Politecnico di Milano
Department of Electronics, Information, and Bioengineering
Computer Science Engineering

Software Engineering 2

eMall - e-Mobility for All

Group components:
Roberto **CIALINI** 933385
Umberto **COLANGELO** 935073
Vittorio **LA FERLA** 224509

Academic Year 2022-2023

Contents

1 Introduction

1.1 Purpose

Nowadays sustainability is one of the most important and debated topics in our society. In fact, in the next few years we are going to deal with a huge green transition to limit our carbon footprint on the planet, such as in the transportation field, which finds itself as one of the main contributors of global warming. In this direction, in recent years the old motor vehicles running on gasoline are leaving space for electricity-powered vehicles, even though there are several central aspects to deal with in order to let the electric vehicles be competitive with the old vehicle generation. In this direction, the goal is to create a fully operative and diffused infrastructure for the fast charging of the batteries, which is one of the main limitations for the final customer. In fact, the batteries need to be charged often and nowadays the task of finding an available charging spot is not as easy as it seems.

With this issue in mind, *eMall* is an operating system, itself composed of two subsystems, whose goal is to offer a way to find available charging stations for electric vehicles, offering at the same time to the user the possibility to access several features such as the reservation of a specific socket at a certain timeframe or the reception of personalised proactive suggestions by the system.

This document contains a description of the architectural design for the system, including the components involved and how they interact. Additionally mockups of the user interface are presented, along with a plan for the implementation, testing and integration of the system. Therefore, this document should guide the development of the system.

1.2 Scope

While there are several stakeholders to consider, this document is only concerned about two actors: Drivers and Administrators. The Driver is the final user, the one who interacts with the *eMSP* to have the possibility to book the battery recharge of his vehicles. Instead the role of the system Administrator mainly concerns to monitor the correct behaviour of the system and to take strategic decisions.

The main system is divided into two subsystems: the *eMSP* and *CPMS*. The *eMPS* is designed to be an interface and to communicate both with the Driver and the Administrator, driving their requests. The *CPMS*, instead, is modelled on the *OCPI 2.2.1* protocol and is referred to as a specific *CPO*. The main task of the *CPMS* is to supply information about its *CPO* charging stations to the *eMSPs* it is linked to, both for the Driver and the Administrator usage.

Although *CPO* and *DSO* are mentioned in this document along with the other entities described before, we will not consider either their internal system or their decision making.

The architecture of *eMall* follows the three-tier pattern with a presentation layer, business logic layer and a data layer.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Definitions	Description
Driver Identifier	To identify a specific driver, this could be an identification number such as her/his SSN
Car Identifier	To identify a specific car, this could be the licence plate
Station Identifier	To identify a specific charging station

1.3.2 Abbreviations

Abbreviations	Definitions
RASD	Requirements Analysis and Specification Document
API	Application Programming Interface
RX	Requirements number X
GPS	Global Positioning System
CLI	Command Line Interface
DBMS	Database Management System
DB	Database
MVC	Model View Controller
eMall	e-Mobility for All
EV	Electric Vehicle
Driver	Electric vehicle driver
Administrator	*CPO* administrator

REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
OCPI	Open Charging Protocol Interface
OCPP	Open Charge Point Protocol

1.4 Revision History

Version 1.1.0

1.5 Reference Documents

- The specification document "Assignment RDD AY 2022-2023_v3.pdf"
- RASD

1.6 Document Structure

This document is composed of seven sections, detailed below.
In the first section the problem is introduced together with the purpose of this specific report and a recap of the context. Additionally, some necessary information in order to read the report is given, such as definitions and abbreviations.
Section two contains the description of the architectural design of the system together with motivations and reasons that led to opting for these solutions. It starts with a high-level overview of the architecture and then breaks each part down into components. The components are described and their interdependence are shown in the component diagram. Moreover, the section contains a component interface diagram, a deployment view and sequence diagrams describing the interactions between components in the runtime view.
In section three design mockups of the user interface is presented.
Section four contains the requirement traceability matrix, where each of the components described in section two is mapped to the requirements specified in the *RASD*. The mapping is based on whether the component contributes to the fulfilment of the requirement.
Section five describes the suggested implementation order and test plan of the system.
In section six is shown the total effort spent by each of the project members.

Section seven contains the references used.

2 Architectural Design

2.1 Overview: High-level components and their interaction

In this chapter the architectural structure of the System will be described. In the following diagram the high level overview of the System is shown. In the next sections we will describe in detail the various components and their interactions.

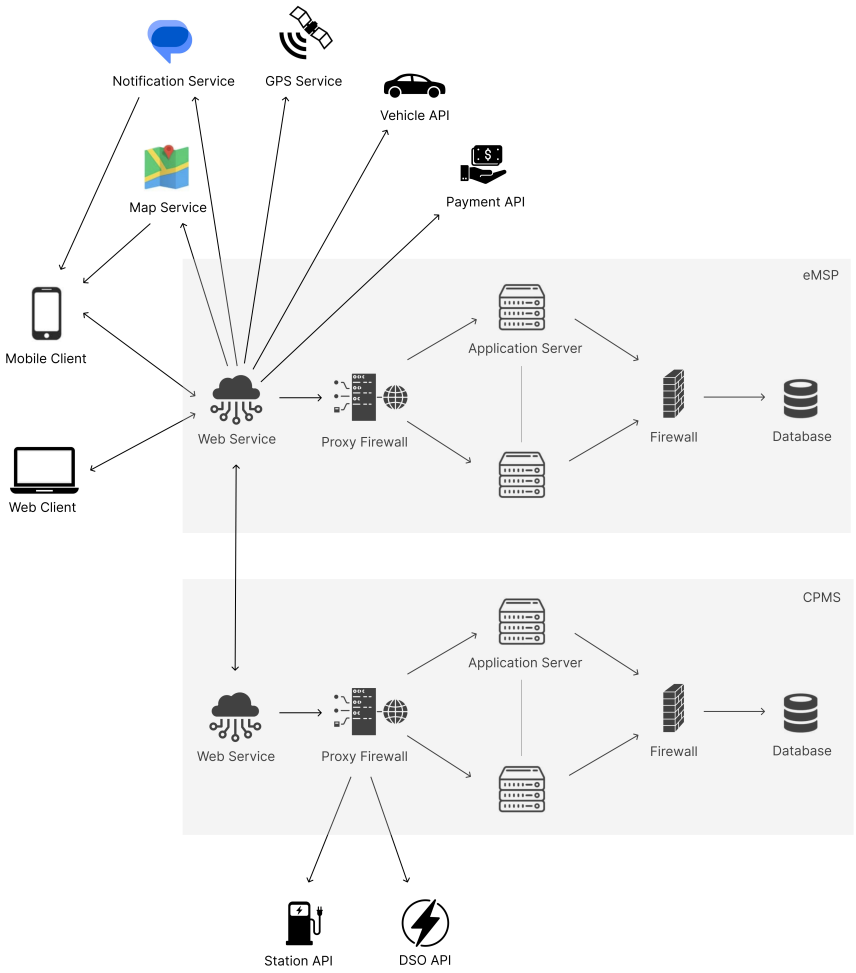

Fig. 1: *High-level View*

2.2 Component view

In this section the component view is presented with a component diagram and corresponding descriptions.

2.2.1 eMSP

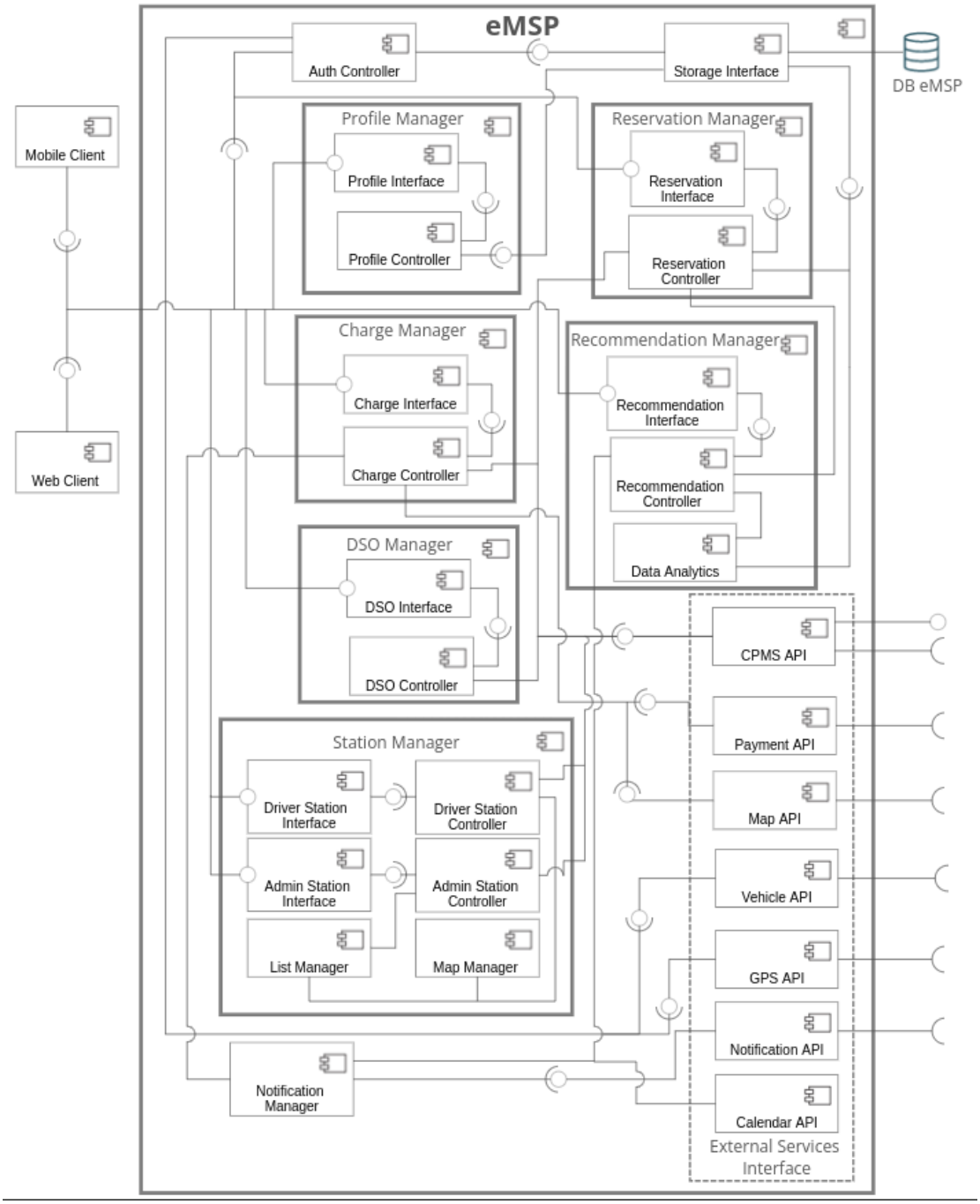

Fig. 2: *Component View of eMSP*

- **Mobile Client** and **Web Client**: represents the two machines that accesses the entire System and it's functionalities.

- **Auth Controller:** this component handles all the operations for the authentication such as registration process and login process. It communicates with the DBMS through Storage Interface in order to retrieve and insert user credentials.
- **Storage Interface:** it provides methods to access the database. This interface is needed in order to decouple all the components from the DBMS technology.
- **Notification Manager:** this component provides Notifications to the user, such as new Recommendation or to notify the end of the charge.
- **Profile Manager:** this component handles all changes to the user profile, such as change the Active Vehicle or enter a new payment method.
- **Station Manager:** this component is needed to handle all the operations about the stations. Basically provides all the views in which are involved the stations.
- **Charge Manager:** this component is dedicated to handle all the operations about the charging status, such as provide the views to monitor the status of charge or to forward the request to stop the charge.
- **Recommendation Manager:** this component aim is to calculate and provide recommendations to the users.
- **DSO Manager:** this component provides the views to the Administrator for the management of DSO settings.
- **Reservation Manager:** this component it's required to handle all the reservations made by the users and provide an interface to visualize them. It has also the duty to store all the reservation for each user on the DB.
- **External Services Interfaces:** this set of components have as main objective to make API calls to the necessary third party services:
  - **CPMS API:** it needs to communicate with the CPMS system in order to make requests. This component is in charge of the communication between the two systems.

- **Payment API:** it needs to communicate with the payment methods of the user to make the payments.
- **Map API:** it is required to communicate with the Device map system of the users.
- **Vehicle API:** it is required to retrieve all the information of the vehicle's user, such as battery status and socket type.
- **GPS API:** it is required to access the user's GPS device
- **Notification API:** communicates with the user's device in order to send notifications.
- **Calendar API:** retrieve the calendar information of the user to calculate the recommendations.

