AY 2022/2023

Politecnico di Milano

DD: Design Document

Marcello De Salvo Riccardo Grossoni
Francesco Dubini

Professor
Elisabetta Di Nitto

Version 0.1
December 29, 2022

Contents

1 Introduction

1.1 Purpose

The objective of this document is the realization of a full technical description of the system presented in the RASD document. Here we will analyze both hardware and software architectures, focussing on the interaction between components that constitute the system. Additionally, we will also delve into the implementation, testing and integration process. This document will use technical language as it's focussed to programmers, but stakeholders are also invited to read as it can be useful to understand the characteristics of the development.

1.2 Scope

The scope of this design document sets down the definition of the behavior of the system, for general and critical cases, and in the design of the system architecture by analyzing the logical designation of the components and their interactions. As stated before, this document will also delve into the implementation, the testing plan and a possible mock-up for the user interface.

1.3 Definitions, acronyms, abbreviations

Definitions

- **def1**: text.
- **def2**: text.

Acronyms

- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document
- **ITD**: Implementation Document
- **API**: Application Programming Interface

- **DBMS**: Database Management System
- **DMZ**: Demilitarized Zone
- **OCPP**: Open Charge Point Protocol
- **UML**: Unified Modeling Language
- **GPS**: Global Positioning System
- **IT**: Information Technology
- **GUI**: Graphic User Interface
- **UI**: User Interface
- **HTTPS**:HyperText Transfer Protocol Security
- **HTML**: HyperText Markup Language
- **CSS**: Cascade Style Sheet
- **JS**: JavaScript

1.4 Revision history

- Version 0.1: setup version
  - text

1.5 Reference documents

- Specification document: "Assignment RDD AY 2022-2023"
- Requirements Analysis Specification Document (RASD)
- UML documentation: https://www.uml-diagrams.org/
- Slides of the lectures

1.6 Document structure

- **Section 1** gives a brief description of the design document, it describes the purpose and the scope of it including all the definitions, acronyms and abbreviations used.
- **Section 2** delves deeply into the system architecture by providing a detailed description of the components, the interfaces and all the technical choices made for the development of the application. It also includes detailed sequence, component and ArchiMate diagrams that describe the system in depth.
- **Section 3** contains a complete description of the user interface (UI), it includes all the client-side mockups with some graphs useful to understand the correct execution flow.
- **Section 4** maps the goals and the requirements described in the RASD to the actual functionalities presented in this DD.
- **Section 5** presents a description of the implementation, testing and integration phases of the system components that are going to be carried out during the technical development of the application.

2 Architectural Design

2.1 Overview

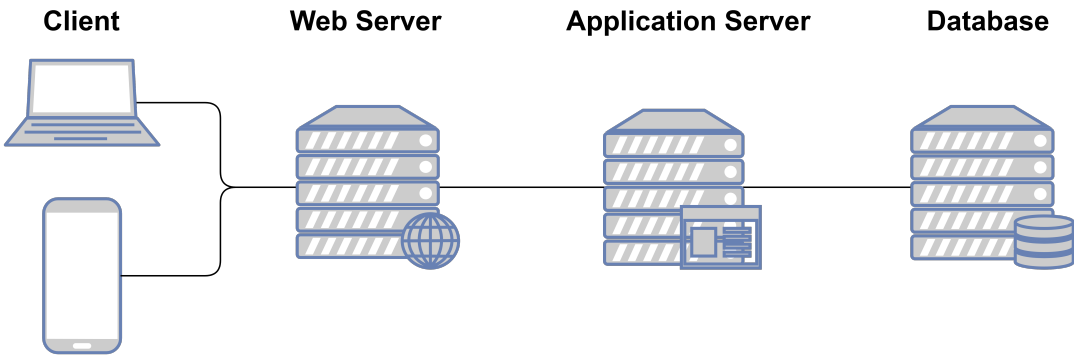


Figure 1: Three Tier Architecture

The system is an application distributed across multiple devices and follows the common client-server paradigm. The architecture is organized in three logical layers:

- **Presentation Layer (P)**: It's responsible for rendering the application's user interface to the client through the Web Server. It's expressed by a GUI designed to enable the user to interact with the application efficiently.
- **Application Layer (A)**: It's responsible for the business logic of the application. It receives requests from the Presentation Layer, processes them, and then sends back the results to be displayed.
- **Data Layer(D)**: It's in charge of the access to data sources, it provides data through the database and direct it to the other layers. It's also necessary in order to grant a high level of abstraction from the database for an easy to use model.

