

eMall: e-Mobility for All

Nicolas Benatti (10668784)
Lorenzo Biasiolo (10629367)
Melissande Simonin (10918410)

Design Document (DD)
Version 1.0, 8th January 2023



POLITECNICO
MILANO 1863

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Scope	3
1.3	Definitions, Acronyms, Abbreviations	3
1.3.1	Definitions	3
1.3.2	Acronyms	4
1.3.3	Abbreviations	4
1.4	Revision history	4
1.5	Reference documents	4
1.6	Document structure	4
2	Architectural Design	6
2.1	Overview: high level components and their interaction	6
2.1.1	High-Level view of the system	7
2.2	Component view	7
2.3	Deployment view	9
2.4	Runtime view	11
2.4.1	Login	11
2.4.2	User registering	12
2.4.3	Request with authorization	13
2.4.4	Book a charge	14
2.4.5	Get information about nearby stations	15
2.4.6	User pays for a charge	15
2.4.7	Notify when a charge ends	16
2.4.8	Provide internal status of a CS	16
2.4.9	Suggest the user when to charge	17
2.4.10	Charge the vehicle	18
2.4.11	Provide vehicle status while charging	18
2.5	Component interfaces	19
2.6	Selected architectural styles and patterns	19
2.7	Other design decisions	20
3	User Interface design	21
3.1	Login	21
3.2	Sign up	22
3.3	Dashboard	23
3.4	List of bookings	24
3.5	List of nearby stations	25
3.6	Map	26
3.7	Charging station details	27
3.8	Booking process	28
3.9	Charging process	29
3.10	Payment	30
3.11	Profile	31

3.12	Operator	32
4	Requirements traceability	33
5	Implementation, integration and test plan	36
5.1	Implementation	36
5.2	Integration & Test plan	36
5.2.1	Model	37
5.2.2	Service	37
5.2.3	ClientApplication	38
6	Effort spent	39
6.1	Lorenzo	39
6.2	Melissande	39
6.3	Nicolas	39
7	References	40

1 Introduction

1.1 Purpose

This document is intended to guide the development process for *eMall*: an e-mobility platform which allows users to plan charges for electric vehicles in order not to interfere with their daily schedule. Smart charging stations allow to automatically decide from which provider to fetch energy, based on the selling price. Furthermore, the user can be notified when it's time for a charge and be driven to the best station thanks to apposite algorithms.

This document contains a description of the architectural design for the system, including the components involved and how they interact. Additionally mockups of the user interface are presented and a plan for the implementation, testing and integration of the system. All in all, this document should guide the development of the system.

1.2 Scope

Among the various stakeholders involved in the EV charging network, This document considers EV drivers, CPOs and DSOs. We can assume that the system has been designed to target a national customer base, and specifically, only the one of this specific eMSP. However, please note that the Italian EV charging network is not entirely owned by one company: many CPOs are present on the market. Thus, integration mechanisms are strongly leveraged to allow one eMSP to communciate with the entire grid of charging stations.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Identifier	Description
Charging Station	Set of electrical sockets deployed on the same site and managed by the same CPO
Charging slot	A Single electrical socket which is part of the charging station
Reservation	the act of booking a charge in a specific charging Station for a certain timeframe
Energy Source	Can be either DSO-supplied electricity or local batteries inside the charging station

1.3.2 Acronyms

Acronym	Description
API	Application Programming Interface
CPMS	Charge Point Managemenet System
CPO	Charging Point Operators
DSO	Distribution System Operator
eMSP	e-Mobility Service Provider

1.3.3 Abbreviations

Abbreviation	Description
RASD	Requirements Analysis and Specification Document
DD	Design Document
WP	World Phenomena
SP	Shared Phenomena
G_n	Goal no. n
D_n	Domain Assumption no. n
eMall	e-Mobility for All
EV	Electric Vehicle
CS	Charging Station

1.4 Revision history

1.5 Reference documents

- The specification document "Assignment RDD AY 2022-2023_v2.pdf".
- RASD

1.6 Document structure

This document contains 7 sections, detailed below.

- In the 1st section background to the problem is given, along with the purpose of this specific report. Additionally, some necessary information in order to read the report is given, such as definitions and abbreviations.
- The 2nd section focuses on describing and motivating the architectural design of the system to be. It starts with a high-level overview of the architecture and then breaks each part down into components. The components are described and their interdependence are shown in the component diagram. Moreover, the section contains a component inter-face diagram, a deployment view and sequence diagrams describing the interactions between components.
- In the 3rd section design mockups of the user interface are presented.

- The 4th section contain the requirement traceability matrix, where each of the components described in section two is mapped to the requirements specified in the RASD. The mapping is based on whether the component contributes to the fulfilment of the requirement.
- Section 5 describes the suggested implementation and test plan for the system to be.
- Section 6 contains the effort spent on this report by the authors.
- Section 7 contains the references used.

2 Architectural Design

2.1 Overview: high level components and their interaction

eMall is a distributed web application which mainly consists of a mobile client and a distributed CPMS in which nodes are dislocated among the various stations. To guarantee a good degree of decoupling and flexibility among components, we decided to make the system to follow a 3-tier architecture. This choice allows to increment the independence between client and server side, but also, thanks to the application server, to detach the data sources (Databases, Data Warehouses) from the logic of the application, thus allowing to reuse the data by connecting it to other application servers too.

A short description of every layer follows:

- **Presentation:** The presentation tier handles the interaction with the user, by communicating with the business tier and presenting the information retrieved (through APIs) This layer will be entirely contained in the client application).
- **Business logic:** This tier contains all of the application's business logic, such as deciding from which DSO to get energy from, whether to use internal batteries, and so on. Being the middle layer, it also handles the moving, and translation, of data back and forth between data and presentation tier. It's also capable to act as a proxy (if needed), e.g. to hide location of the servers or cache client requests. Moreover, the business tier is responsible for the communication with external services. In our case this would mean APIs of other CPOs, DSOs as well as payment gateways.
- **Data:** The data tier contains and handles the persistent data of the system. Various data sources can be used, such as "standard" relational DBs, as well as Data warehouses which enable various kinds of analytics on data.

2.1.1 High-Level view of the system

Here we illustrate the multi-tier nature of the application on a very high level of abstraction.

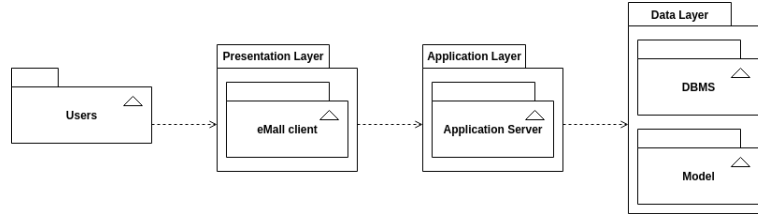


Figure 1: UML *model diagram* illustrating the 3-tier architecture of eMail

2.2 Component view

In this section the component view is presented via a component diagram and corresponding descriptions. The diagram complies with the 3-tiered architecture previously described. Viewed left to right, starting with the presentation tier (contained ClientApplication), passing through the business logic (contained in the ApplicationServer), and the data tier, here represented by the DBMS component.

The following component diagram gives a specific view of the system focusing on the representation of the internal structure of the application server, showing how its components interact. The application server contains the business logic of our software. Other elements in the diagram have been depicted in a simpler way just to show how the communication is structured among these components and the application server.

