POLITECNICO
MILANO 1863

eMall System

e-Mobility for All

DD

Design Document

Version 2.0 - 07/01/2023

Alireza Yahyanejad - 10886993

Mohammad Hosein Behzadifard – 10880732

Alireza Azadi - 10888648

Table of Contents

Figures

Tables

1. INTRODUCTION

1.1 Purpose

This document aims to provide more technical information about the eMSP application and its interaction with different components of the eMSP system and the CPMS system of charging stations. In fact, developers of the application use this document as a guide to redevelop the application. In general, the main different features of this document are:

- The high-level architecture
- Main components of the system
- Interfaces provided by the components
- Design patterns adopted
- Implementation, integration, and testing

1.2 Scope

eMall is a system that has two kinds of applications. One of them is eMSP, developed for end end users, and another is CPMS, implemented for CPOs. CPOs installed and maintained charge stations in charging stations. Also, they can own and operate a set of charging stations or use them for third parties. Some companies made some DSOs that are extra batteries, and they can be used instead of batteries for charging stations. More detailed information can be found on the RASD document.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Table 1 Definitions

Definition	Description
Notification	A message is shown to the user by the system when she/he must be notified about something (ex: the end of the charging process will be shown to him/her).

External Status	Number of charging sockets available, their type such as slow/fast/rapid, their cost, if all sockets of a certain type are occupied, the estimated amount of time until the first socket of that type is freed
Internal Status	Amount of energy available in its batteries, if any, number of vehicles being charged, and for each charging vehicle, amount of power absorbed, and time left to the end of the charge
Time Frame	A specified period in which something occurs or is planned to take place
Valid QR code	A QR code is defined as valid in 5 minutes after the end user wants to pay for the services

1.3.2 Acronyms

Table 2 Acronyms

Acronyms	Description
CPO	Charging Point Operator
CPMS	Charge point Management System
DSO	Distribution System Operator
eMSP	e-Mobility Service Provider
eMall	e-Mobility for All
GPS	Global Positioning System
API	Application Programming Interface
IT	Information Technology
HTTP	Hyper Text Transfer Protocol
XML	Extensible Markup Language
JSON	JavaScript Object Notation
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
RMI	Remote Method Invocation
TLS	Transport Layer Security
GC	Global Cache
RAC	Real Application Cluster

1.3.3 Abbreviations

Table 3 Abbreviations

Abbreviations	Description

G	Goal
R	Requirement
C	Component

1.4 Revision history

Table 4 Revision history

Version	Date	Modification
1.0	22/12/2022	First version
2.0	07/01/2023	• Update runtime view • Add section 6

1.5 Reference Documents

- Specification Document: "Assignment RDD AY 2022-2023.pdf"
- Course slides
- https://evroaming.org/app/uploads/2021/11/OCPI-2.2.1.pdf

1.6 Document Structure

- *Section 1*
  Brief description of DD and introduction of purpose and scope. It also includes definitions, acronyms, and abbreviations.

- Section2
  The main part of DD is this section that contains architectural design choices, including all the components, and the interfaces used for the development of the application. Also, it contains a deployment view and a runtime view. In the end, is explained the architectural patterns chosen with the other design decisions.

- Section3
  Contains how should be the user interface on mobile applications.

- Section4
  Contains the traceability matrix that shows which components satisfy certain requirements.

- Section 5
  The implementation plan, integration plan, and testing plan show how these plans are executed.

- Section 6
  Shows how much time is spent by each member of the group.

- Section 7
  Includes the reference documents.

2. ARCHITECTURAL DESIGN

2.1 Overview

The architecture of this application is structured by 3 logic layers:

1) Presentation Layer: The presentation Layer provides means of Input, allowing the users to manipulate the system Output, allowing the system to produce the results of user manipulation SAP is having Graphical User interface (SAP GUI). The SAP GUI is installed on Individual machines which act as a presentation layer.

2) Application Layer: In this layer business logic is executed. The application layer can be installed on one machine, or it can be distributed among more than one system.

3) Database Layer: The database layer holds the data. SAP supports any relational database. SAP does not provide any database. But it supports any RDBMS. The database layer must be installed on one machine or system. Major databases which are being used in SAP implementations are Oracle and DB2.

2.1.1 High-level components and their interaction

The chosen hardware architecture for this application is Three-Tier. The architecture contains three application layers. A tier to interface with the client, the second tier for the application level, and another server for database management. This division is efficient because the server tier can make a permanent connection with the DBMS, which is less expensive. As well as, having a middle tier between the user and the server can guarantee more security to the access control of the database from the users.

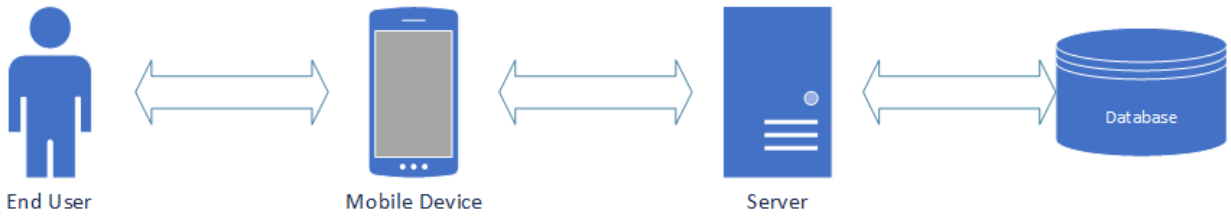

Figure 1 Three-tier architecture for user

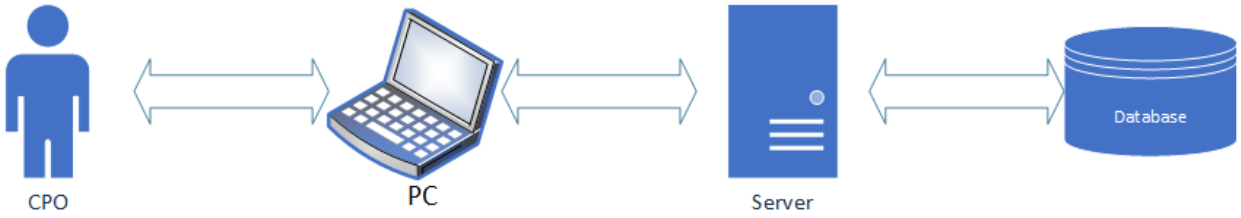

Figure 2 Three-tier architecture for CPO

The following figure represents our system architecture, its high-level components, and their interaction. The client could use this application on a smartphone or a computer. Then, user interactions will be processed by the application server. As well as an application server connected to the database. Finally, the application server is allowed to use APIs for Google Maps, Google calendar, and interaction between eMSP and CPMS.

Figure 3 does not include all details about the architecture of the application and in the next section, more details are represented.

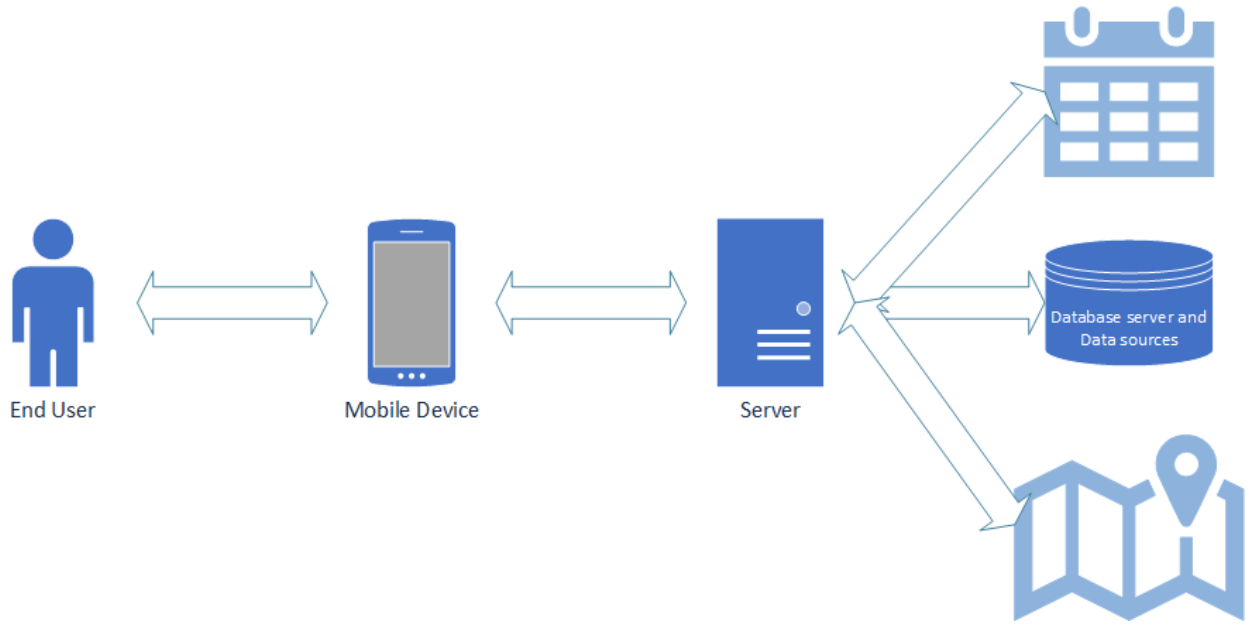

Figure 3 System architecture

2.2 Component view

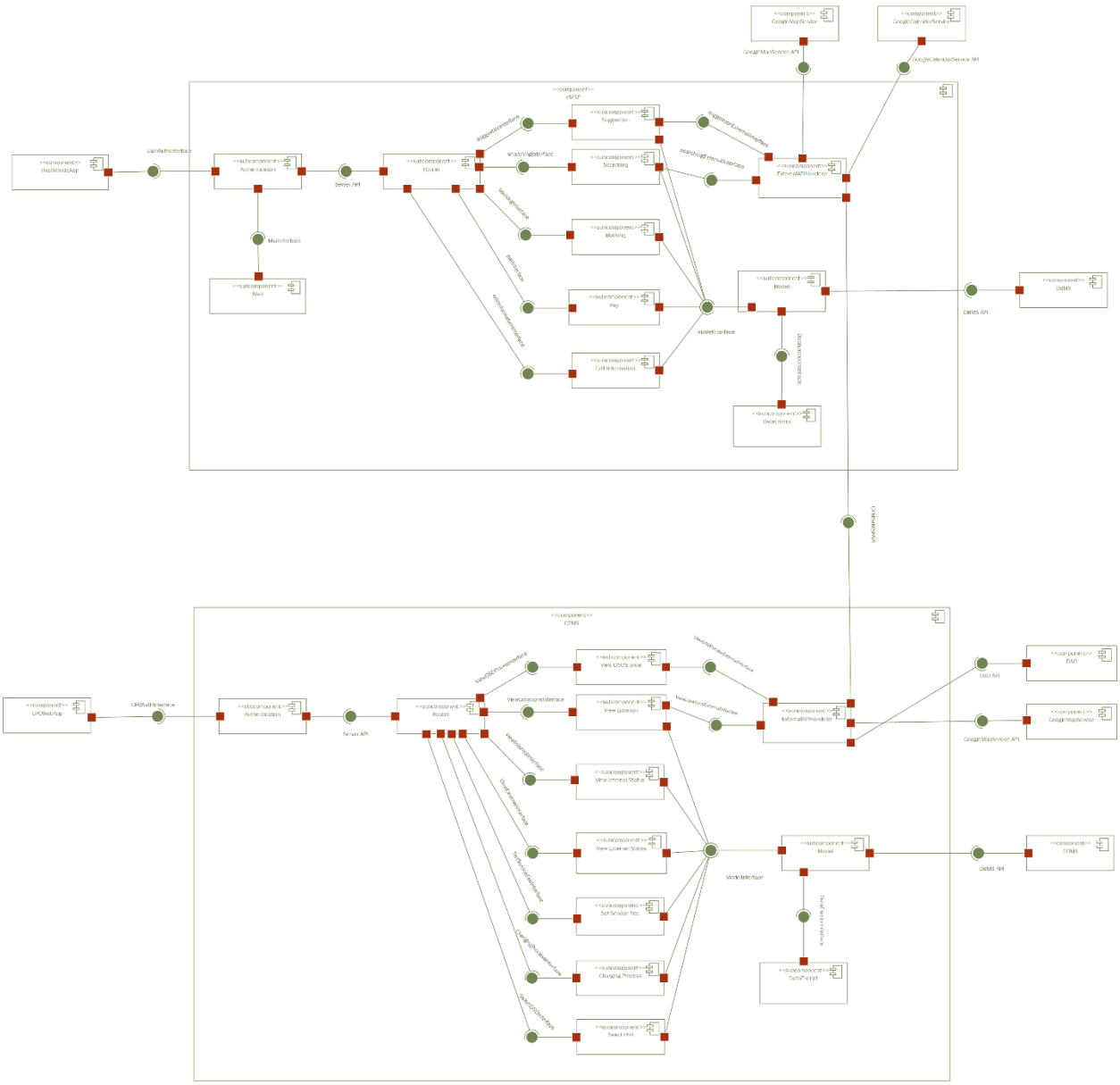

Figure 4 Component view

- **DBMS**
  This is the main component that accesses the database. More details are in following sections.

- **GoogleMapService**
  This component is responsible for gathering maps of different regions.

- **UserMobileApp**

This component is responsible for interactions between the system and users who are not logged in to the app. So, it offers registration and login functions. When users log in, they redirect to eMSP App through this component. User application component. E.g application on the smartphone

- **Authentication**
  For verifying user credentials and allowing user/CPO to log in.

- **Mail**
  This component is responsible for sending emails for verification to the user.

- **Router**
  The entry point of client requests to the server. Handles requests by rerouting them to the correct component if authorized.

- **Suggestion**
  This component handles suggestions that interacts with google calendar and google map services.

- **Searching**
  This component finds and shows the charging stations that the user searched for.

- **Booking**
  This component is used for booking a charging station.

- **Pay**
  This component is responsible for the payment section which users can pay for the services.

- **Edit Information**
  Component handling editing information by the user.

- **DataEncript**
  This component encrypts data in eMSP and CPMS application.

- **Model**
  This component contains an object-relation mapping (ORM) that other components use when retrieving data from the database. Thus, it is the component solely responsible for communicating with the data tier.

- **ExternalAPIHandeler**

This component is responsible for handling communication with external services. In our case this is currently of two types, either fetching of e.g. water irrigation data, or querying for confirming farmer connection to farm (when registering).

