



POLITECNICO
MILANO 1863

e-Mobility for All

Design Document (DD)

v4.0

L. Padalino (10695959)

G. Paolino (10696774)

A. I. Pascu (10703025)

Academic Year

2022/2023

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Scope	3
1.3	Acronyms and Abbreviations	5
1.4	Reference Documents	6
1.5	Document History	6
1.6	Overview	6
2	Architectural Design	7
2.1	Overview: High-Level Components and Interactions	7
2.1.1	Presentation Tier	7
2.1.2	Business Logic	7
2.1.3	Data	8
2.2	Component View	8
2.3	Deployment View	12
2.4	Runtime View	15
2.4.1	External User Registration into eMall	16
2.4.2	User Login	17
2.4.3	Charging Station Reservation	18
2.4.4	Charging Process with Payment	19
2.4.5	CPO Special Offer Making	20
2.5	Component Interfaces	21
2.6	Selected Architectural Styles and Patterns	22
2.6.1	Model-View-Controller (MVC) Pattern	22
2.6.2	Observer Pattern	22
2.6.3	Three-tier Architecture	22
2.7	Other Design Decisions	22
3	User Interface Design	23
3.1	User Mobile Application	23
3.1.1	Registration	23
3.1.2	LogIn	24
3.1.3	Home Page	25
3.1.4	Charging Station Details	26
3.1.5	Booking	27
3.1.6	Trip Planning	28
3.1.7	Charging Process	29
3.1.8	Payment	30
3.1.9	User Profile	31
3.2	CPO Client Application	32
3.2.1	Special Offer Proposal	32
4	Requirements Traceability	32

5	Implementation, Integration and Test Plan	35
5.1	Implementation	35
5.2	Integration and Test Plan	37
5.2.1	DBMS	38
5.2.2	Data Visualization	38
5.2.3	DBMS Handler and Data Visualization	39
5.2.4	SpecialOfferService	40
5.2.5	Integration of different WIP sub-systems	41
5.2.6	Controller	42
5.2.7	Looking Service	43
5.2.8	Authorization and Authentication	44
5.2.9	RequestHandler	45
5.2.10	ClientApplication	46
6	Effort Spent	47
6.1	47
6.2	47
6.3	47
7	Bibliography	48

1 Introduction

1.1 Purpose

The main challenge of a massive deployment of electric mobility is the reduction of transportation's impact on climate by limiting the carbon footprint caused by our urban and sub-urban mobility everyday needs.

With this in mind, our aim is to develop and implement a new system called eMall having the purpose to expand the electric charging infrastructure with the means of realizing a world-wide network which fully connects all the actors involved in the charging processes. This is achieved by making all the back-end operations and transactions transparent to the everyday user which is fully supported in the charging process.

eMall will act as a platform deployed as a mobile application and implemented through several servers (eMSPs) located throughout the globe allowing to monitor electric mobility by communicating with various charging stations owned by CPOs, each of them administrated through a CPMS.

Thereby the platform will expose several services such as:

- knowing where to charge the EV (locate charging stations owned and managed by CPOs)
- planning charging processes in a way to limit constraints on our daily schedule giving the possibility to the system to access our personal calendar
- choose from various charging possibilities based on special offers set by CPOs which can dynamically decide from where to acquire energy to be distributed (from which DSO)

In order to guide the development of the system to be step by step, the document will focus on the description of the architectural design for the system's components alongside with their interaction. Additionally, several mock-ups of the user interface are displayed. Finally, we shift the focus on the Integration and Test Plan that we think is to be adopted in order to facilitate the development process.

1.2 Scope

This document focuses on the architectural design choices made to guide the building of the system.

Interaction between the most important stakeholders as our scope is concerned is also taken into consideration, even though several ones might be left undefined.

In particular we consider the users, which might want to use the services provided by the eMSP sub-system, the CPOs (e.g. ENEL X [1], IONITY [2] or others) physically deployed through charging stations and administrated by

ad hoc pre-implemented CPMS sub-systems, and DSOs, external third party energy providers. The system we are focusing on is the eMSP, a remote sub-system (passive server) physically deployed worldwide, which receives requests from several users and uses uniform APIs to interact with other external sources such as CPMSs and other external systems (DMV, Payment Provider, etc.) in order to execute requests and return a valid reply to the requester. On the other hand we also focus on the CPOs who may want to apply special offers based on the performance of its charging stations or automatically or manually decide from which DSOs to acquire energy from.

Our main purpose is also to achieve the maximum geographic coverage possible with our system. To do so we would want to deploy several eMSPs servers throughout the globe.

As far as the architectural design, our main purpose is to adopt a three tier architectural style for the eMall system, where the Business Logic is to be integrated to that of the already pre-existent CPMSs' one. The client will have only the presentation layer; this choice is taken because having a thin client can give the possibility to make requests from any device (smartphone or EV's infotainment system) regardless of its hardware specifications. Data layer is then decoupled in order to guarantee data safety, facilitate data migration and last but not least for importance to ensure Data Warehousing along with ETL operations.

Further details are to be found in the next sections.

1.3 Acronyms and Abbreviations

Below there's list of abbreviations used throughout the document.

Term	Definition
API	Application Programming Interface
COTS	Component Off The Shelf
CPMS	Charge Point Management System
CPO	Charging Point Operator
DB	Database
DBMS	Database Management System
DD	Design Document
DMV	Department of Motor Vehicles
DSO	Distribution System Operator
DW	Data Warehouse
eMall	Electric Mobility for All (Software Name)
eMSP	e-Mobility Service Provider
ETL	Extract Transform Load
EV	Electric Vehicle
JWT	Json Web Token
LAN	Local Area Network
MPI	Message Passing Interface
MVC	Model View Control pattern
ORM	Object Relational Mapping
OS	Operating System
RASD	Requirements Analysis and Specification Document
SD	Sequence Diagram
TCP/IP	Transmission Control Protocol (over Internet Protocol)
VIN	Vehicle Identification Number
VM	Virtual Machine
WP	Work Package
GUI	Graphical User Interface
WIP	Work In Progress
UI	User Interface

1.4 Reference Documents

For the development of this document we refer to the specification document "Assignment RDD AY 2022-2023_v3.pdf", the corresponding RASD6.0 file and the source adapted from ISO/IEC/IEEE 29148 dated Dec 2011.

1.5 Document History





 DD1.0.pdf	DD1.0	last month
 DD2.0.pdf	DD2.0	5 days ago
 DD2.1.pdf	DD2.1	4 days ago
 DD3.0.pdf	DD3.0	yesterday

Figure 1: DD Version Log

1.6 Overview

This DD is composed of seven sections.

In the first section we introduce the purpose of the project and the scope, in which we specify the stakeholders involved and present a preview of the chosen architectural design. Finally we provide a table with the necessary information including acronyms and abbreviations that are used throughout the document in order to simplify and clarify the reading.

In the second section we present an overall detailed description of the architecture design adopted for eMall, including a Component diagram in which we present the components involved in the system with a brief description of each of them, along with their interaction. Subsequently, we present the Deployment diagram of the system with a guideline to achieve the best structure of the nodes possible with the means of load balancers and firewalls. Additionally, we show a bunch of Sequence Diagrams in order to better understand how the components interact and communicate to achieve the eMall functionalities. The methods used by each component can be found in the Component Interfaces diagram, after which the choice of architectural styles and patterns are described.

In the third section we present several design mock-ups of the user interface and CPO interface with brief descriptions.

The fourth section contains the requirement traceability matrix, in which each component described in section two is mapped to the RASD requirements. The mapping is based on contribution of the component to fulfill the corresponding requirement.

In the fifth section we focus on the description of how the implementation, integration and testing of the system should be performed.

In the sixth section we introduce a history record of estimated effort spent for developing the document.

The section seven contains the references used throughout the document.

2 Architectural Design

2.1 Overview: High-Level Components and Interactions

The architecture of eMall shall adopt a three-tier pattern architecture [3] in order to decouple the Presentation Layer, the Business Logic one and the Data Layer. Thereby we can separate the Business Logic for each subsystem (eMSP and CPMS) and integrate them to create the whole eMall system.

We use firewalls at the entry of each business logic layer in order to associate to each subsystem its own data layer to provide data protection and prevent SQL injections and other malicious exploits. Each tier can run on a separate OS and server platform, so the services of each tier can be customized and optimized without impact on the other tiers, granting in this way flexibility and scalability.

Some benefits of three-tiers architecture:

- since each tier can be developed simultaneously by different teams, an organization can bring the application to market faster, and programmers can use the latest and best languages and tools for each tier;
- any tier can be scaled independently of the others as needed;
- an outage in one tier is less likely to impact the availability or performance of the other tiers

Following is a brief description of all tiers.

2.1.1 Presentation Tier

The presentation tier handles the interaction with the users, it communicates with the Business Logic of the eMSP, showing the information retrieved by this one and collecting information from the user. This top-level tier can run on a web browser, as desktop application, or a graphical user interface (GUI). Therefore we have chosen the GUI implementation through an app which allows to handle and format requests and responses, and runs both on the smart-phone and on the EV's infotainment system.

