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, -/12/2022
Version 1.0

Contents

1 Introduction **3**
1.1 Purpose . . . 3
1.2 Scope . . . 3
1.3 Definitions,Acronyms,Abbreviations . . . 3
1.4 Revision history . . . 4
1.5 Reference Documents . . . 4
1.6 Document Structure(TO DO) . . . 4

2 Architectural Design(TO BE BETTER EDITED FROM THE ORIGINAL DOCUMENT) **5**
2.1 Overview: . . . 5
2.2 Component view . . . 5
2.3 Deployment view . . . 9
2.4 Runtime view . . . 9
2.5 Component Interfaces . . . 22
2.6 Selected architectural styles and patterns . . . 22
2.6.1 Model-View-Controller Pattern (MVC) . . . 22
2.6.2 Tiny client . . . 23
2.6.3 RESTful architecture . . . 23
2.6.4 Database . . . 24
2.7 Other design decisions . . . 24

3 User interface design **25**

4 Requirements Traceability **31**

5 Implementation, integration and test plan **34**
5.1 Implementation and unit testing . . . 34
5.2 Integration testing . . . 35

6 Effort spent **37**

List of Figures

2.1 UML Component Diagram 6
2.2 UML Component Diagram for *Router* component 6
2.3 UML Component Diagram for *BookingsManager* component 7
2.4 UML Component Diagram for *AccessManager* component 8
2.5 UML Component Diagram for *ChargingStationsManager* component 8
2.6 UML Deployment Diagram 10
2.7 UML Sequence Diagram for the End-User Mobile App registration 14
2.8 UML Sequence Diagram for the End-User Mobile App login 15
2.9 UML Sequence Diagram for Mobile App map load 16
2.10 UML Sequence Diagram for End-User charge booking 17
2.11 UML Sequence Diagram for End-User charging process 18
2.12 UML Sequence Diagram for CPOW selecting a DSO for a charging station . . . 19
2.13 UML Sequence Diagram for CPOW selecting batteries usage for a charging station 20
2.14 UML Sequence Diagram for Web Application showing to a CPOW date about a charging station 21
2.15 UML Class Diagram 22

3.1 Mobile Application Mockups for End-User Log in, End-User Registration and Vehicle Charging State Popup Screens 25
3.2 Mobile Application Mockups for Home Page, Charging Stations Map and Personal Home Page Screens 26
3.3 Web Application Mockup for CPOW Registration Screen 26
3.4 Web Application Mockup for DSO Offers Screen 27
3.5 Web Application Mockup for Selected Charging Station Screen 28
3.6 UML Activity Diagram for the End-User Mobile App navigation 29
3.7 UML Activity Diagram for the CPO Worker Web App navigation 30

5.1 Integration DBMS 35
5.2 Integration Back-End 35
5.3 Integration Router 36
5.4 Integration Client 36

List of Tables

4.1 Table for mapping requirements to components 31

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

- |-12-2022 Version 1.0.

1.5 Reference Documents

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

1.6 Document Structure(TO DO)

2 Architectural Design(TO BE BETTER EDITED FROM THE ORIGINAL DOCUMENT)

2.1 Overview:

The application will be developed using the client-server paradigm on a three-tiered architecture. The three layers of the application (Presentation, Application and Data) are divided into clusters of machines (i.e. tiers) that actually cooperate to provide a specific functionality. In this case we have three tiers and each tier is responsible for one of the three layers. The client tier is responsible for the **Presentation** layer; therefore, in this architecture, are included eMPS and CPMS applications. The UIs provided are just meant to show results and to allow clients to choose what they want. In particular there are two types of clients: a **mobile app** for the End-Users which use it to book and take advantage of electric vehicle charges, directly connected to the associated eMSP and a **web app** for CPOWs for managing associated charging stations, directly connected to the related CPMS.
The Application tier takes care of the application layer encapsulating all is needed concerning the application logic. It receives the requests from the clients and handles them. It's also responsible for sending asynchronous notifications to a CPMS (which can be forwarded to an End-User trough eMSP APIs) in the presentation layer when certain conditions are met.
The **Application** tier communicates with the **Data** tier, responsible for the Data Access layer, that is the layer responsible for accessing the DataBases and performing queries on them.

2.2 Component view

In the figure 2.1 the *Component Diagram* for the *eMall* system is reported. This is a high level view in which only the main components are immediately shown, while some components will be better detailed afterwards.

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 has the role of dispatching the requests coming from the users applications. Before doing this it has also the important role of verifying the user authentication. In figure 2.2 a more detailed view is provided, putting in evidence the various interfaces for the eMSP and CPMS side.
- **BookingsManager**: this component is in charge of all concern the management of the

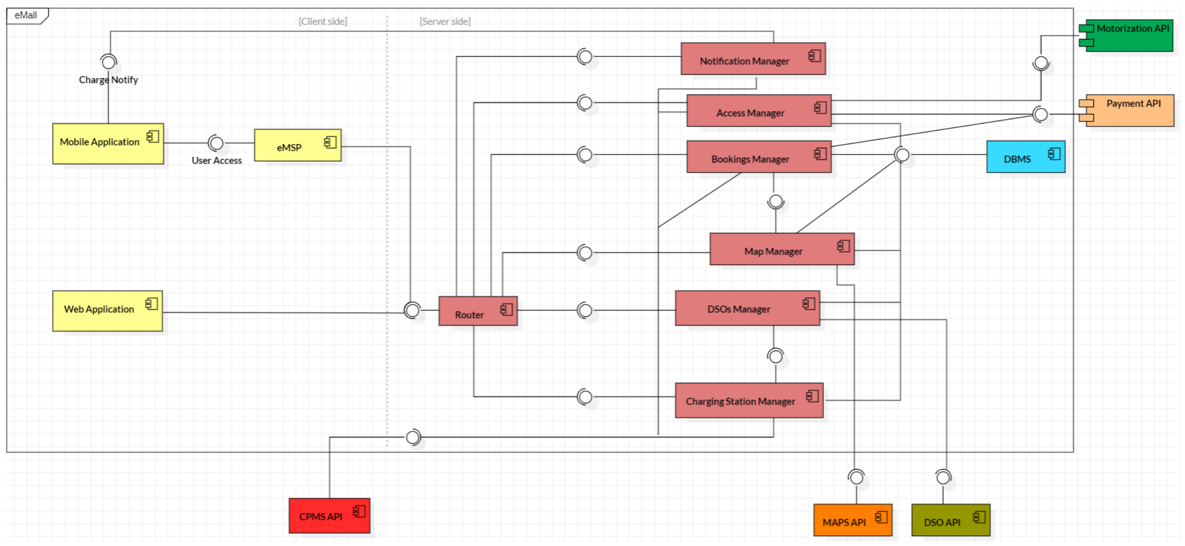


Figure 2.1: UML Component Diagram

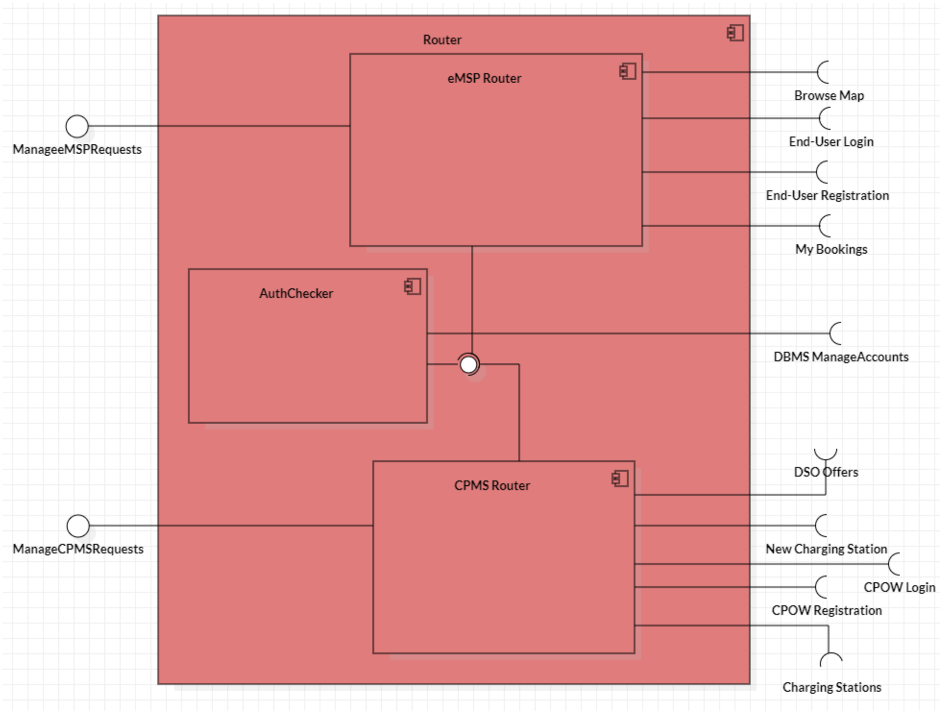


Figure 2.2: UML Component Diagram for *Router* component

bookings associated to the End-Users. It permits to make a new booking or retrie all personal bookings from the database trough the *DBMS* component and proceed from one of them with the charging process (trough an interface directly connected to an CPMS). This component is described in figure 2.3 .

