Politecnico di Milano
School of Industrial and Information Engineering

Software Engineering 2 Project

e-Mobility for All

Design Document

Authors:
Enrico Brunetti
Matteo Gionfriddo

Academic Year 2022/2023

Milano, 07/01/2023
Version 1.0

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The purpose of this document is to detail the design of the software and of the architecture regarding the eMall system. The analysis has been made in a more detailed approach for the description of each component and the overall architecture of the system, by also covering the relationships between each one of them. Furthermore has been inserted mockups of Mobile Application for End-Users and Web interface for CPOWs and a plan for the implementation, testing and integration of the system.

1.2 Scope

eMall is an application system designed for help End-Users to take advantage of the services for charging electric vehicles and helps Charging Point Operators manage associated charging stations.
The End-Users can use the mobile app to browse a map that displays the charging stations nearby, their cost and any special offer they have and book and use an electric vehicle charge.
The CPO Workers can use the web application to manage their associated charging stations and interact with DSO in order to acquire energy furniture. The system is supported by APIs provided by a map service and suited for interaction between the various providers (eMSPs, CPOs, and DSOs) as explained in the RASD and in the following sections of this document.

1.3 Definitions, Acronyms, Abbreviations

- *API*: Application Programming Interface.
- *DBMS*: Database Management System.
- *eMSP*: e-Mobility Service Provider.
- *CPO*: Charging Point Operator.
- *CPOW*: Charging Point Operator Worker.
- *CPMS*: Charge Point Management System.
- *DSO*: Distribution System Operator.

1.4 Revision history

- 07-01-2023 Version 1.0.

1.5 Reference Documents

- Assignment Document "Assignment RDD AY 2022-2023_v3.pdf"
- RASD Document "RASD1.pdf"

1.6 Document Structure

The document consists of 7 different chapters:

Chapter 1: in this first chapter the purpose and the scope of this Design Document for eMall system are briefly presented. Furthermore, some basic information about acronyms and abbrevations is provided in order to help in the understanding of the whole document.

Chapter 2: the second chapter purpose is to describe and motivate design choices. In particular, it starts with an overview of the chosen architecture from a more general and abstract perspective first to a more specific one consisting in the description of components then. After, interdependencies and interactions between different components are shown through different diagrams. In the end, also architectural styles, patterns and other minors design decisions are briefly reported.

Chapter 3: here the two different types of user interfaces are presented. In particular, different mockups show how interfaces should be once implemented.

Chapter 4: the fourth chapter contains all informations about requirement traceability. More specifically, it's described which requirements are satisfied by which components in a sort of mapping.

Chapter 5: in this chapter, the testing implementation and integration for the system are described.

Chapter 6: in this chapter, the effort spent by each team member is briefly presented in order to provide an overview of how the work has been done.

Chapter 7: the last chapter contains the references used during the developing process of the document.

2 Architectural Design

2.1 Overview:

The architectural style chosen for the eMall system is a three-tiered architecture which exploits the client-server paradigm. Furthermore, the system can be divided into three different subsystems: the presentation layer, the application layer and the data layer. Each tier is responsible for one of the three layers. The first layer, the **Presentation** layer is made by the couple Mobile application-eMSP (that will be used by an End-User for doing actions such that view all prices and offers of nearby charging stations and/or booking and use an electric charge) and the Web application (that is destinated for a CPOW that needs to manage their associated charging stations and communicate with DSO in order to make energy furniture agreements) and provides the GUI of the system. The second layer, the **Logic** layer, implements all the logic of the application receiving and executing the requests done by the End-User or by the CPOW. The third layer, the **Data** layer stores and maintains all the data that the system needs in order to provide correctly the services offered by the system, such that all the users accounts, all bookings and all information about the charging stations registered.
The aim of the system architecture is to allow multiple eMSPs interact with multiple CPMS. Every eMSP is connected from one side to his associated Mobile Application in order to provide the Presentation layer, and from the other side communicates with the Logic layer. In this way, every End-User side GUI will have the same fundamental structure (that is optimized in order to communicate with the Logic layer of eMall), but will contain different design decisions that are different from one eMSP to another. On the other hand the interaction with the CPMS is made by a proper CPMS API that is directly connected with the system. As can be seen clearly in the next section, the Logic layer of eMall can interact with a CPMS API in different ways, with the purpose to take into account every request that can be done by a CPOW.

2.2 Component view

The figure 2.1 is a *Component Diagram* for the *eMall* system that displays in an high level how every component are connected each other.

In this diagram when two components can communicate using different interfaces a single interface link is reported, for the sake of readability. The various components are now described and detailed:

- **Router**: it dispatches the requests coming from the users applications. Figure 2.2 represent how is composed in detail.
- **Bookings Manager**: it performs all operations related to the management of the charge bookings allowing to pick all personal bookings associated to an account and creating a

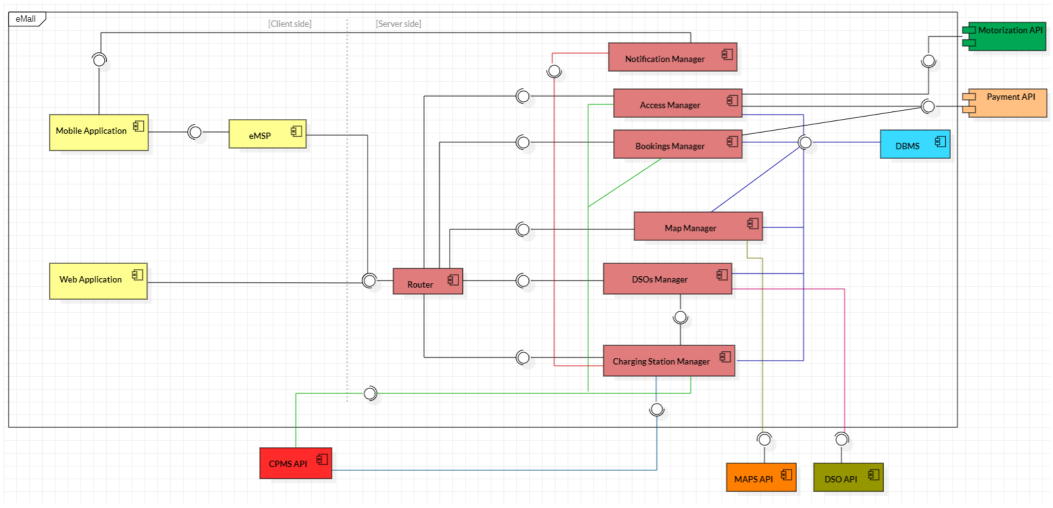


Figure 2.1: UML Component Diagram

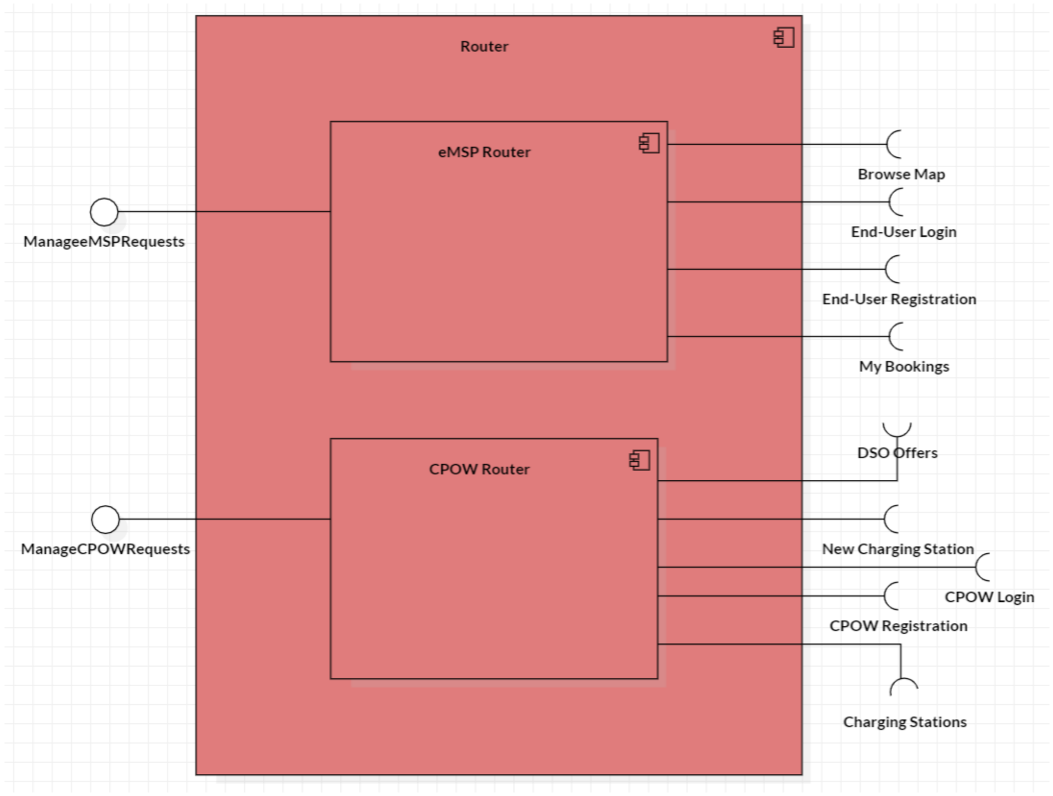


Figure 2.2: UML Component Diagram for *Router* component

new one, interacting with the *DBMS*. Moreover it allows to start a charging process talking directly with the *CPMS API* and makes the End-User payment communicating with the *Payment API*, in both cases via a dedicated interface. This component is described in figure 2.3 .

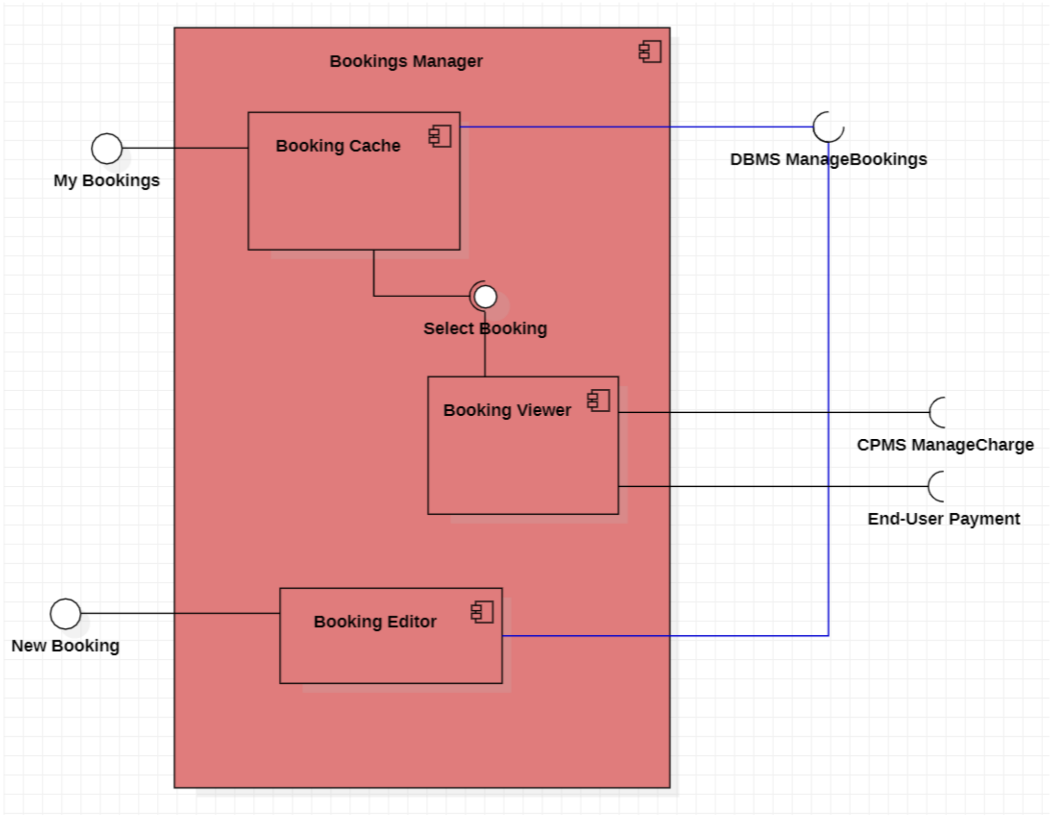


Figure 2.3: UML Component Diagram for *Bookings Manager* component

- **Access Manager**: it manages everything about registration and login of the users. It allows to create a new account and checks the login credentials communicating with the *DBMS* and also checks if the data inserted during the registration is correct through interfaces connected with the *Payment API*, *Motorization API* and *CMPS API*. This component is detailed in figure 2.4.
- **Charging Stations Manager**: it permits to view all charging stations associated to a certain CPMS and do all management operations for each one of them. It requires to communicate, by some interfaces, with the *DBMS* component , the *Notification Manager* and the associated CPMS. If set properly, the sub-component *Charging Station Viewer* can also interact automatically with the *CPMS API*. It is shown in the diagram in figure 2.5.
- **Notification Manager**: it sends to the End-User the information about the state of a charging process. Indeed, it is connected to the *Charging Station Manager* and the *Mobile Application* component via interfaces.