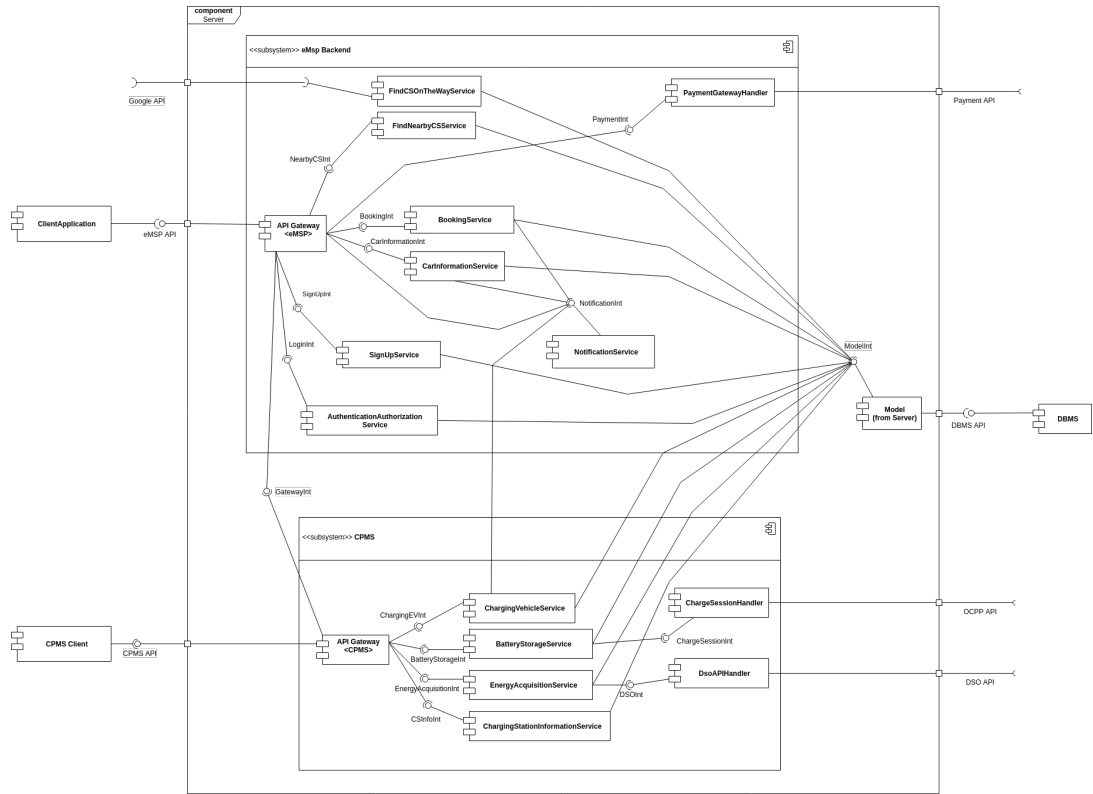


Figure 2: UML *Component Diagram* illustrating the 3-tier architecture of eMall

A short description of the components follows:

- **ClientApplication:** Client smartphone application.
- **CPMS Client:** Web interface through which station operators can access the CPMS, in order to perform the operations stated in the RASD.
- **API Gateway:** Single entry-point in charge of routing and load balancing the requests (see 2.3)
- **FindNearbyCSService:** Given the current position of the user, retrieves the closest stations.
- **FindCSOnTheWayService:** Finds stations along the route the user is following. This is possible thanks to the interaction with the phone nav system, as stated in RASD.
- **Authentication&AuthorizationService:** Component in charge of handling user access as well as granting access permissions.

- **BookingService:** This component is in charge of handling charge bookings coming from users.
- **CarInformationService:** This component is in charge of fetching real-time information about vehicles.
- **NotificationService:** This component sends push notifications to the eMall client app.
- **ChargingStationInformationService:** Retrieves internal and external status of a station.
- **EnergyAcquisitionService:** This component communicates with the DSO (via the *DSOAPIHandler*) in order to acquire energy.
- **BatteryStorageService:** Handles energy storage in internal batteries of stations.
- **ChargingVehicleService:** This component communicates with the *ChargeSessionHandler* in order to start/stop the charging session and to supply real-time statistics to the client app.
- **ChargeSessionHandler:** Communicates with the charging slot, managing the entire lifecycle of a charging process as well as providing real-time infos. Communication happens via OCPP protocol, as stated in RASD.
- **DSOAPIHandler:** Communicates with the DSOs of the providers present on the EV network.
- **PaymentGatewayHandler:** Integrates the system with the most diffused payment services.

2.3 Deployment view

This section presents a deployment diagram of the system. It specifies the environments and tools to be used to build the system, coupled with the protocols needed to communicate between the various parts.

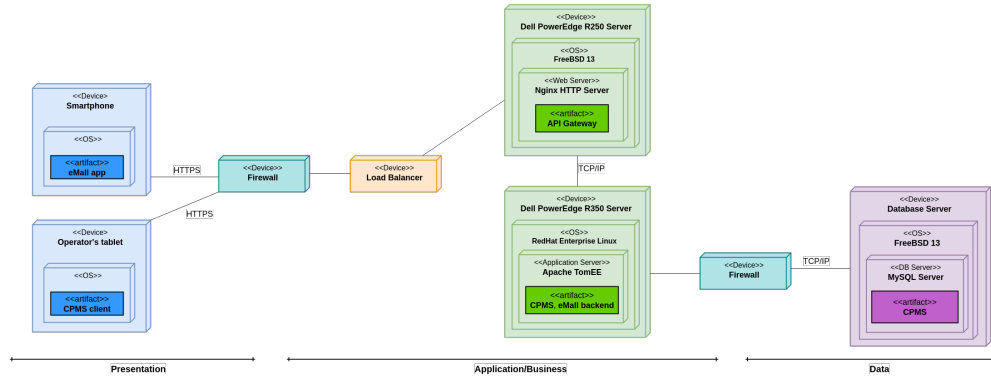


Figure 3: UML *Deployment diagram*

- **Smartphone:** eMail is mainly thought to be used from a smartphone by *on-the-move* users. Hence, a web interface won't be available. Thus, the eMail website would just redirect to the app store.
- **Operator's tablet:** Station operators can access the CPMS web interface via tablets connected to the backend via VPN, for better security.
- **Web server:** Hosts the web tier of the application (e.g. the servlet container in the JEE stack) and the API Gateway (could be the webserver, such as NGINX), replies to HTTP(S) requests by serving static and dynamic content. NGINX has been preferred over Apache due to its inherently even-based architectures which guarantees high performance even under heavy load.
- **Application server:** Hosts the application logic (in JEE, these are classes mapping to the DB entities and *Enterprise Java Beans*).
- **Firewall:** Analyzes the incoming traffic and filters it according to rules, thus blocking unwanted packets and guaranteeing an acceptable security. (for the rest of the security policies, see the RASD) Stateful firewalls are highly preferred over stateless ones.
- **Load balancer:** Balances the incoming traffic redirecting it to different webserver in order to avoid overloading them.
- **Database server:** Hosts the DBMS.

Application and web servers, as well as DB server, should be replicated in order to increase the availability of the system.

2.4 Runtime view

In this section sequence diagrams describing the way in which components of the system interact for the main functionalities are presented. Since some functionalities are similar, or contained in one another, only one is presented and the rest briefly described. Also, login/registration/authorization are assumed to take place before the described flow, and therefore omitted.