2.2.2 CPMS

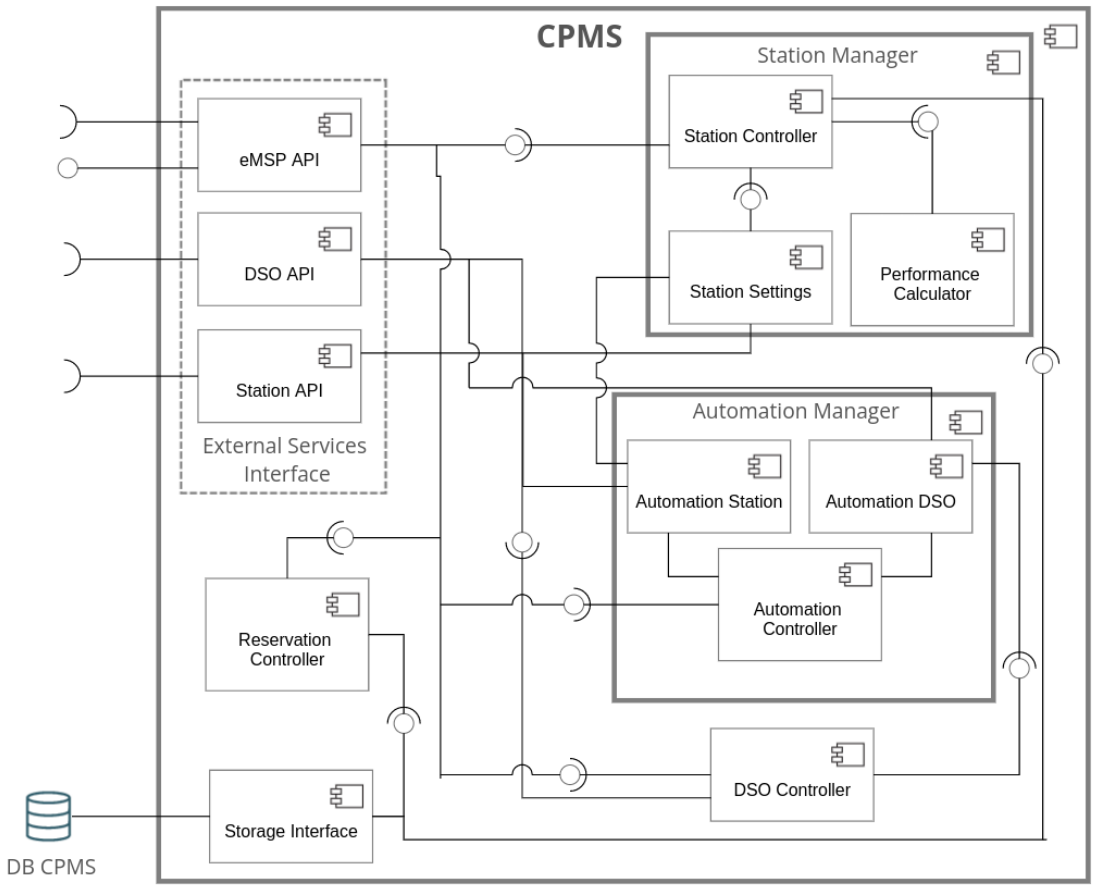

Fig. 3: *Component View of CPMS*

- **Storage Interface:** it provide methods to access to database. This interface is needed in order to decouple all the components from the DBMS technology.
- **Reservation Controller:** this component handles all the operations about the stations reservations , such as create a new one or delete it.
- **DSO Controller:** this component it is used to handle the DSO settings
- **Station Manager:** this component handles all the stations setting. It also needed to retrieve all the information about the stations.
- **Automation Manager:** this component is required to handle the automatic mode for the Energy Mix of the stations and the choosing of the DSO.

- **External Services Interfaces:** this set of components have as main objective to make API calls to the necessary third party services:
  - **eMSP API:** it needs to communicate with the eMSP system in order to forward the answer of the requests made by the eMSP.
  - **DSO API:** it needs to communicate with various DSOs to retrieve all the information or to change settings.
  - **Station API:** it is needed to communicate with all the station to modify their internal status or to retrieve the information about them.

2.3 Deployment view

The following diagram shows how the whole system is distributed in various components. In particular it is possible to see how the system is structured in a 3 tier architecture in which every component has their role, as specified below.

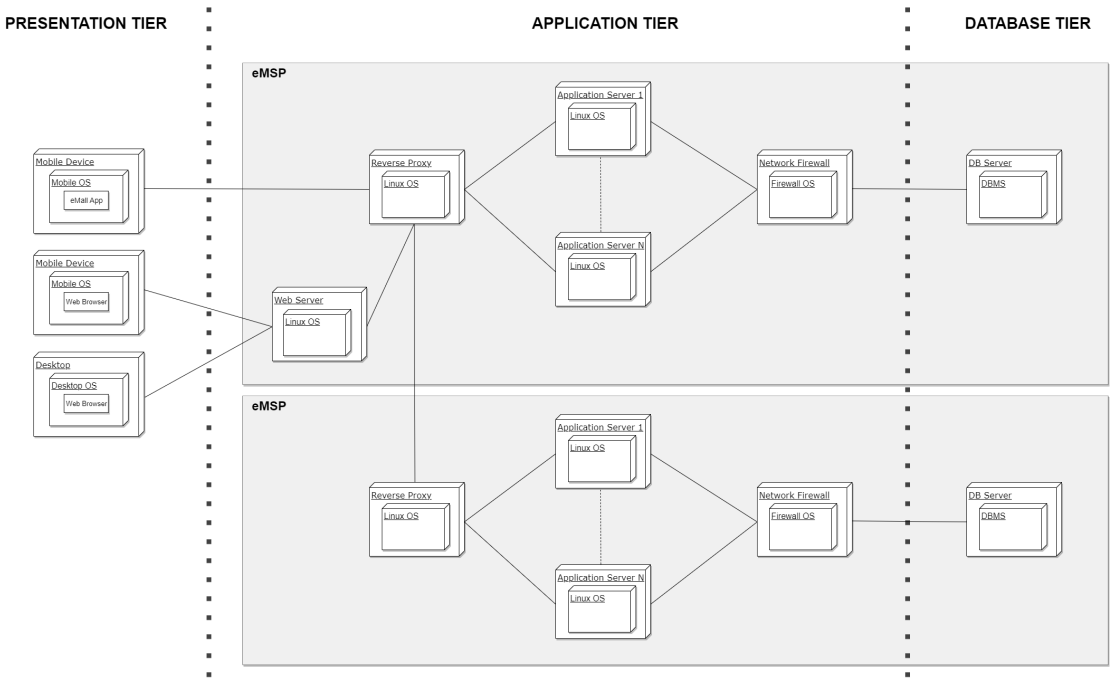

Fig. 4: *Deployment diagram of the system*

- **Clients:** The Clients make up the Presentation tier of the system. They could be Desktop Clients, that use the application through a Web Browser, or Mobile

Clients, that can access the application either through a Web Browser or the eMall mobile app.

- **Web Server:** The Web Server is used to process and present web pages to desktop Clients. The communication between Clients and Server take place using HTTP protocol.
- **Reverse Proxy:** The Reverse Proxy is used for load balancing of requests, distributing them between the various Application Servers of the eMSP and CPMS subsystems. It also increases parallelism, scalability and security of the subsystems.
- **Application Server:** Application Servers contains the business logic of each subsystem.
- **Network Firewall:** The Network Firewall is used to create a DMZ by adding a security layer that is placed before the Database Server. Its presence avoids unauthorized data access.
- **Database Server:** Each Database Server contains the data of the corresponding subsystem of the application.

2.4 Runtime view

2.4.1 Registration Runtime

When a user wants to register to access the functionalities of the app, they fill a form with the required credentials. These are sent through a post request HTTP to the Auth Controller. This component checks that everything sent by the client is in the apposite format (if it is not, it responds with HTTP error code). Then it communicates with the Storage interface which sends a query to the DB to insert the credentials if they have not been already used. It also checks if the information entered about the first vehicle are correct and if the user accepted to use the localization on his device. If some of the inspections failed the AuthController sends a 420 HTTP error code.

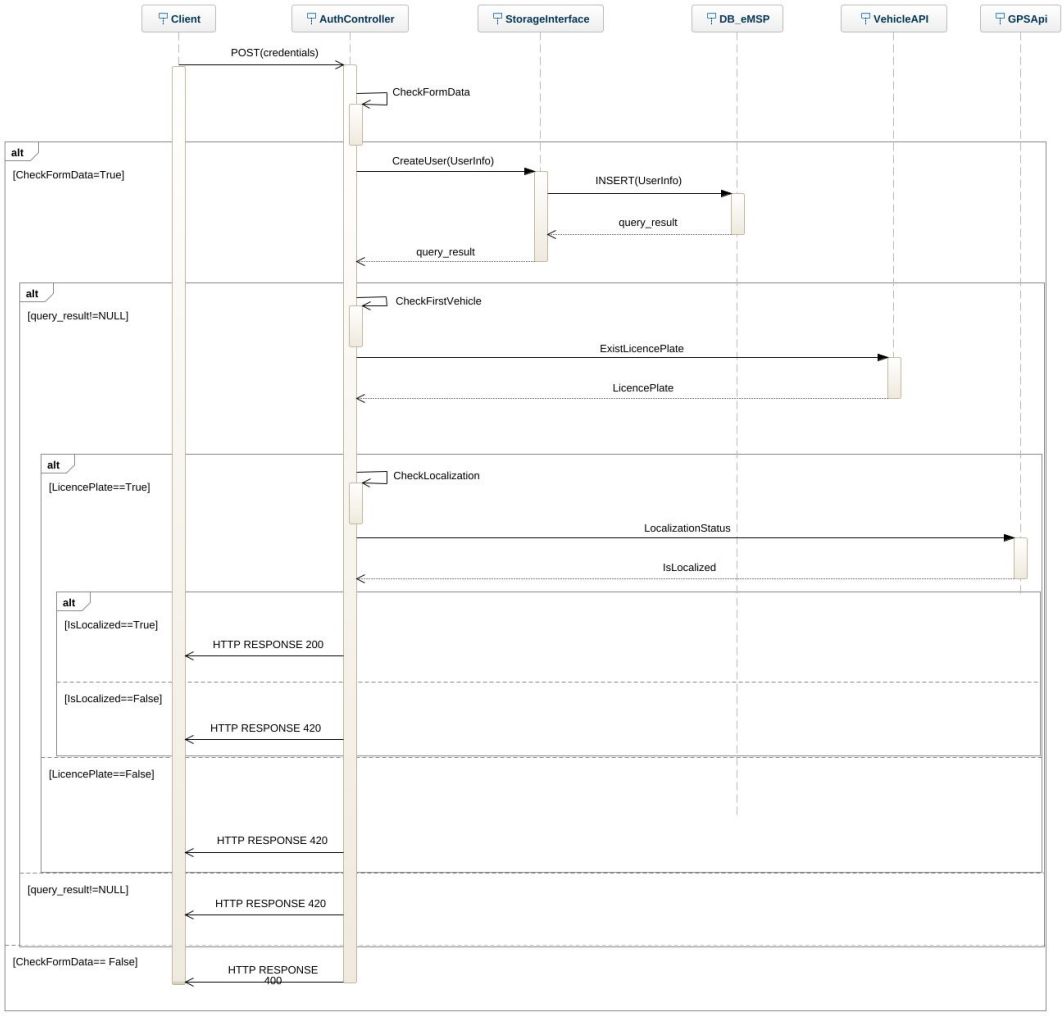

Fig. 5: *Sequence diagram of the registration process for a Driver*

2.4.2 Login Runtime

Once a user is registered, they can log into the app. The login process is handled by the Auth Controller. This receives a put request HTTP from the client and, after checking that the credentials have been sent in the right format, it delegates to the Storage interface the task of creating a query to check if the credentials sent are correct and if the Authorization required is correct. If they are not, the Auth Controller sends a response with an HTTP error code.