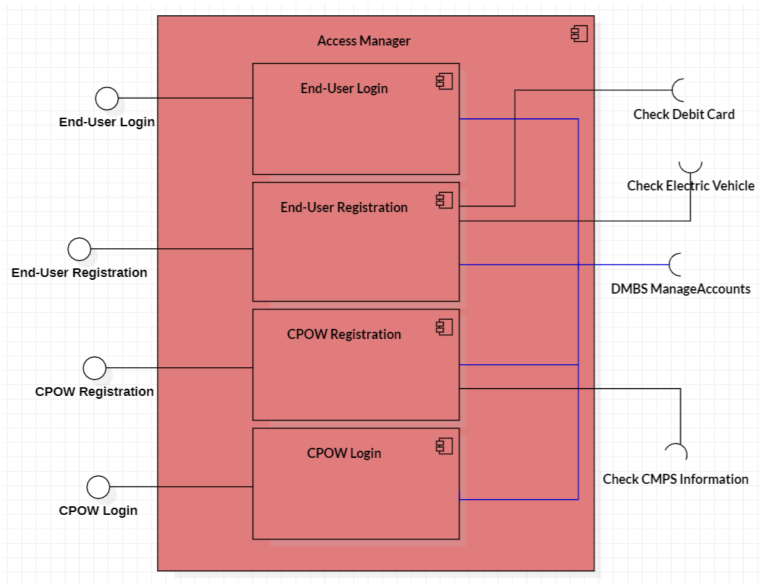


Figure 2.4: UML Component Diagram for *Access Manager* component

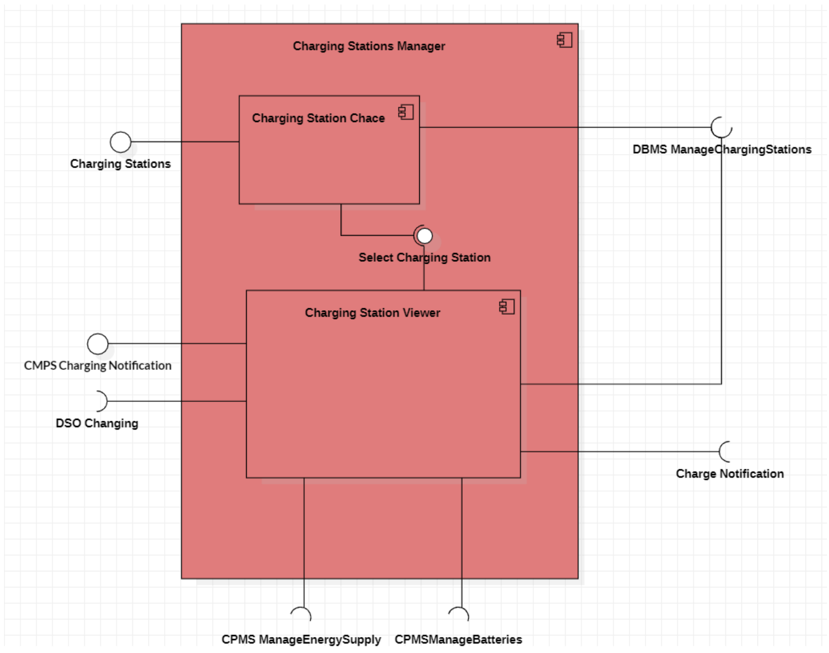


Figure 2.5: UML Component Diagram for *Charging Stations Manager* component

- **Map Manager**: it builds the charging station map interacting with the *DBMS* and with the *Maps API*
- **DSOs Manager**: it deals with everything about the management of DSO agreements. It is connected with the *DSO API* in order to capture the DSOs offers and contact a DSO in order to make an energy furniture agreement and is also connected to the *DBMS* with the purpose to get all DSO energy furniture of a certain CPMS.
- **DBMS**: this component manages the operational database.
- **Motorization API**: is an external component used to check the vehicle information inserted by an End-User.
- **Payment API**: is an external component used to check the debit card information inserted by an End-User.
- **Maps API**: is an external component used to retrieve maps for the realization of the charging stations map.
- **CPMS API**: is an external component that permits to communicate with a CPMS.
- **DSO API**: is an external component that allows to interact with a DSO.
- **Mobile Application** and **eMSP**: they constitute together the *Presentation Layer* for the mobile devices allowing the End-Users to use all services that the system offers, such that view nearby charging stations offers and book and use electric vehicle charges,
- **Web Application**: this component is the *Presentation layer* of the web app which allows CPOW to interact with his associated CPMS in order to take management decision of the related charging stations.

Note : **Motorization API**, **Payment API** and **Maps API** are described in detail in the *Components Interfaces* section.

2.3 Deployment view

A more detailed description of the system is given in this section. In figure 2.6 there is an UML Deployment Diagram which shows the allocation of the software components in the physical tiers of the system.

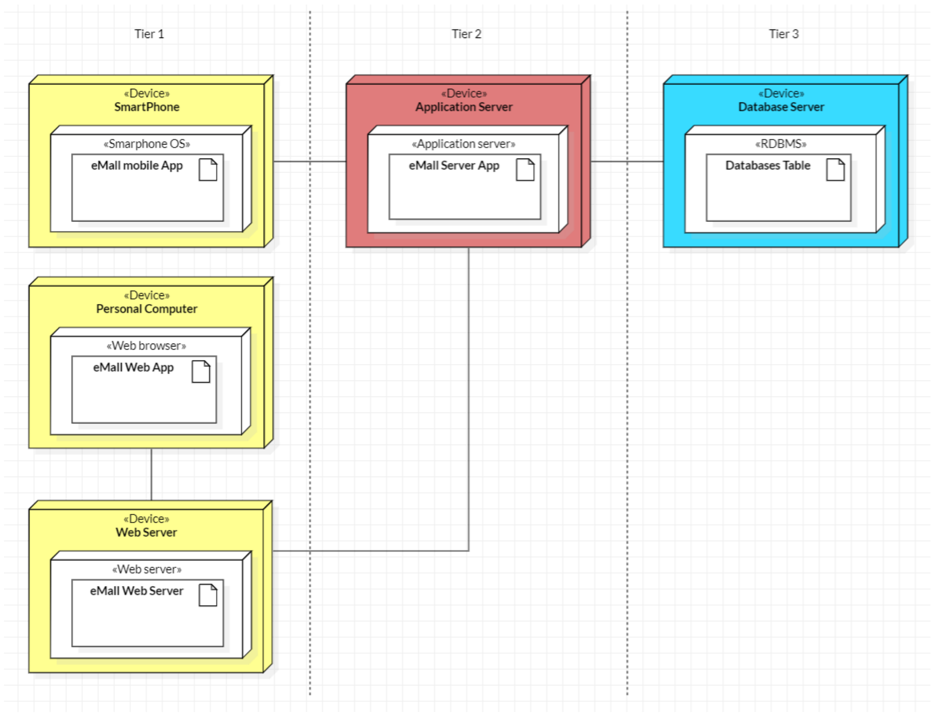


Figure 2.6: UML Deployment Diagram

As can be seen in the component diagram (figure 2.1), the different colors indicate the allocation of components. It's evident that about the first tier, the *Smartphone* contains the mobile application while the *Personal Computer* and the *Web Server* contains the *Web application*.
In the application logic all 'Manager' components are provided by the *Application Server* while in the Data tier the *DBMS* component is deployed by the *Database Server*.
Is important to notice that from the Web side, the *Presentation Layer* is contained by the *Web App* that actually provides the graphic interface of the application, and by the *WebServer* that however handle the client requests and responses interacting with the *Application Layer*.

2.4 Runtime view

Figures 2.7, 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, 2.14 are UML sequence diagrams exploited to show how the different components of the system interact each other during the execution of the most important functions provided. In order to show more cleaner and comprehensible diagrams, we assume that in all figures after 2.8 users are already correctly logged in.

In the sequence diagram of figure 2.7 the registration process of an End-User to the eMall system through the **Mobile Application** is shown. First of all the potential user must compile the registration form and, as a first check, the **Mobile Application** checks directly if provided data are in the correct format. Subsequently, an *endUserRegistration* request is sent to the **eMSP** the user has decided to register to, which will send it to the **Router**, which will route it to the **Access Manager**. This component communicates with **Motorization API** in order to check if the driving license provided by the potential user exists and it's active. If all data are correct the **Access Manager** sends all of them to **DBMS** in order to store information about the new End-User and, in the end, a success message is shown on the **Mobile Application**. Also the registration process of a new CPOW works in the same way, exception for the fact that the CPMS identification has to be checked instead of the driving license. So, since the representation of that by a sequence diagram is very similar to this one and trivial, is not reported.

In the sequence diagram of figure 2.8 the End-User login process and the interactions between all components involved are shown in detail. First of all, an End-User must fill the respective form with username and password on **Mobile Application**. Inserted information reaches **Router** End-User's respective **eMSP**. Then the router will route them to **Access Manager**, which will ask the **DBMS** to provide the stored password related to that specific user. Subsequently, the **Access Manager** checks if the password provided by **Mobile Application** is equal to the one provided by the **DBMS**. If it's not the case a login error is sent back to the **Mobile Application**, which allows the user to retry again inserting the correct data. On the other hand, if the data were correct, the user is successfully logged in, a new token associated with the specific logged user is generated and stored in the **DBMS** and, in the end, the End-User is notified of login success. Also in that case, the sequence diagram of CPOW login is not provided since it works exactly in the same way.

In the sequence diagram of figure 2.9 the functionality which allows the **Mobile Application** to show a map of all available charging stations is shown. Note: this function can take place independently from the fact that the End User is logged in or not since all available charging stations are shown also to not registered users. In detail GPS coordinates of End-User are routed from the **Router** to the **Map Manager** that will get an updated world map from **MAPS API**. Then, the **Map Manager** gets from the **DBMS** all information about charging stations in a certain range of distance from provided coordinates and mounts charging stations on the map, exploiting their coordinates. After, the updated map will be returned to **Mobile Application** that will load it and show it to End-User.

In the sequence diagram of figure 2.10 the whole charge booking process is presented together with relations between all involved components. First of all, a logged-in End-User must select a charging station, a socket type, and a date from the **Mobile Application**. After that, the router will receive a get availability request from the involved **eMSP** and will route it to **Booking Manager**, that will get from **DBMS** all availability for that specific day. If there is at least one are all sent to **Mobile Application**, otherwise, an error is sent back. At this point, the End-User must select, through the **Mobile Application**, a time slot from the available ones, a vehicle, and also a payment method from the ones connected to his account. A booking request will after follow the same path as before in order to reach the **Booking Manager**, which will exploit the **Payment API** in order to check if the selected payment method has enough funds for the payment and provisionally block them. If that operation has success the new booking is stored in the **DBMS** and a success message is sent back to the **Mobile Application**, otherwise, a booking error is returned.

In the sequence diagram of figure 2.11 is presented how all components react and interact with each other during a whole charging process that is assumed to have been already booked. First of all, a correctly logged-in End-User, which is assumed to be near the right charging column, must click on the start charging button of the **Mobile Application**, which will send the request with the associated *id* to the **Router**, trough the **eMSP**. Then, the request is routed to the **Booking Manager** that will ask all dates about the specific charge to the **DBMS** and then will check them. If the charge exists and it's the right time, the **Booking Manager** sends an *enableCharge* message to the **CPMS** for that specific charge at that specific time in that specific socket of that specific charging station. After that, the **Booking Manager** sends also a *charge_ ok* message to the mobile application. At this point, the user inserts the plug of his car into the charging socket and the **CPMS API** checks if the charge for that car is enabled. If it's the case the charge starts and the **CPMS API** sends periodically to the **Charging Station Manager** the current status of recharge, which will be reported, firstly to the **DBMS** and secondly to the **Notification Manager** that will send it to the **Mobile Application**, which displays charging status to the End-User. When the charge is completed, the **CPMS API** communicates it to the **Charging Station Manager** that will save it to the **DBMS** and then ask to the **Notification Manager** to send a notification of completed charge directly to the **Mobile Application**, that will show it to the user that will consequently remove the plug. To keep this heavy diagram a little cleaner has not been represented the fact that when the charge is complete, funds are charged to the End-User payment method from **Payment API**, and also this information is stored in the **DBMS**.

In the sequence diagram of figure 2.12 is presented the process of DSO selection from the CPOW point of view by showing how different involved components communicate with each other. First of all, an already logged-in CPOW must ask from the **Web Application** for all available DSOs for a specific charging station (this depends on the charging station location, i.e. different geographical areas may have different DSOs). Then a get DSOs request will follow the following path: **Web Application-Router-DSO Manager**. This last component will after get from **DSO API** the updated list of all available DSOs, check which ones are available for the specific charging station (after getting charging station data from **DBMS**), and then finally return back a list that is shown by the **Web Application**. At this point, the CPOW, through his web interface, selects one DSO and a change DSO request travels until reaches **Charging Station Manager**, that checks if the selected DSO is already active for that specific charging station. If it's not the case, a change DSO request is sent firstly to the **DSO Manager** that will communicate to **DSO API** the change and secondly another request is sent to **CPMS API**, in order to apply the effective changing to the charging station. At the end, the new DSO status for the charging station is stored from the **Charging Station Manager** to the **DBMS** and a success message is sent back to the **Web Application**. On the other hand, if the DSO is already selected for that specific charging station the system returns a message, and nothing changes.