- **GoogleCalendarService**
  This component is used when a user sets a plan for charging in her/his google calendar and it will be shown on eMSP application.

- **DSO**
  This component is for giving price to the CPMS and CPO views them.

- **Select DSO**
  This component allows CPO to select a DSO for acquiring energy.

- **Charging Process**
  For showing the charging process of any car to the CPO.

- **Set Service Fee**
  This component handles prices that users must pay for the services.

- **View External Status**
  Responsible for showing external status to the CPO.

- **View Internal Status**
  Responsible for showing internal status to the CPO.

- **View Location**
  With this component, CPO can view the locations of each charging station.

- **View DSO's price**
  This component is responsible to show DSOs' prices to the CPO.

- **CPOWebApp**
  This component offers all the functions which CPMS offers, and it stands for CPOs who use a web browser. E.g. Web browser

2.3 Deployment view

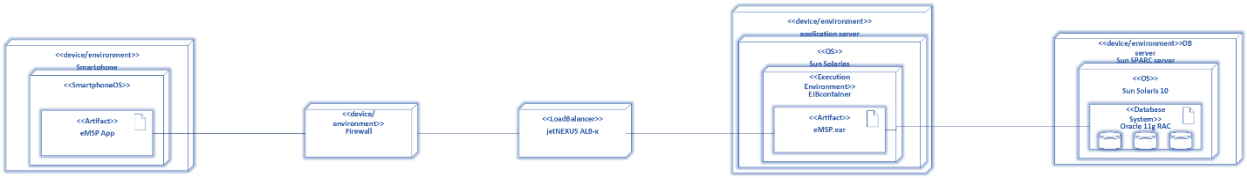

Figure 5 Deployment view for eMSP

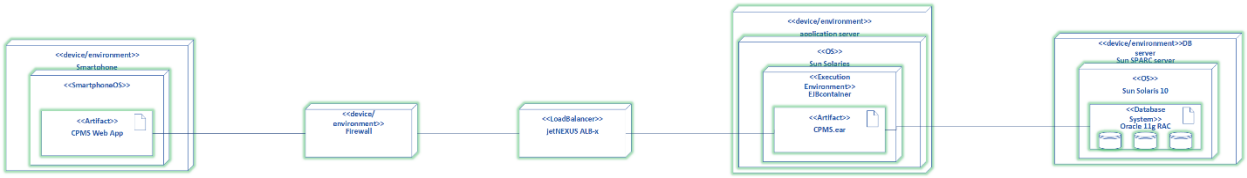

Figure 6 Deployment view for CPMS

- **Smartphone**
  The user (EV driver) can use the smartphone to use the application. She/he can register and log in to her/his account, book a charging station, and pay for the services.

- **Computer**
  CPOs can use web applications by computer or other devices that contain web browsers. The web application contains all features that we have discussed in RASD section.

- **Firewall**
  A firewall is used for providing safe access to the server and database, as well as, preventing external attacks.

- **Load balancer**
  For handling the workload of the application, we use a load balancer that distributes the workloads across multiple servers to increase the capacity and reliability of applications trying to avoid the overload of any single server.

- **Web Server**
  Receives queries from clients through the web application and sends received data to the application server for processing by using java RMI.

- **Application Server**
  This part contains application logic, but not all of it because the client also the client has a part of this. The Application Server handles all the requests and provides the appropriate answers for all the offered services.

- **Database Management System (DBMS)**

  For performing the job of DBMS Oracle RAC has been selected. A cluster comprises multiple interconnected computers or servers that appear as if they are one server to the end-users and applications. Oracle RAC enables you to cluster Oracle databases and it is a configuration of Relation DBMS (RDBMS). Some nodes contain the same instance of the database, and the instances are connected to each other and share the cache in the GC. Then, there is a pool of replicated databases that contain data files. Datafiles are common and shared and they are accessed from all instances in a parallel and synchrony way.

2.4 Runtime view

In the following sequence diagrams, we discuss the detailed interactions between the user and each component. The main interactions between the user and the system have been already discussed in RASD. In order to keep the diagrams readable, we just visualize the main flow of the events and don't consider the exceptions.

- End user-Register

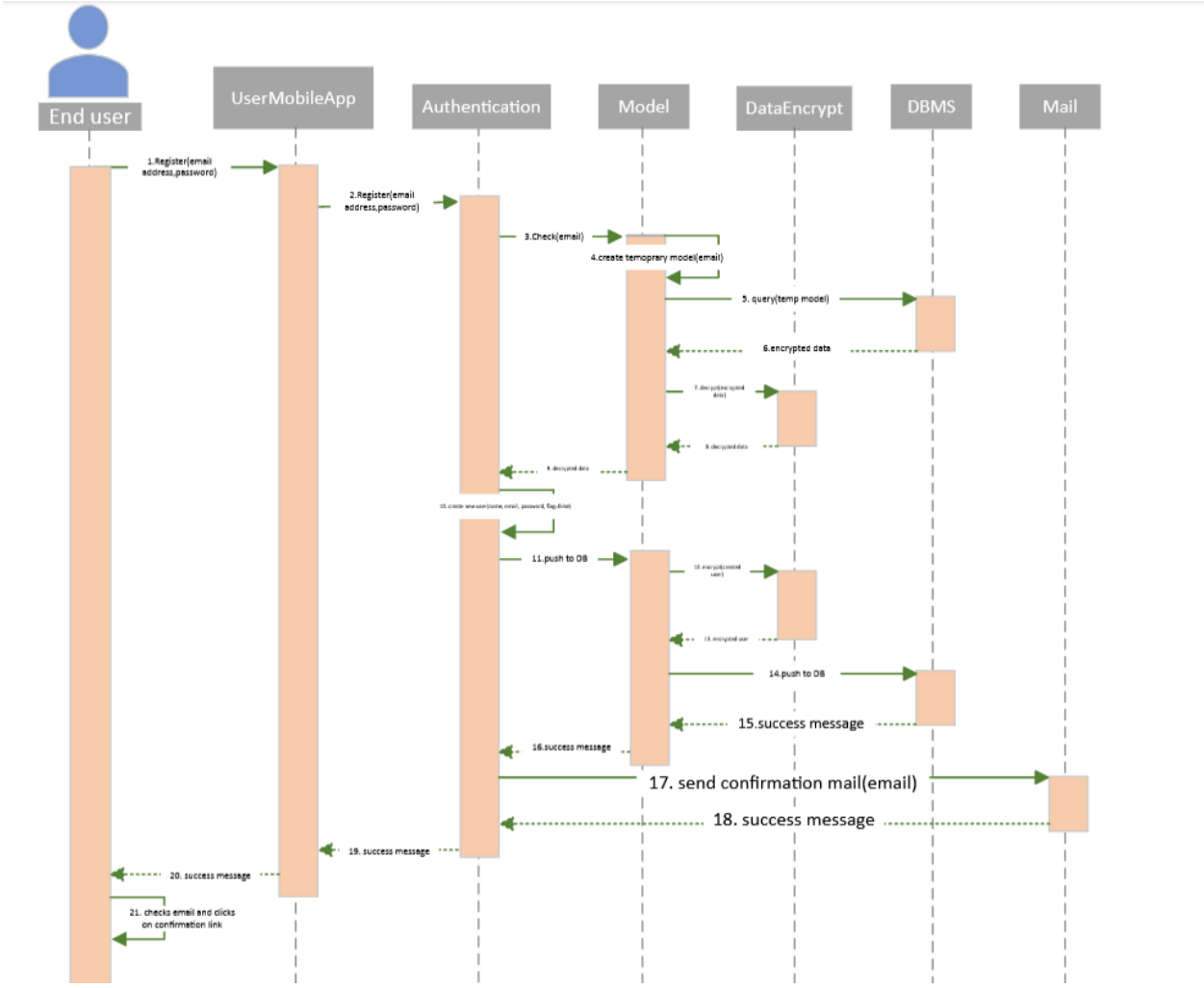

Figure 7 Runtime view - End user - Register

The end user fills the registration fields with her/his information through the “UserMobileApp” or component. Then, the information is sent to the “Authentication” component. In this step, we should check whether there is another user with the same email. So, the “Authentication” component sends the entered email to the “Model” component. Here, a temporary model with entered email is created. Then, a query is sent to the database to check the possibility of existing users with the same email. The answer to the database is encrypted. So, it is passed to the “DataEncrypt” component to be decrypted. If the retrieved data from the database is null, it means there is no user with entered email, so the “Authentication” component creates a new user with the inserted information, plus a flag equal to false, and pushes it to the database. Then, an email is sent to the registering user with a confirmation by “Mail” component. When the customer clicks on the confirmation link, he/she will be redirected to a web page. Finally, a query will be sent to the database to update this user flag to true.

- End user-Login

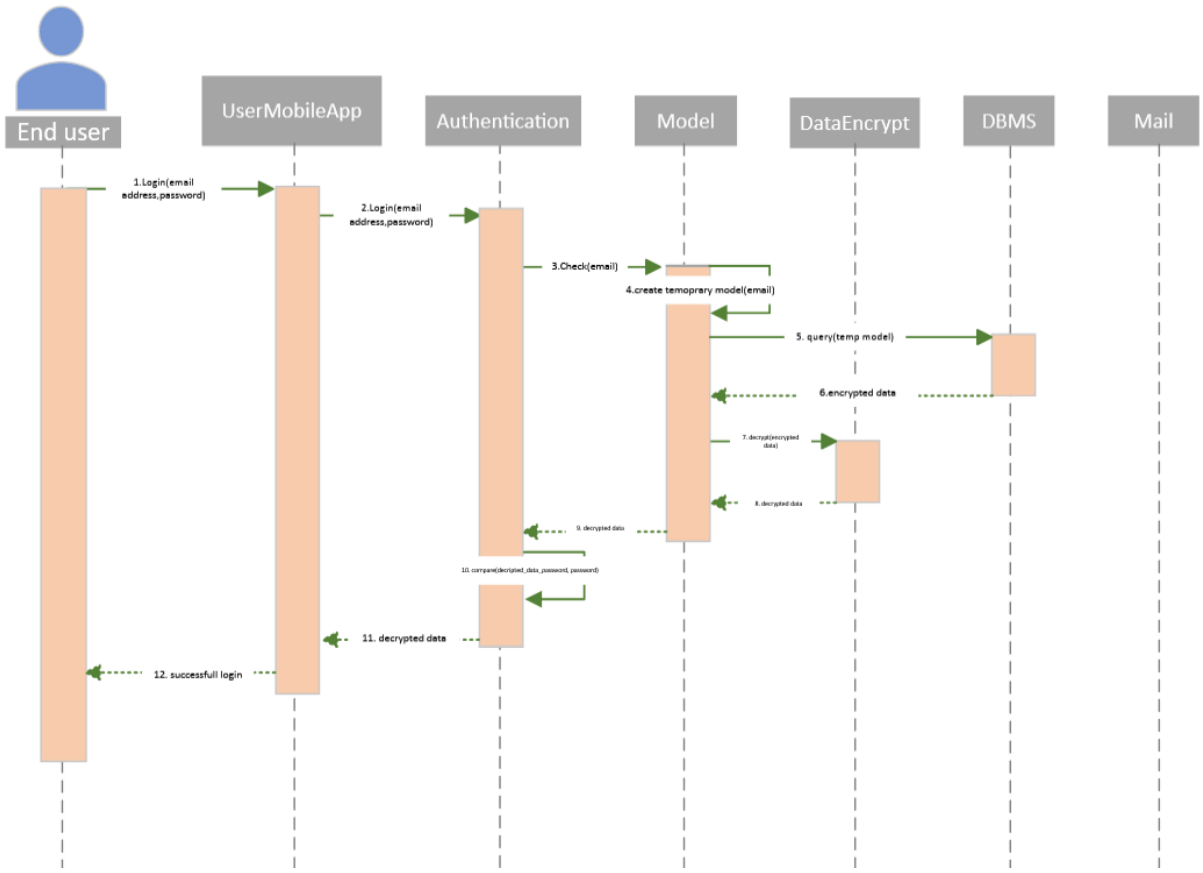

Figure 8 Runtime view - End user - login

The user enters the email and password through the "UserMobileApp" or component. Then, his/her information is sent to the "Authentication" component. In this step, we should check whether the user has been registered. So, the "Authentication" component sends the entered email to the "Model" component. Here, a temporary model with entered email is created. Then, a query is sent to the database to check the possibility of existing users with the same email. The answer to the database is encrypted. So, it is passed to the "DataEncrypt" component to be decrypted. If the retrieved data from the database isn't null, it means the user with entered email exists. Then, the "Authentication" component should check the correctness of entered password. In case the check is positive the retrieved user is returned, otherwise a null value is returned.