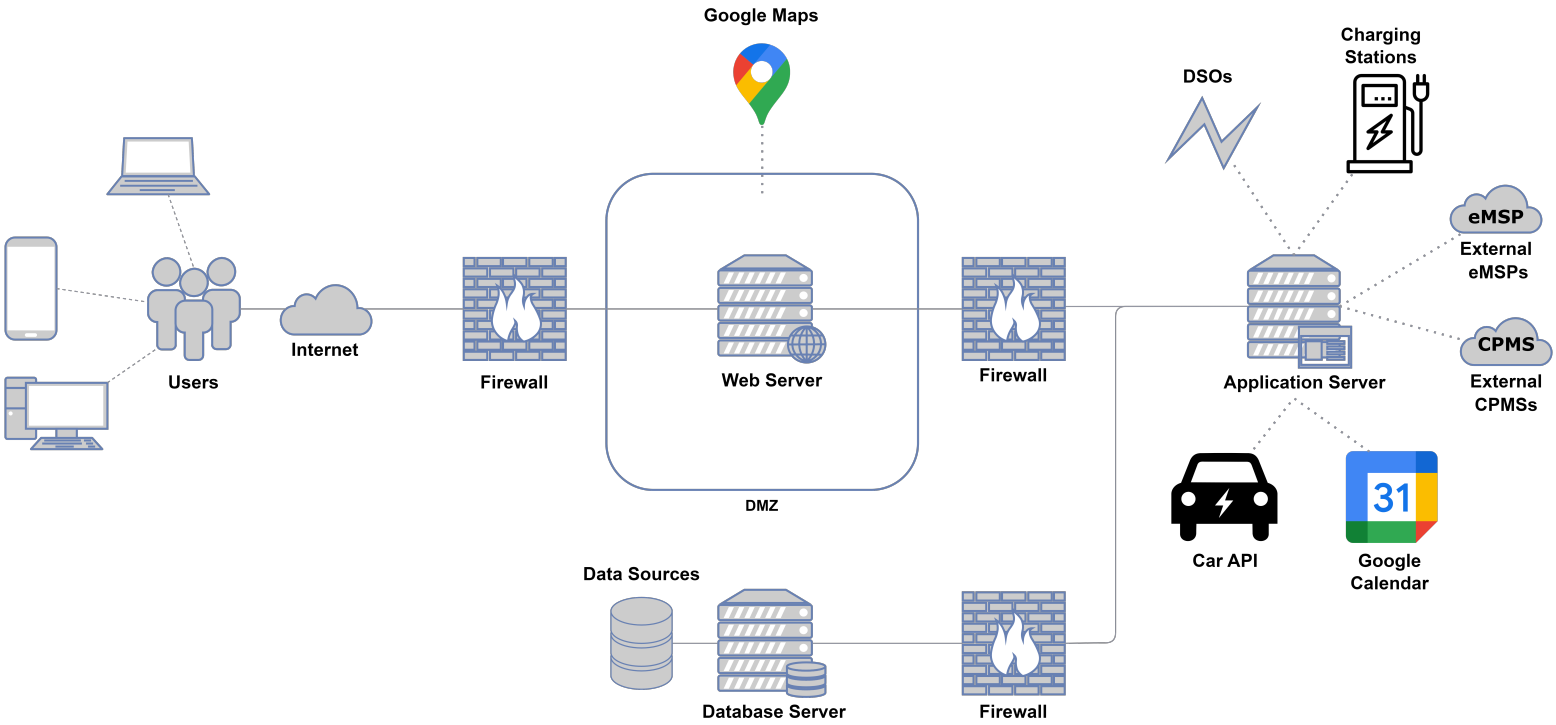


Figure 2: High Level Architecture

The system's architecture, as shown in **??**, is composed by a DMZ, that includes the Web and the Mail Server in order to prevent a direct access to the internal system and improve the overall security, and firewalls that divide each newtwork segment. To reduce computation client side a thin client is used, which allows all the heavy operations to be performed at server side. The Web Server handles the HTTP requests by the users and directs them to the Application Server, while also managing the GoogleMaps API. The Application Server communicates with the Database Server and elaborates data following its business logic. The Application server also handles external APIs and services, such as GoogleCalendarAPI, CarAPI, and the exetrnal CPMSs and eMSPs systems. The Apllication Server's CPMS subsystem will also communicate with its Charging Stations and their relative sockets through the OCPP API, and with the external DSOs through their proprietary API. The Email Server will manage all the e-mail notification

by communicating with the Application Server, The database server is responsible for storing and managing the data that is used by the application. It receives requests for data from the application logic tier, retrieves the requested data from the database, and returns the data to the application logic tier.

2.2 Component view

General Component View

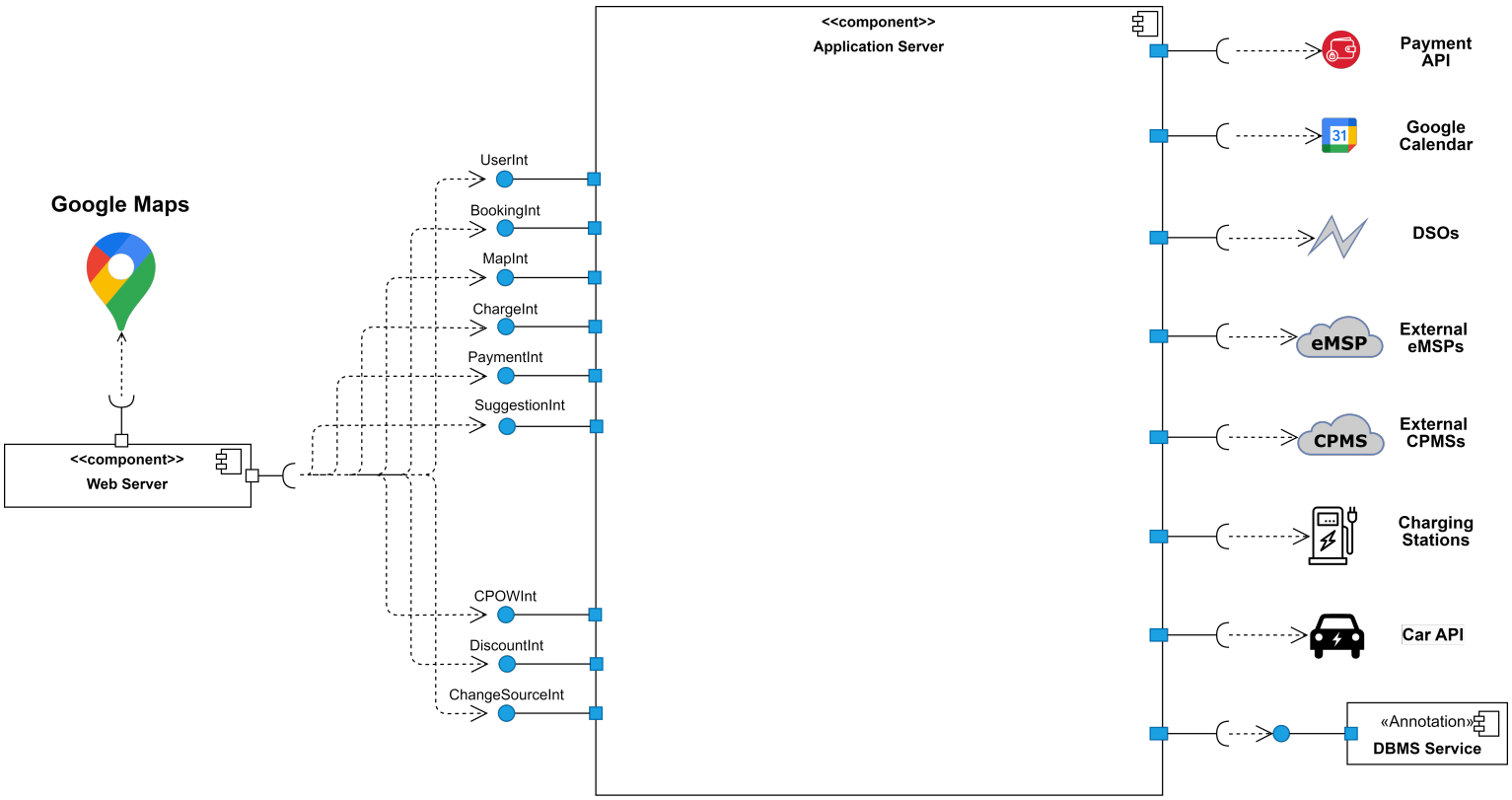


Figure 3: General Component Diagram

This image gives a high level representation of the components of the system. In the scheme the application server is represented by an empty

box, since a complete description will be provided in the next section. On the left the interfaces between the web server and the application server are shown; these basically represent the main functionalities requested by the client applications. On the opposite site the external interfaces are presented, among which there are the interfaces to other CPMSs and to the charging stations. Additionally, there's also the DBMS iterface, which manages the DBMS service and handles all communications between the Applicantion server and the Database server.