2.4.1 Login

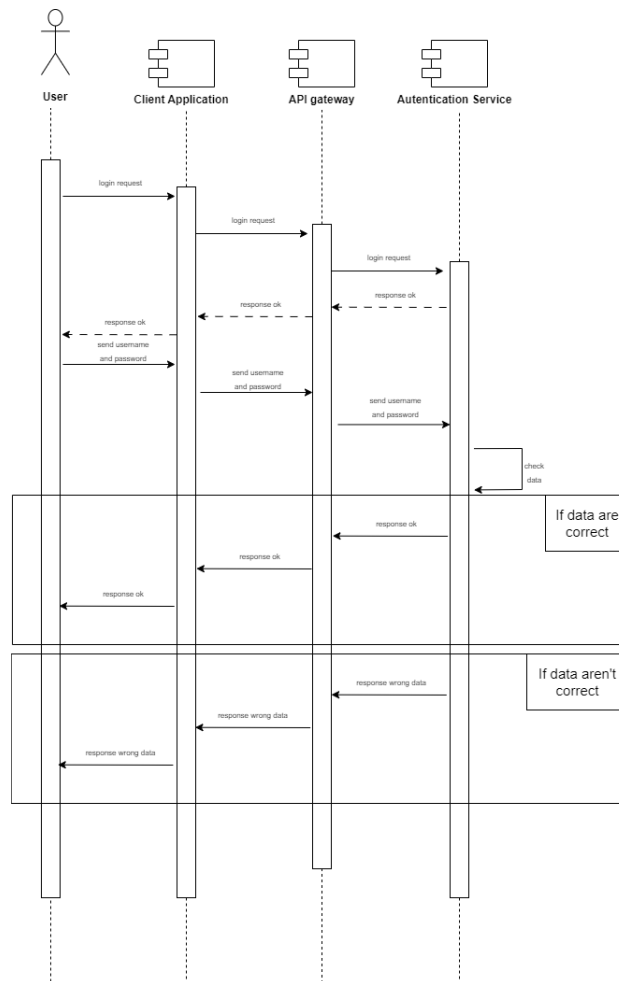


Figure 4: Login sequence

2.4.2 User registering

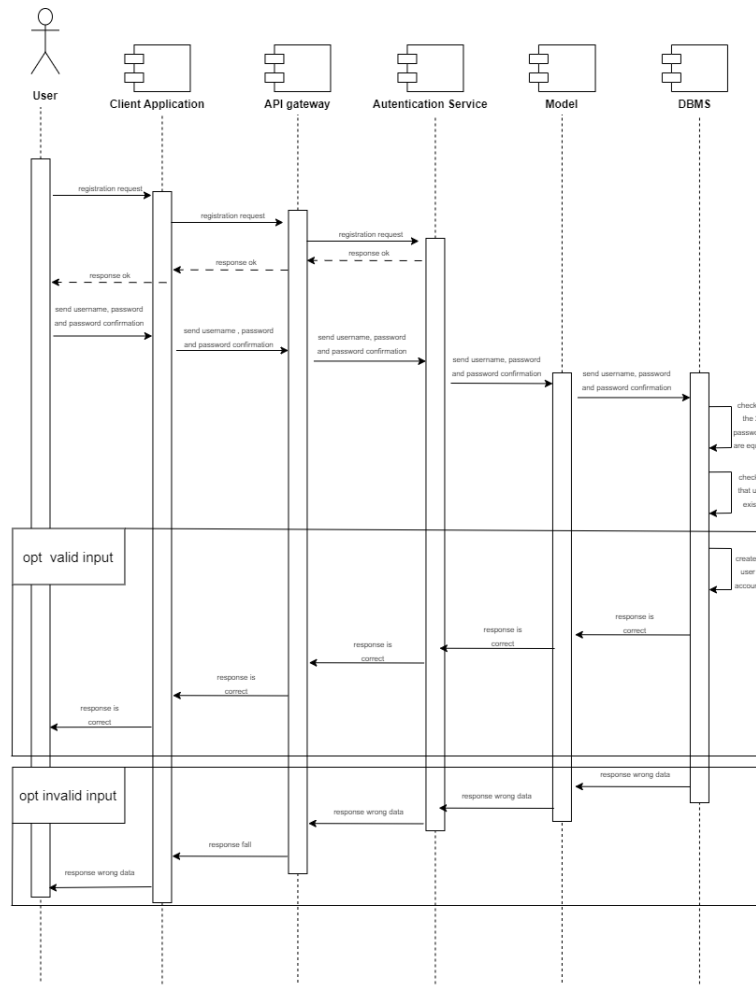


Figure 5: Sign-up sequence

2.4.3 Request with authorization

This is meant to be a generic authorized request to the backend.

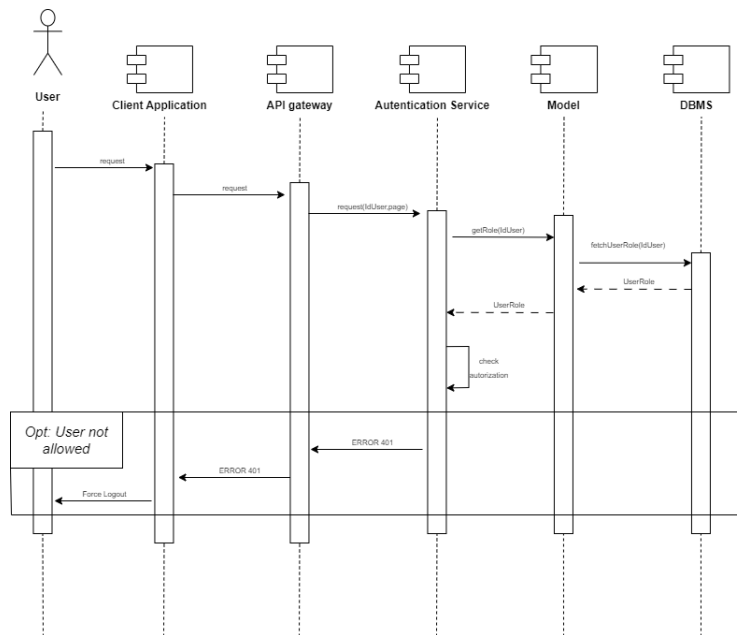


Figure 6: request sequence

2.4.4 Book a charge

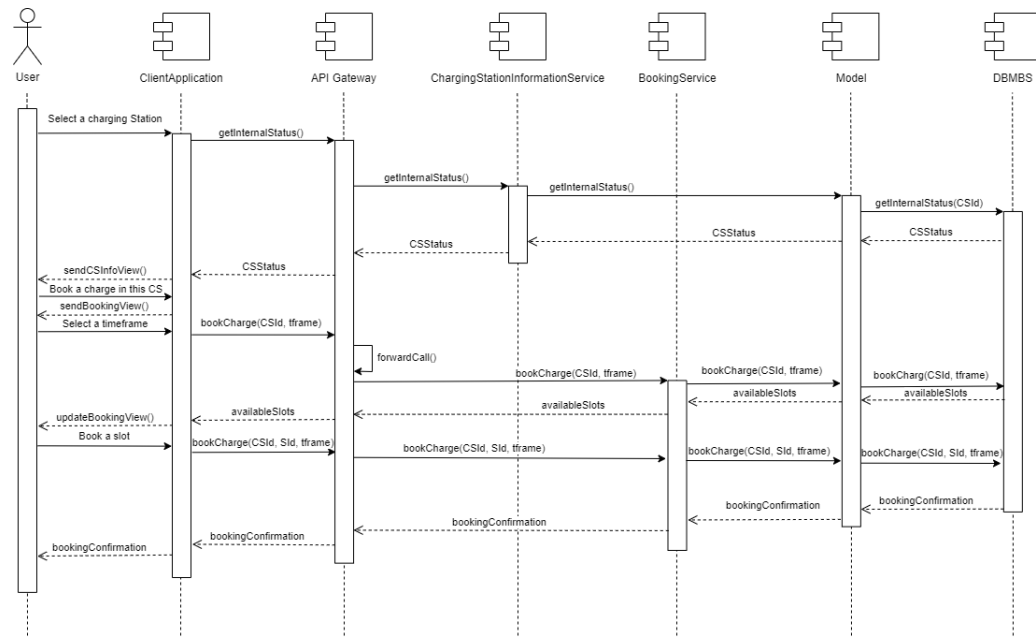


Figure 7: Booking a charge sequence

2.4.5 Get information about nearby stations

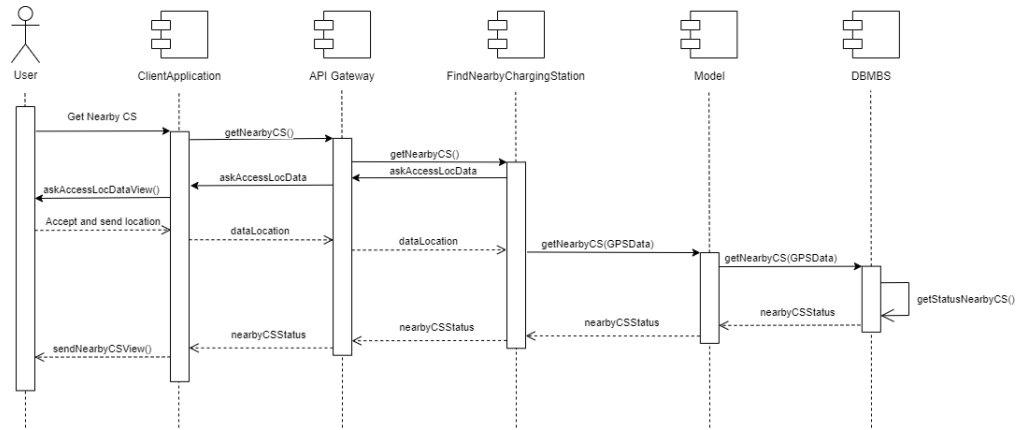


Figure 8: Get information about nearby charging station sequence

2.4.6 User pays for a charge

The whole interaction is seen as synchronous because the user cannot do anything else while waiting for the payment to be approved.