2.1.2 Business Logic

eMSP's Business Logic This tier contains all of the eMSP's Business Logic, containing for example the looking service procedure, the booking service one and the payment one along with other procedures. This component is responsible to pass the information to the presentation tier and communicates with external services such as DMV, Digital Identity Provider and Payment Provider(s).

CPMS's Business Logic This tier contains all the CPMS's Business Logic which is already implemented, deployed and up and running. For this case we assume the presence of two ad-hoc components that help us to integrate our eMSP

sub-system: `CPMSRequestHandler` and `CPOHandler`. The first one is responsible to receive the requests sent from the eMSP Business Logic (**Controller**) and retrieves the information after having done a query to its own DB. The second one manages the operations that a CPO could perform such as the application of a special offer initiative, the decision to acquire energy from certain DSOs rather than others and to manage charging stations.

2.1.3 Data

The data tier, sometimes called "database tier", "data access tier" or "back-end", is where the information processed by the application is stored and managed. Both eMSP's Data and CPMS's Data handles the persistent data of the system.

In a three-tier system, all communication from the presentation tier goes through the business tier because the presentation tier and the data tier cannot directly communicate with one another.

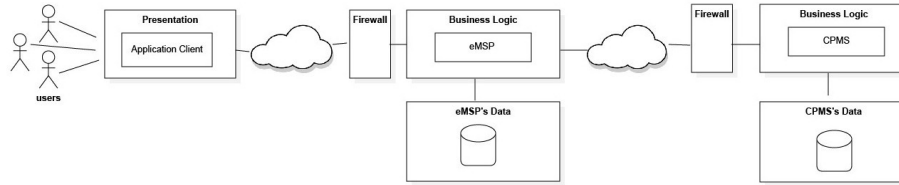


Figure 2: System Architecture Overview

2.2 Component View

The component diagram presented in this section shows how components of each tier interacts by using interfaces: it's reported the interaction between logged user and the whole application tier, and between the application tier and the data one.

Below there is a description of each component:

- **Presentation**

- **ClientApplication**: application that runs on the EV's infotainment system or on a smart-phone;
- **CPOApplication**: application used by CPOs to manage charging stations and dialogue with DSOs and eMSP's users;
- **DSOApplication**: application used by DSOs to charge their prices and energy availability;
- **ChargingStation**: endpoint of CPMS subsystem. It runs the application to perform the charging process by delivering power to the connected EV and managing the charging status by communicating to the CPMS subsystem and notifying events to the user who can interact by starting or stopping the charging process.

- **Business Logic**

- **RequestHandler**: component that manages requests sent by the user and so, after having authorized the operations, triggers the components of the eMSP subsystem properly;
- **RegistrationService**: service for registering new users within the eMSP system. It dialogues with third-party providers for vehicle association (DMV), public digital identity system (SPID or CIE) association, and payment method (PayPal account linkage) association, all reached using ad-hoc handlers;
- **AuthenticationService**: service that takes care of user authentication by verifying the existence of an account in the database after a login. When a digital identity system is used, the authentication and authorization processes occur together in the second one;
- **AuthorizationService**: service that deals with user authorization process. It verifies the validity of session, privileges and information (e.g., an owned vehicle previously entered and verified through interaction with DMV) entered. Allows the access to functionalities provided by eMall based on user access regulations ;
- **UserDataHandler**: component that handles user data in the moments after registration and authorization. It is used to create the session during which the user uses the application, or to make changes to the profile, taking care of database updates;
- **LookingService**: service that is responsible for conducting charging station looking up operations. It is requested by an authorized user, who submits his/her data (filters and/or position) to finalize a the look up;

- **BookingService**: service that is responsible for carrying out booking operations for charging stations. It is requested by an authorized user and interacts with the availability of the charging station and user's calendar;
- **ChargingProcessService**: service that deals with user by sending notification about charging process status. It receives command from the user to start the charge and stop in case of emergency or if explicitly requested by the user;
- **PaymentService**: service that contacts the available payment providers. It deals with requesting operations to verify the data entered for the payment, as well as managing the transaction and the corresponding result (commit or roll-back);
- **SpecialOfferService**: service that deals with proposing special offers to CPOs based on the trend usage of each charging station;
- **PerformanceCalculator**: component that is responsible for calculating charging station performance with respect to users' choices and routines. The final purpose is to periodically monitor the performance of charging stations and to suggest special offer initiatives to the corresponding CPO;
- **DataVisualization**: component responsible for marshalling and serialization operations of the data that will be stored on the database or sent to the client application to be visualized;
- **CPOHandler**: component that manages the operations of CPOs. It is responsible for configuring charging stations and interacts with DSOs to manage the process and decisions related to the supply and distribution of energy to charging stations;
- **CPMSRequestHandler**: component that manages requests incoming from the eMSP sub-system;
- **DBMSHandler**: component that handles database connectivity, manages queries to the DB and provides their results;
- **Model**: main component which all other components refer to, in order to apply changes on DB. It contains an object-relation mapping (ORM) that maps data objects to DB records;
- **Controller**: component that redirects the requests incoming from the eMSP services (e.g LookingService, ...) to the CPMS. It then manages the response which is forwarded to the **DataVisualization** component in order to be serialized and marshalled and finally redirected to the **ClientApplication** to be visualized or to the **DBMSHandler** with the goal to store it onto the DB. It also handles notifications sent by the CPOs thorough the **CPMSRequestHandler**;
- **CalendarHandler**: component that communicates with external calendar systems in order to check for matches between charging station

reservations and user's activities. This is done to avoid overlapping between activities. Finally the component create an event onto the user's calendar for the charging reservation;

- **CarManufacturerHandler**: component that provides vehicle status information when invoked by the eMSP subsystem. It periodically requests EV's status such as battery level and position;
- **DMVHandler**: component that manages the interaction with the DMV in order to perform license plate and VIN verification operations to associate the vehicle to the user;
- **DigitalIDProviderHandler**: component that manages the interaction with external public digital identity systems, such as SPID or CIE, in order to facilitate the retrieval of user data;
- **PaymentHandler**: component that is responsible for communicating with third-party payment providers which verifies user's payment details in order to commit the transactions which are saved as logs to DB;

- **External Components**

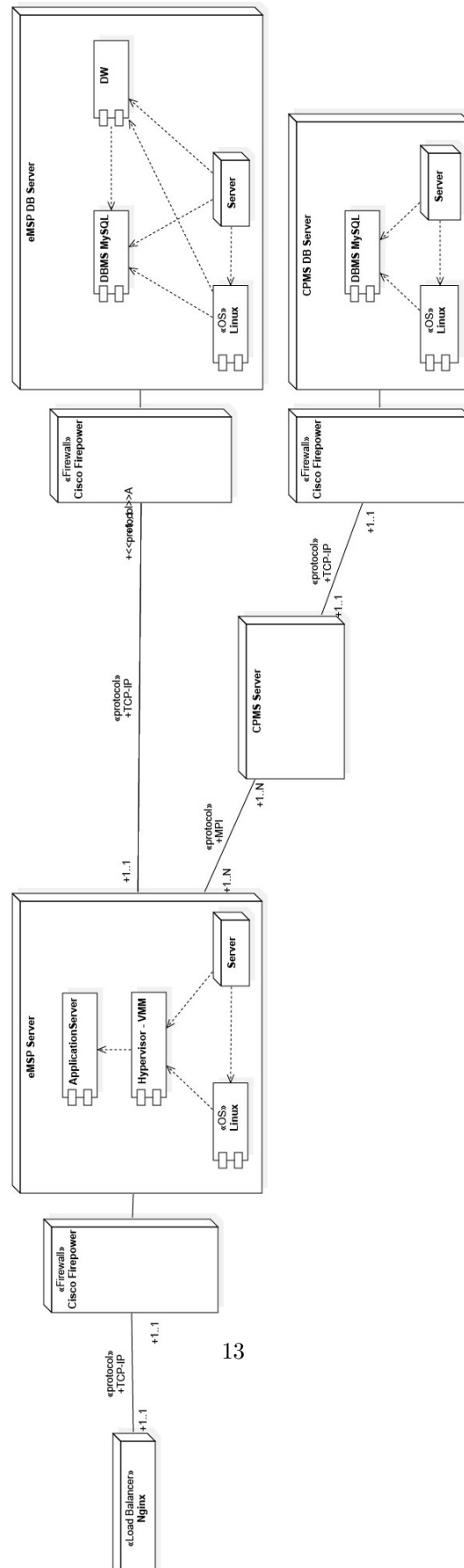
- **Calendar**: external system for user calendar (it can be Google Calendar or Outlook) reached by eMSP sub-system through the handler in booking operations;
- **CarManufacturer**: external system that collects info about the vehicle status. It provides the info to eMSP when this last one performs requests;
- **DMV**: external system that allows to check EV's details validity and so to associate the EV to registered user;
- **DigitalIDProvider**: external system for digital identification reached by eMSP sub-system through its own handler;
- **PaymentProvider**: external system for payments reached by eMSP sub-system through its own handler;

- **Data**

- **CPMS DBMS, eMSP DBMS**: component designed to enable efficient creation, manipulation and querying of databases;
- **CPMS DB, eMSP DB**: distinct databases of CPMS and eMSP subsystems.