Application Server Component View

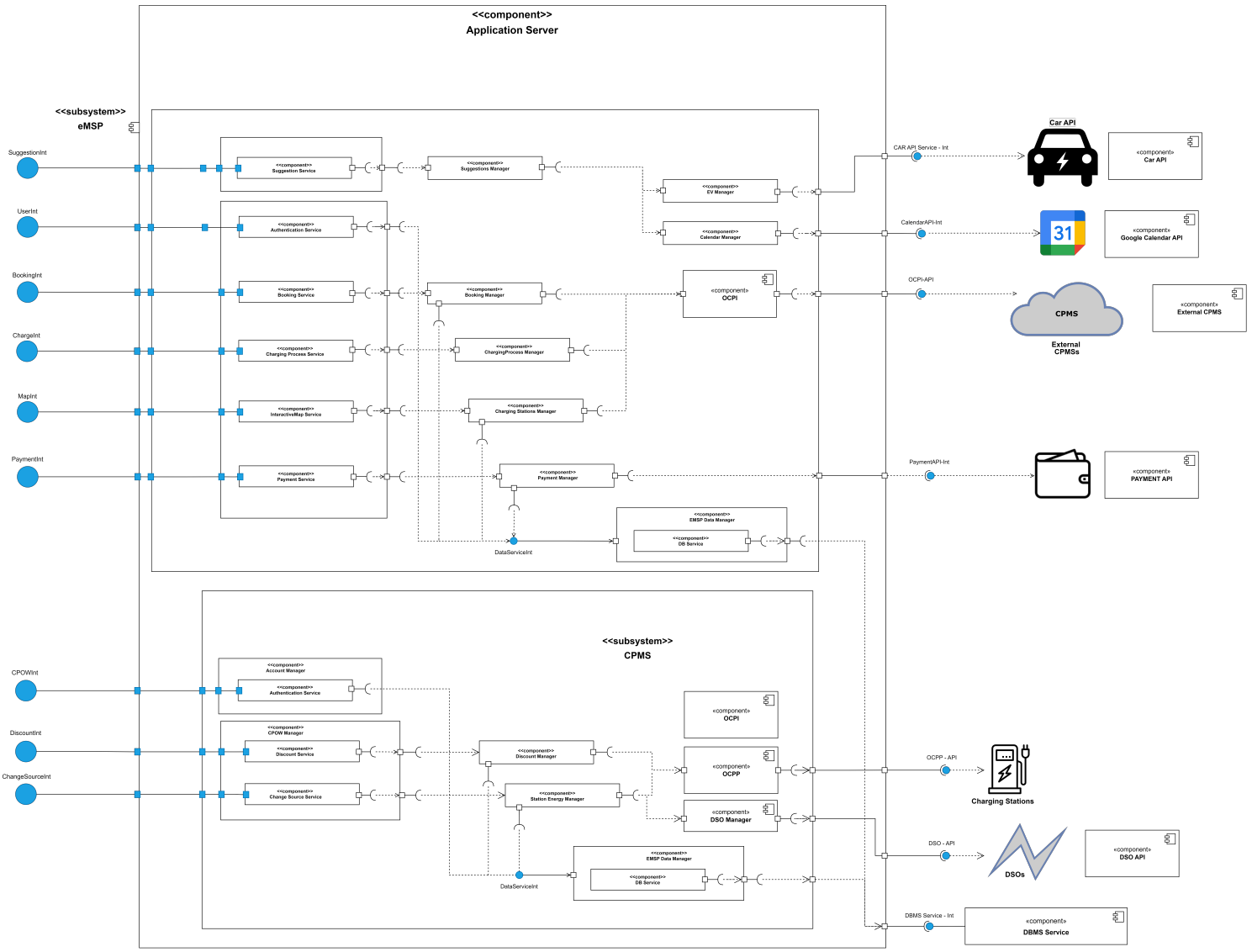


Figure 4: Application Server Component Diagram

The following component diagram gives a detailed view of the Application Server. It shows the internal structure and the interaction between the

components. External elements in the diagram are represented in a simplified way.

eMSP subsystem

- **AccountManager**: Handles all the basic requests that a client can make. Among those the *AuthenticationService* manages the authentication process (log in, log out, sign up). All the specific functionalities are then provided once the user logs in.
- **eMSP_UserManager**: It manages all the services available to the eSMP subsystem's users. Various services are then handled by other components, for example the *BookingService* is handled by the *BookingManager*, which communicates through the OCPI component with the CPMSs. Most services also communicate with the Database through an interface.
- **BookingManager**: It handles all the booking requests made by the clients and is responsible for sending the requests to the CPMS through the OCPI component.
- **ChargingProcessManager**: This component manages the charging process service of the user's car, in particular it provides updated informations about the charging status while interacting with the OCPI component to retrive informations and also managing the start and stop charging functions.
- **ChargingStationManager**: This component handles the Charging Stations general informations and status to be displayed through the *InteractiveMapService*. To do so it interacts with the OCPI component to retrive all the info from the CPMSs.
- **PaymentManager**: It manages the paymentService. It interacts with the external component *Payment*.
- **SuggestionManager**: It handles the service that generates suggestions for the users on when to charge the car. In order to do that it communicates with the *EVmanager* and with the *CalendarManager* to get all the informations needed to formulate suggestions, in particular the user's schedule and the car battery status.

- **CalendarManager**: This component manages the information relative to the user's schedule for the *SuggestionService*. To do so it interfaces with the external component *Calendar*.
- **EVManager**: This component handles the information relative to the user cars' battery status for the *SuggestionService*. To do so it interfaces with the external component *CarAPI*.

Web Server Component View

Regarding the Web Server the main components are:

- **first component**: description of component.

2.3 Deployment view

Figure 5: Deployment Diagram

The deployment diagram in *Figure (5)* shows the most important components necessary for the correct behaviour of the system. The devices shown in the diagram are:

- **first device/component**: description of device (e.g. pc or Smartphone, firewall, load balancer, web server, app server, database server...).

2.4 Runtime view

In this section we list some relevant use cases of the system, represented through sequence diagrams. In some diagrams, interactions like the login phase, returning to home page or retrieve the same kind of data were omitted for the sake of clarity.

Shared diagrams

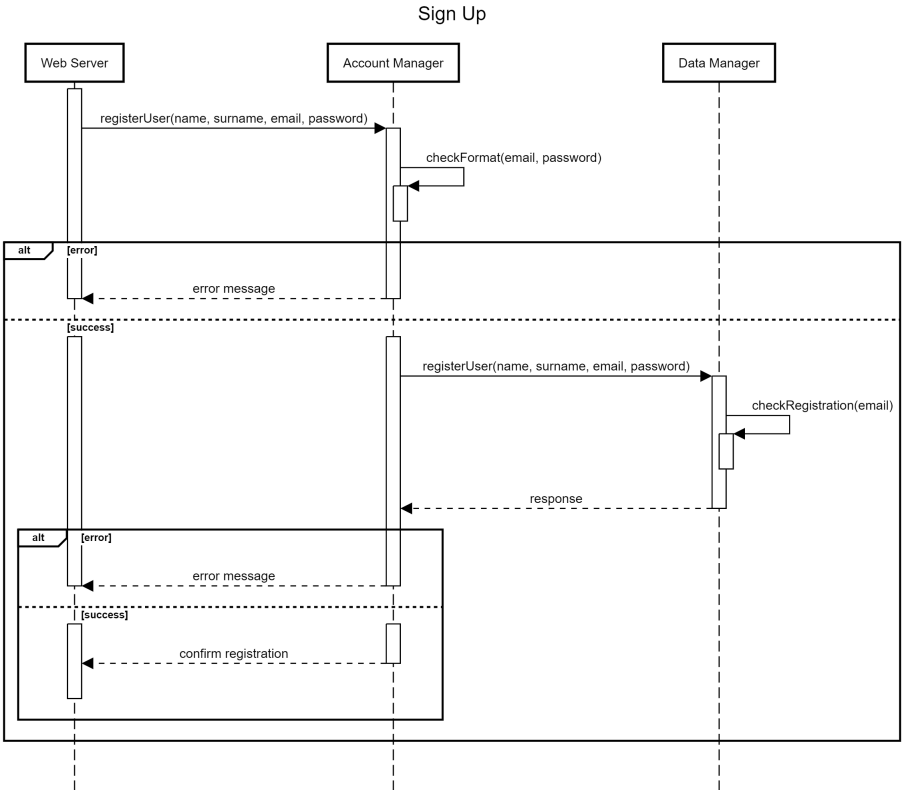


Figure 6: Sign Up

The Sign-up phase consists in the user action of inserting their personal data in the sign-up fields, then the system checks if the email (unique key in the database) is already present in the database. If it's not, the user can sign up and use the system functionalities available for that particular type of account depending if he's registering through the eMSP or the CPMS.