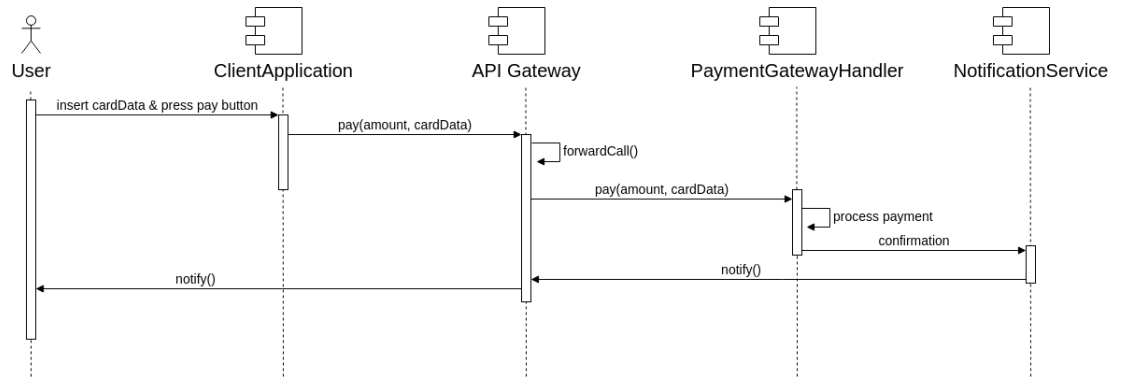


Figure 9: Payment sequence

2.4.7 Notify when a charge ends

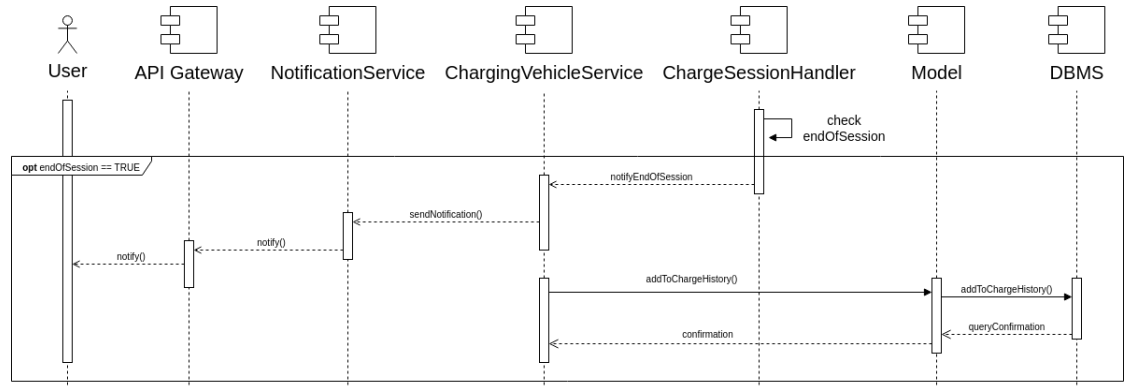


Figure 10: *End-Of-Charge* notification sequence

2.4.8 Provide internal status of a CS

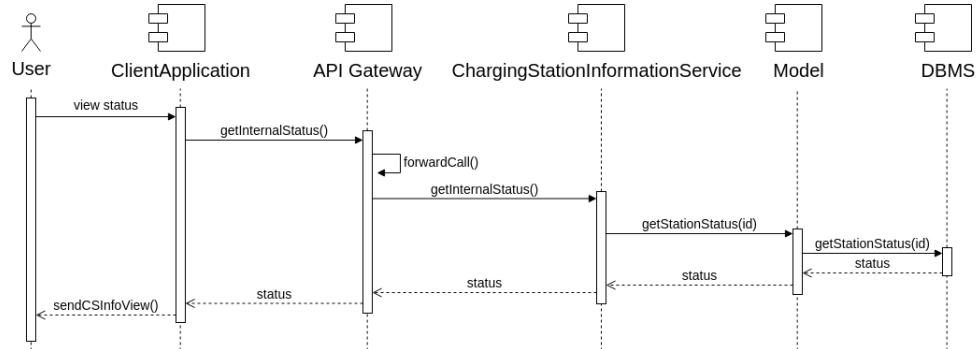


Figure 11: Get *internal status* of a station sequence

2.4.10 Charge the vehicle

Please note that the act of starting a new charging session is modeled as synchronous, although surely performed with an asynchronous technology (such as AJAX calls) because, from a conceptual standpoint, the system (especially the client) should not be able to perform any other action between the activation request and the actual activation of the charging session.

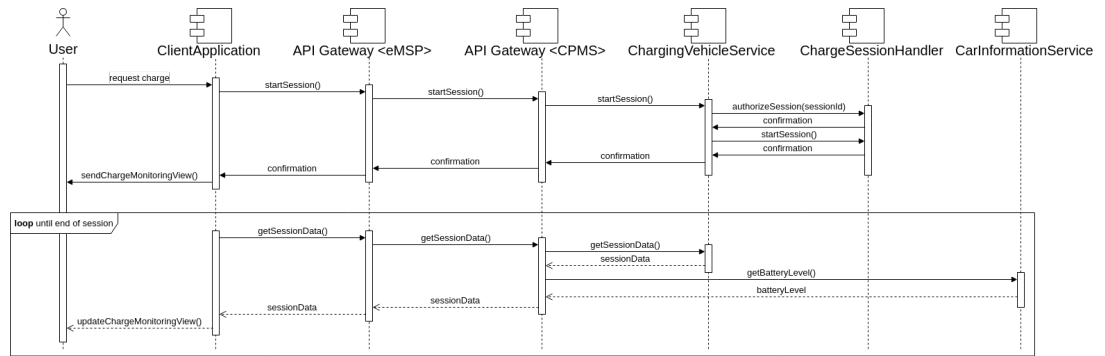


Figure 13: Vehicle charging sequence

2.4.11 Provide vehicle status while charging

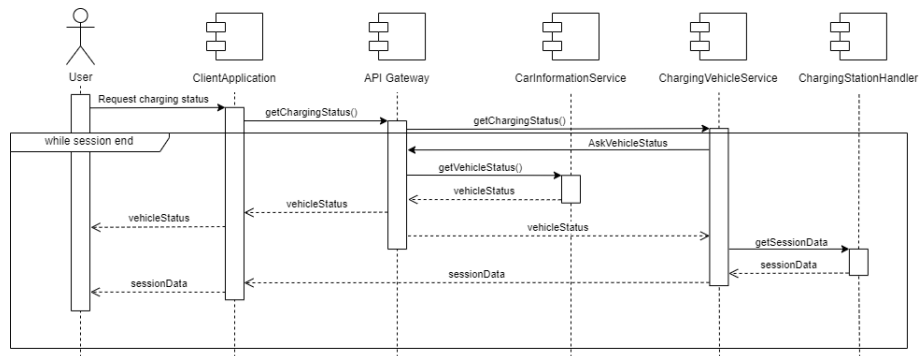


Figure 14: Vehicle status while charging sequence

2.5 Component interfaces

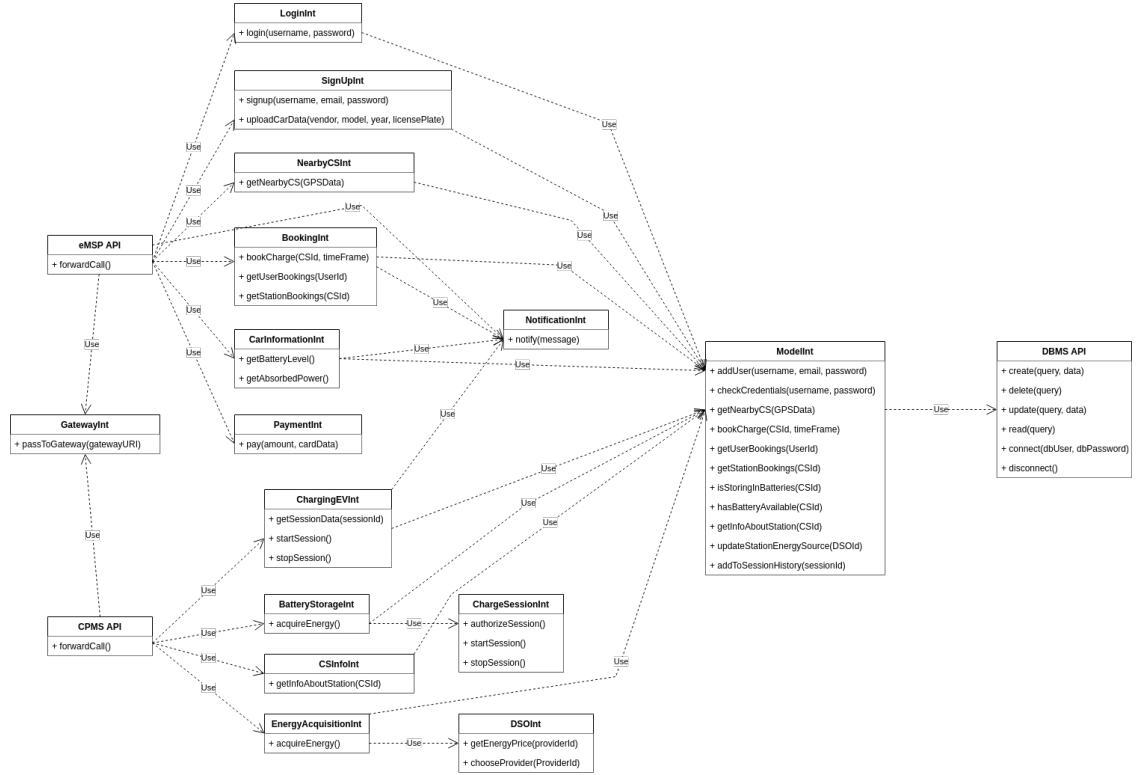


Figure 15: Vehicle status while charging sequence

2.6 Selected architectural styles and patterns

MVC: In order to guarantee a high degree of conceptual separation between the various components, we decided to build eMall following the *Model-View-Controller* pattern. This separates components into:

- **Model:** Manages backend data and business logic
- **View:** User Interface of the application (presentation layer)
- **Controller:** These components receive input from the View, convert them into commands and query the Model in order to perform the operation.