- End user-Book a time

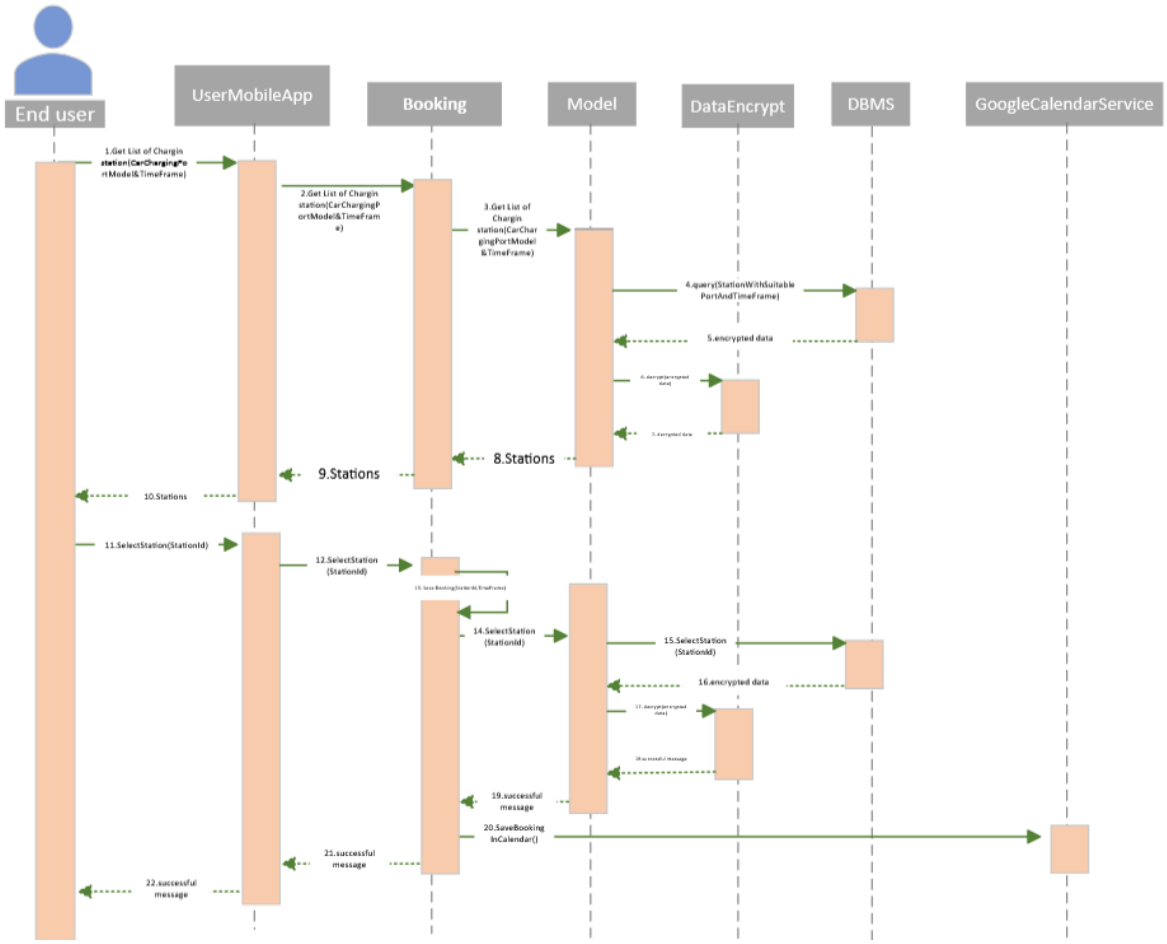

Figure 9 Runtime view - End user - Book a time

The user has already logged and now he/she is interacting with the “UserMobileApp” component. First, the user goes to the charging stations section through the “UserMobileApp” component and enters her/his time frame for charging. Then press on the search button, and So request is forwarded to the “Booking” component. This component sends a request to the server asking for all Charging stations with Suitable and free charging ports for the user time frame in the database. The answer to the database is encrypted. So, it is passed to the “DataEncrypt” component to be decrypted. The End user selects a Charging station and his/her request forwards to the “Booking” component and another request asks to book a time frame in a specific charging station to the server. After saving data in DBMS, the “Booking” component sends a request to “GooleCalendarService” to save a booking in the user calendar.

- End user-Edit personal information

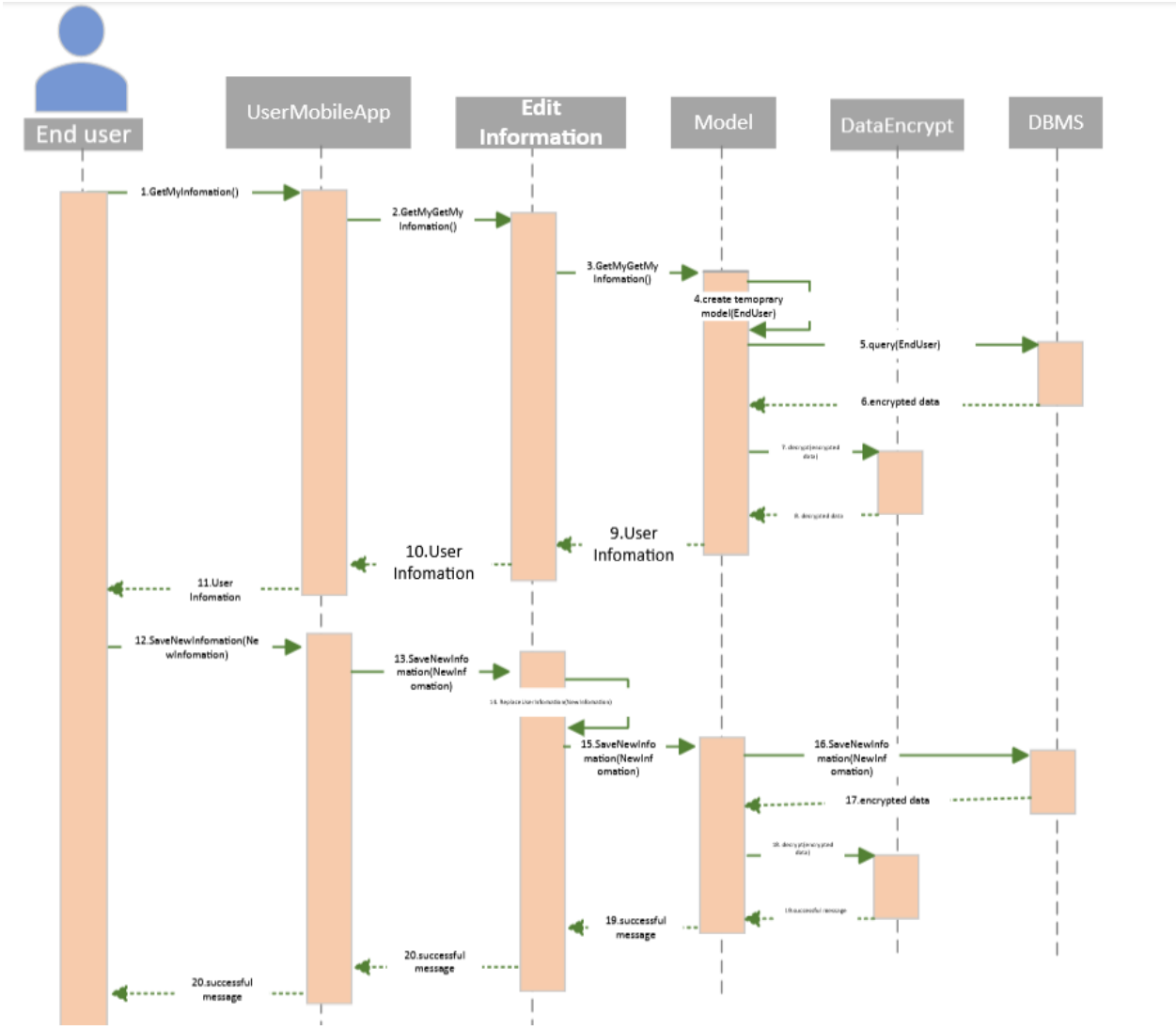

Figure 10 Runtime view - End user - Edit personal information

The user has already logged and now he/she is interacting with “UserMobileApp” component. First, the user goes to edit the personal data section through “UserMobileApp” component and “UserMobileApp” component sends a request to the server to retrieve end-user information. Then, the User can edit his/her personal information like first name, last name, and his/her car model. After that, the end user press on saves button. So, new requests are forwarded to the “Edit Information” component. This component sends a request to the server asking for replacing all new data of the user to replace with the old ones. The answer to the database is encrypted. So, it is passed to the “DataEncrypt” component to be decrypted.

- End user-pay by credit card

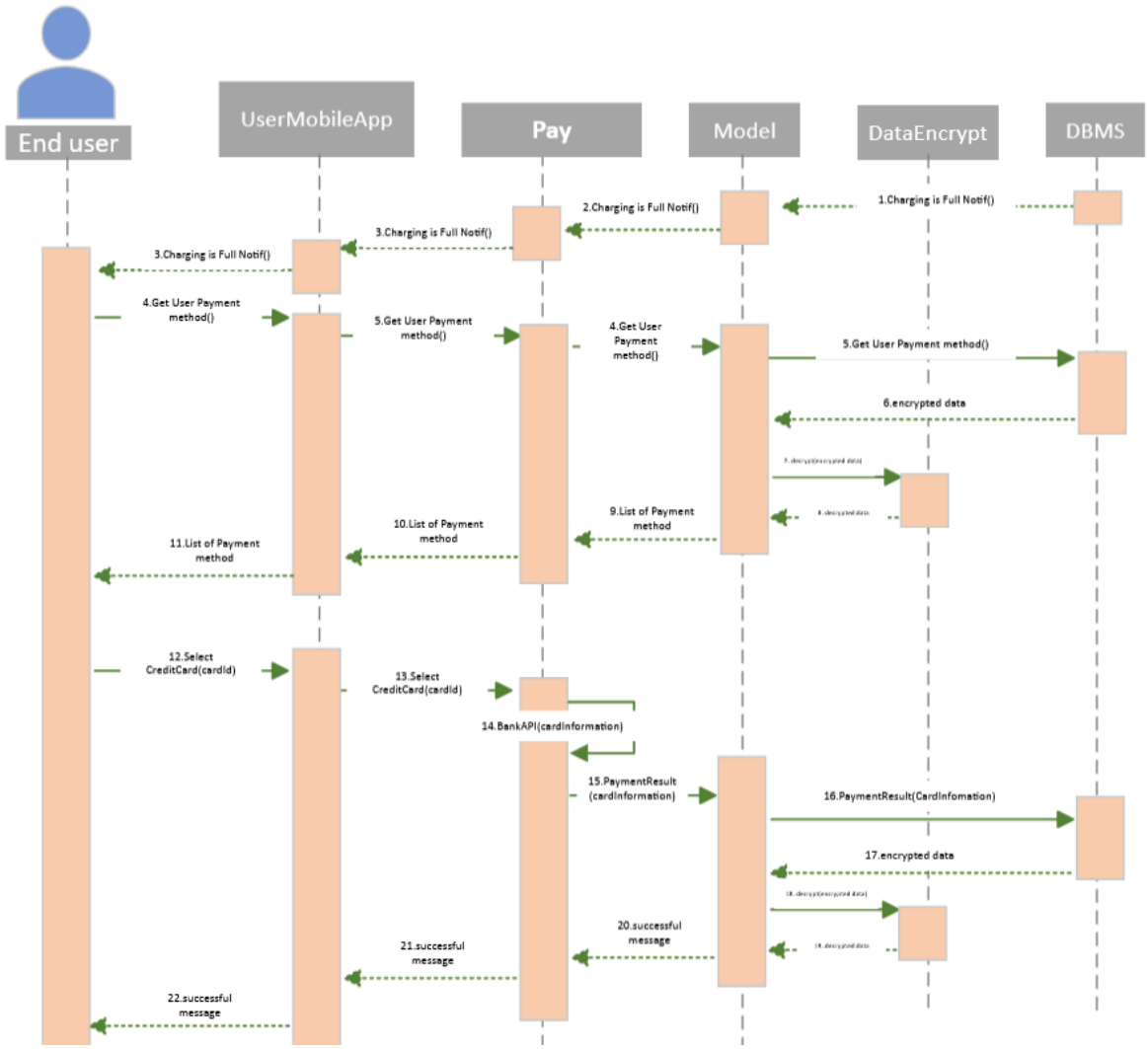

Figure 11 Runtime view - End user - pay by credit card

The user has already logged and now he/she is interacting with “UserMobileApp” component. Through this, the end-user sees a full charging notification. First, the user clicks on the notification so he/she can see his/her car status. At the top of the status page, can click on payment. So, the “UserMobileApp” component sends a request to the server to retrieve user payment methods like account balance and credit card. The end user clicks on the credit card. Through the Bank API in the “Pay” component, the money was deducted from her/his bank account, but data is encrypted so with the “DataEncrypt” component data is decrypted and shown the result to the user.

- End user- Get some suggestion

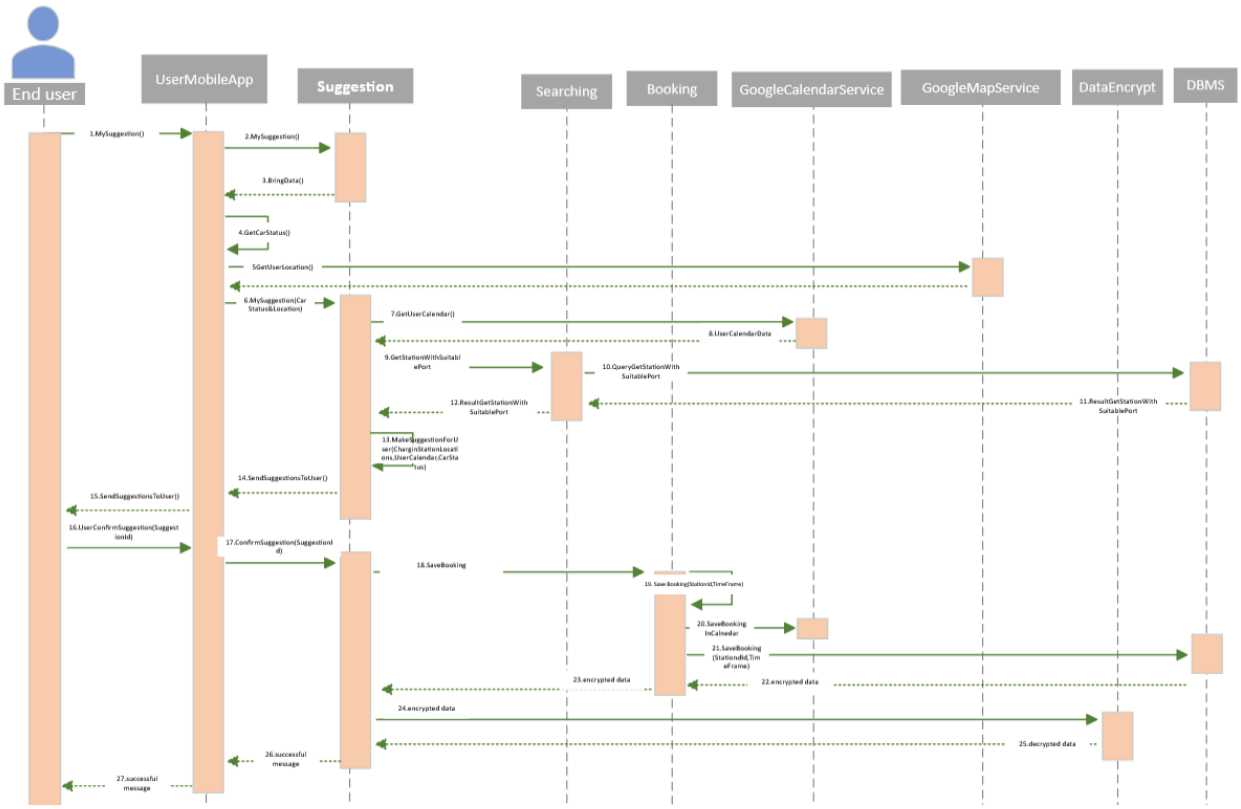

Figure 12 Runtime view - End user - get some suggustion

The user has already logged and now she/he is interacting with the “UserMobileApp” component. First, the user goes to the Suggestion section through the “UserMobileApp” component and the “UserMobileApp” component sends a request to the “Suggestion” component. The “Suggestion” component needs to know about the user’s car status like battery percentage, user current location, and user calendar. So, the “UserMobileApp” component gets the user’s current location via “GoogleMapService” component and gets car status via Bluetooth or API, and sends this data to the ”Suggestion” component. This component retrieves the user calendar with the “GoogleCalendarService” component. Then send a request to “Searching” component to find charging stations that are suitable for user cars and with her/his time frame that finds out with “GoogleCalendarService”. After that, the “Suggestion” component makes a list of suggestion for the user and send it to “UserMobileApp” component. The user confirms one suggestion. So, the “Suggestion” component sends a request to “Booking” component to save a booking for the user and set a calendar appointment with “GoogleCalendarService” component.