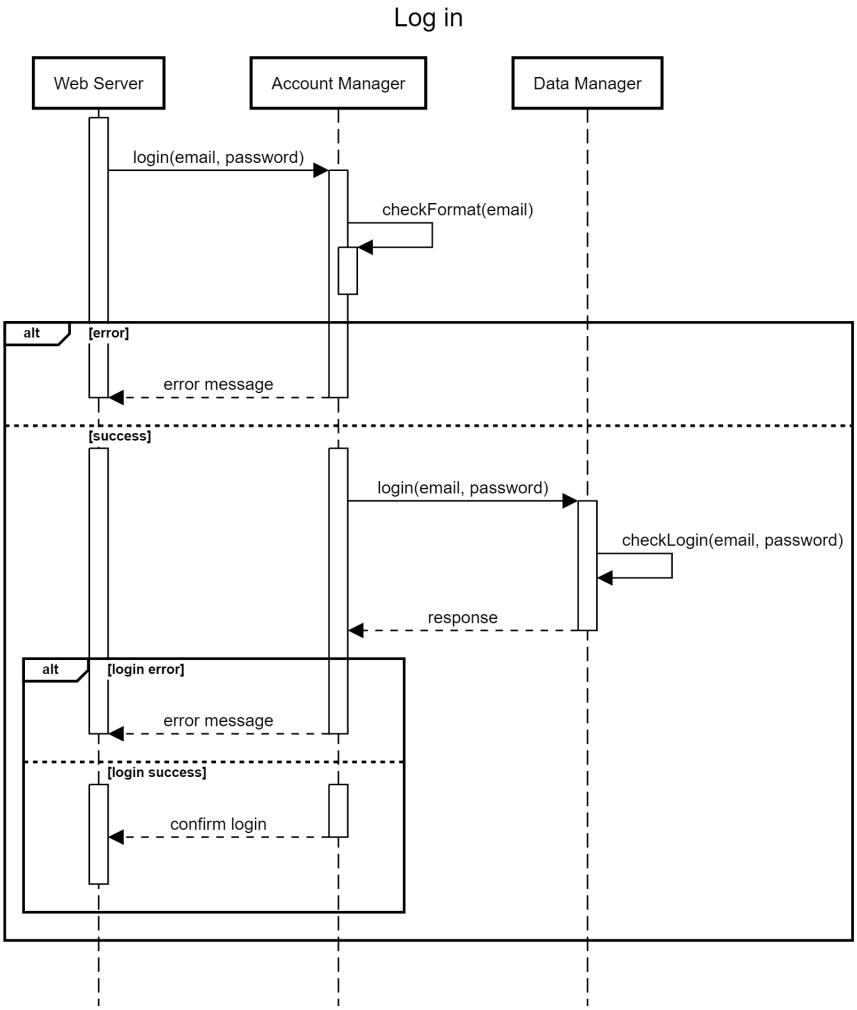


Figure 7: Log In

The Log In phase consists in the user action of inserting their email (unique key in the database) and password in the login fields, then the system checks if the pair corresponds to a user entry in the database. In case of success, the user can log-in and use the system functionalities available for that particular type of account depending if he's logging in through the eMSP or the CPMS.

User diagrams

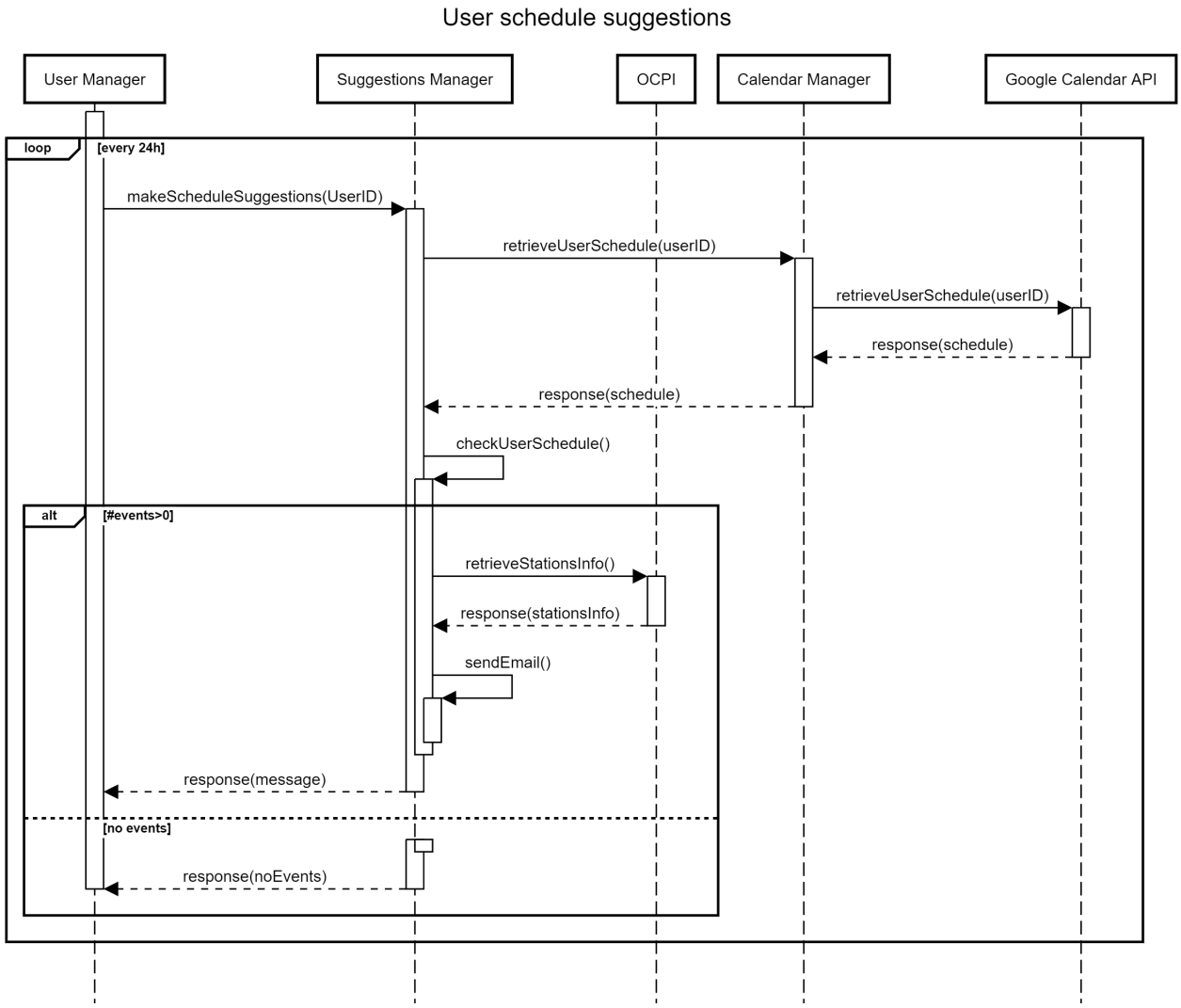


Figure 8: Schedule suggestions

The user's schedule suggestion service triggers a check every day at 00:00 to see if the user has planned some events for the next day. When it triggers, the SuggestionManager calls the CalendarManager in order to retrieve the user's schedule for the next day by using the Google Calendar API. Then the SuggestionManager checks all the new events: if there is at least one event it retrieves a list of charging stations information that are nearby the event's location by communicating with the OCPI component, and then sends an eMail with a list of charging stations based on their distance, cost and special offers. The email will contain a link for each station that will redirect the user to a pre-filled booking form, in order to easily book a socket.