In the sequence diagram of figure 2.13 is shown the process through which a CPOW can perform operations of charging station management exploiting the **Web Application**. In this particular example, we describe how that works in the case of enabling battery usage, but we can assume that the process is quite the same for all other similar operations, e.g. disabling batteries. First of all, an use batteries request for a specific charging station is routed by the **Router** from the **Web Application** to the **Charging Station Manager**. Then, this last component gets the current energy usage status of the selected charging station from the **DBMS** and checks if batteries are already enabled for that charging station. If it is the case an *already_ using_ batteries* message is sent back to the **Web Application** and nothing changes. If it is not, the **Charging Station Manager** communicates to the **CPMS API** to enable batteries and update the charging station status on the **DBMS**. In the end, a success message is sent back to the **Web Application**.

In the sequence diagram of figure 2.14 the focus is on how the **Web Application** gets all data about charging stations and is able to show them to a CPOW. When a CPOW opens his web interface connecting to the **Web Application** a request to obtain a list of all charging stations of his CPO passes through the **Router** and reaches the **Charging Station Manager**, which will get the list from the **DBMS** and return it back to the **Web Application** following the same path. At this point, when the CPOW selects a specific charging station on which to see all details and information, another get info request will follow again the path, but this time it's specific to a charging station. When this request reaches the **DBMS** it starts to send back, again on the path, all data to the **Web Application** every 5 seconds, to ensure that data are always updated (e.g. if another CPOW or the system dynamically has disabled batteries). This loop ends when the CPOW exit the page of that specific charging station on the web interface of the **Web Application** and then the **DBMS** is informed to stop sending data.

