POLITECNICO DI MILANO

SOFTWARE ENGINEERING 2 - A.Y. 2022-23

PROF. ELISABETTA DI NITTO

eMall – e-Mobility for All

Design Document (DD)

Authors
Claudio ARIONE
Riccardo BEGLIOMINI
Niccolò BINDI

Version: 1.0
January 3, 2023

Contents

1 Introduction

The unique challenges posed by climate change have recently led to a growing push in the adoption of new technologies to reduce carbon emissions. In particular, road transportation accounted for more than 70% of the total transport emissions in the EU, which in turn was responsible for about a quarter of the EU's total CO2 emissions in 2019 (source: shorturl.at/anryE).

Electric vehicles represent a viable solution to tackle this problem, but they require specific infrastructure and knowledge about availability of chargers, cost of energy and distribution.

eMall, a new startup based in Italy, is aiming at improving the experience of charging an electric vehicle. They offer a various range of services that will be able to take care of every aspect of charging, from displaying the location and properties of charging stations nearby and making smart suggestions that take into consideration both economic and logistical factors, to actually owning and managing charging points themselves.

1.1 Purpose

The system should provide the user with information about every charging point location nearby. Each location should present the characteristics of its charging stalls: cost, availability, charging speed, compatibility with the charging port, special offers.

The user through the system should also be able to book a charge at a specific location for a certain timeframe, selecting through the GUI a charging station and inserting a starting time and ending time (to be able to create a real schedule, as to give the other users the possibility to book a consecutive timeframe if they're willing to).

The system should also allow the user to manage the whole charging process: they should be able to start the charging, to monitor its status and to be notified when it is done or when the booked timeframe has finished and make the user pay for the service.

Additionally, the system should provide some smart suggestions to the users, based on their location, their schedule (by having access their calendar), the current offers, the battery status of the vehicle and the availability

of charging stations.

1.2 Scope

eMall system, as stated in the RASD, should interact with many users, so the UI should be easy to use in order to be accessible to the majority of people.

The requirements and goals defined in the RASD regarding eMall lead us to choose a microservice architectural style, to be able to differentiate all the features of the system and to handle them independently, for availability reasons.

The system also exploits the REST paradigm, being the server stateless and using selected and limited request types (GET, POST and PATCH). Other design choices, such as replication and security features have been taken to best suite the system and are explained and well detailed in the following sections.

1.3 Definitions, Acronyms, Abbreviations

- **eMall** – e-Mobility for All
  The system we are analyzing and specifying in this document
- **eMSP** – e-Mobility Service Provider
  A company (like eMall) which offers the user various features regarding the e-Mobility field, from locating a charging station, to starting the charging process and paying for it
- **CPO** – Charging Point Operator
  A company which manages the energy supply and the features of a physical charging station. eMall is a CPO as well.
- **CPMS** – Charging Point Management System
  The backend infrastructure of a CPO, which takes decisions and interfaces with providers of energy
- **DSO** – Distribution System Operator
  A company which produces and provides electric energy to CPOs

- **CP** – Charging Point
  A place where charging sockets are located and where a user can charge their vehicle

- **API** – Application Programming Interface
  An interface which a system offers for other systems (human beings or software) to be able to perform specific operations or retrieve information on it

- **DBMS** - DataBase Management System
  A software system designed to store and organize collections of data

1.4 Revision History

- First release: DD v1.0 (January 3, 2023)

1.5 Reference Documents

- Specification document: "Assignment_RDD_AY_2022-2023_v3"

- Requirements Analysis and Specification Document (RASD)

1.6 Document Structure

The document is organized in seven different sections, which are as well divided into sub sections accordingly to modularity and concept separation.

- **Section 1: Introduction**
  This section provides the context of the problem, the scope, with some anticipation about the chosen architecture. At the end of section 1 we can also find this paragraph, the reference documents and the abbreviations and definitions necessary to understand the technical and non-technical sections.

- **Section 2: Architectural Design**
  This section provides a description of the architectural choices taken to design the system: it analyzes the choices themselves and adds the reasons behind them. To be specific, here we can find several diagrams to specify the different components of the S2B and their interactions, with also the sequence diagrams to detail them even more.

- **Section 3: User Interface Design**
  In this section the design of the user interfaces is analyzed: several mockups are presented and are accompanied by various diagrams to specify the way the user can interact with them and the "paths" to go from one to another

- **Section 4: Requirements Traceability**
  In this section the mapping of the requirements defined in the RASD onto the various design elements is presented

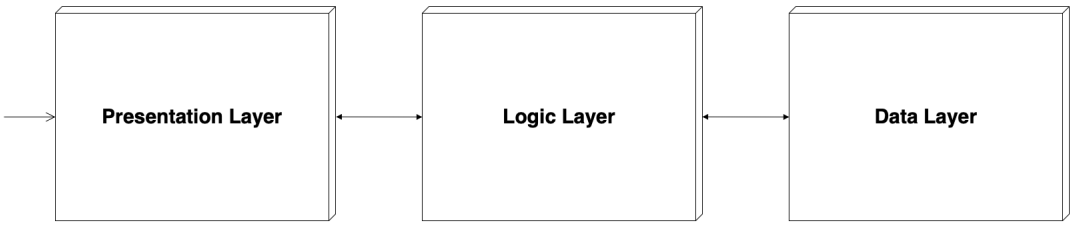
- **Section 5: Implementation, Integration and Test Plan**
  In this section are presented the next steps that will be taken to implement the system, to integrate the various components and then test the complete architecture.

- **Section 6: Effort Spent**
  This section includes a table with the details about the effort spent on each section by each member of the group tackling the project.

- **Section 7: References**
  Here are listed all the reference to documents and web pages used to write this document.

2 Architectural Design

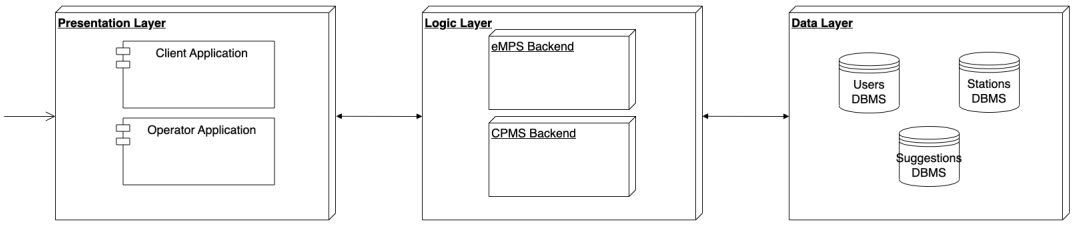
2.1 Overview



The eMall system is designed as a three-tier architecture:

- **Presentation layer**: manages the interaction with the end user. It processes the data coming from the other layers and renders them in a human-friendly way. It also sends back information coming from the user to the rest of the system.
- **Logic layer**: manages and processes the data coming from/to the data layer (explained below) and the presentation layer. It is also responsible to communicate with third-party entities using APIs.
- **Data layer**: provides a safe and persistent storage for data, both from users and entities associated with eMall (charging stations, external CPOs, DSOs).