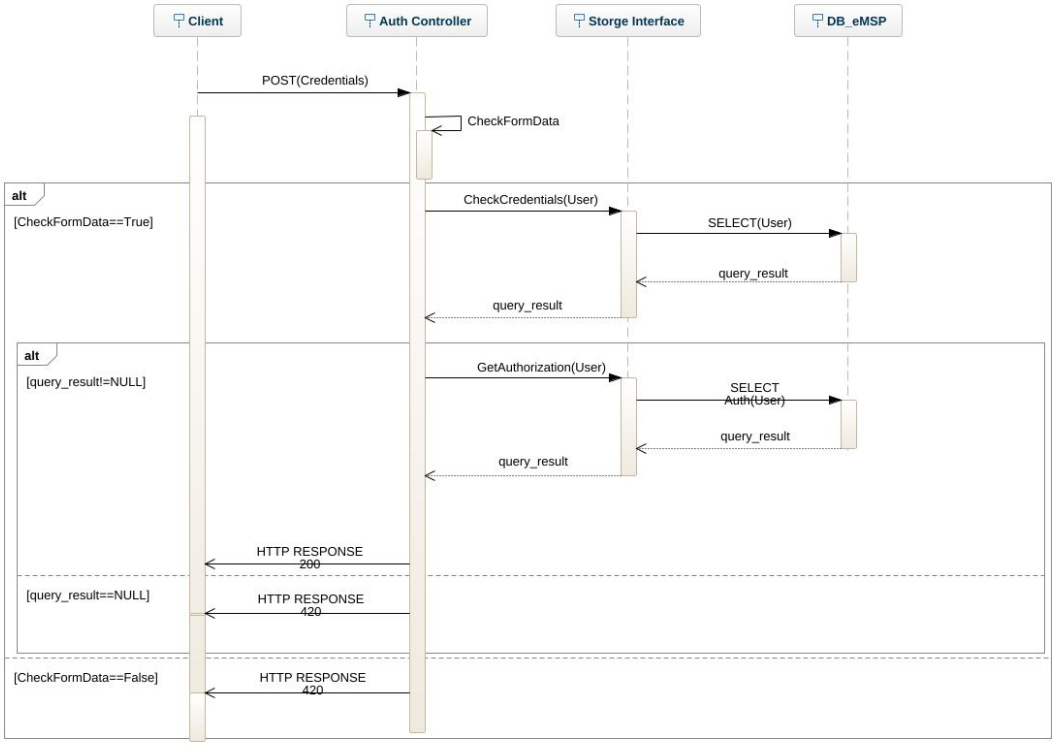

Fig. 6: *Sequence diagram of the login to eMall*

2.4.3 View Map Page

Here is described the process that the System follows when it has been requested the Map Page.

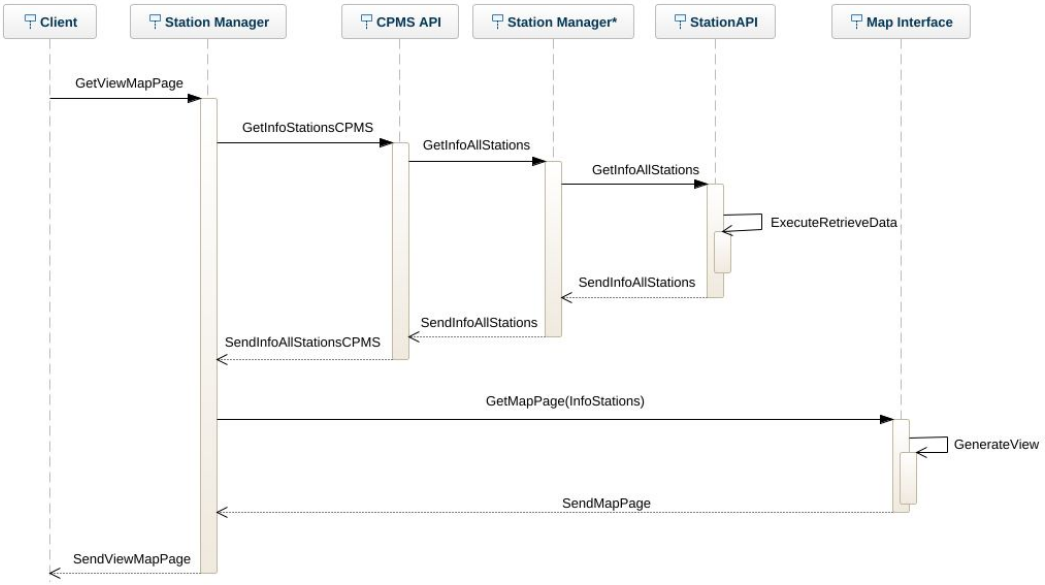

Fig. 7: *Sequence diagram of the visualization of Map Page*

2.4.4 View Stations Page

Here is described the process that the System follows when it has been requested the Stations Page.