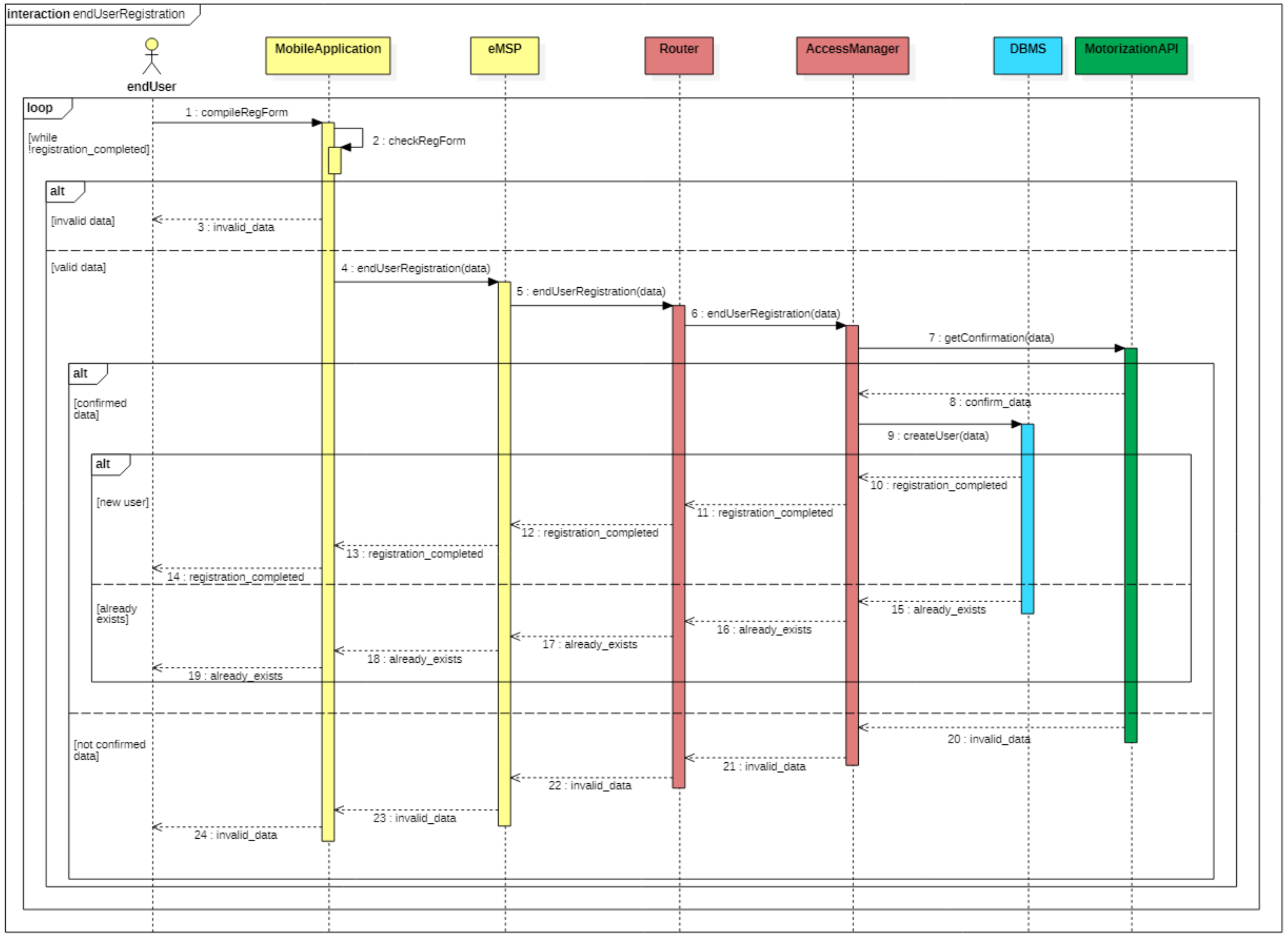


Figure 2.7: UML Sequence Diagram for the End-User Mobile App registration

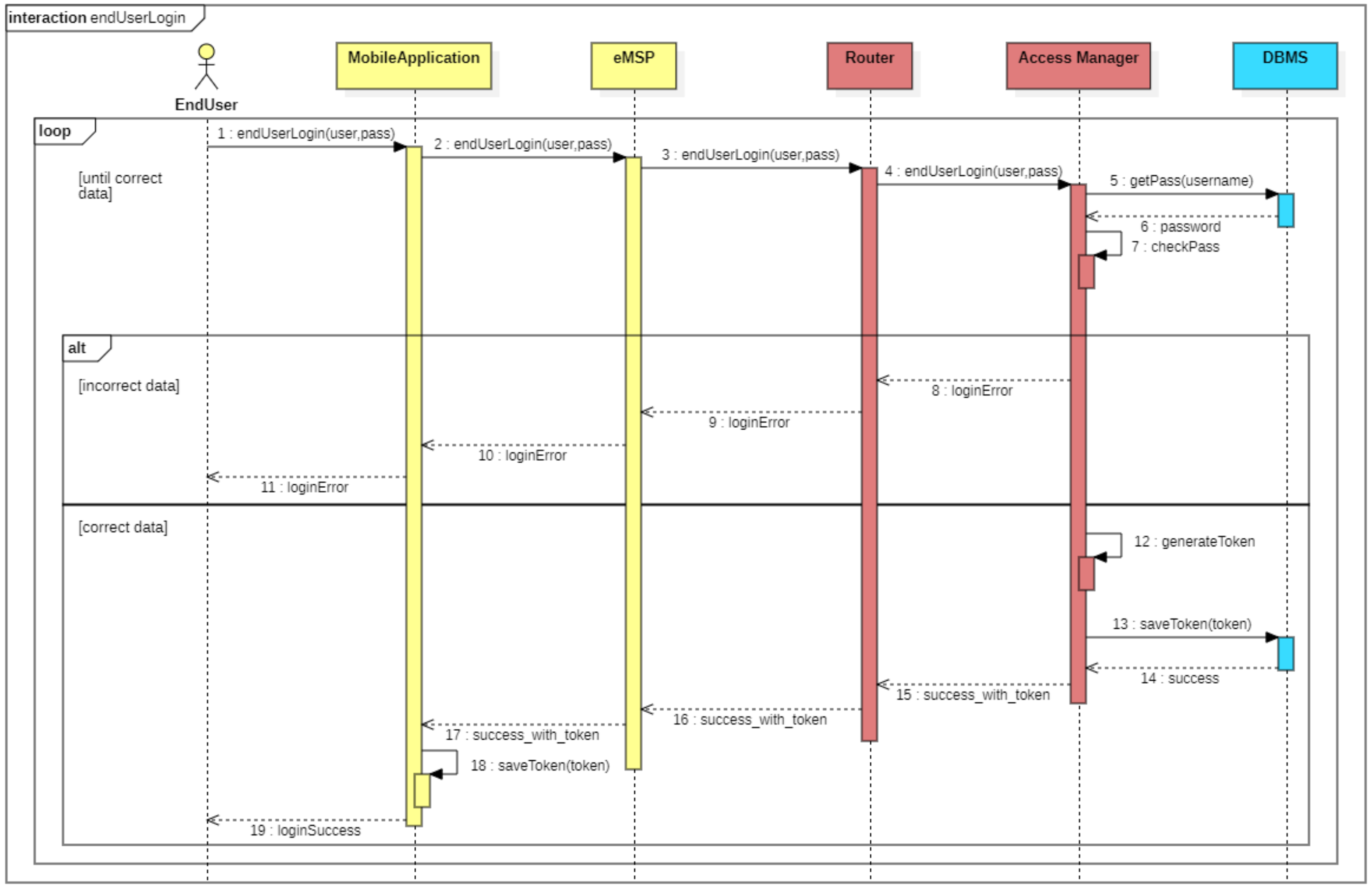


Figure 2.8: UML Sequence Diagram for the End-User Mobile App login

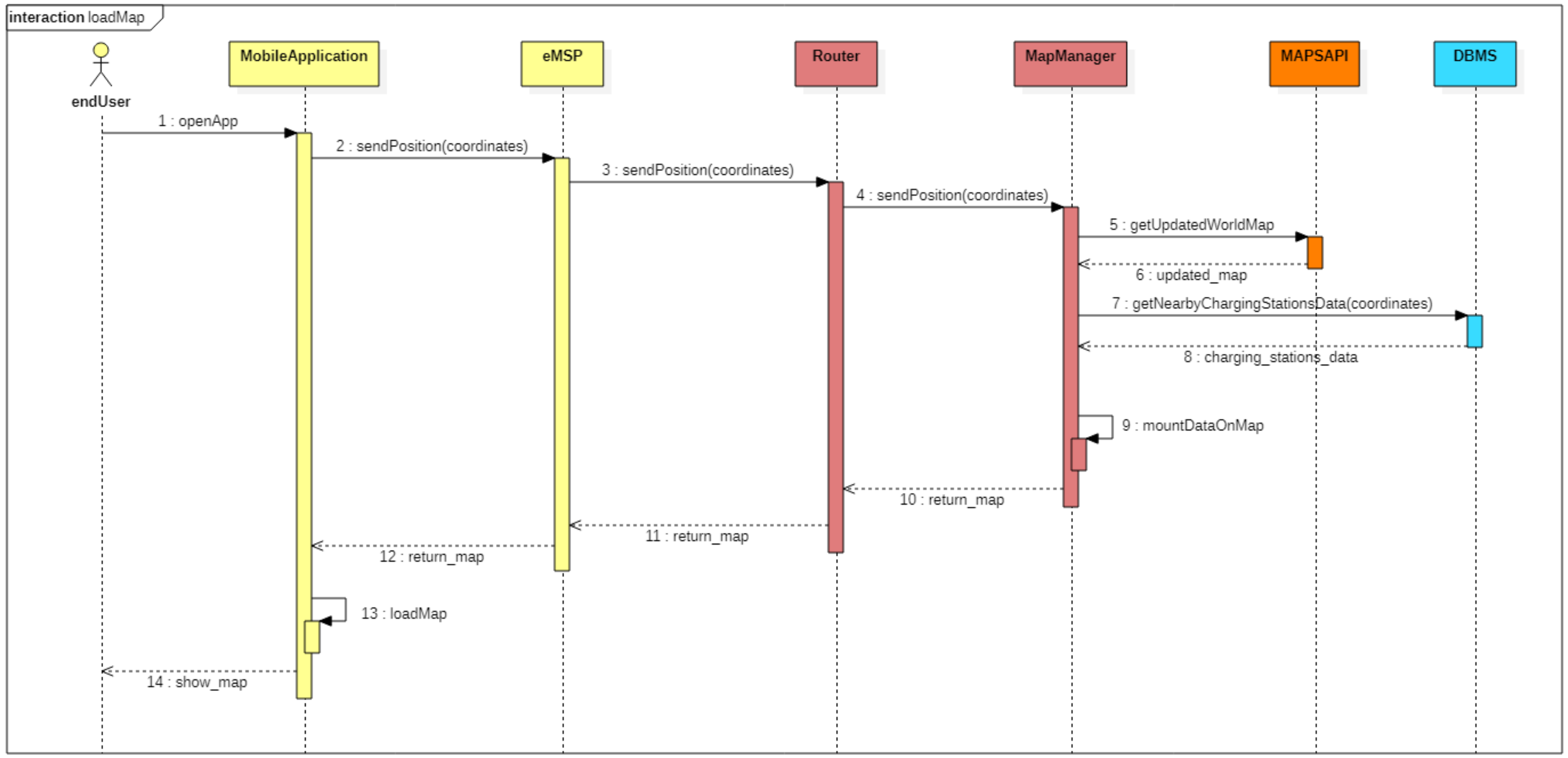


Figure 2.9: UML Sequence Diagram for Mobile App map load

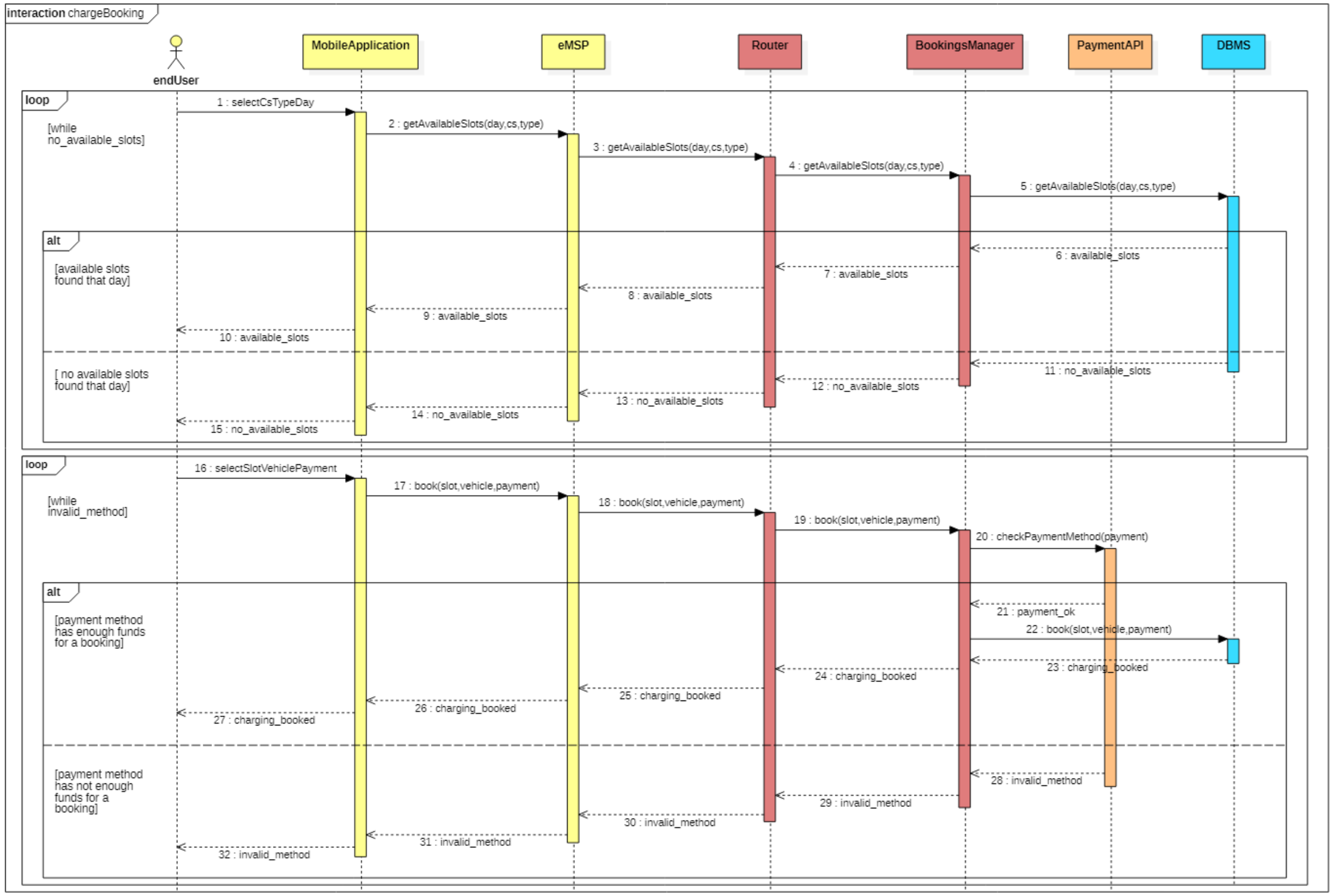


Figure 2.10: UML Sequence Diagram for End-User charge booking

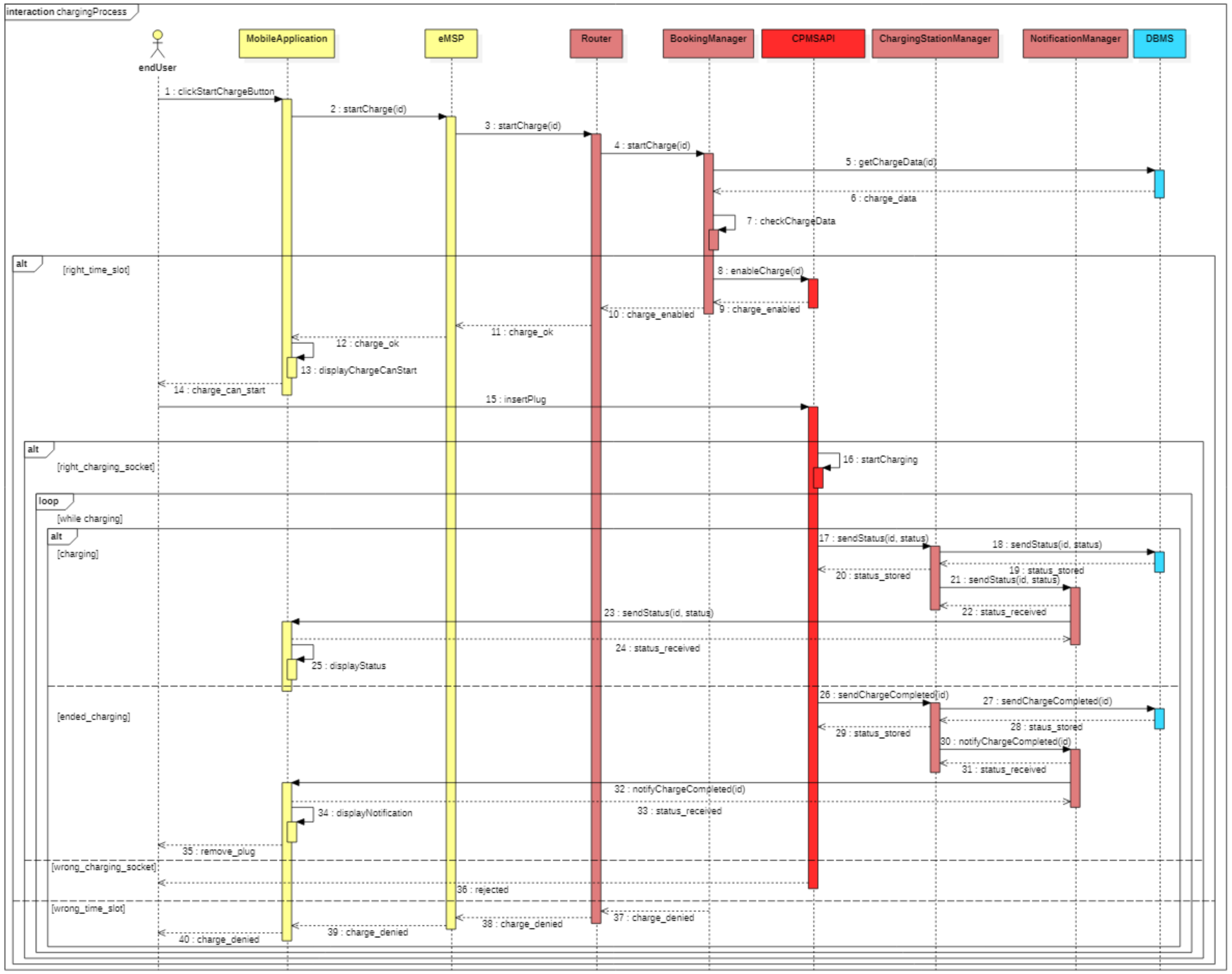


Figure 2.11: UML Sequence Diagram for End-User charging process

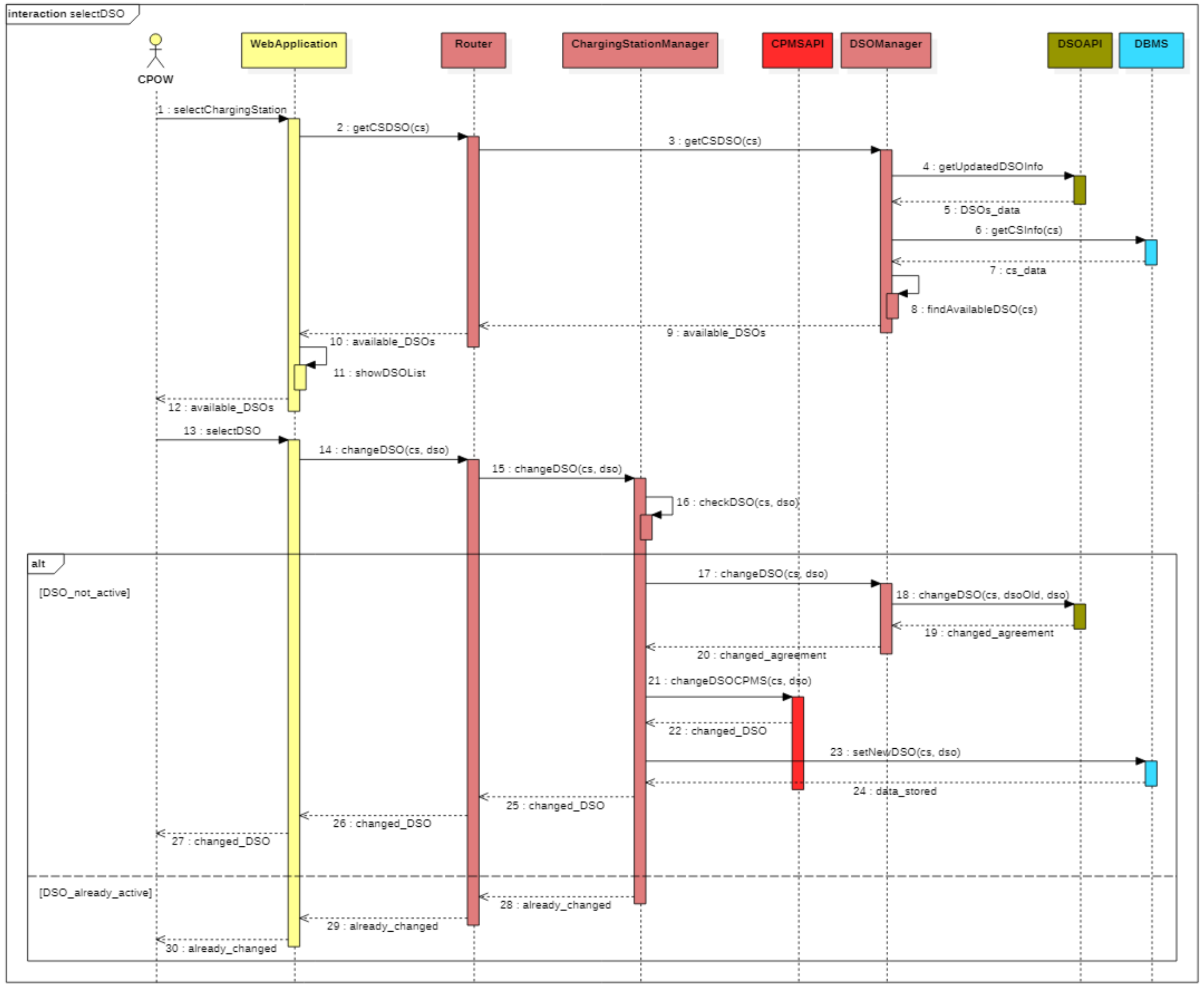


Figure 2.12: UML Sequence Diagram for CPOW selecting a DSO for a charging station