This design choice is common and suits well the needs of a mobile application, which is the main objective of the project: the distinction between presentation and logic layers is natural because they will be physically separate entities, installed on different machine; the decision to further separate the logic and data layers comes from the need to store potentially large quantity of heterogeneous data. To do so, the system needs different DBMSs (described in greater detail in the coming sections) that satisfies specific needs, hence the decision to separate these two entities.



The presentation layer is further divided in two distinct entities: the Client Application and the Operator Application. The former is installed on smartphones by end users and for them will provide the only interface with the rest of the system. The latter is available only to employees of eMall that are responsible of modifying data in the backend. Each of these two entities are independent and they don't communicate directly (only indirectly via the logic layer).

The logic layer is designed using a microservices architecture. This choice allows the system to be modular and flexible, so it can better withstand future modifications, which is important given the ever-changing nature of mobile applications. It also allows to separate components that serve very different purposes within the system and thus should be logically treated as separate entities. In fact, the components that are part of this layer can be divided in two blocks:

- **eMSP Backend**: groups all the components that take part in the management of data for the mobile application and hence have a more direct interaction with end users
- **CPMS Backend**: groups all the components that together manage the charging stations and all the aspects that relates to them.

These two blocks are very interconnected and rely heavily on each other for the whole system to operate properly.
The components that make them up will be analyzed one by one in the next section.

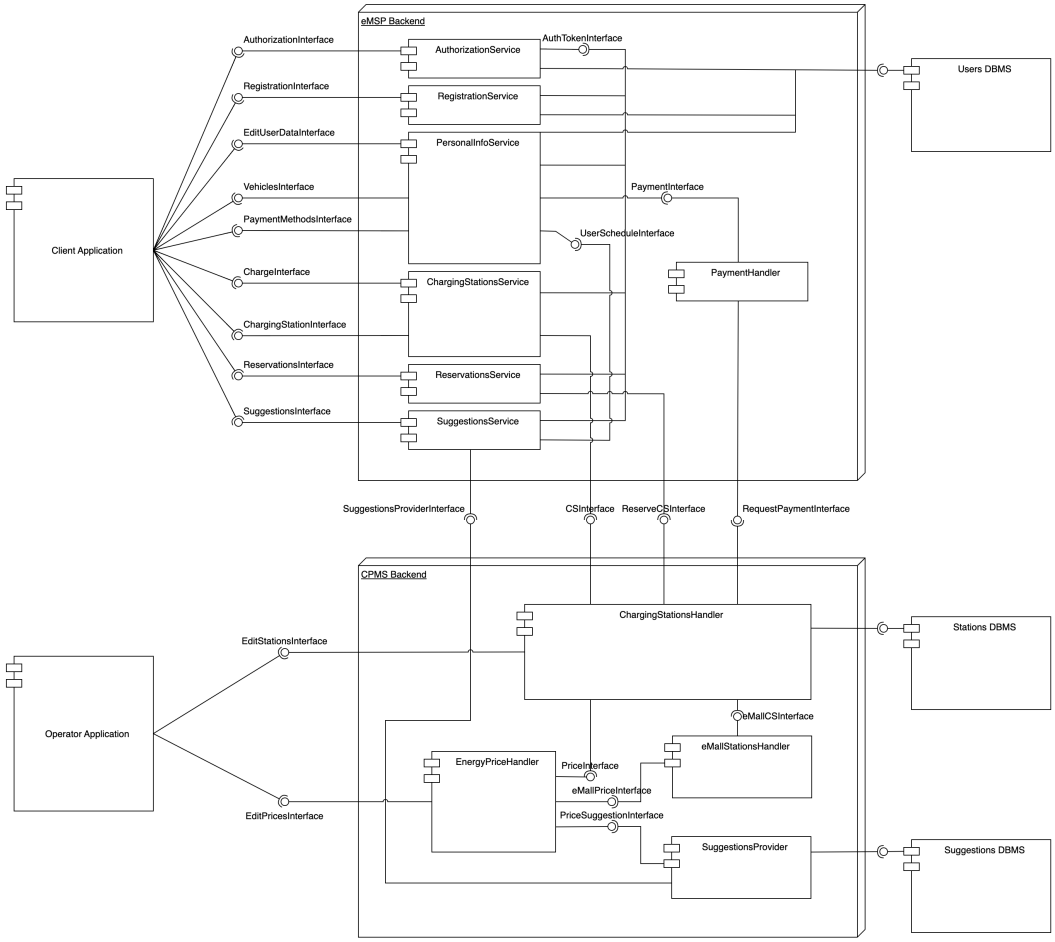
The data layer is composed of three separate DBMSs:

- **Users DBMS**: stores personal data of end users
- **Stations DBMS**: stores data about every charging station in the system

- **Suggestions DBMS**: stores suggestions specifically aimed for every end user

Each of them is independent from the others and there is not direct communication between the three, but they all interact with different parts of the logic layer.
The technology with which every DBMS is implemented is also different based on the peculiarities of the purpose they serve (more on this in section 2.3).

