReservationService then sends a notification about the missed reservation to the eMSP_notificationService, which sends a confirmation message. The app_notificationService checks for notifications every minute and, if a notification about the missed reservation is present, sends a notification to the eMSP registered user.

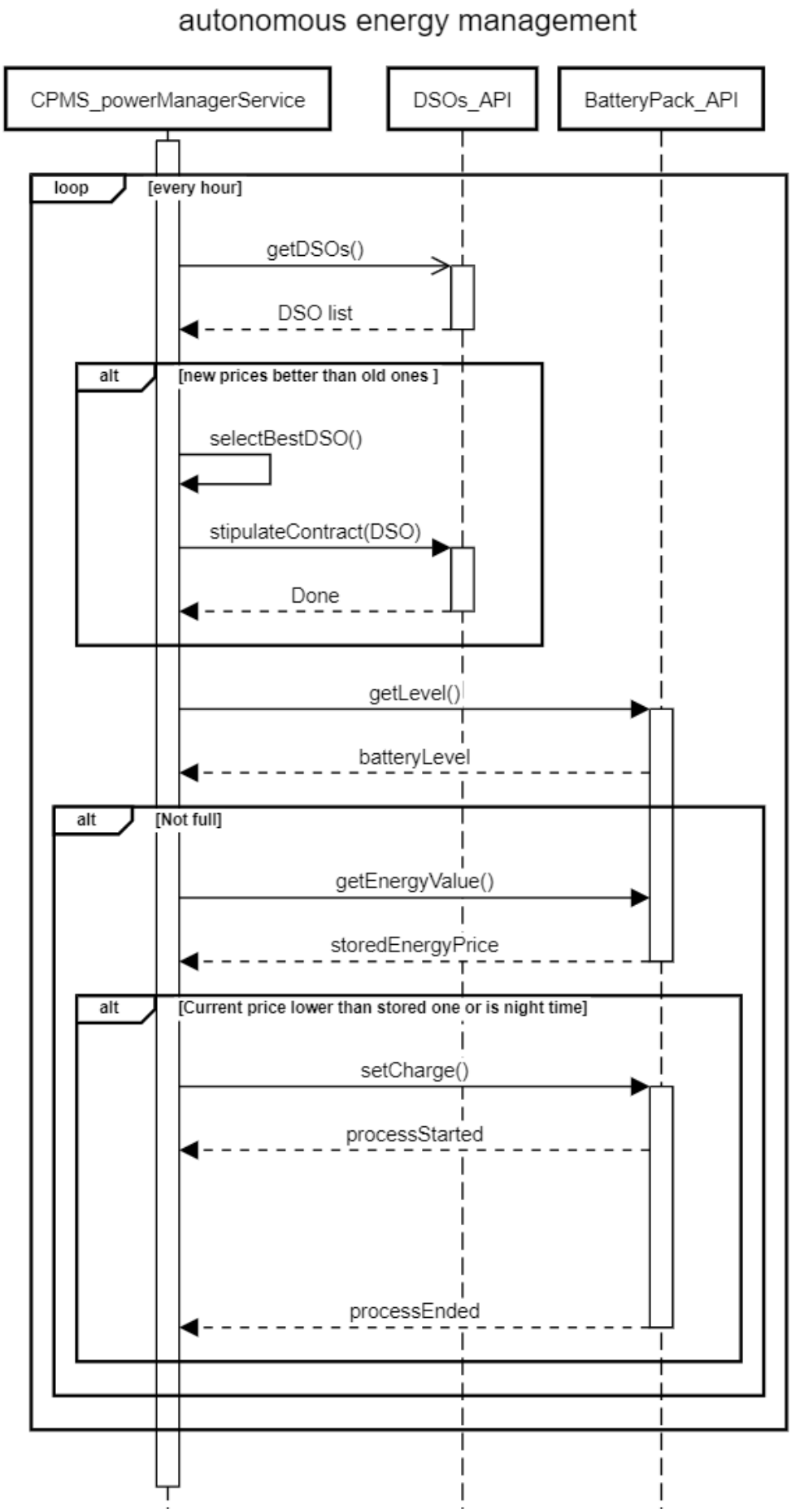
2.4.4 User registration

User registration



In this component diagram, the process of a user registering for an app is described. The process involves the Unregistered User interacting with the appMainActivity and appSigningActivity, and the eMSP_registrationService, Users_DBMS, and payment_API collaborating to handle the registration. The process begins when the Unregistered User opens the app and the appMainActivity determines whether the user is logged in. If the user is not logged in, the appMainActivity starts the appSigningActivity and displays its interface to the user. The user clicks the register button and submits their registration data to the appSigningActivity, which checks if the input is valid. If the input is valid, the appSigningActivity sends a request to the eMSP_registrationService to register the user. The eMSP_registrationService inserts the user's information into the Users_DBMS and, if successful, verifies the user's email and payment method. If the payment method is valid, the eMSP_registrationService updates the user's status in the Users_DBMS to "activated" and sends a message indicating success to the appSigningActivity. The appSigningActivity then starts the appMainActivity and displays it to the user. If the input is invalid or the user is already registered, the appSigningActivity sends a message indicating failure to the user.

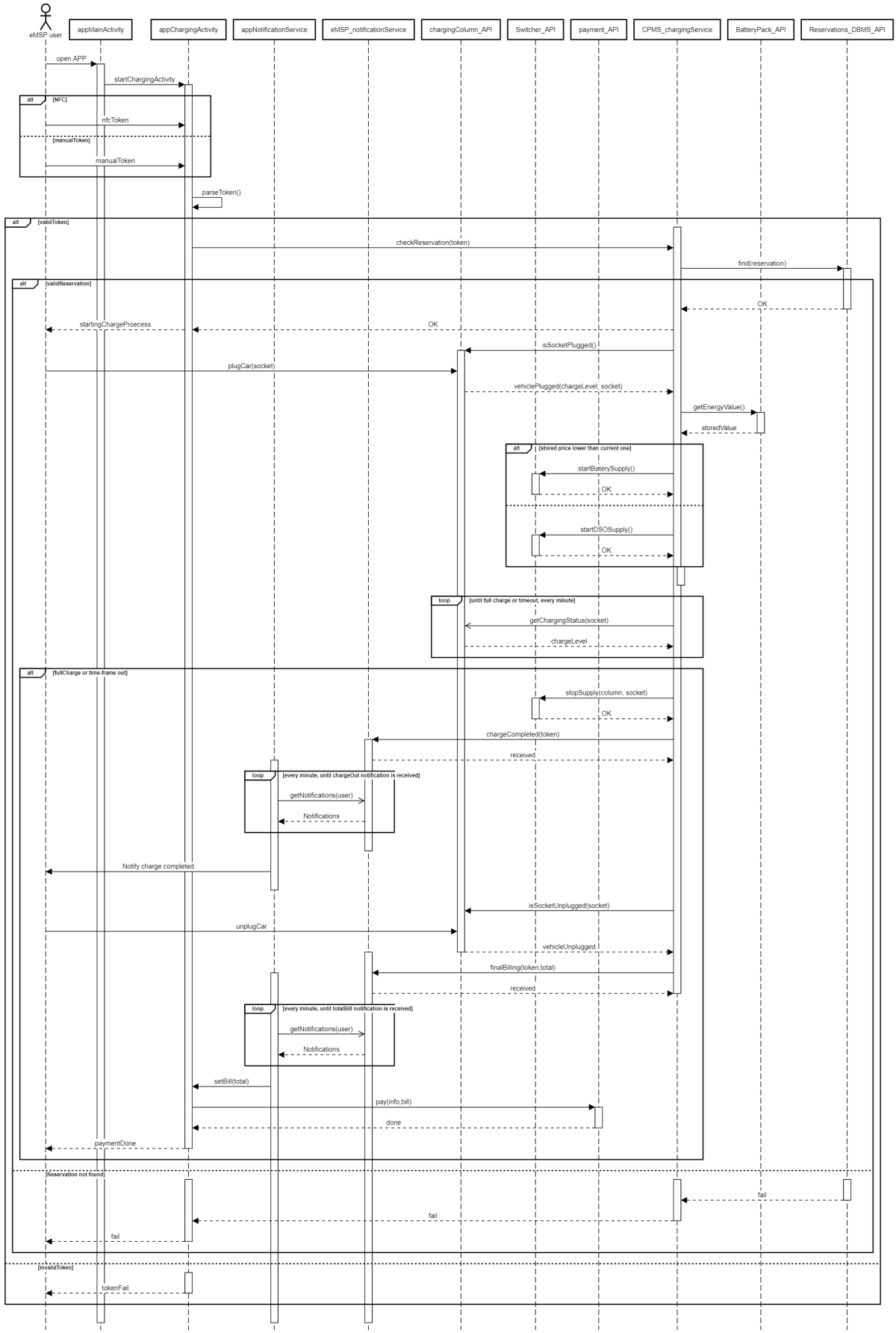
2.4.5 Autonomous energy management


autonomous energy management
CPMS_powerManagerService
DSOs_API
BatteryPack_API
loop
[every hour]
getDSOs()
DSO list
alt
[new prices better than old ones]
selectBestDSO()
stipulateContract(DSO)
Done
getLevel()
batteryLevel
alt
[Not full]
getEnergyValue()
storedEnergyPrice
alt
[Current price lower than stored one or is night time]
setCharge()
processStarted
processEnded

The CPMS_powerManagerService in this component diagram is responsible for autonomously managing energy by retrieving a list of DSOs and selecting the best one based on price. If the battery is not full, the CPMS_powerManagerService checks the current price and, if it is lower than the stored price or it is night time, tells the BatteryPack_API to start charging. This process is repeated on a loop every hour.