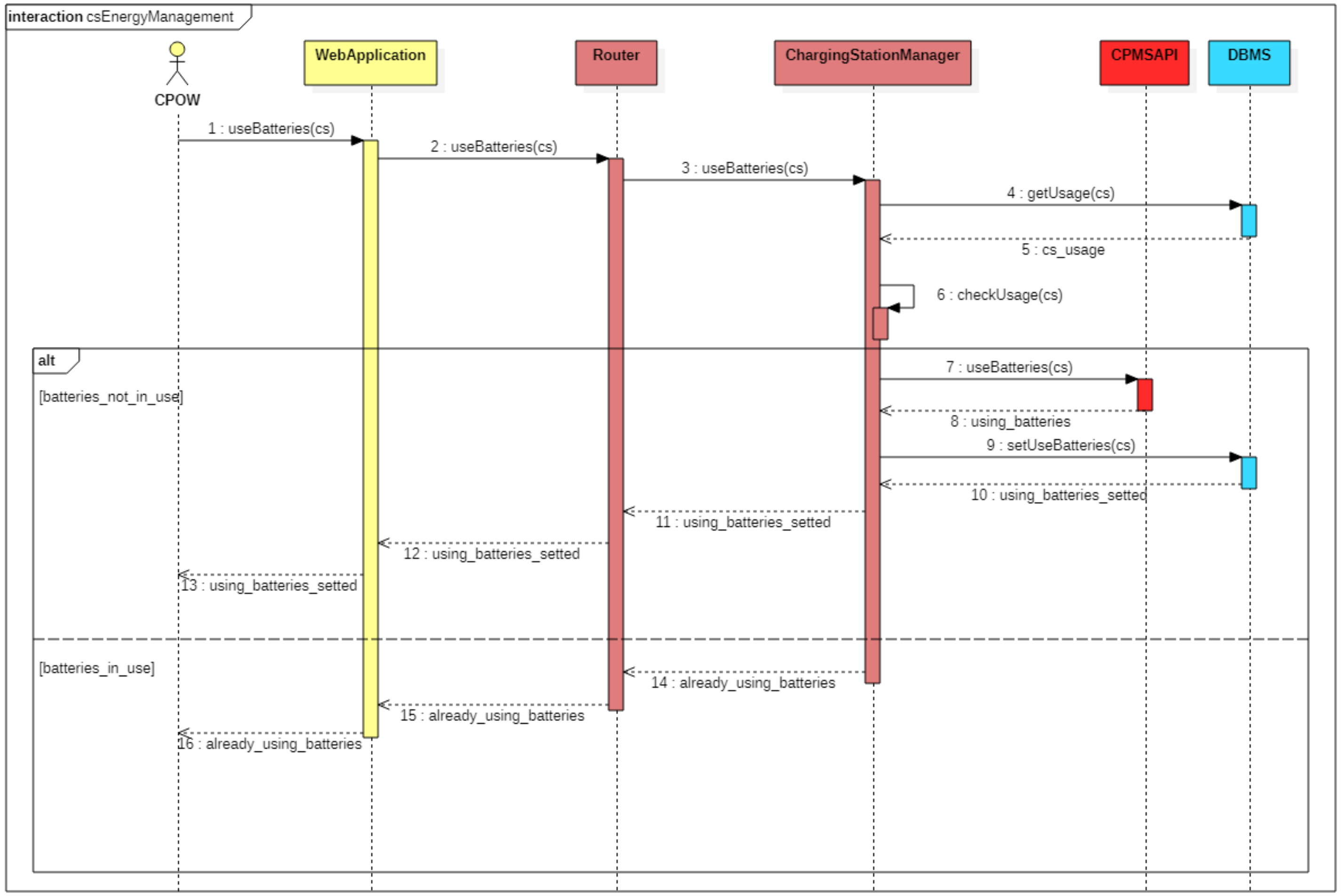


Figure 2.13: UML Sequence Diagram for CPOW selecting batteries usage for a charging station

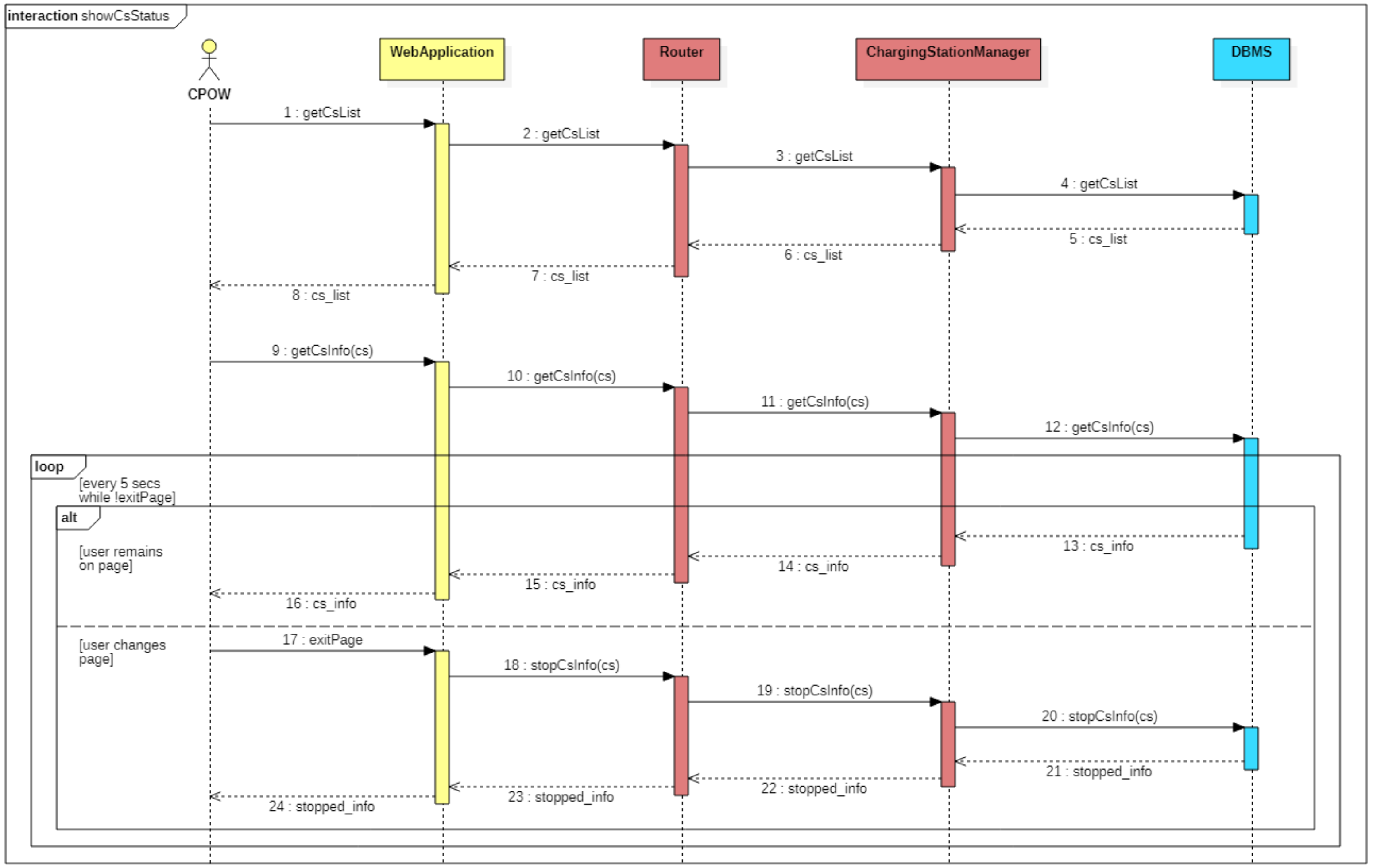


Figure 2.14: UML Sequence Diagram for Web Application showing to a CPOW date about a charging station

2.5 Component Interfaces

In the diagram of figure 2.15 all component interfaces are described with respect to what is shown in the component diagram of figure 2.1.

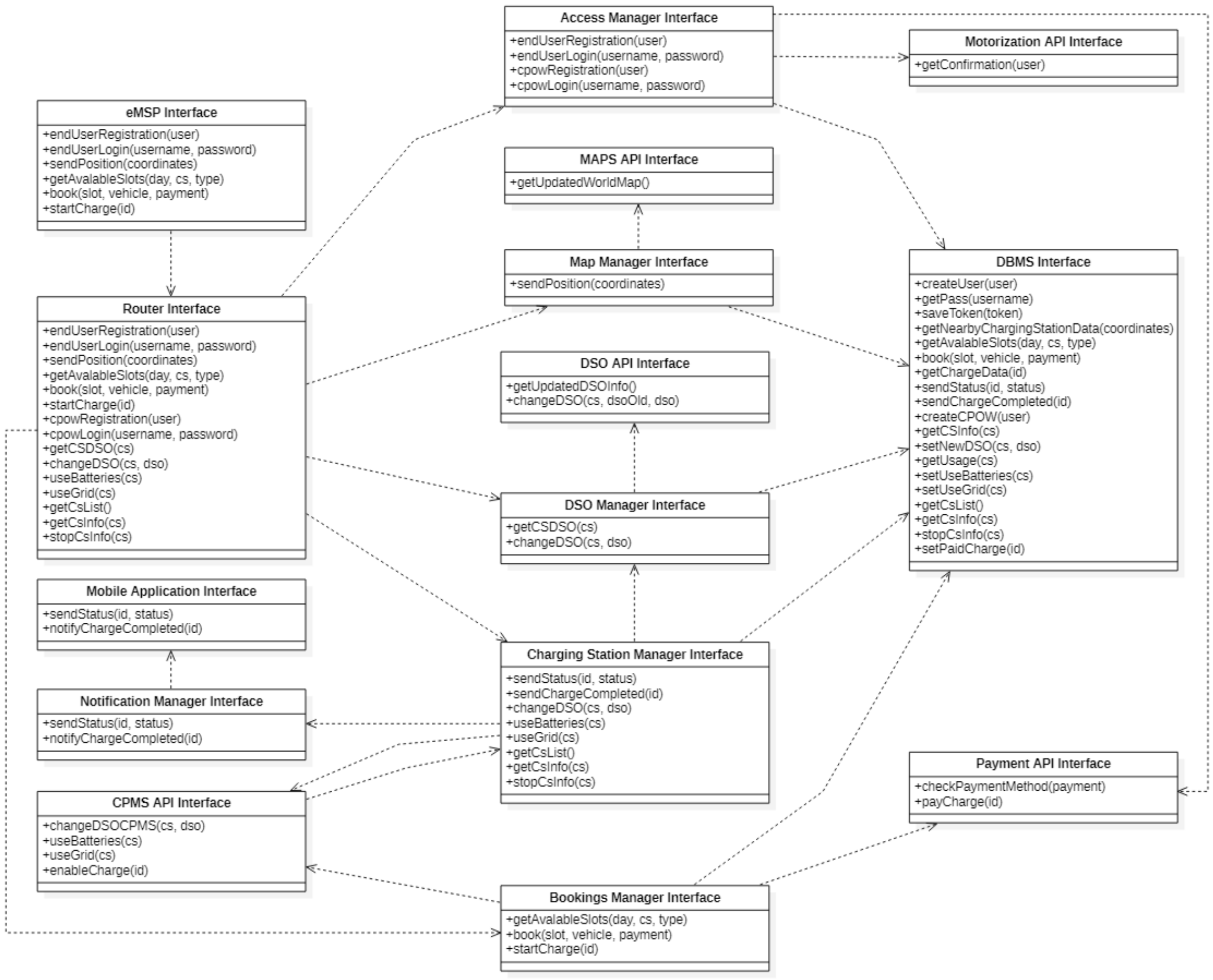


Figure 2.15: UML Interfaces Diagram

In figure 2.16 is reported the Class Diagram already provided in the RASD document with a modification. During the develop of Design Document, considering all system requirements and functions to be implemented would have been useless to maintain a ChargingColumn class which has been removed with respect to the RASD version of the diagram. Charging columns numbers are now described as attributes of ChargingSocket class since their only usage could be for descriptive purpose. The diagram is still high level and it could be more refined before the effective implementation.

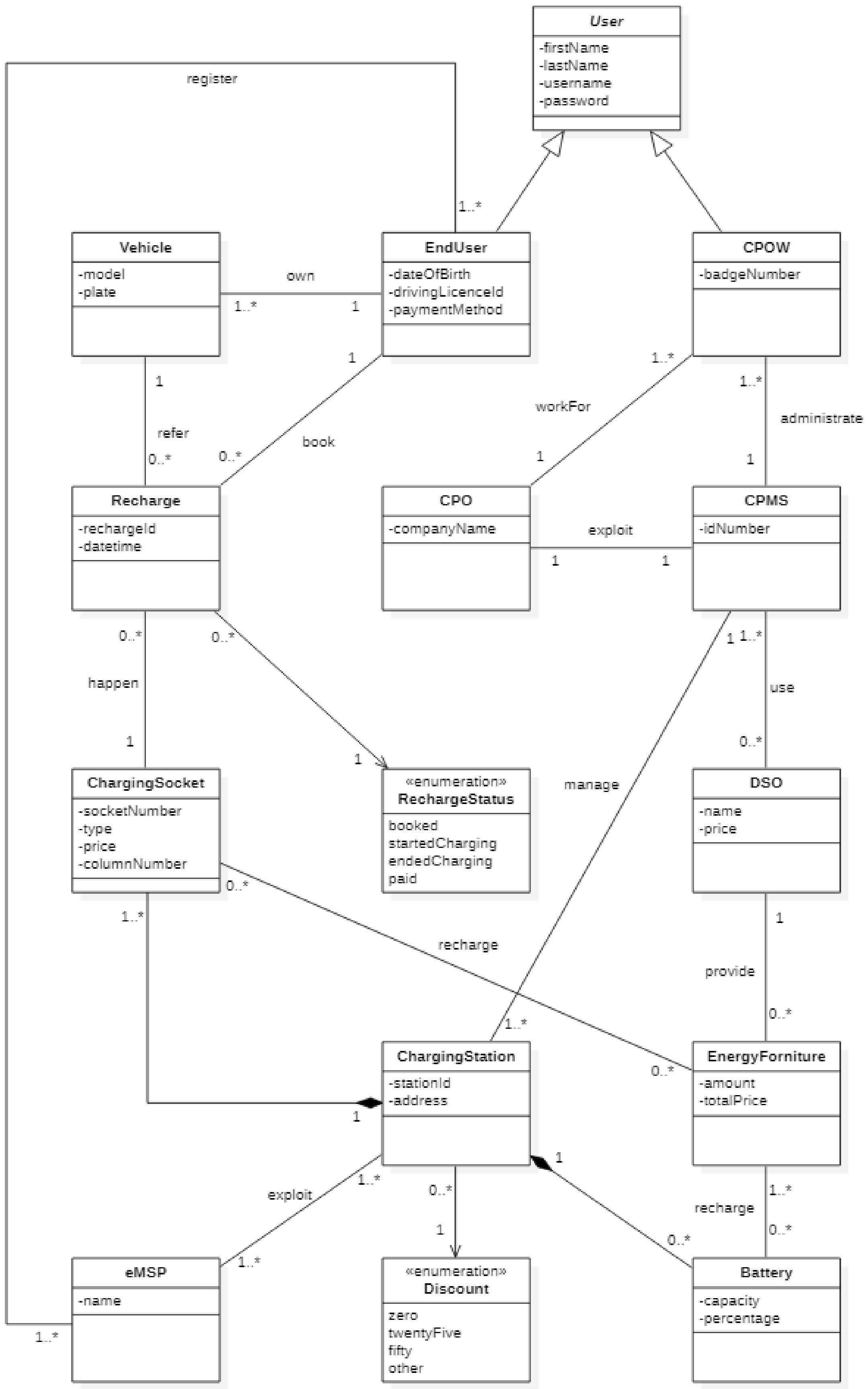


Figure 2.16: UML Class Diagram

2.6 Selected architectural styles and patterns

2.6.1 Model-View-Controller Pattern (MVC)

The eMall system is built exploiting the Model-View-Controller pattern in order to achieve some important benefits like:

- Easier building of large-scale applications.
- Easily Modifiable: allows easy modification of the entire application and helps to increase flexibility and scalability.
- Faster development process: As there is segregation of the code among the three levels, developing web applications using the MVC pattern allows one developer to work on a particular section (say, the view) while another can work on any other section (say, the controller) simultaneously. This allows for easy implementation of business logic as well as helps to accelerate the development process.
- Easy planning and maintenance: MVC pattern is also helpful during the initial planning phase of the application because it gives the developer an outline of how to arrange their ideas into actual code. It is also a great tool to help limit code duplication, and consequently, allow easy maintenance of the application.
- Multiple views: following the MVC pattern, it's also easy to develop more view components for model components limiting code duplication as it separates data and business logic. This was particularly useful for eMSP and Web Application view components.

In the component diagrams of figure 2.1 and also in other diagrams like the sequence ones, different colors have been used to underline the different tiers of the pattern and their different functionalities. In particular, yellow components are view components, red components are controller components and blue components are model ones.

2.6.2 Tiny client

The eMall system is developed on a three-tier architecture, as previously explained. Furthermore, the client is thin, which means that, in this specific case, the client doesn't need to make particular computations or know anything about the system logic but has only a presentation purpose to the final user. This implies hardware resource optimization, reduced software maintenance, and improved security and in particular, is meaningful for both Mobile Application and Web Application.

2.6.3 RESTful architecture

The eMall system has been developed following also the REST architecture. The main motivations are:

- RESTful system can be used by any type of client, regardless of programming language or execution environment.
- REST uses a stateless approach, which means that request state information is provided by the client to the server, allowing for greater scalability compared to other architectures that require server-side storage of state information.

- REST is based on the HTTP standard, which makes it easily interoperable with other technologies based on HTTP, such as our Web Interface.
- REST uses a cache-based approach, which can improve system performance as responses can be easily cached and reused in the event of future requests.
- REST uses a simple interface based on URI and HTTP verbs, which makes it easy to understand and use.

2.6.4 Database

The final database choice for eMall is a relational one since all data have pretty well-defined structures. We have a lot of entities (e.g. user, vehicle, booking, charging station, cpow, etc.) whose instances have been easily stored in relational tables that are accessed by SQL queries.

2.7 Other design decisions

2.7.1 Maps API

Since for this system is not worthy to develop, maintain and update a dedicated map system the final decision is to rely on an external service represented by the MAPS API Component. In particular, we will exploit google maps APIs, since are well documented, easy to implement and with a lot of features that allow to add markers and signals that represent the different charging stations directly on the map that is in the end sent to the Mobile Application.

2.7.2 Motorization API

Another necessity of the system is to check if the driving licences provided by users are vaild, in other words if users are enabled to drive the vehicle they want to book a charge for. This is made by the Motorization API component which exploits, for italian driving licences, the APIs, provided after have received an authorization, from the government website of ministry of infrastructure and transport. For driving licences of all other countries is assumed to obtain the same data from respective government websites.

2.7.3 Payment API

Also for the payment processing and for checking if the amount available on end user debit or credit cart is enough isn't worthy to develop a proprietary system. The final decision is to exploit a third party service, called Stripe, that supports a wide range of payment types, including major credit and debit cards, Apple Pay, Google Pay, and other digital wallets. It provides also an advanced fraud protection and compliance features to help ensure the security of transactions. Furthermore has a good documentation and a customer service that can help in setting up this side of the system.

3 User interface design

In this chapter are presented mobile application and web application mockups.

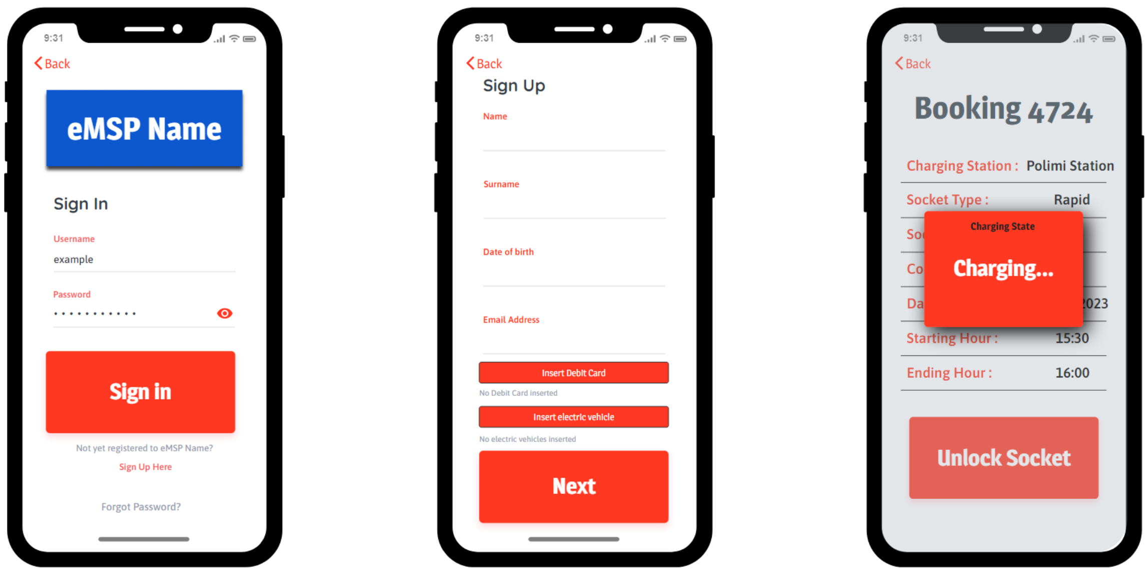


Figure 3.1: Mobile Application Mockups for End-User Log in, End-User Registration and Vehicle Charging State Popup Screens

In the figures 3.6, 3.7 are shown activity diagrams that describe how a End-User can navigate in the UIs offered by the application. Notice that End-Users can quit the application from any state to reach the end state of the diagrams. No end state is represented and so are all the arrows that lead to them; this has been done for the sake of readability of the diagrams.

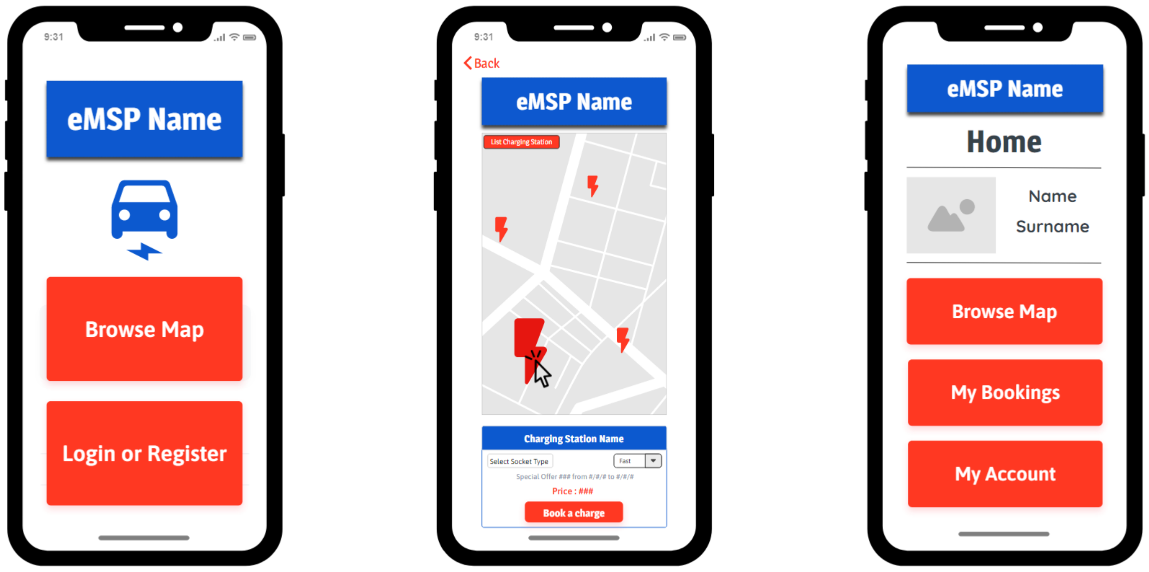


Figure 3.2: Mobile Application Mockups for Home Page, Charging Stations Map and Personal Home Page Screens