2.2 Component View



In this figure it is explained in greater detail the overall structure of the system. In particular, the microservices that compose the logic layer are shown explicitly. The function of all the components is now briefly described:

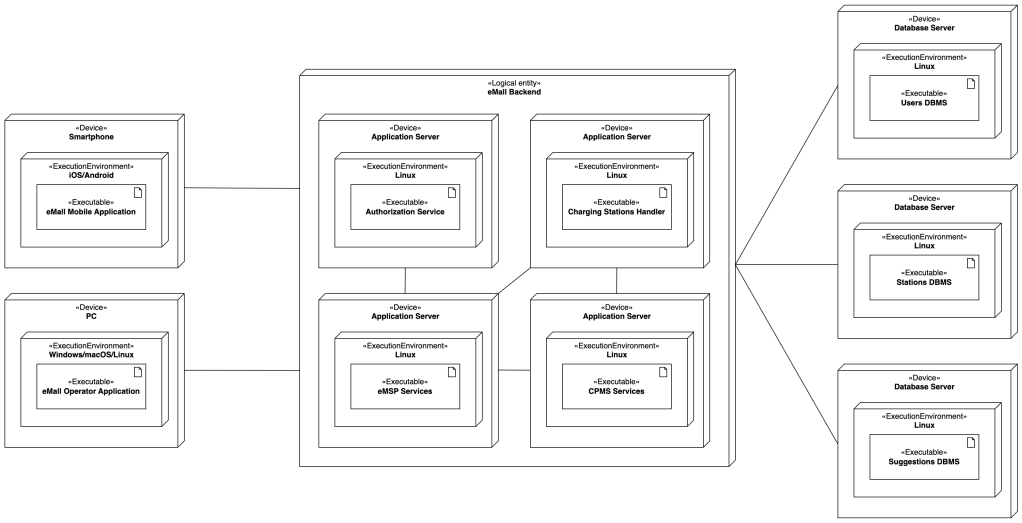
- **Client Application**: provides the entry point for end users to the system, implemented with a mobile application installed on their smartphone. It is responsible to present/gather data in a human-friendly way, which it retrieves by communicating with the components in the Client Services Backend block.
- **Authorization Service**: is responsible of authenticating the registered users by providing them a unique token when they log in, and checking its validity every time a request is sent from the client to another component in the backend.
- **Registration Service**: is responsible to manage the registration of a new user the first time they access the system.
- **Personal Info Service**: is responsible of managing personal information provided by end users via the client application, in particular those about vehicle details, payment methods and user's schedule.
- **Charging Stations Service**: is responsible of communicating with the client about everything that concerns the charging stations. It can retrieve data about them (position, status, prices...) and handle the charging process (starting, monitoring, finishing).
- **Reservations Service**: is responsible of communicating with the client about everything that concerns reservations created by end users when they want to book a charging socket at a certain charging station.
- **Suggestions Service**: is responsible of communicating with the client when a new suggestion is created for a certain user. It also communicates with Personal Info Service to get the user's schedule (appointments)
- **Payment Handler**: provides a unified interface between the various APIs of different payment service providers (like PayPal) and the rest of the system.
- **Users DBMS**: provides a safe and persistent storage for data about personal information of registered users.

- **Operator Application**: provides a GUI with which employees at eMall can monitor the status of their charging stations and manually override decisions made automatically by the CPMS
- **Charging Stations Handler**: is responsible of monitoring and controlling the status of all the charging stations/sockets in the system (both those owned by eMall and by third-party entities)
- **eMall Stations Handler**: is responsible of communicating with charging sockets that are owned directly by eMall.
- **Energy Price Handler**: is responsible of communicating with DSOs and choosing the best one to buy energy from.
- **Suggestions Provider**: is responsible of computing new suggestions for a certain end user based on new energy prices or their schedule.
- **Stations DBMS**: provides a safe and persistent storage for data about charging stations.
- **Suggestions DBMS**: provides a safe and persistent storage for data related to suggestions for end users

The Authorization Service, when the end user logs in, provides a unique token that is valid for 15 minutes and is stored locally by the client application along with the user's credentials. After 15 minutes have passed, the client application sends to the Authorization Service the credentials again and, if they are still valid, the latter provides a new token.
Every time the client application sends a request to one of the components in the logic layer that require the user to be logged in, the token is sent along with the request to be validated. To do so, all these components can communicate directly with the Authorization Service to validate the token the user has sent.

The choice of creating separate entities that may seem redundant (e.g., Charging Stations Service and Charging Station Handler) is made to logically separate those components that directly interact with the end user (eMSP) from those that manage the charging stations themselves (CPMS).

2.3 Deployment View



In the figure above, the deployment diagram of the eMall system is shown.

The two blocks on the left are those that make up the Presentation layer. As already mentioned in previous paragraphs, one is the mobile application, which runs on end users' smartphones (that use iOS/Android) and the other is the operator application, which runs on PCs that use Windows/macOS/Linux.

The block in the center represents the Logic layer. Because of its microservices architecture, it is further distributed on separated physical machines to better balance the workload. In the picture are shown four groups of components:

- **Authorization Service**
- **eMSP Services**: Registration service, Personal Info service, Charging Stations service, Reservations service, Suggestions service, Payment handler
- **Charging Stations Handler**

- **CPMS Services**: Energy Price handler, eMall Stations handler, Suggestions provider

It is expected that the real allocation of components to distinct physical devices will be decided or modified later when the system will be deployed, based on the actual load that every machine will have to bear. Here in the picture, the components that are expected to receive the greatest number of requests were separated from the rest.

The three blocks on the right are the DBMSs in the Data layer. The needs that each of them must satisfy are very different, so they will be implemented using different technologies:

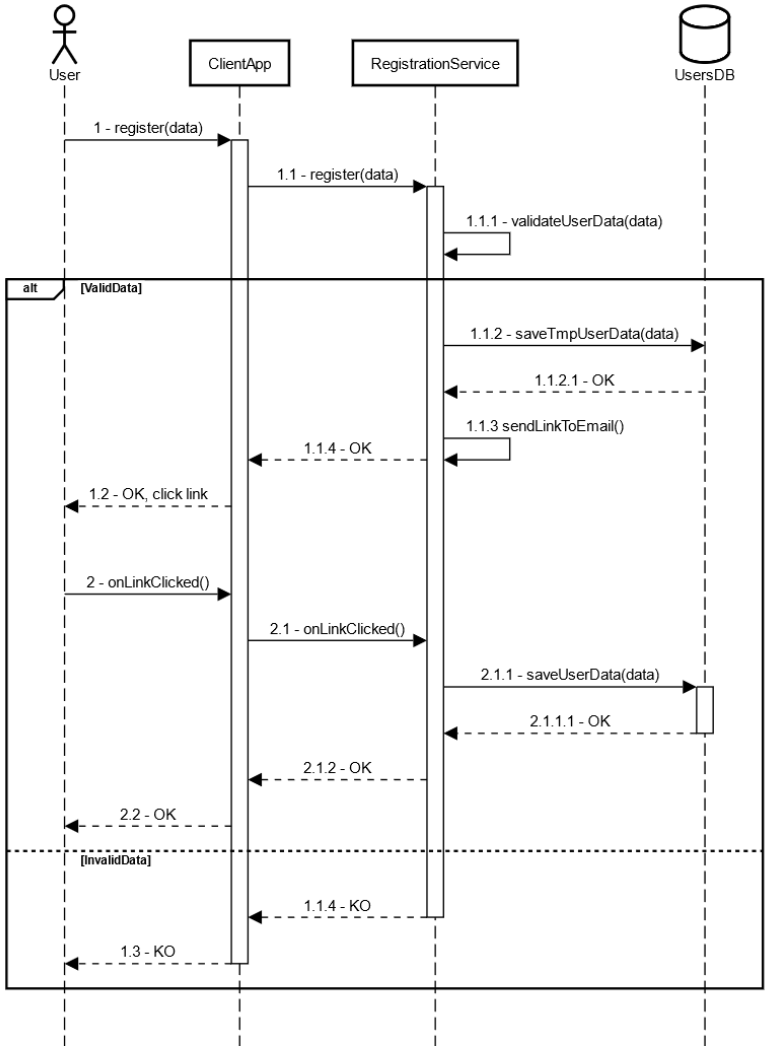
- **Users DBMS**: will store structured data about personal information of end users, so it will be implemented with a relational database in MySQL
- **Stations DBMS**: will store structured data about the status of every charging status, so it is reasonable to assume that it will have to manage a large volume of frequently-updated transactions, so it will be implemented with a relational database in PostgreSQL
- **Suggestions DBMS**: will store unstructured data (possibly with many incoherencies), so it will be implemented with a NoSQL database, for example in MongoDB

2.4 Runtime View

In this sequence we present nine sequence diagrams which show how the various components interact with each other in order to offer the main functionalities of the S2B. The system relies much on the concept of session token, which has been briefly introduced in the previous paragraphs and will be examined in-depth in paragraph 2.4.3; apart from sign up and login, which obviously can be accessed by unregistered users, every other service can be accessed only providing in the request a valid token. It is the token itself which is responsible of univocal identification of the user which makes the request and associate them with their correct permissions.

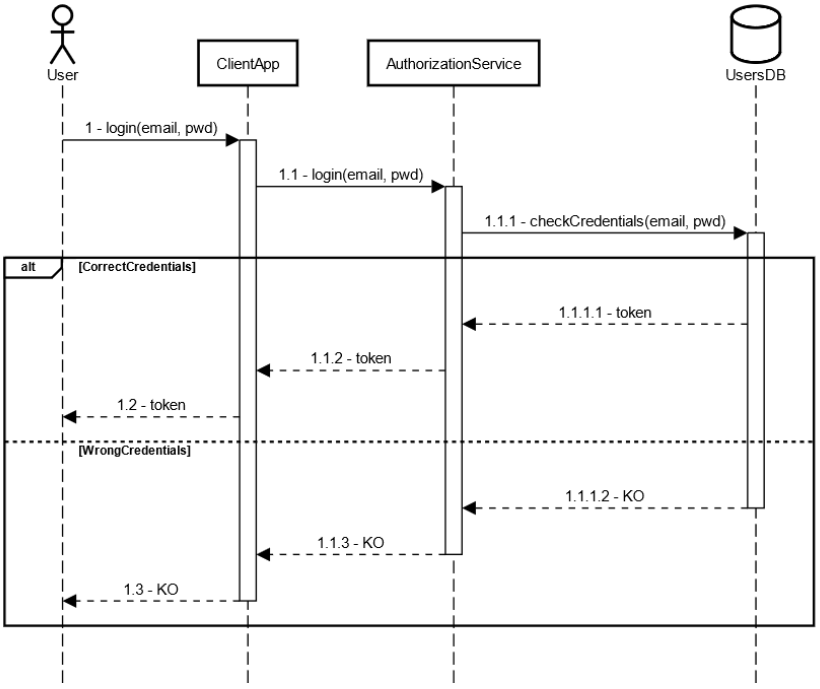
A description of the diagram will be provided in case it's not self-explanatory, otherwise it will be omitted.

- **Sign up**
  @POST("/users/signup")



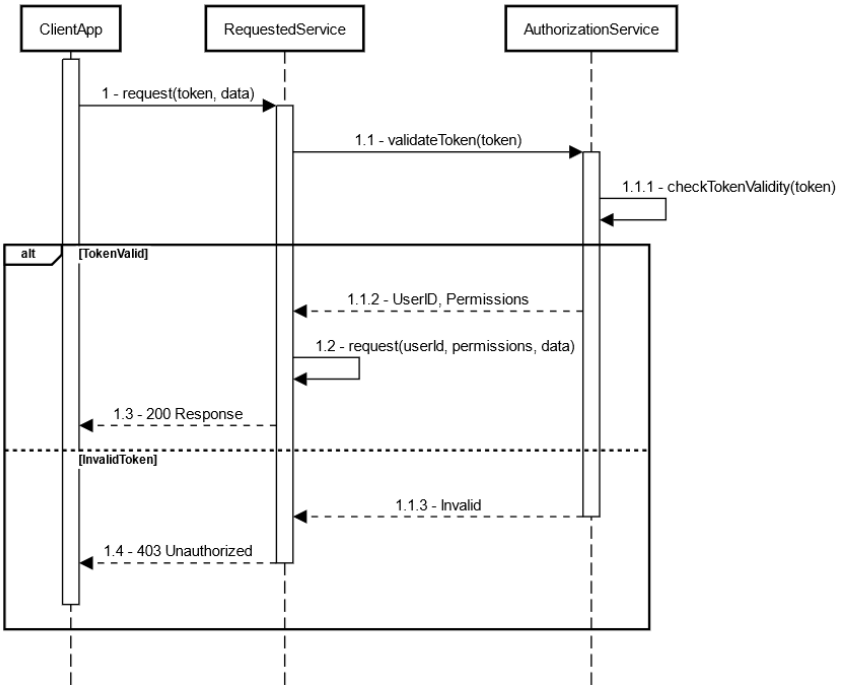
The user sends the registration request containing their data – name, surname, phone number, email and chosen password – to the correct endpoint, offered by RegistrationService. If the provided data is correct, this component attaches a verification link within a message sent to the user's email address. If the user clicks on it, the account is verified and becomes active.