- CPO-view external, internal status and location of charging stations

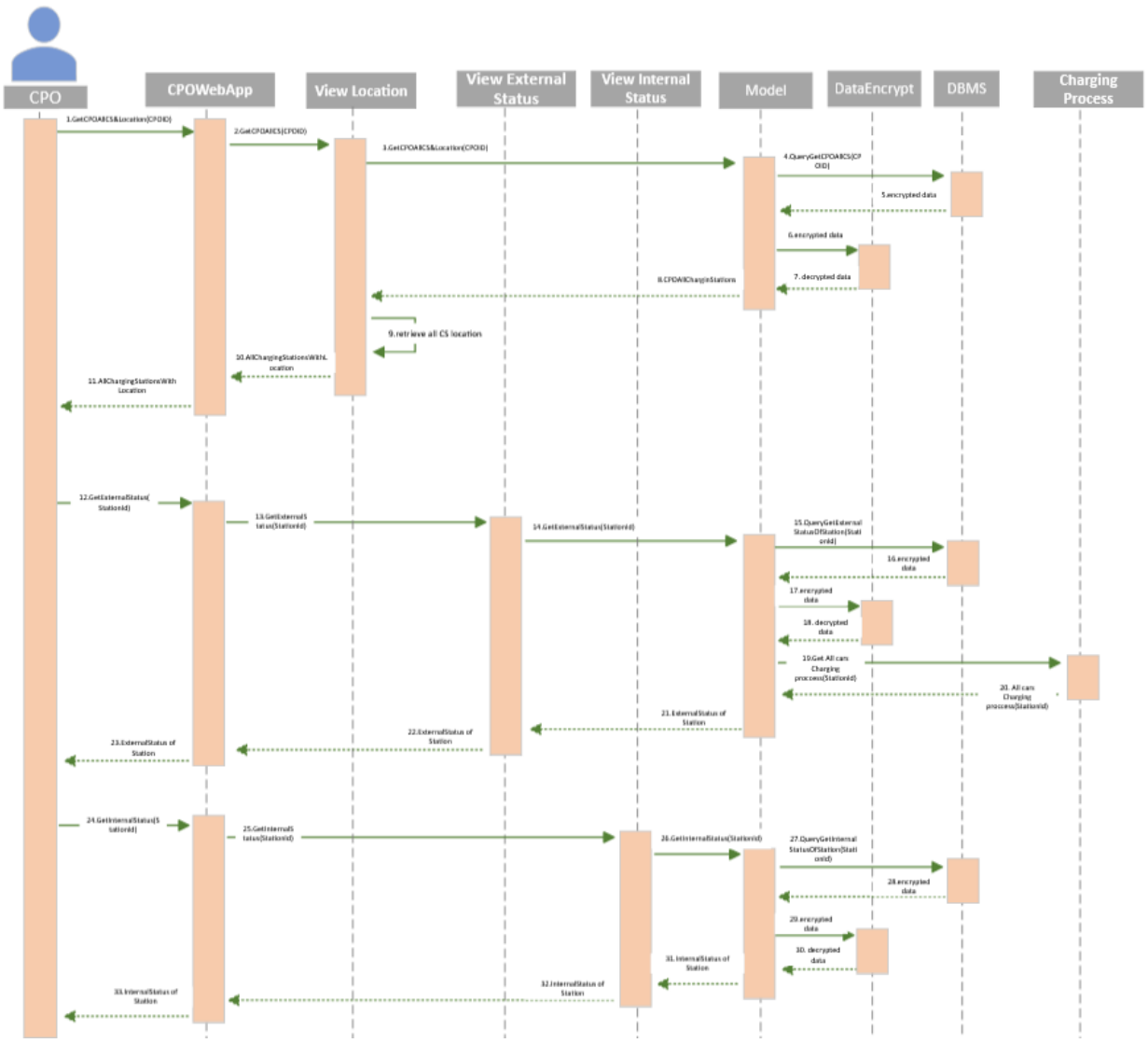

Figure 13 Runtime view - CPO-view external, internal status and location of charging stations

The CPO has already logged and now she/he is interacting with the "**CPOWebApp**" component. First, the CPO goes to my charging station section, "**View Location**" component sends a request to the "Model" component and this component sends a query to the "DBMS" component to find all the charging stations of the CPO with their location then return the list to "**CPOWebApp**" component. The CPO can see all external status and charging processes with the "View External Status" component and "Charging Process Component". Also, the CPO can see all internal statuses through the "View Internal Status" component. All data are encrypted, so with "DataEncrypt" component, we try to decrypt it.

- CPO-Select DSO to acquire energy and Edit Price of energy

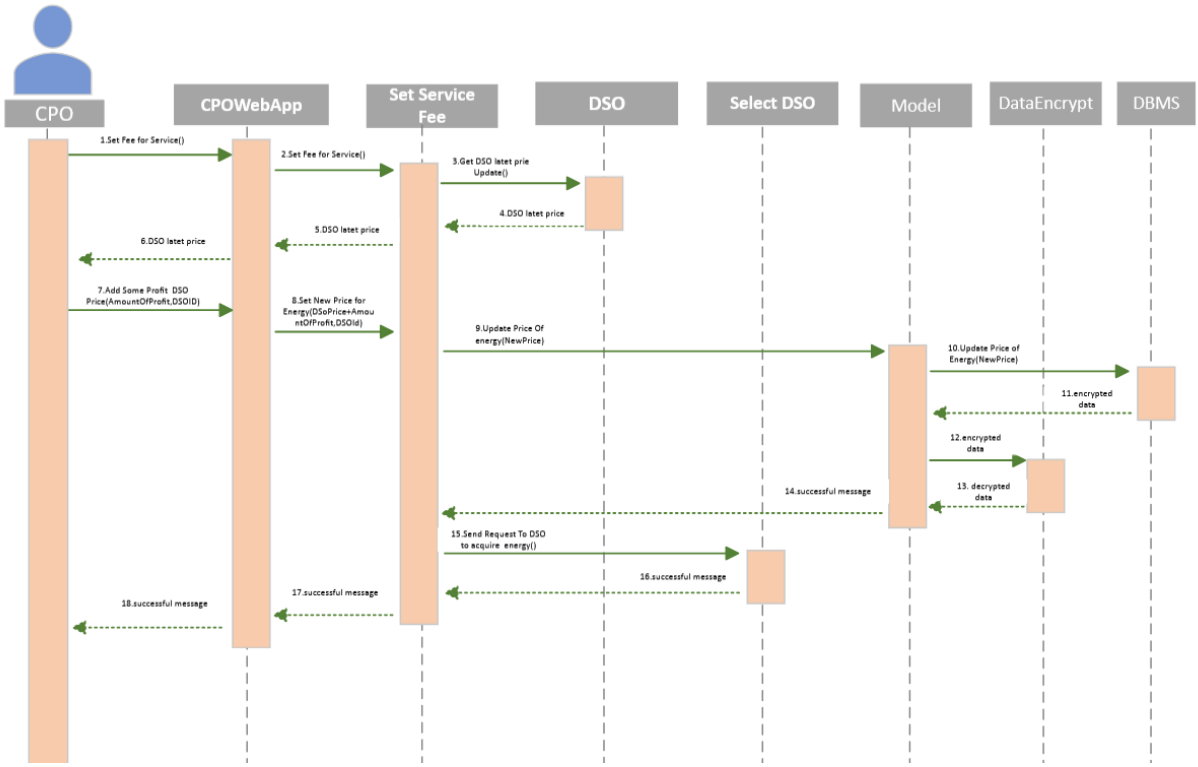

Figure 14 Runtime view - CPO-Select DSO to acquire energy and Edit Price of energy

The CPO has already logged and now she/he is interacting with "**CPOWebApp**" component. If CPO wants to edit the price of energy or to know the latest price for energy, must send a request through the "Set Service Fee" component. This component calls another component which names the "DSO" component. The "DSO" component calls all DSO's API to get the latest update of price and return it to "CPOWebApp". CPO can add some profit to one DSO price and save it. In this case, "Set Service Fee" send a request to the "DBMS" component to update the price of energy and also send a request to DSO to acquire the energy with the "Select DSO" component.

2.5 Component interfaces

The following diagram represents the interfaces that are offered by the different components and their interactions. These interfaces are based on the most important processes which are mentioned in the runtime view section.

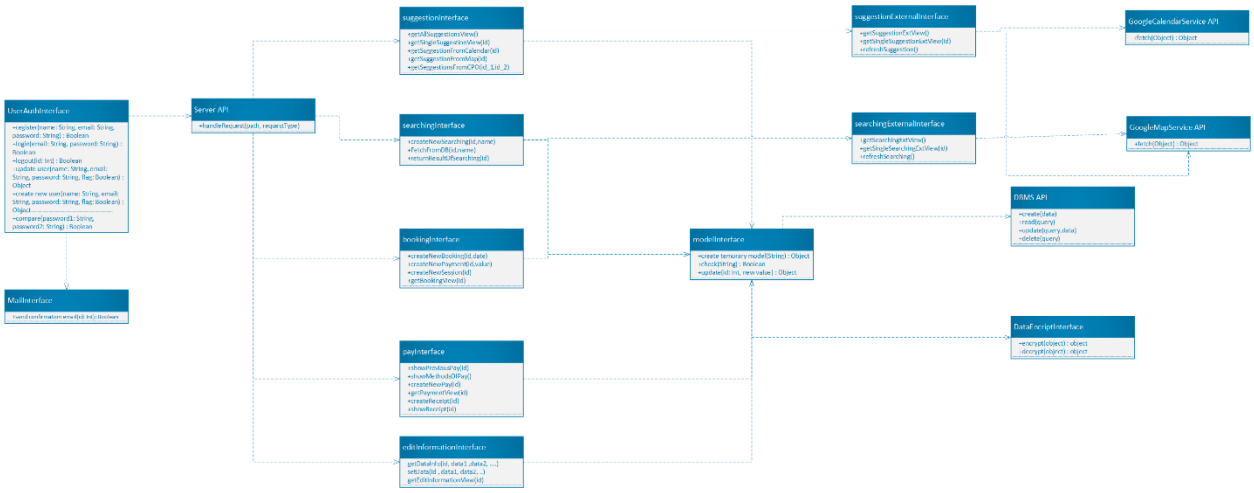

Figure 15 Component interfaces for user

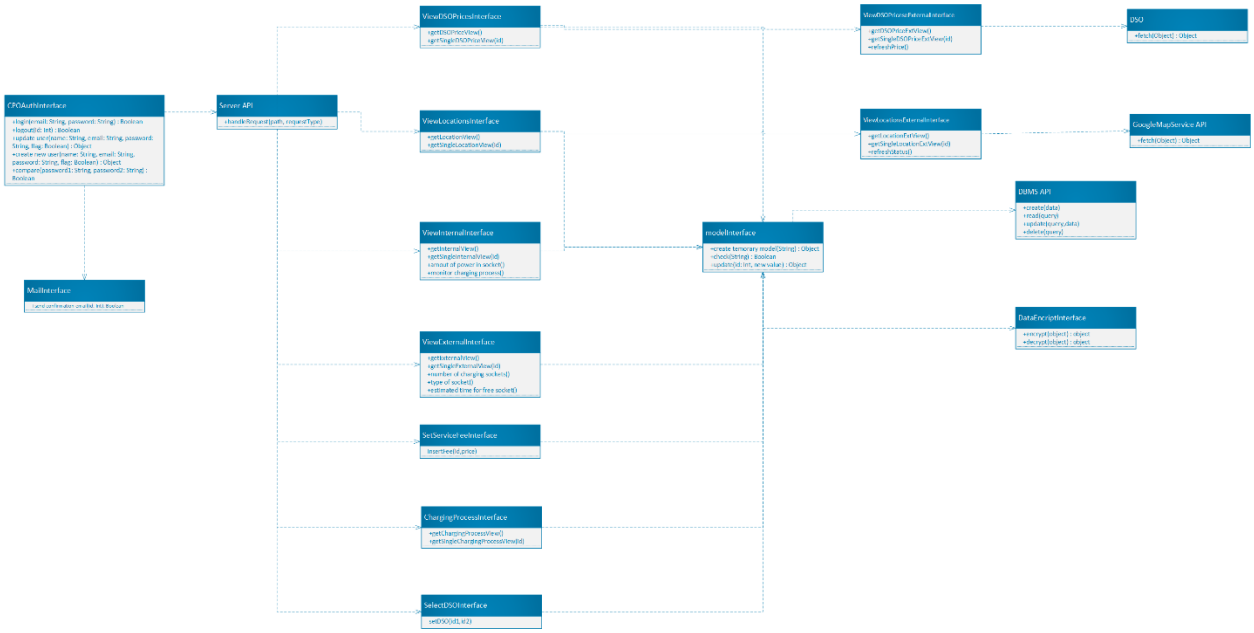

Figure 16 Component interfaces for CPO

2.6 Selected architectural styles and patterns

As mentioned in the overview, developing the system is used a three-tier client-server architecture: a presentation tier, an application tier, and a data tier. Dividing architecture into these 3 parts increases its flexibility, scalability, and reusability. As well as existing components in the application server make the system more comprehensible and modifiable. In the eMall system except for user information, we do have not sensitive data, thus HTTP is used to exchange messages. This is also endorsed by the importance that data plays in the system. In this way, it is possible to reduce the coupling between client and server

components. When dealing with sensitive data, TSL is used to guarantee the security and reliability of the connection.
Furthermore, the system uses the cache on the client tier: this allows for avoiding of some interactions between the client and the server and speeds up communication. Using the cache is useful when the client needs more data during one session. Regarding the format in which the data are transmitted, JSON is used for its simplicity. It is less verbose, and this makes it more readable and allows a much faster parsing. The JSON file is transmitted over the network via XMLHttpRequest.
For using Google Maps and Google Calendar, the communication protocol used is REST (REST is web standards-based architecture and uses HTTP protocol): Google services are indeed used as an external service and this protocol clearly suits this situation. Furthermore, the public REST API provided by Google services is used. It also gives the possibility to customize the requested content. For dealing with other parts like DSO for viewing prices of DSOs, the communication protocol is SOAP (SOAP uses HTTP to exchange the XML messages). Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, the initial focus of SOAP is remote procedure calls transported via HTTP. SOAP has an independent security mechanism, and due to this is useful for connecting with DSO's services. The application server works as a client and makes queries and receives responses. To perform the application the following pattern is used:

Model View Controller (MVC)
MVC is an architectural pattern consisting of three parts: Model, View, and Controller (MVC separates the application into these 3 multiple layers of functionalities).

