POLITECNICO DI MILANO

Computer Science and Engineering

Design Document

eMall - e-Mobility for All

Software Engineering 2 Project

Academic year 2022 - 2023

5 February 2023

Version 2.0

Authors:
Balestrieri Niccolò
Bertogalli Andrea
Tombini Nicolò

Professor:
Matteo Camilli

Version history

Date	Revision	Notes
29/12/2022	v.1.0	First release.
06/01/2023	v.1.1	Final release.
05/02/2023	v.2.0	Fixing typos and editing database schema

Contents

Chapter 1

Introduction

1.1 Purpose

The goal of **D**esign **D**ocument lies in a meticulous explanation of the infrastructure that the eMall system uses: reference will be made to a high-level description of the technologies and components used paying attention also to the interactive behavior of system users.
The audience of the presented document includes certainly teams of developers who have to implement all the features.

1.2 Scope

Electric mobility for All (eMall) is a user-friendly application which is intended to facilitate the use of the system to electric vehicle owners and charging station administrators.
To reach these goals, the eMSP provides an interface for end users that allow them to search for a nearby charging station that has a certain type of charging sockets (slow, fast, rapid), visualizing also its special offers providing a full integrated payment system that enable end users to pay for the service. Consequently, end users can charge their vehicles in a specific station by reserving a time slot receiving a notification when charge is completed. This feature allows to avoid traffic at the station and reduce the contemporary presence of vehicles.
The eMSP software also proposes a very smart feature: indeed, eMSP is able to proactively suggest the user to go and charge the vehicle, depending on the status of the battery, his schedule, the special offers made available by some CPOs, and the availability of charging slots at the identified stations.
Through the Web App (CPMS) managed by CPOs includes also a charging station management system which is used by the charging station administrators to manage the infrastructure (batteries, etc) of charging stations and, moreover, it has the possibility to buy the energy from 3rd parties in the smartest possible way and distribute it to end user vehicles. The managment system will also keep track of the internal and external status of each charging station respecting the physical constraints of the infrastructure.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Term	Definition
End user	Identifies electric vehicles owners who use the service, can also be referred as user.
Charging station	Place of charging of electric vehicles.
Charging Station available	At the moment at least one socket is free
Charging Station not available	At the moment all the sockets are occupied
Charging socket	Single charging point.
Booking	Reserved time slot generated from the system for charging vehicles. It also can be referred to Reservation.
Time slot	Slot of time in which custumers can charge their vehicles.
QR-Code	A QR code is a type of matrix barcode (or two-dimensional barcode).
System	Set of hardware and software tools, that provide the desired service. It can be considered as eMall, Application and Platform.
Internal status	For internal status of a charging station: the amount of energy available in its batteries, the number of vehicles being charged and, for each charging vehicle, the amount of power absorbed and time left to the end of the charge.
External status	For external status of a charging station: the number of charging sockets available, their type such as slow/fast/rapid, their cost, and, if all sockets of a certain type are occupied, the estimated amount of time until the first socket of that type is freed.
CPO	The charging station's administrator.
WebSocket	WebSocket is a full-duplex communication protocol that enables bi-directional communication between a client and a server over a single, long-lived connection

1.3.2 Acronyms

Acronyms	Meaning
eMall	e-Mobility for All
eMSP	e-Mobility Service Provider
CPO	Charging Point Operator
CPMS	Charging Point Management System
DSO	Distribution System Operator
API	Application Programming Interface
GPS	Global Positioning System
HTTP	Hyper Text Transfer Protocol
SMTP	Simple Mail Transfer Protocol
SHA-256	Secure Hash Algorithm
DB	Database
DBMS	Database management system

1.3.3 Abbreviations

Abbreviations	Meaning
R	Requirement
w.r.t.	With reference to
e.g.	exempli gratia
i.e.	Id est
etc.	Etcetera

1.4 Reference Documents

- *Course slides on WeeBeep.*
- *DD assignament document.*
- *DD review by Prof. M. Camilli.*
- *RASD eMall system*

1.5 Document Structure

This section presents the structure of the document:

1. **Introduction**: this is the first section and provide an overview of the entire document and descriptions of the main eMall functions.
2. **Architectural Design**: this section should provide an high-level analysis of functionalities, responsibilities and the main components of the entire system.
   Describes also how strategies that will be used will affect the system and also focuses on the main architectural styles and patterns adopted in the design of the system.
3. **User Interface Design**: here the graphical interfaces of the users are shown through the relative UI mockups as in the RASD document. Furthermore, it is also used to describe all the possible functions that users can exploit to achieve eMall goals.
4. **Requirements traceability**: it allows to map, in a tabular form, the functional requirements of the RASD with the requirements analyzed and considered in this DD.
5. **Implementation, Integration and Test Plan**: there it is described how the system is implemented and how the different components of the application are integrated. Moreover, a detailed description is provided on how system tests are implemented.
6. **Effort Spent**: in this section, through the use of a table, all the hours of work spent by each member of the group are defined.
7. **References**: this section lists the various documents consulted and analyzed.

Chapter 2

Architectural Design

2.1 Overview: High-level components and their interaction

The eMall system is composed by a 3-tier architecture. It's software application architecture that organizes applications into three logical tiers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where the data associated with the application is stored and managed.

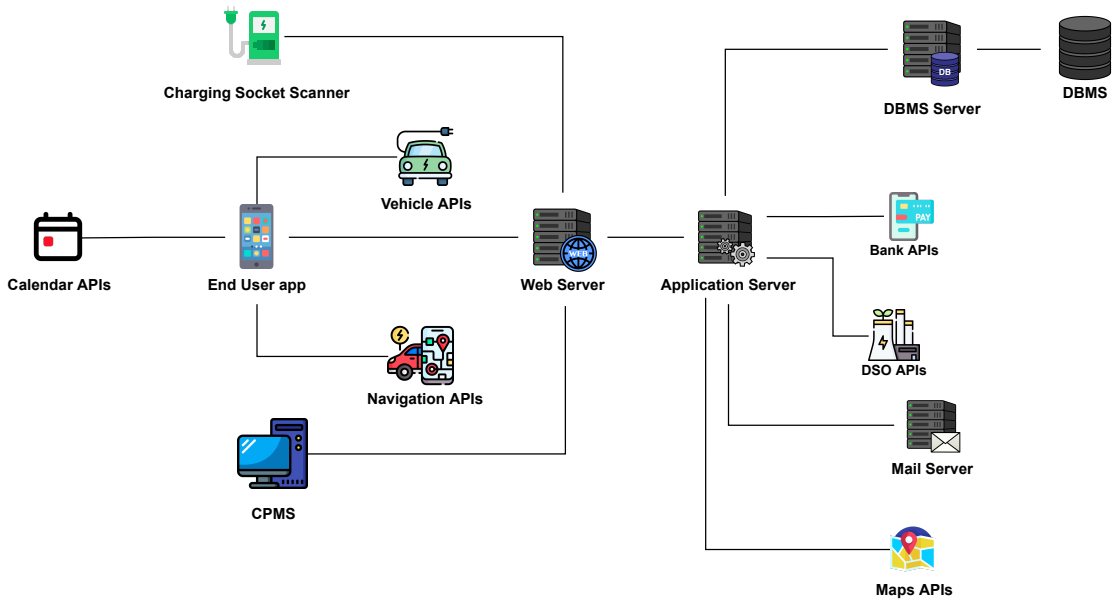

Figure 2.1: Overview eMall architecture

The architecture of the eMall system consists of two macro areas: client-side and server-side.

Client side

On the client side, we note the mobile application of the end users, the web application (CPMS) of the CPOs and the charging socket scanner:

- **End user app**: user-friendly mobile application that allows end users to manage the charging process of their electric vehicles from scratch.
  The choice of a mobile application is due to the fact that is easily to interact with the

navigation and calendar system running on the same device and, moreover, interact with some vehicles APIs.
It offers various functionalities:

  - Access to electric vehicle information (charge level) and user information (personal calendar).
  - Access to geolocation APIs that allow the identification of charging stations nearby.
  - Selecting a station based on one's preferences and needs.
  - Booking of a charging station at a specific timeslot.
  - Management of start and stop of a charge.

- **CPMS**: web application for CPOs to manage their charging stations.
  In particular a web application sufficies because the CPMS has not to communicate with a specfic CPO device or something similar. Specifically:

  - Purchasing energy from different DSOs.
  - Setting different special offers.
  - Displaying battery status.
  - Monitoring the status of own charging stations (internal and external status).

- **Charging Socket Scanner**: a simple QR-Code scanner installed on each charging socket. It allows:

  - Scanning the QR-Code of a user.
  - Connection to the eMall server, sending the code to verify that a particular end user has a valid booking.
  - Enabling charging.

Server side

On the server side, there is an infrastructure enabling the communication of the entire system, realised by various components:

- **Application Server**: server on which all the application logic resides: in addition to this, it also communicates with the Mail Server and the DBMS Server. It's also responsible for managing bank APIs to allow the system to substract end user money after a charge and for managing DSO APIs, that are useful for CPO to buy energy.
- **Web Server**: server that enables communication via HTTP with the various clients listed above and acts as middleware between front-end and back-end.
- **Mail Server**: mail server that takes care of sending e-mails (via the SMTP protocol) to the end user to confirm registration.
- **DBMS Server**: server that communicates with the database engine to manage, store and request all data related to the eMall system.

2.2 Component view

2.2.1 High level view

The component diagram reported below (Figure 2.2) depicts the system components and interfaces from an high level point of view. The **eMall server** component will be detailed in the following sections.

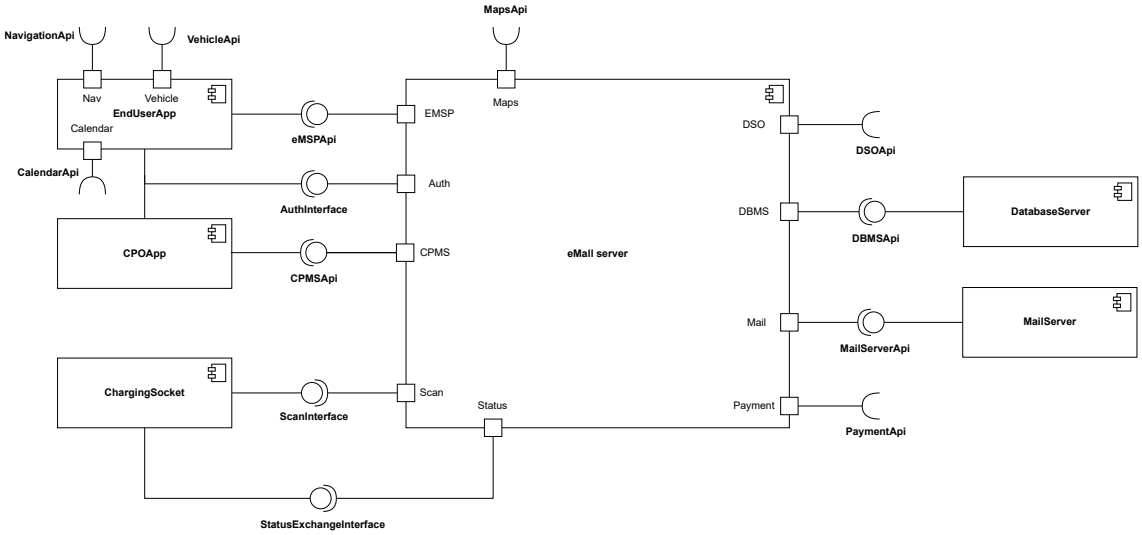

Figure 2.2: High level component description.