- **Login**
  @POST("/users/login/")



This action is performed every time the client starts a new session or notices that the token it owns is expired. The client saves email and password in a secure storage, sends a login request and receives a valid session token that must be included in all the following requests. The token is a JWT (Json Web Token) which represents the hash of the UserId of the user who sent the login request and is valid for fifteen minutes from the time it's generated.
A brute-force attacks to this endpoint will eventually lead, in addition to the slowdown of the system, to password leaks: an attacker could, to retrieve the correspondence between an email address and the associated password, indeed just send multiple requests to the endpoint until the provided password is correct.
A solution to this problem could be a temporary block – e.g., three hours – for a user which has failed to authenticate for five consecutive attempts. Beware: starting from the next sequence diagram, the interaction with the user will be considered implicit, so the messages will start from entity ClientApp.

- **General authorized request**

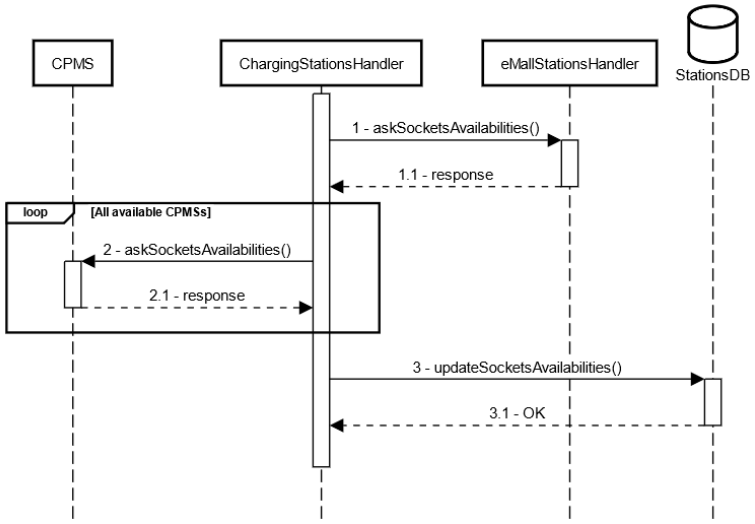


The flow of a general request to an endpoint of the system different from the previously described two is analyzed in the above diagram.

The user inserts in the Authorization header of their request a JWT, whose validity is checked by AuthorizationService: if it is valid and the associated user has the needed permissions to access RequestedService (general name to identify one of the services) , the requests is taken in charge, otherwise (in case of invalid or expired token) it's rejected with 403 status code.

Starting from here, all requests will be considered as correctly authorized, therefore the invocation to AuthorizationService through TokenInterface will be omitted in the diagrams. The interaction of the user with their ClientApp is taken for granted too.

- **Retrieve data about charging stations**

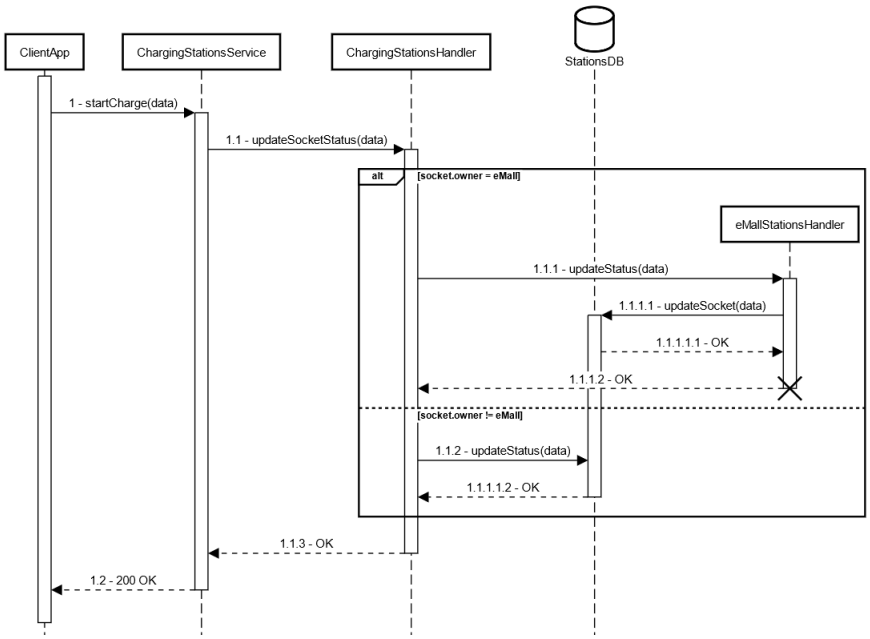


This diagram represents a periodic task, executed automatically by the system according to a schedule (of course editable by an eMall operator through their desktop app), which in parallel interrogates the component eMallStationsHandler (responsible of collecting all data related to the charging stations managed by eMall's CPMS) and all the APIs of the available CPMSs to retrieve updated data about all the charging stations.

This operation should be executed at least once per minute in the central hours of the day and about once every five minutes in the times in which the system is less congested, in order to save resources. In any case the selected time interval must be a reasonable tradeoff between the desired will to show to the users the most updated data in every moment and the computational load required to the machine.

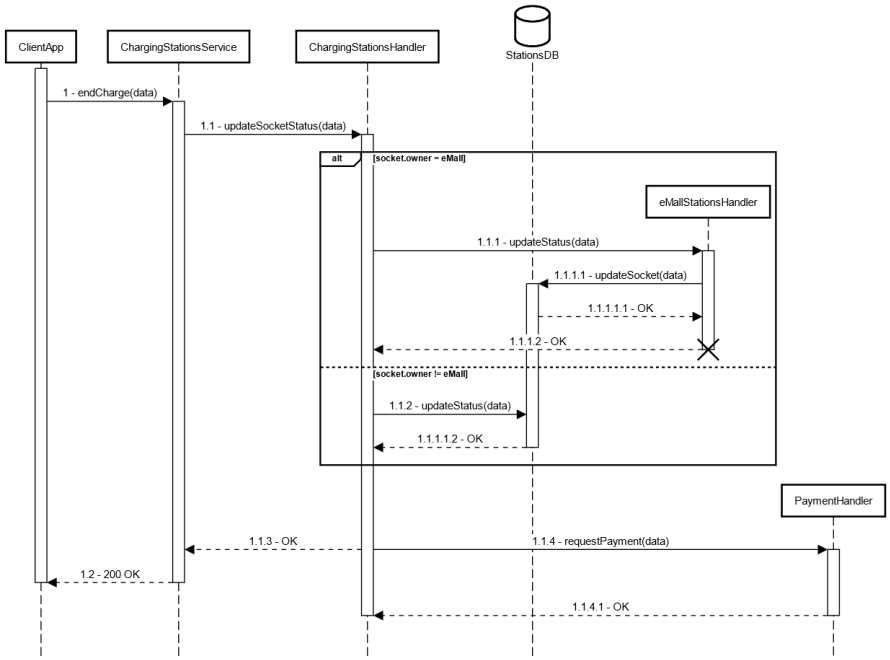
It's important to underline that, as we'll show in the following figures, the system also waits for a further confirmation from the owner CPMS when a socket is booked.