This conceptual separation brings several benefits such as:

- Easiness to implement personalized views (e.g. depending on the device type or user preferences), since this is completely independent from the other aspects of the system.
- The model returns data in no specific format, so it's very easy to manipulate it and produce reports or visualizations (e.g. a monthly report of the user activity)
- Modification on one side don't impact the other ones (e.g. modifying a controller won't change the model) as long as interfaces between them remain stable.

3-Tier architecture: The system is structured in 3 tiers: a presentation tier, business logic tier and a data tier.

As stated in 2.1, the reason for choosing this style instead of a "standard" client-service architecture is mainly to increase the reusability of the data tier.

In the case the data acquired over time will be useful for other services, and thus it makes sense to decouple the data and business tier at this stage. Moreover, it facilitates scalability by allowing one tier to be replaced or given more resources without impacting the other two.

2.7 Other design decisions

From 2.3 it may have been noted that the web and application server are deployed on different machines, this brings advantages such as:

- Application servers don't have to care about handling requests, load balancing and routing the incoming traffic
- DoS attacks will impact only the web servers without touching the application servers. Furthermore, firewall can be used to separate the 2 machines and increment security.
- Updates to the business logic can be performed without rebooting the web servers.

Furthermore, every tier is separated with a firewall, which allows to isolate the application layer and data layer between them, as well as isolating the entire system from the Internet. As stated in the RASD, this practice, together with the use of an IDS/IPS, will increase security a lot.

Lastly, integration between our CPMS and the CPMS of other CPOs must be leveraged (as stated in the Introduction paragraph) this is possible through the *e-Roaming* interface, which acts as a registrar in which the CPOs can register and signal its to the EV network. In order to accomplish this, the e-Roaming architecture may be implemented following an event-based paradigm, for example using a distributed *Publish-Subscribe* system.

3 User Interface design

In this section we illustrate what could the final User Interface of eMall be, via some mockups.

3.1 Login

The login interface is the interface the user is faced with when launching the application.

On it, he enters his login informations (email address and password) in order to have access to his data and to the functionalities of the application.

He can, from this one, create an account if he does not already have one, via the "Sign up" button.

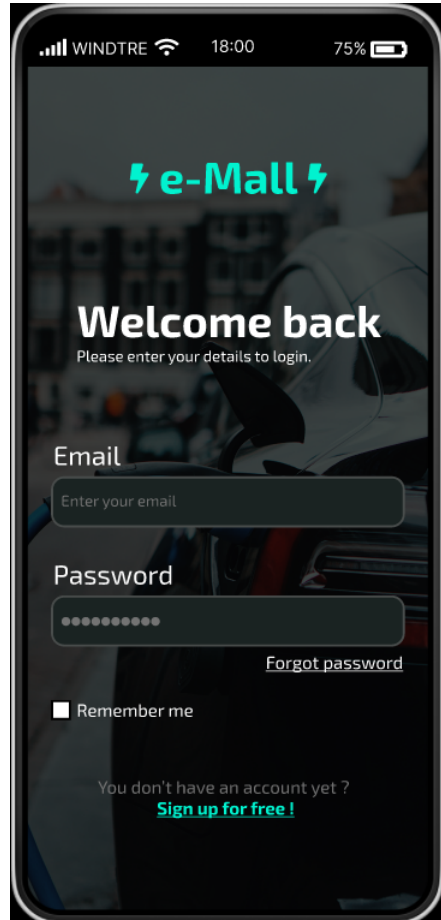


Figure 16: Login UI of eMall

3.2 Sign up

The sign up interface allows non registered users to create an account, and so access the mobile app and its functionalities. The user needs to fill in his name, his car brand and model, his license plate, and an email and a password in order to log in. The phone number is optionnal.

The user must agree to terms and Conditions too. After clicking on the "Sign up" button, the user is registered, and he can connect through the login interface.

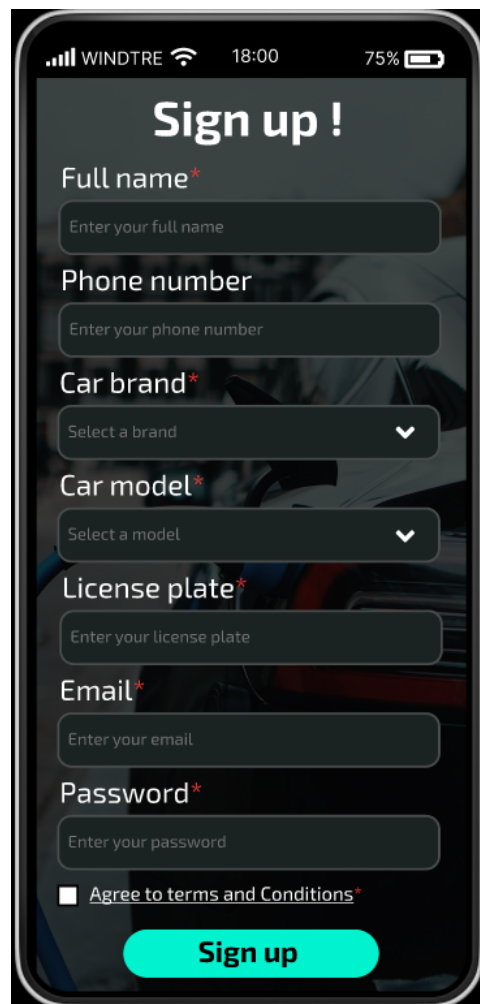
The image shows a mobile application interface for signing up. At the top, the status bar displays 'WINDTRE', signal strength, Wi-Fi, time '18:00', and battery '75%'. The app title 'Sign up !' is centered in white. Below it are seven input fields, each with a red asterisk indicating a required field: 'Full name*' (text input), 'Phone number' (text input), 'Car brand*' (dropdown menu), 'Car model*' (dropdown menu), 'License plate*' (text input), 'Email*' (text input), and 'Password*' (text input). Each field has a placeholder text. At the bottom, there is a checkbox labeled 'Agree to terms and Conditions*' and a large red 'Sign up' button.

Figure 17: Sign up UI of eMall

3.3 Dashboard

The "Dashboard" interface is the one that appears first after the login. It is the main one, accessible through the bottom menu, by clicking on the "home" icon. It summarizes the main functions of the application :

- Information about the car : model, amount of battery, and button to make sure the car is charging, as well as the remaining charging time
- The most used charging station, in order to access them and book a slot quickly
- The special offers available near user location

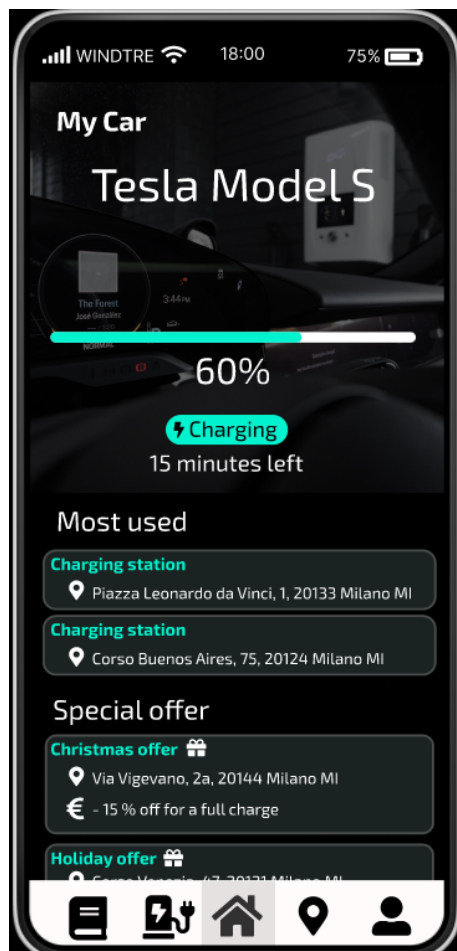


Figure 18: Dashboard UI of eMall

3.4 List of bookings

The "list of bookings" interface allows the user to see his previous charges, and the next ones. For each single booking, he can see the address of the charging station, the slot number he has been assigned, the type of charging slot, and the schedule. He can access it through the bottom menu, by clicking on the "Book" icon.