2.3 Deployment View

The following section has the purpose to present a deployment diagram of the eMall system. The general environment and tools to be used in order to build the system from HW and SW points of view are presented along with the communication protocols used by the various nodes to exchange messages.



Client Node Client Node deployed on an EV’s infotainment system or on a Smart-Phone running the eMall application on a Linux based OS. We have chosen Linux environment thanks to the advantages it brings and the compatibility with various devices. In fact, both Apple Car Play and Android Auto are based on Linux and so are Android and iOS OSs. [6].

Load Balancer A server which handles the requests coming from various clients and manages the workload for a specific eMSP server deployed at a specific geographical location. We want to adopt a COTS so we decided to go for a Nginx solution due to its support and scalability’s efficiency [7].

Firewall A hardware defence point within the LAN’s system which provides protection from sophisticated cyber attacks led by external agents trying to overcome system’s vulnerabilities. Once again a COTS solution comes to our aid: Cisco Firepower for its over 99% threat blocking effectiveness seems to be a good choice [8].

eMSP Server Server which hosts one part of the business logic of the eMall system. It runs on top of some physical machine. The main purpose is to take advantage of the benefits that virtualization provides:

- reduced upfront hardware and operating costs;
- minimized or eliminated downtime;
- increased responsiveness and scalability;
- greater disaster recovery response;
- maximizing the usage of HW resources;

Two different approaches for virtualization can be used [5]:

- **Host Based** approach: where VMs/Hypervisors run within an already installed OS on the machine’s hardware;
- **Bare Metal** approach: where VMs/Hypervisors run directly on the machine’s hardware without the support of an OS;

VM servers are ideal for dynamic workloads stressing apps that prioritize flexibility over consistently high performance.

CPMS Server Server which hosts the other part of the business logic of the eMall system. About what we stated above, the same reasoning can be done here. But, as far as we know, the CPMS is already a deployed and up and running sub-system. We just need to integrate our eMSP subsystem with this last one in order to provide services in the best way possible: this can be done by using **adapter patterns** and ensure compatibility with the CPMS Server exposed interfaces.

eMSP DB Server Server which hosts one part of the Data layer of the system. It consists of different components:

- a DBMS, a SW which ensures and manages the creation, the manipulation and the query efficiency of the database;
- a DB, physical machine storing data in a relational manner;
- a DW [4] which elaborates data with the means of ETL operations whose purpose is to analyze data and obtain information and metadata used then by Machine Learning techniques to predict future possible values for the data yet to be acquired;

CPMS DB Server Server which hosts another part of the Data layer of the system. In particular this concerns the CPO's confidential information.

2.4 Runtime View

Below we present several sequence diagrams describing the way in which components of the system interact with one another to achieve the functionalities provided by the system.

We begin by describing the sequence diagram of the Registration to the eMall system and continue by showing the Login sequence diagram dealing with authorization and authentication. In this way the following diagrams can omit the login part. Since some features provided by eMall are similar in terms of interaction between components, we only focus on the most important ones, with respect to the use cases of RASD.

We assume that in all sequence diagrams, exception made for Registration and Login, the users are already registered and logged onto the system to facilitate the visualization in the DD. In particular, the authentication is only applied when the user accesses to the eMall system. Once he/she is logged in, each request is verified in order to be forwarded to the next components. This is made due to security concerns, to avoid the elaboration of requests coming from unauthorized users and so, manage access control. Of course, caching is preferable, when possible, to facilitate the authorization process.

We used StarUML in order to create the diagrams and due to this software standards, the lifeline of the Client cannot be materialized. However, we assume that the Client is active. For better visualization we recommend to check the sequence diagrams in the project GitHub repository.

2.4.1 External User Registration into eMall

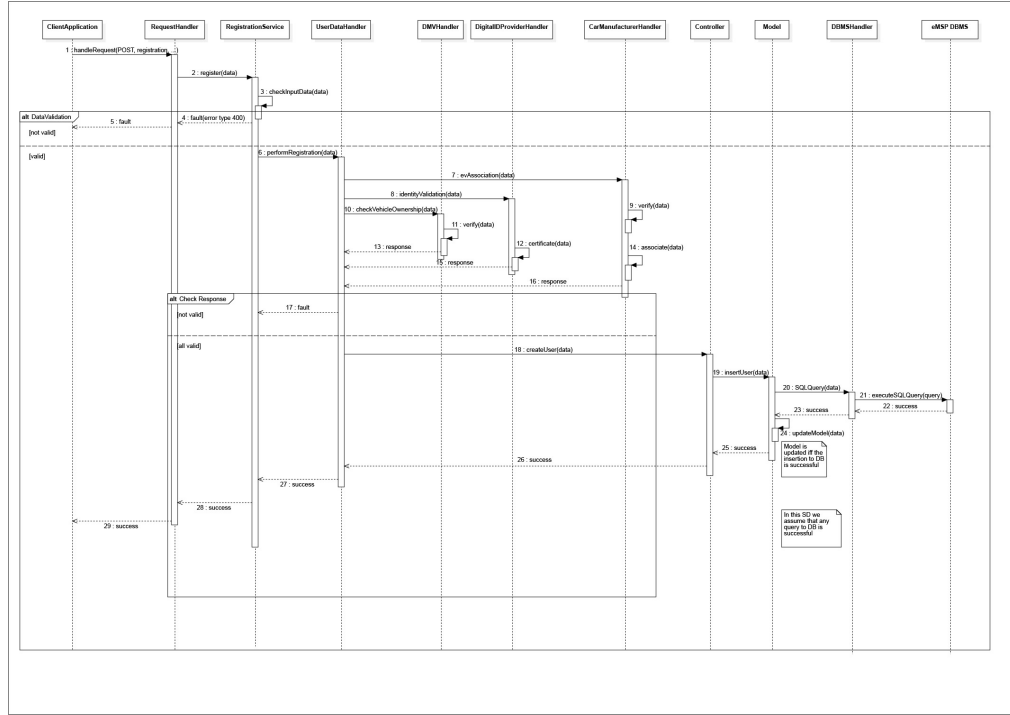


Figure 5: Registration Sequence Diagram

In this SD, the **Model** of eMall is being updated (message 24) only if the query to DB is successful. In other terms, we assume that queries are always successful.

2.4.2 User Login

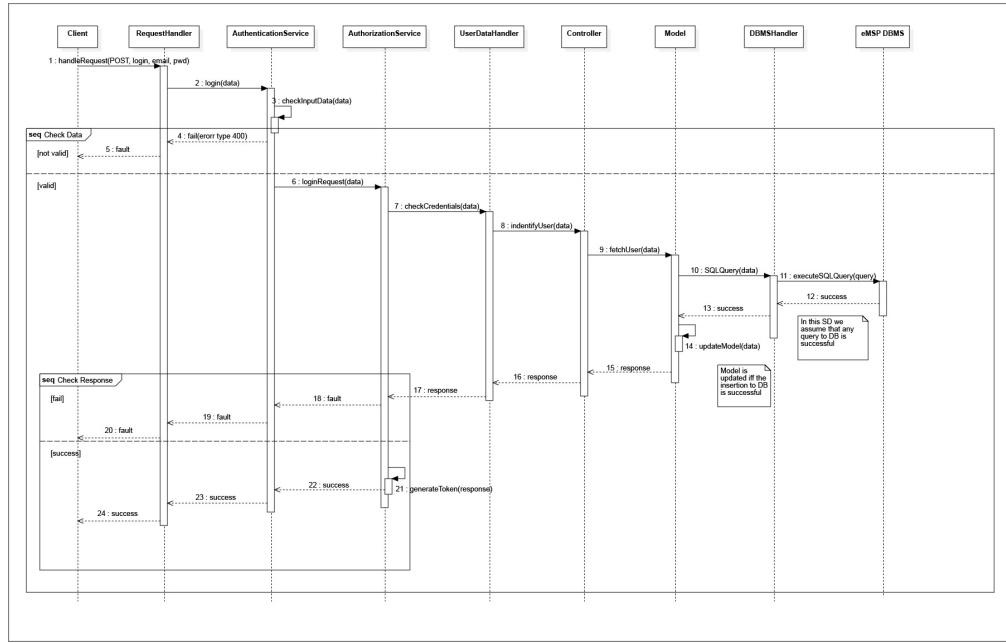


Figure 6: Sequence Diagram for User Login (authentication + authorization)

The Login scenario is where JSON Web Tokens (JWT) might come to our aid for the Authorization procedure. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access services, and resources that are permitted with that token. JWT is widely used nowadays, because of its small overhead and its ability to be easily used across different domains [9].