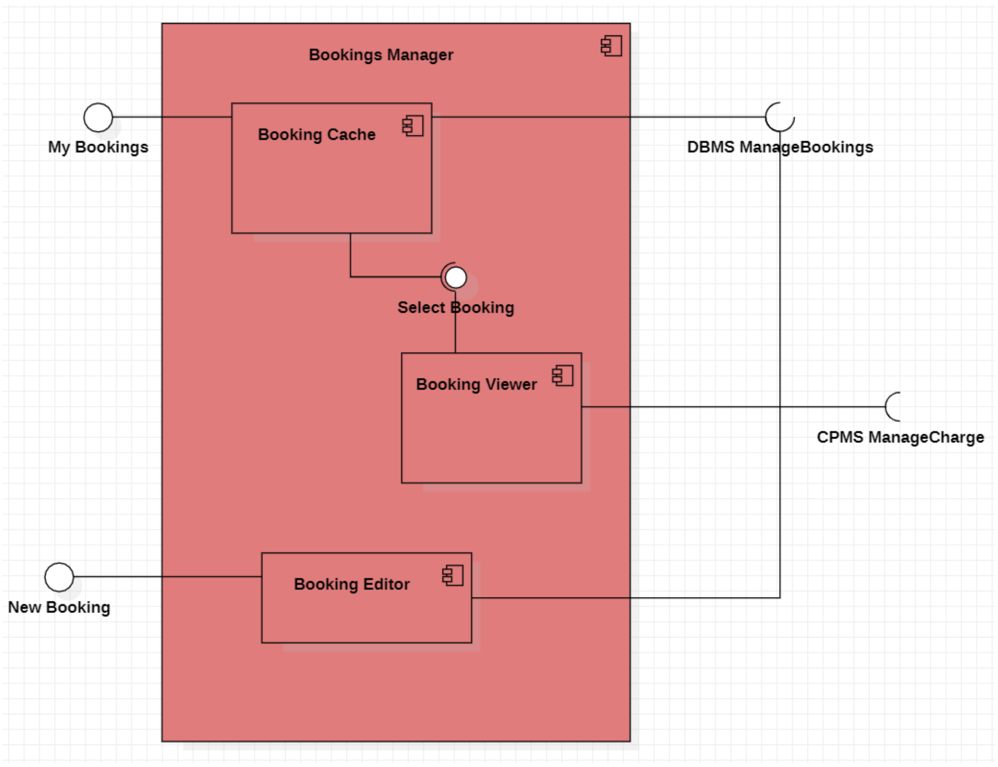


Figure 2.3: UML Component Diagram for *BookingsManager* component

- **AccessManager**: it manages everything about registration and login of the users. He stores the accounts data through the *DBMS* component and calls it also to verify the correctness of the login credentials. This component is detailed in figure 2.4.
- **ChargingStationsManager**: 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 and the associated CPMS. It is shown in the diagram in figure 2.5.
- **NotificationManager**: TO DO.
- **DBMS**: this component manages the operational database.
- **DataWarehouse**: TO DO .
- **DSOAPI**: this is an external component, managed by the DSO (an instance for each of them which supports the system), whose interface is standardized as described in the *Components Interfaces* section. It is necessary in order to retrieve prices and offers of energy furniture provided by the different companies.

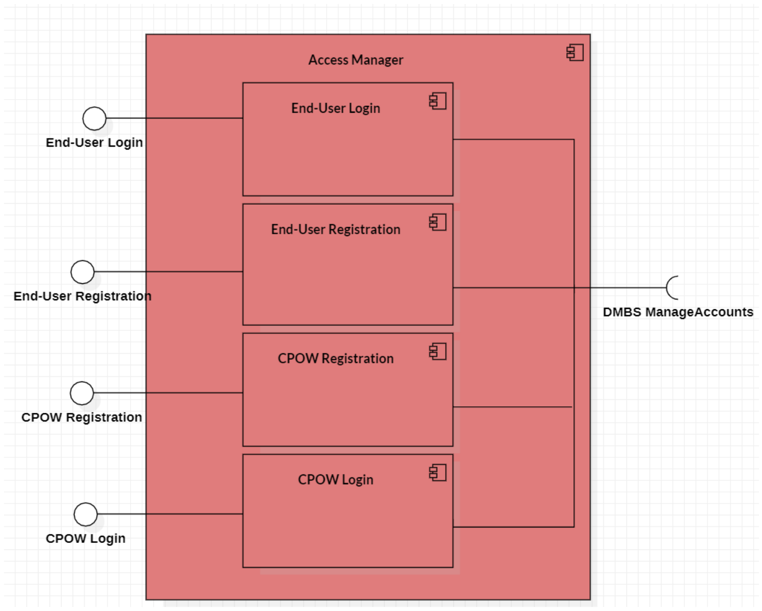


Figure 2.4: UML Component Diagram for *AccessManager* component

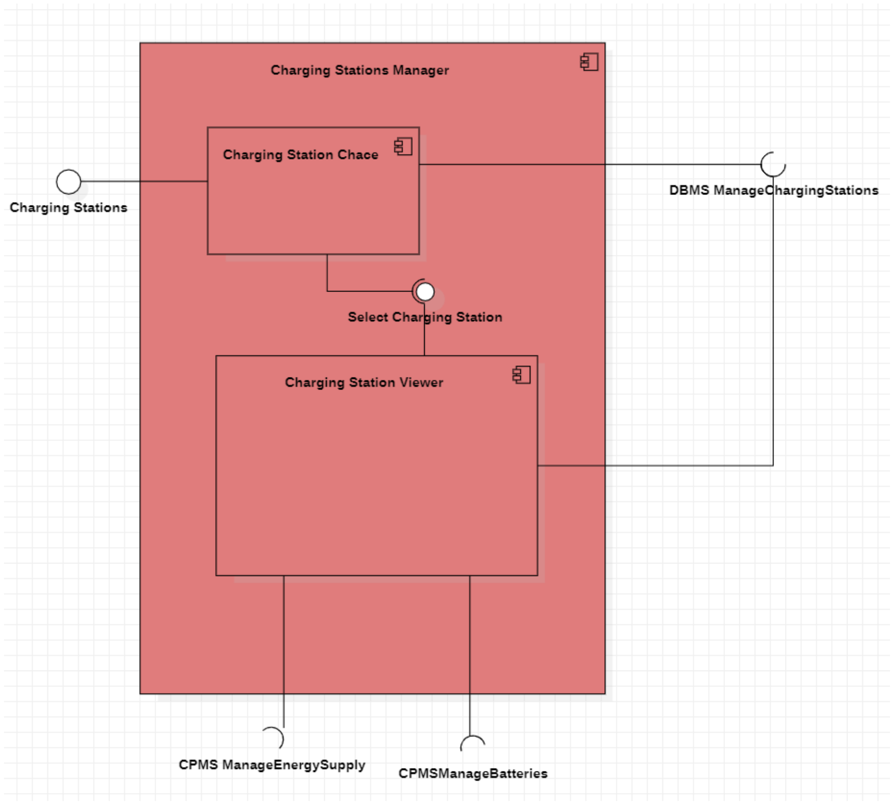


Figure 2.5: UML Component Diagram for *ChargingStationsManager* component

- **MapsAPI**: this is an external component which is exploited to obtain maps, which will be shown when browsing charging stations map.
- **CPMSAPI**: TO DO.
- **eMSP**: TO DO.
- **MobileApplication**: this component entirely resides in the *Presentation layer* and it's just an interface to allow the users to take advantage of the services offered by eMSP on their mobile devices, such that view nearby charging stations offers and book and use electric vehicle charges.
- **WebApplication**: 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.

2.3 Deployment view

So far, only an abstract and general view of the system has been provided. In this section a better detailed view of the system shows how the component previously described are actually deployed in different machines and how each tier is organized. 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.

W.r.t to the component diagram (figure 2.1), the different colors indicate the allocation of components. For the presentation tier clearly the *smartphones* contain the *MobileApplication* while the *personal computers* and the *web server* contain the *Web Application.*
All application logic components (orange in the *Component Diagram*) are deployed on the *Application Server*, while the *DBMS* component is deployed on the *DataBase server* and the *DataWarehouse* component is deployed on the *DataWarehouse server.*

In the *Mobile Application* case the client contains all the *presentation layer* while in the *Web Application* case the layer is splitted between the *Web App* and the *WebServer*; the *WebServer* is responsible for contacting the application server and forward the client requests and responses.

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, a *register* 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

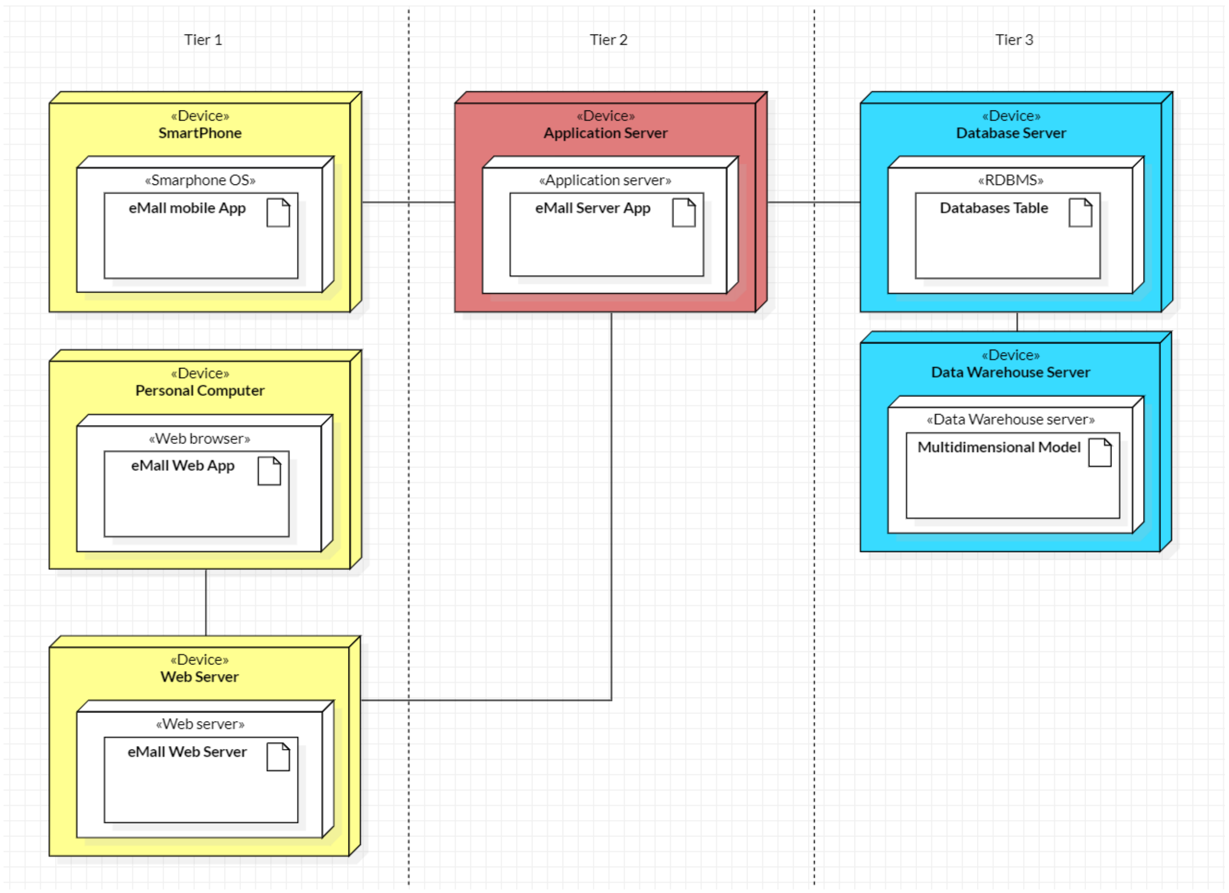


Figure 2.6: UML Deployment Diagram

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, through the **Router**, an *abilitateCharge* message to the **Charging Station Manager** 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_ can_ start* message to the mobile application. At this point, the user approaches his mobile phone to the RFID reader of the charging socket and the **Charging Station API** sends a get enabling message to the **Charging Station Manager** for the specific charge *id* obtained through RFID. If the *id* is correct the charge starts and the **Charging Station API** sends periodically to the **Charging Station Manager** the current status of recharge, which will be reported, firstly to the **DBMS** and secondly, through the **Router**, as a *current_ status* messages to the **Mobile Application** that displays charging status to the End-User. When the charge is completed, the **Charging Station API** communicates it to the **Charging Station Manager** that will save it to the **DBMS** and then ask, always through the **Router**, 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.