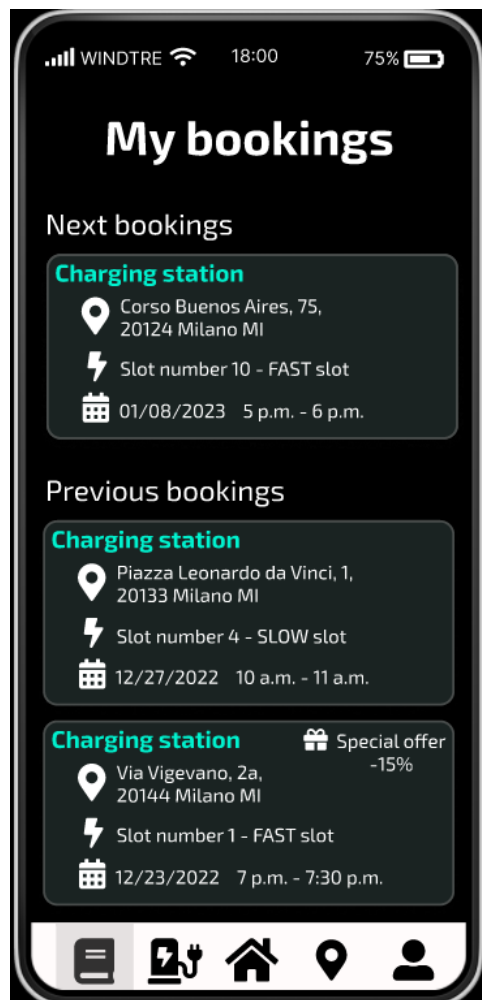


Figure 19: "List of bookings" UI of eMall

3.5 List of nearby stations

The "list of nearby stations" interface allows the user to see the main details of charging stations close to his location (Address, energy price, availability, special offer, ...). He can access it through the bottom menu, by clicking on the "Charging station" icon.

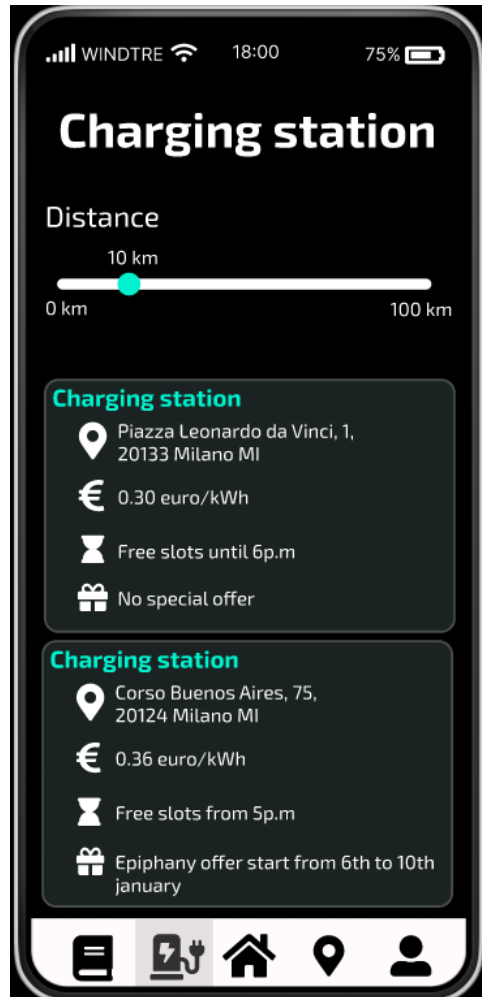


Figure 20: "List of nearby stations" UI of eMall

3.6 Map

The "Map" interface allows the user to find the closest charging stations on his way. He can choose a destination, and the path will be displayed to him, as well as the charging stations closest to it. This interface is quite similar to the "List of charging stations" one, but is more intuitive and graphical. However there are less informations about charging stations at the first sight. The user have to click on them to see the whole details, provided by the "Charging station details" interface. He can access it through the bottom menu, by clicking on the "drop point" icon.

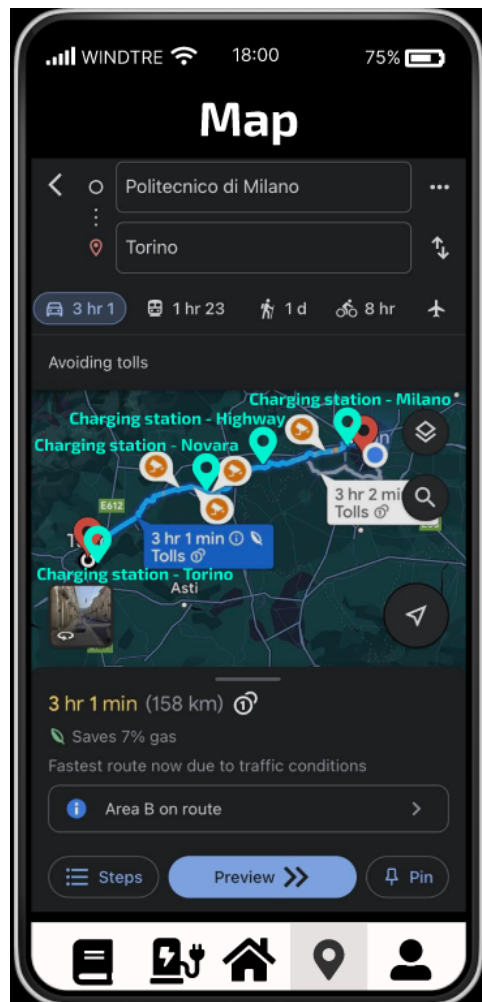


Figure 21: Map UI of eMall

3.7 Charging station details

The "Charging station details" interface allows the user to see all the information about a charging station (address, number of slots, type of slots, availability, energy price, special offer). Moreover, he can book a charge for a precise time slot by clicking on the "Book" button. This interface is accessible by clicking on a single charging station, through the "map" interface or through the "List of nearby stations" interface.

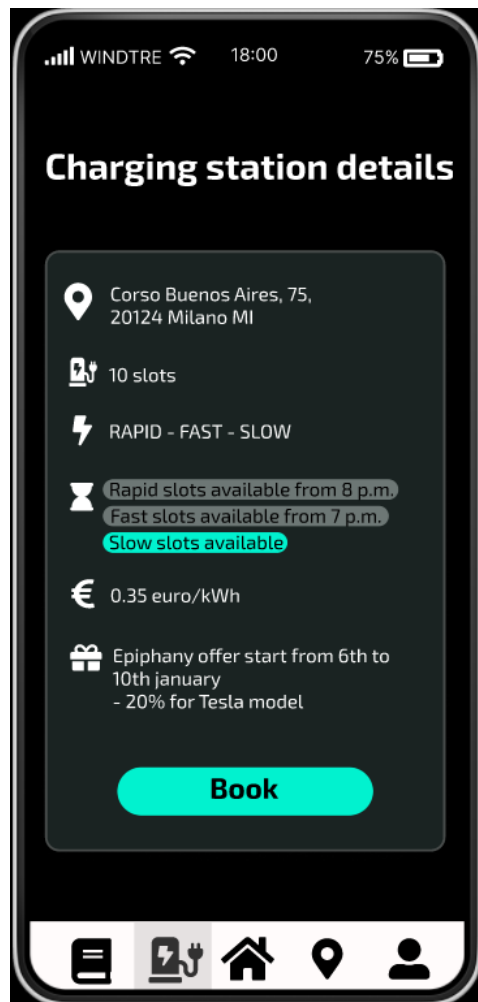


Figure 22: "Charging station details" UI of eMall

3.8 Booking process

The "Booking process" interface allows the user to book a certain type of charge (rapid, fast, slow) in a specific station, at a certain timeframe. This interface is accessible by clicking on the "Book" button from the "Charging station details" interface.

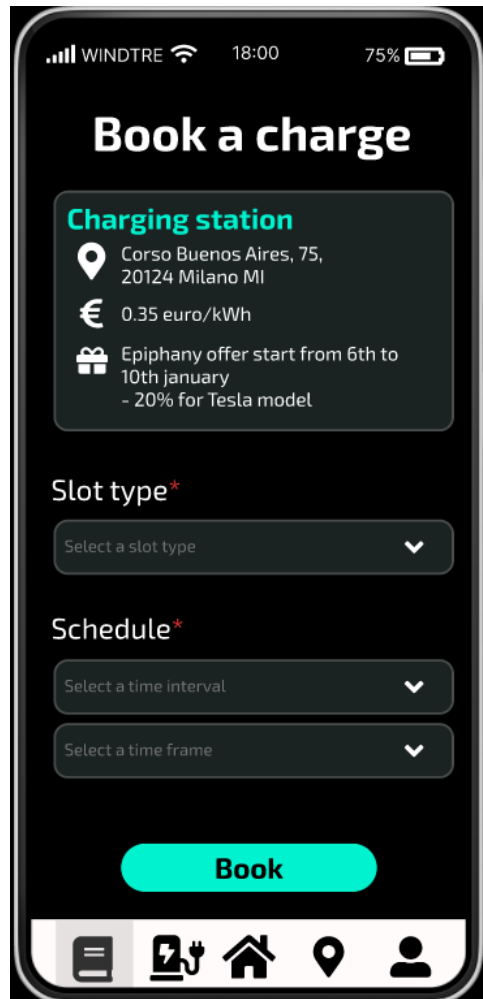


Figure 23: "Booking process" UI of eMall

3.9 Charging process

The "Charging process" interface can only be displayed when the car is plugged to the right charging slot. It allows the user to monitor the charging process, and have information about the current level of battery, the charging slot, the estimated end time of the charge, and an icon indicated that the car is charging. It also include a "Payment" button in order to pay for the service. This interface is automatically accessible when the car is plugged, but the user can quit this interface to navigate through the rest of the application. So it can also be accessible by clicking on the correspondent booking in the "List of Bookings" interface, and from the dashboard by clicking on the "Charging" button.