- **eMall server**: represents the core of the eMall system, this component contains all the logic that enables the system to process and exchange information between the various systems connected to it. More specifically the eMall server provides interfaces to both, the client and the server side. This component is detailed in Figure 2.3.
- **EndUserApp**: represents the mobile application installed on the users' devices. Through authentication (**AuthInterface**) allows access to all the features offered by the system through the **eMSPApi**.
- **CPOApp**: represents the web application available for the cpos. Through authentication (**AuthInterface**) allows access to all the features offered by the system through the **CPMSApi**.
- **ChargingSocket**: represents the charging socket, that, through the **ScanInterface** can exchange data relative to the QRCodes scanning process, while through the **StatusExchangeInterface** can exchange data relative to the status of the socket.
- **DatabaseServer**: represents the DBMS, this component provides to the eMall server, through the **DBMSApi**, the access to de persistent data saved in the database.
- **MailServer**: represents the mail server, this component provides to the eMall server, through the **MailServerApi**, all the features relative to the mail exchanging process.

2.2.2 eMall server detailed view

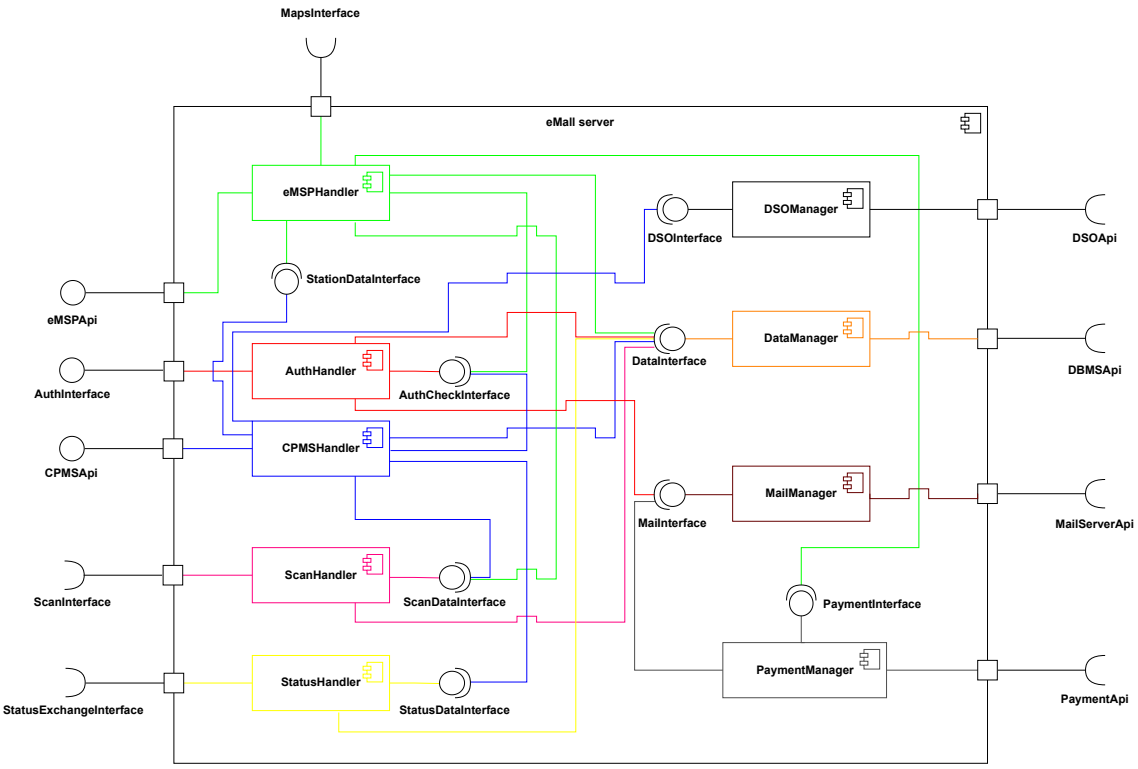

Figure 2.3: eMall server component diagram.

- **eMSPHandler**: handles all the functionalities offered in the eMSP, for example it handles the bookings creation / deletion, view all the charging stations and charging process managment. More specifically it interfaces with the CPMS through the **StationDataInterface** to get the data regarding the charging stations and their status. The **ScanDataInterface** is used to deal with the designed charging socket, while the **AuthCheckInterface** is needed to check the authorization on all the eMSP operations. The interaction with the PaymentManager through the **PaymentInterface** is usefull to allow the user to pay for a charge. The **MapsApi** are used by the component to satisfy the charging station search requests filtering by position.Finally we have the interaction with the DBMS through the **DataInterface**, this interface is used almost by all the components because is the only access point to the persistent data.

- **AuthHandler** handles logins and registration from users and CPOs, it also offers an interface **AuthCheckInterface** to allow other component to check if an operation is authorized. The interface with the mail server (**MailInterface**) is used to manage the confirmation e-mails, while the **DataInterface** is used to check the credentials correctness.

- **CPMSHandler** it handles, similarly to the eMSPHandler, all the functionalities offered to the CPO. The **DataInterface** is used to access to the persistent data. The **ScanDataInterface** and the **StatusDataInterface** are used to monitor the sockets, while the **AuthCheckInterface** is used with the same purpose as for the eMSPHandler. The last

interface to consider is the **DSOInterface** which allow the CPMS to deals with the DSOs Api.

- **ScanHandler** handles all the actions relative to the scanning function provided by the charging sockets, and expose an interface (**ScanDataInterface**) to allow the access to the scans data, for example to check a QRCode. This component is further detailed in figure 2.4.
- **StatusHandler** handles the status updates of the charging sockets. Its job is to provide an interface (**StatusDataInterface**) for other components to access the data in the various sockets.
- **DataManager** handles all the accesses to the persistent data saved on the DB, it is used by almost all the components that need the data access.
- **MailManager** handles the access to the mail server, it provides an interface (**MailInterface**) that allows the components to interact with the mail server.
- **PaymentManager** handles user payment requests and forwards them to the banking system via the external **PaymentApi**. it also interacts with the MailServer (**MailInterface**) to send receipts to users.
- **DSOManager** handles the interactions between the CPMS and the DSO through an external api, to allow this the component provides a **DSOInterface**.

2.2.3 Scan component view

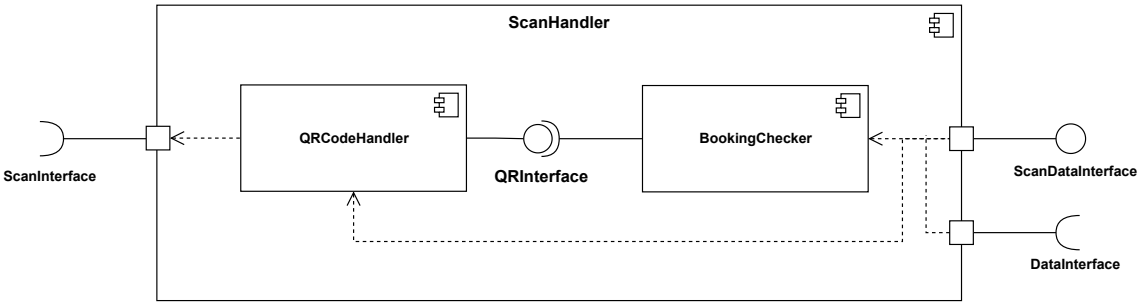

Figure 2.4: Scan component view.

- **QRCodeHandler**: handles the Scan requests, interprets them and provides the data via the interface **QRInterface**.
- **BookingChecker**: is the component which aim is to check if a code matches with a booking code. This is made by exploiting the two interfaces **DataInterface** and **ScanDataInterface**.

2.2.4 Auth component view

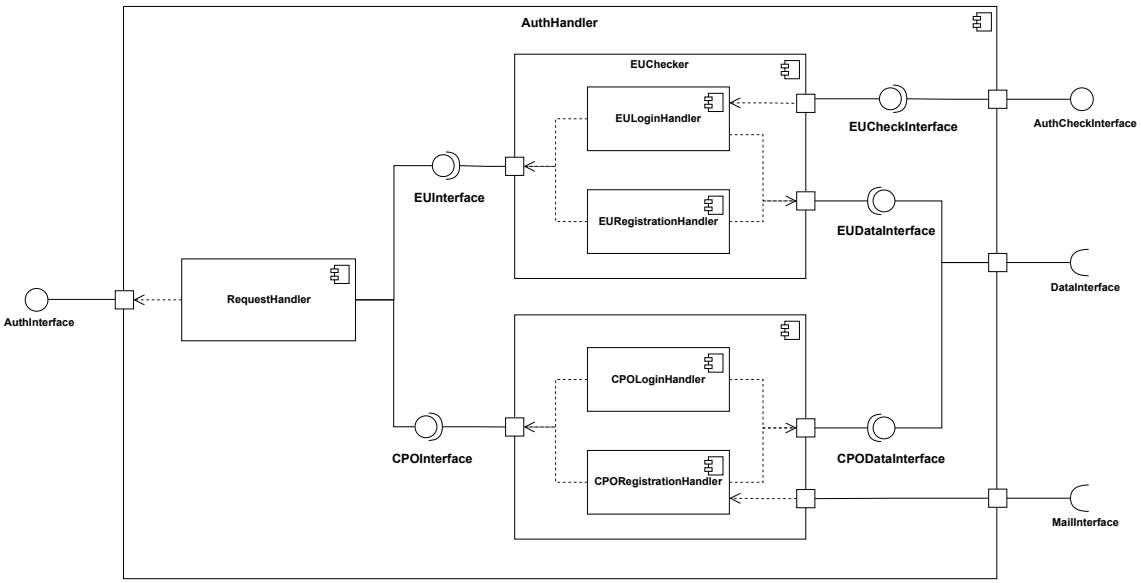

Figure 2.5: Auth component view.

- **RequestHandler**: this component handles the requests, and it acts like a router sorting requests between users' requests and CPOs' requests.
- **EULoginHandler**: handles the login requests made by the users.
- **EURegistrationHandler**: handles the registration requests made by the users. The **EUCheckInterface** allows an external component to verify the authorization of an operation.
- **CPOLoginHandler**: handles the login requests made by the CPOs.
- **CPORegistrationHandler**: handles the registration requests made by the CPOs. The **MailInterface** is used to send confirmation e-mails.