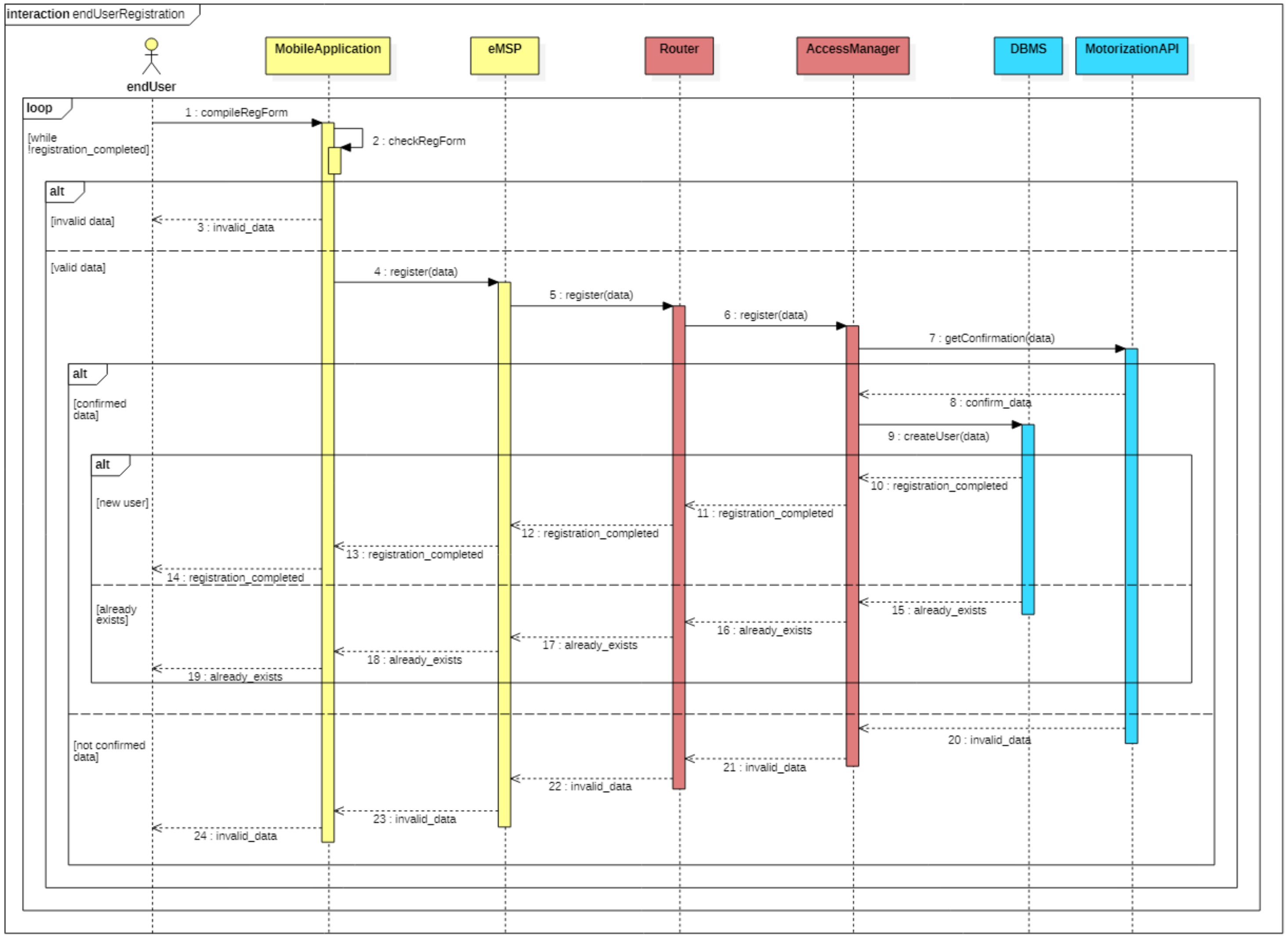


Figure 2.7: UML Sequence Diagram for the End-User Mobile App registration

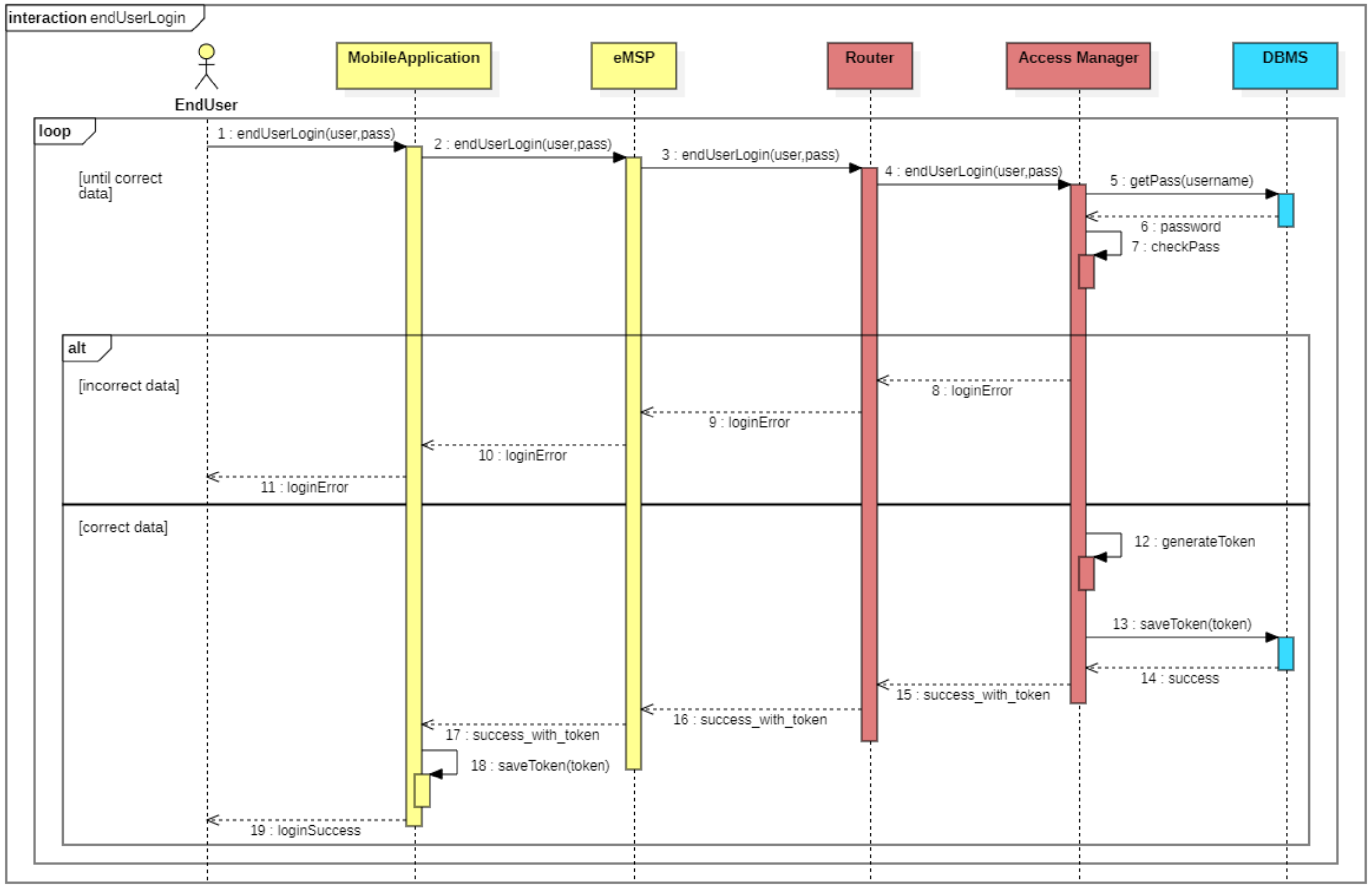


Figure 2.8: UML Sequence Diagram for the End-User Mobile App login

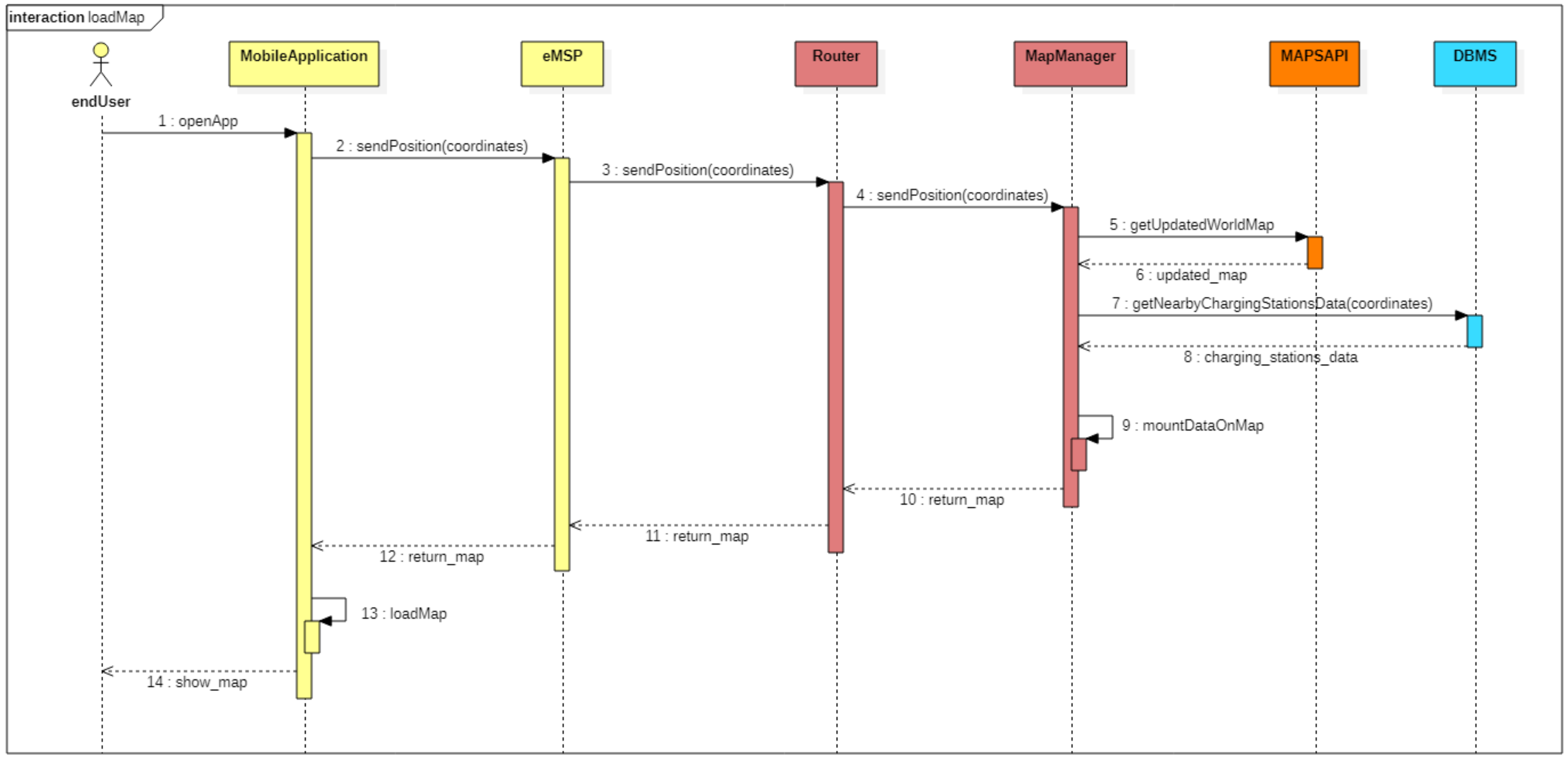


Figure 2.9: UML Sequence Diagram for Mobile App map load