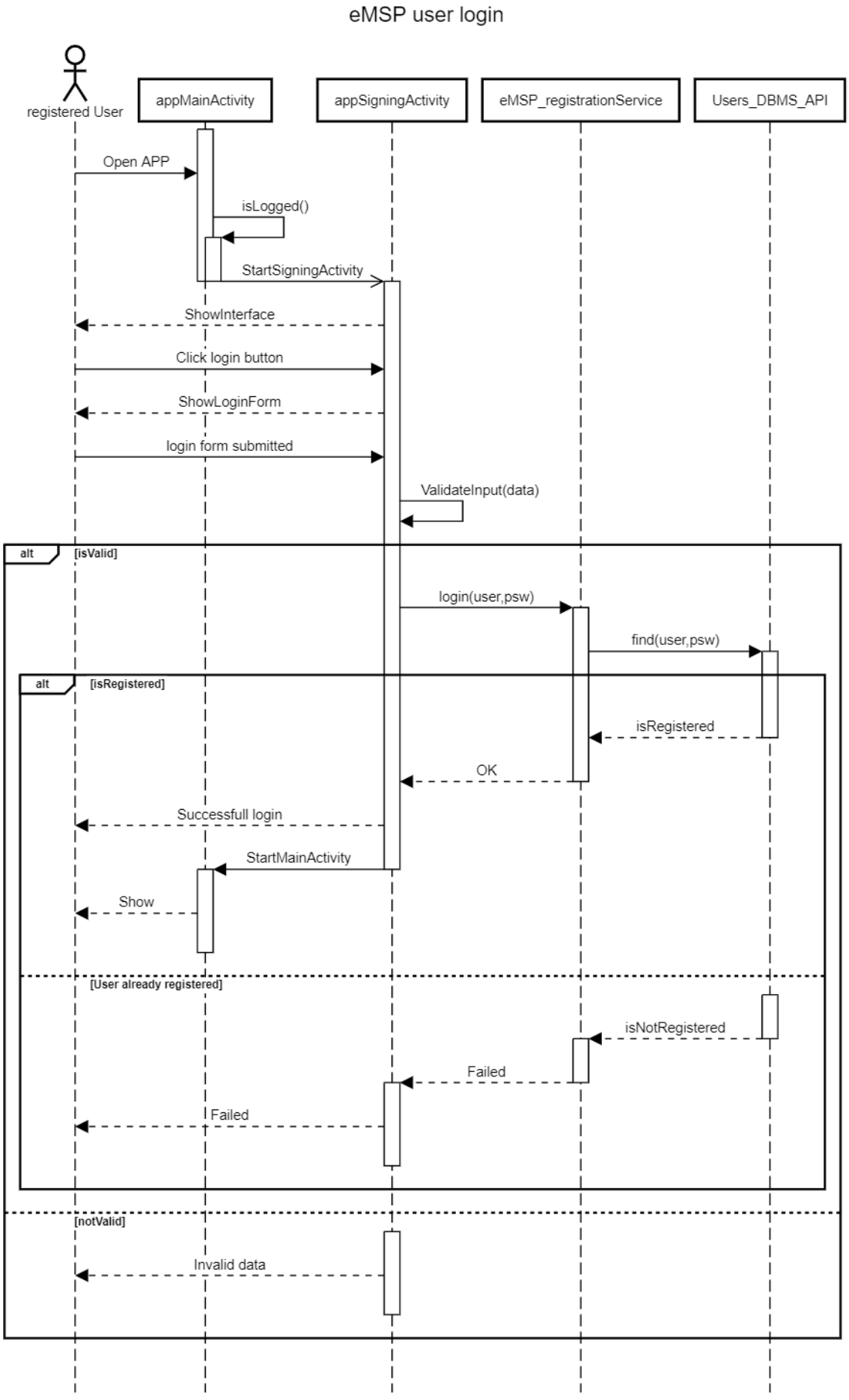
2.4.6 Charging process

charging process



In this component diagram, an eMSP user opens the app and starts the charging process through the appChargingActivity. The appChargingActivity verifies the validity of the NFC or manual token provided by the user and communicates with the CPMS_chargingService to check for a valid reservation. If the reservation is valid, the CPMS_chargingService checks whether the stored energy price is lower than the current price of the associated DSO. If it is, the car is charged using battery suppl, if not, the car is charged using a DSO. The charging process continues until the car is fully charged or a time-frame has been reached. The eMSP user is notified when the charge is completed and is prompted to unplug the car. The chargingColumn_API verifies that the car has been unplugged and the CPMS_chargingService sends a final billing notification to the eMSP_notificationService. The appNotificationService receives the notification and alerts the eMSP user of the final price paid.

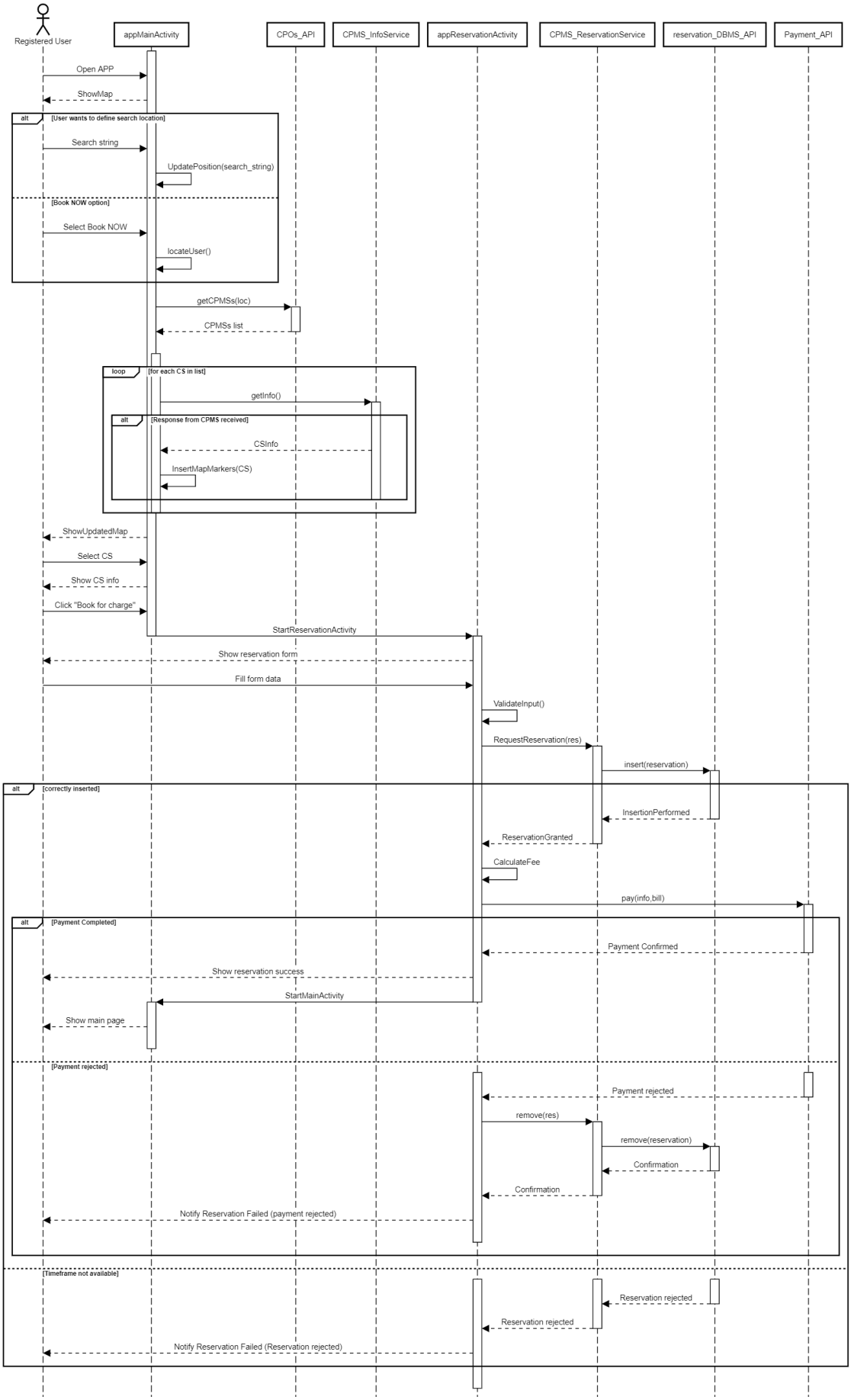
2.5.7 eMSP user login


eMSP user login
registered User
appMainActivity
appSigningActivity
eMSP_registrationService
Users_DBMS_API
Open APP
isLogged()
StartSigningActivity
ShowInterface
Click login button
ShowLoginForm
login form submitted
ValidateInput(data)
alt
[isValid]
login(user,psw)
find(user,psw)
alt
[isRegistered]
isRegistered
OK
Successfull login
StartMainActivity
Show
[User already registered]
isNotRegistered
Failed
Failed
[notValid]
Invalid data

In this component diagram, a registered user is attempting to log in to an app using the app's main activity and signing activity. The user submits their login information to the app signing activity, which checks if the input is valid. If the input is valid, the app signing activity sends a request to the eMSP registration service to log in the user. The eMSP registration service checks if the user is registered in the Users DBMS API. If the user is registered, the login is successful and the app's main activity is started. If the user is not registered or the input is invalid, the login is unsuccessful and an error message is displayed to the user.