2.4.3 Charging Station Reservation

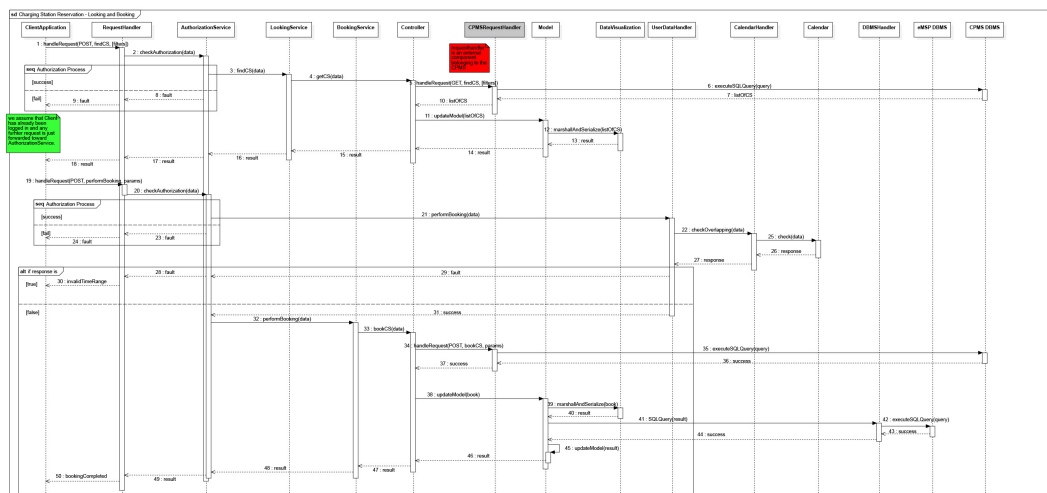


Figure 7: Sequence Diagram for Charging Station Reservation, with Looking and Booking operations

We assume here that login was already been performed and any future request is forwarded toward the **Authorization Service**. Again, JWT is good way of securely transmitting information between **ClientApplication** and eMSP subsystem. JWTs use public and private key pairs so we can be sure the senders of the request are who they say they are. Additionally, as the signature is calculated using the header and the payload, we can also verify that the content of the request hasn't been tampered by external agents [9].

2.4.4 Charging Process with Payment

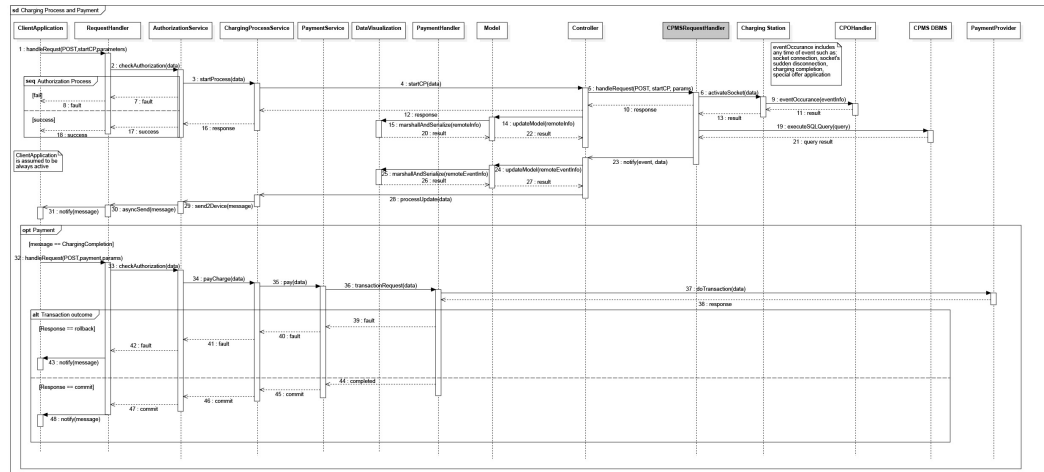


Figure 8: Sequence Diagram for Charging Process and Payment

2.4.5 CPO Special Offer Making

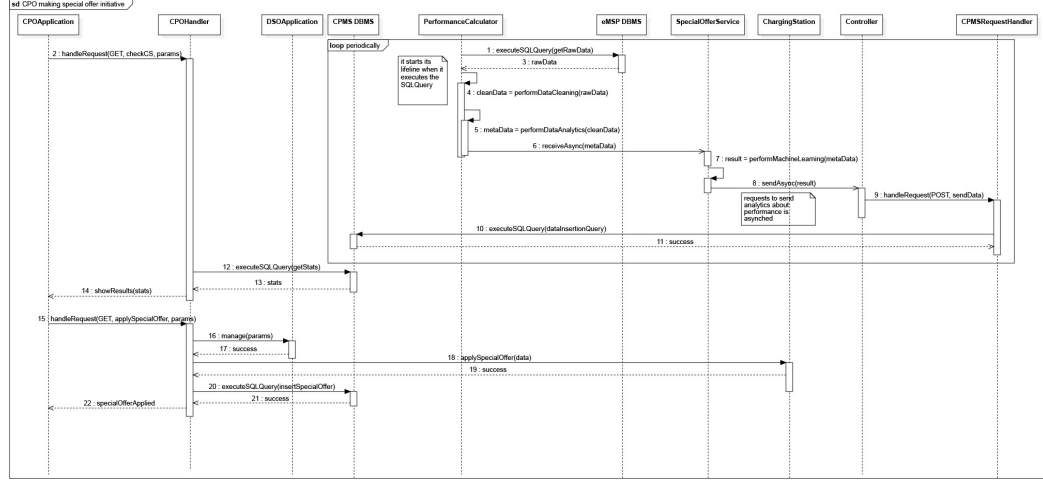


Figure 9: Sequence Diagram for CPO making special offer

In this SD we focus on the special offer initiatives along with the interaction of CPOs with DSOs.

The PerformanceCalculator component periodically gathers raw data from the DB in order to perform data analytics. Message 5 and message 8 are asynchronous messages on purpose as we assume that the requests wanting to send analytics about performance do not require acknowledgements. Additionally, we also assume the presence of reliable links and absence of failure. The analytics are stored in the CPOs' DBs and upon request by CPOs they are presented in order to suggest the CPO which could be the best special offer to apply to a certain charging station to achieve client fidelization and maximize profit.

2.5 Component Interfaces

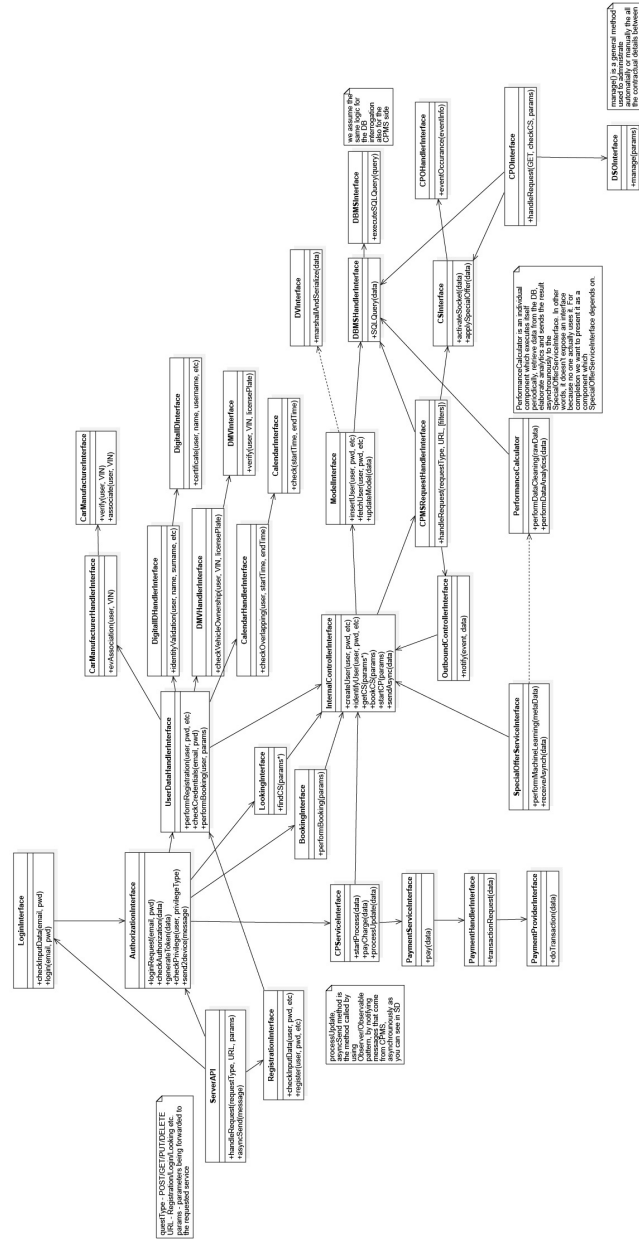


Figure 10: Component Interfaces Diagram of eMail

2.6 Selected Architectural Styles and Patterns

2.6.1 Model-View-Controller (MVC) Pattern

In project implementation, we wanted to use the Model View Controller (MVC) [10] pattern, which allows for a modular and easily scalable application. In detail, the Model plays the role of a container of information, appropriately taken from the database, implementing also the logic to dialogue with the business layer of the application; the Controller has the task of interpreting the actors' requests and instantiates and enhances the appropriate model and, subsequently, of routing the request to the right components; the view is the template according to which the model is represented: it does not contain any application logic and is only and exclusively in charge of displaying the data coming from the model. In our project, it was implemented client-side using frameworks in order to optimize resources.