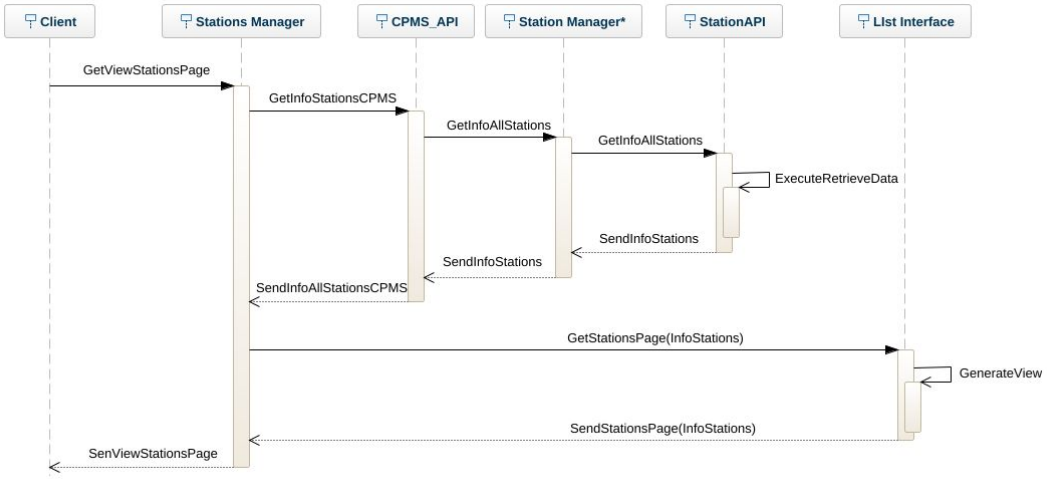

Fig. 8: *Sequence diagram of the visualization of the list in Stations Page*

2.4.5 Change Active Vehicle

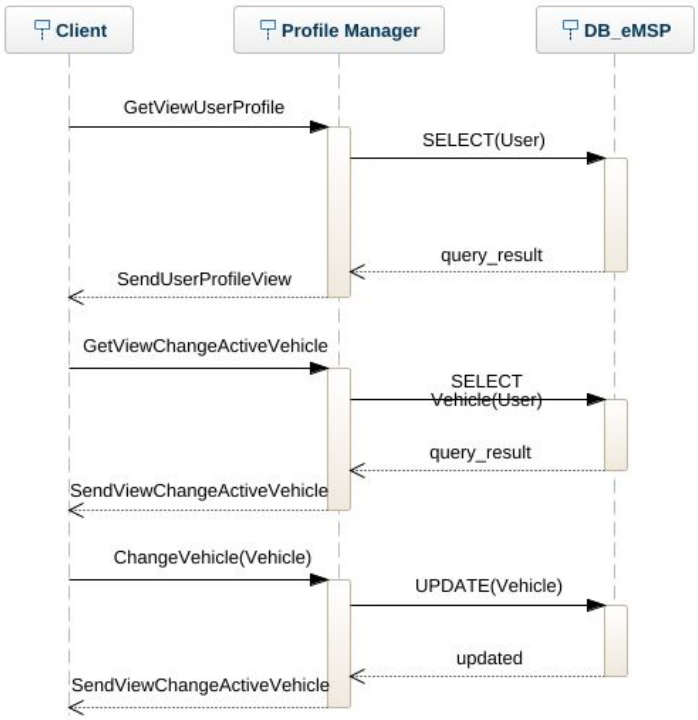

Fig. 9: *Sequence diagram to change the Active Vehicle*

2.4.6 Add Payment Method

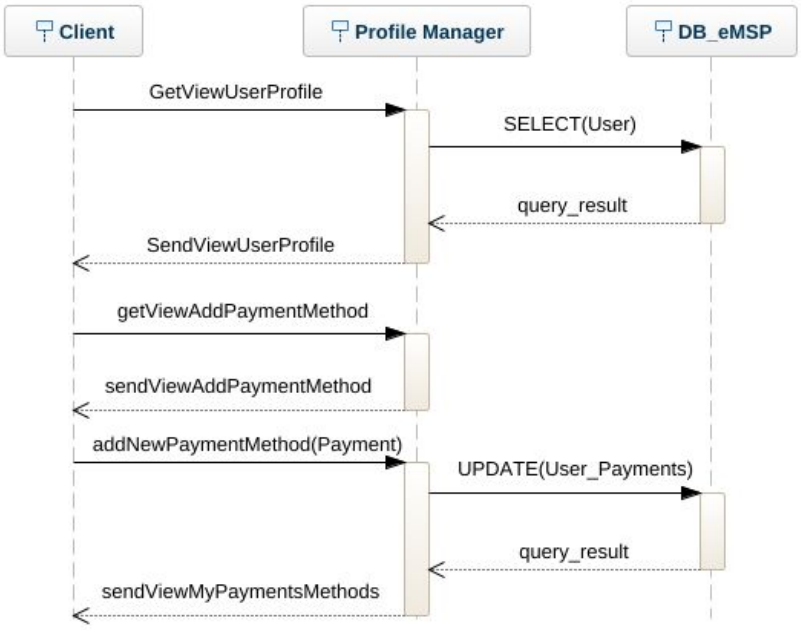

Fig. 10: *Sequence diagram to add a new payment method*

2.4.7 Make a reservation

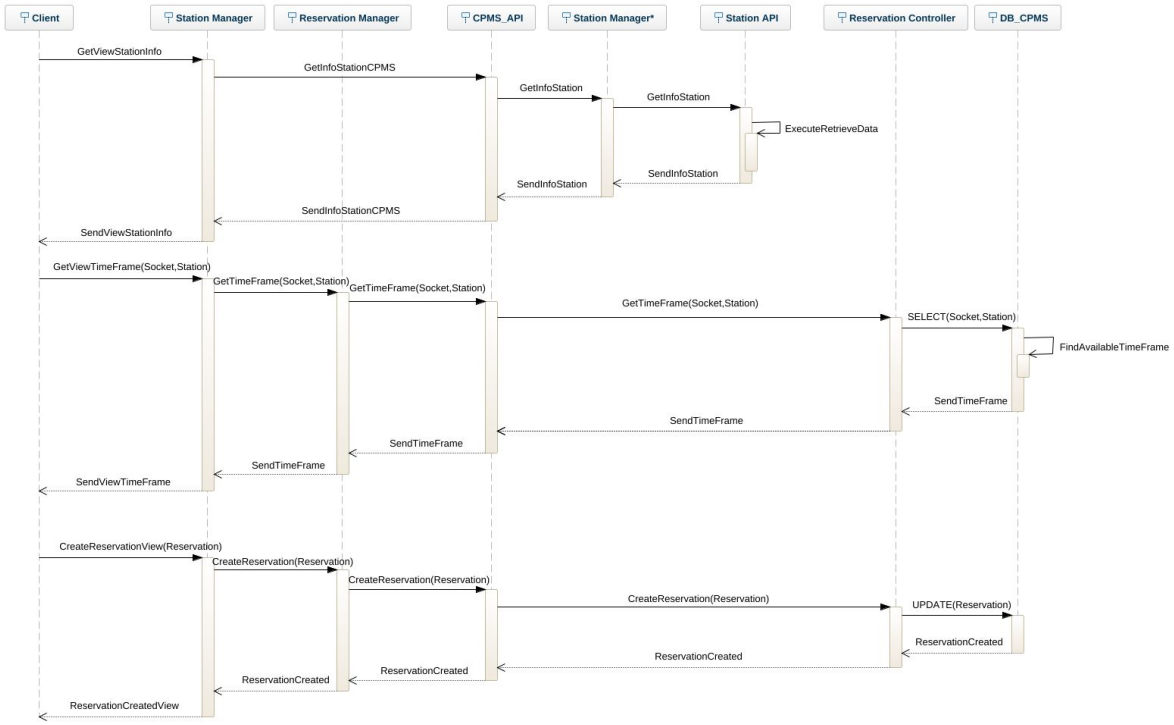

Fig. 11: *Sequence diagram for making a reservation*

2.4.8 Monitor the charge

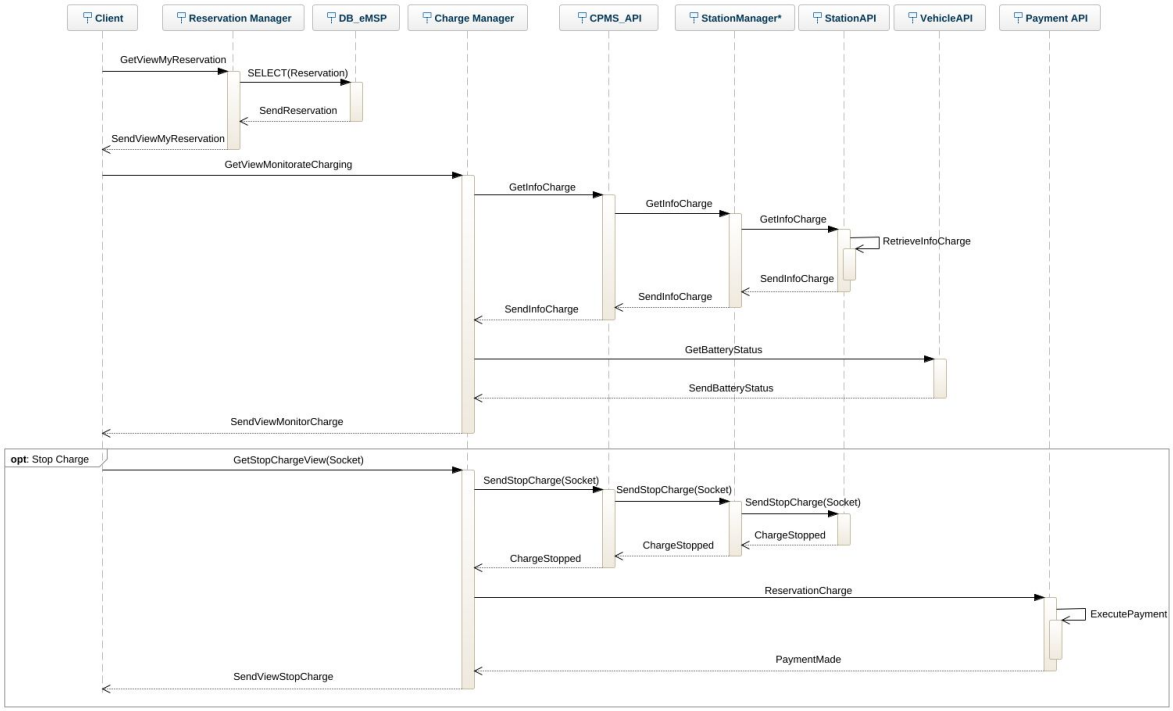

Fig. 12: *Sequence diagram for monitoring the charge*

2.4.9 Start the charge

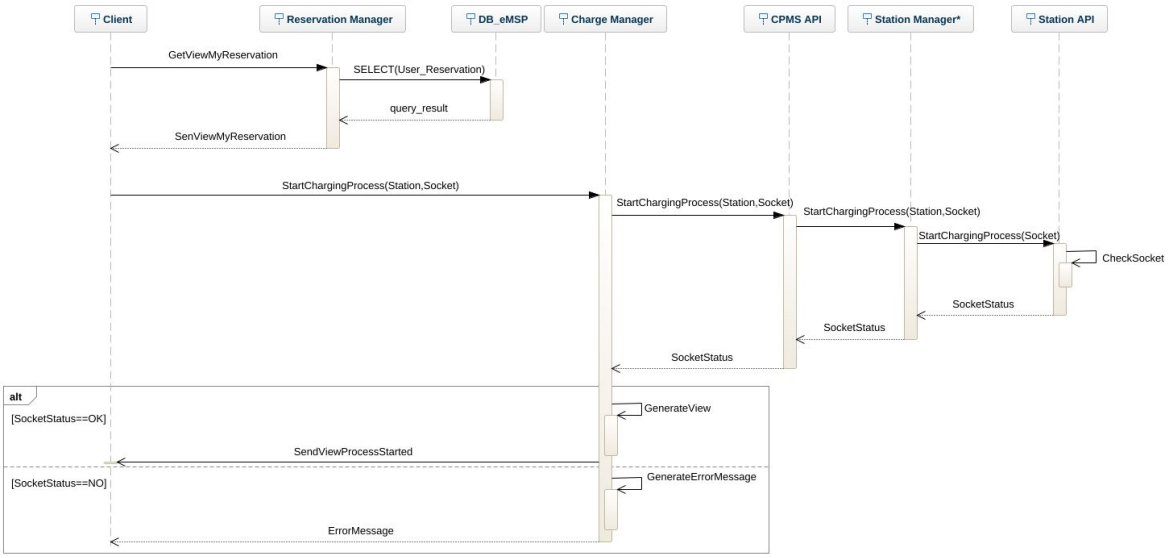

Fig. 13: *Sequence diagram for starting the charge*

2.4.10 View Station Performance

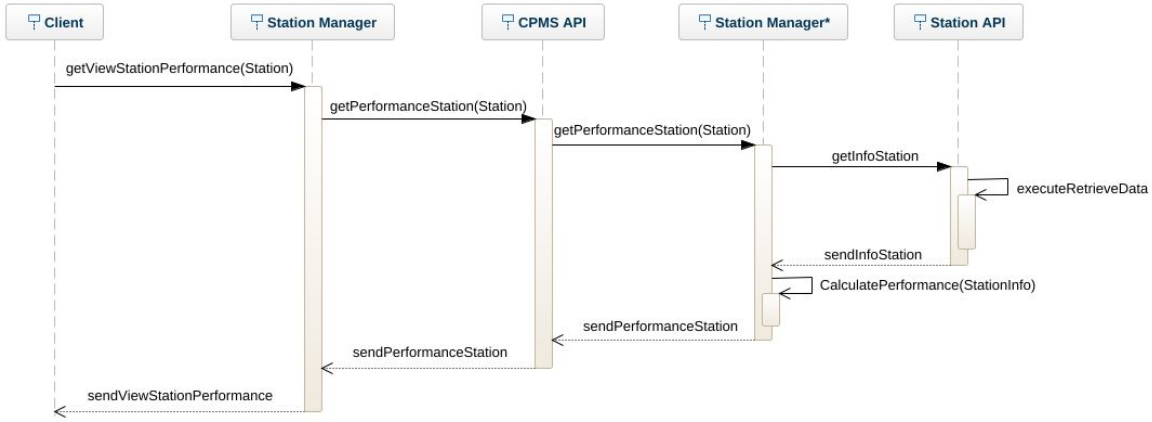

Fig. 14: *Sequence diagram to view the station performance*

2.4.11 Change Energy Mix

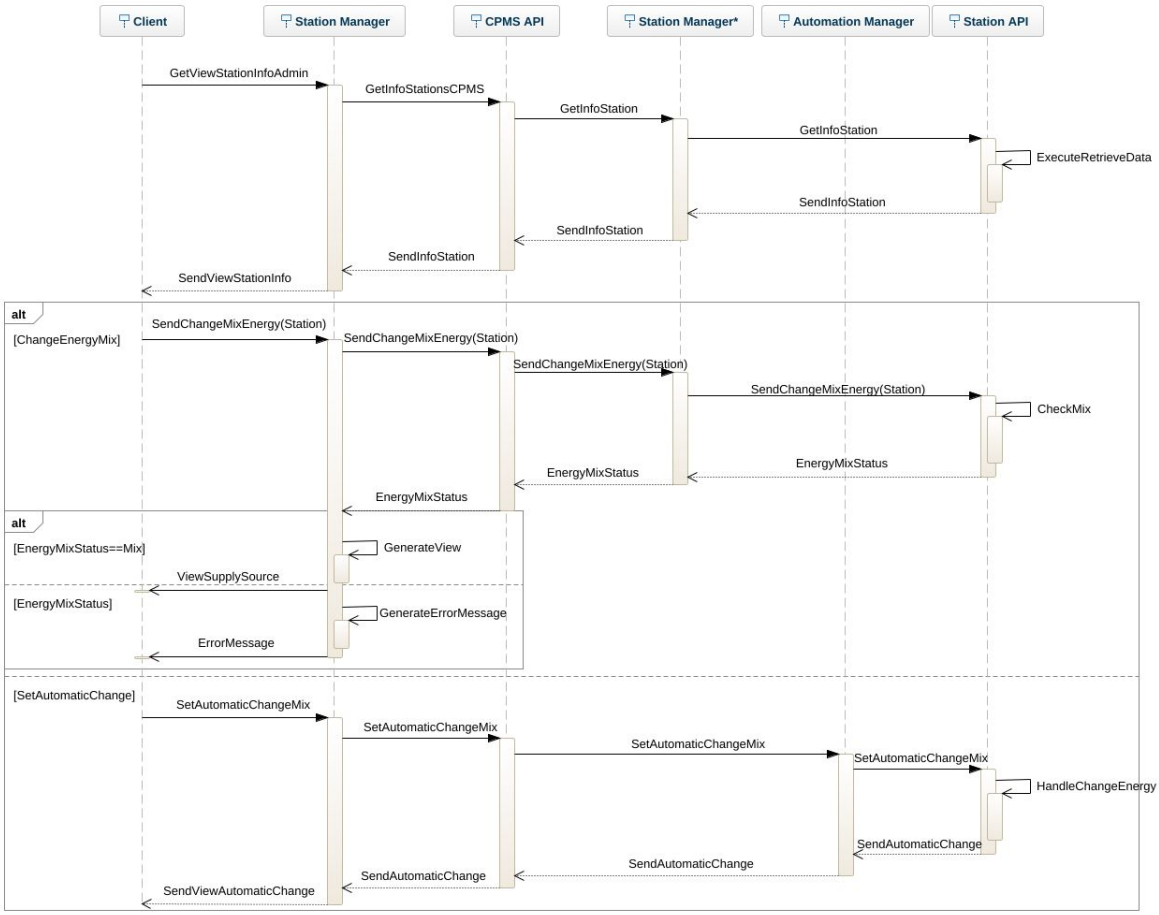

Fig. 15: *Sequence diagram for energy mix settings*

2.4.12 Change DSO

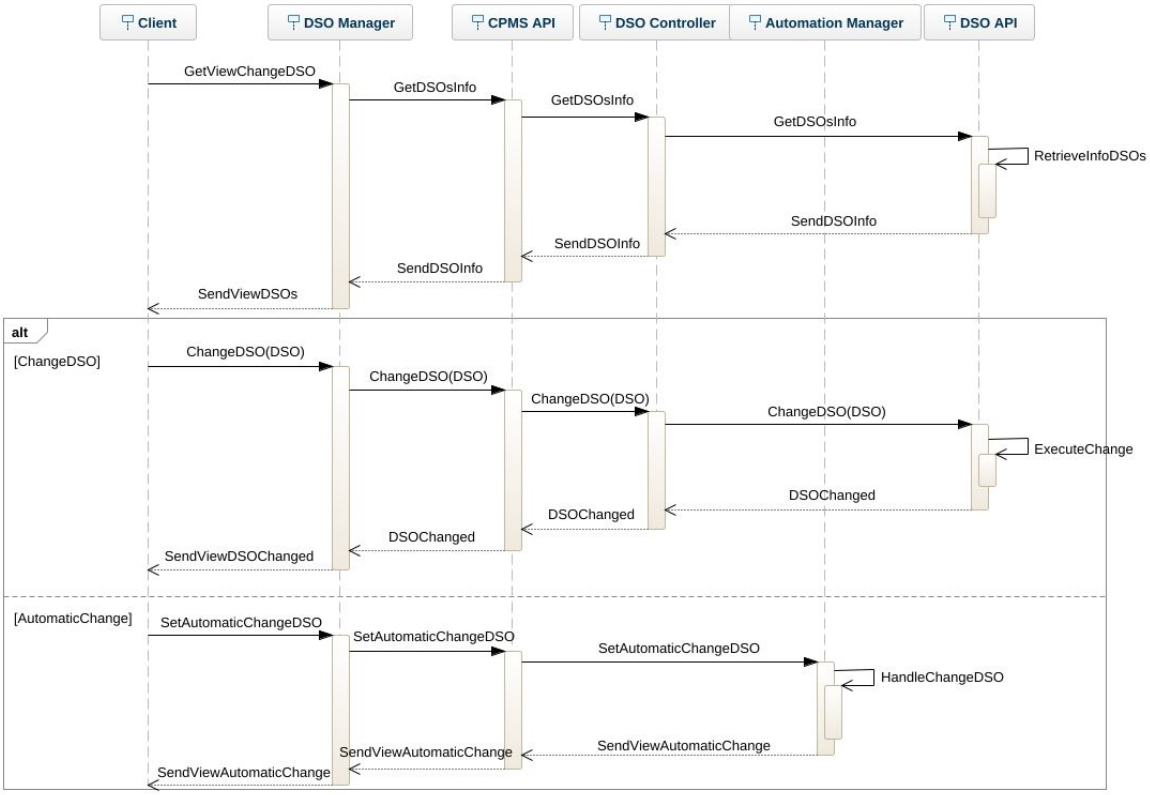

Fig. 16: *Sequence diagram for DSO settings*

2.4.13 Accept a recommendation

The user can accept a recommendation provided by the system. Once the user wants to access the recommendation page the system retrieves all the data about the user, such as his calendar, old reservations, vehicle information and calculate the recommendation. Once the user accept the recommendation the new reservation is created.