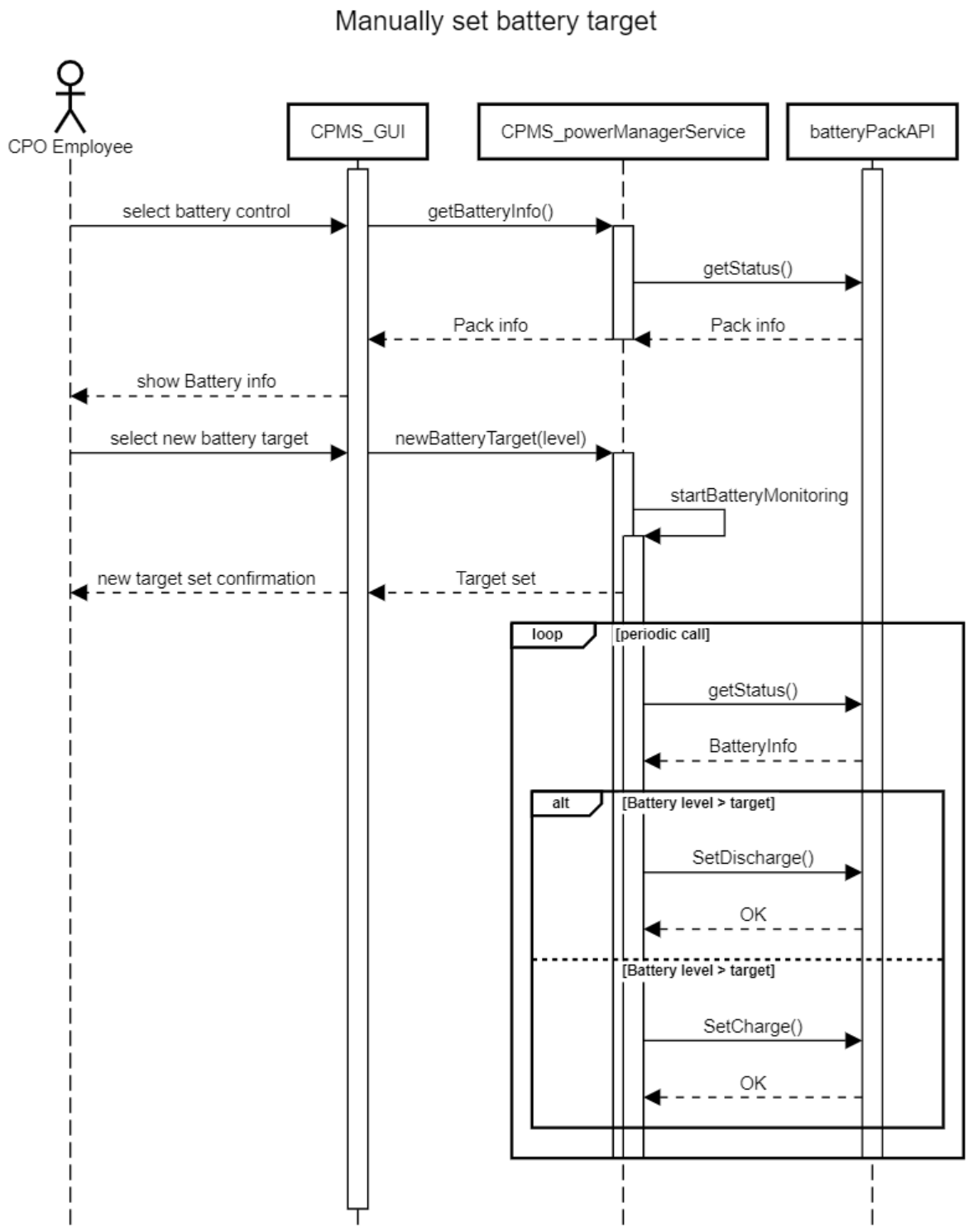
2.5.8 Make a reservation

Making a reservation



The component diagram describes the process of making a reservation through an app. The user can either search for a location to make a reservation at or select the "Book NOW" option to automatically search for locations near the user. The app will then retrieve a list of available charging stations (CPMSs) and display them on a map for the user. The user can select a charging station and fill out a reservation form. The form will be validated and the reservation request will be sent to the reservation service. If the reservation is successful, the user will be asked to make a payment. If the payment is successful, the reservation is confirmed and the user is shown a success notification. If the payment is not successful or the reservation is not available, the user will be notified of the failure and the reservation process will end.

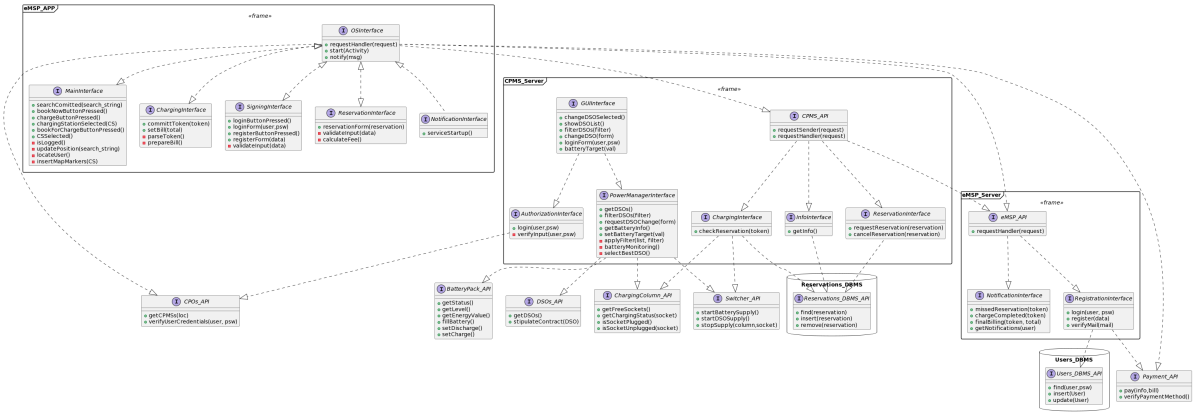
2.5.9 Change battery target



In this component diagram, a CPO Employee uses the CPMS_GUI to select battery control and view battery information. The CPMS_GUI communicates with the CPMS_powerManagerService to retrieve this information, which is then obtained from the batteryPackAPI. The CPO Employee can then select a new battery target in the CPMS_GUI, which is passed on to the CPMS_powerManagerService. The CPMS_powerManagerService starts monitoring the battery level and periodically

checks the battery level against the target. If the battery level is above the target, the CPMS_powerManagerService tells the batteryPackAPI to set the discharge. If the battery level is below the target, the CPMS_powerManagerService tells the batteryPackAPI to set the charge.

2.5 Component interfaces



2.6 Selected architectural styles and patterns

The system is divided into three layers: a presentation layer, a business logic layer, and a data layer. This architecture was chosen to increase the indipendence of the various sections and to make it easier to scale the system. By separating the data and business layers, it is possible to modify or add resources to one layer without affecting the other two.