2.6.2 Observer Pattern

To handle event notification in the crucial phases of vehicle charging rather than in the phases of event notification to users, we devised the Observer pattern [10]. This is a software design pattern in which an object, called a "subject," maintains a list of its dependencies, called "observers", and automatically notifies them of any state changes, usually by calling one of their methods. It is often used to implement distributed event management systems in event-driven software.

2.6.3 Three-tier Architecture

The system is structured into three layers: a presentation layer, a business logic layer, and a data layer. The choice of this architecture was mainly for reasons of security, re-usability and scalability of the system. The data needs greater protection and should be separated from the application logic both conceptually and physically, defended by very restrictive security policies and at the same time able to allow multiple accesses quickly. The other layers have been separated in order to ensure possible node replication (load balancing) and possible resource replacements and upgrades, so as not to impact the other nodes in the architecture.

2.7 Other Design Decisions

As can be seen in the component interface, we thought of separating the access points to the system to ensure different priority and security for requests coming from CPOs/CPMSs and those coming from users. Each interface has a different level of protection, particularly the one that allows the application tier to talk to the databases via the DBMS. In addition, the system makes use of external APIs with which to perform verification and associations of input data.

3 User Interface Design

Next we present some prototypes of the application user interfaces that users use to register and access the system, search, book, and monitor the status of their EV charges.

3.1 User Mobile Application

3.1.1 Registration

9:41

e-MALL
E-MOBILITY

Registration

Enter your FIRST NAME

Enter your LAST NAME

Enter your DATE OF BIRTH

Enter your ADDRESS

Enter your EMAIL

Enter your PASSWORD

REGISTER

Already have an account? [LOGIN](#)

Or

Login with Facebook

Login with Google

Login with SPID

Figure 11: Registration Form UI

In this mock-up user has to fill the form in order to register to eMall. It's possible to use external provider such as Google or Facebook or use digital identity provider such as SPID. In order to perform the association between user and EV, the app asks the user to provide EV details.


3.1.2 LogIn

9:41

E-MALL
E - M O B I L I T Y

Login
Welcome back!

Enter your EMAIL


Enter your PASSWORD 


[Forgot Password?](#)

LOGIN

Don't have an account? [REGISTER](#)

Or

 Login with Facebook

 Login with Google


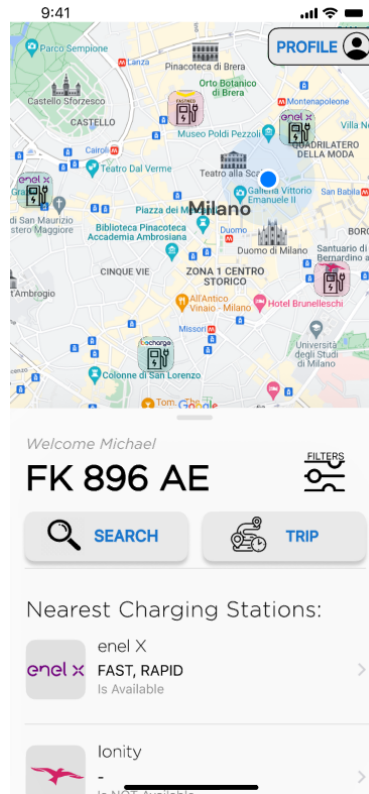
 Login with SPID

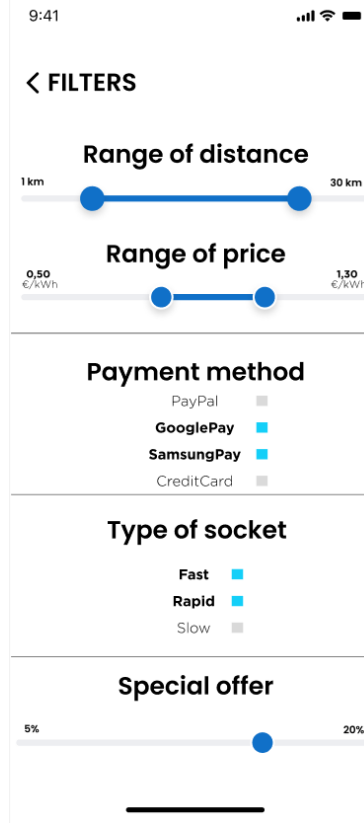
Figure 12: Login Form UI

To access to all app features user must be logged by filling this form or using external provider used in the registration phase.

3.1.3 Home Page



(a) Home UI



(b) Filters UI

Figure 13: Home and Filters UI

After having logged into eMall, the home page loads user's position and EV details, then displays nearby charging stations along with the corresponding CPO and charging station availability.

The user can use this UI also if he/she wants to look up for a charging station by using custom filters or to plan a trip.

3.1.4 Charging Station Details

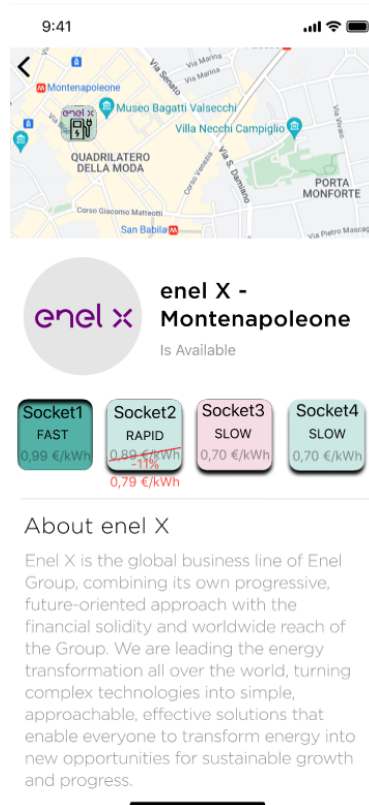
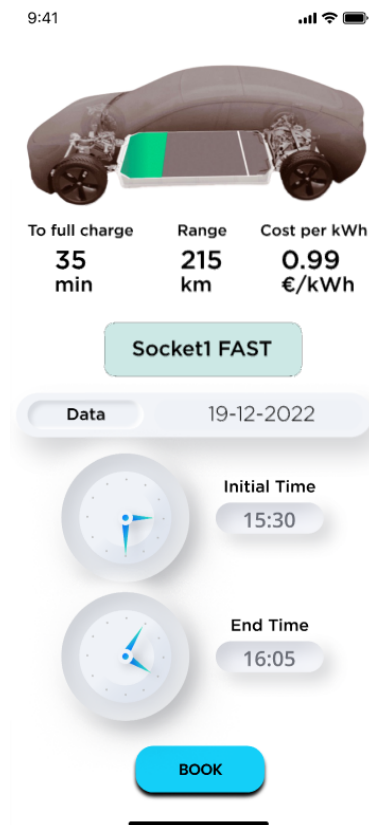


Figure 14: Charging Station UI Details

This is the UI displayed after selecting a charging station containing all the available information in terms of sockets, prices, special offers and status. Each socket can be selected in order to book a charge. In the UI **Socket1** is being selected (as the inner shadow of the box suggests).

3.1.5 Booking



9:41

Socket1 FAST

To full charge 35 min

Range 215 km

Cost per kWh 0.99 €/kWh

Data 19-12-2022

Initial Time 15:30

End Time 16:05

BOOK

The image shows a mobile application interface for booking a charging station. At the top, there's a status bar with the time 9:41, signal strength, and battery level. Below that is a 3D rendering of a car with a green battery level indicator. A table-like structure displays charging statistics: 'To full charge' (35 min), 'Range' (215 km), and 'Cost per kWh' (0.99 €/kWh). A green button labeled 'Socket1 FAST' is positioned below the statistics. A date selector shows 'Data' as '19-12-2022'. Two clock interfaces are used for time selection: 'Initial Time' set to 15:30 and 'End Time' set to 16:05. At the bottom, a prominent blue button labeled 'BOOK' is visible. The entire interface is set against a light gray background with a subtle grid pattern.

Figure 15: Charging Station Booking UI

The user can access to this UI after having clicked on the socket in the previous screen. After having inserted date and time of charge, he/she can book the socket of the charging station and so save the task in his/her calendar (e.g. available if the user has associated his Google account or others) if and only if there aren't event overlaps.