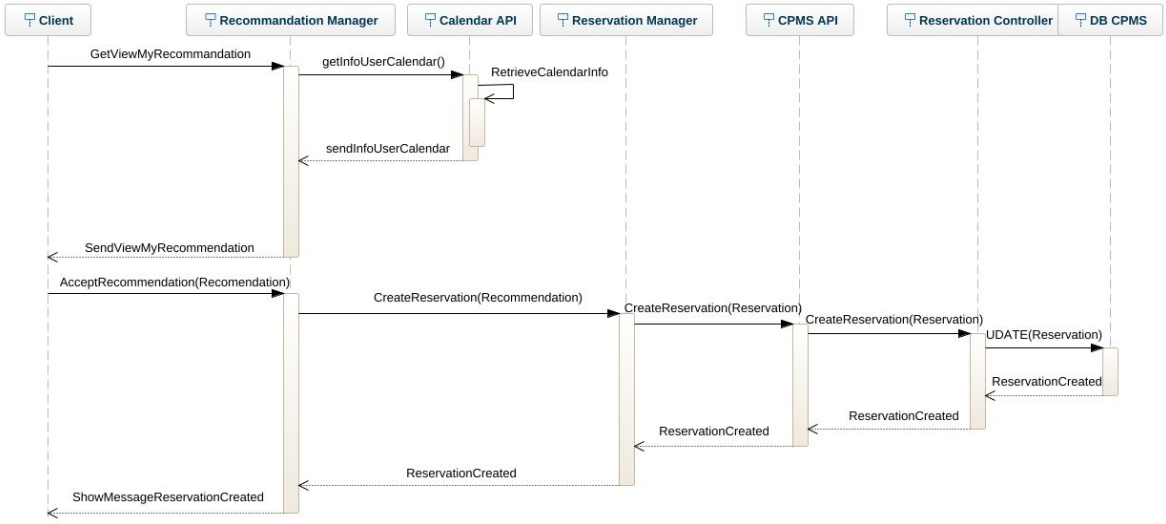

Fig. 17: *Sequence diagram of the acceptance of a recommendation*

2.5 Component interfaces

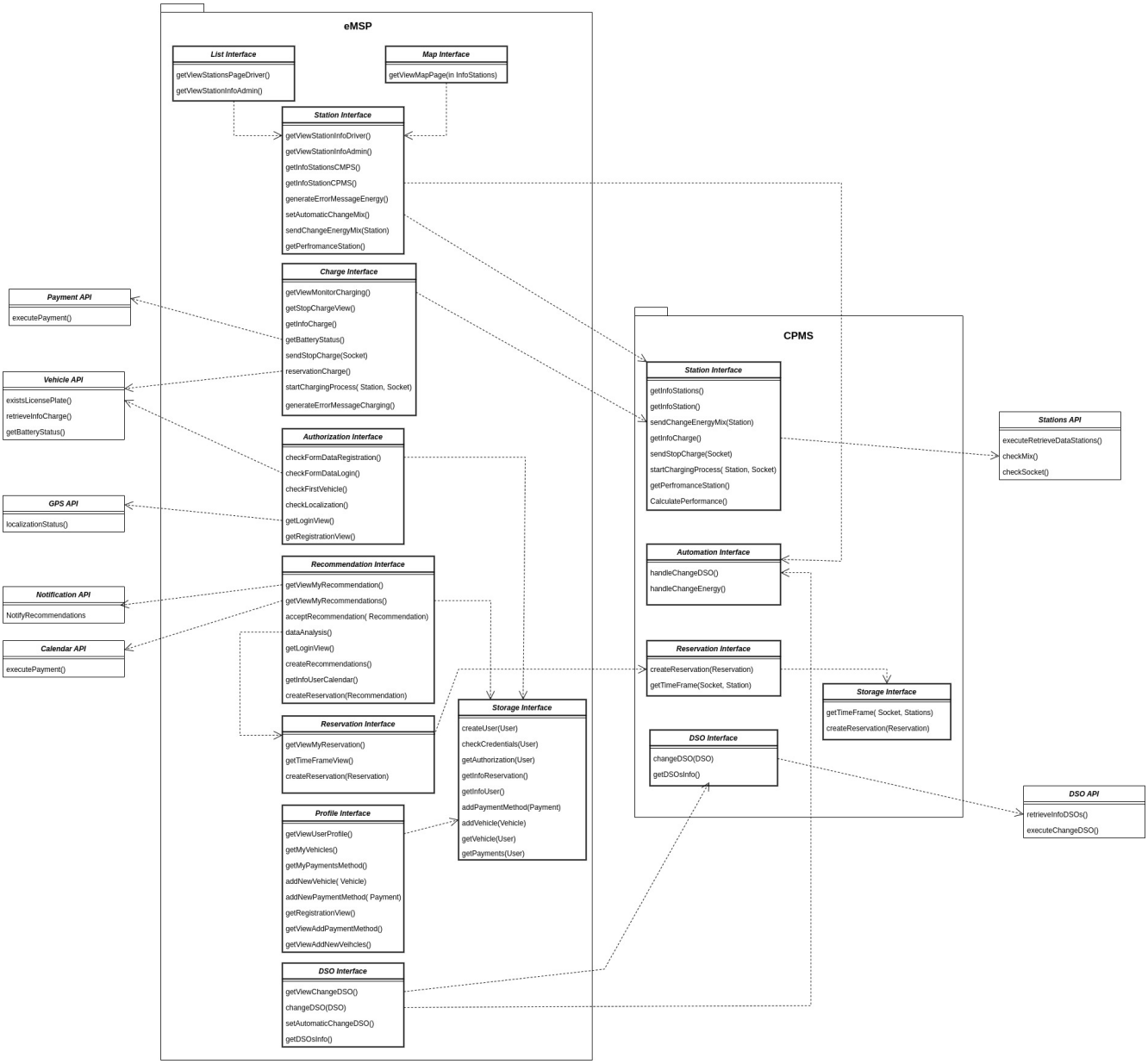

Fig. 18: *Component interfaces diagram*

2.6 Selected architectural styles and patterns

2.6.1 Model View Controller - MVC

Model–view–controller is a software design pattern commonly used for developing software systems. It specifies that the program logic of an application consists of three interconnected elements:

- **Model:** it directly manages data and rules of the application and contains no logic describing how to present the data to a user.
- **View:** it knows how to access the model's data and how to present it to the user, but it does not know what this data means or what the user can do to manipulate it.
- **Controller:** the controller stays between the view and the model, listening to events and executing the appropriate reactions to these events.

Using the MVC methodology allows easy modification of the entire application being every section independent from the others. So, any changes in a certain section of the application will never affect the entire architecture. This, in turn, will help to increase the flexibility and scalability of the application.

2.6.2 Three Tier Architecture

Three-tier architecture is a client-server software architecture pattern in which the user interface, functional process logic , computer data storage and data access are developed and maintained as independent modules. This is achieved through the division of the system in three tiers:

- **Presentation Tier:** it is the top-most tier and consists of the interface which displays information useful to the end user. It is in fact the only tier accessible by the Client.
- **Application Tier:** contains the functional business logic of the application and controls the application's core functionality.

- **Database tier:** comprises of the data storage system. It includes the data persistence mechanisms and the data access layer.

2.6.3 REST Architecture

REpresentational State Transfer, or REST, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other. It is mainly characterized by:

- **Statelessness:** Statelessness mandates that each request from the client to the server must contain all of the information necessary to understand and complete the request. The server cannot take advantage of any previously stored context information on the server. For this reason, the client application must entirely keep the session state.

- **Layered System:** The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior. In a layered system, each component cannot see beyond the immediate layer they are interacting with.

- **Client-Server Pattern:** the client-server design pattern enforces the separation of concerns, which helps the client and the server components evolve independently. By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components. While the client and the server evolve, we have to make sure that the interface/contract between the client and the server does not break.

2.6.4 Reverse Proxy

Reverse proxies help increase scalability, performance, resilience and security of the system, by sitting in front of back-end applications and forwarding client requests to those applications. The resources returned to the client appear as if they originated from the web server itself. A reverse proxy also allows to balance the load between the internal Application Servers. Reverse proxies can keep a cache of static content, which

further reduces the load on these internal servers and the internal network. It can also add features such as compression or encryption to the communication channel used with the clients. Reverse proxy servers are implemented in popular open-source web servers such as Apache and Nginx.

2.7 Other design decisions

2.7.1 OCPI Protocol

OCPI (Open Charging Protocol Interface) is a standardized communication protocol designed to ensure interoperability between different charging platforms. The OCPI protocol allows eMSPs to interact with CPMSs in a standardized way. For example, using OCPI, a service provider can find and reserve a charging station, get information about the charging station's status, and receive updates on the progress of the charging session. OCPI supports communication and file transfer via the HTTP protocol. By providing a common language for communication between service providers and CPOs, OCPI helps to streamline the charging process and provide a more seamless experience for electric vehicle drivers. It also allows different eMSPs and CPOs to work together more efficiently, as they can rely on a standardized set of protocols and data formats.

2.7.2 OCPP Protocol

OCPP (Open Charge Point Protocol) is a communication protocol used for electric vehicle charging stations. It enables charging station manufacturers to integrate their charging infrastructure with a CPMS, which can be used to manage and monitor the charging stations remotely. The OCPP protocol defines a set of messages and operations that can be used to perform various tasks, such as starting and stopping a charging session, requesting charging station status information, and sending fault reports. These messages and operations are based on a request-response model, where the CPMS sends a request to the charging station and the charging station responds with a corresponding response. OCPP is an open standard designed to be interoperable, so that charging stations from different manufacturers can work together and be managed by a common CPMS.

2.7.3 Data retrieval

The system is designed with the idea of retrieving data from external APIs whenever it is needed. In particular, there is a strong dependence of the eMSP subsystem from the CPMS subsystem for the management of the reservations, which in turn, relies on the real time data provided by the Station API and DSO API. Although this approach makes the system more vulnerable, in the sense that external services are required for full functionality, it guarantees that data is always updated in real time. This characteristic is fundamental in order to provide the needed quality of the various services offered to the system's users.

3 User Interface Design

In this section are presented the mockups of the user interface. In particular, it is shown a mobile app view for the Drivers, while for the Administrators it is shown a PC view.