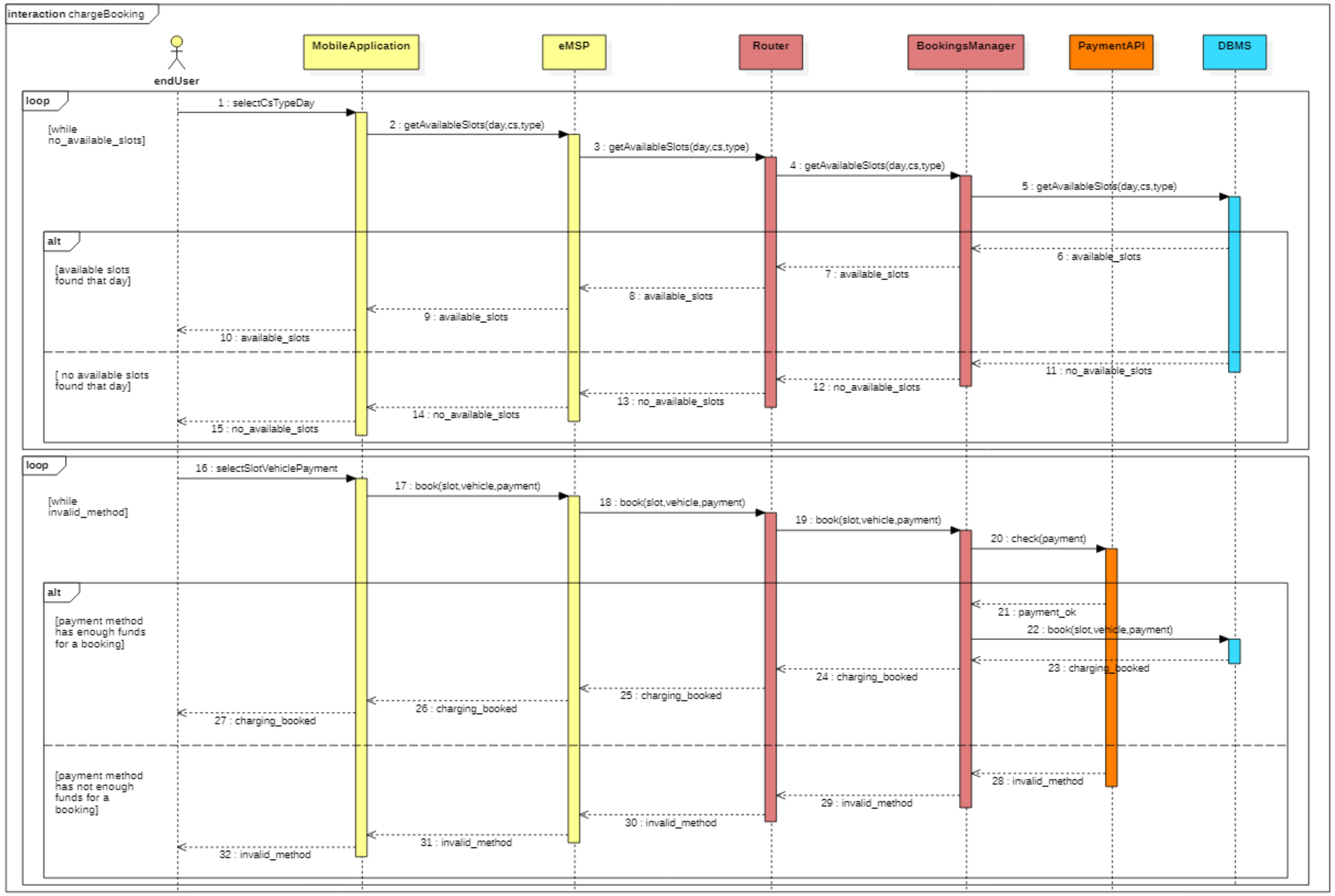


Figure 2.10: UML Sequence Diagram for End-User charge booking

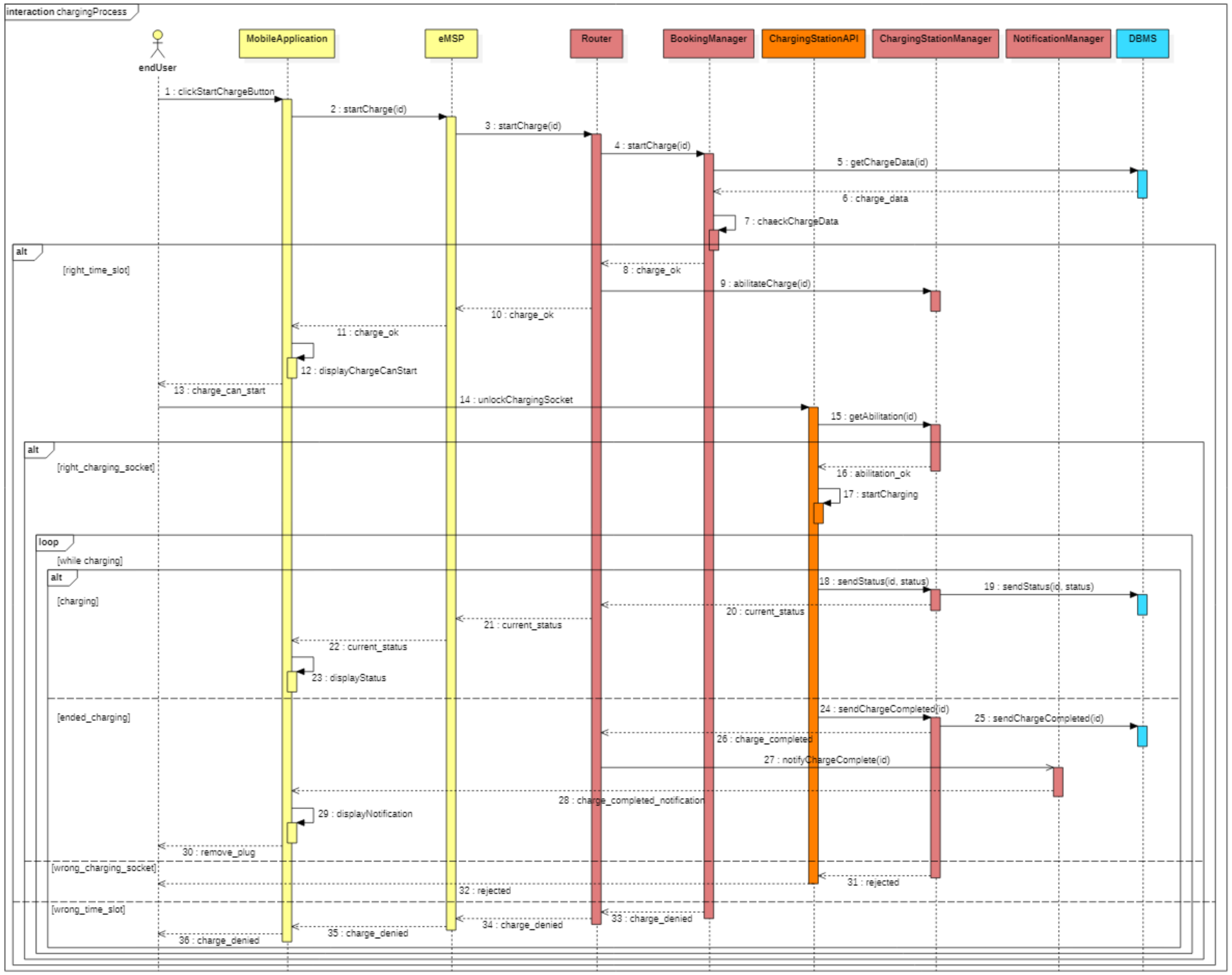


Figure 2.11: UML Sequence Diagram for End-User charging process

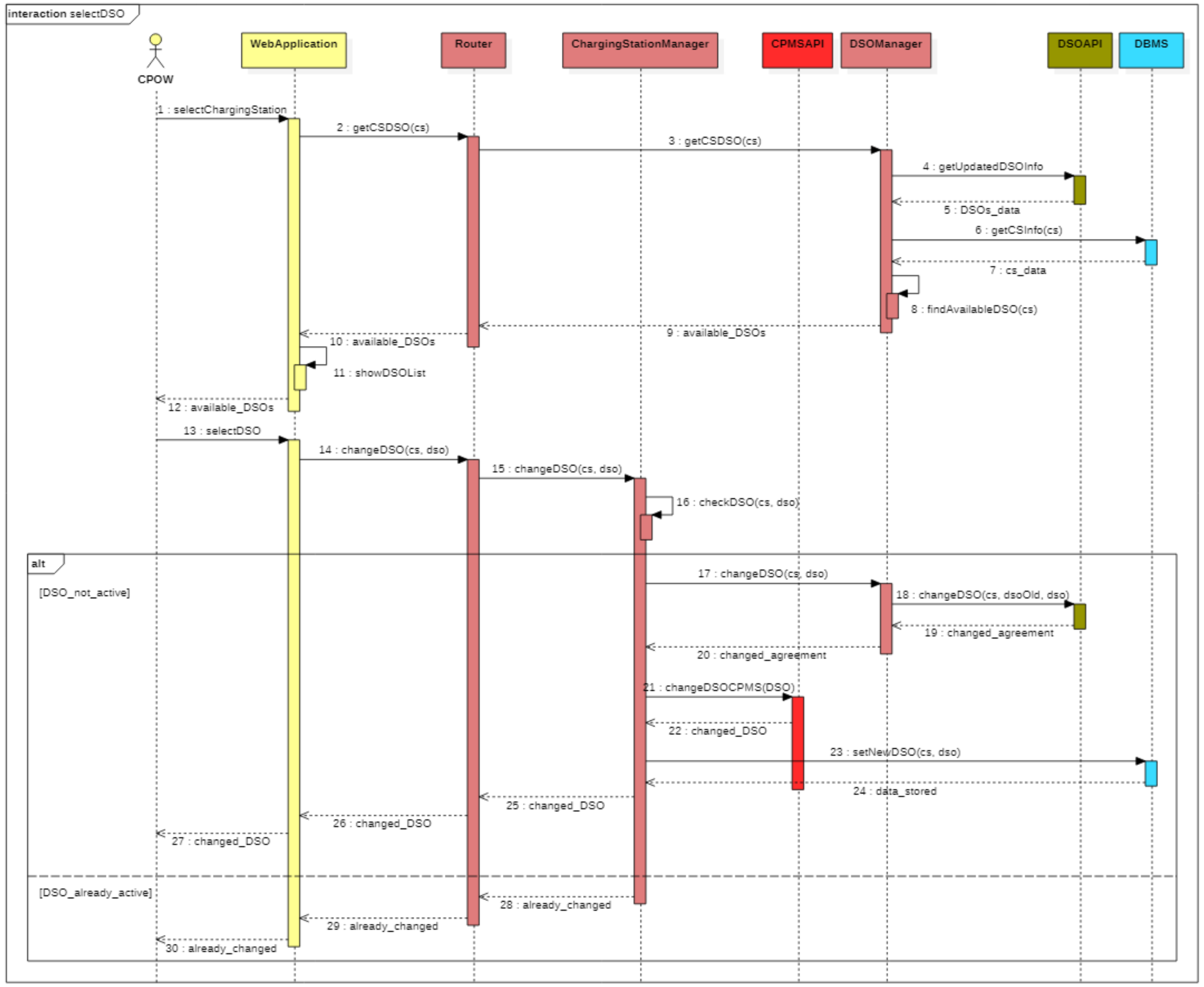


Figure 2.12: UML Sequence Diagram for CPOW selecting a DSO for a charging station