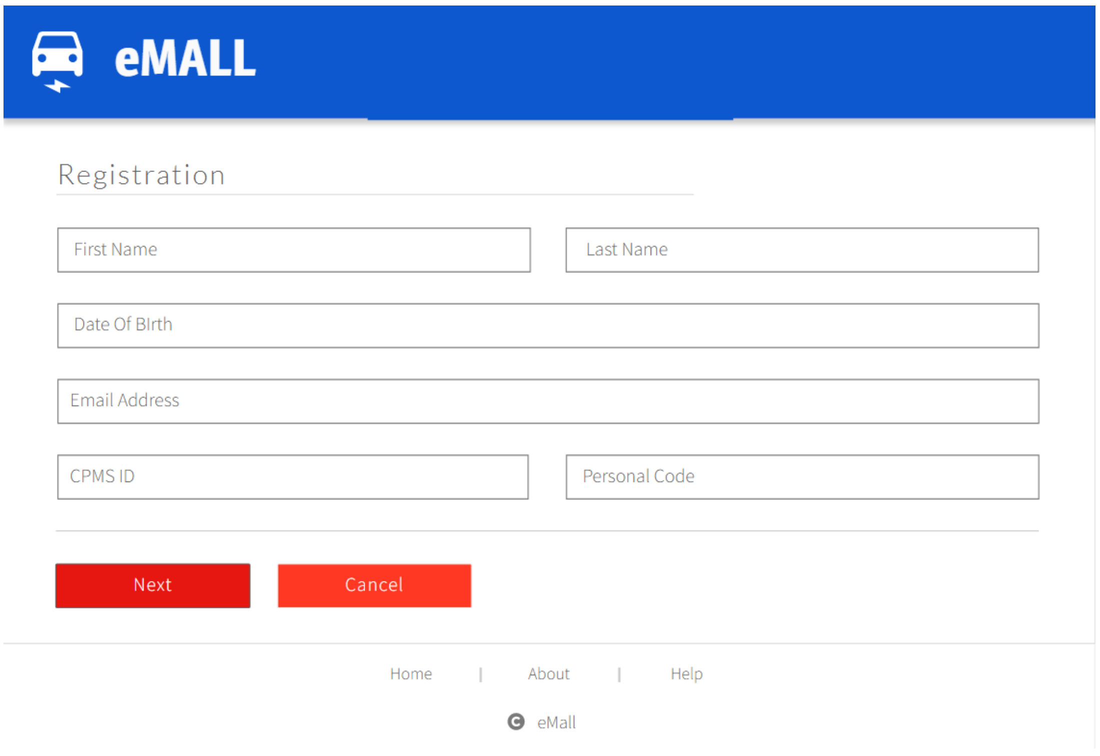


Figure 3.3: Web Application Mockup for CPOW Registration Screen

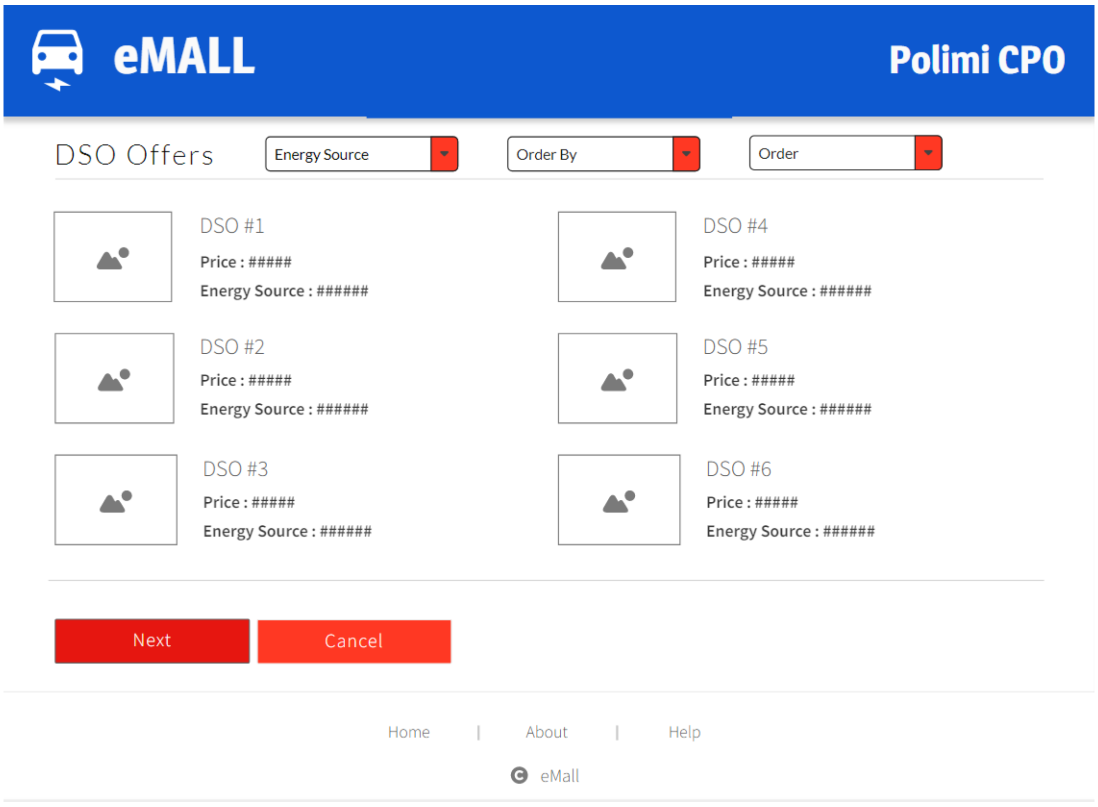
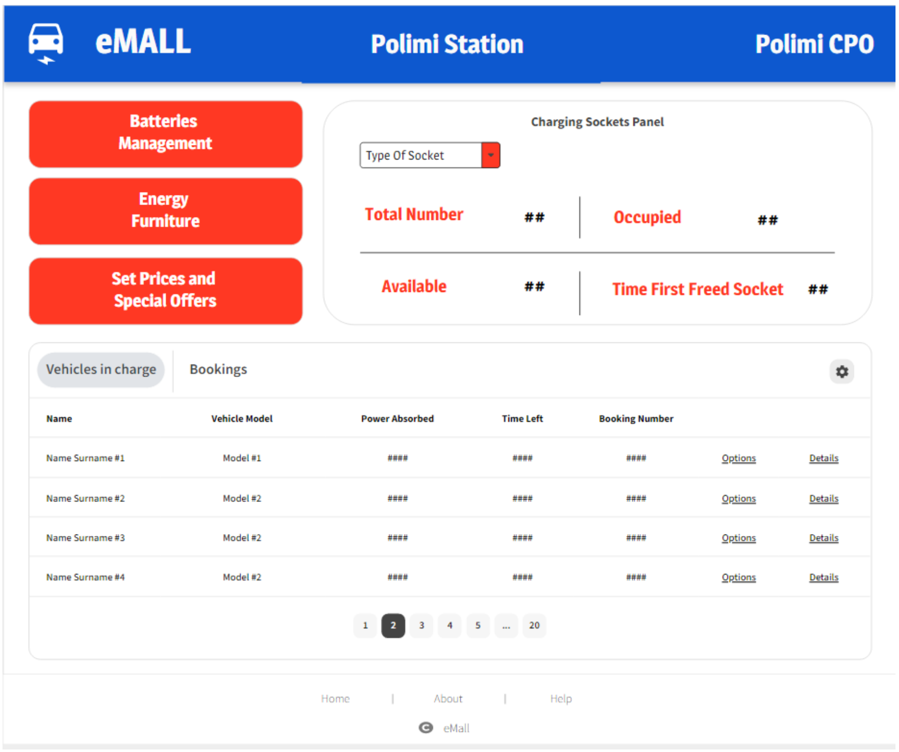


Figure 3.4: Web Application Mockup for DSO Offers Screen



Name	Vehicle Model	Power Absorbed	Time Left	Booking Number		
Name Surname #1	Model #1	####	####	####	Options	Details
Name Surname #2	Model #2	####	####	####	Options	Details
Name Surname #3	Model #2	####	####	####	Options	Details
Name Surname #4	Model #2	####	####	####	Options	Details

Figure 3.5: Web Application Mockup for Selected Charging Station Screen

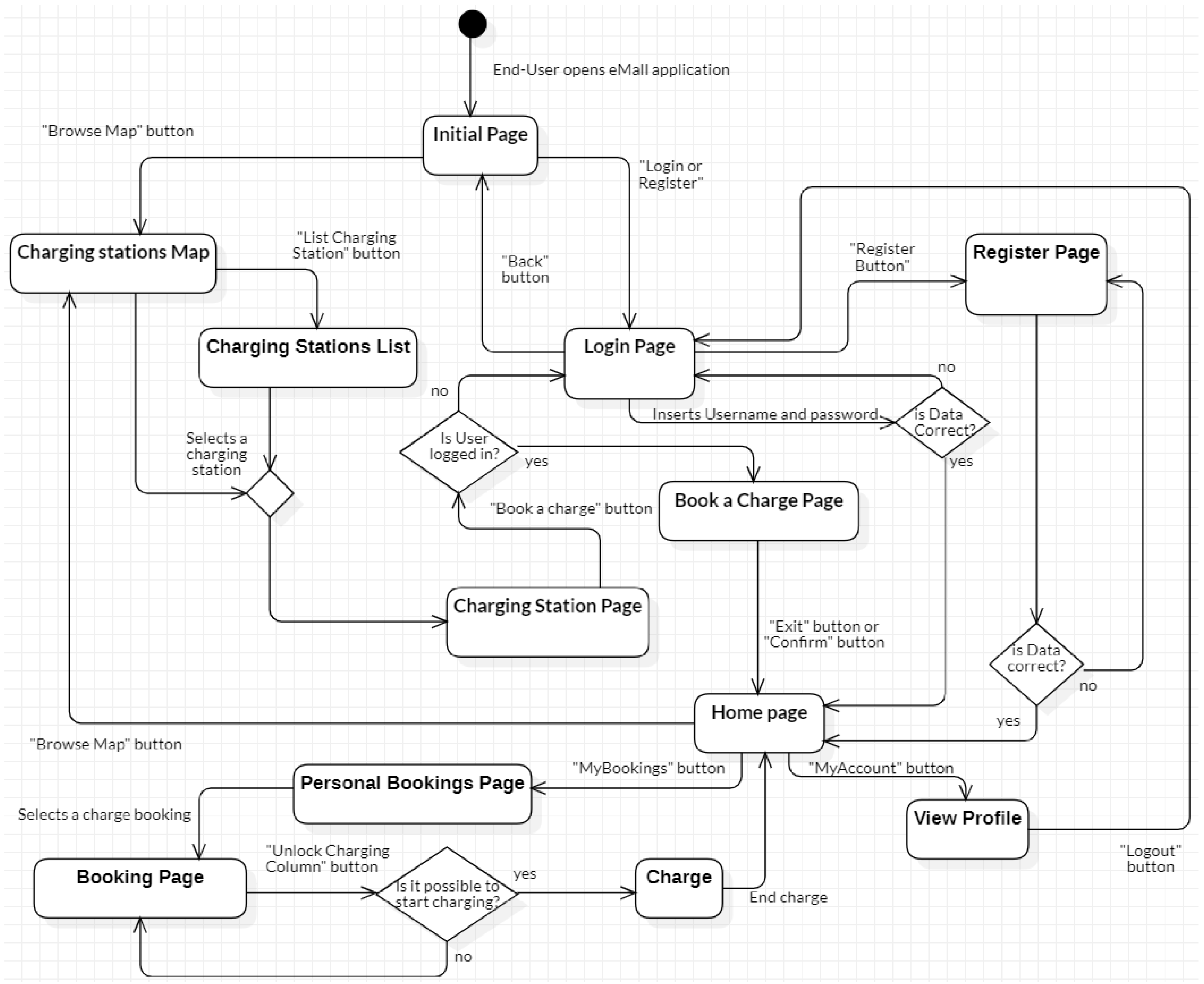


Figure 3.6: UML Activity Diagram for the End-User Mobile App navigation

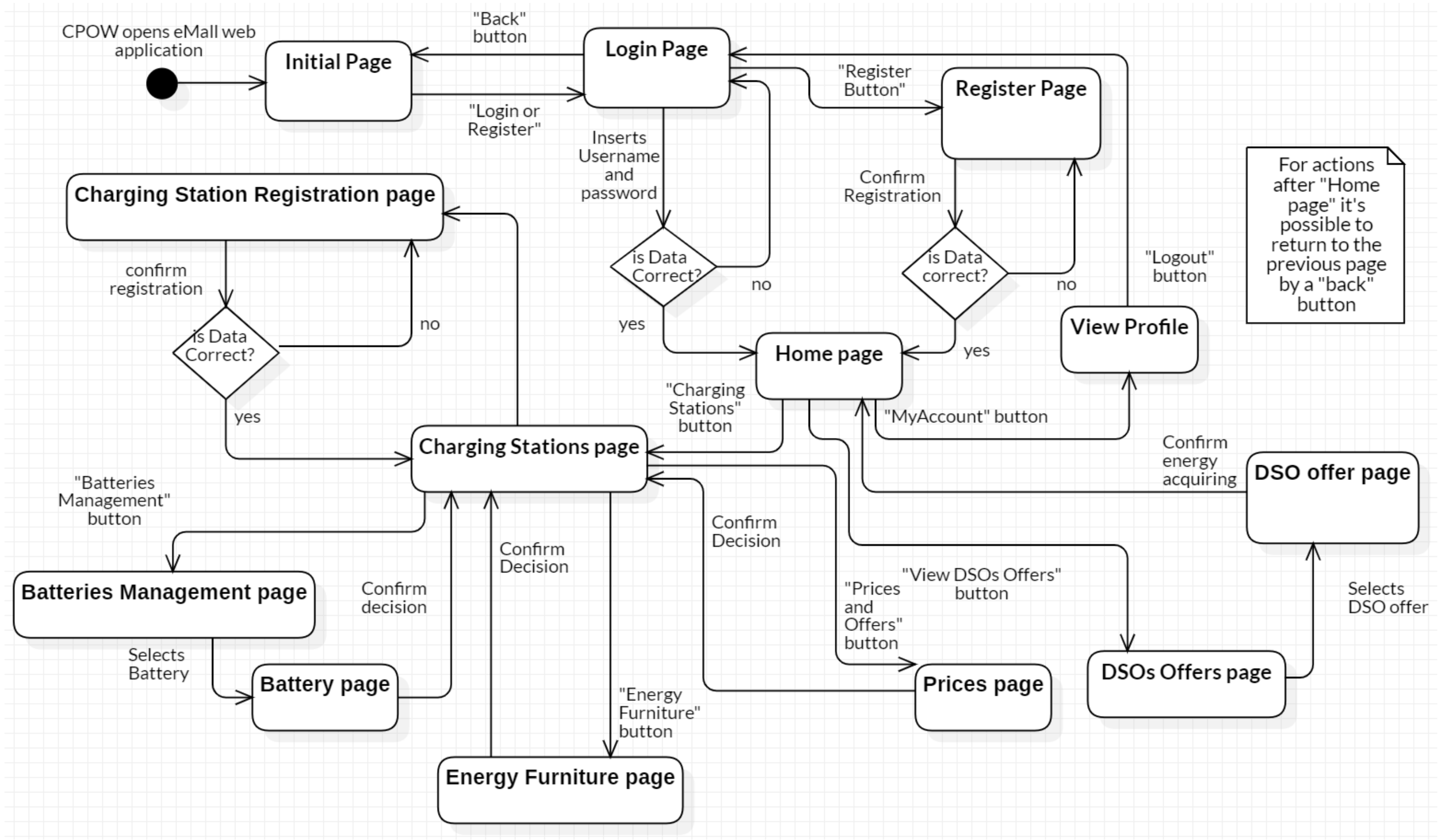


Figure 3.7: UML Activity Diagram for the CPO Worker Web App navigation

4 Requirements Traceability

In this section is shown how the requirements are actually ensured and which components actually ensure them. It's worth to notice that for the sake of simplicity Mobile Application and eMSP components are considered as one because they work together doing the same activities, and the Router has been ignored but it's always involved when dealing with a request coming from one of the clients. Table 4.1 has been added to highlight for each component what requirements it ensures.