2.3 Deployment View

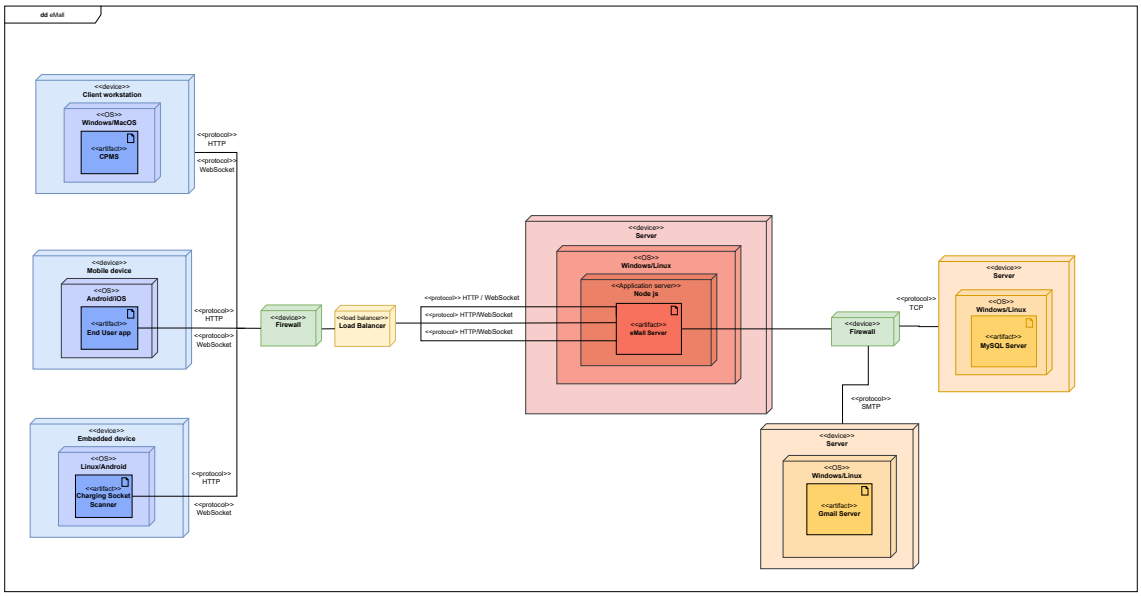

Figure 2.6: Deployment Diagram eMall system.

As shown in the figure 2.6 above, the architecture of the system is composed of **3-tiers**.
Also the presence of non-logical elements are useful to ensure a lot of benefits in the whole system:

- **Firewalls**: network security device which monitors all incoming and outgoing traffic and based on a defined set of security rules it accepts, rejects or drops that specific traffic. It allows to have greater security in the whole system.
- **Load Balancer**: A load balancing system acts as a single point of contact for clients. The load balancing system automatically distributes incoming traffic between multiple destinations. This increases the availability of the application in case of a lot of connections.

In the following, each tier is meticulously analysed:

- **Presentation tier**: It's the front-end of the application and concerns everything to do with graphics for users. It's the user interface and communication layer of the application, where the end user interacts with the application: this top-level tier can run on a web browser, as desktop application or as a mobile application. In particular:
  - *End User app*: It's installed on the user's mobile device and is compatible with Android and iOS. It communicates with the application tier through HTTP.
  - *CPMS*: It's a web application that runs on modern browsers (e.g. Mozilla Firefox, Google Chrome) and communicates with the application tier always via HTTP.
  - *Charging Socket Scanner*: It's an application installed exclusively on the charging socket in order to scan the code of a user's booking. It communicates with the application tier sending data related to the charging socket.

- **Application tier**: It's the system's application server and it's the heart of the application. It contains all the logic that is executed on multiple instances of NodeJS.
  In this tier, information collected in the presentation tier is processed and managed: moreover this tier is able to add, delete or modify data in the data tier. It also communicates with the dedicated mail server via the SMTP protocol.
- **Data tier**: consists of the DBMS server that uses MySQL to store data processed by the application in tabular form (relational database).

2.4 Runtime view

This section describes the most important components interactions of the system. For the sake of simplicity the sequence diagrams are based on the first level of components.

2.4.1 End user app login

At the beginning the end user must log in to use the main functions of the application. The login is done by entering the username and password, if the credentials are present in the database the process will be successful and he will be able to use the application, otherwise he will have to repeat the procedure.

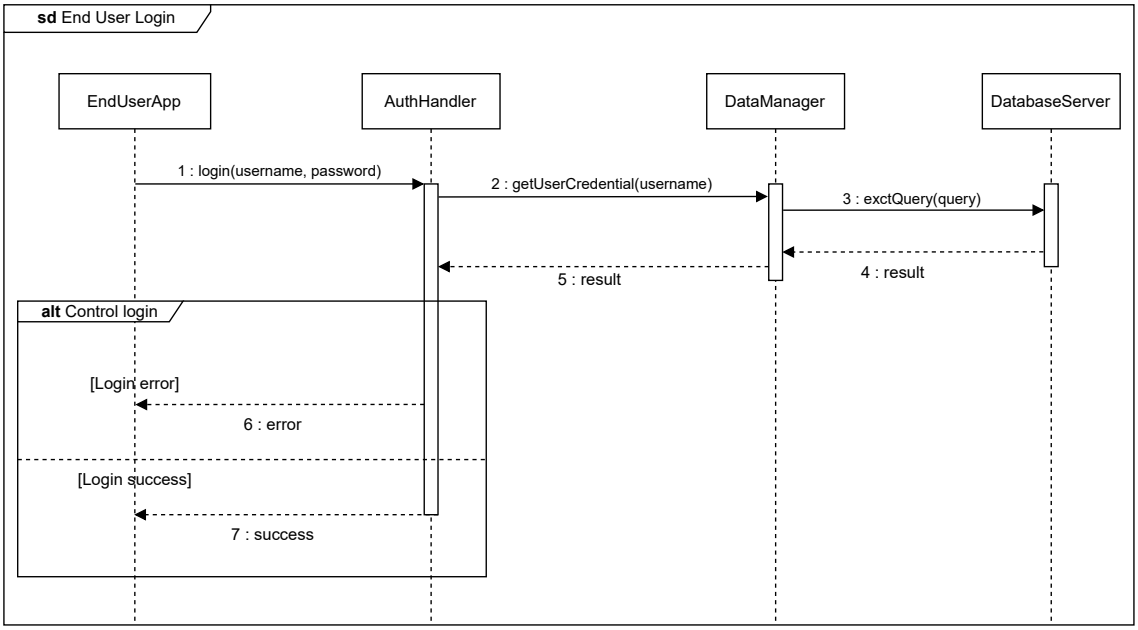

Figure 2.7: End User Login sequence diagram

2.4.2 CPO app login

In this case the login is the same as the one seen previously, the only difference is that the credentials are entered by a CPO and access to a different system.

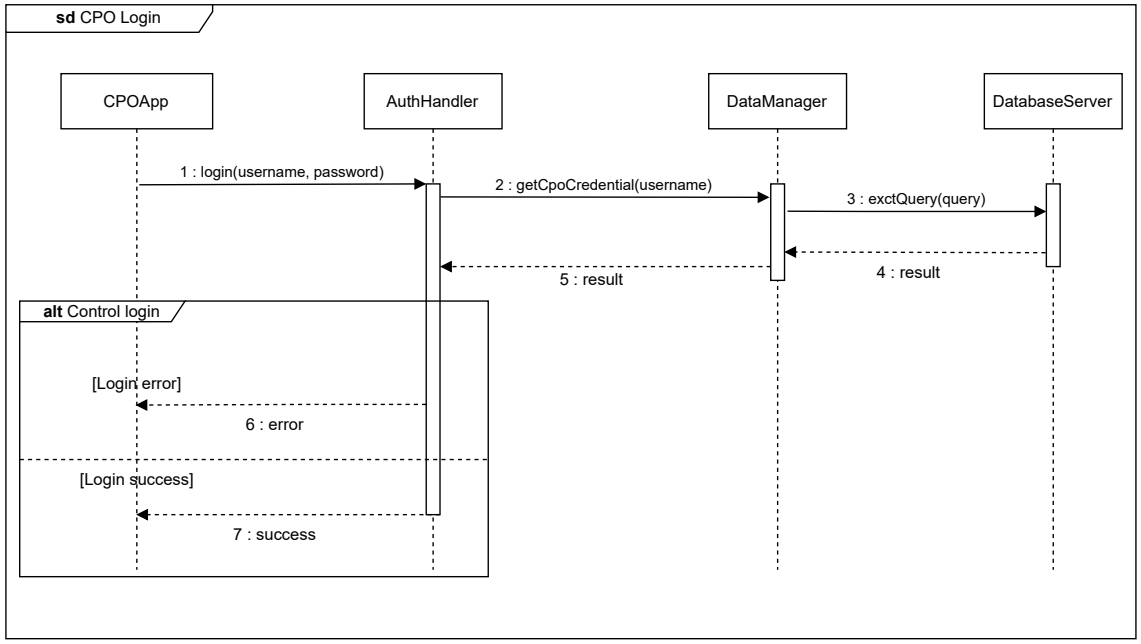

Figure 2.8: CPO Login sequence diagram

2.4.3 Make a booking

The following sequence diagram is used to explain how to make a reservation. The end user from his application can create a reservation, which is sent to the database. Within the database it is verified that those data are not already present, if so the reservation is confirmed while otherwise it is denied. If the booking is confirmed, a unique QR-Code is generated.

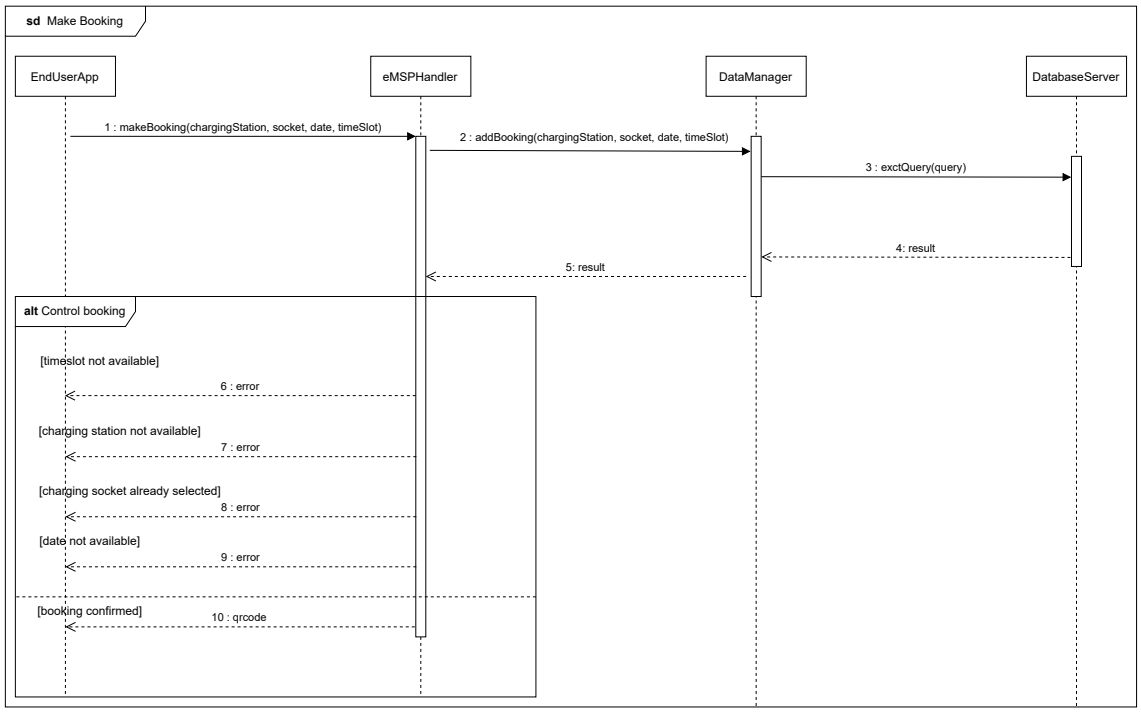

Figure 2.9: Make a booking sequence diagram

2.4.4 Check QR-Code

The socket is used to check, through a scanner, the QR-Code of the pronotation. Through this scan, the QR-Code is associated with the reservation and by communicating with the database, it verifies that the reservation is associated with the correct QR-Code.