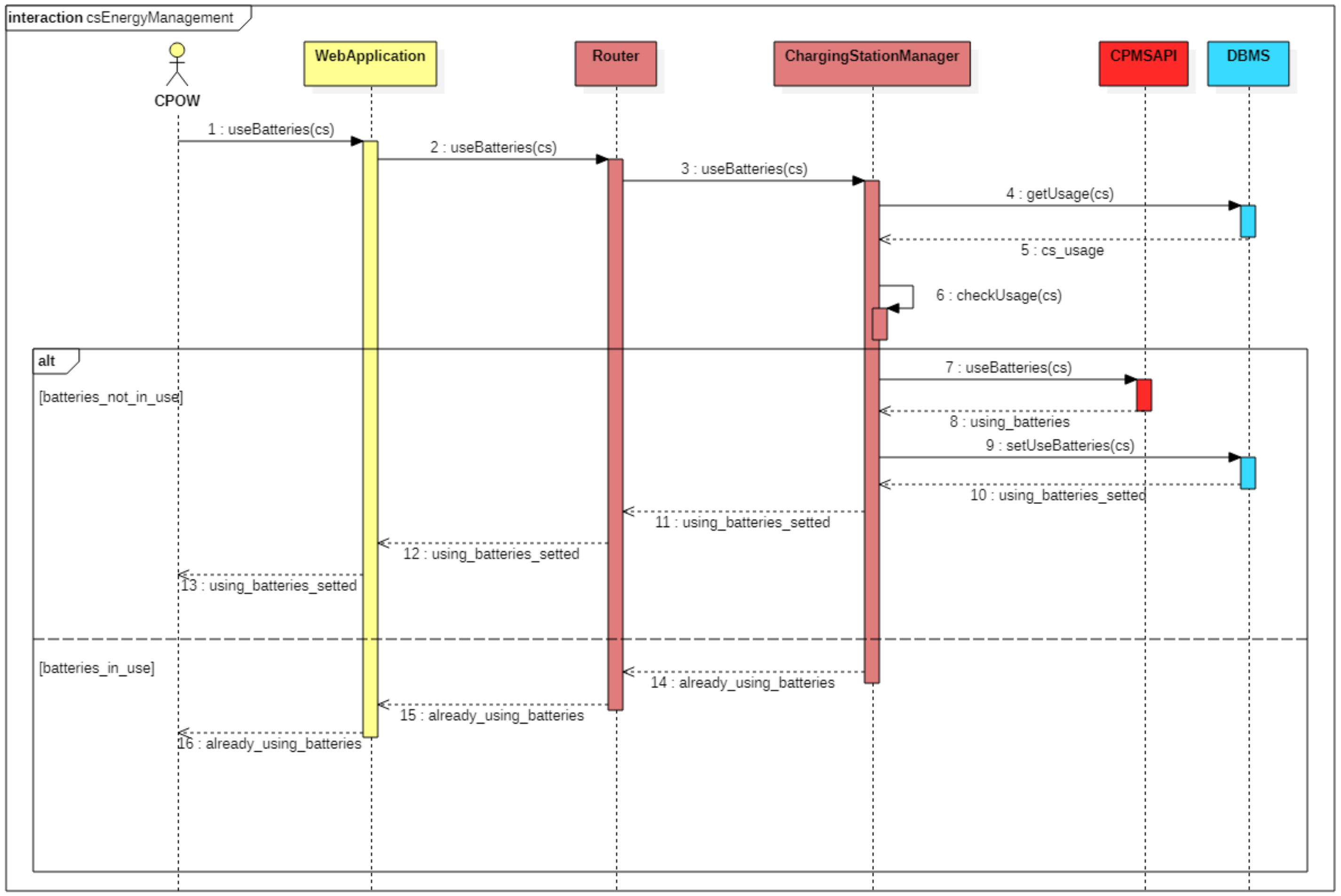


Figure 2.13: UML Sequence Diagram for CPOW selecting batteries usage for a charging station

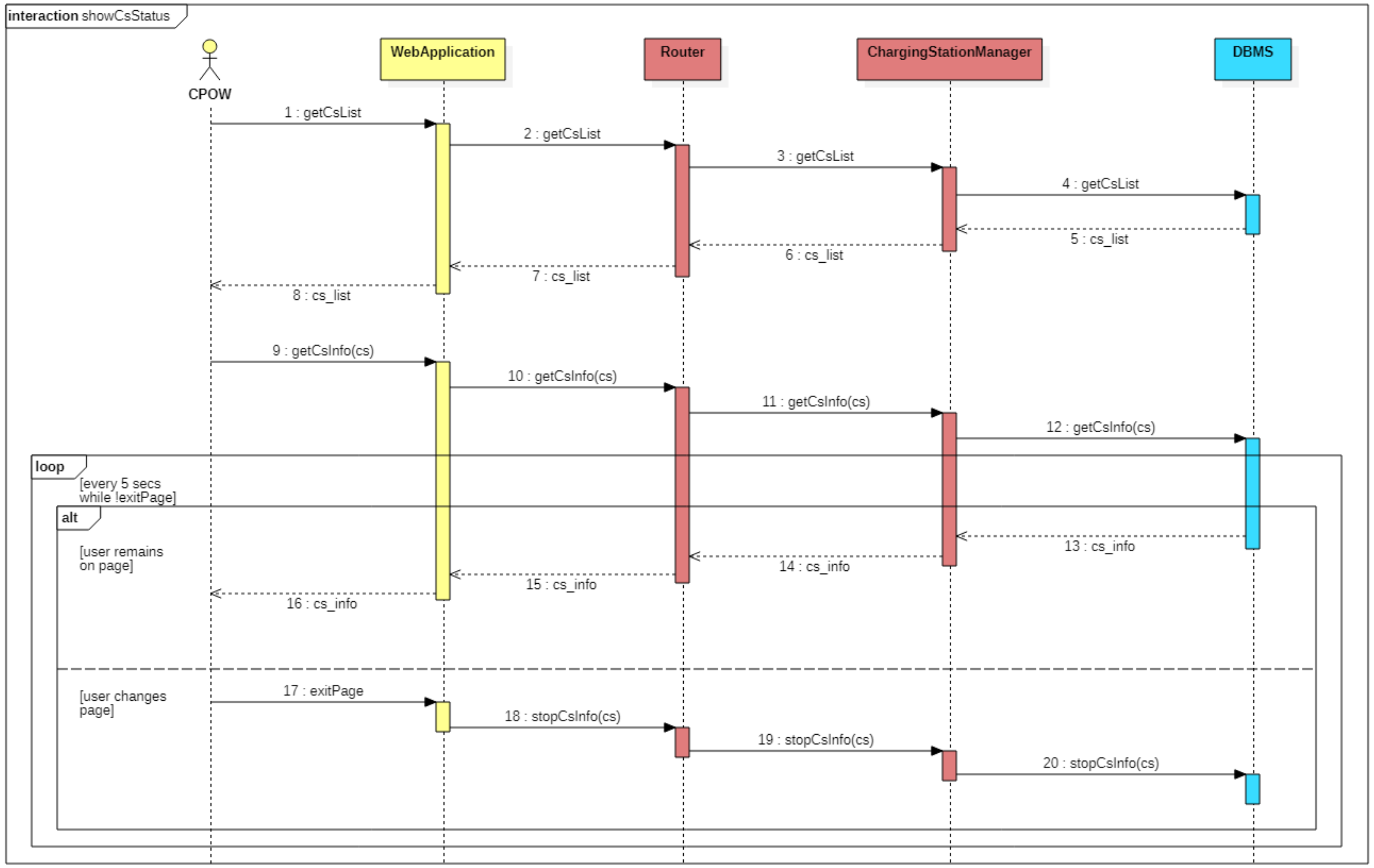


Figure 2.14: UML Sequence Diagram for Web Application showing to a CPOW date about a charging station

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.