2.7 Other design decisions

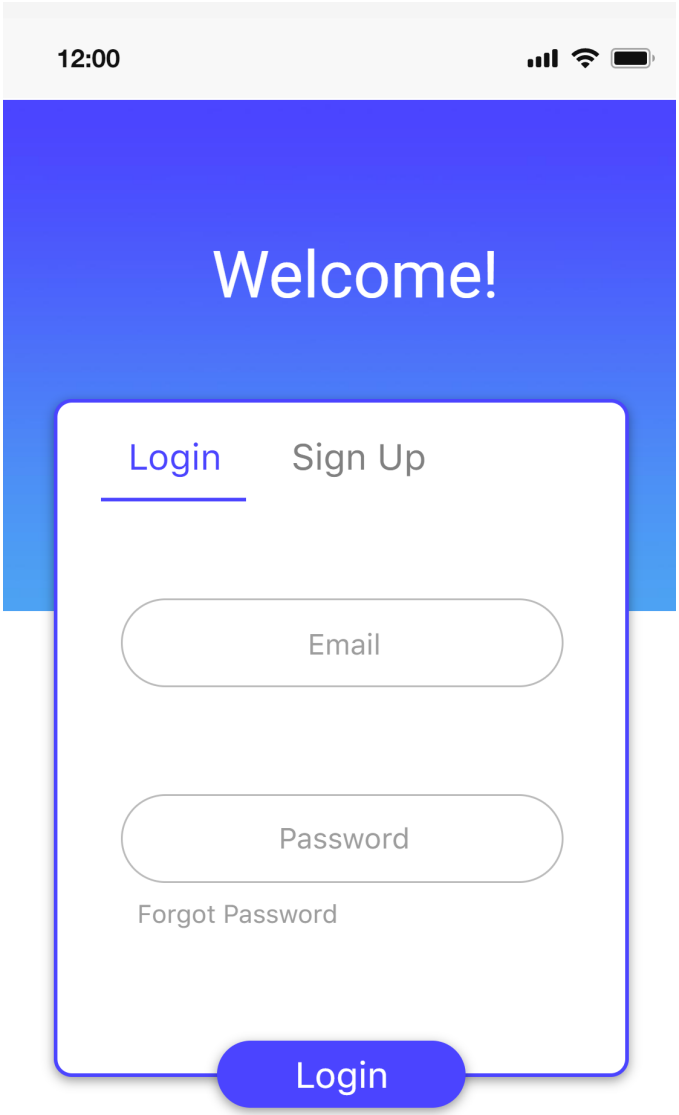
The system is designed to retrieve data from external APIs as needed, rather than continuously fetching data on a schedule. This approach allows for greater flexibility by allowing new data sources to be easily incorporated by updating the business logic layer. However, it is important to note that this design also makes the system more reliant on external services for full functionality. To mitigate this vulnerability, it may be beneficial to store the results of computations, especially those that involve combining and analyzing data from multiple sources. API requests that need to be up to date, such as CPMS prices and free timeframes, are instead performed based on the user necessity to view updated informations.

3 User interface design

In this section mockups of the user interface are presented. For the eMSP system there are only the screens regarding the usage of the app while for the CPMS all interfaces that the CPO employee uses are reported.