3.1.6 Trip Planning

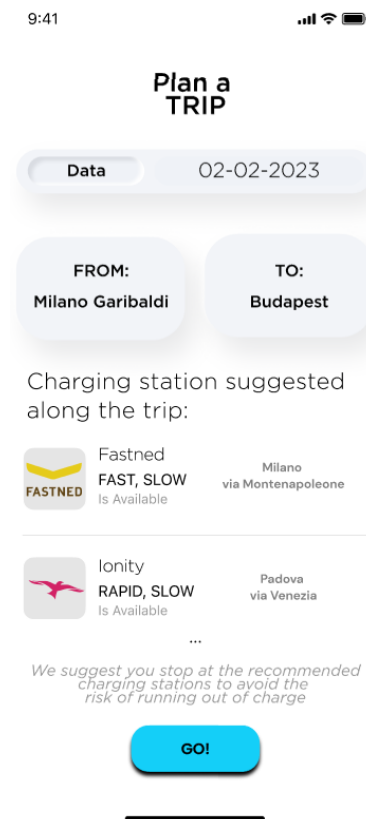


Figure 16: Trip Planning UI

If the user wants to plan a trip he/she can access all the available charging stations along his/her travel path. The charging stations are displayed according to EV's battery level at the time of trip planning with the purpose to optimize time spent on charging operations.

3.1.7 Charging Process

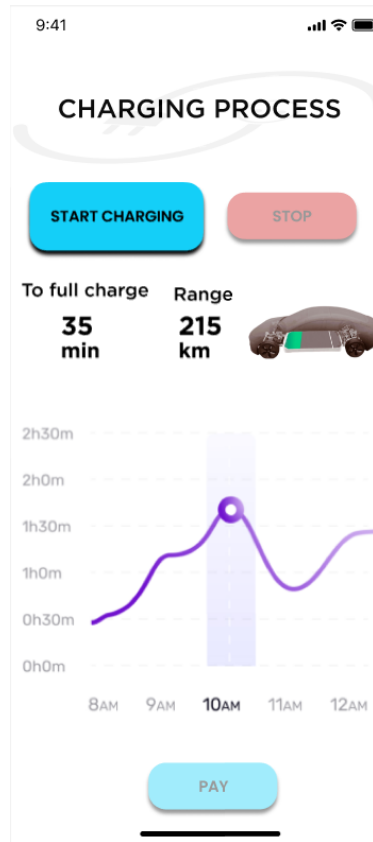


Figure 17: Charging Process UI

This is the UI that user must deal with to start a charge and manage all possible events which can occur during charging process. The user can access to this screen after having received a notification about having connected his/her EV to the socket. Once the process started, he/she can decide to stop and to pay only for the supplied energy, thus interrupting the charging process.

3.1.8 Payment

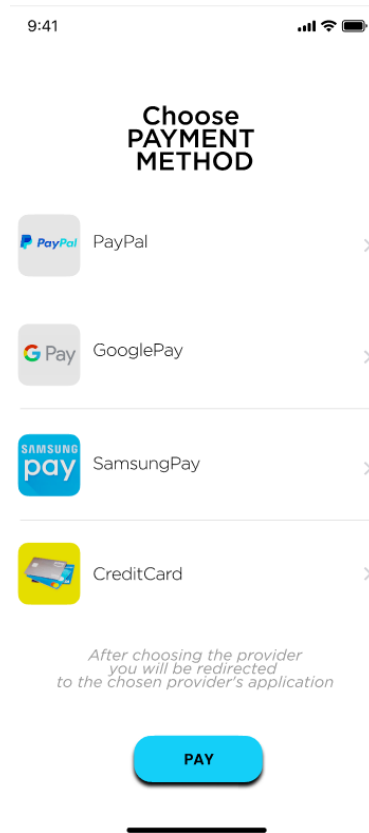


Figure 18: Payment UI

In this UI user must choose an external payment provider to complete the charging process. The transaction will be managed by the external provider, and the result will be transmitted to eMall. Now the user can resume his/her journey.

3.1.9 User Profile

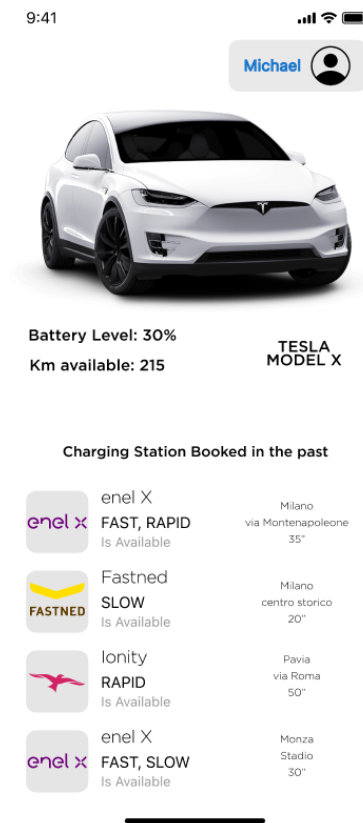


Figure 19: User Profile UI

This UI shows the User Profile details, his/her EV with data about the battery status level and the remaining estimated kilometres. Additionally, there's a list of the previous reservations the user has made.

3.2 CPO Client Application

3.2.1 Special Offer Proposal



Figure 20: Special Offer UI

After CPO has logged onto the application, it can deal with all its charging stations, and it can also monitor their performances. Above there is the screen for making special offers based on charging station performance.

4 Requirements Traceability

In the next page you can find the mapping table between requirements we defined in RASD and all the components explained in section two.

RASD Requirements																			
Components	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19
AuthenticationService		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AuthorizationService		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BookingService									✓								✓		
Calendar			✓						✓								✓		
CalendarHandler			✓						✓								✓		
CarManufacturer	✓						✓								✓				
CarManufacturerHandler	✓						✓								✓				
ChargingProcessService															✓	✓		✓	✓
ChargingStation														✓	✓	✓	✓	✓	✓
ClientApplication	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓	✓		✓	✓
Controller	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓
CPMS DB		✓		✓	✓			✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
CPMS DBMS		✓		✓	✓			✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
CPMSRequestHandler		✓		✓	✓			✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
CPOApplication													✓	✓					
CPOHandler													✓	✓					
DataVisualization		✓		✓	✓	✓	✓	✓	✓		✓	✓			✓	✓	✓	✓	✓
DBMSHandler	✓			✓	✓	✓	✓	✓	✓		✓	✓			✓	✓	✓	✓	✓

RASD Requirements																			
Components	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19
DigitalIDProvider	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DigitalIDProviderHandler	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DMV	✓																		
DMVHandler	✓																		
DSOApplication													✓						
eMSP DB	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	
eMSP DBMS	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	
LookingService		✓		✓	✓	✓		✓			✓	✓							
Model	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓
Payment																✓			
PaymentHandler																✓			
PaymentService																✓			
PerformanceCalculator													✓						
RegistrationService	✓																		
RequestHandler	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓
SpecialOfferService													✓						
UserDataHandler	✓		✓	✓	✓		✓	✓	✓	✓					✓		✓	✓	✓

5 Implementation, Integration and Test Plan

This section focuses on the description of how the implementation, integration and testing of the system is to be applied.

5.1 Implementation

The implementation of the eMall system is doomed to follow an idealistic bottom-up-based strategy as far as implementation of components is concerned. However, situations in which a mixture of bottom-up and top-down approaches may also occur within the implementation process. This is underlined beneath, in the next subsection, when we present a possible Integration and Test plan.

Individual components of the eMSP system (defined as a sub-system of the whole eMall system) are first specified in great detail and implemented. In particular, we divide the implementation into paths to be associated to team groups working on the project. This allows to breakdown the whole work into tasks and apply a separation of concerns in order to maintain a clear organization of the development and facilitate the resolution of problems in case they may occur during the implementation process.

This approach also allows the monitoring and controlling of milestones and the associated deliverables in such a way to check the progress incrementally and decrease the possibility of failure of the whole project.

The Component Diagram (Figure 3) plays a key role in dividing the tasks into the so called Work Packages (WPs) to associate to the groups involved into developing the project:

- **Client Application**

View part of eMall system only concerning with the Client's interaction. This component is only responsible for the GUI and allows the Client to interact with the system and the functionalities it exposes.

- **Model, eMSP DBMS, Data Visualization**

These components are linked to data model. We use a Relational Data Base Model in order to facilitate the storage of data by using tuples. The ORM is easily achieved thanks to Java EE. `DataVisualization` is used for marshalling and serialization which both allow a more efficient usage of data in terms of storage space maximization and also data transferring.

- **Request Handler**

This component receives various requests from the `ClientApplication` and forwards them based on the type of request to the correct Service. This could be achieved in terms of COTS or by implementing the Handler from scratch.

- **Services, Controller**

All the Components which provide services can be implemented in parallel

due to their independent feature, exception made for the Authentication and Authorization. This last two services secure the correct access to the system and so to other Services; for this reason, they should be implemented together. Lastly, the Controller redirects the correct request to the CPMS and communicates with the Model in order to store information into the local DB.

- **Handlers**

Handlers are components that allow the communication with external sources; they can be implemented from scratch or acquired as COTS.