3.1 Driver

3.1.1 Login

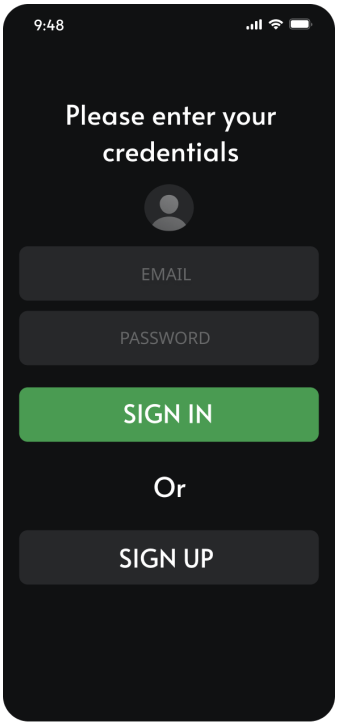

Fig. 19: *UI of the Login Page*

3.1.2 Register

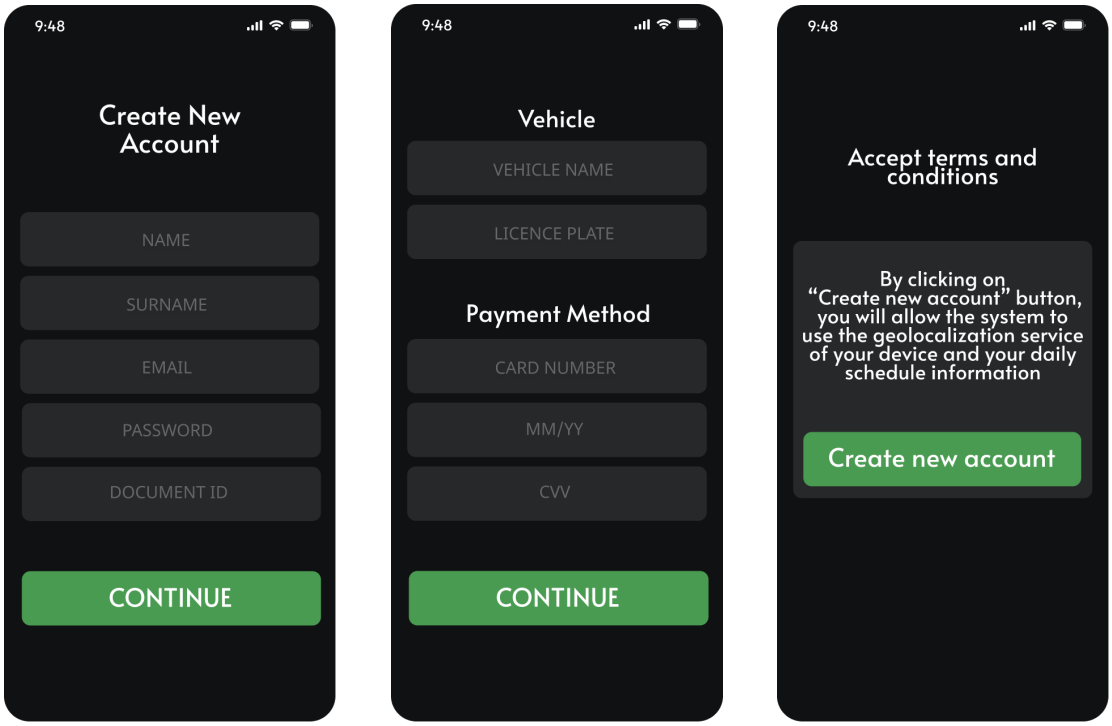

Fig. 20: *UI for registering as a Driver, divided in three separate steps*

3.1.3 User Profile

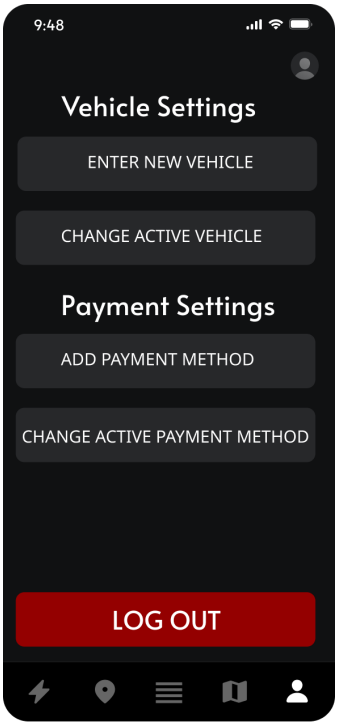

Fig. 21: *UI of the User Profile Page*

3.1.4 Vehicles

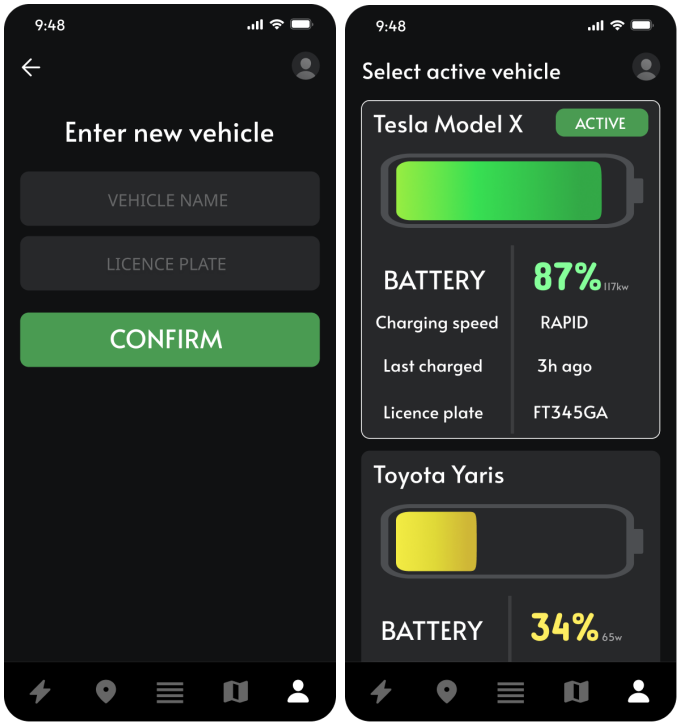

Fig. 22: *UI of Driver entering a new vehicle (on the left) and changing the active vehicle (on the right)*

3.1.5 Payment Methods

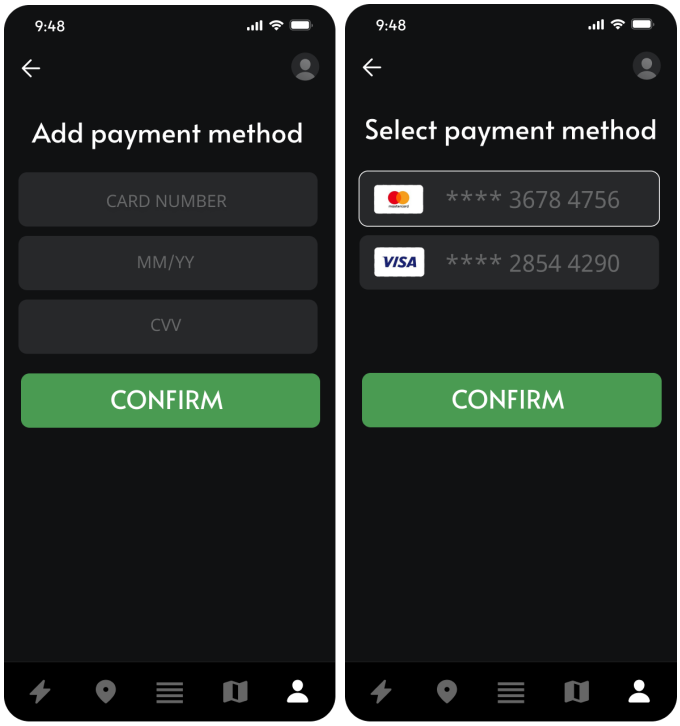

Fig. 23: *UI of Driver adding a new payment method (on the left) and changing the active payment method (on the right)*

3.1.6 Map and Stations

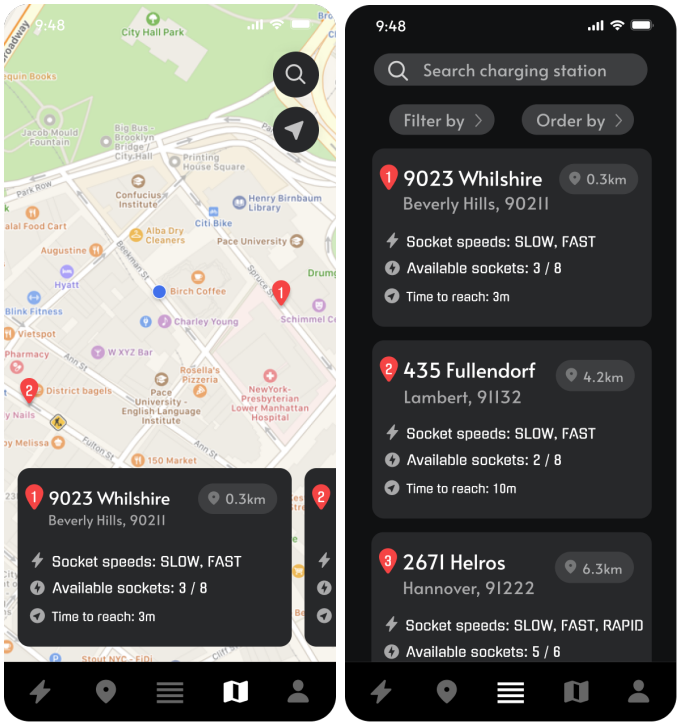

Fig. 24: *UI of Map Page (on the left) and Stations Page (on the right)*

3.1.7 Reservations

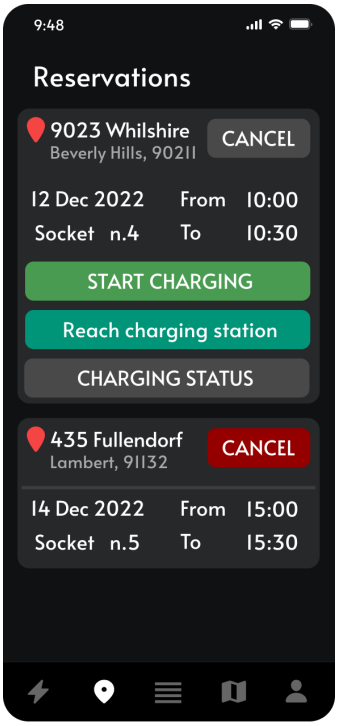

Fig. 25: *UI of the MyReservations Page*

3.1.8 Recommendations

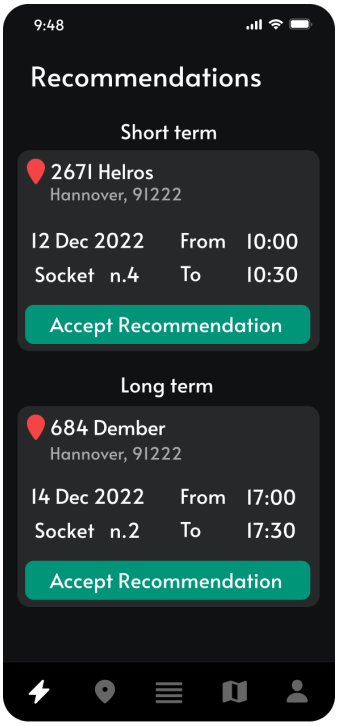

Fig. 26: *UI of the MyRecommendations Page*

3.1.9 Create Reservation

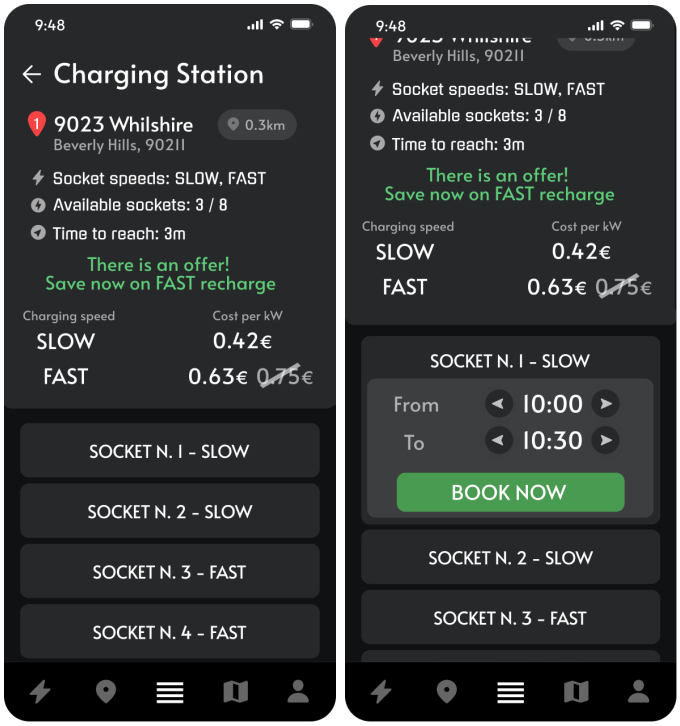

Fig. 27: *UI after selecting a station (on the left) and after selecting a socket (on the right)*