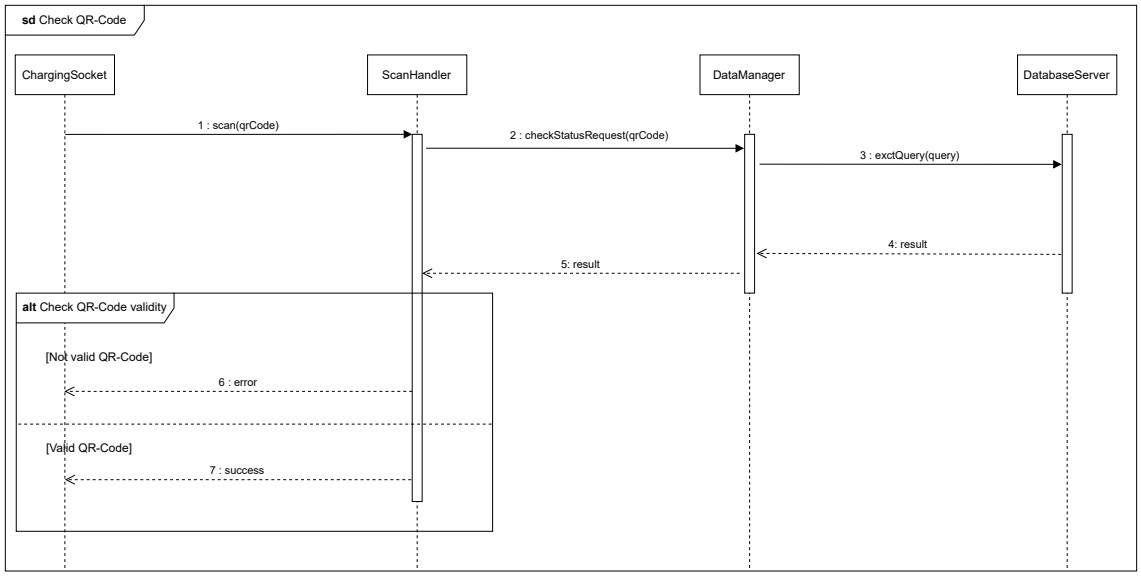

Figure 2.10: QR-Code scan sequence diagram

2.4.5 Charging station registration

The charging station registration is performed by a CPO via the web app after his login. Prompted information are checked from the system and then, if the charging station doesn't already exist, the charging station is saved into the database.

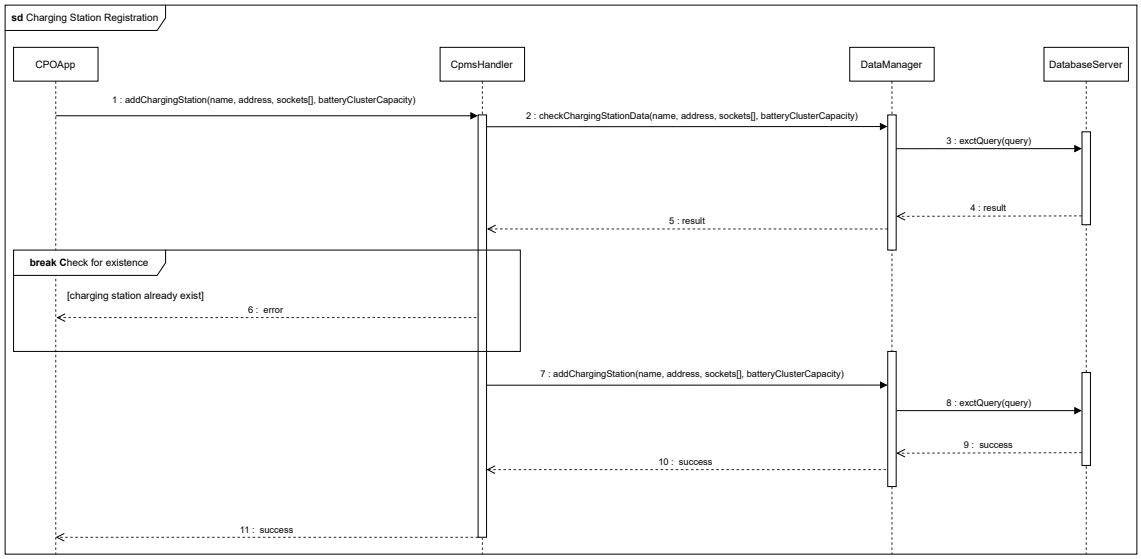

Figure 2.11: Charging station registration sequence diagram

2.4.6 End User Payment

The following sequence diagram is used to describe how money is withdrawn from the user's cards after using the vehicle top-up service. First of all, it is verified that the payment method and its data are correct through the database; after that, it is verified that there is sufficient credit to pay the requested sum and if there is, the payment receipt is sent via email.

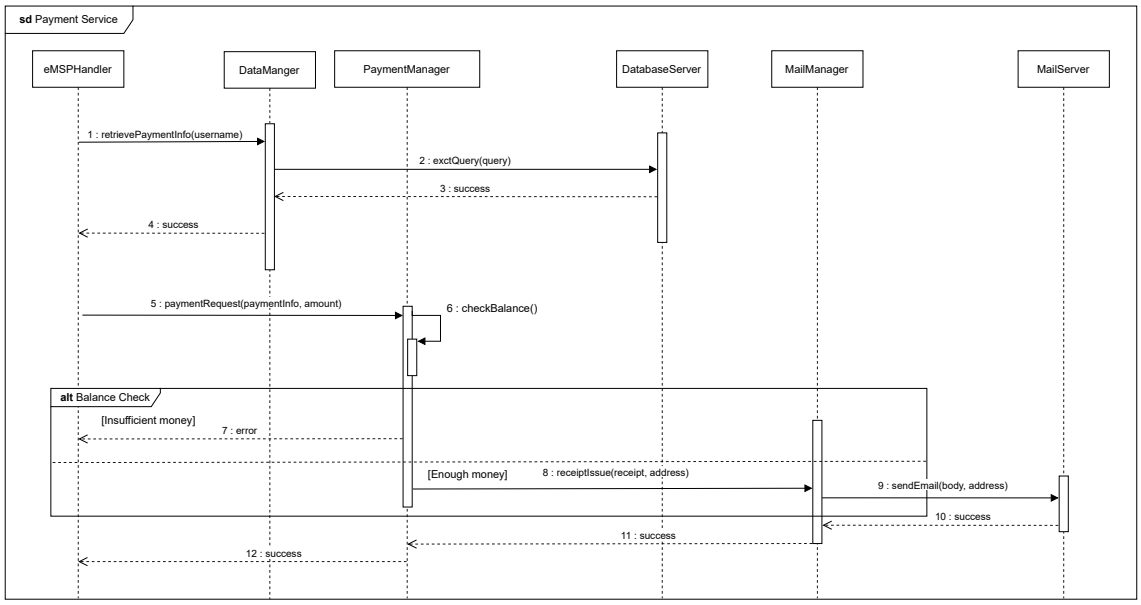

Figure 2.12: End User Payment sequence diagram

2.4.7 DSO Interaction

The interaction with DSO is performed by a CPO via the web app after his login. Through the web app, the CPO is able to view the list of energy offers presented by the various DSOs. Furthermore, thanks to the payment method external to the system for CPOs, they can buy energy from the DSOs simply by selecting the offer that most convinces them.

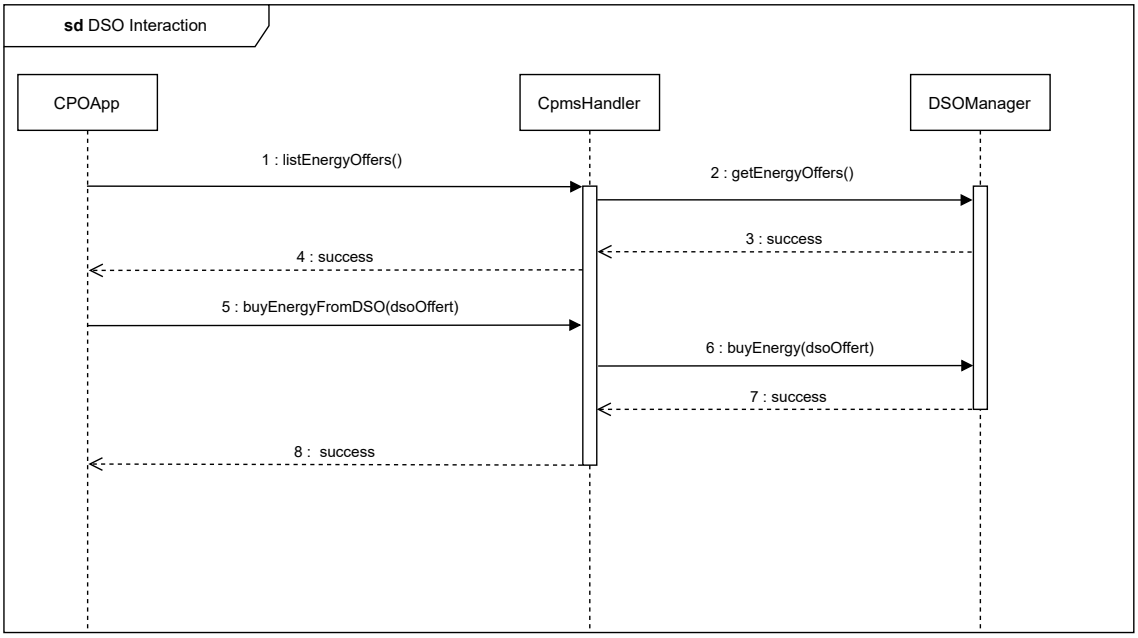

Figure 2.13: DSO Interaction sequence diagram

2.4.8 Check charging socket's status

The following sequence diagram is used to describe how sockets are checked to be busy or not. To do this it is necessary to send a request to the database regarding the status of a given socket, it is able to tell if the socket is free for use or not via a "status" flag.