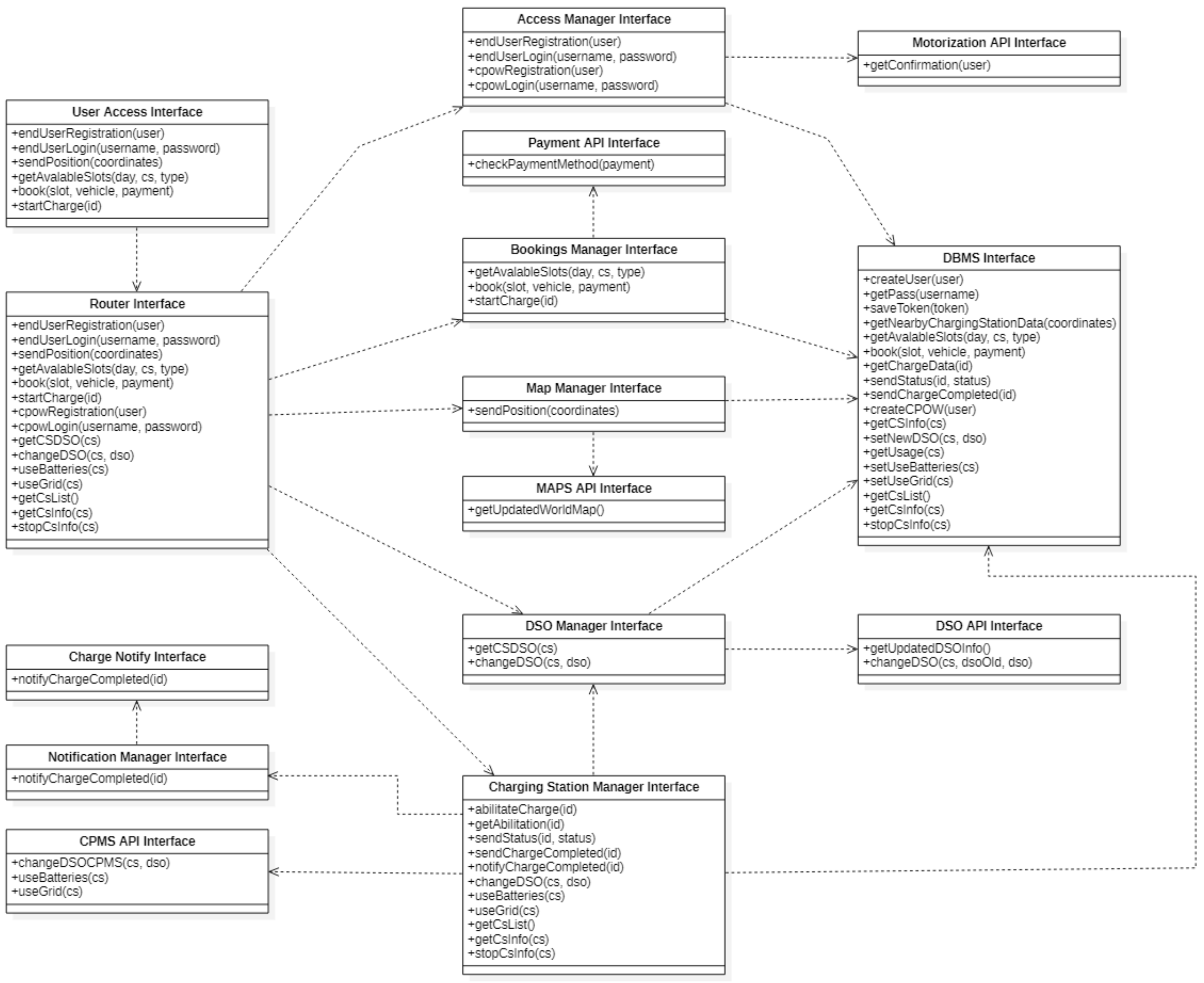


Figure 2.15: UML Class Diagram

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.

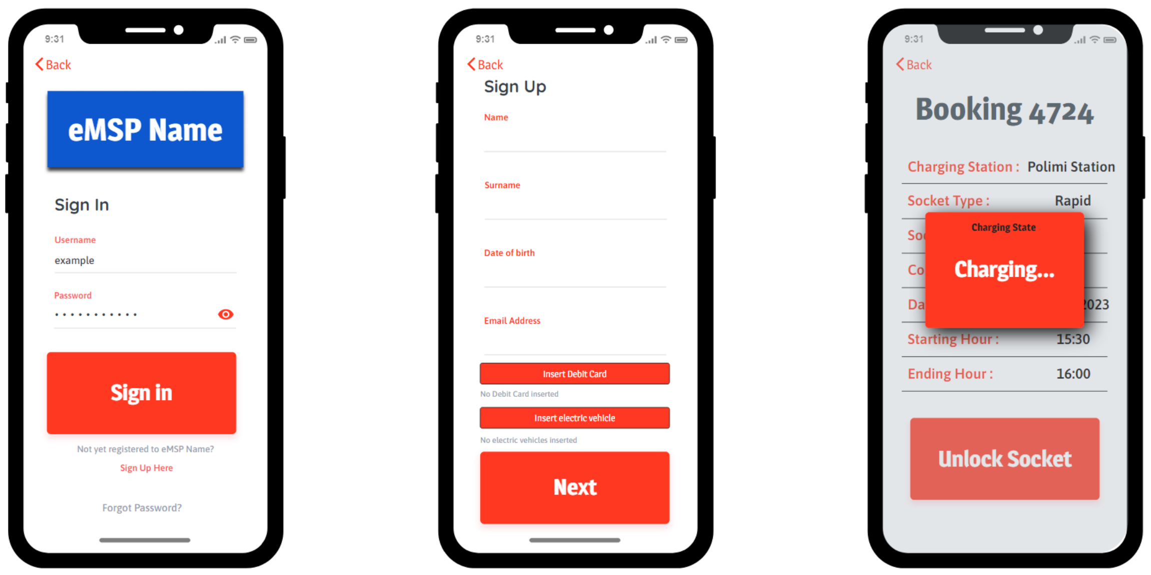
2.6.4 Database

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

3 User interface design

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



Username: example
Password: ···········

Name
Surname
Date of birth
Email Address
No Debit Card inserted
No electric vehicles inserted

Booking 4724
Charging Station : Polimi Station
Socket Type : Rapid
So
Co
Da 023
Starting Hour : 15:30
Ending Hour : 16:00

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.

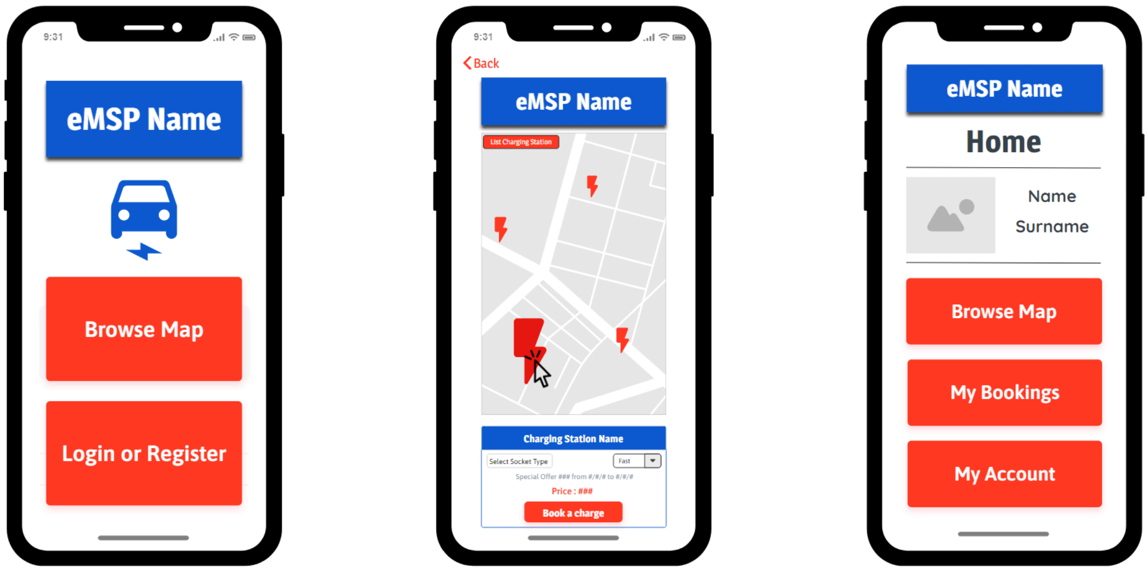


Figure 3.2: Mobile Application Mockups for Home Page, Charging Stations Map and Personal Home Page Screens