- Model
  Handles data logic.
- View
  It displays the information from the model to the user.
- Controller
  It controls the data flow into a model object and updates the view whenever data changes.

Using this architecture guarantees the maintainability and reusability of the code. This software pattern is particularly adapted for the development of both web and mobile applications in an object-oriented style of programming as java. This method allows components to be created independently and simplifies simultaneous development.

Facade pattern
In this pattern, a component offers an interface that simplifies the use of another component. The DSO GoogleMapService and GoogleCalendarService components use it to adapt and simplify the use of external services.

2.7 Other design decisions

As mentioned in the previous section, a database has been selected for storing all the eMall and CPMS data. This role is performed by Relational databases, most specifically by the Oracle RAC database, as noted under the "Deployment View" paragraph. Relational databases are chosen because it is easy to categorize and store data, which can then be queried and filtered to produce reports with specific information.

Customers using Oracle Real Application Clusters (RAC) can run a single Oracle Database across multiple servers allowing for increased availability and horizontal scalability while sharing storage. Multiple hardware systems make up the database, but the application sees it as a single entity. Commodity hardware enables the utilization of low-cost computing environments and provides scalability for a wide range of applications. As well as these, users can still access Oracle RAC instances during outages, and changes can be safely replicated without any impact on end-user applications, masking the impact of the outage for end users.

Here are some benefits of a relational database:

- **Simple Model**

Relational databases are the simplest types of databases because the structuring and querying processes are simple. It does not require repetitive architectural processes like hierarchical database structure or definition.

- **Accuracy**

Multiple tables can be related to one another in a relational database system by using a primary key and a foreign key. This makes the data to be non-repetitive.

- **Easy Access to Data**

Relational databases do not have a pattern or pathway for users to access the data, as other databases can only be accessed by going through a tree structure or hierarchical structure.

- **Flexibility**

A Relational Database system by itself can expand and level up because its structure is flexible and able to adapt to ever-changing requirements. This facilitates the increasing incoming amount of data, as well as the update and deletes wherever required.

- **High Security**

Since the data is split up amongst the tables in the relational database system, it is possible to mark some tables as confidential while others are not. Unlike other databases, a relational database management system allows you to easily implement this separation.