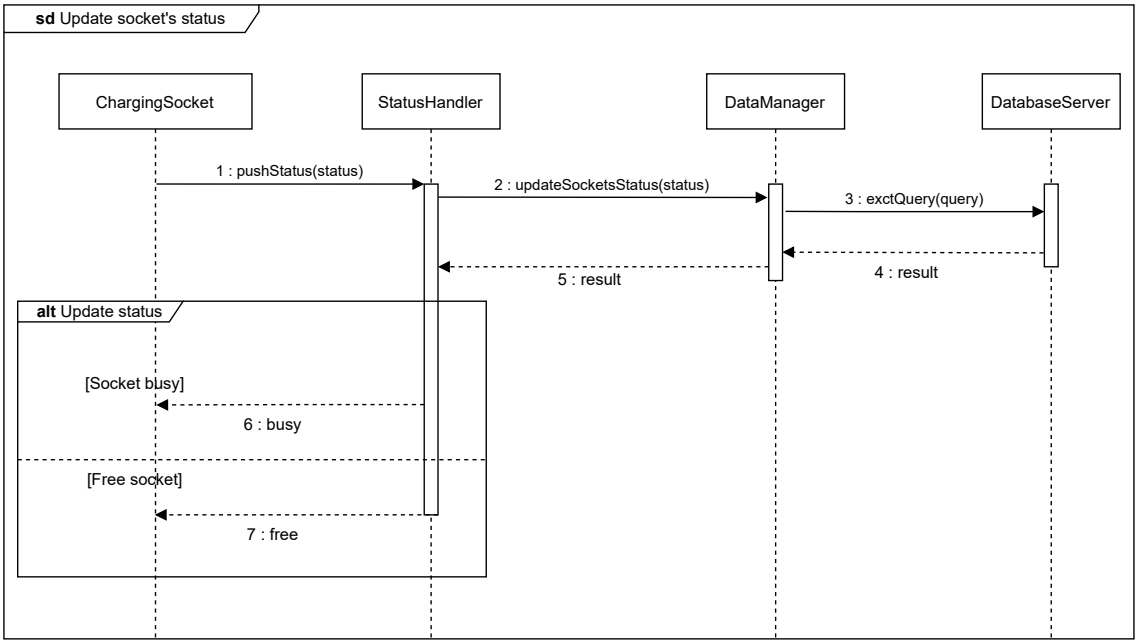

Figure 2.14: Check status sequence diagram

2.5 Component interfaces

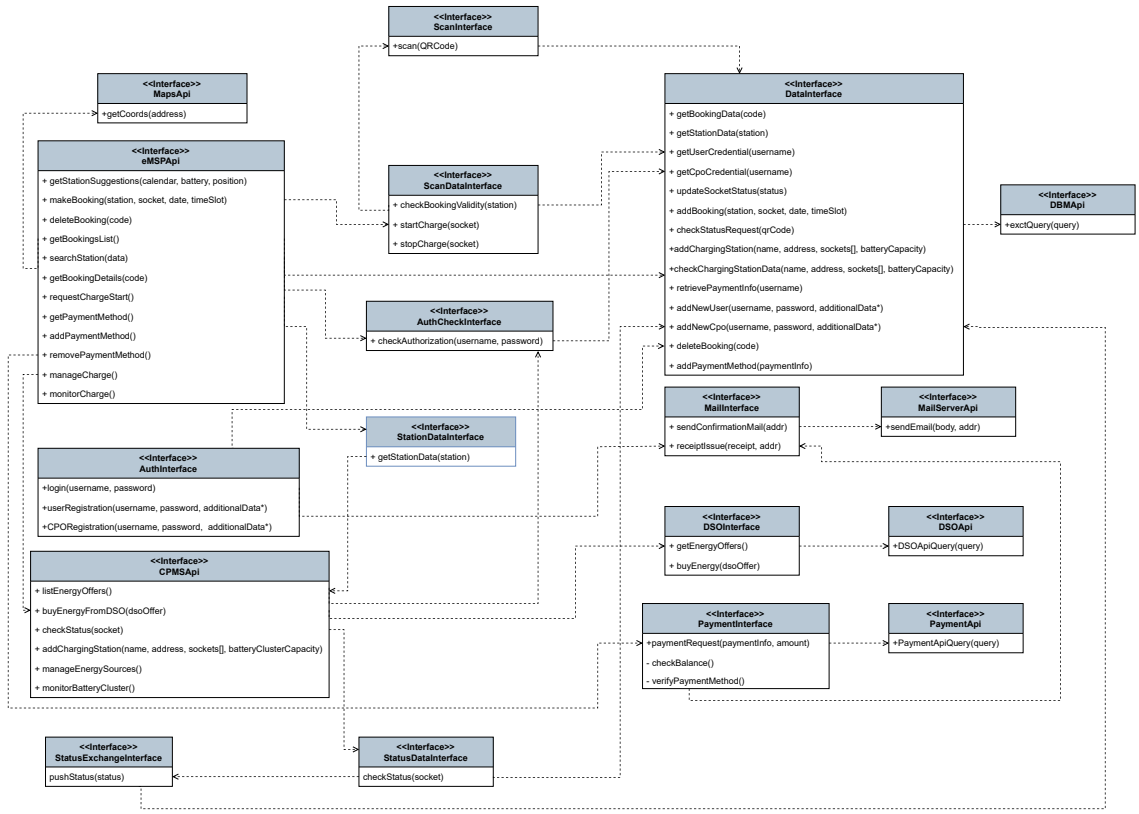

Figure 2.15: Interface diagram.

2.6 Selected architectural styles and patterns

In this section, the architectural styles and patterns of the eMall system will be analysed.

- **3-tiers architecture**: eMall system, as illustrated before, is composed by 3 tiers: presentation tier, application tier and data tier.
  This architectural choice is due to the fact that the macro roles of the system are separated as much as possible, favouring the quality of each component. Indeed, the most important features of this architecture it's logical and physical separation of functionality: each tier can run on a separate operating system and server platform and runs on at least one dedicated server hardware or virtual server, so the services of each tier can be customized and optimized without impact the other tiers. Furthermore, below are shown some 3-tiers architecture benefits:
  - **Faster development**
  - **Improved scalability**
  - **Improved reliability**
  - **Improved security**

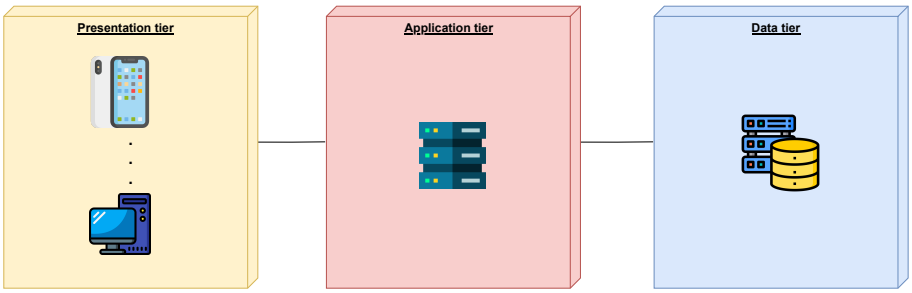

Figure 2.16: 3-tier architecture

- **RESTful APIs**: RESTful API is an interface that two computer systems use to exchange information securely over the internet. In the eMall system are used to exchange object and information in JSON format through HTTP.
- **Model View Controller**: Model-View-Controller (MVC) is a software architecture model. Provides an architecture consisting of three different parts: data (Model), data visualisation (View) and input management (Controller).
  These three components are interconnected: the Model is shown via the View to the user, who produces the input with which the Controller updates the Model.

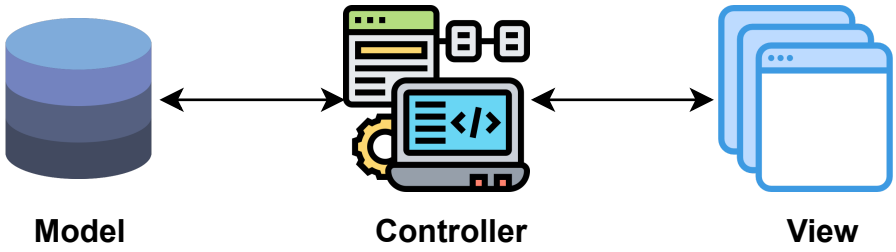

Figure 2.17: Model View Controller pattern

2.7 Other design decision

2.7.1 Real time communication

The interaction between CPMS, eMSP, the charging socket and the eMall server, for some tasks need to be real time, to do this the WebSocket protocol is used.

2.7.2 Maps APIs

The eMall system has to provide a service that allows the end user to visualize himself on the map and the nearby charging station. To achieve this goal is necessary to use specifics maps APIs that retrieve these information.

2.7.3 Database structure

In the below figures is important to note that in order to simulate the interaction between CPOs and DSOs about the purchasing of energy, has been created a DSO table on database. That is, obviously, a simplified choice given an absence of external DSOs APIs.

ER diagram

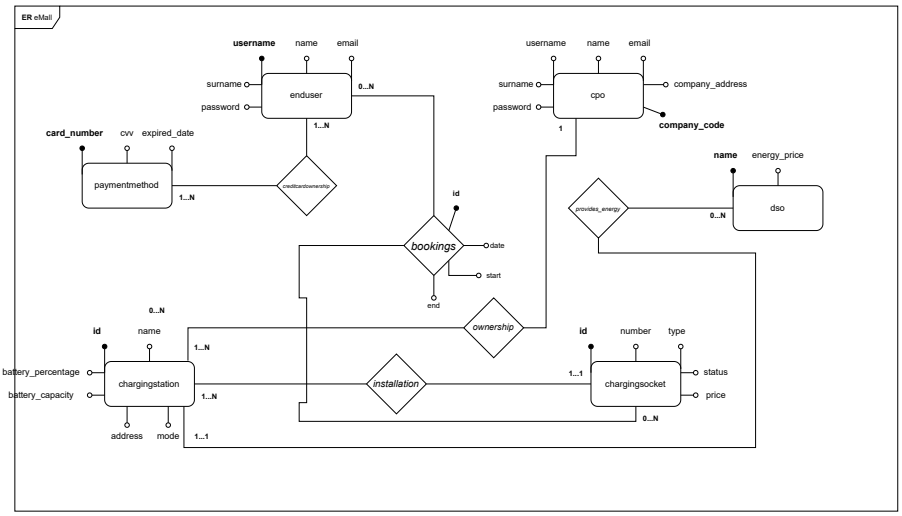

Figure 2.18: Entity-Relation diagram of eMall database

Relational schema

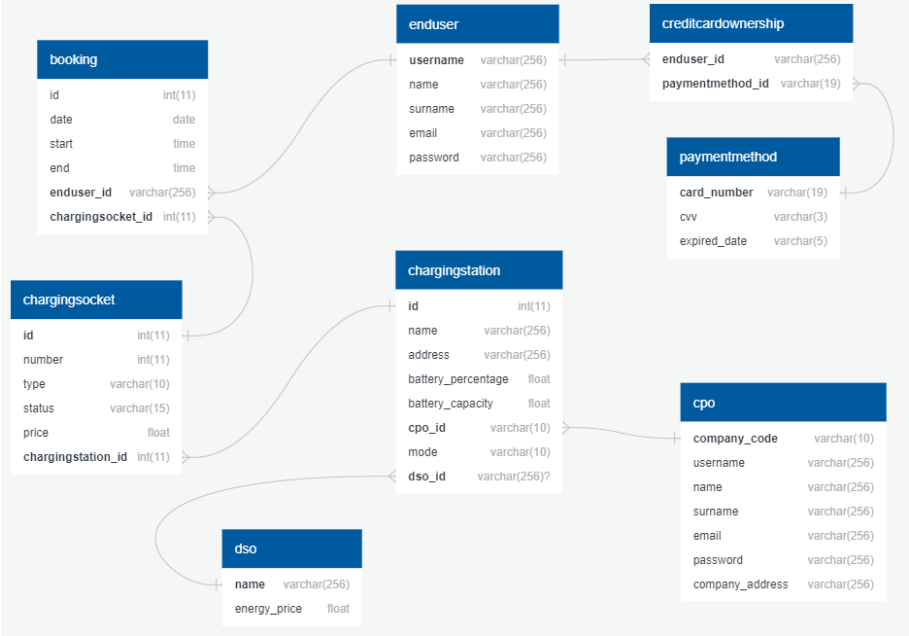

Figure 2.19: Relational schema of eMall database

Chapter 3

User Interface Design

3.1 Mockups

The registered end users use a mobile interface, to use the services provided by the eMall system. It is an application that must be downloaded onto a mobile phone to be used. The mockpus of the interfaces are shown below.