3.1.10 Charging Status

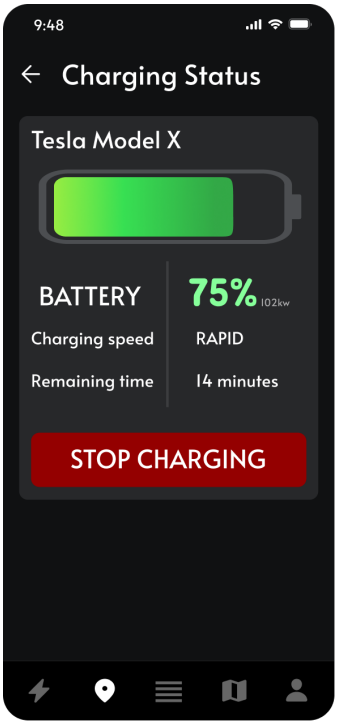

Fig. 28: *UI of the ChargingStatus Page*

3.2 Administrator

3.2.1 Login

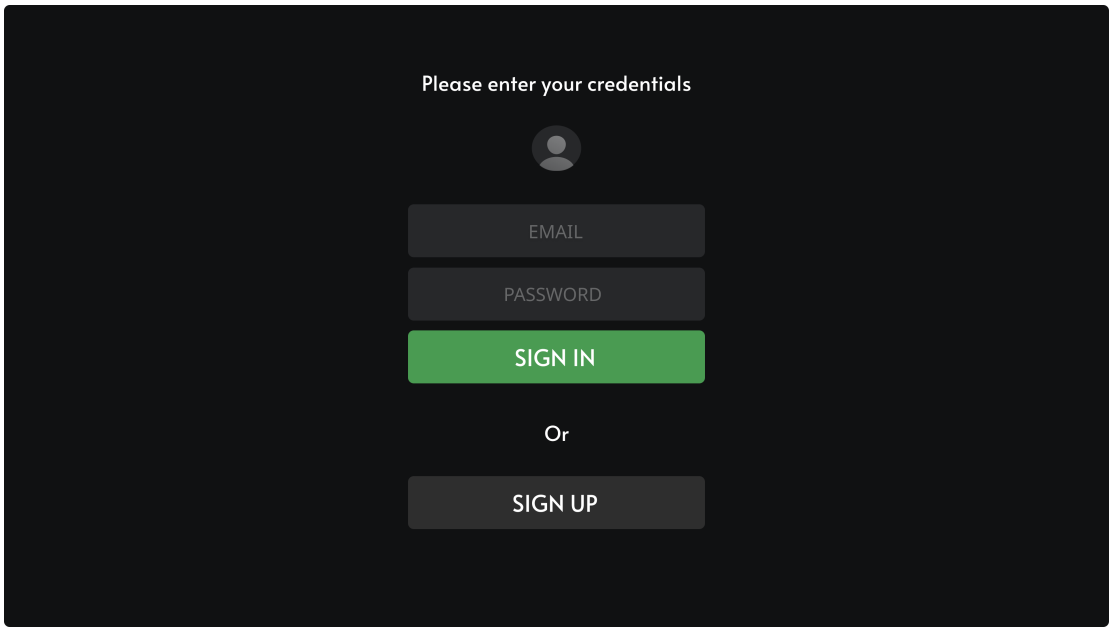

Fig. 29: *UI of the Login Page*

3.2.2 Stations

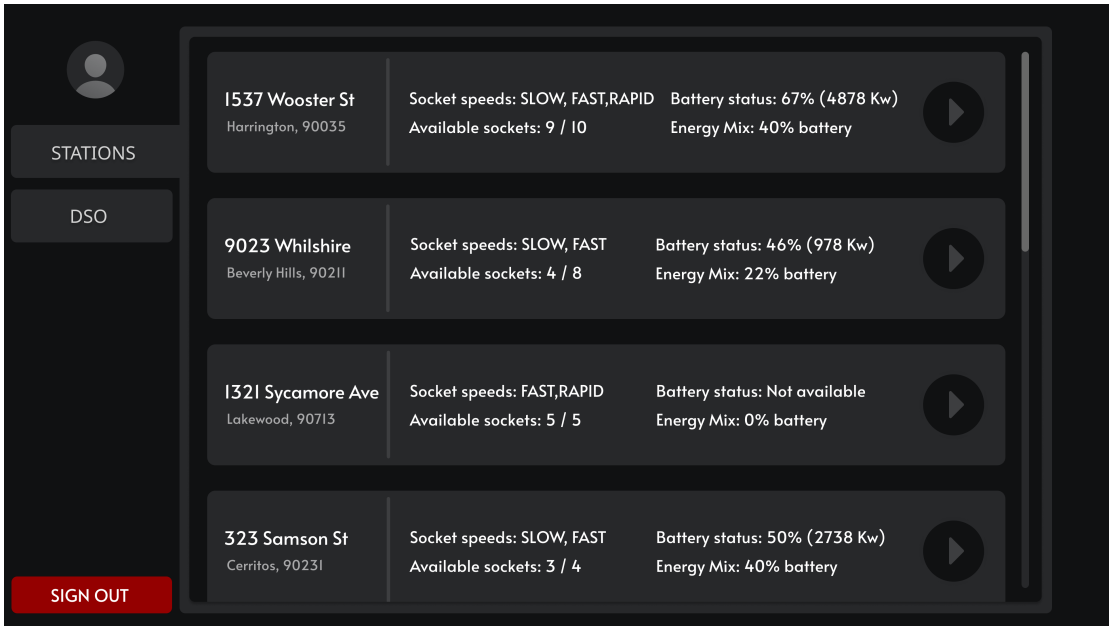

Fig. 30: *UI of the Stations Page*

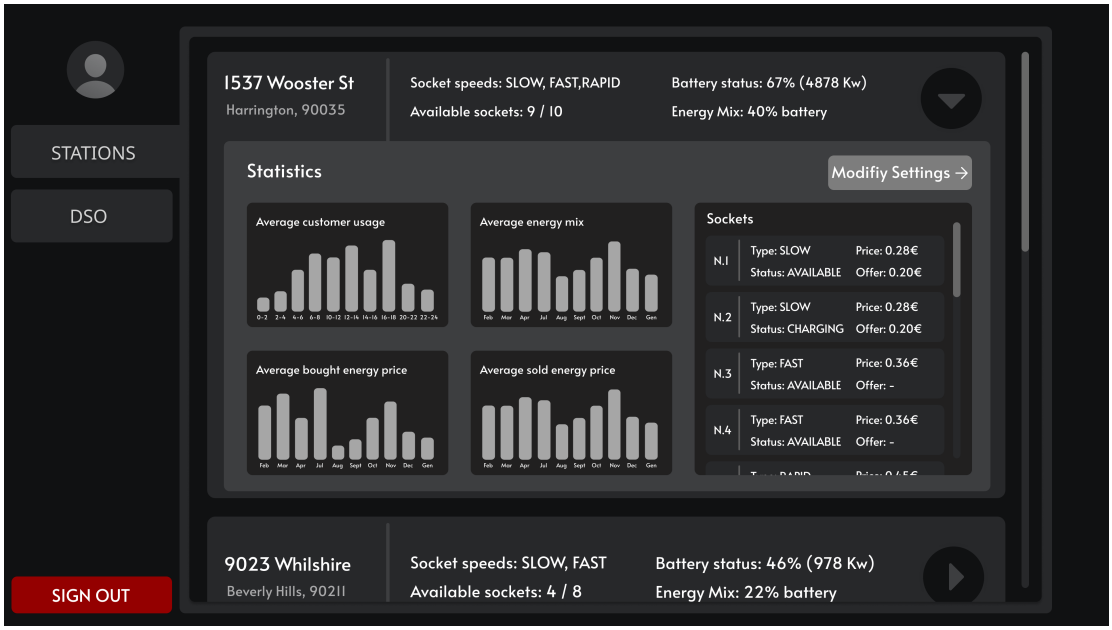

Fig. 31: *UI of the Stations Page after selecting a station*

3.2.3 Station Settings

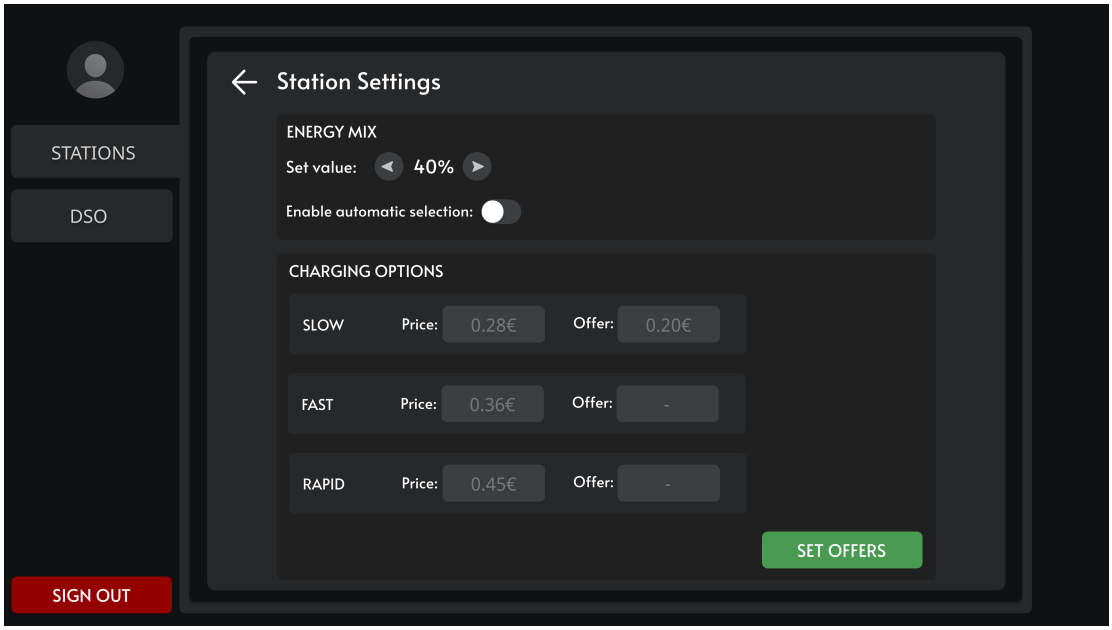

Fig. 32: *UI of the Station Settings Page*

3.2.4 DSO

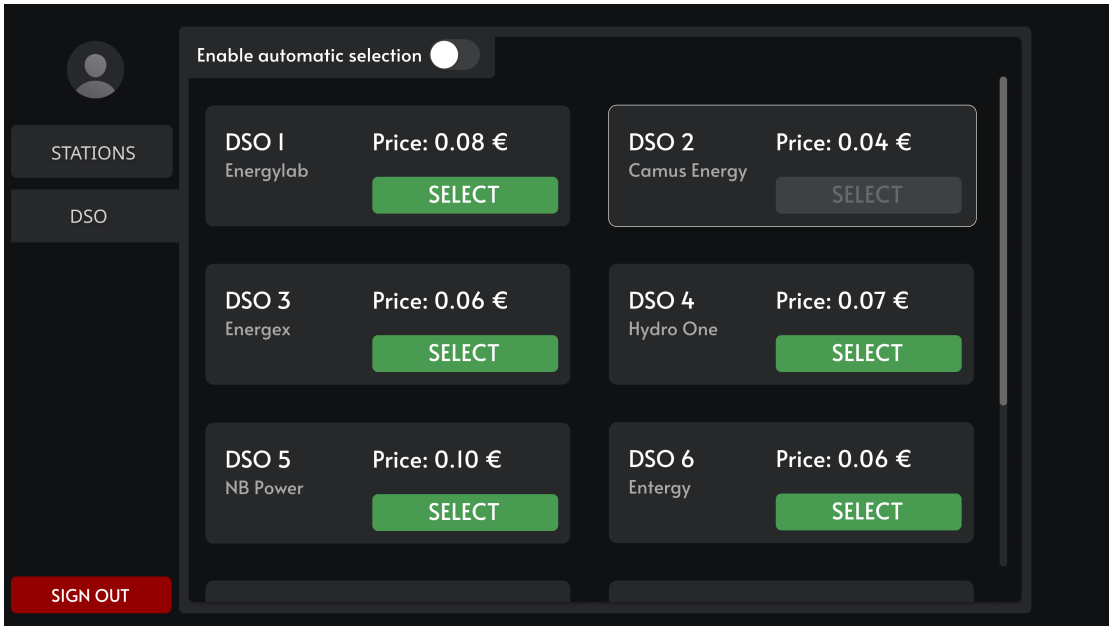

Fig. 33: *UI of the DSO Page*

4 Requirements traceability

This section contains a table explaining what components, according to their abbreviations specified in the list below, are required in order to fulfil each of the requirements specified in the *RASD*.

- **MC** - Mobile Client
- **WC** - Web Client
- **ACC** - Authentication Controller
- **SI** - Storage Interface eMSP
- **SI*** - Storage Interface CPMS
- **DB** - Database eMSP
- **DB*** - Database CPMS
- **PM** - Profile Manager
- **SM** - Station Manager eMSP
- **SM*** - Station Manager CPMS
- **RSM** - Reservation Manager eMSP
- **RSM*** - Reservation Manager CPMS
- **CM** - Charge Manager
- **REM** - Recommendation Manager
- **DM** - DSO Manager eMSP
- **DM*** - DSO Manager CPMS
- **NM** - Notification Manager
- **AM** - Automation Manager
- **EAI** - External API Interfaces eMSP
- **EAI*** - External API Interfaces CPMS

R	MC	WC	ACC	SI	SI*	DB	DB*	PM	SM	SM*	RSM	RSM*	CM	REM	DM	DM*	NM	AM	EAI	EAI*
R1	X	X	X	X		X													X	
R2	X	X		X		X		X											X	
R3	X	X		X		X		X											X	
R4	X	X							X	X									X	X
R5	X	X							X	X									X	X
R6	X	X							X	X									X	X
R7	X	X			X		X		X	X	X	X							X	X
R8	X	X			X		X		X	X	X	X							X	X
R9	X	X		X		X					X									
R10	X	X		X		X			X	X	X								X	X
R11	X	X								X			X						X	X
R12	X	X								X			X						X	X
R13	X	X								X			X				X		X	X
R14	X	X												X			X		X	X
R15	X	X												X			X		X	X
R16	X	X			X		X		X	X	X	X							X	X
R17		X		X	X	X	X		X	X									X	X
R18		X		X	X	X	X		X	X									X	X
R19		X		X	X	X	X		X	X									X	X
R20		X		X	X	X	X		X	X									X	X
R21		X		X	X	X	X		X	X					X	X		X	X	X
R22		X		X	X	X	X		X	X					X	X			X	X
R23		X		X		X			X											
R24																	X		X	
R25																	X		X	
R26				X	X	X	X												X	X
R27	X	X	X		X		X													
R28	X	X	X		X		X													

5 Implementation, Integration and test Plan

5.1 Implementation

The implementation of this system would follow a planned structure, dividing the components into various groups in order to maintain a clear development path.
The components can be divided into the following categories:

- **Frontend components**: Mobile Client, Web Client.
- **Backend components**: Authentication Controller, Storage Interface eMSP, Storage Interface CPMS, Notification Manager, Profile Manager, Station Manager eMSP, Station Manager CPMS, Reservation Manager eMSP, Reservation Manager CPMS, Charge Manager, Recommendation Manager, DSO Manager eMSP, DSO Manager CPMS, Automation Manager.
- **External components**: Database eMSP, Database CPMS, eMSP API, CPMS API, Map API, Vehicle API, Station API, DSO API, GPS API, Payment API, Notification API.

In order to implement, integrate and test the System, a bottom-up strategy will be used. As previously explained, to support this strategy, the system has been divided

into smaller subsystems in order to implement and test their components separately. From the implementation and test of each subsystem it is possible to obtain a reliable and modular product. External APIs are supposed to be reliable and without the necessity to be tested. The development process would follow this path:

1. Implementation and integration of different components of the same subsystem.
2. Integration of different subsystems (client and server application).

The subsystems (and the relative components) to be developed are:

- **Storage subsystem**: Storage Interface eMSP and Storage Interface CPMS.
- **Station subsystem**: Driver Station Interface, Driver Station Controller, Admin Station Interface, Admin Station Controller, Map Manager, List Manager, Station Controller and Station Settings.
- **DSO subsystem**: DSO Interface, DSO Controller eMSP and DSO Controller CPMS.
- **Automation subsystem**: Automation Controller, Automation Station, Automation DSO.
- **Reservation subsystem**: Reservation Interface, Reservation Controller eMSP, Reservation Controller eMSP.
- **Charge subsystem**: Charge Interface and Charge Controller.
- **Recommendation subsystem**: Recommendation Interface, Recommendation Controller and Data Analytic.
- **User subsystem**: Profile Interface, Profile Controller, Authorization Controller.
- **Notification subsystem**: Notification Manager.
- **Client**: Web Client and Mobile Client.