- **Start the charge of a vehicle**
  @POST("/stations/charges/start")



The user inserts all details related to the charge and then invokes the endpoint hosted by ChargingStationsService: this component calls a procedure on ChargingStationsHandler which updates the status of the socket in StationsDB. To maintain the diagram clean and readable, we assumed that the considered socket continues to be free in the interval between the user's click on it and the moment in which the availability is checked in the database.

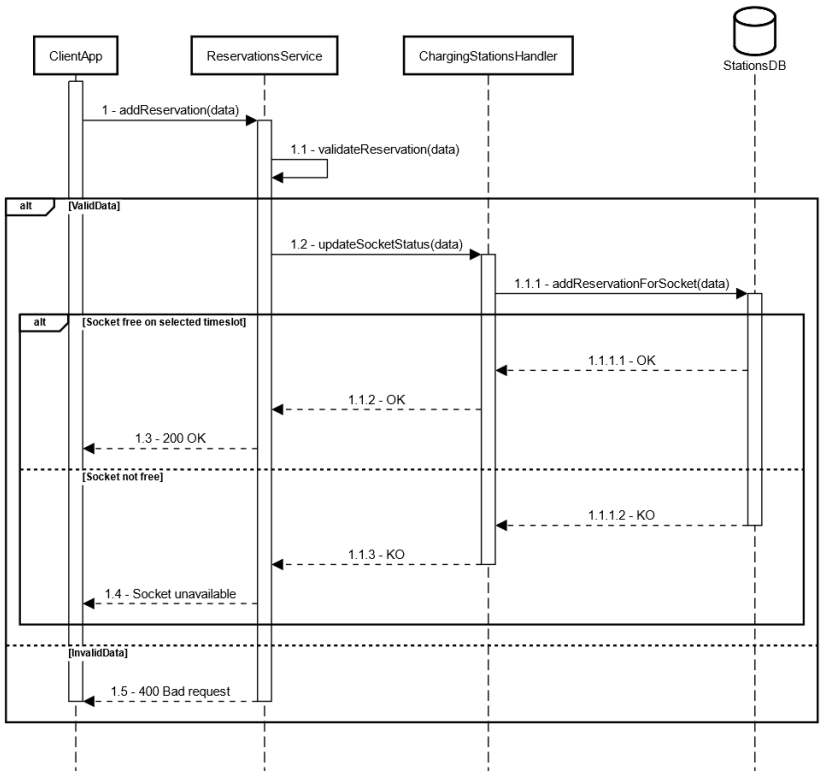
- **End recharge of a vehicle**
  @POST("stations/charges/end")



The component ChargingStationsHandler updates the status of the socket, communicating with the appropriate component according to the owner of the socket. Then, in parallel, it requests the payment of the owed amount to PaymentHandler and notifies ChargingStationsService (which then acknowledges the user).

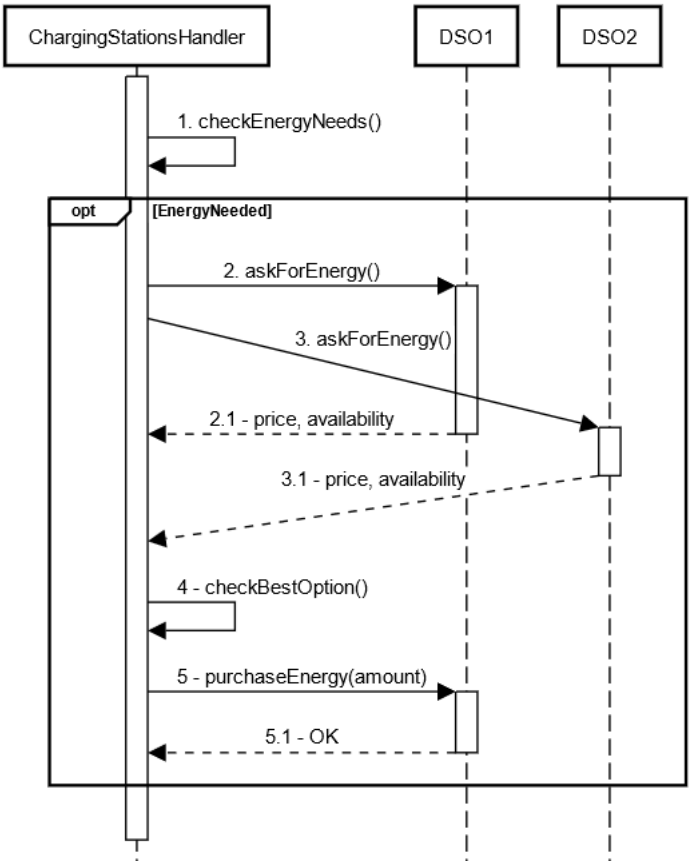
Note that the interaction with the client application ends after the system informs it that the charge is correctly terminated, so it's not required that the user awaits the outcome of the transaction – it can be seen in the proper section of the application, as described in RASD.

- **Create a reservation**
  @POST("/stations/book")



When the client asks to book a socket, there's always a double-check on the validity of the reservation on the component ChargingStationsHandler, to avoid concurrent reservations coming from different users. Apart from this clarification, the diagram should be clear enough.