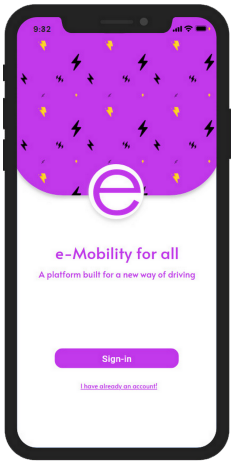

Figure 3.1: Home page

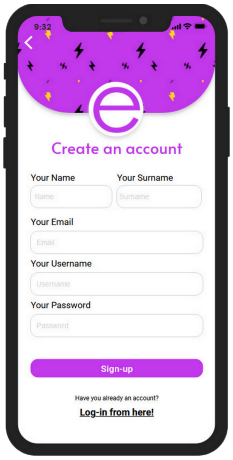

Figure 3.2: Sign up

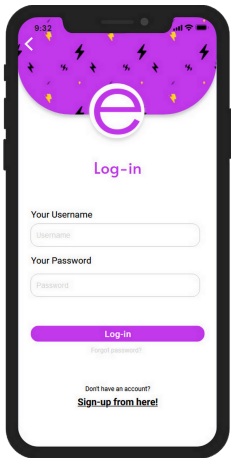

Figure 3.3: Log-in

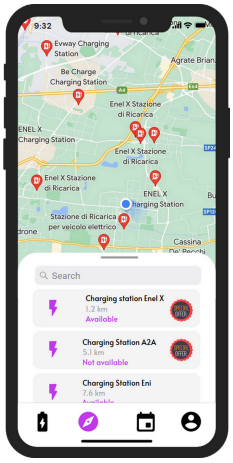

Figure 3.4: Charging stations map

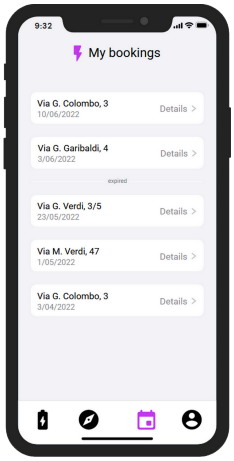

Figure 3.5: My bookings

Figure 3.6: Bookings detail

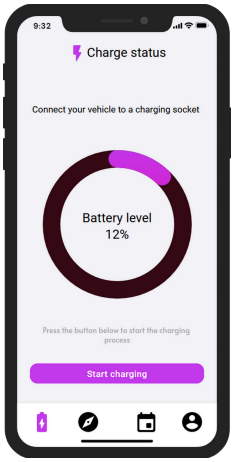

Figure 3.7: Charge status (start)

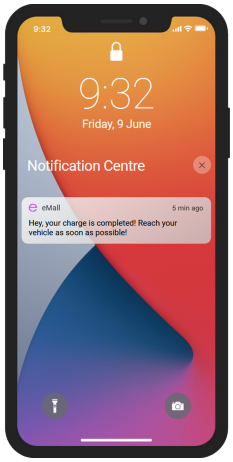

Figure 3.8: Notification

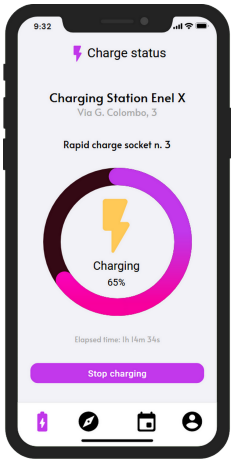

Figure 3.9: Charge status (stop)

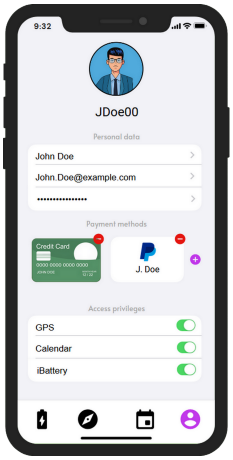

Figure 3.10: Personal information

The above mockups can be described as follows:

- **Figure 3.1**: The home page, allows the end user to choose between login and registration.
- **Figure 3.2**: The sign up page, allows the end user to register himself.
- **Figure 3.3**: The log-in page, allows the end user to access the created profile and use the app.
- **Figure 3.4**: The charging stations map, allows the end user to view the charging stations closest to him and to select the desired one.
- **Figure 3.5**: The "My bookings page" allows the end user to view past, present and future bookings. It is also possible to view where the reservations were made and by clicking on it you can get more details.
- **Figure 3.6**: The "Booking details page", allows the end user to view more details of the reservation such as: time, date, name of the charging station, name of the street, type of socket and number. It is also possible to view the Qr-Code of the reservation, which must be shown to the scanner installed on the charging sockets to allow the supply of energy. If necessary, it is possible to cancel the reservation.
- **Figure 3.7**: The "Charge status page" allows the end user to view the status of his vehicle's battery and to start the charging process.
- **Figure 3.8**: The eMall application sends a notification to the end user when his vehicle recharging process is finished.
- **Figure 3.9**: This is the same page of the **figure 3.7**, after starting the charging process, if necessary, it will be possible to stop it.
- **Figure 3.10**: There is possible for the end user to view or modify their personal information such as: name, surname, email, password, credit cards. In addition, the end user can give consent to the use of GPS and own calendar.

The CPOs use a web based interface to take advantage of the features provided by the CPMS. The CPO admins can login at the same interface as well with credential. The mockpus of the interfaces are shown below.

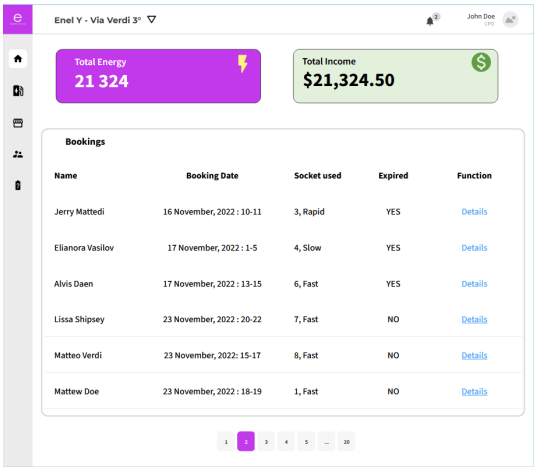

Figure 3.11: Home page

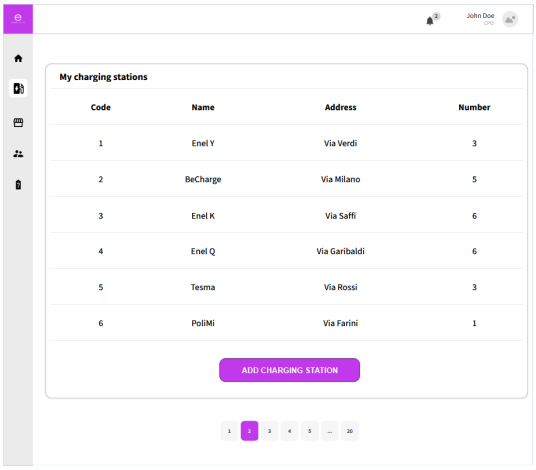

Figure 3.12: My charging stations

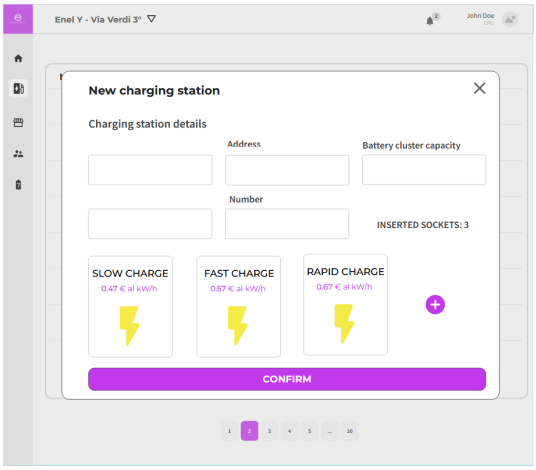

Figure 3.13: Add charging station

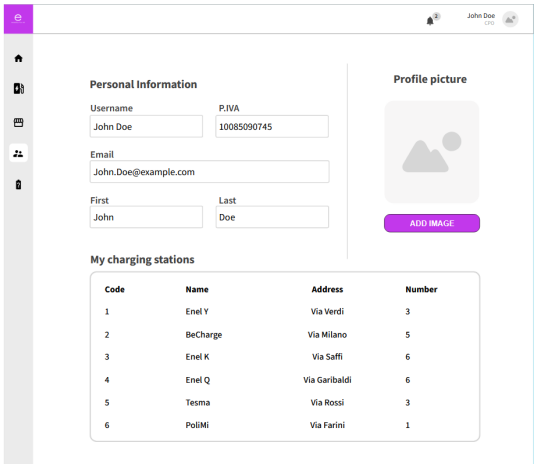

Figure 3.14: Personal information

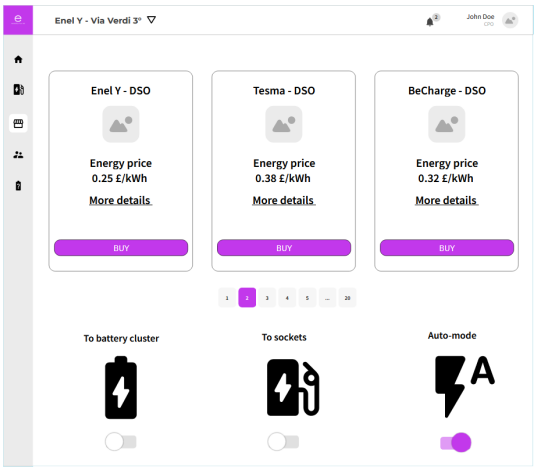

Figure 3.15: DSO interaction

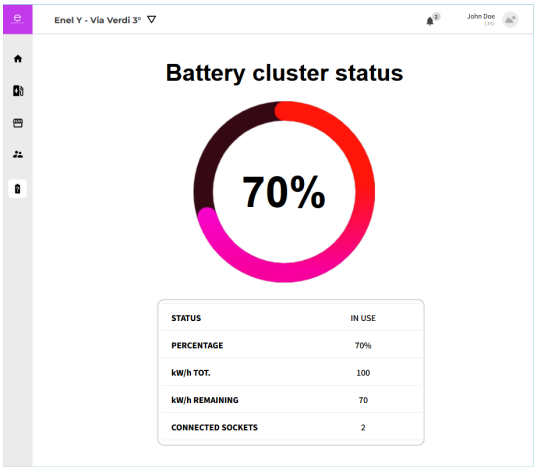

Figure 3.16: Battery cluster status

The above mockups can be described as follows:

- **Figure 3.11**: The home page allows CPO to visualize reservations made in his charging stations.
- **Figure 3.12**: There the CPO have a list of its charging stations with the related information: code, name, street. He can also add other charging station.
- **Figure 3.13**: By adding specific information required by the system, the CPO can add a new charging station with specific charging socket.
- **Figure 3.14**: There is possible for the CPO to view or modify their personal information such as: username, email, name, surname, password and company address.
- **Figure 3.15**: It is possible, for the CPOs, to view various offers made by the DSOs for energy and to buy it from the offer they deem most valid. Furthermore, here he can decide on how to manage the energy he has just bought.
- **Figure 3.16**: The "Battery cluster status page", allows the CPO to view more details of the Battery cluster such as: status, percentage, kW_h TOT., kW_h REMAINING, connected sockets.