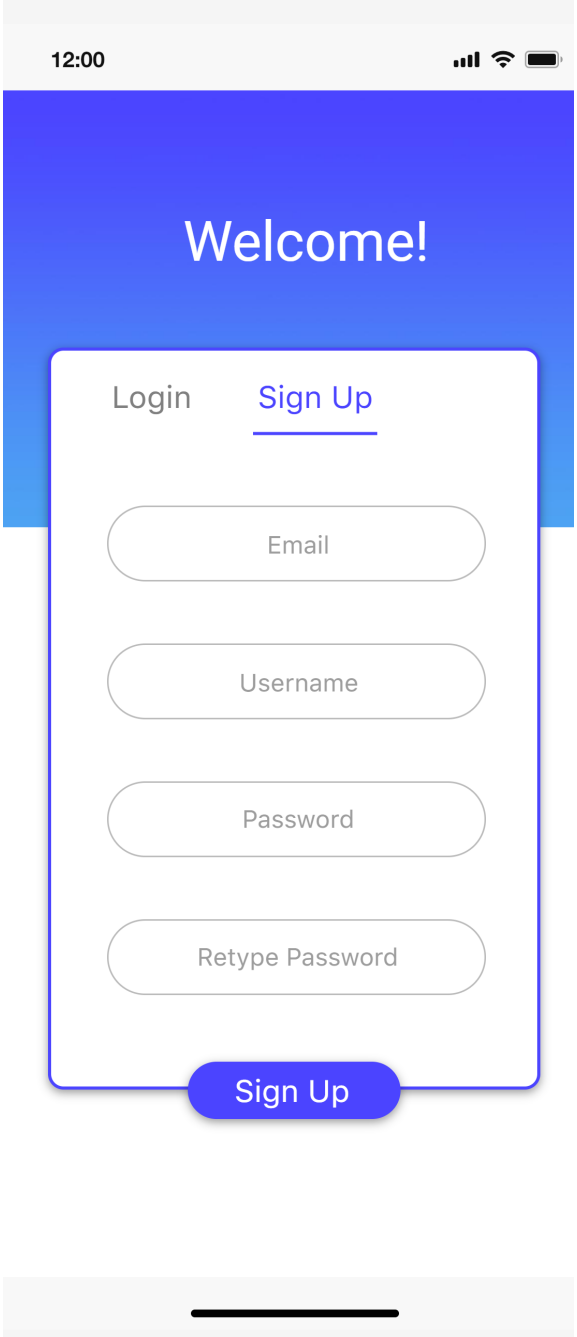
3.1 eMSP app user

3.1.1 Login

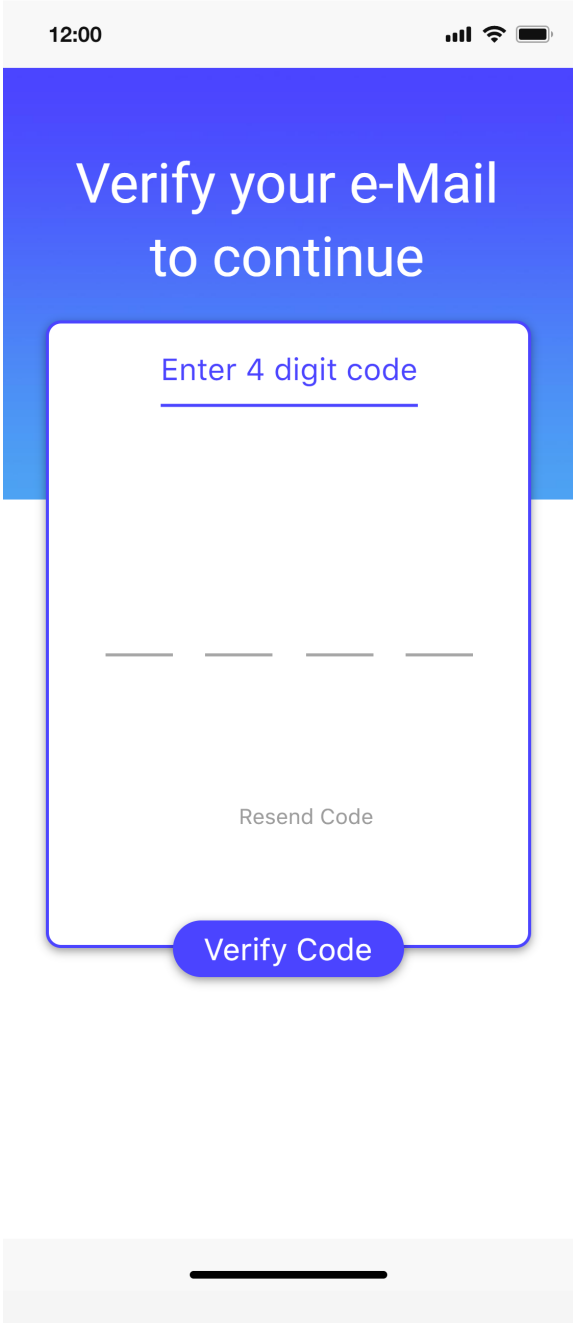


Login form

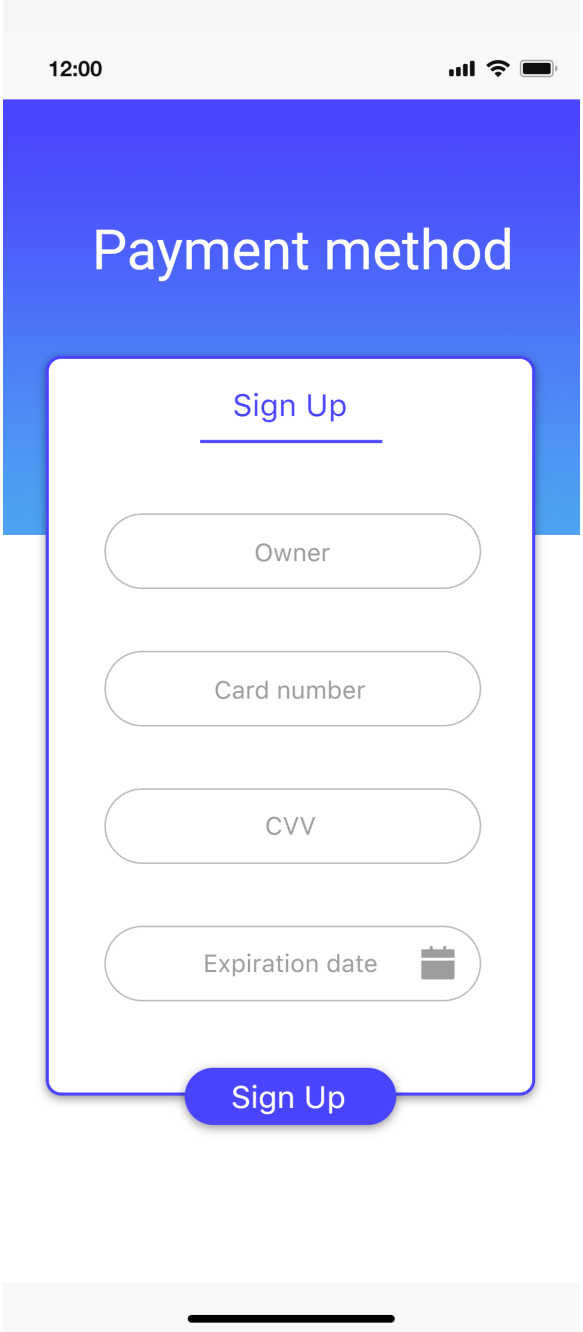
3.1.2 Registration



Registration form

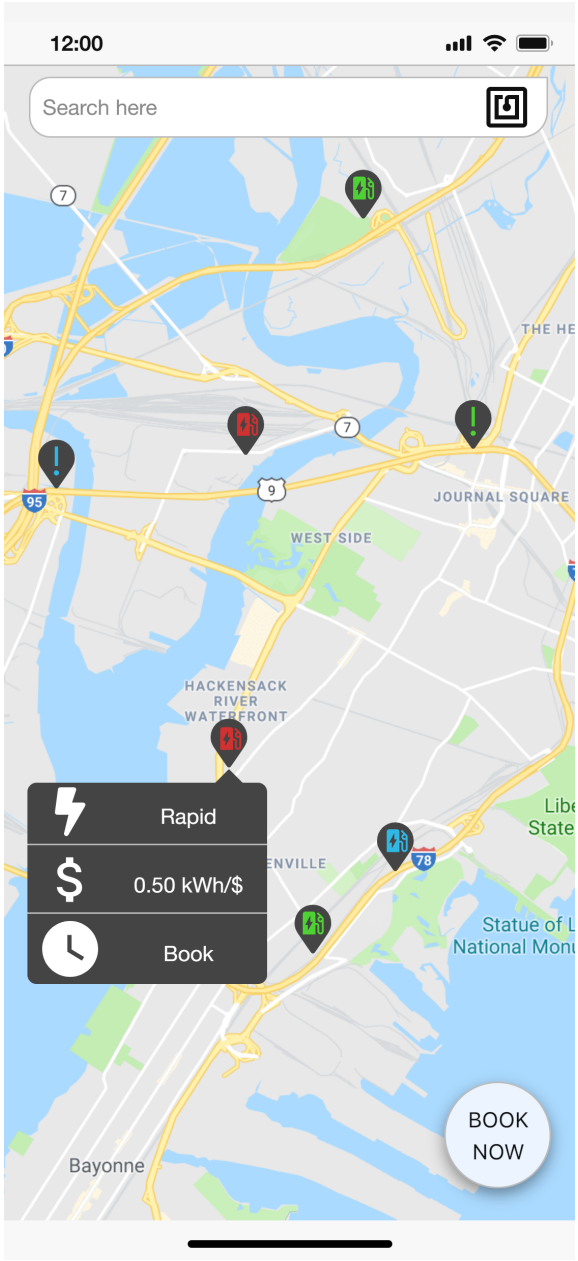


Code insertion for email verification

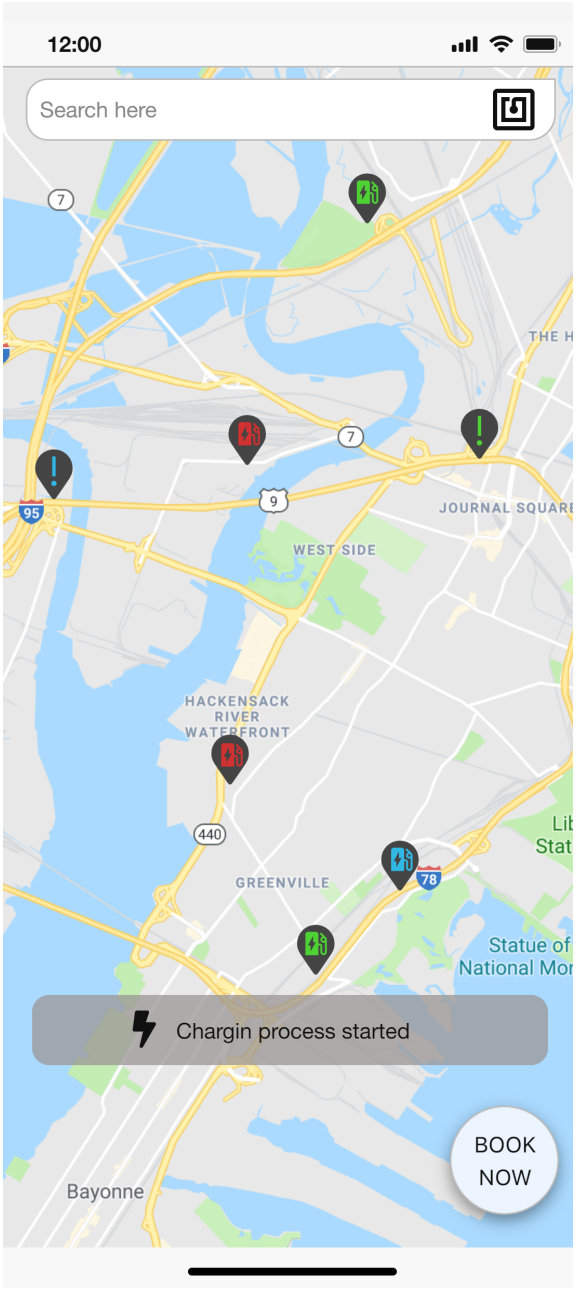


Payment method form

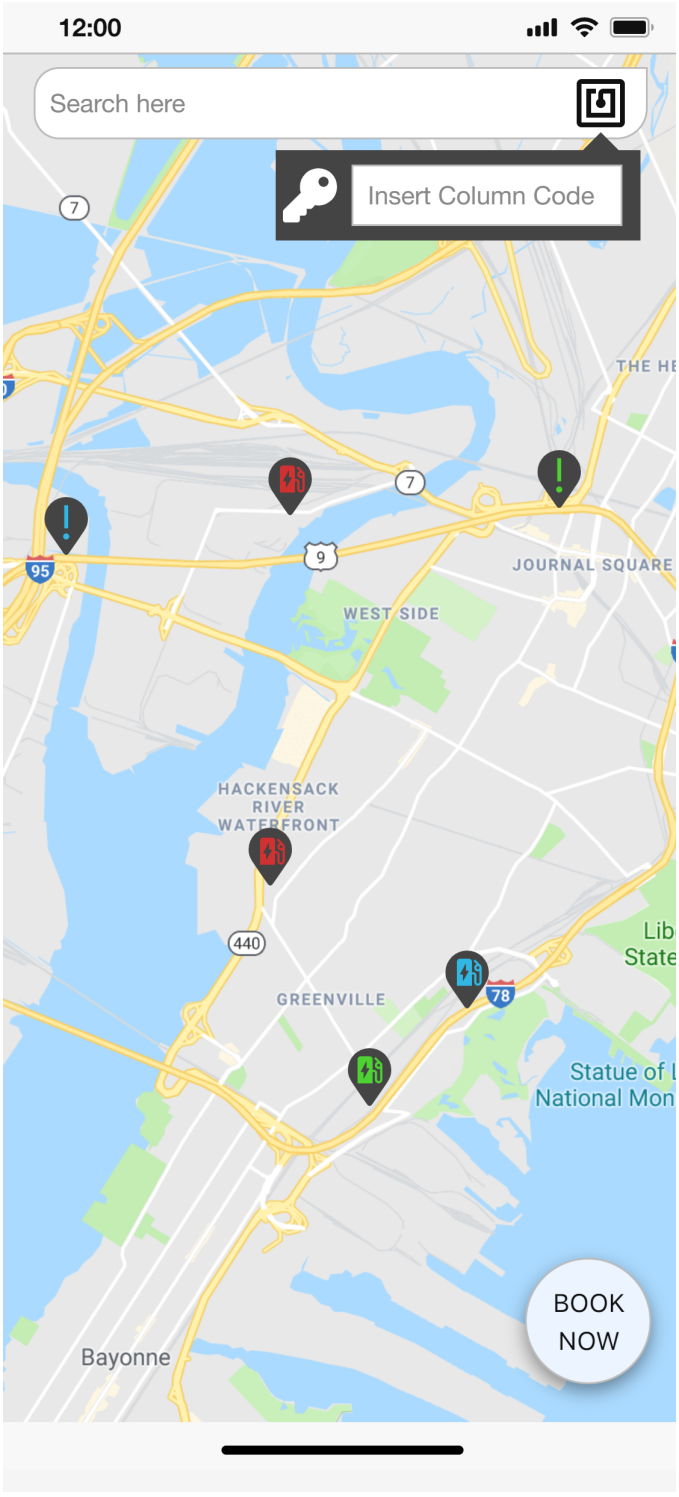
3.1.3 Main interface



Main interface map with markers

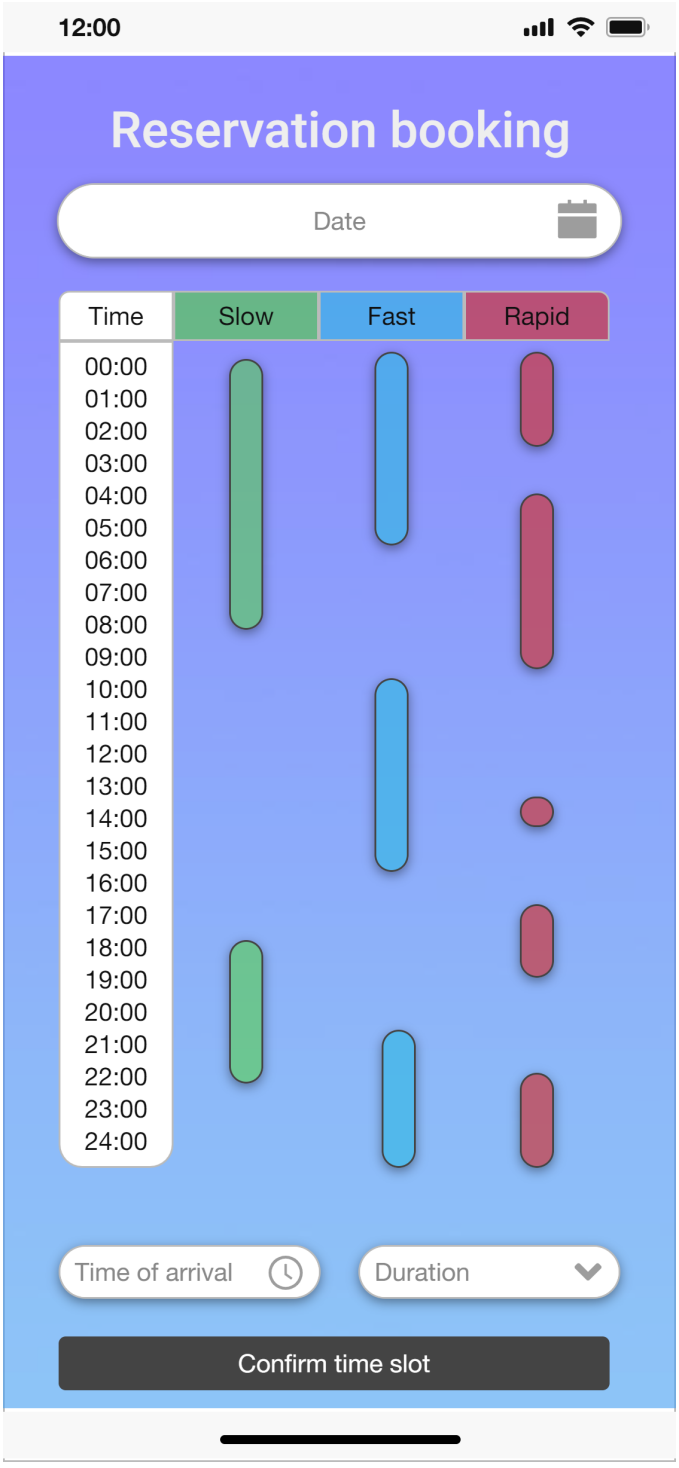


Main interface with lable of correct NFC recognition



Manual column token insertion

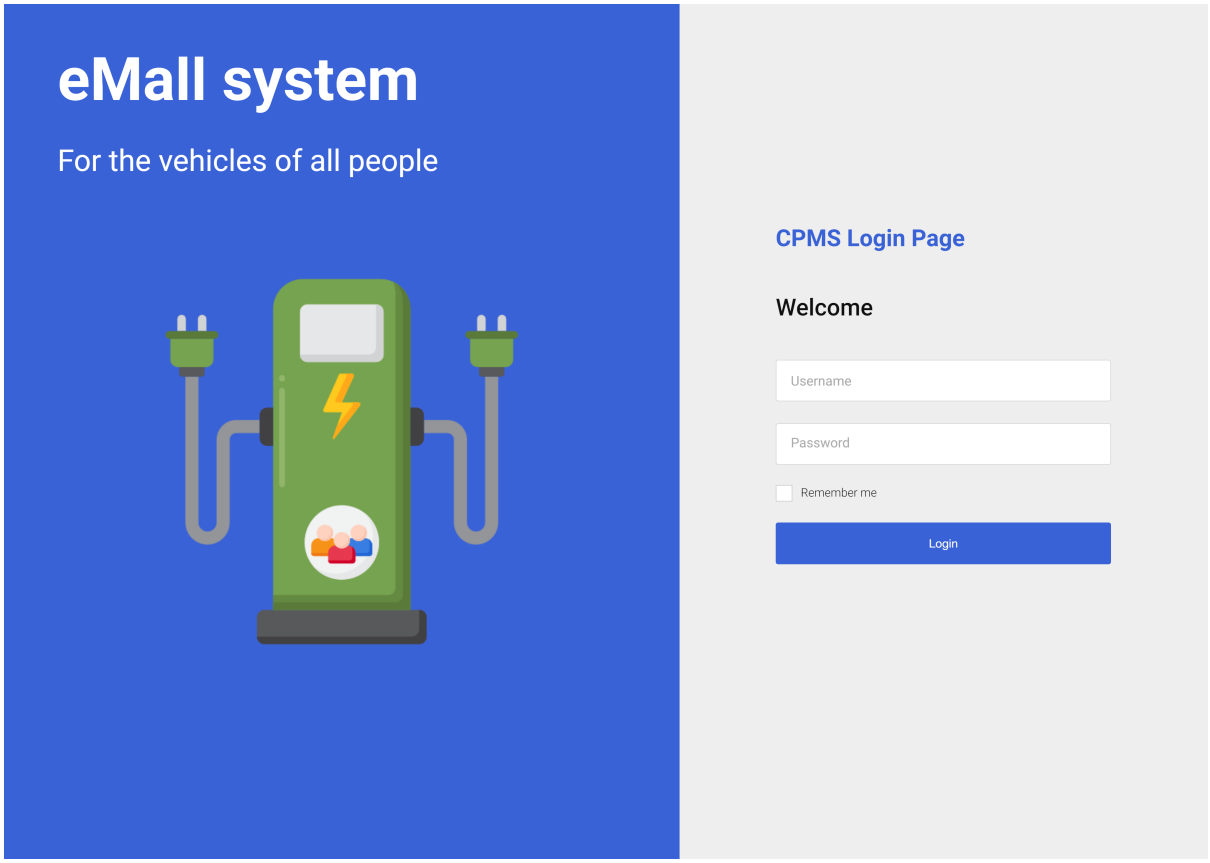
3.1.4 Booking form



Form for the selection of a reservation

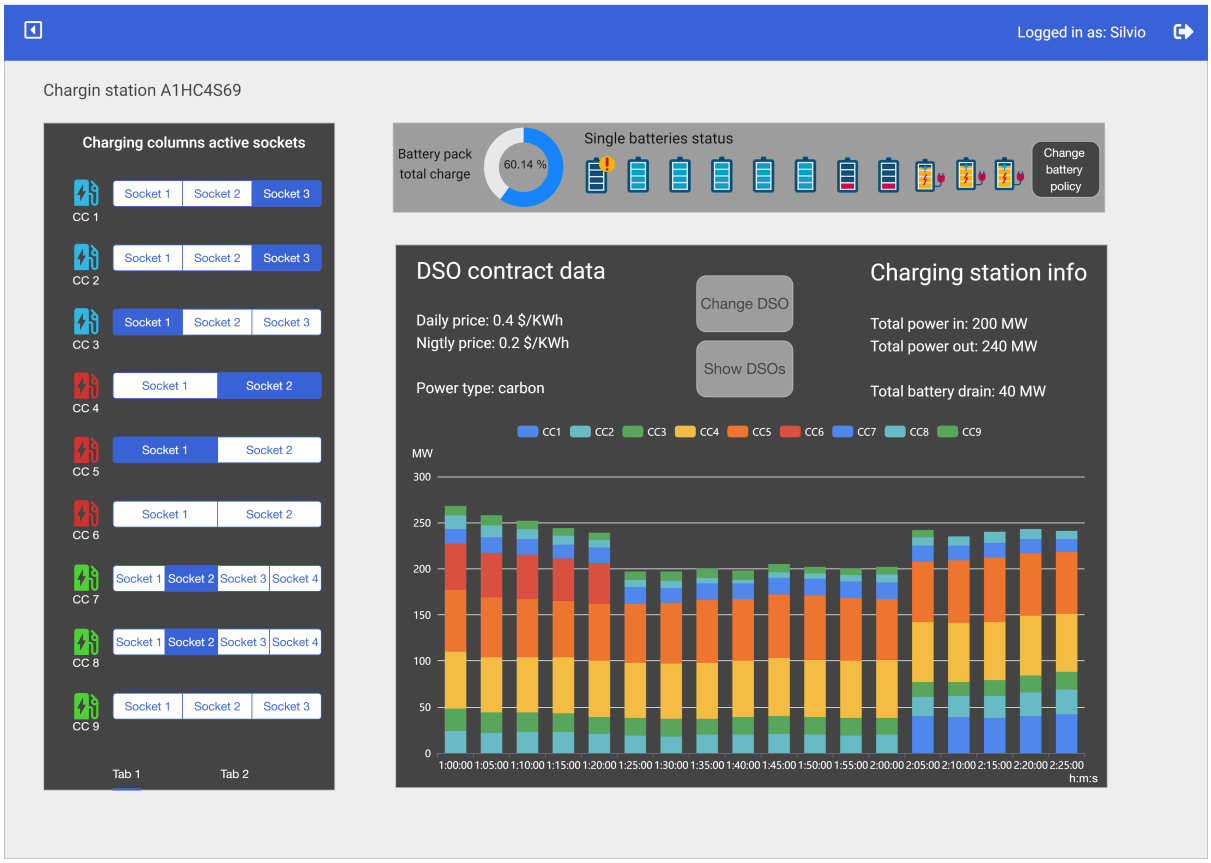
3.2 CPMS interface

3.2.1 Login



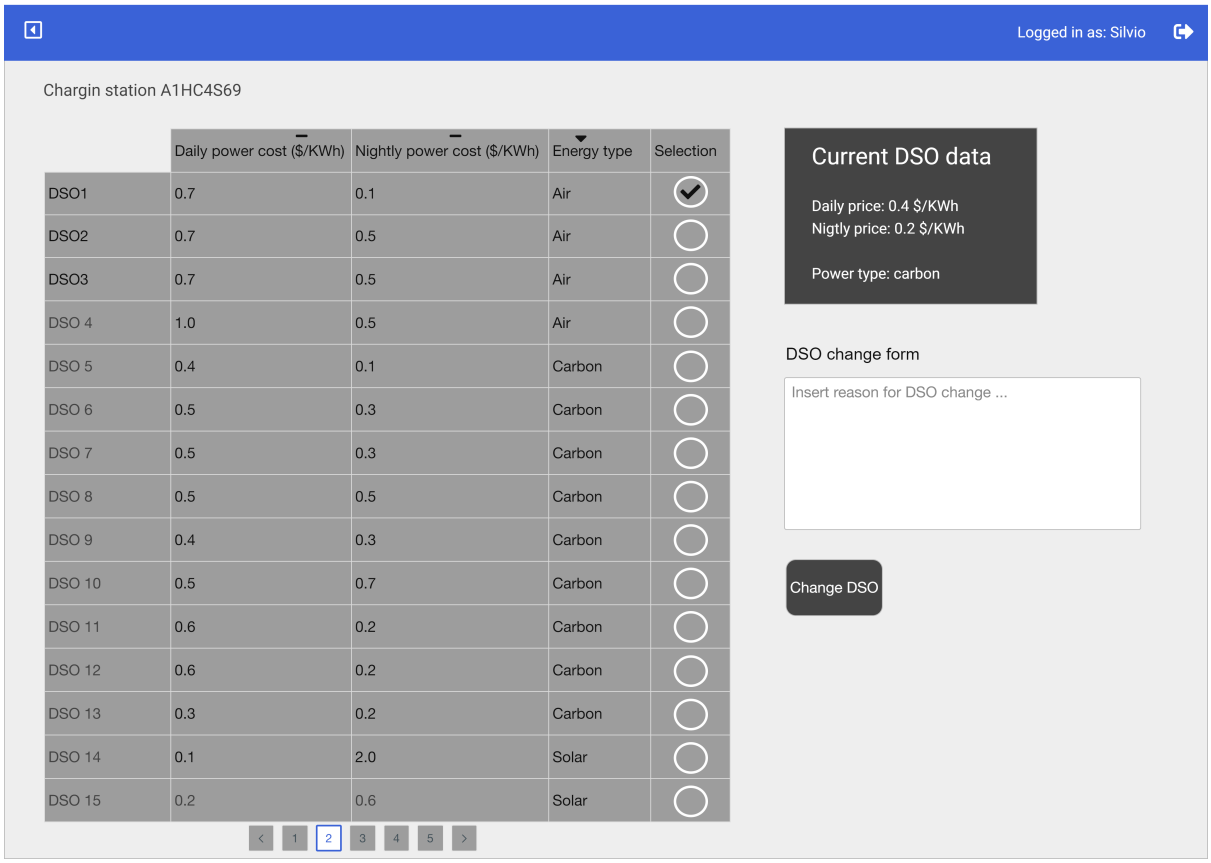
Login interface for CPO employees

3.2.2 Main interface



Charging station status interface for CPO employees

3.2.3 DSO change form



DSO change form and visualizer for CPO employees

4 Requirments traceability

4.1 Requiremets definition

Funqtional requirements are here reported:

Requirements	Description
R1	eMSP shall allow users to sign up to the eMSP service
R2	eMSP shall allow registered users to sign in
R3	eMSP shall allow users to link a payment method to his account
R4	eMSP must verify users payment methods

Requirements	Description
R5	eMSP must verify users email upon registration
R6	eMSP shall allow registered users to make a reservation to a specific CS at a specific date