3. USER INTERFACE DESIGN

3.1 User interface design for eMSP

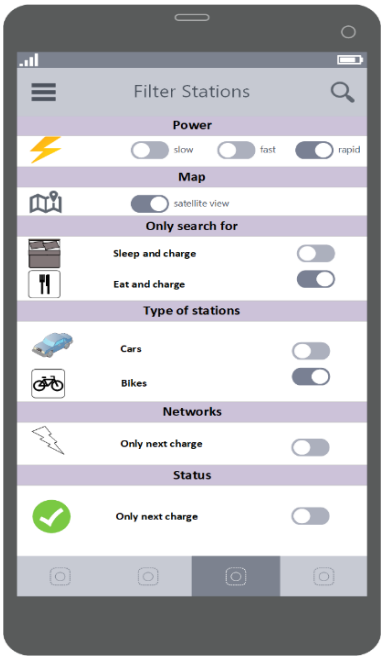

Figure 17 filter options

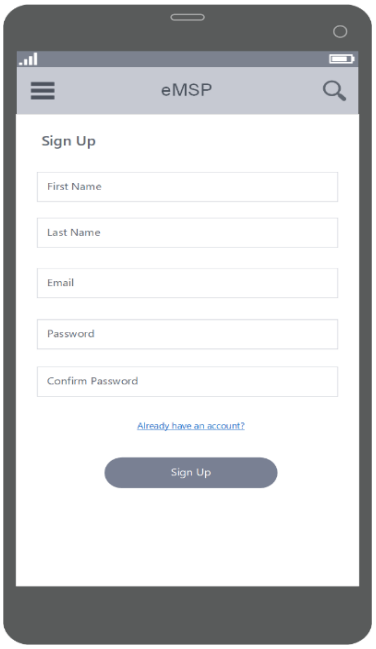

Figure 18 sign up

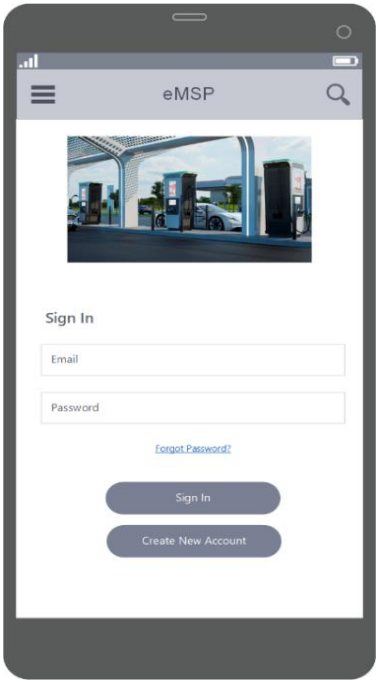

Figure 19 sign in

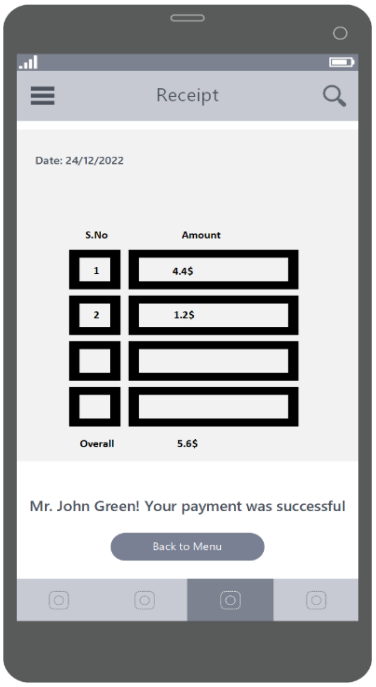

Figure 20 receipt

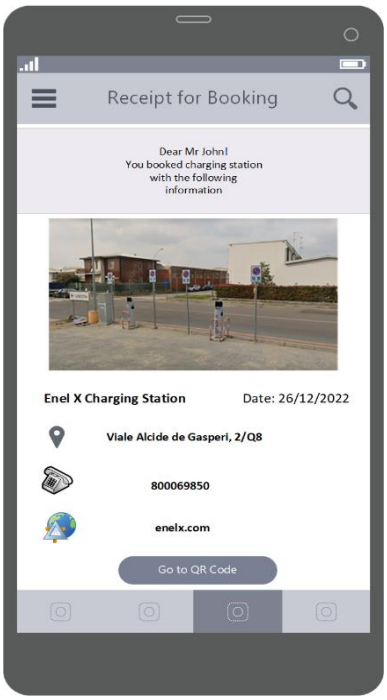

Figure 21 Receipt for Booking

Figure 22 QR code for booking

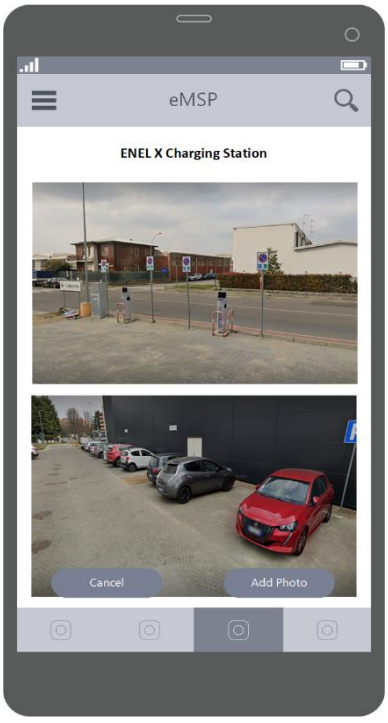

Figure 23 photos of charging station

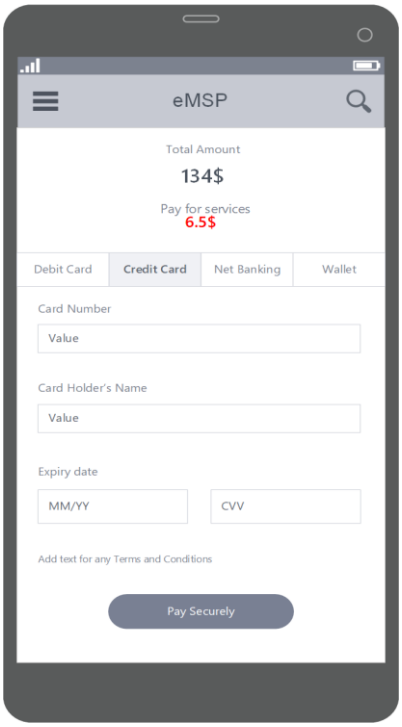

Figure 24 payment method

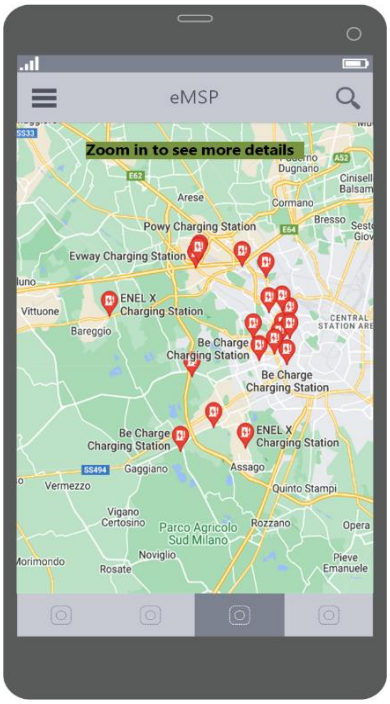

Figure 25 map

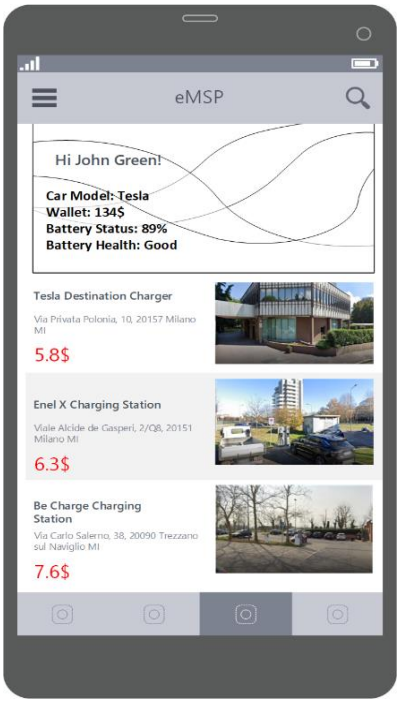

Figure 26 main page

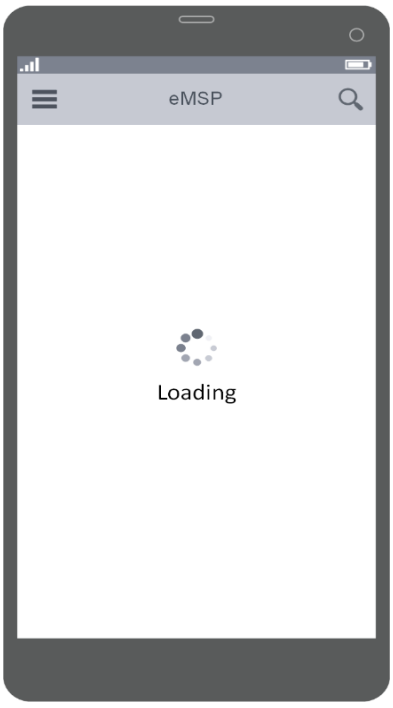

Figure 27 loading

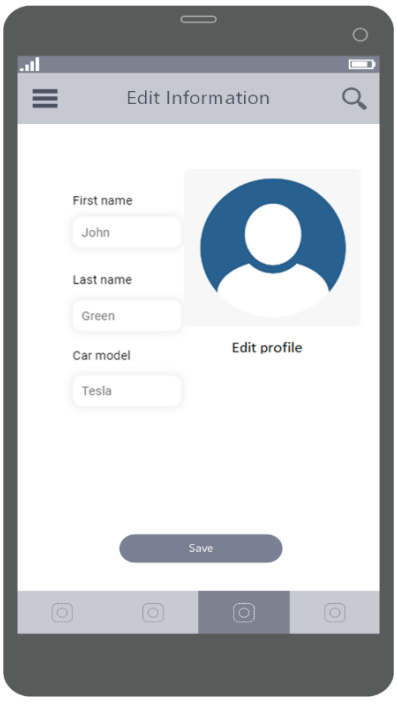

Figure 28 edit information

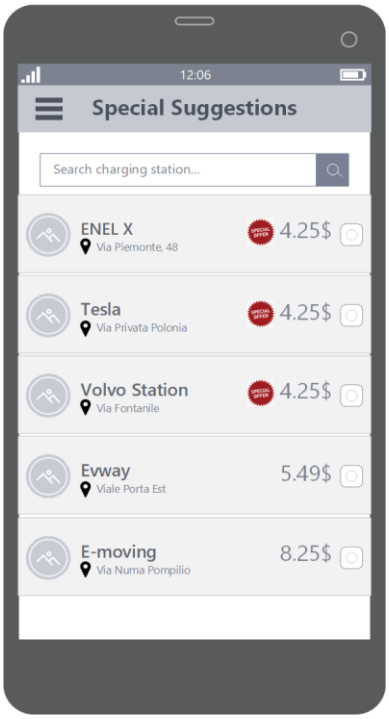

Figure 29 special suggestion

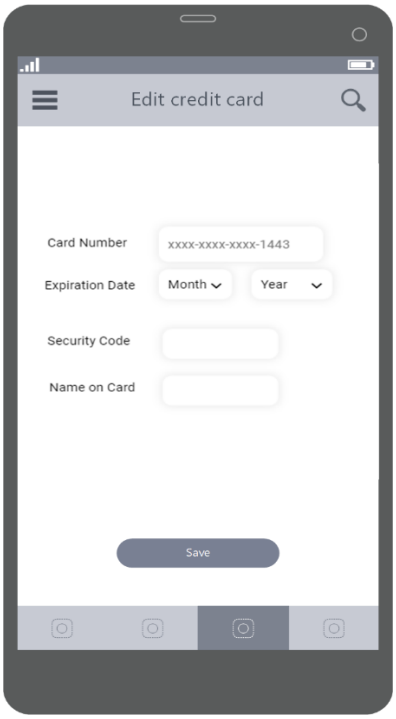

Figure 30 edit credit card

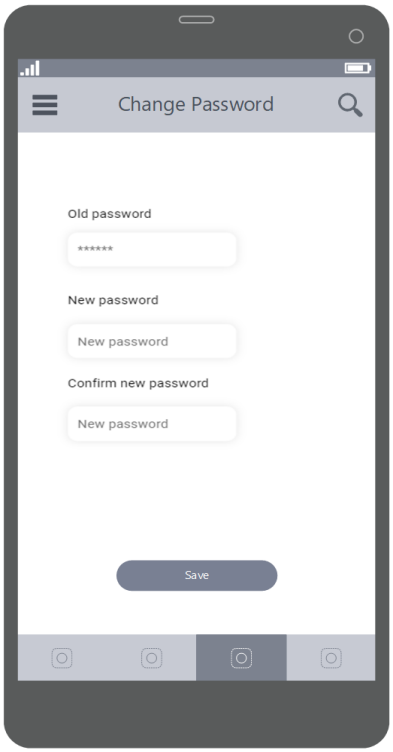

Figure 31 change password

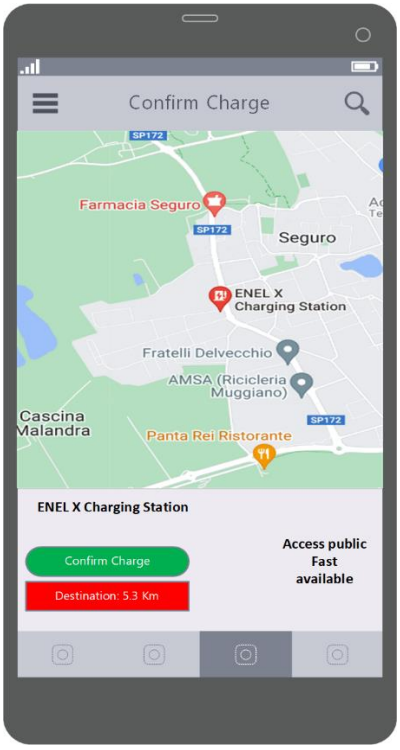

Figure 32 confirm charge

3.2 User interface design for CPMS

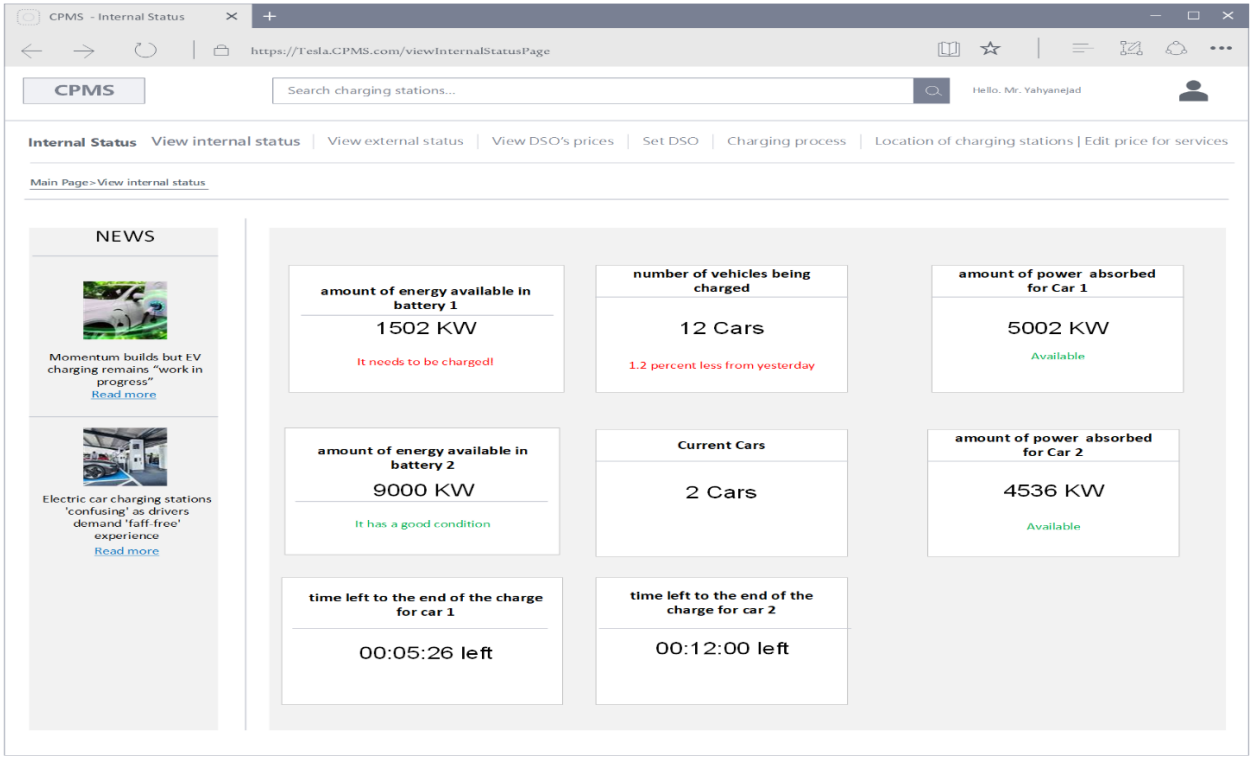

Figure 33 internanal status

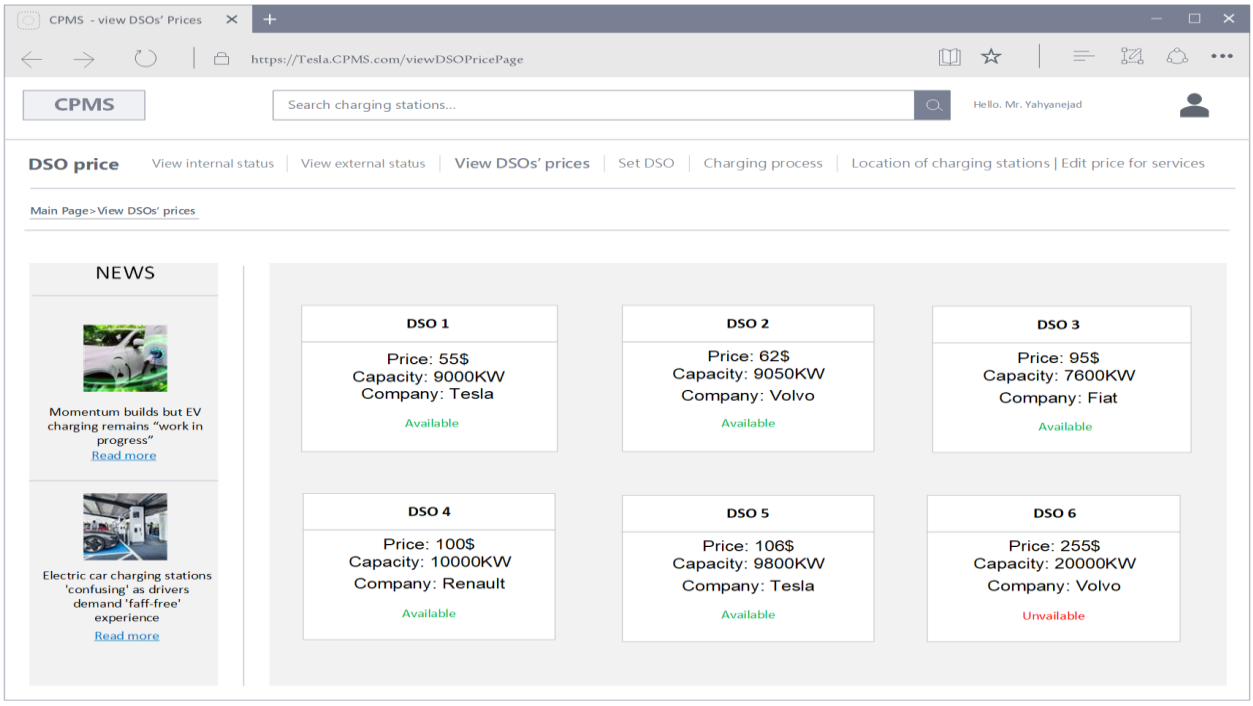

Figure 34 DSO price

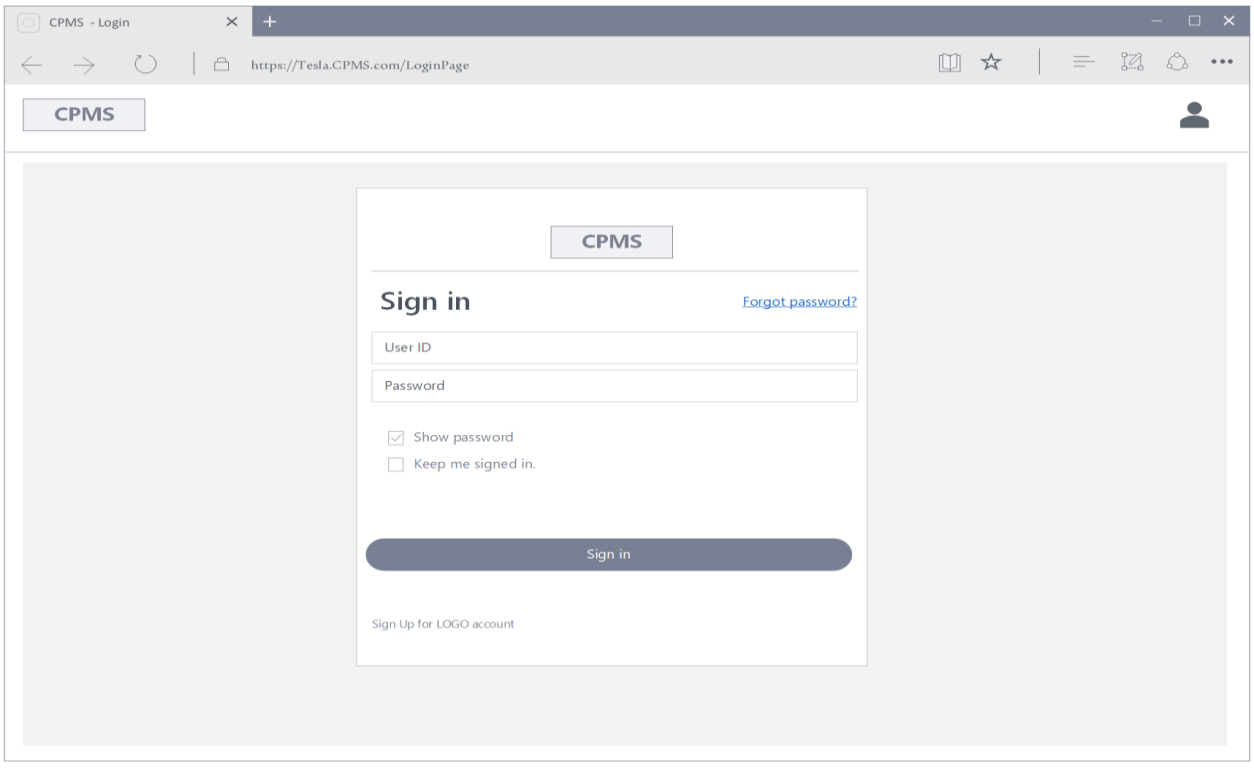

Figure 35 sign in

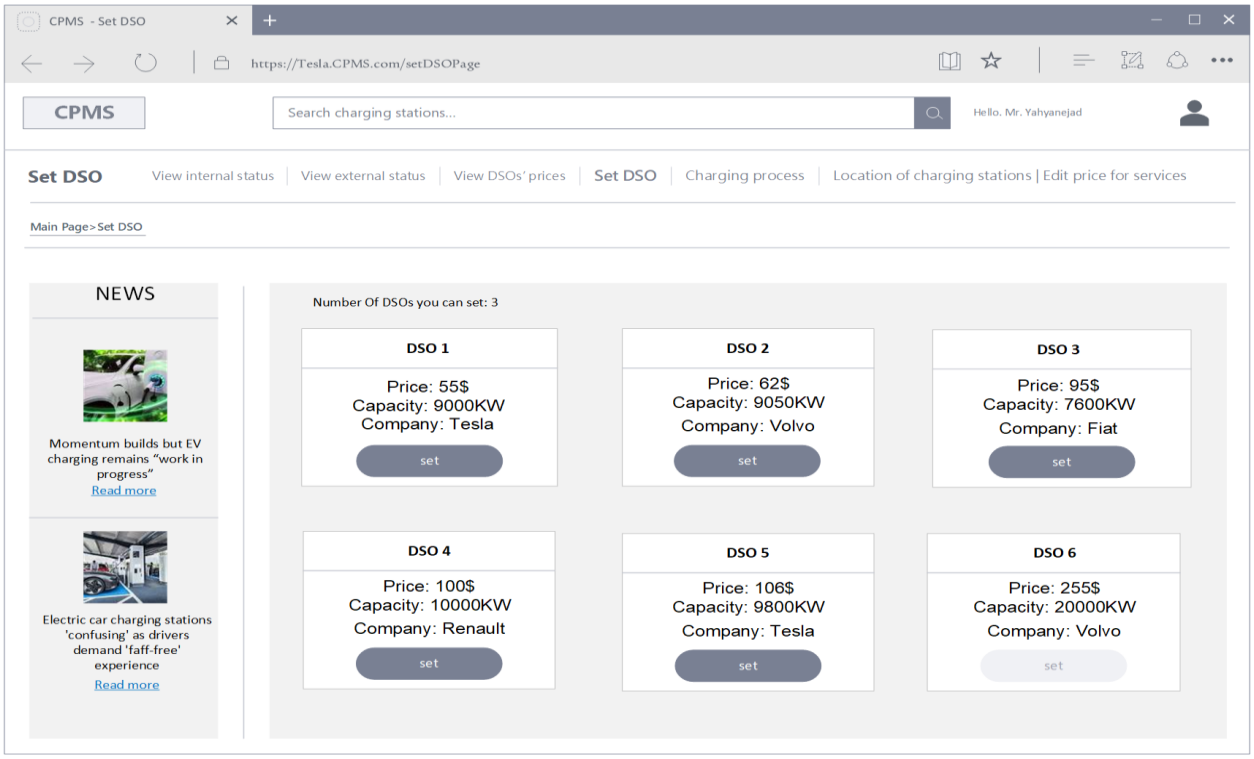

Figure 36 set DSO

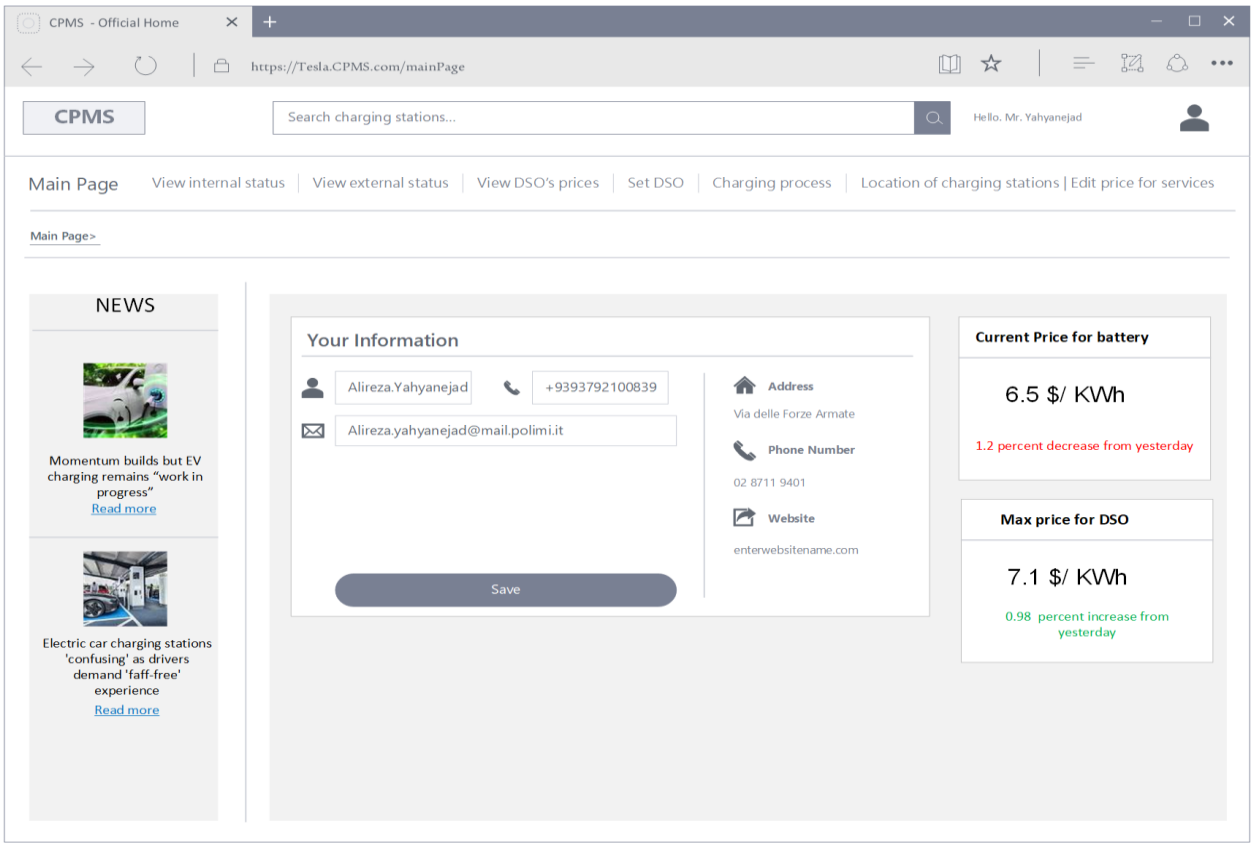

Figure 37 main page

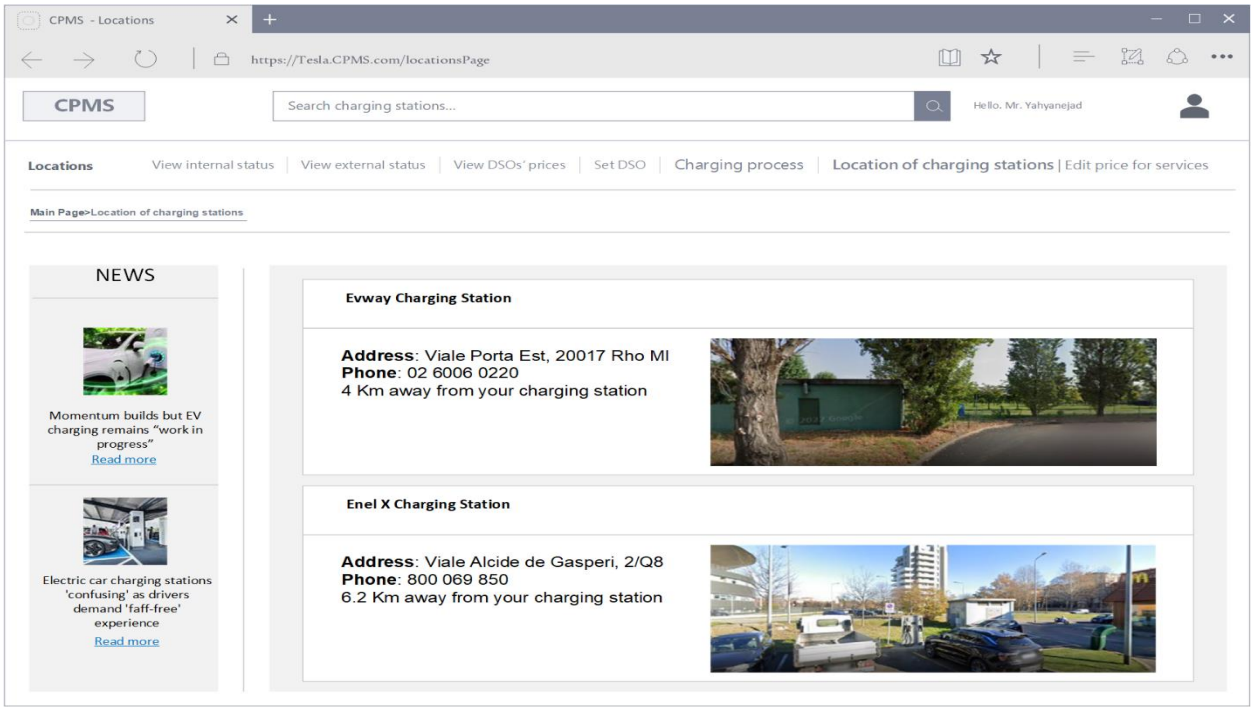

Figure 38 location of charging stations

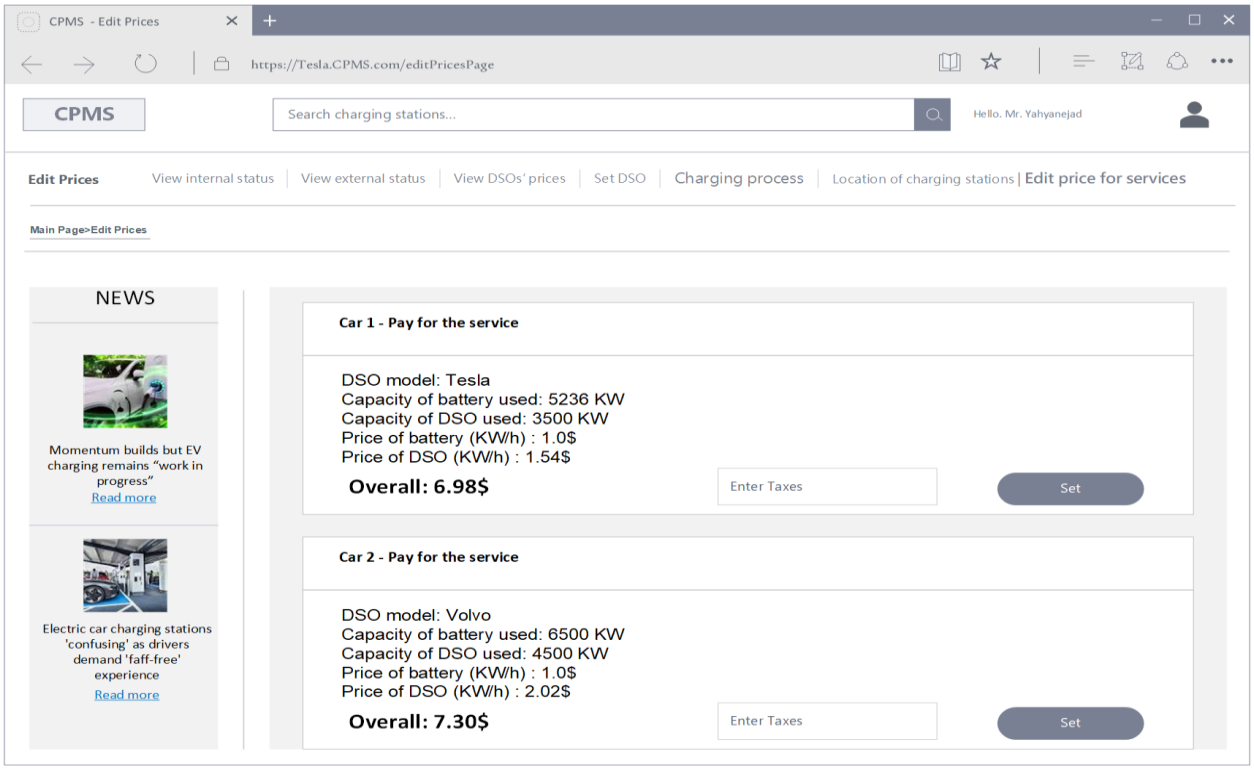

Figure 39 edit price for services

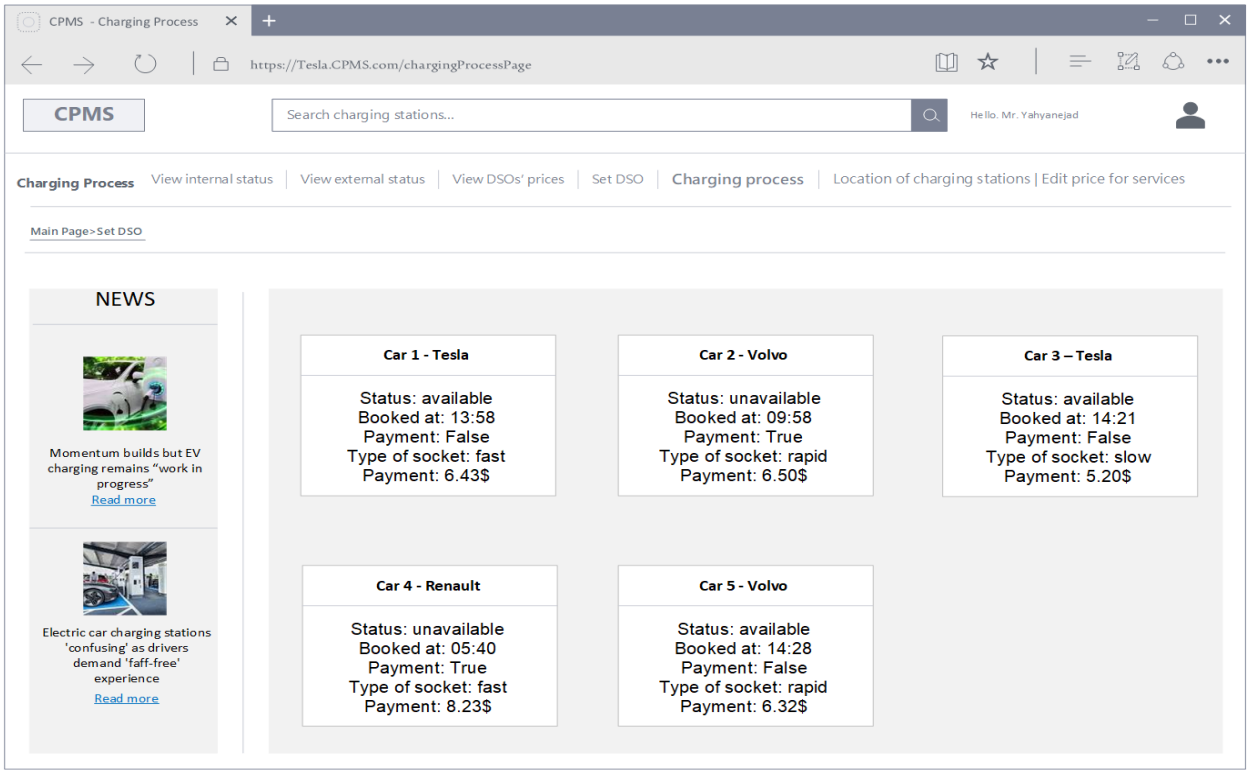

Figure 40 charging process

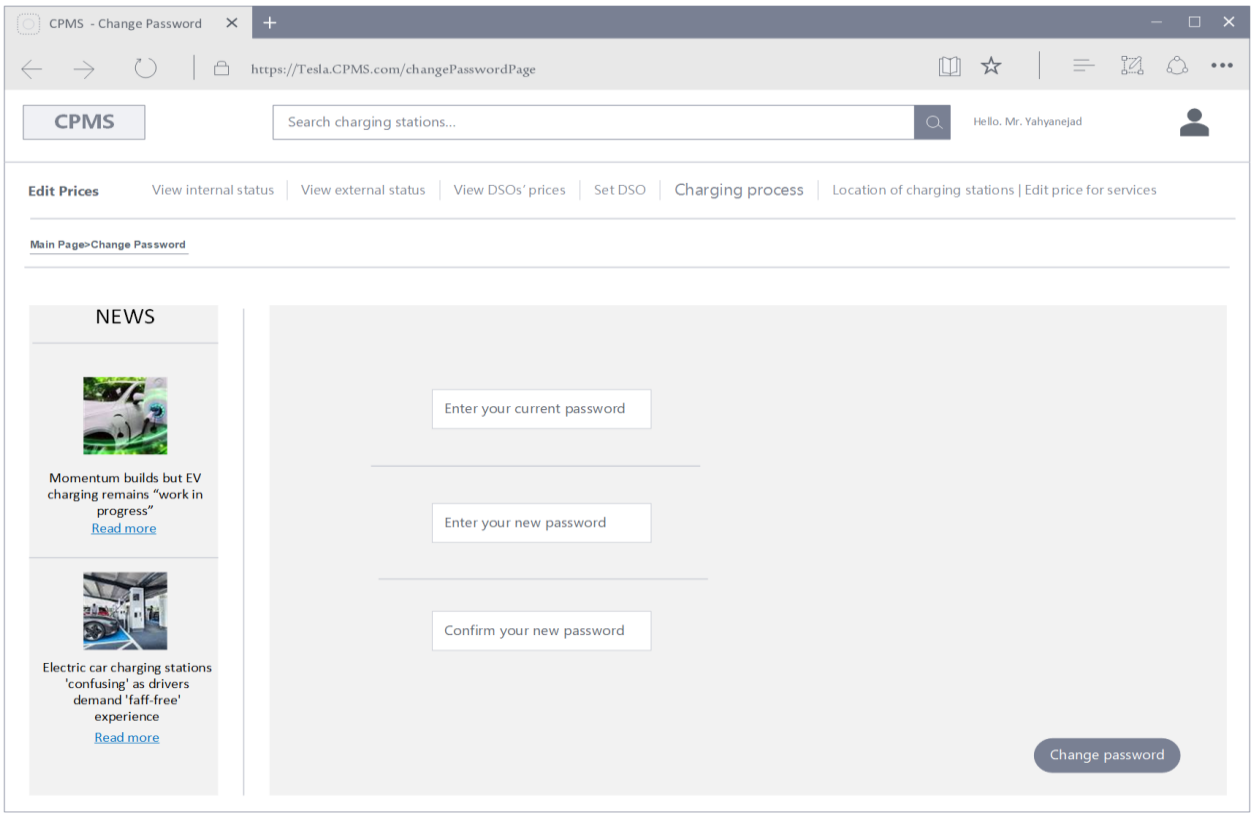

Figure 41 change password

4. REQUIREMENTS TRACEABILITY

In this section, we discuss a mapping between the requirements which have been defined in RASD and the design components. For more readability, we number the components as follows:

C1: DBMS

C2: GoogleMapService

C3: UserMobileApp

C4: Authentication

C5: Mail

C6: Router

C7: Suggestion

C8: Searching

C9: Booking

C10: Pay

C11: Edit Information

C12: DataEncript

C13: Model

C14: ExternalAPIHandeler

C15: GoogleCalendarService

C16: DSO

C17: Select DSO

C18: Charging Process