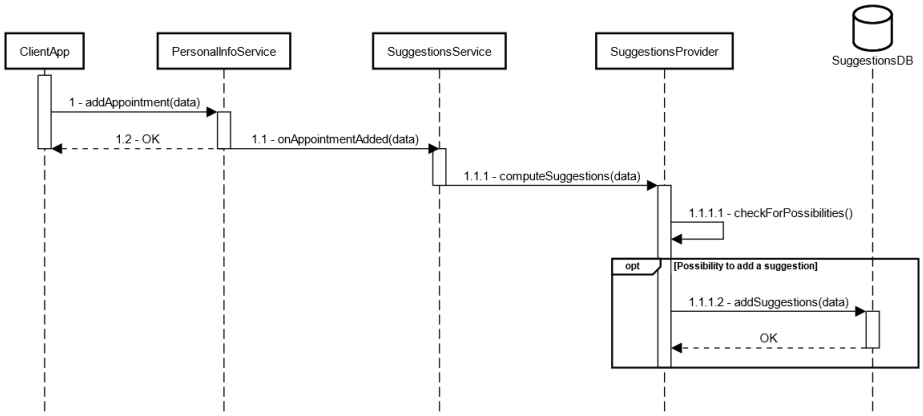
- **Purchase of energy from a DSO**



This diagram represents a periodical task performed by the component ChargingStationsHandler, which is dispatched automatically by the system according to a schedule that can be modified by an eMall operator.

Many DSOs are queried in parallel to know their availability of energy and the proposed prices, then, when all data are collected, a decision is taken and one or more DSOs are contacted back to purchase energy.

- **Computation of a suggestion based on user schedule**

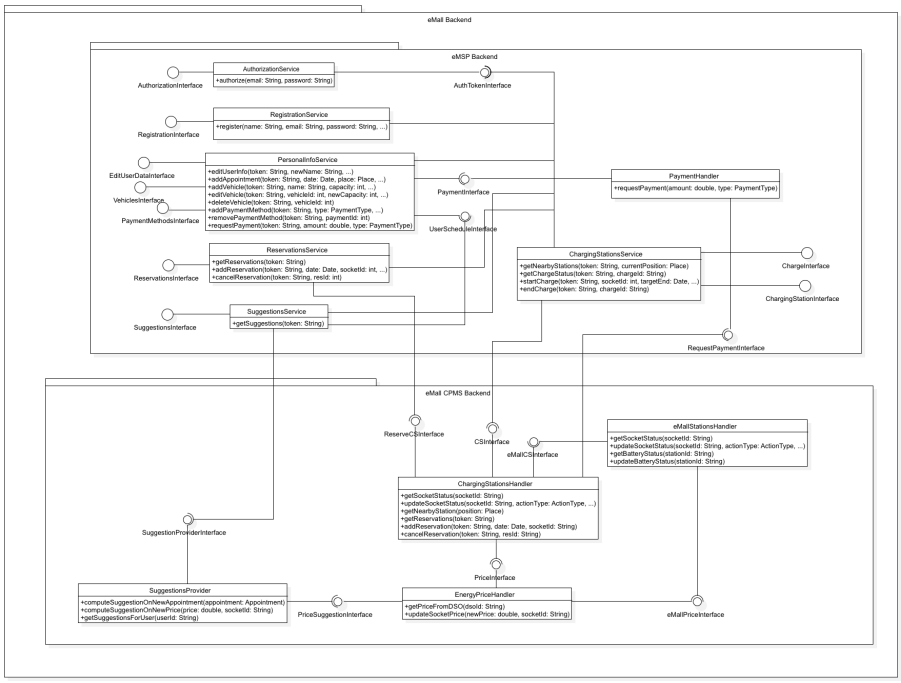


When the client applications, which has the permissions to access the calendar, detects that the user has added a new appointment to their schedule, communicates it to PersonalInfoService, which then invokes SuggestionsService without expecting a response; SuggestionsService does the same with SuggestionsProvider.

This component checks whether it makes sense to add a suggestion related to the new appointment of the user and, if so, saves it into the correct database. The user can retrieve suggestions with a simple call to the endpoint @GET("/suggestions/all"), which will scan SuggestionsDB and will provide them all the computed clues, after checking the temporal availability of each socket.

2.5 Component Interfaces

In this section it is provided a global view of the system's interfaces with some of the methods needed for it to be implemented



2.6 Selected Architectural Styles and Patterns

- **Microservices architecture**
  Since eMall's backend is composed of many different components that not necessarily depend on each other, it becomes possible to adopt a microservices architecture rather than a monolithic architecture.
  This approach brings several advantages, including the possibility to build different teams working independently on different schedules and functionalities, to rewrite a service in a higher performance language if it struggles to scale, and to replicate and scale each service on its own, instead of being forced to scale the entire application as it would happen in a monolith.
  However, this architectural style introduces some complexities that must be tackled in order to guarantee the proper working of the sys-

tem, in particular on how to communicate between services and to deal with partial failures.

- **RESTful architecture**
  Both the interactions between the mobile application and the system and between the latter and the external services follow the principles of REST: the server is stateless (does not contain any information on the status of a client, which is handled client-side), all the requests are made through appropriate conventional methods (GET/POST/-PATCH) and each response follows a predefined schema, according to the called endpoint. Requests and responses are encoded in JSON.

2.7 Other Design Decisions

- **Scale-out**
  This approach consists of replicating the nodes which hosts the services which have the maximum number of requests or that take a lot of computational power to process them, in order to increase the availability of the service – and therefore of the system in its totality. The components that for sure need to be replicated, considered the tasks they have to perform and their importance, are Authentication-Service and StationsHandler. After the system is deployed, regular analysis on requests flow and on servers' performances can identify new bottlenecks, stating whether it's needed to replicate other components.
  Of course, the choice of replicating the machines that provide a service increases the complexity of the development of the system, because it requires a load balancer which redirects the requests to the node with the lowest workload in that moment.

- **Backup of databases**
  It's crucially important that no data of the system is lost after a failure, so each database must have a backup copy in a machine different from the one which hosts the primary copy.

- **Firewalls and protections** The access to the system must be monitored accurately, to ensure that no malicious actions are performed by external users. The firewalls must be configured to route only the traffic coming from the official client application or from one of the authorized domains and to limit the rate of the requests coming from the same IP address, in order to prevent DOS-based attacks.

- **Continuous fetching of stations data** In the previous sections, we described how eMall's CPMS retrieves data about its stations and the ones managed by other CPMSs, i.e., by proactively querying the responsible components at fixed time intervals. This decision is a reasonable tradeoff between the intention to show the users information which are as updated as possible and the costs involved by a proactive approach. The schedule can be changed at any time by a human operator, in order to optimize the performances of the system. Of course, every charge or reservation request on socket not directly owned by eMall is double-checked with the responsible CPMS, guaranteeing the safety of the approach.

3 User Interface Design

As stated in the introduction, eMall needs to be accessible to many people, also for ecological reasons, and their ability in using mobile application could vary drastically; hence we opted for a clean and easy-to-use style for the user interfaces, with few and required possible actions for the end user and not too much personalization (it is unnecessary and only increases the complexity level).