Component	**Requirements**
Mobile Application / eMSP	R1, R2, R3, R5, R6, R7, R8, R9, R10, R11, R12, R13
Web Application	R3, R5, R15, R17, R19, R21, R24, R26, R27
Notification Manager	R12, R13
Access Manager	R3, R4, R5
Bookings Manager	R9, R10, R11, R14
Map Manager	R1, R2, R9
DSOs Manager	R19
Charging Station Manager	R4, R12, R13, R15, R17, R19, R21, R24, R26, R27
DBMS	R1, R2, R4, R6, R7, R8, R15, R26, R27

Table 4.1: Table for mapping requirements to components

In the following lines is explained how the requirements are provided by the components:

- [R1] The system must allow unregistered/registered users to see a map of available charging stations and their prices and offers. This requirement is provided by the Mobile Application component , the DBMS component and the Map Manager Component. The Mobile Application allows the End-User to access the charging stations map/list that is provided by the Map Manager through the DBMS and the Map APIs.
- [R2] The system must allow unregistered/registered users to see a list of charging stations and filter on it (e.g. offers, prices and positions). This requirement is provided as the same way as requirement R2.
- [R3] The system must allow unregistered users to register as end user or CPOW (if they have a badge id). The Mobile application allows the End-User to fill the registration while the Access Manager checks if the data is correct and allows to store the account information into the database communicating with the DBMS.
- [R4] The system must verify if the all data inserted by users are correct (e.g. personal data, vehicle and payment method information, charging station ID). This requirement is provided by the Access Manager component.

- [R5] The system must allow registered end users or CPOW to log in through their username and password. This requirement is provided by the Mobile Application component, the Web Application component and the Access Manager Component. The End-User/CPOW can fill the username and password through the Mobile Application/Web Application, while the Access Manager component checks if all data is correct and allows store it into the database by the DBMS.
- [R6] The system must allow registered end users to associate vehicles to their account and keep track of them. The Mobile Application component allows the End-User to insert all vehicle information that are stored by the DBMS component.
- [R7] The system must allow registered end users to associate payment methods to their account. This requirement is met in the same way as requirement R6.
- [R8] The system must allow registered end users to view the available time-slots for a certain type of socket of a certain charging station in a specific day. The Mobile Application component allows the end user to view the above information provided by the DBMS.
- [R9] The system must allow registered end users to book a charge in a specific charging station. The Mobile Application allows the user to visualize the charging stations map provided by the Map Manager and fill and confirm a charge booking through the Bookings Manager.
- [R10] The system must allow end users to unlock a charging socket if they have booked it and it's the correct time-slot. The Mobile Application component allows the user to visualize a personal booking page where it's possible to unlock the associated charging socket by the Bookings Manager that communicates with the related CPMS.
- [R11] The system must let a charge start if a vehicle is correctly connected and the charging socket is unlocked. This requirement is provided by the Bookings Manager component that directly communicates with the related CPMS.
- [R12] The system must show to end users the charging status (e.g. remaining time to complete charge). This requirement is met by the Mobile Application component and the Notification Manager component. The Mobile Application component provides this information to the End-User by the Notification Manager, which is directly connected to the Charging Stations Manager..
- [R13] The system must notify the end-user when the charge is complete. This requirement is provided in the same way as requirement 12
- [R14] When a charging process is correctly finished the system must charge on the end user selected payment system the correct import. This requirement is provided by the Bookings Manager component that is directly connected with the right Payment API.
- [R15] The system must allow CPOW to register a new charging station through its physical id. The Web application allows a CPOW to insert all information related to the registration which are checked and than saved by the Charging Station Manager that communicates with the DBMS.
- [R16] The system must be able to dynamically select if using batteries, directly the network or a mix of the two for each charging station.*

- [R17] The system must allow CPOWs to manually select if using batteries, directly the network or a mix of the two for each charging station associated to them. **
- [R18] The system must be able to dynamically select which DSO use to provide energy to a specific charging station.*
- [R19] The system must allow CPOWs to manually select which DSO use to provide energy to a specific charging station associated to them. This requirement is provided by the Web Application component and the DSO Manager component. The Web Application component allows the CPOW to select as input a DSO furniture and then the DSO Manager will communicate with the right DSO in order to execute the decision.
- [R20] The system must be able to dynamically decides if a certain energy furniture is destinated to a battery or directly to sockets of charging stations.*
- [R21] The system must allow CPOWs to manually decides if a certain energy furniture is destinated to a battery or directly to sockets of charging stations associated to them. **
- [R22] The system must keep track of the current status of each charging process (e.g. booked, startedCharging, etc.).*
- [R23] The system must keep track of the structure of each charging station (e.g. number of columns, number and type of sockets).*
- [R24] The system must allow CPOWs to view the structure of each charging station associated to them. **
- [R25] The system must keep track for each charging station of all bookings related to it.*
- [R26] The system must allow CPOWs to view all the bookings of a certain charging station associated to them and their status. **
- [R27] The system must allow CPOWs to set prices and special offers for a certain charging station associated to him. **

* R16, R18, R20, R22, R23, R25 requirements refers to the functions that can be done automatically by the Charging Stations Manager interacting with a CPMS.
** These requirement are provided by the Web Application component and the Charging Station Component. The Web Application component allows the CPOW to input the different decisions, while the Charging Station component executes them by communicating directly with the relevant CPMS. In the case of requirements 26 and 27, the DBMS is used to store the input information.

5 Implementation, integration and test plan

5.1 Implementation and unit testing

The implementation and test plan is defined starting from the general component diagram in Figure 2.1 and considering the level of importance of every component. All modules need to be tested individually using the white testing technique. Note that the external components (DSOAPI, MapsAPI, PaymentAPI , MotorizationAPI and CMPSAPI) are considered to be available from the beginning. The system is to be developed according to the following five subparts :

1. **Data**
   This subpart consists of the database and the *DBMS* that handles it. This subpart is the most important to implement since is the core of the system.

2. **First Managers**
   After the installation of the Data subpart follows the implementation of the Application Logic. First of all are created the "Manager" components that are designated to manage the structure on which the entire system is based, i.e. the *Access Manager*, the *Bookings Manager* and the *Charging Stations Manager*, that in fact handles respectively the Accounts, the Charge Bookings and the Charging Stations.

3. **Second Managers**
   This subpart consists of the remainder of the "Manager" components, i.e. the *DSOs Manager* , the *Notification Manager* and the *Map Manager*.

4. **Router**
   When all the previous components implementation is finished the Router component can be implemented.

5. **Client side**
   Finally the presentation layer is implemented. It is important to test the Mobile Application component on various types of mobile devices (smart-phones and tablets), with different versions of Android and iOS. The same holds for the Web Application that will need tests on various browsers (Google Chrome, Mozilla Firefox, Microsoft Edge), versions and operating systems.

In order to do the unit tests some components will need stubs :

- *Access Manager*, *Bookings Manager*, *Map Manager*, *DSOs Manager* and *Charging Stations Manager* will need a stub for the *DBMS*.
- *DSOs Manager* will need a stub for the *Charging Stations Manager*.

- *Maps Manager* will need a stub for the *Bookings Manager*.
- *Notification Manager* will need a stub for the *Mobile Application*.
- *Router* will need a stub for all the "Manager" components.
- *Mobile Application* will need a stub for the *eMSP*.
- *eMSP* and *Web Application* will need a stub for the *Router*.

5.2 Integration testing

Following the implementation order explained above, has been chosen a **bottom-up** approach as integration testing strategy. The choice of this testing technique is clearly based on the evident hierarchical structure of the system. Note that as a consequence it will be necessary to construct drivers for each module and when a driver will be replaced by module the integration tests need to be checked again. Has been also decided to integrate elements of a **critical-module-first** approach in order to give precedence to the fundamental elements of the system in order to conduct a more important testing on them and find system-breaking bugs as early as possible. The figures below helps to practically understand the integration order of the components.

1. **First Managers**
   The integration starts with the First Managers components subpart since the *DBMS* and databases are solutions that are commercially available and they only need to be configured to communicate properly with the rest of the system. At this step is possible to observe how Accounts, Charge Bookings and Charging Stations are generated, are stored and how interact with prevoius istances saved.

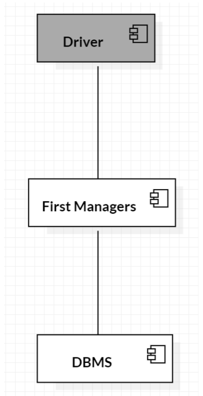


Figure 5.1: Integration First Managers

2. **Second Managers**
   Then Second Managers components can be integrated

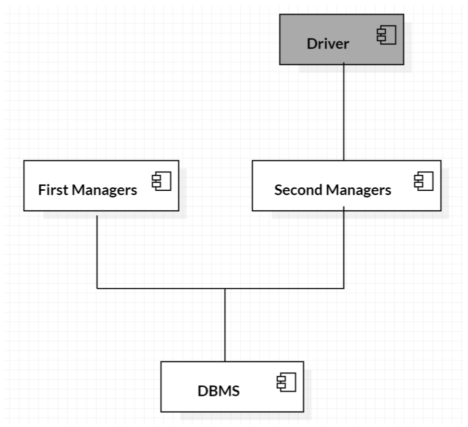


Figure 5.2: Integration Second Managers

3. **Router**
   After the Managers components are ready the *Router* can be integrated

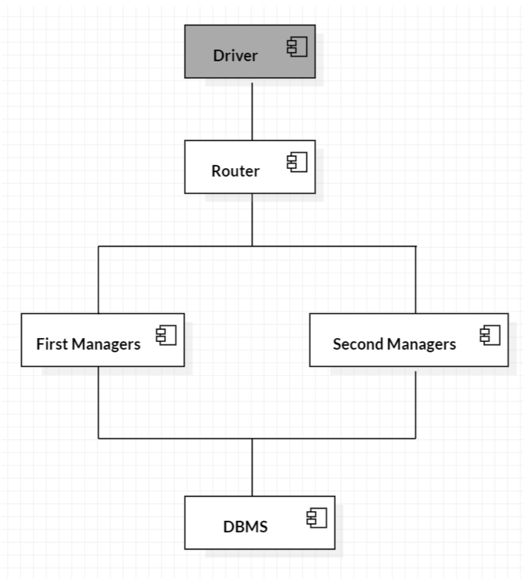


Figure 5.3: Integration Router

4. **Client**
   The integration concludes with the Presentation Layer

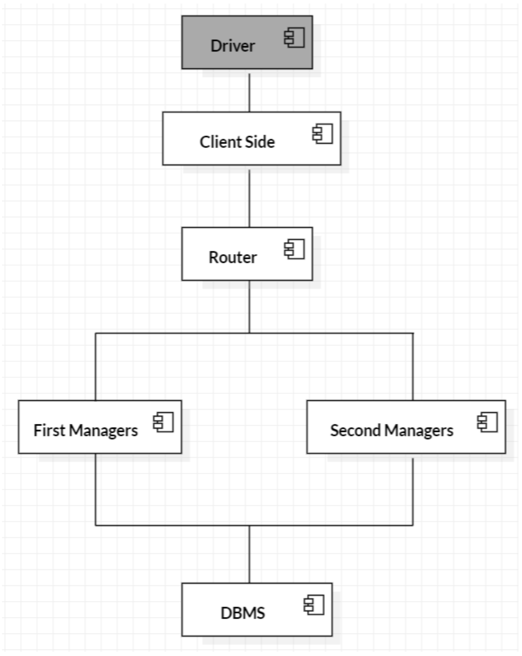


Figure 5.4: Integration Client

After the integration testing has been finished the system testing can be done with the purpose to test the functional and non-functional requirements. This will be mainly load type since the main job of the application is to store information such that personal accounts or bookings, thus is important to observe how it works when handling big amount of data. In the second instance also performance testing has to be done in order to understand how the system responds during a End-User vehicle charging process.

6 Effort spent

Enrico Brunetti

Date	*Hour*	*Section*
21-12-2022	3 *	Introduction and Architectural Design
22-12-2022	2	Architectural Design
23-12-2022	2	Architectural Design
24-12-2022	3 *	Architectural Design
27-12-2022	3	Architectural Design
28-12-2022	2	Architectural Design
29-12-2022	2	Architectural Design
30-12-2022	3	Requirements Traceability
02-01-2023	1.5 *	Architectural Design
03-01-2023	1.5 *	Architectural Design
04-01-2023	2.5 *	Architectural Design
05-01-2023	1.2	Architectural Design
06-01-2023	2	Implementation, integration and Test Plan
07-01-2023	5 *	Final Work

Table 6.1

Matteo Gionfriddo

Date	*Hour*	*Section*
21-12-2022	3 *	Introduction and Architectural Design
22-12-2022	2.5	Architectural Design
23-12-2022	3	Architectural Design
24-12-2022	3 *	Architectural Design
26-12-2022	2	User Interface Design
27-12-2022	1.5	User Interface Design
28-12-2022	2	Requirements Traceability
03-01-2023	1.5 *	Architectural Design
04-01-2023	2.5 *	Architectural Design
05-01-2023	3	Implementation, integration and Test Plan
06-01-2023	2	Implementation, integration and Test Plan
07-01-2023	5 *	Final Work

Table 6.2

* *Group work*

7 References

- Specification Document: “Assignment RDD AY 2022-2023_v3.pdf”
- RASD Document: “RASD1.pdf"
- Google Maps API: https://developers.google.com/maps
- Motorization API: https://www.mit.gov.it/documentazione/accesso-al-sistema-informativo-motorizzazione-civile
- Payment API: https://stripe.com/docs/api