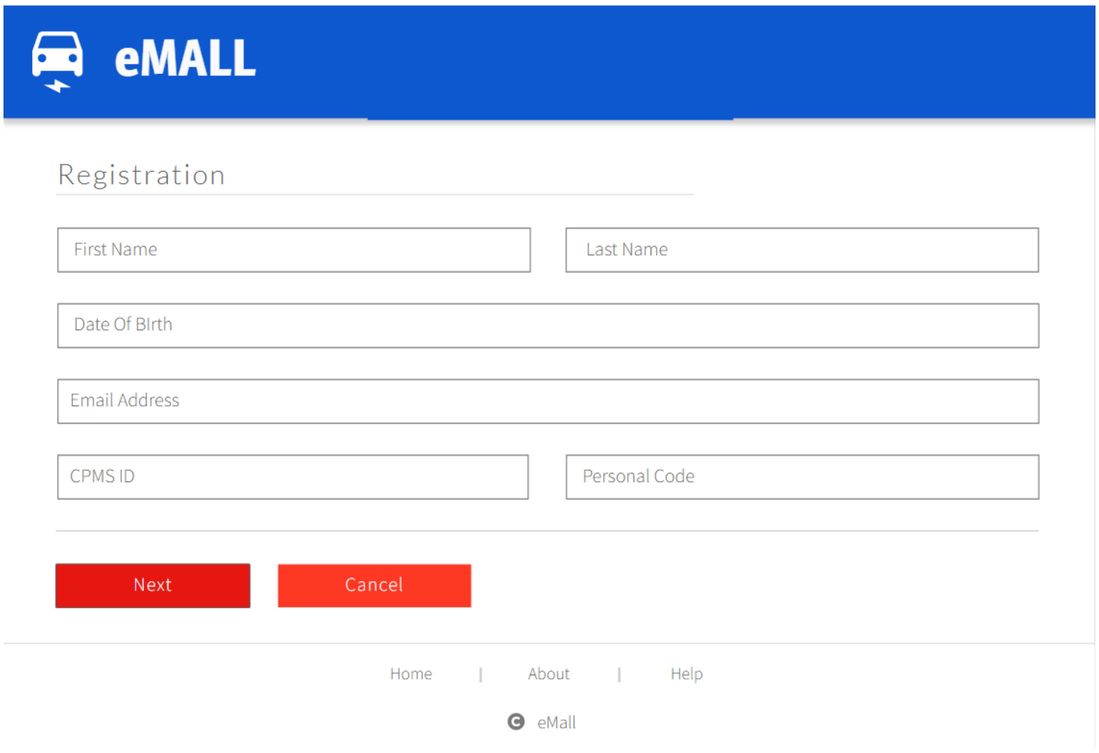


Figure 3.3: Web Application Mockup for CPOW Registration Screen

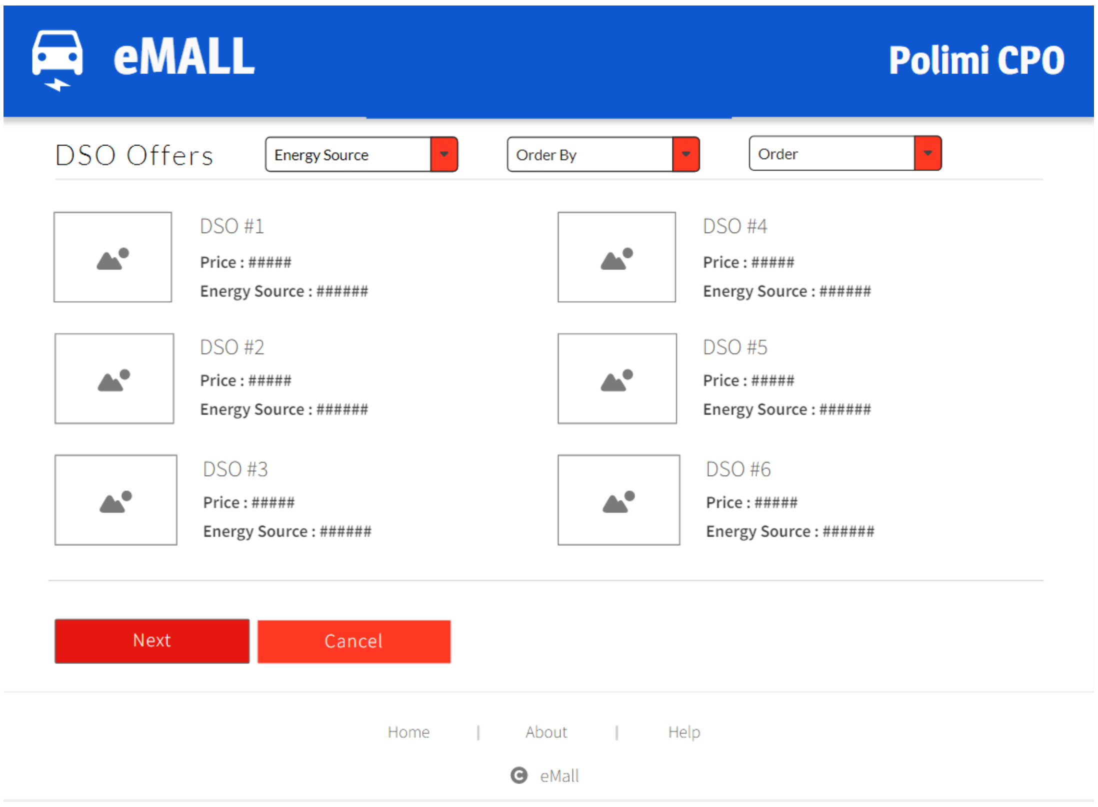
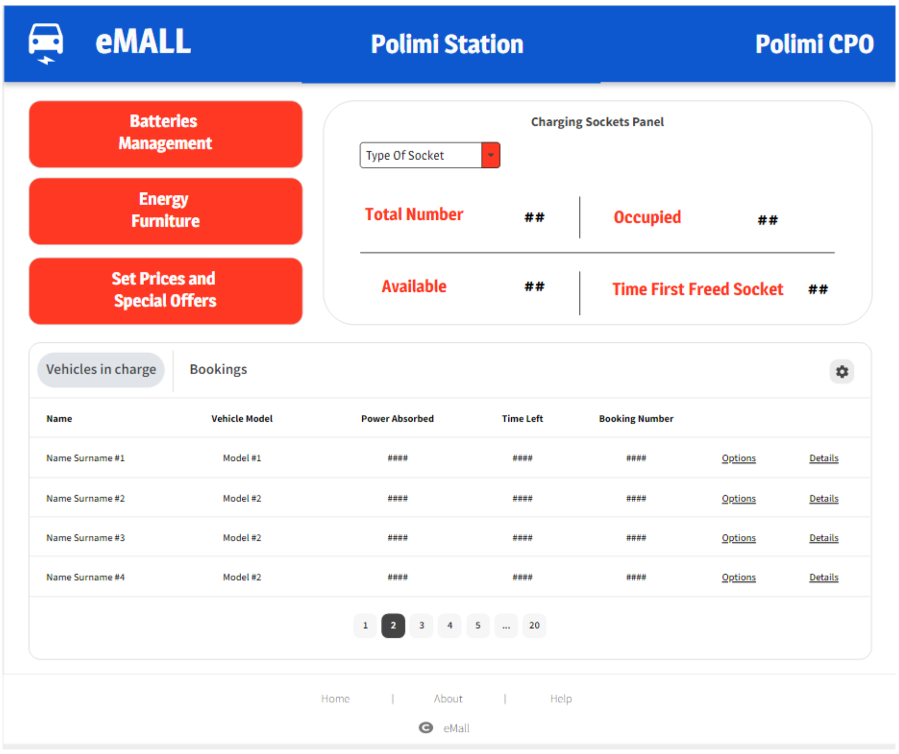