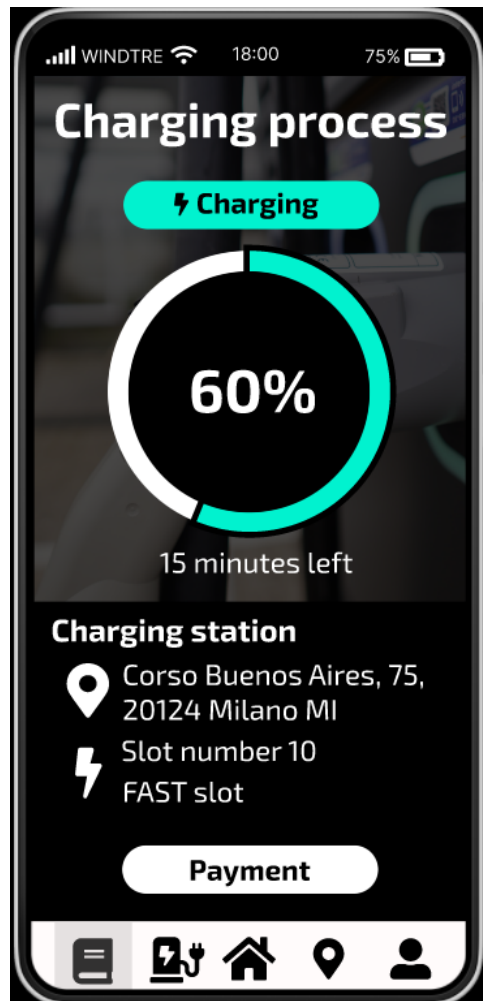


Figure 24: "Charging process" UI of eMall

3.10 Payment

The "Payment" interface is accessible as soon as the charging process has begun, through the "Payment" button in the "Charging process" interface. It allows the user to choose a payment method, and to perform it. Once the payment is done, the button disappears and the user can unplug his car as soon as the charging process is done.

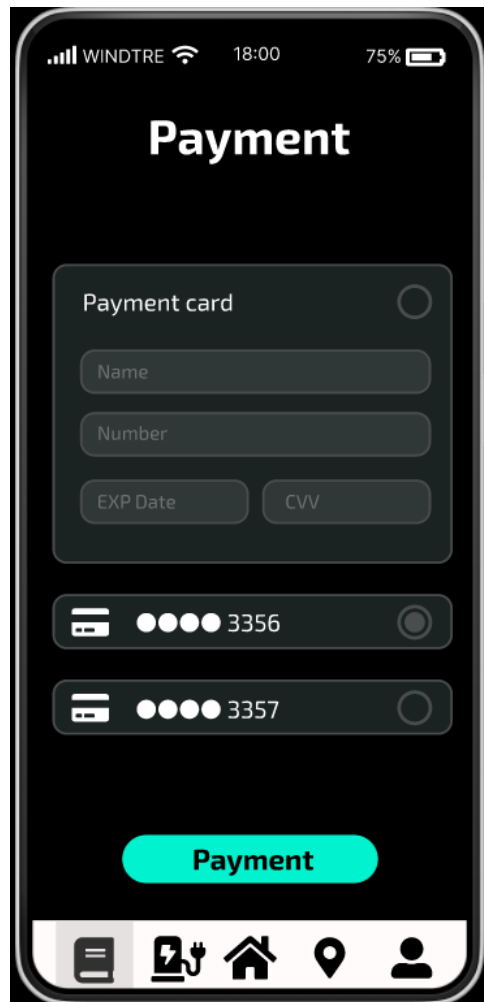


Figure 25: Payment UI of eMall

3.11 Profile

The "Profile" interface allows the user to see and modify his personal informations and manage his notifications. It also allows him to connect his car or to accept the sharing of his schedule and location data.

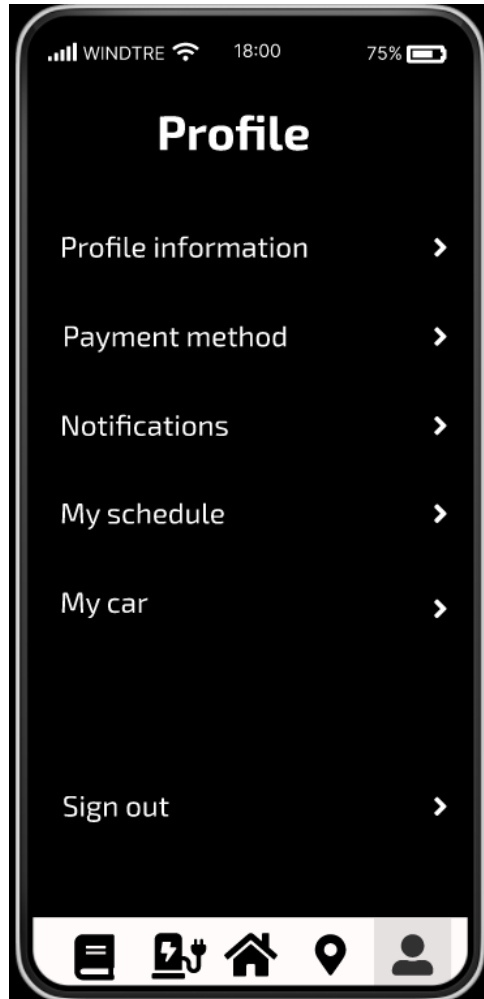


Figure 26: Profile UI of eMall

3.12 Operator

The "Operator" interface allows the CPO operators to manage the storage of the different charging stations, and to manage the special offers for the same list of charging stations. This interface is accessible from the CPMS, from phone, tablet or computer.

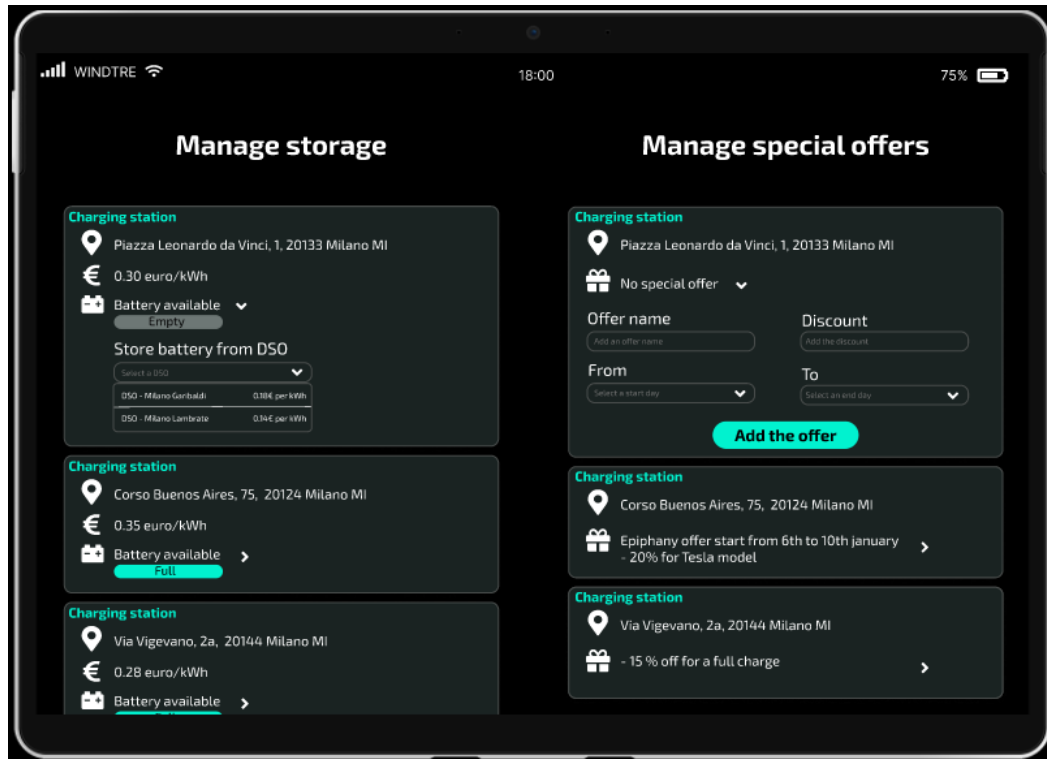


Figure 27: Operator UI of eMall

4 Requirements traceability

In this section we map requirements (initially exposed in the RASD) and architecture components which are needed to satisfy them. For the sake of clarity (and *Self-Containability* of this document) we report the requirements again.