R7	eMSP must show registered users correct positional data and nearby charging stations
R8	eMSP shall allow registered users to select an area in which to search for charging stations
R9	eMSP must be able to order payments from given payment methods
R10	eMSP must notify registered users of missed reservations
R11	eMSP must notify registered users of the final billing for the received service
R12	eMSP must notify registered users when the booked time-frame is up
R13	eMSP must notify the user when the CPMS sends a charge complete response
R14	eMSP must notify CPMS of code read through NFC
R15	eMSP must notify CPMS of the one-time-token inserted through the app
R16	CPMS must be able to allow CPO employees to manually decide how to manage battery and net power
R17	CPMS must be able to allow CPO employees to manually decide which DSO to buy energy from
R18	CPMS must be able to automatically decide how to manage its own battery and net power
R19	CPMS must be able to automatically decide which DSO buy energy from
R20	CPMS must be able to answer eMSP requests for free timeframes and relative prices
R21	CPMS must allow a user to charge it’s vehicle at a specific time-frame if and only if that specific time-frame has been reserved by that user
R22	CPMS must be able to manage reservations through a DBMS
R23	CPMS must be able to authorize users with reserved time-frames
R24	CPMS must be able to interact with all APIs that manage charging stations components
R25	CPMS must notify eMSP when the vehicle of a user is fully charged
R26	CPMS must be able to notify user of exceptions
R27	CPMS must be able to notify user of successful actions

Requirements	Description
R28	eMSP must be able to notify user of exceptions
R29	eMSP must be able to notify user of successful actions
R30	CPMS bust be able to notify eMSP of exceptions
R31	CPMS bust be able to notify eMSP of successful actions
R32	eMSP must be able to retrieve charging stations locations from a CPO list
R33	CPMS must notify eMSP of final power bill
R34	CPMS shall allow CPO employee with corporate credential to sign in
R35	CPMS must be able to show CPO employees a filtered and sortable list of DSOs

4.2 Requirements-component mapping

This section contains a table explaining what components are required in order to fulfill each of the requirements specified in the RASD. To save some space, the components have been given abbreviations, see list below. Please note that all requirements that are fulfilled completely or partially through the eMSP app involves the usage of the OS component.

eMSP_r	eMSP_registrationService
eMSP_n	eMSP_notificationService
eMSP_t	eMSP_router
usr_DBMS	users_DBMS
appM	appMainActivity
appS	appSigningActivity
appC	appChargingActivity
appR	appReservationActivity
appN	appNotificationActivity
CPMS_t	CPMS_router
CPMS_p	CPMS_powerManagerService
CPMS_a	CPMS_authorizationService
CPMS_G	CPMS_GUI
CPMS_c	CPMS_chargingService
CPMS_i	CPMS_infoService