Moreover, the final subsystem integration that will be performed is between Business Logic and Client subsystem. It's important that the verification and validation processes starts as soon as the development of the system begins in order to find errors as quickly as possible. As specified before, will be used an incremental approach for integration process in order to isolate bugs in single subsystems and don't propagate them. Each component of each subsystem is tested using Unit Testing.

5.2 Integration & Test plan

The first subsystem implemented is the backend. It contains all the subsystems previously introduced except for the Client. All the components are implemented and unit tested in parallel, following the group division previously described.
Each group is integrated and tested with the required external services, according to the Component View path, to check the behavior of the components when they work together.
Once finished this phase, components are integrated and will be performed integration testing.
Finally, the frontend will be integrated and tested with the backend. In particular the subystem to be integrated with the Client Application are User subsystem, Station subsystem, Reservation subsystem and Recommendation subsystem.

The following diagrams describe the process of implementation, integration and testing. The CPMS part of each subsystem has to be developed before the complementary eMSP one. A subsystem is considered completed after the verification of the correct integration between the two parts previously mentioned.
Before the final system integration, each backend subsystem is tested with the others it requires to work properly.

- **Backend**

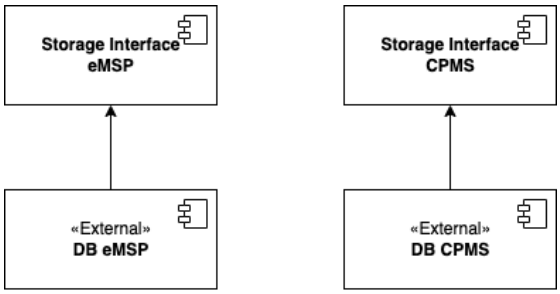

Fig. 34: *Storage subsystem*

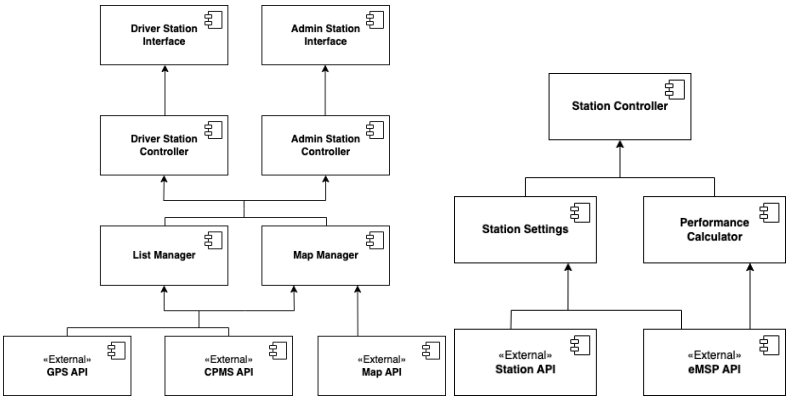

Fig. 35: *Station subsystem*

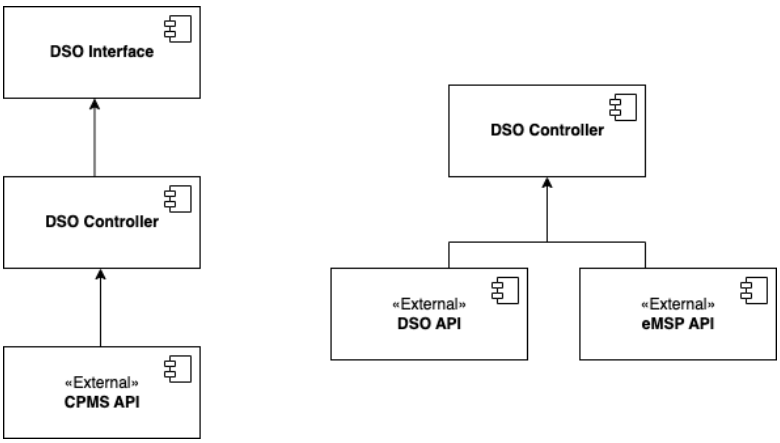

Fig. 36: *DSO subsystem*

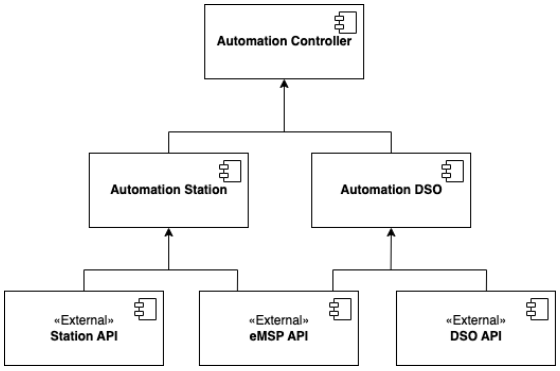

Fig. 37: *Automation subsystem*

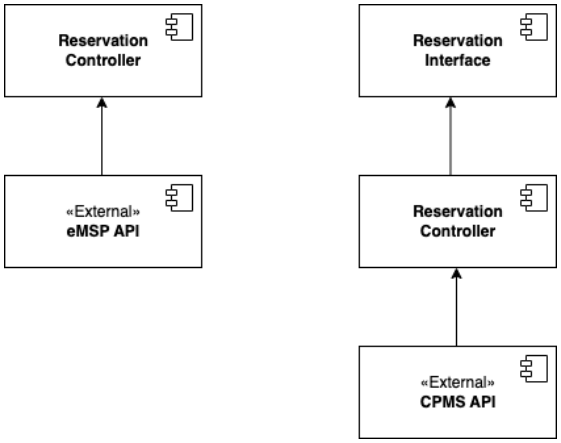

Fig. 38: *Reservation subsystem*

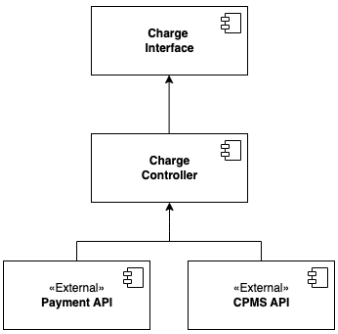

Fig. 39: *Charge subsystem*

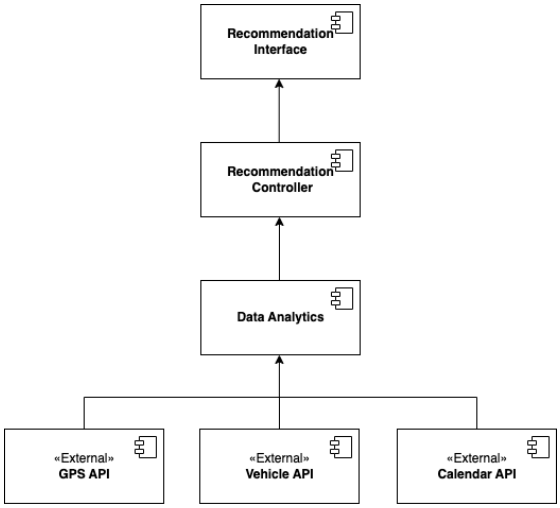

Fig. 40: *Recommendation subsystem*

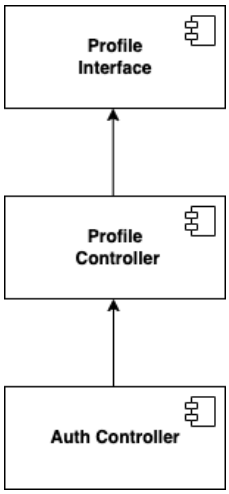

Fig. 41: *User subsystem*

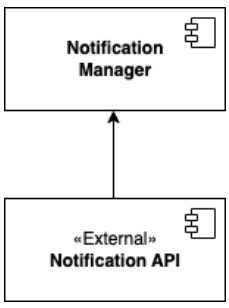

Fig. 42: *Notification subsystem*

- **Client**

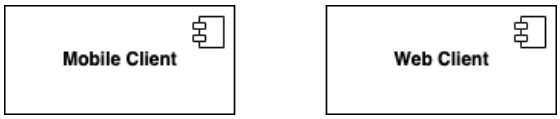

Fig. 43: *Client subsystem*

- **System Integration**

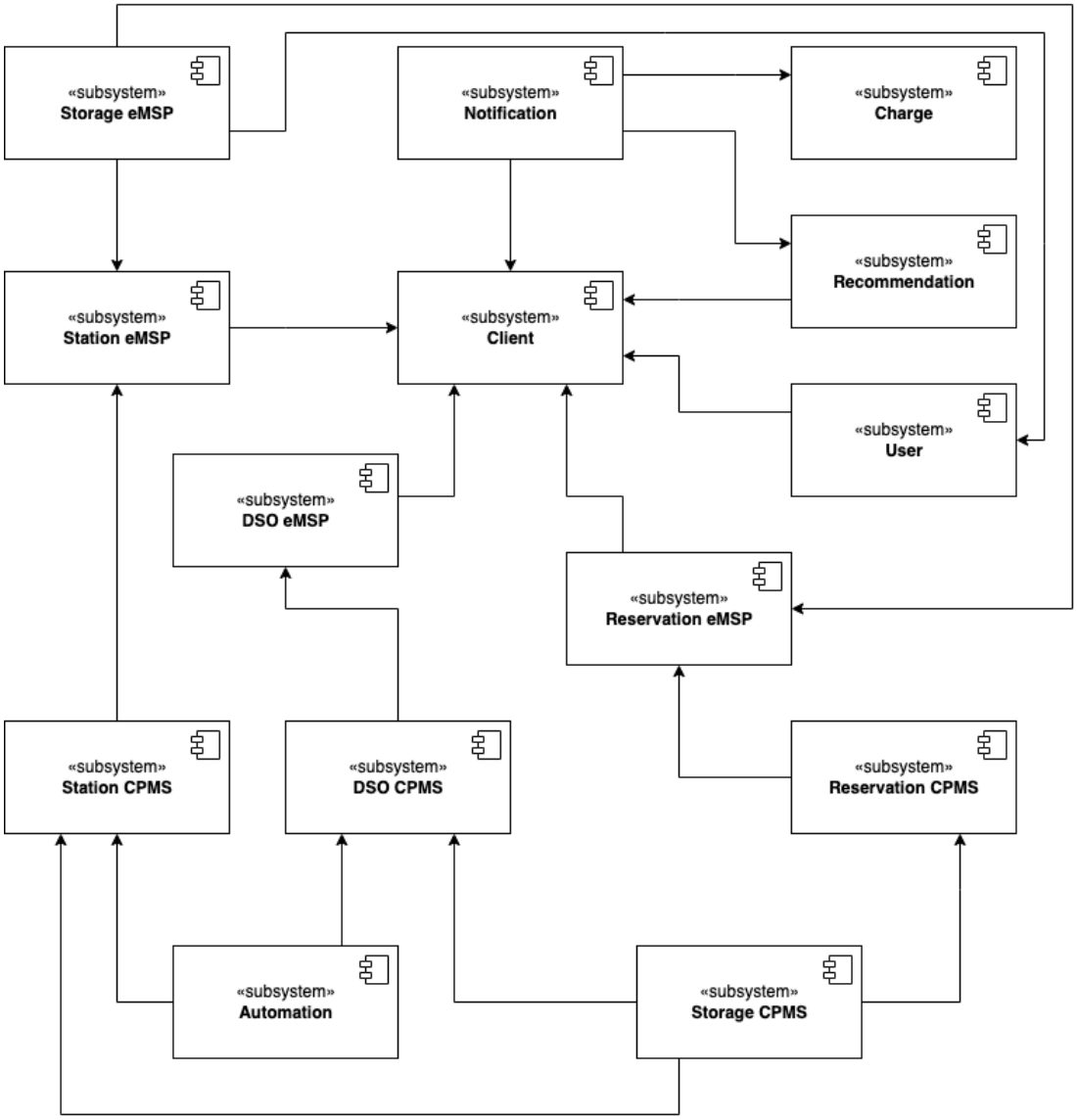

Fig. 44: *System integration*

6 Effort Spent

6.0.1 Roberto Cialini

Section	Time spent
Introduction	1
Architectural design	10
User Interface Design	12
Requirements Traceability	1
Implementation, Integration and Test Plan	1
Reasoning	7
Total time	**32**

6.0.2 Umberto Colangelo

Section	Time spent
Introduction	2
Architectural design	7
User Interface Design	3
Requirements Traceability	7
Implementation, Integration and Test Plan	10
Reasoning	7
Total time	**36**

6.0.3 Vittorio La Ferla

Section	Time spent
Introduction	1
Architectural design	20
User Interface Design	2
Requirements Traceability	1
Implementation, Integration and Test Plan	1
Reasoning	7
Total time	**32**

7 References