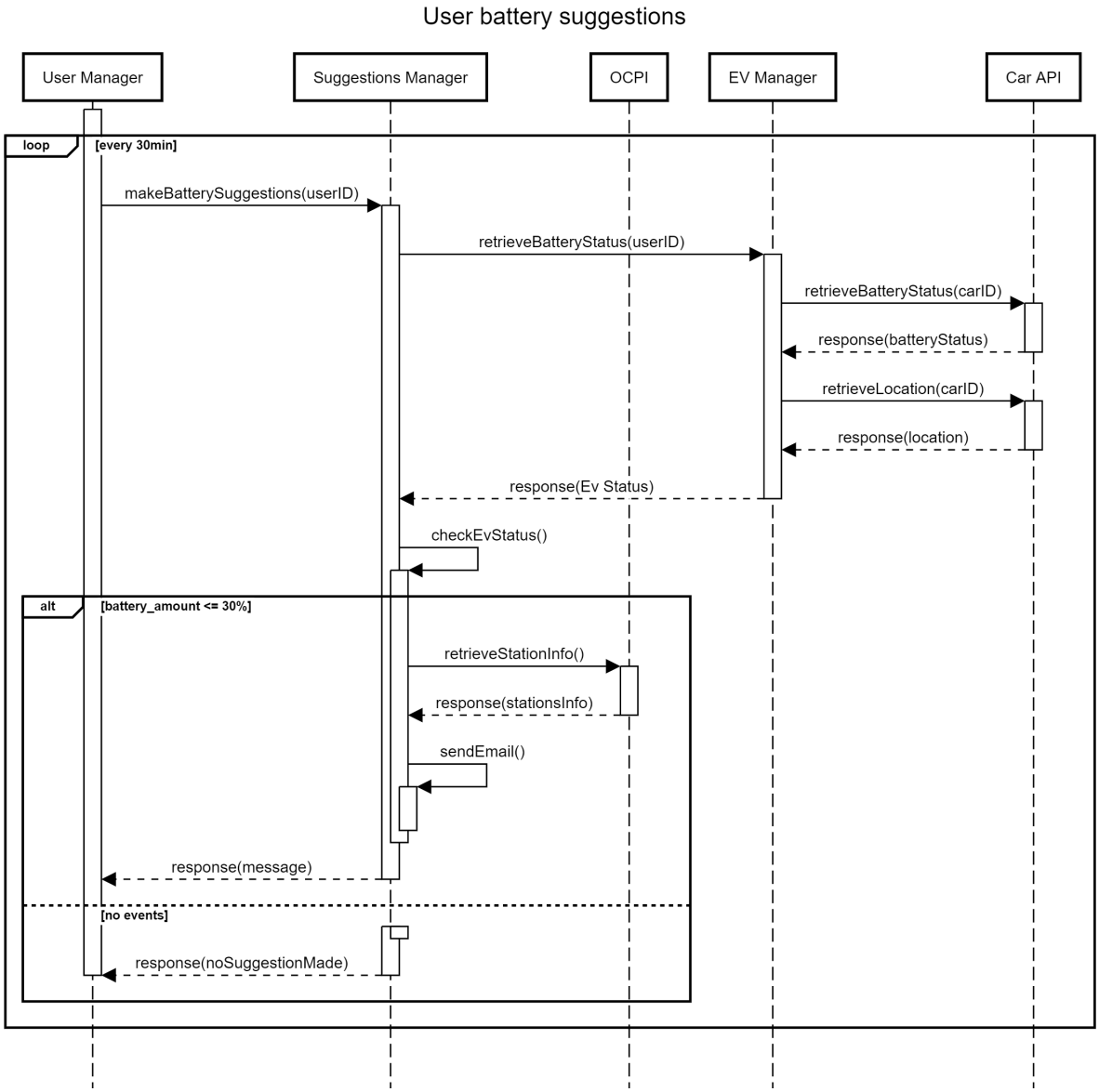


Figure 9: Battery suggestion

The battery suggestion service triggers a check every 30 minutes to see if the user'EV has battery status below a certain threshold. When it triggers, the SuggestionManager calls the EVManager in order to retrieve the user's car battery status and location by using the CarAPI. Then the SuggestionManager checks the battery status: if it's below a certain threshold it retrieves a list of charging stations information that are nearby the user's home or the car location by communicating with the OCPI component, and then sends an eMail with a list of charging stations based on their distance, cost and special offers. Note that this service is available only for the users that have a car registered in the system and that have a brand that supports this features with their API.