C19: Set Service Fee

C20: View External Status

C21: View Internal Status

C22: View Location

C23: View DSO's price

C24: CPOWebApp

R	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24
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	*		*			*			*			*	*					*						
R36	*		*			*					*	*	*											
R37	*		*			*							*	*										
R38	*					*							*	*		*							*	*
R39	*	*	*			*							*	*										
R40	*		*			*							*											
R41	*					*						*	*	*			*							*
R42	*					*							*											*
R43	*					*	*					*	*											*

5. Implementation, Integration and Test Plan

5.1 Overview

It is possible to show the existence of bugs through program testing, but not to show their absence. In order to accomplish this goal, the aim is to find as many bugs as possible before the release date. All the preliminary considerations needed to implement and test eMSP and CPMS are presented and explained in this section.

5.2 Implementation plan

To implement the whole system, the bottom-up approach is selected so that each section of the system can be implemented and tested separately. The components and subcomponents described in the component view section can be divided.

in various subsystems:

- UserMobileApp
- CPOWebApp
- Application server
- External systems: DBMS, Google map service, Google calendar service.

The implementation should be done from the lower components up to the top ones. Because there are some components that rely on others. For example, every component and sub-system described in the component diagram interact with the DBMS. Thus, we implement the DBMS component. On the other hand, Google Maps and Google calendar are two other components that play a role like a database. So, they should be implemented in this step. After that, we can proceed to the implementation of the subcomponents in the Application Server component. Here, there are some main subcomponents such as the Model and the DataEncrypt subcomponents. Most components use them as a connection to access the database. Then, it is Mail and Authentication subcomponent's turn. The Mail subcomponent should be implemented before the other one because the functionalities of Authentication rely on the Mail subcomponent, and we want to keep using the bottom-up approach while testing. In the next step, we can go for implementing all the subcomponents in the UserMobileApp and CPOWebApp. Each of these subcomponents works independently, so they can be implemented at the same level.