3.2 End User Interface Flow Diagram

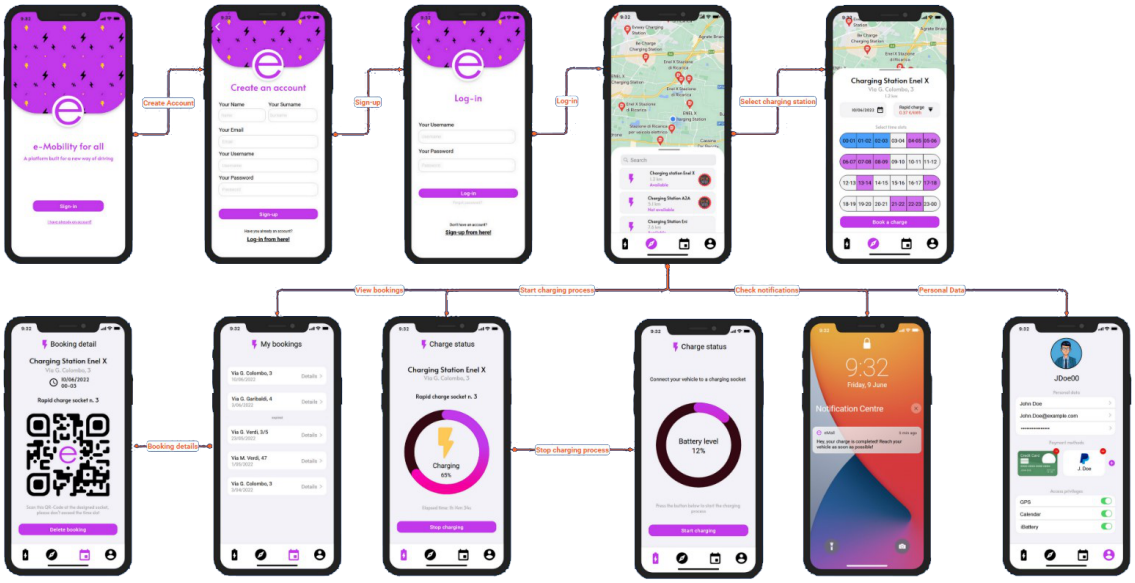

Figure 3.17: End User Interface Flow Diagram

3.3 CPO Interface Flow Diagram

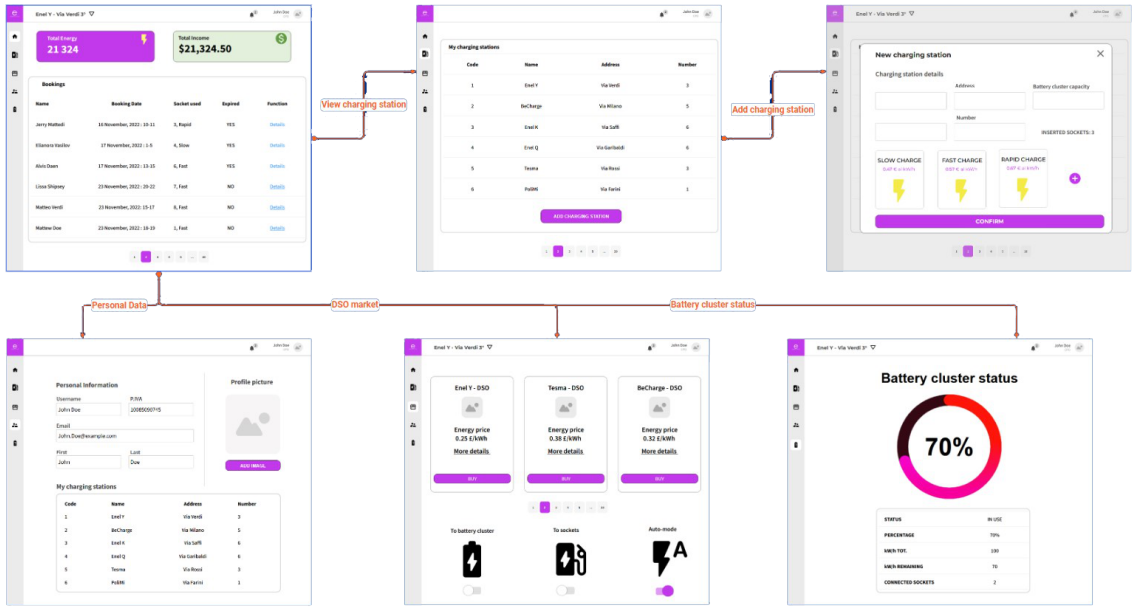

Figure 3.18: CPO Interface Flow Diagram

Chapter 4

Requirements Traceability

4.1 eMSP

Requirement	Description	Design element
R.1	The system must allow the end user to register himself to the system by providing all the mandatory information.	EndUserApp, AuthHandler, RequestHandler, EURegistrationHandler DataManager DatabaseServer
R.2	The system must allow the end user to register a payment method.	EndUserApp, eMSPHandler, DataManager, DatabaseServer
R.3	The system must verify the uniqueness of the mail and username of the end user to be registered.	AuthHandler, RequestHandler, EURegistrationHandler DataManager, DatabaseServer
R.4	The system must send a confirmation on the end user's email after his registration.	AuthHandler, RequestHandler, EURegistrationHandler, DataManager, DatabaseServer, MailManager, MailServer
R.5	The system must allow the end user to log into their account by entering their username and password.	EndUserApp, AuthHandler, RequestHandler, EULoginHandler, DataManager, DatabaseServer

R.6	The system must allow the end user to book a charging socket of a certain charging station in a specific time slot.	EndUserApp, eMSPHandler, DataManager, DatabaseServer, StatusHandler, ChargingSocket
R.7	The system must generate a QR-Code after the end user's reservation.	EndUserApp, eMSPHandler, DataManager, DatabaseServer
R.8	The system must show to the end user the suggested charging stations.	EndUserApp, eMSPHandler, DataManager, DatabaseServer
R.9	The system must show to the end user the available and unavailable time slots of a certain charging socket.	EndUserApp, eMSPHandler, DataManager, DatabaseServer, StatusHandler, ChargingSocket
R.10	The system must access to the end user's location through GPS.	EndUserApp, eMSPHandler
R.11	The system must access to the end user's calendar.	EndUserApp, eMSPHandler
R.12	The system must access to the end user's vehicle status battery.	EndUserApp, eMSPHandler
R.13	The system must allow the end user to delete his reservation.	EndUserApp, eMSPHandler
R.14	The system must notify the end user when the charge of his vehicle is completed.	EndUserApp, eMSPHandler, DataManager, DatabaseServer
R.15	The system must subtract the cost of the charge from end user's payment method.	eMSPHandler, PaymentHandler
R.16	The system must send to the end user an email with the invoice for the used service.	PaymentHandler MailManager, MailServer
R.17	The system must allow the end user to book a time slot.	EndUserApp, eMSPHandler, DataManager, DatabaseServer
R.18	The system must stop the charge when the upper bound of timeslot is exceeded.	StatusHandler, ChargingSocket

R.19	The system must allow the end user to see his current bookings.	EndUserApp, eMSPHandler, DataManager, DatabaseServer,
R.20	The system must allow the end user to search for a certain charging station.	EndUserApp, eMSPHandler, DataManager, DatabaseServer,
R.21	The system must allow the end user to stop the charging process.	EndUserApp, eMSPHandler, StatusHandler, ChargingSocket, DataManager, DatabaseServer
R.22	The system must allow the end user to start the charging process.	EndUserApp, eMSPHandler, StatusHandler, ChargingSocket, DataManager, DatabaseServer, QRCodeHandler, BookingChecker
R.23	The system must allow the end user to monitor the charging process.	EndUserApp, eMSPHandler, StatusHandler, ChargingSocket, DataManager, DatabaseServer
R.24	The system must prevent the end user from reserving a charging socket in a certain time slot, if he already has another reservation in that time slot.	EndUserApp, eMSPHandler, DataManager, DatabaseServer
R.25	The system must prevent at least two different end users from reserving the same charging socket in the same time slot.	EndUserApp, eMSPHandler, DataManager, DatabaseServer
R.26	The system must allow the CPO to register himself to the system by providing all the mandatory and commercial information.	CPOApp, AuthHandler, RequestHandler, CPORegistrationHandler, DataManager DatabaseServer

R.27	The system must verify the uniqueness of the mail of the CPO to be registered.	CPOApp, AuthHandler, RequestHandler, CPORegistrationHandler, DataManager DatabaseServer
R.28	The system must verify the correctness of the CPO's commercial data.	AuthHandler, RequestHandler, CPORegistrationHandler, DataManager DatabaseServer
R.29	The system must send a confirmation on the CPO's email after his registration.	AuthHandler, RequestHandler, CPORegistrationHandler, DataManager , DatabaseServer, MailManager, MailServer
R.30	The system must allow the CPO to log into their account by entering their username and password.	CPOApp, AuthHandler, RequestHandler, CPOLoginHandler, DataManager DatabaseServer
R.31	The system must allow the CPO to know the internal and external status of charging sockets of his charging stations.	CPOApp, CPMSHandler DataManager DatabaseServer, StatusHandler
R.32	The system must allow the CPO to monitor the charging process.	CPOApp, CPMSHandler, DataManager, DatabaseServer, StatusHandler
R.33	The system must allow the CPO to acquire information about the price of energy from DSOs.	CPOApp, CPMSHandler, DSOManager
R.34	The system must allow the CPO to acquire energy from DSOs.	CPOApp, CPMSHandler, DSOManager
R.35	The system must allow the CPO to decide whether to store or not energy.	CPOApp, CPMSHandler, StatusHandler

R.36	The system must allow the CPO to decide whether to use available energy in the batteries.	CPOApp, CPMSHandler, StatusHandler
R.37	The system must forbid to charge the vehicle to a user that scans a QR-Code which is not valid (expired, wrong etc.).	CPOApp, CPMSHandler, ScanHandler, QRCodeHandler, BookingChecker
R.38	The system must indicate whether a slot is booked or not, and it must forbid to book an already booked time slot.	CPOApp, CPMSHandler, StatusHandler, DataManager, DatabaseServer
R.39	The system must manage the concurrent booking of a slot: if a user waits too much to book a slot and meanwhile another user books that slot, the system must notify the first user when he tries to perform the booking.	CPOApp, CPMSHandler, StatusHandler, DataManager, DatabaseServer
R.40	The system must allow the CPO to add a new charging station.	CPOApp, CPMSHandler, DataManager, DatabaseServer
R.41	The system must allow the CPO to view, via the CPMS, the charge percentage of the battery cluster.	CPOApp, CPMSHandler, StatusHandler
R.42	If the user initiates charging and there is not enough charge in the battery cluster, the system, to avoid service disruption, allows the charging station to receive power directly from the DSO channel.	CPOApp, CPMSHandler, StatusHandler, DSOHandler

Why this mapping? The table above reports the mapping of the design components on the requirements. It is useful so that you have a clear and immediate idea about which components are involved in a certain to satisfy a requirement. In general, we have a division between the requirements handled by the CPMS and by the eMSP. As we can see from the table above all the components are mapped, so each component is important to satisfy a requirement.

Chapter 5

Implementation, Integration, and Test Plan

In this section it's presented the order of the implementation of subsystems and its components: moreover, also a plan of the integration and test of the integration will be presented.
Subsystems are developed and integrated following a bottom-up approach: this choice is due to the fact that individual parts of the system are specified in detail and all parts are linked to form larger components which are in turn linked until a complete system is formed. In this way can be made decisions about reusable low-level utilities and then decide how there will be put together to create high-level construct.
Important to note is that both client infrastructure and server infrastructure are developed and tested simultaneously and incrementally, thanks to their independency.