- **SpecialOfferService**

This is a special component that needs to be implemented having in mind the **PerformanceCalculator** as a driver and the **Controller** as a stub. It periodically receives analytics elaborated by **PerformanceCalculator** which works on raw data after having it gathered from the eMSP DB; this raw data is being cleaned, and elaborated to extract useful analytics which finally is asynchronously sent to the **SpecialOfferService** component that performs Machine Learning algorithms to find out the best way to improve the services provided by a specific CPO and to maximize its profit based on certain parameters.

5.2 Integration and Test Plan

The integration of the deliverables of each milestone such the completion of the implementation of one single component starts after its functionalities have been fully tested in isolation, which requires the presence of drivers and if strictly needed, stubs. Drivers are needed as some components cannot work only in isolation so we need to simulate their dependencies.

Additionally, each component will be integrated following the bottom-up incrementally approach which facilitates bug tracking: for example, two components, tested in isolation and fully operational, are to be merged together in order to achieve other functionalities of a WIP sub-system. If the Oracle, which checks the test and the execution of the WIP sub-system composed of the two components, assesses a correct state of the sub-system and the expected output corresponds to the actual one, then the integration may further proceed until the whole eMSP sub-system is fully integrated and operational.

Finally, the whole eMSP sub-system needs to be tested with the CPMS sub-system in order to verify the correct behaviour of eMall. At first, we simulate the presence of CPMSs but the integration test needs to be run again to avoid inconsistencies and to be sure that everything works properly.

Below a graphical representation on how to follow an Integration and Test plan. We just cover a few of the several bottom-up paths in which the whole project can be divided into.

In particular we start implementing and integrating the DBMS up to the Model and various other components to finally reach the Authorization and Authentication components.

This path involves also little deviation from the ideal bottom-up approach stated above. In order to integrate in an efficient way the components, we also predict the need of a top-down approach with the help of some stubs. Nevertheless, this just resembles the inconveniences that may come up when trying to follow an ideal approach, and with this in mind we argue that a mix incrementally testing approach will best fit our project.

5.2.1 DBMS

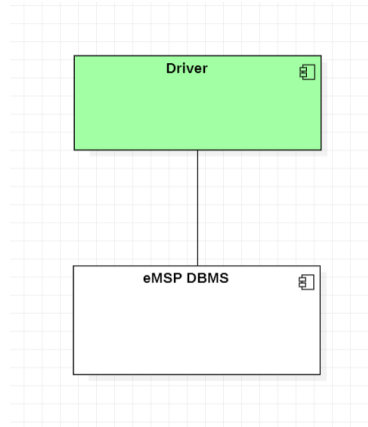


Figure 21: Integration test of the DBMS

A integration and test plan could start from the DBMS.

5.2.2 Data Visualization

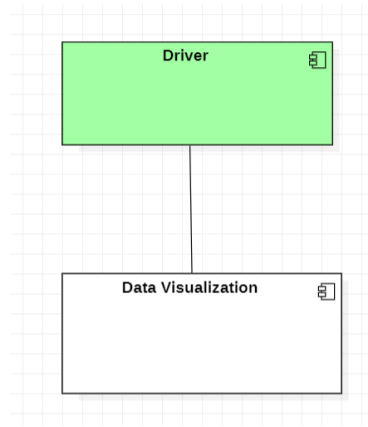


Figure 22: Integration test of the Data Visualization component

At the same time the implementation, the test and integration of the Data Visualization component could be performed.

5.2.3 DBMS Handler and Data Visualization

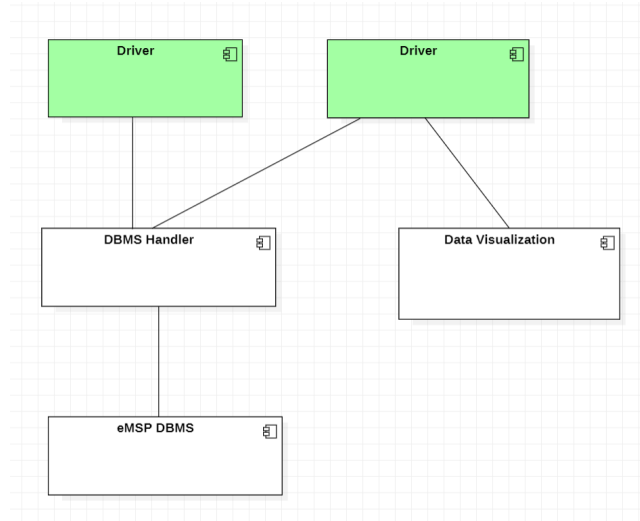


Figure 23: Integration of the two previous parallel paths

After finishing implementing the previous two components, the integration of the paths could follow.

5.2.4 SpecialOfferService

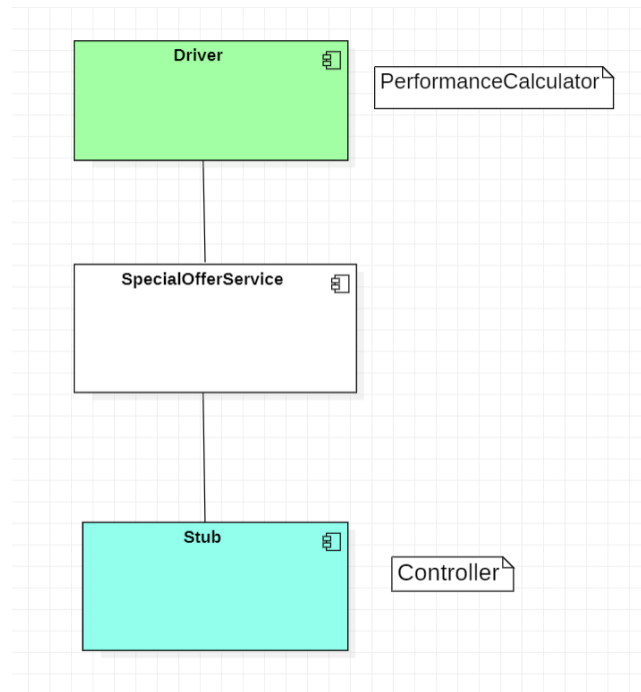


Figure 24: Integration test of SpecialOfferService

In isolation we need to implement and test the `SpecialOfferService`, and, as it is a unique component that uses and depends on two different components, this is the moment in which we use a mix approach based on driver/stub.

5.2.5 Integration of different WIP sub-systems

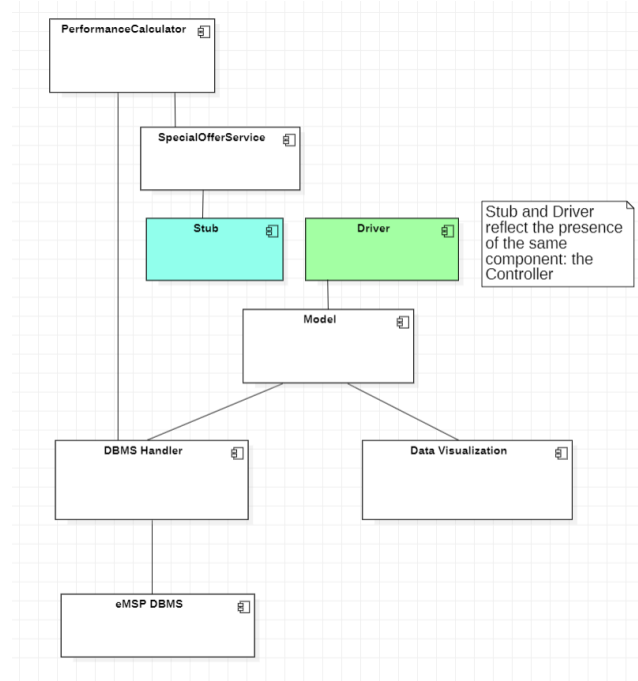


Figure 25: Integration test of various sub-systems

Once the Driver of the previous path has been implemented and up and running, we integrate it with the rest of the current subsystem at Figure 23. In this case, the Stub and the Driver represent the same component: the Controller, yet to be implemented.

5.2.6 Controller

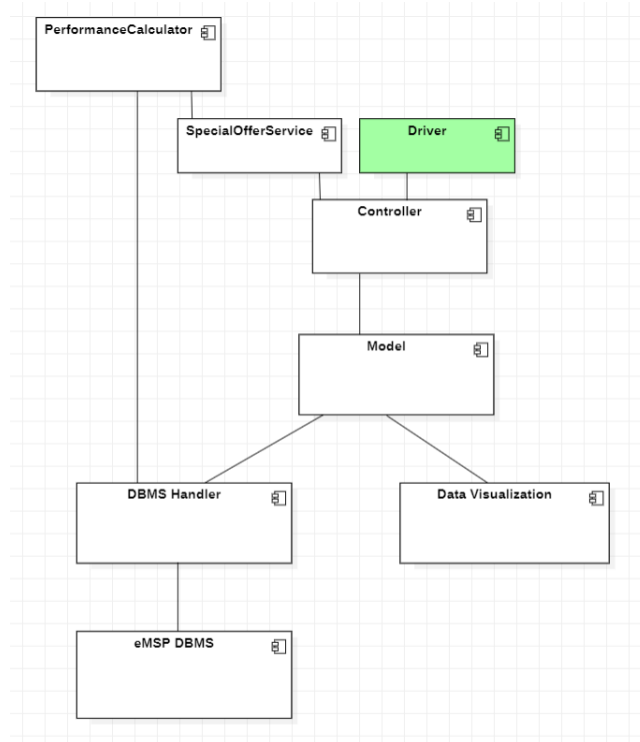


Figure 26: Integration test of Controller

The controller being implemented and integrated into the sub-system.