Identifier	Description
R1	The system must allow an unregistered user to register by providing his personal information (e-mail, first name, last name)
R2	The system must allow a registered user to log in using the credentials input at registration time
R3	The system must allow a registered user to connect his car
R4	The system must ask the user for permission to access his location data
R5	The system must allow the user to retrieve external information on charging stations within a certain distance from their current location, such as: location, cost, special offer, number of charging sockets available, their type, and if they are all occupied, the amount of time until the first socket of each type is freed
R6	The system must ask the user for permission to access his schedule on phone's calendar
R7	The system must allow a registered user to choose a charging station and book a charging slot according to his schedule
R8	The system must allow a registered user to connect his car to the charging slot booked and start the charge
R9	The system must allow a registered user to connect his car to a free charging slot even if they didn't book it, and start the charge
R10	The system must be able to retrieve internal information about charging stations (amount of energy available in their batteries (if any), number of vehicles currently charged, and the amount of power absorbed and the time left to the end of the charge of each plugged vehicle)
R11	The system must be able to deliver energy to the connected vehicle as soon as it is plugged-in
R12	The system must be able to stop the charge when the connected vehicle battery is full

R13	The system must notify the user when the charging process is finished
R14	The system must allow a user to pay for the charge
R15	The system must be able to signal the user in the event of an exception
R16	The system must be able to retrieve in real time the battery level of the user's vehicle
R17	The system must be able to proactively suggest the user to charge their vehicle depending on his schedule, battery conditions and available stations
R18	The system must be able to get information from DSOs (current price of energy) if available
R19	The system must be able to accumulate energy in charging station's batteries (if any) when the price is adequately low
R20	The system must be able to choose the cheapest way of acquiring energy during charge (from charging station battery or DSO), according to availability and cost
R21	The system must be able to access phone's navigation system

To save up space, names of components are going to be shortened. A table explaining such abbreviations follows:

- **CA:** ClientApplication
- **CC:** CPMS Client
- **GW:** API Gateway
- **FN:** FindNearbyCSService
- **FW:** FindCSOnTheWayService
- **AA:** Authentication&AuthorizationService
- **BS:** BookingService
- **CI:** CarInformationService
- **NS:** NotificationService
- **SI:** ChargingStationInformationService
- **EA:** EnergyAcquisitionService
- **BT:** BatteryStorageService

- **CV:** ChargingVehicleService
- **CS:** ChargeSessionHandler
- **DS:** DSOAPIHandler
- **DB:** DBMS
- **M:** Model

R	CA	CC	GW	FN	FW	AA	BS	CI	NS	SI	EA	BT	CV	CS	DS	DB	M
R1	✓		✓			✓										✓	✓
R2	✓		✓			✓										✓	✓
R3	✓		✓			✓		✓								✓	✓
R4	✓		✓			✓			✓								
R5	✓		✓	✓		✓				✓						✓	✓
R6	✓		✓			✓											
R7	✓		✓			✓	✓									✓	✓
R8			✓			✓	✓							✓			
R9			✓			✓								✓			
R10	✓		✓			✓				✓						✓	✓
R11			✓								✓		✓				
R12			✓								✓		✓				
R13	✓		✓			✓			✓				✓	✓		✓	✓
R14	✓		✓			✓											
R15	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
R16	✓		✓			✓		✓					✓				
R17	✓		✓	✓	✓	✓		✓	✓	✓						✓	✓
R18		✓	✓												✓		
R19		✓	✓								✓	✓					
R20		✓	✓												✓		
R21	✓		✓			✓			✓								

5 Implementation, integration and test plan

5.1 Implementation

The implementation must follow a precise scheme: components will be divided in groups based on the component diagram (description follows) and implemented following a Bottom-Up strategy. An Agile Development pattern will be used (e.g. SCRUM). In addition to this, usage of the MVC pattern intrinsically brings to a faster development, because work on, view, model and controller can be performed in parallel.

1. **DBMS & Model:** This involves modeling and creating the DB as well as creating the EJBs mapping to the DB entities.
2. **API Gateway:** Both the Gateways can be implemented and tested in parallel on their respective backends.
3. **Controllers:** All the controllers follow almost the same patterns and can be written in parallel.
4. **ClientApplication:** This involves developing the UI as well as the required code to fetch data from the backend (will be a thin client and therefore won't contain business logic).

5.2 Integration & Test plan

Adoption of the MVC pattern also brings perks while testing,

The sequence in which we integrate components will follow the implementation order described in the previous section.

A mix of Bottom-Up and Top-Down integration strategies is preferable, since it allows to keep an eye on the general view of the system as well as having encapsulated, decoupled and reusable components. All this works really well together with Agile development patterns stated in 5.1.

Incremental testing must be followed: as soon as new components are ready, we should integrate them with the rest of the system and re-test every time. Before testing to integrate a component, two conditions should be fulfilled: the component's interface should offer all functionality specified in the *component interface diagram* and should pass all Unit Tests. By applying regression testing technique, all Unit Tests must be re-run at every component modification, to make sure that it still works as expected after a change.

The order in which to run the integration tests is presented with explanatory schemes (which only show one of the controllers, though the rest of them follow in the same way).

5.2.1 Model

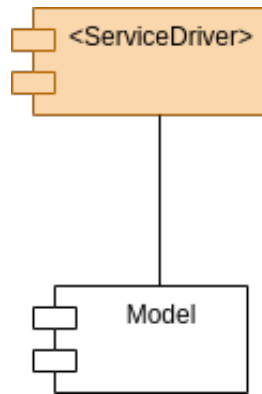


Figure 28: Integration test of the model : Bottom-up strategy

5.2.2 Service

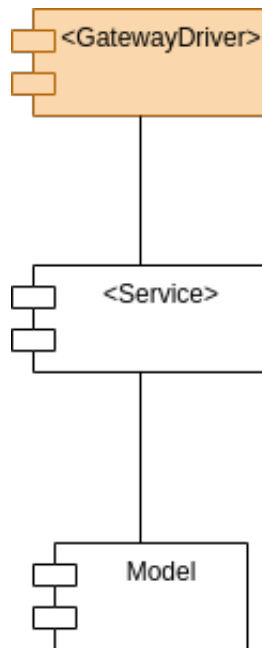


Figure 29: Integration test of the model and the services : Bottom-up strategy

5.2.3 ClientApplication

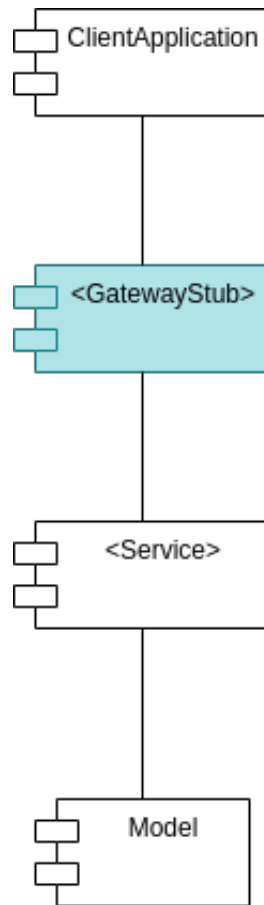


Figure 30: Integration test of the Client App : Top-down strategy

6 Effort spent

6.1 Lorenzo

Task	Time spent
Introduction	1h
Architectural design	10h
User interface design	1h
Requirements traceability	2h
Implementation, Integration and Test Plan	1h
Discussion	3h
Total	18h

6.2 Melissande

Task	Time spent
Introduction	1h
Architectural design	15h
User interface design	5h
Requirements traceability	2h
Implementation, Integration and Test Plan	2h
Discussion	3h
Total	28h

6.3 Nicolas

Task	Time spent
Introduction	1h
Architectural design	15h
User interface design	2h
Requirements traceability	2h
Implementation, Integration and Test Plan	1h
Discussion	3h
Total	24h

7 References

- More information on API Gateways and their benefits was found on NGINX Docs
- This forum thread provides some point on separating deployment of web and application servers.
- Here and here we can find information and examples about the usage of UML Model Diagrams to model multi-tiered applications.
- This article explains various advantages of using API Gateways for the entire organization.
- This article points out several advantages of applying the MVC pattern to the development of web applications.