CPMS_r	CPMS_reservationService
res_DBMS	reservations_DBMS

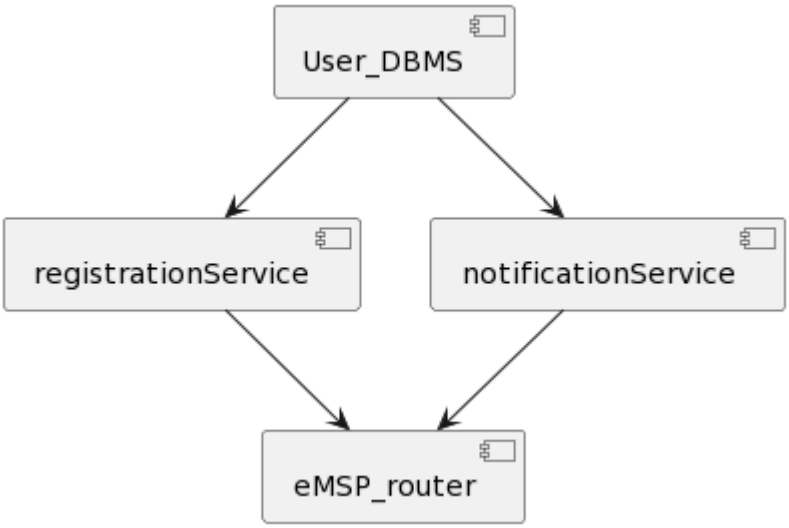
	eMSP_r	eMSP_n	eMSP_t	usr_DBMS	appM	appS	appC	appR	appN	CPMS_t	CPMS_p	CPMS_a	CPMS_G	CPMS_c	CPMS_i	CPMS_r	res_DBMS
R1	✓		✓	✓	✓	✓											
R2	✓		✓	✓	✓	✓											
R3	✓		✓	✓	✓	✓											
R4	✓		✓	✓													
R5	✓		✓	✓													
R6					✓			✓		✓						✓	✓
R7					✓					✓					✓		✓
R8					✓												
R9							✓	✓									
R10		✓	✓						✓	✓						✓	✓
R11		✓	✓						✓	✓				✓			
R12		✓	✓						✓	✓				✓			
R13		✓	✓						✓	✓				✓			
R14							✓			✓				✓			
R15							✓			✓				✓			
R16											✓		✓				
R17											✓		✓				
R18											✓						
R19											✓						
R20										✓					✓		✓
R21														✓			✓
R22																✓	✓
R23														✓			✓
R24											✓			✓			
R25		✓	✓							✓				✓			
R26		✓	✓						✓	✓	✓	✓	✓	✓	✓	✓	
R27		✓	✓						✓	✓	✓	✓	✓	✓	✓	✓	
R28	✓	✓			✓	✓	✓	✓	✓								
R29	✓	✓			✓	✓	✓	✓	✓								
R30		✓												✓	✓	✓	
R31		✓												✓	✓	✓	
R32					✓												
R33		✓	✓							✓				✓			
R34												✓	✓				
R35											✓		✓				

5 Implementation, integration and test plan

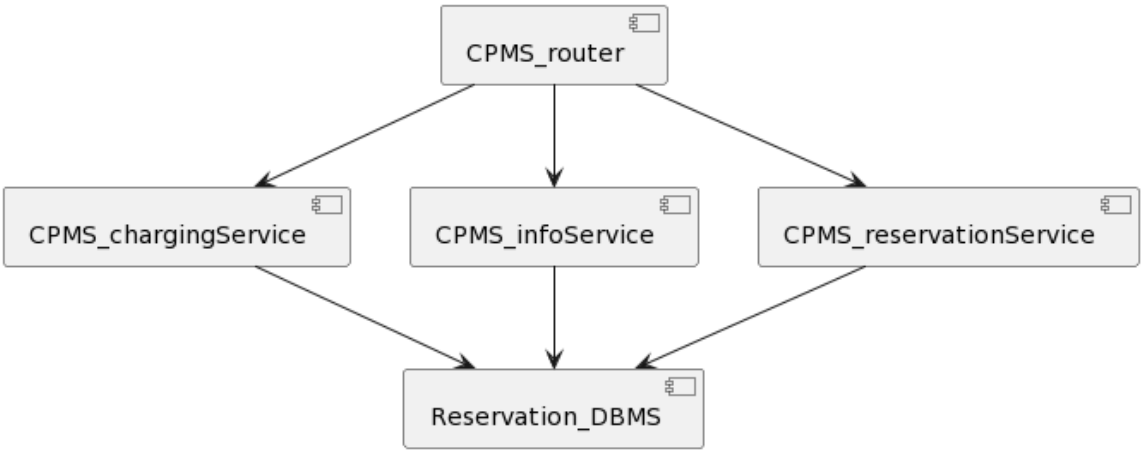
This section outlines the steps for implementing and testing the system.

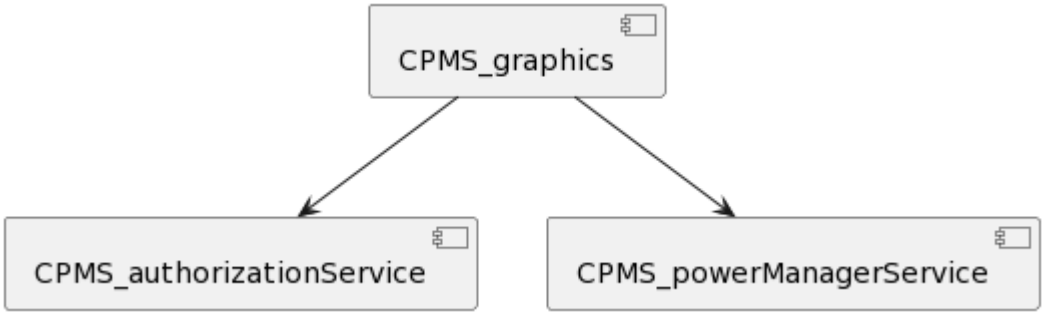
The system implementation will be structured and divided into groups based on the component diagram, to ensure a clear development path. The division is based on the subsistems identifiable in the component diagram: eMSP_app, eMSP_server and CPMS_server. Since all the communication API are well defined it’s possible to

integrate the three systems at the end of the implementation process of each one. The three subsystems can thus be implemented either in parallel or in sequence depending on the availability of resurces. For the eMSP_server subsystem the integration plan choosen is the bottom-up approach, since both services have a clear dependency on the user_DBMS.

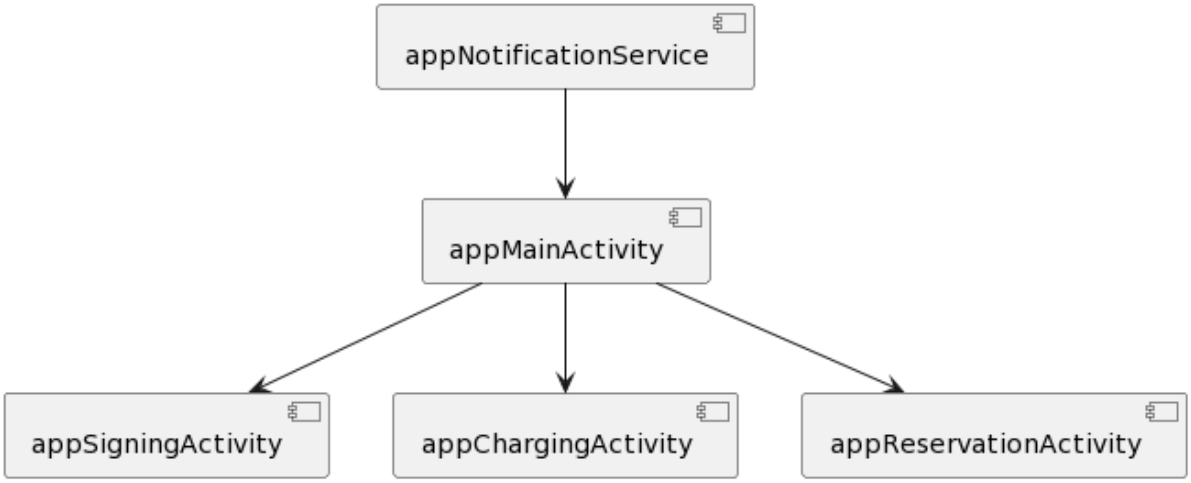


CPMS_server can be divided in two parts: the first composed by the CPMS_graphics, CPMS_authorizationService and CPMS_powerMangerService, and the second one composed by CPMS_router, CPMS_infoService, CPMS_chargingService, CPMS_reservationService and reservation_DBMS. This two parts can be implemented either in parallel or in sequence, each one following a top-down approach described in the following images.





The eMSP_app can be also implemented following a top-down approach described in the following image.



6 Effort

6.1 Riccardo Bravin

Task	Time spent
Introduction	30 min
Architectural design	22 h
User interface desing	3 h
Requirement traceability	3h
Implementation, Integration and Test Plan	6h 30 min
Total	32**h 30 min**

6.2 Elia Feltrin

Task	Time spent
Introduction	30 min
Architectural design	25 h
User interface desing	30 min
Requirement traceability	3h
Implementation, Integration and Test Plan	4h
Total	33 h

7 References

https://webeep.polimi.it/course/view.php?id=9008

https://plantuml.com/

https://www.omg.org/spec/UML/