5.2.7 Looking Service

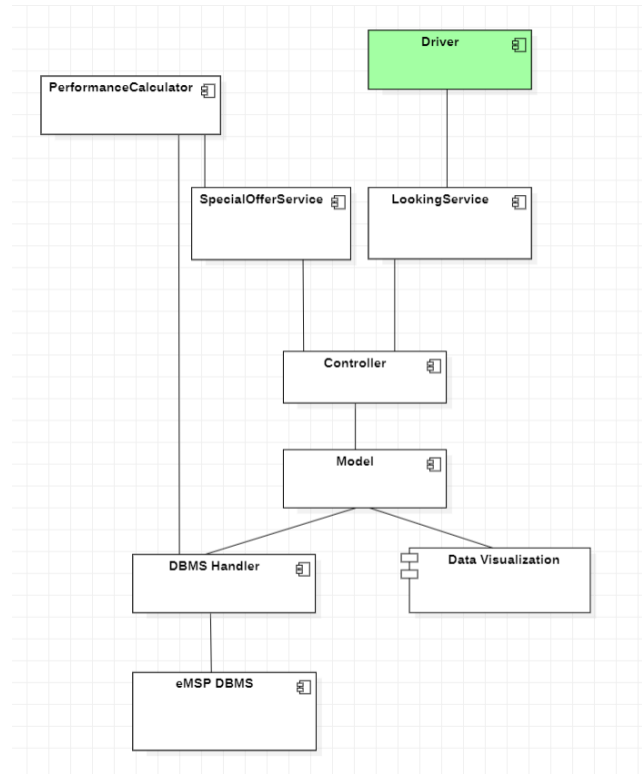


Figure 27: Integration test of LookingService component

In order to facilitate the reading of the DD, we only focus on one service component of the whole eMSP sub-system: LookingService. The same approach is to be adopted for all the other services.

5.2.8 Authorization and Authentication

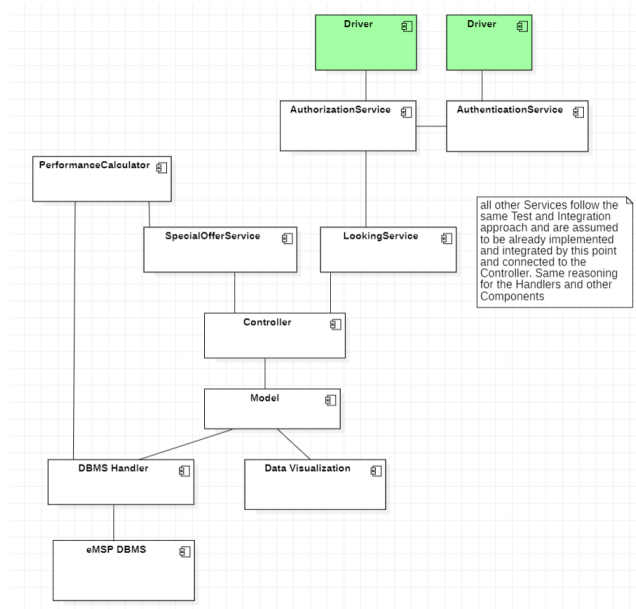


Figure 28: Integration test of Authorization and Integration

5.2.9 RequestHandler

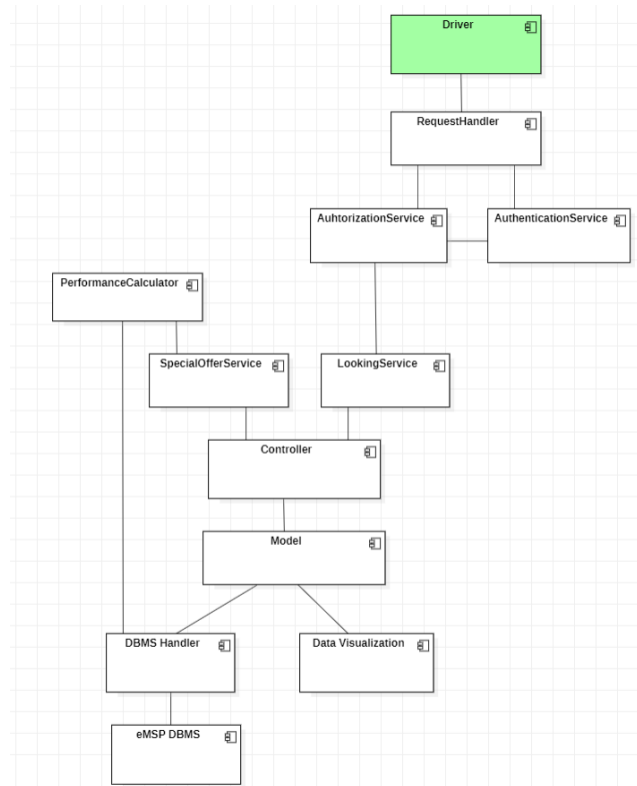


Figure 29: Integration test of RequestHandler

5.2.10 ClientApplication

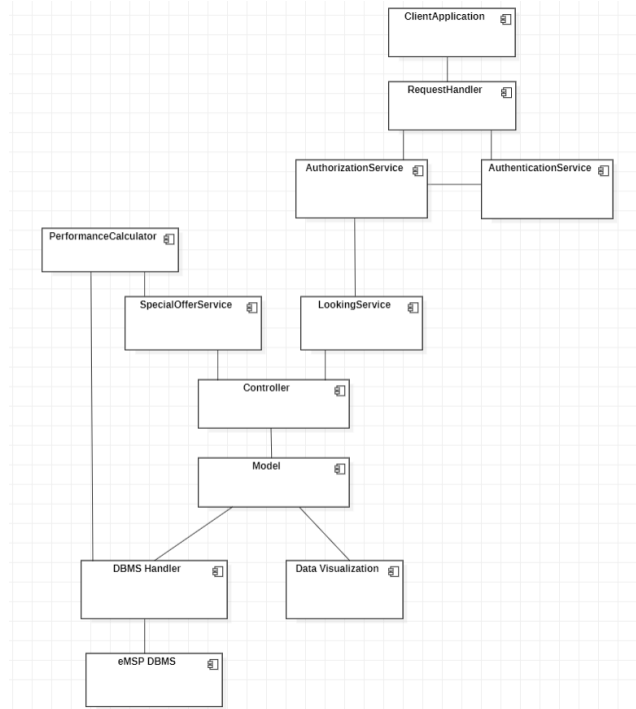


Figure 30: Integration test of ClientApplication

Final integration of the ClientApplication component which concludes the Testing and Integration Plan.

6 Effort Spent

6.1

Task	Time spent
Introduction	4 h
Architectural design	14 h
User Interface Design	4 h
Requirements Traceability	3 h
Implementation, Integration and Test Plan	5 h

6.2

Task	Time spent
Introduction	5 h
Architectural design	13 h
User Interface Design	9 h
Requirements Traceability	1 h
Implementation, Integration and Test Plan	2 h

6.3

Task	Time spent
Introduction	6 h
Architectural design	15 h
User Interface Design	4 h
Requirements Traceability	1 h
Implementation, Integration and Test Plan	4 h

7 Bibliography

References

- [1] *Enel X Website*, <https://www.enelx.com/n-a/en.html>
- [2] *Ionity Website*, <https://ionity.eu/>
- [3] *Three Tier Architecture Definition*, <https://www.ibm.com/cloud/learn/three-tier-architecture>, IBM
- [4] *Data Warehouse Definition*, <https://www.oracle.com/it/database/what-is-a-data-warehouse/>, Oracle
- [5] *Key Differences between Bare Metal and VM Servers*, <https://phoenixnap.com/blog/bare-metal-vs-vm>
- [6] Anshul Saxena *Everything You Need to Know About In-Vehicle Infotainment Systems*, <https://www.einfochips.com/blog/everything-you-need-to-know-about-in-vehicle-infotainment-system/>
- [7] *Architecture and Scalability Definition* in NGINX Documentation, https://nginx.org/en/#architecture_and_scalability
- [8] *FirePower 1000 Series DataSheet*, <https://www.cisco.com/c/en/us/products/collateral/security/firepower-1000-series/datasheet-c78-742469.html>, Cisco
- [9] *JSON Web Tokens*, <https://jwt.io/introduction>
- [10] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Abstraction and Reuse of Object-Oriented Design*, https://books.google.it/books/about/Design_Patterns.html?id=6oHuKQe3TjQC&redir_esc=y