CPOW diagrams

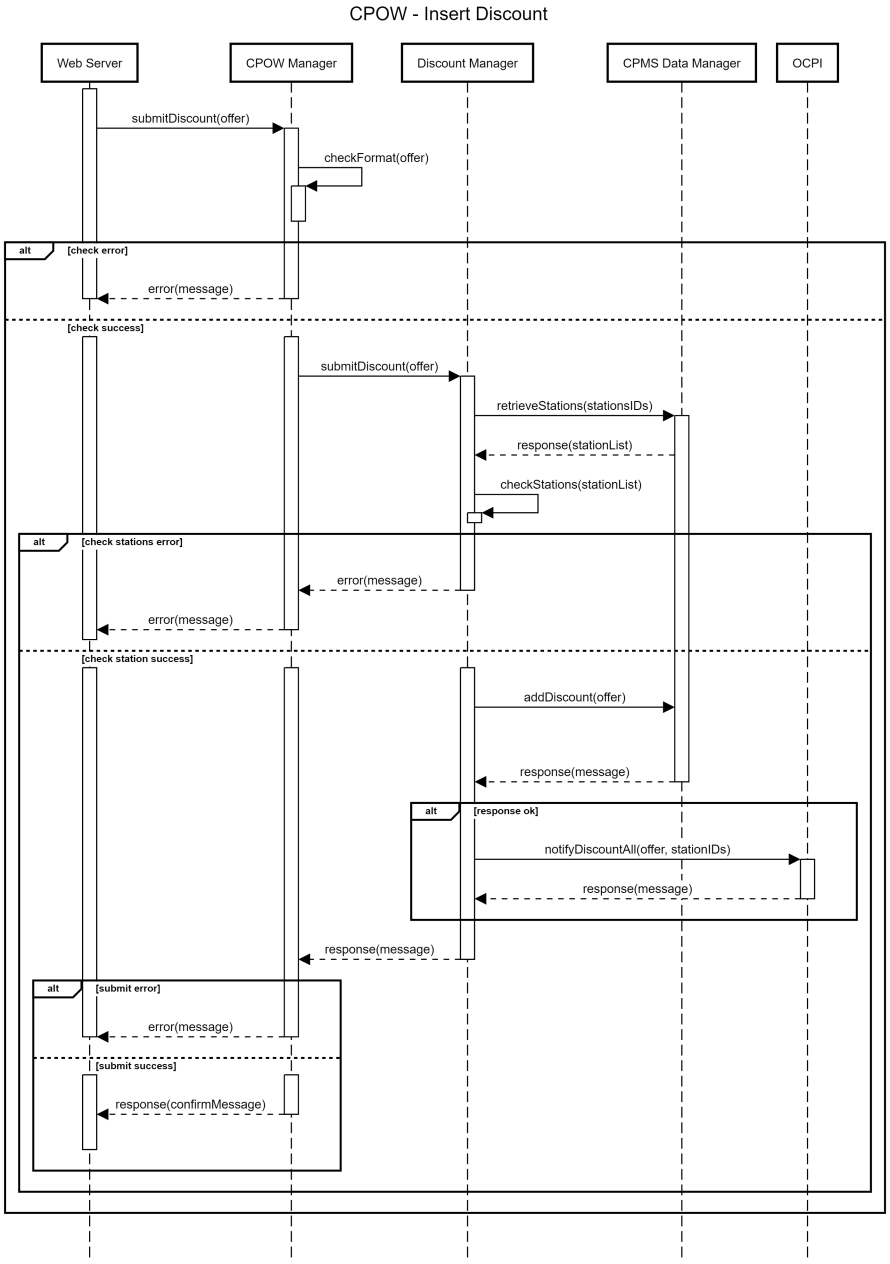


Figure 10: CPOW - Insert discount

The insert discount interaction triggers when a CPOW wants to insert a new discount for a list of charging stations. After receiving the form information, the CPOW Manager calls the Discount Manager in order to check if the stations are valid and if the discount is valid. Then the discount is inserted in the database and the Discount Manager calls the OCPI component in order to notify all EMSPs that are connected to the CPMS about the new discount. Note that in this diagram it is not showed the interaction with the OCPP component because it will apply the discount to the charging stations only when the discount's start date is reached.

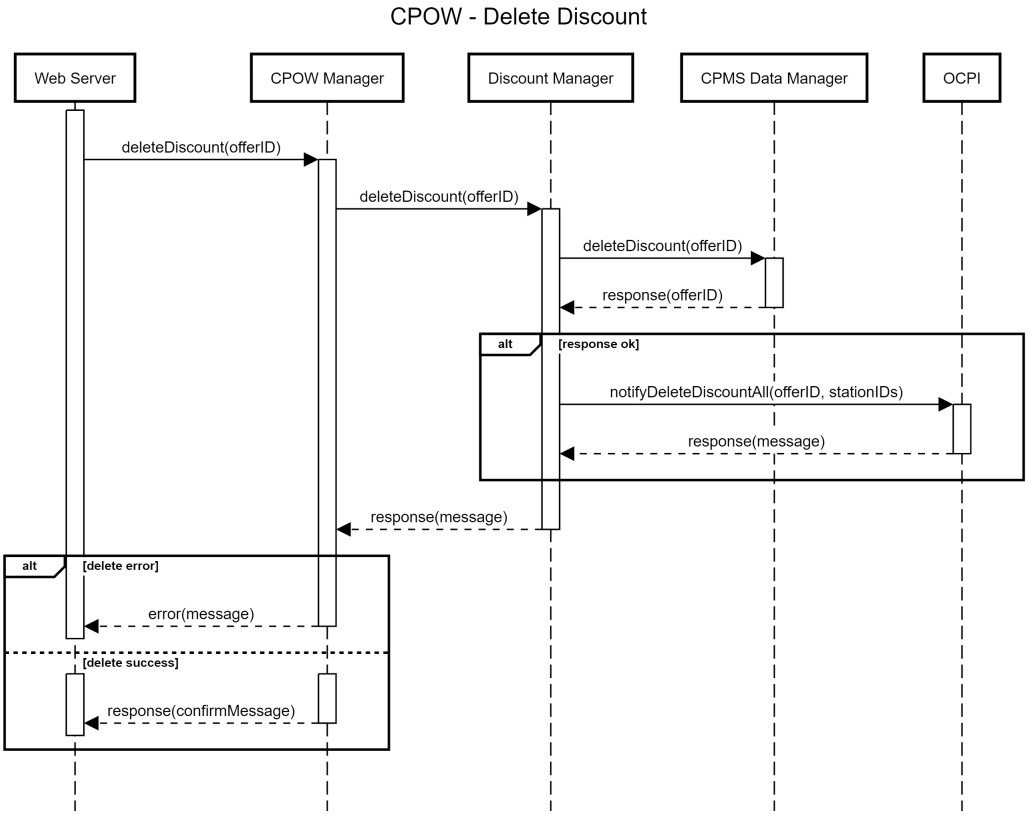


Figure 11: CPOW - Delete discount

The delete discount interaction triggers when a CPOW wants to delete a discount for all the charging stations to which it was applied. After receiving the discountID, the system will check if the discount exists. If it is, it will delete it from the database and notify all EMSPs that are connected to the CPMS.

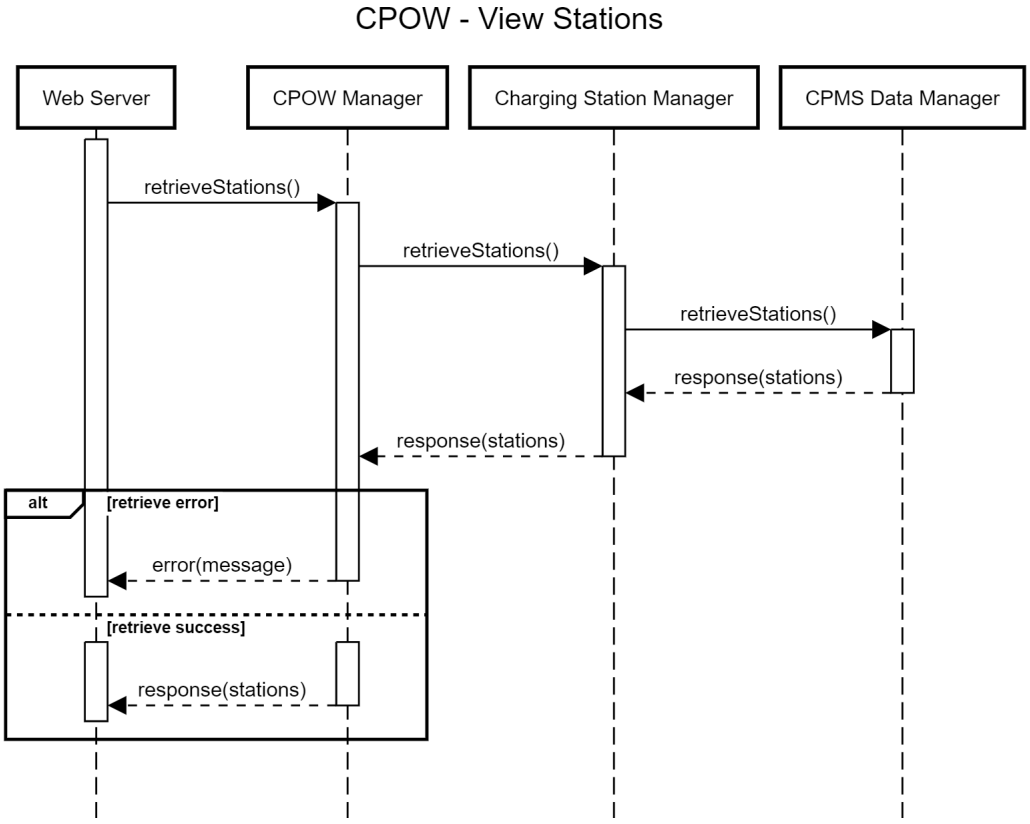


Figure 12: CPOW - View charging stations

This diagram describes the data fetching interaction between the Web Server and the Applicantion Server in order to retrieve the list of charging stations that are connected to the CPMS. The list will contain all the stations details, such as the ID, the location, the energy provider, the status and the socket details and a button to change the energy provider.