Figure 3.4: Web Application Mockup for DSO Offers Screen



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

1 2 3 4 5 ... 20

Figure 3.5: Web Application Mockup for Selected Charging Station Screen

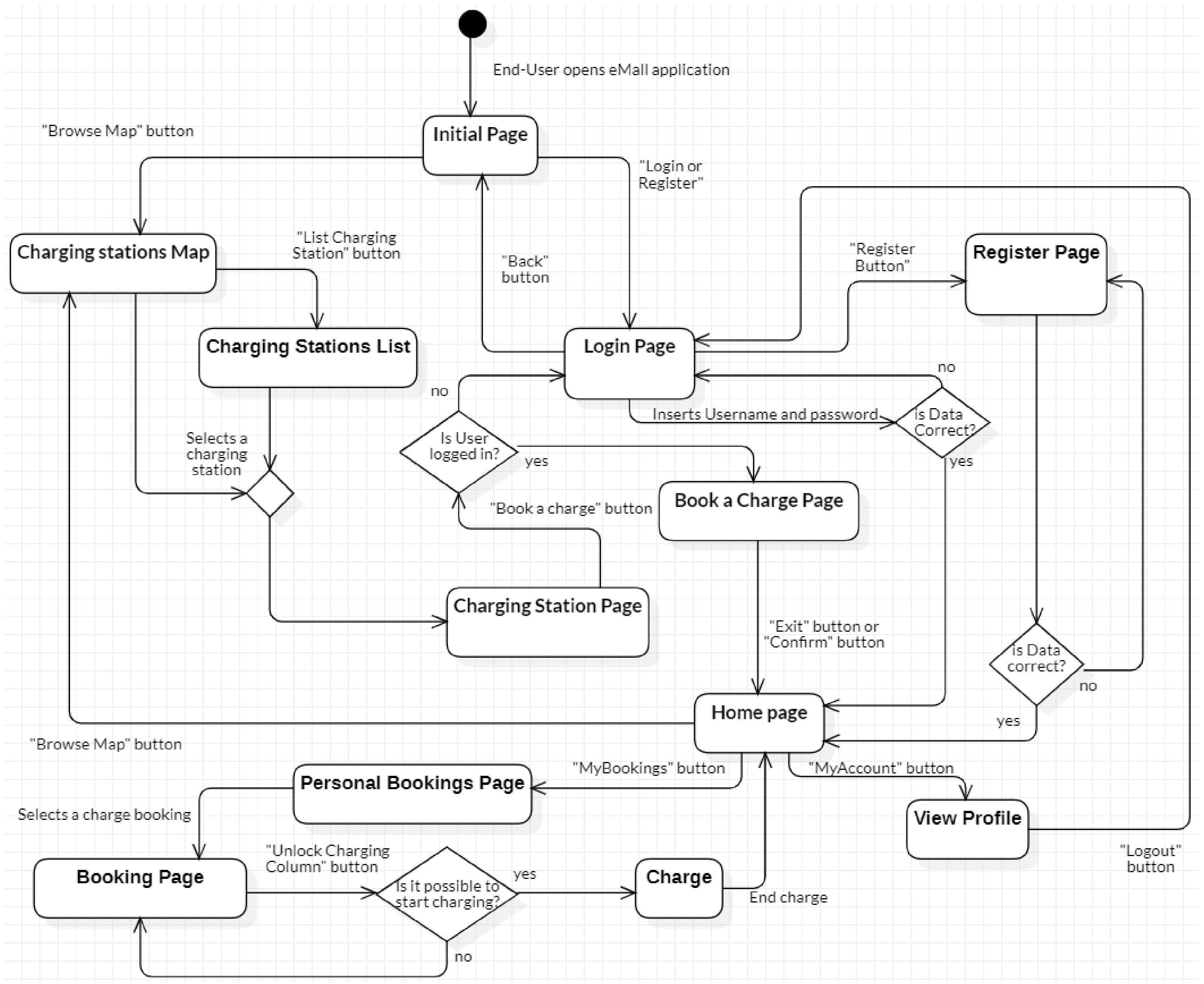


Figure 3.6: UML Activity Diagram for the End-User Mobile App navigation

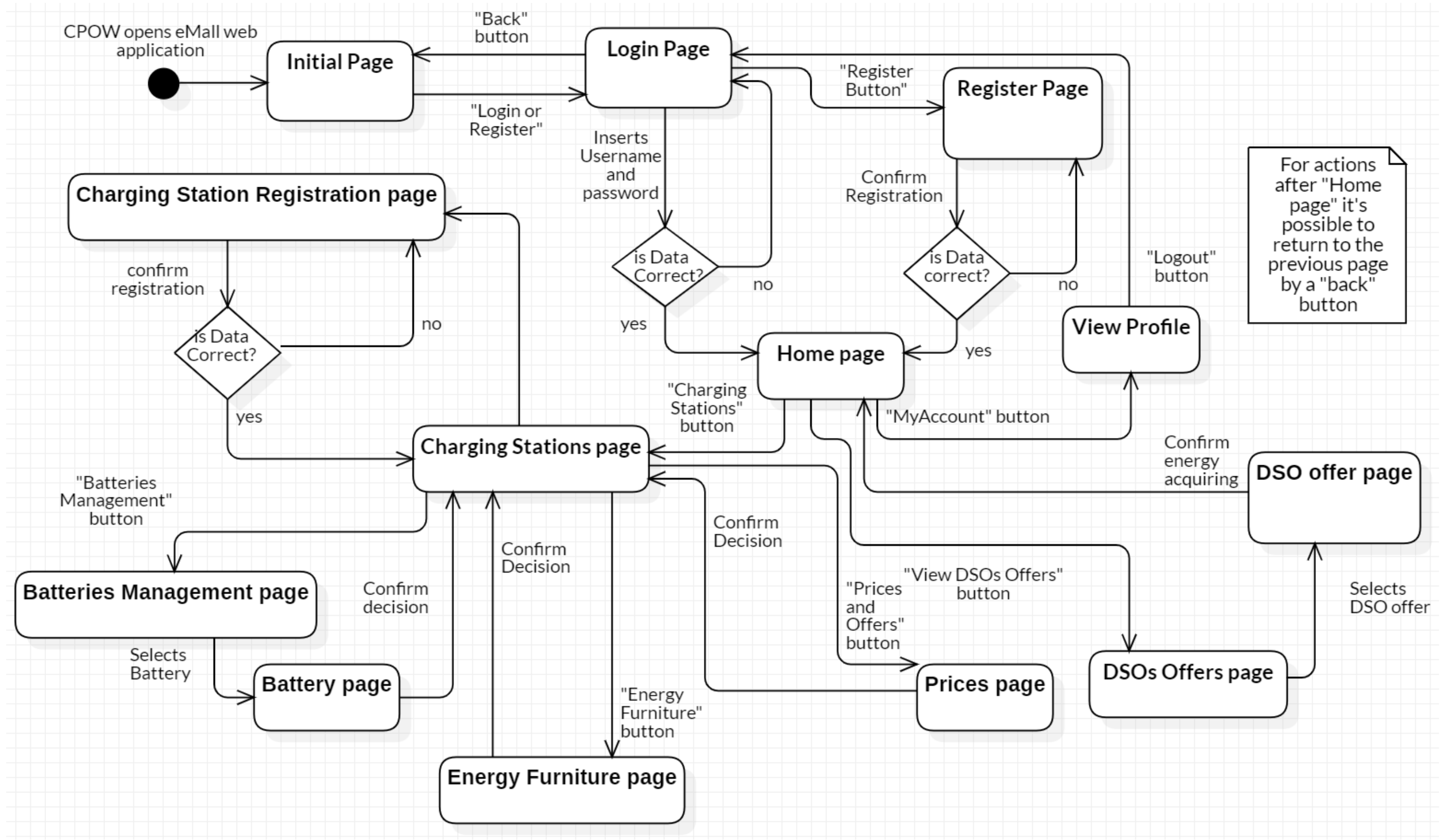


Figure 3.7: UML Activity Diagram for the CPO Worker Web App navigation

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 togheter doing the same acrivities, 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, 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 the End-User with the information sent by the Notification Manager, which is directly connected to the related CPMS.
- [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 done automatically by the 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. **CPO Managers***
   This subpart consists of the components that relays on the CPO Worker experience (DSOs and Charging Station Manager). Charging Station Manager component will be implemented as first. In order to do the testing operation both components will need a stub of the DBMS.

3. **End-User Managers***
   This subpart consists of the components that relays on the End-User experience (Notification, Access, Bookings and Map Manager). The Bookings Manager component will be implemented as first. In order to do the testing operation, every component will need a stub of the DBMS, while the Notification Manager component will need also a stub of the Mobile Application component.

4. **Router**
   When all the previous components implementation is finished the Router component can be implemented. Note that the Router component need a stub of all Manager components in order to do the testing operation

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.

* Sub parts 2 and 3 constitute the Back-End of the system and can be implemented in parallel since they are independent of each other.

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. **DBMS**

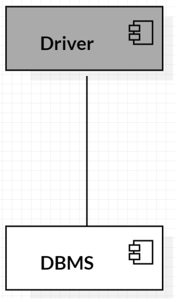


Figure 5.1: Integration DBMS

2. **Back-End**

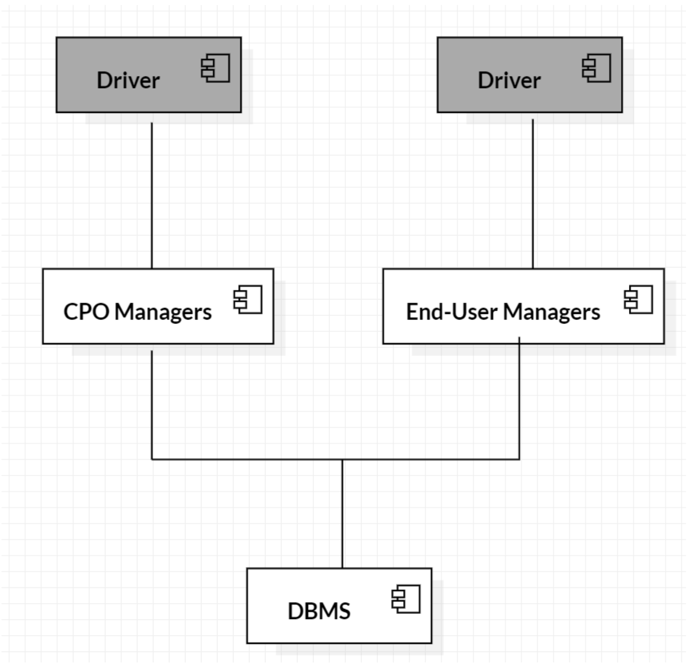


Figure 5.2: Integration Back-End

3. **Router**

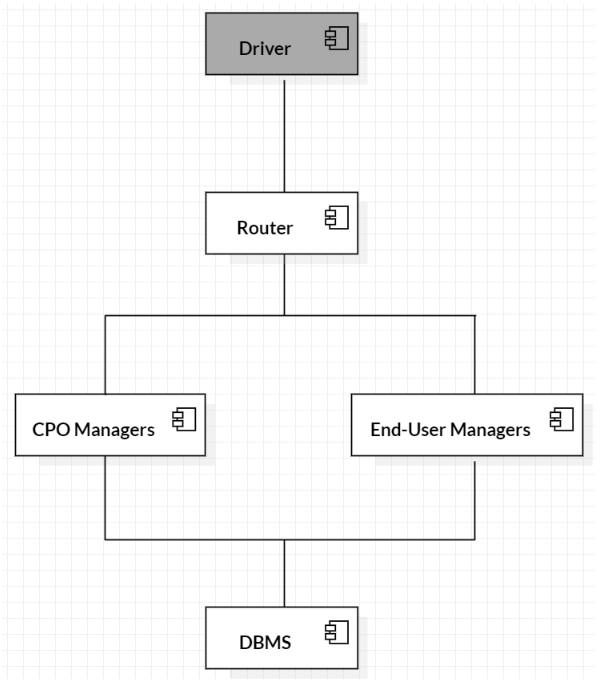


Figure 5.3: Integration Router

4. **Client**

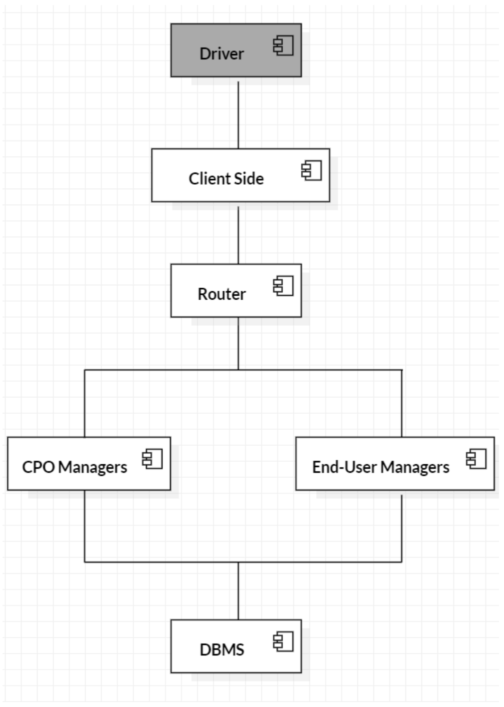


Figure 5.4: Integration Client

6 Effort spent