5.3 Integration and Testing

The following diagrams illustrate how the process of integration testing takes place, according to a bottom-up approach.

The arrows start from the component which "uses" the other one.

- At first the Model, DataEncrypt, and DBMS have been integrated and unit tested. They are not complex components, but other components rely on them to provide some services: that's why they are integrated and tested first.

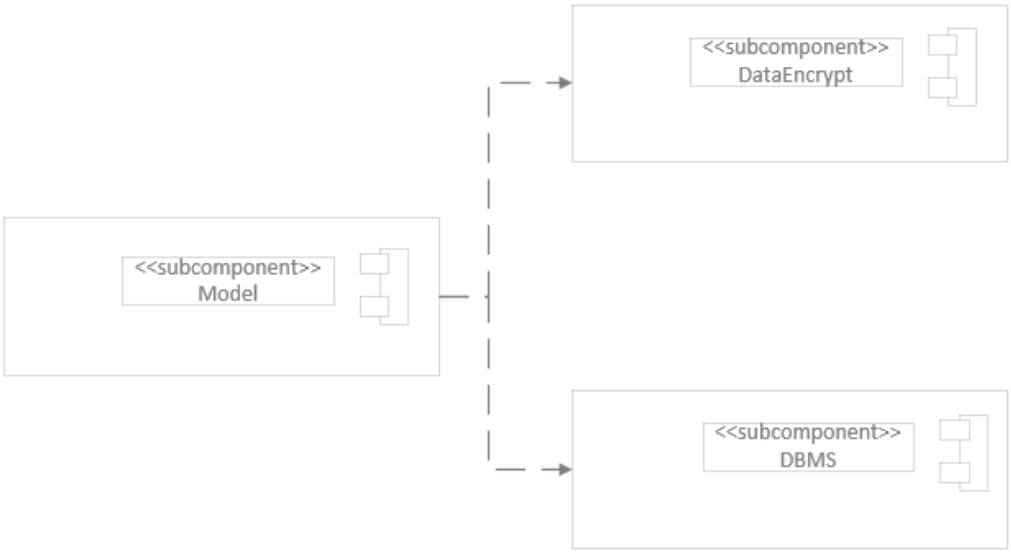

Figure 42 Integration: Model, DataEncrypt, DBMS

- Then, as we mentioned in the previous section, two other important components named Authentication and Mail should be integrated with the Model component.

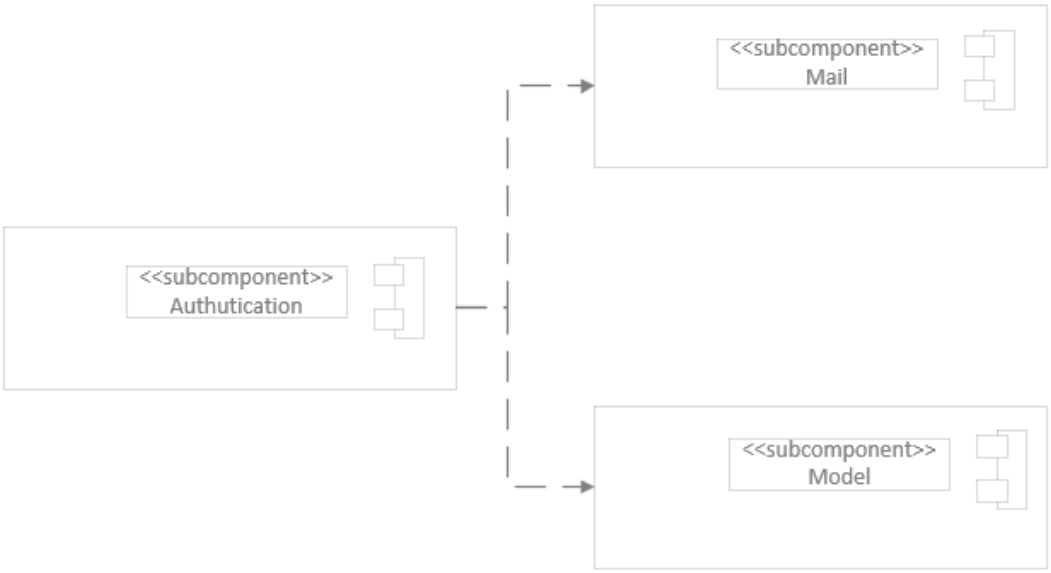

Figure 43 Integration: Model, Mail, Authentication

- In the next step, the components responsible for gathering information about car status and maps have been integrated and tested for the suggestion components.

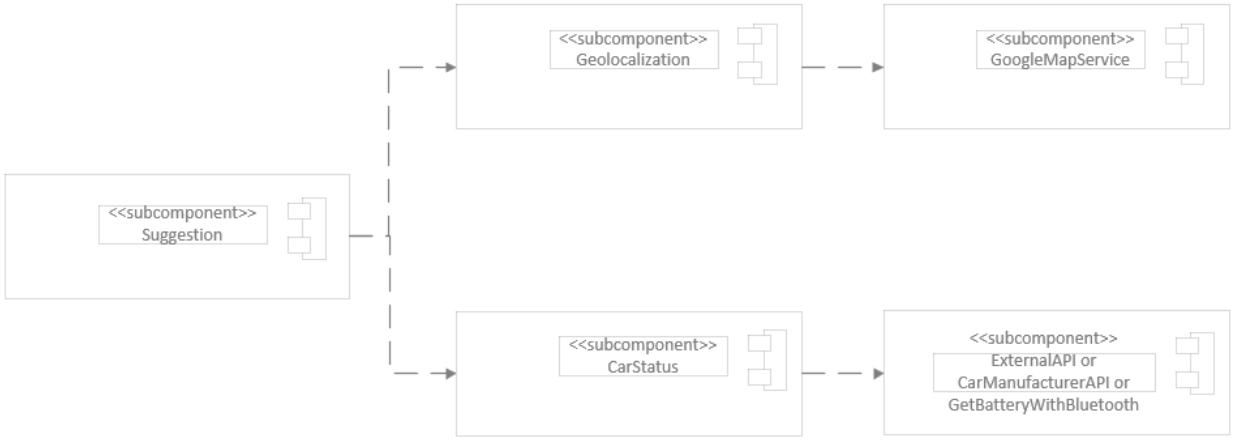

Figure 44 Integration: Suggestion, Geolocation, CarStatus

- The next two steps can be implemented and tested parallel. Because their functionalities are independent.

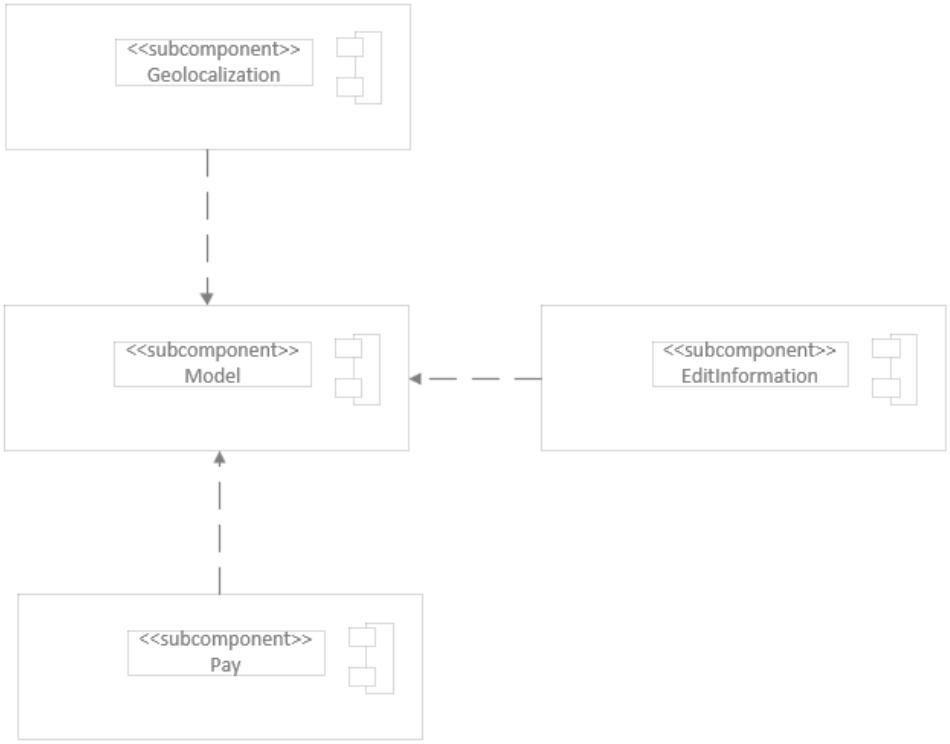

Figure 45 Integration: Model, Geolocation, EditInfomartion,Pay

6. EFFORT SPENT

6.0.1

Table 5 Effort spent for student 1 - Alireza Yahyanejad

Task	Time Spent
Introduction	1.5h
Overall description	2h
Specific requirenments	8h
Formal analysis	11h
Reasoning	9h
Total	31.5h

6.0.2

Table 6 Effort spent for student 2 - Mohammad Hosein Behzadifard

Task	Time Spent
Introduction	0.4h
Overall description	1h
Specific requirements	1.5h
Formal analysis	2h
Reasoning	1.8h
Total	6.7h

6.0.3

Table 7 Effort spent for student 3 – Alireza Azadi

task	Time Spent
Introduction	0h
Overall description	2h
Specific requirements	3h
Formal analysis	1.2h
Reasoning	0h
Total	6.2h

7. REFERENCES

- Specification Document: “Assignment RDD AY 2022-2023.pdf”
- Course slides
- https://evroaming.org/app/uploads/2021/11/OCPI-2.2.1.pdf