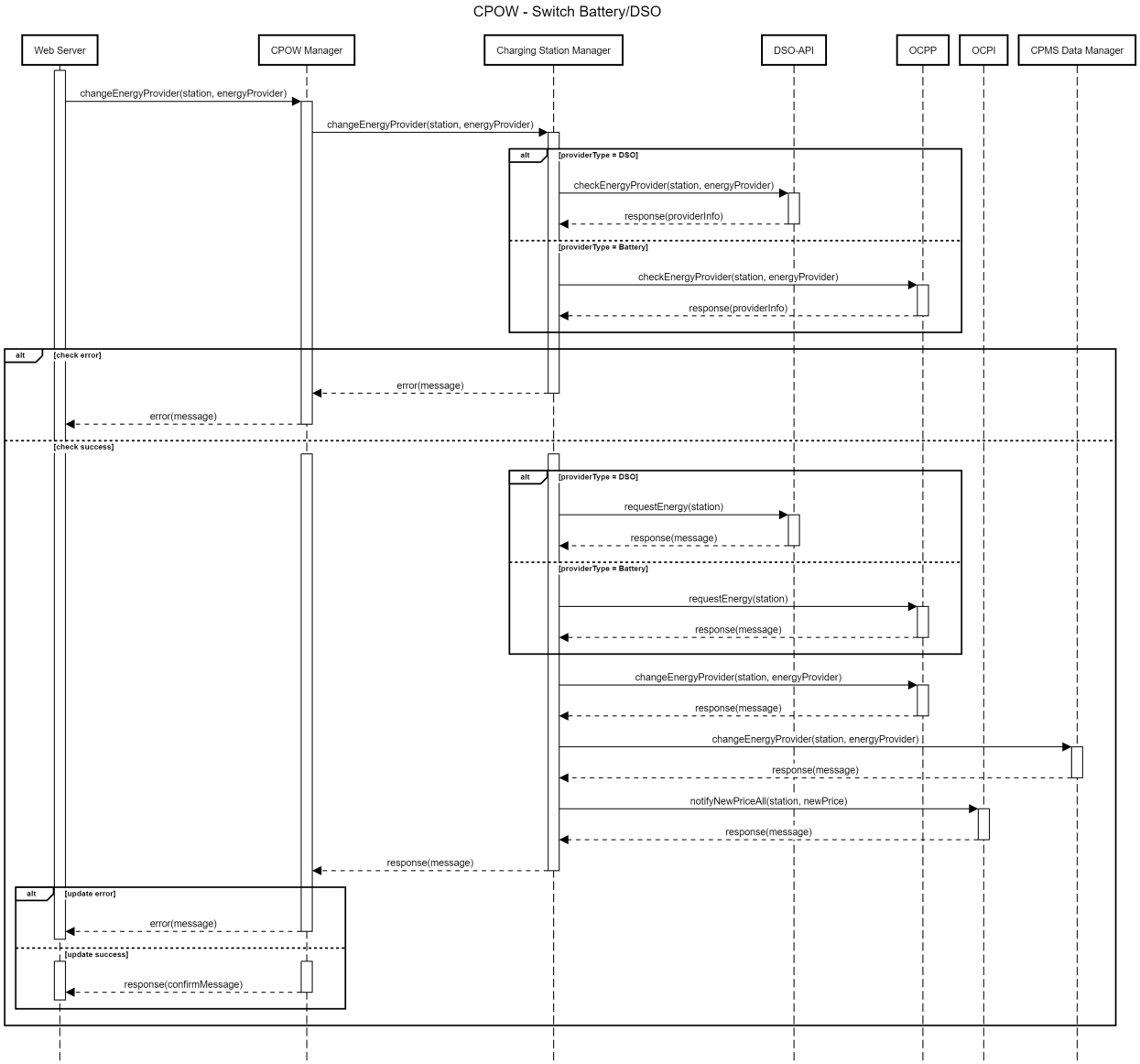


Figure 13: CPOW - Change energy provider

This diagram describes the interaction between the CPOW and the Applicantion Server in order to change the energy provider of a charging station. It starts by the CPOW sending a request with the charging station ID and the new energy provider (after retrieving a list with all the stations details, described in Figure 12). Then, the Charging Station Manager will check if the station exists and if the energy provider is valid by communicating with the OCPP or the DSP-API component, depending if the selected provider is a DSO or an internal battery. If the selected provider is available, the Charging Station Manager will ask the Provider to start supplying the energy, notify the station through OCPP, notify the connected eMSPs through OCPI, and update the station's current energy provider in the database.

2.5 Component interfaces

Figure 14: Component Interfaces Diagram

This diagram above describes in detail the interfaces and the corresponding methods offered by each component, it also shows the interaction between them. *other text*

2.6 Selected architectural styles and patterns

- **Three tier architecture**
  The selected architecture for this system is a three tier architecture, which consists of a presentation tier, a logic tier, and a data tier. The presentation tier is responsible for displaying information to the user and accepting input, the logic tier is responsible for processing the input and performing any necessary calculations or operations and lastly the data tier is responsible for storing and retrieving data from a database.

- **Thin client**
  The thin client approach increases security, since sensitive data will

not be stored locally, and scalability, because it allows for an easier deployment of new clients, while also granting a more centralized management.

- **Scalability**
  A three tier architecture allows for a more modular design, meaning that it's easier to scale individual components of the system as changes to one component do not necessarily require changes to the other components.

- **Model View Controller**
  The Model-View-Controller (MVC) design pattern is a software design pattern that separates an application into three main components: the model, the view, and the controller. The model represents the data and business logic of the application, the view represents the user interface, and the controller mediates communication between the model and the view. In particular:

  - Model: The central component of the pattern, it's the dynamic data stracture of the application, meaning that it changes as data is added modified and deleted, it's indipendent of the user interface as it's not concerned with how the data is displayed and how the user interacts with the application and it directly manages the data, logic, and rules of the application.
  - View: It represents the user interface of the application. It responsible for displaying information and accepting imput from the user. It defines how the application data should be presented.
  - Controller: It's responsible for the mediation between the model and view components, as it receives input from the user through the view and performs the necessary actions on the model based on that, then it updates the view to reflect the modified model. As a mediatior it's indipendend from both the components.

2.7 Other design decisions

2.7.1 Charging Stations Offers

All the offers have a standardized structure: a starting date, an ending date and a value, which represents the percentage of discount on the current price. This structure will be used on all the offers shown in the eMSP-side of the application. Additional options, for example the application of different offers to different sockets of a charging station, will not be available, but different offers can be applied to different charging stations simultaneously.