5.1 Implementation plan

As said before, the implementation of the system is realised following a bottom-up method with continuous intermediate releases: obviously the attention is focused on server-side implementation because the client-side uses only APIs and functionalities that server provides.
Specifically:

- **DBMS and DataManager**: this is the first part to implement. It incorporates all the eMall system data and allows to access to all the system data.
- **Authentication and AuthHandler**: authentication module is necessary at this point to provide privileges and authorizations to all kind of users of the system.
- **MailManager, DSOManager, PaymentManager**: external APIs module that are developed in parallel because they are each other independent.
- **ScanHandler, StatusHandler, eMSPHandler, CPMSHandler**: developed in parallel but after the development of the manager components of the point above.
- **End user app, CMPS app, Charging Socket Scan**: in the end of the whole process, it's time to implement all the client application that are responsible of UI.

5.2 Integration and Test Plan

5.2.1 Integration steps

The following images represent the integration steps for the eMall system: they provide a visual organization of the layered architecture developed.

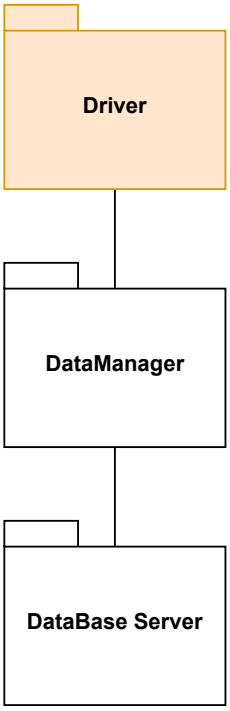

Figure 5.1: First level of integration

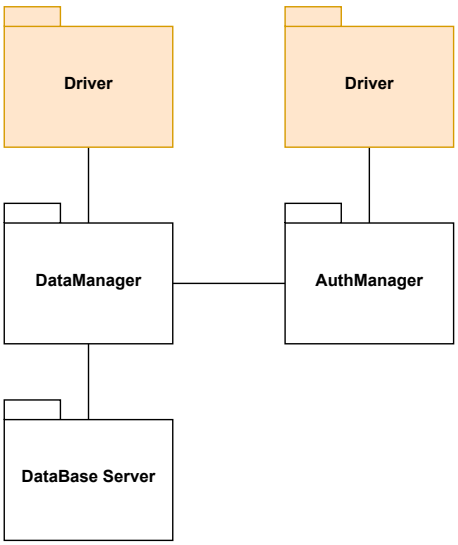

Figure 5.2: Second level of integration

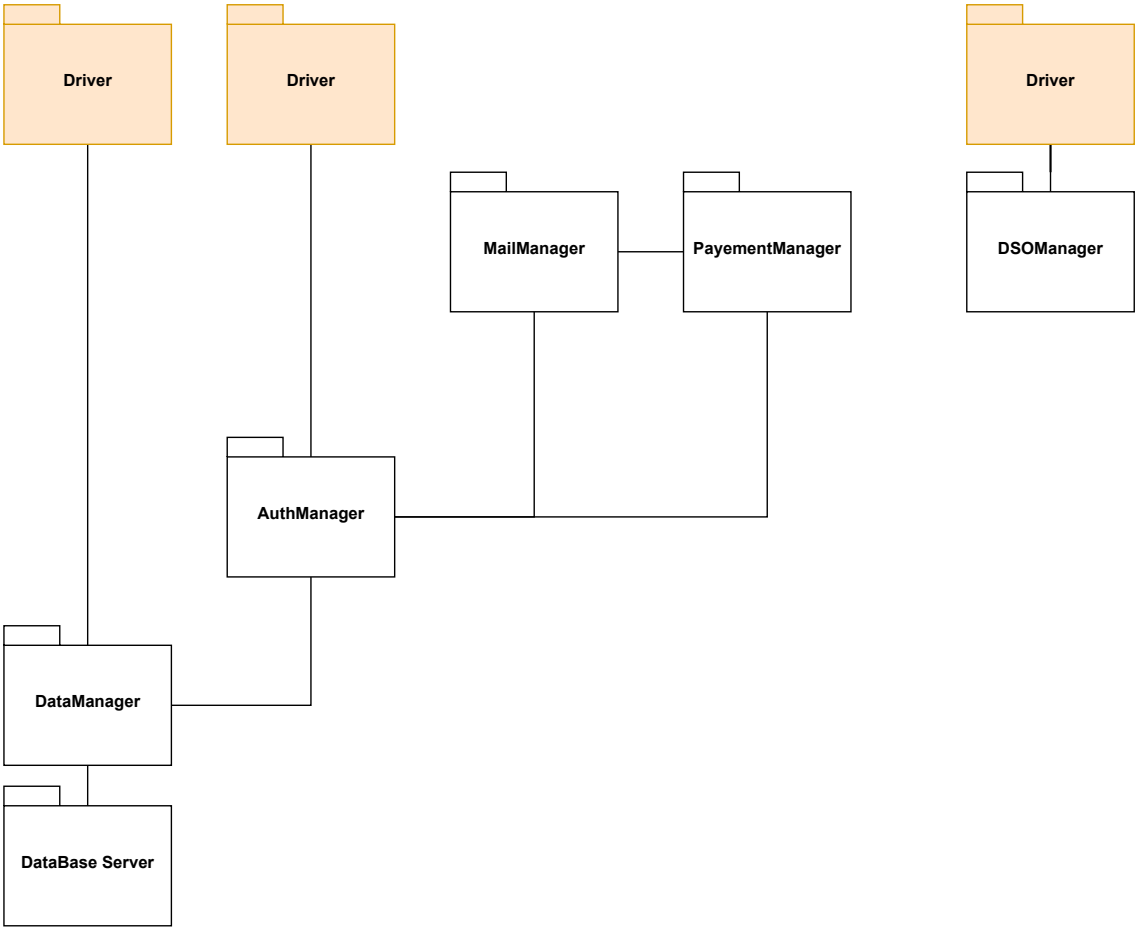

Figure 5.3: Third level of integration

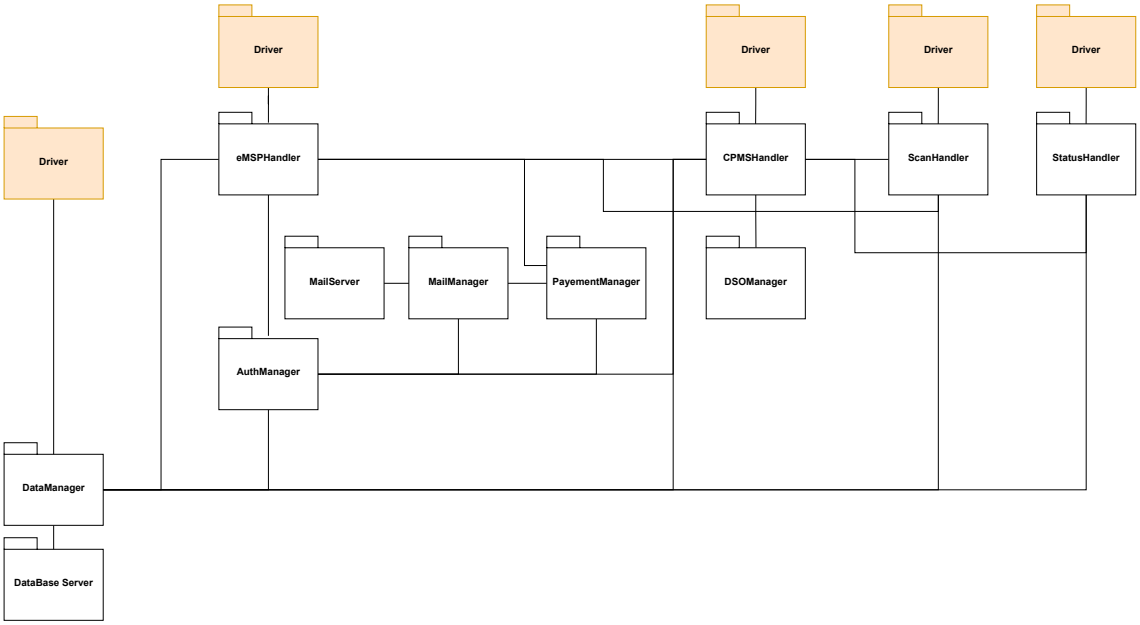

Figure 5.4: Fourth level of integration

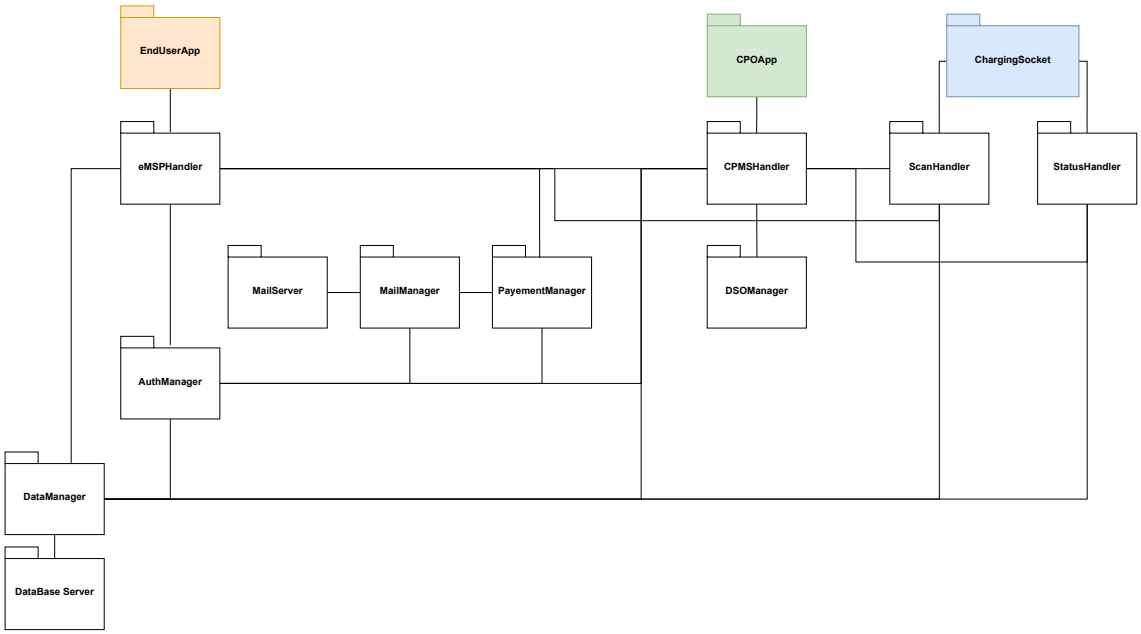

Figure 5.5: Fifth level of integration

5.2.2 Testing

During and after the developping of the eMall system, a meticulous testing phase can start to ensure that all requiremets are met. Indeed, some components can be developed in parallel and this behaviour allows to perform some testing during the implementation phase enabling intermediate releases.
Obviously, when the system is ready to be released a full testing phase can be performed on the whole system.

Chapter 6

Effort Spent

6.1 Team discussions

Topic	Hours
General discussion	0.5h
Briefing on architectural design	2.5h
Briefing on Implementation, Integration and Test Plan	2.0h
Final revision	6h

6.2 Balestrieri Niccolò

Topic	Hours
Introduction	1h
Runtime View	10h
User Interface Design	1h

6.3 Bertogalli Andrea

Topic	Hours
Component view	10h
Requirements Traceability	1h
Implementation, Integration and Testing Plan	1h

6.4 Tombini Nicolò

Topic	Hours
Deployment View	2h
Other Design Decision	7h
Implementation, Integration and Testing Plan	3h

Chapter 7

References

- *3-tier architecture*
- *MVC pattern.*
- *Network Load Balancer*