2.7.2 Automatically change DSO or Station Battery

The CPMS subsystem will be able to dynamically change The source of energy for the charging stations. This operation can be overwritten by a human operator, and will take in consideration different aspects in order to decide which energy source and provider is currently the best. Among these aspects there are the current price of energy for each DSO, the median price of the energy used by the station in the previous period, the current charge of the battery station (if there's any) and the availability of the DSO. If a certain threshold is met the CPMS will start the change process with the best option.

2.7.3 Car position for suggestions

Most brands of EVs already offer the possibility of seeing the battery status of the car and the position of the car. Our application will use that information to make suggestions on near charging locations when the battery is low based on the Car location, which will be given through the car's specific API.

2.7.4 Schedule based Suggestions

Each day at a specific time the user will be notified, if at least one event is present, with suggestions on charging locations located near the event, ranked by price (also considering current active offers). This means that if the user changes the schedule a new suggestion cannot be granted.

2.7.5 Startining a booked charging session

If a user reserved a socket for a specific time, during that time only that user will be able to start a charging session. Other users might still be able to physically connect their car to the socket, depending on how the booking is handled by the specific CPO system, but a charging session will not be able to be started by different users.

2.7.6 Automatically cancelling a booking

If enough time passes from the start of the booked time where the user does not start a charging session, the reservation will be cancelled and any user will be able to start a session with that socket.

3 User Interface Design

Figure 15: Sign Up and Log In

Figure 16: carOwner interactions

Figure 17: CPOW interactions

4 Requirements Traceability

Requirements	**Components**
R1) text. R2) text. R3) text.	• text. • text. • text.
R4) text.	• text.

R5) text.
R6) text.
R7) text.
	• text.
	  – text.
	  – text.
	  – text.
	• text.
	• text.
	• text.
	• text.

R8) text.	• text.

R9) text.	• text. – text. • text.
R10) text.	• text. – text. • text. • text. • text.
R11) text.	• text. – text. • text. • text.
R12) text.	• text. – text. • text.

R13) text. R14) text. R15) text. R16) text. R17) text.	• text. – text.
R18) text.	• text. • text. – text.
R19) text.	• text. • text. • text.
R20) text.	• text. • text.
R21) text.	• text.

Here, we present a summary of the table above for a more immediate visualization. *component: name associated in the next table*

component 1:	c1
component 2:	c2

Table 2: Components' legend

	c1	c2
R1	x	x
R2	x	x
R3	x	x
R4	x	
R5		
R6		
R7		
R8		
R9		
R10		
R11		
R12		
R13		
R14		
R15		
R16		
R17		
R18		
R19		
R20		
R21		

Table 3: Component and requirement mapping

5 Implementation, Integration and Test Plan

5.1 Implementation Plan

Multiple components will be implemented at the same time, in order to parallelize the development when possible. The general plan is to follow a bottom-up approach, so that core and basic functionalities with very few dependencies can be tested as soon as their incapsulating component is done. By doing so, the application will be built up with solid and tested foundations that will ease the further testing of bigger and complex components. In any case, unit testing will be performed on each component on the go, in order to

find flaws out in advance. This will positevely impact the necessary actions to fix the faults since they will be done in an earlier stage.

The implementation's order of the component will be as follow:

1. list, of, components
2. list, of, components, 2
3. list, of, components, 3

Each group is composed by independent modules so they can be easily developed in parallel. Furthermore, it is expected that external services (e.g. GoogleMapsAPI, *others* and DBMS Service) work properly since they're not a responsibility of the *name* app.

explanation of the implementation plan component order

5.2 Integration Strategy

Considering both the overall system's architecture and the implementation plan, the chosen integration strategy is the bottom-up approach. System integration begins with the integration of the lowest level modules and uses test drivers to drive and pass appropriate data to the lower-level modules. As and when the code for the other module gets ready, these drivers are replaced with the actual module.

This approach allows to start the integration and testing without necessarily waiting for the completion of the development and the unit testing of each system's component. Being the low-level modules and their combined functions often invoked by other modules, it is more useful to test them first so that meaningful effective integration of other modules can be done. Moreover, starting at the bottom of the hierarchy means that the critical modules are built and tested first and therefore any errors in these modules are identified early in the process.

Each integration in the same level (defined by the groups of the previous section) is independent and there is no specific order in which to complete them. In this way, the integration process and its testing are more flexible.

5.2.1 Integration and Testing

In this section it is defined the order of the integration between components. Test drivers will be used to simulate higher components not yet implemented.

order of the components shown in the next figures, with references

Figure 18: Integration of first component

Figure 19: Integration of second component

(a) component 3a

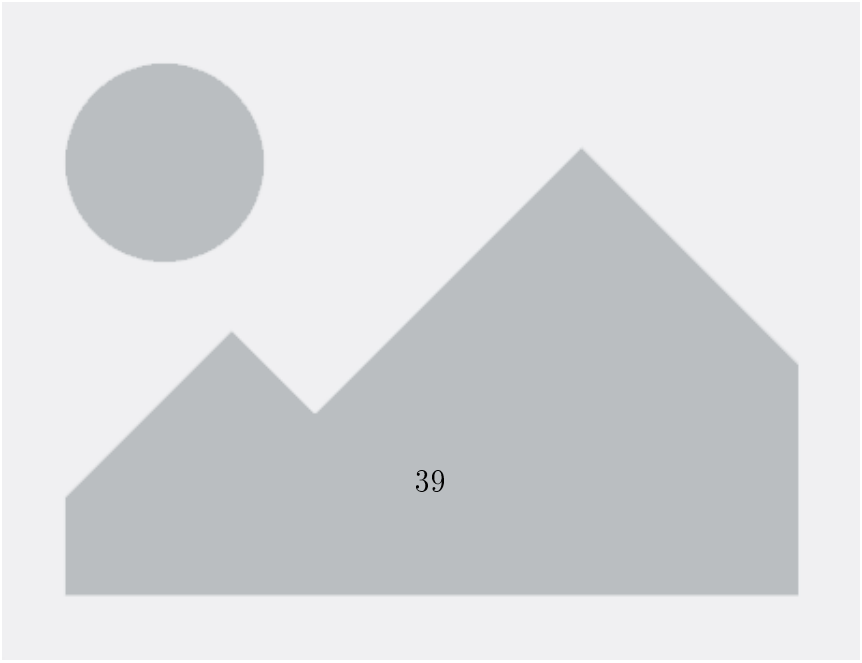
(b) component 3b

Figure 20: Integration of 3a and 3b components

order of components for the middle tier, with references

(a) component middle1-a

(b) component middle1-b



(c) component middle1-c

Figure 22: Integration of middle2 component

order of last group components, with references to the images

Figure 23: Integration of carOwnerManager components

Figure 24: Integration CPOWmanager components

Figure 25: Integration of otherManager components

Figure 26: Integration of AccountManager components

5.3 System testing

text for sys testing

6 Effort Spent

Student	Time for S.1	S.2	S.3	S.4	S.5
stud1	0h	0h	0h	0h	0h
stud2	0h	0h	0h	0h	0h
stud3	0h	0h	0h	0h	0h

7 References

References

[1] MDN Web Docs Glossary: Definitions of Web-related terms -> MVC `https://developer.mozilla.org/en-US/docs/Glossary/MVC`

[2] description: `urlhere`