Most of the screens regarding the UIs are located in section 3.1.1 of the RASD, in this section instead are shown the interactions among them to better have it clear, if the interfaces were not sufficiently self-explanatory. It is important to specify, as stated in the RASD, that not all the screens are represented, but only the most significative ones.

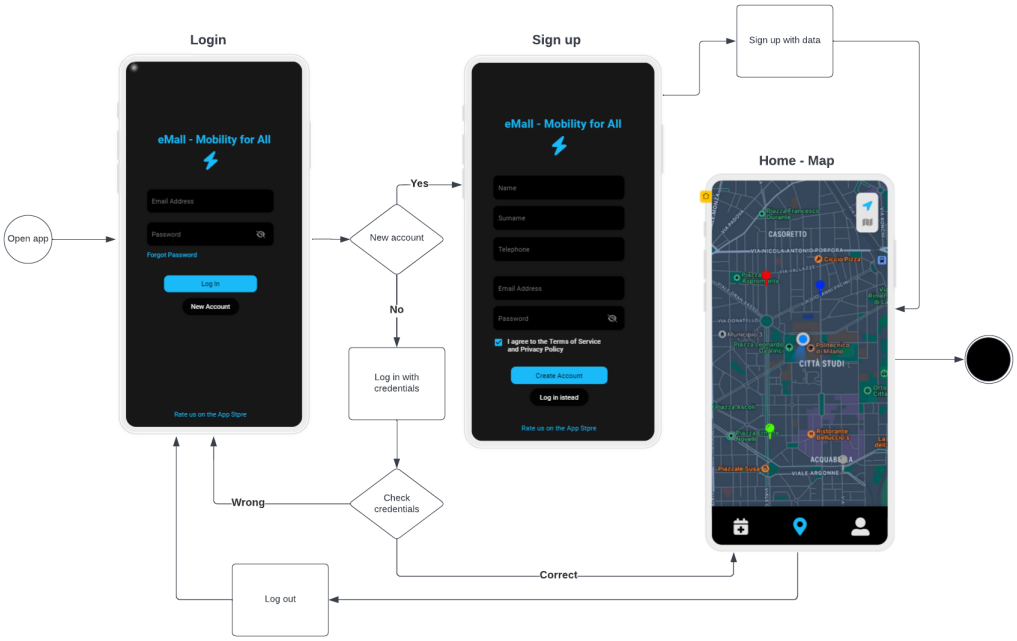


Figure 1: Login and sign up process for the user

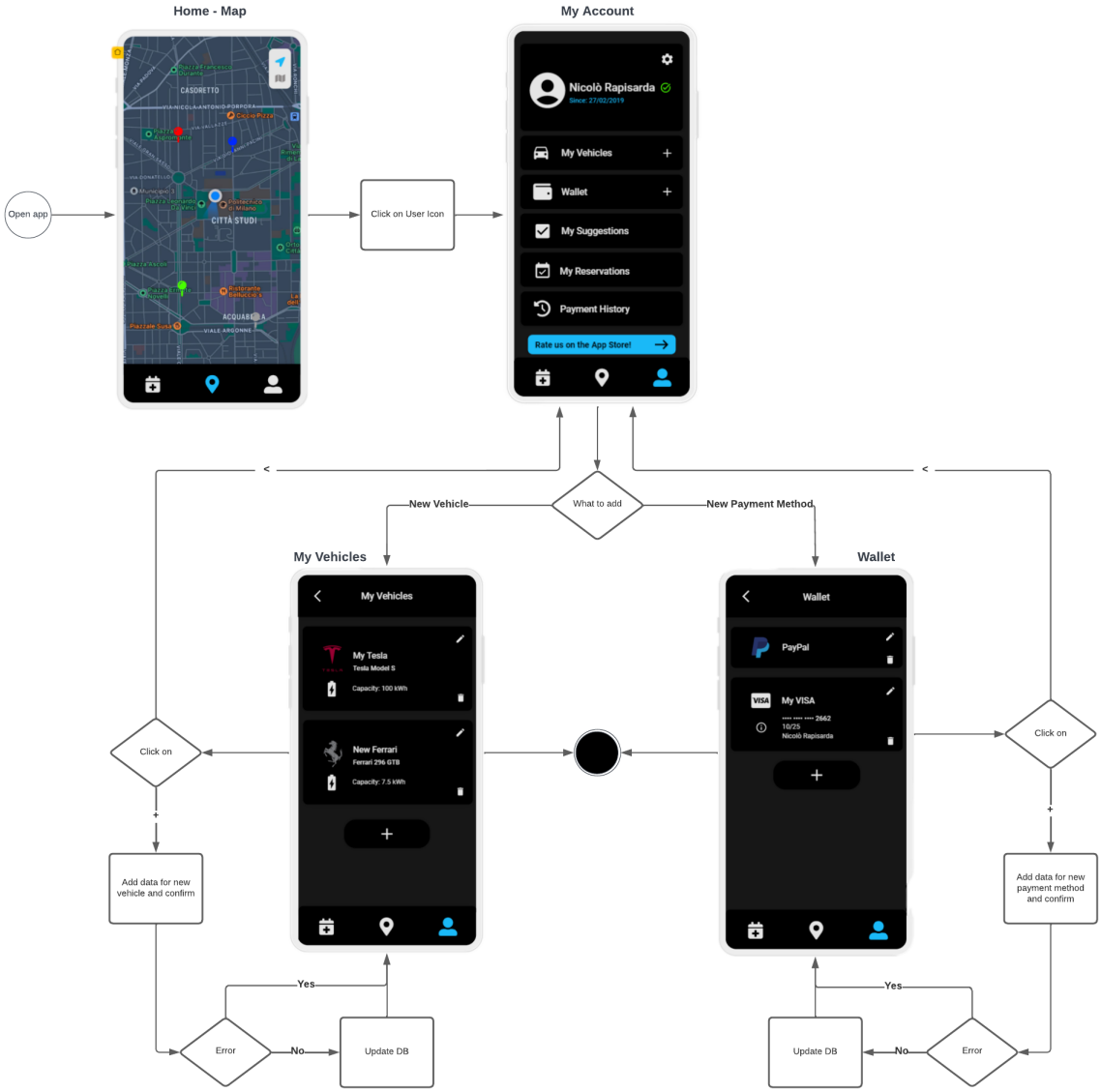


Figure 2: Add a vehicle and a payment method to the user account

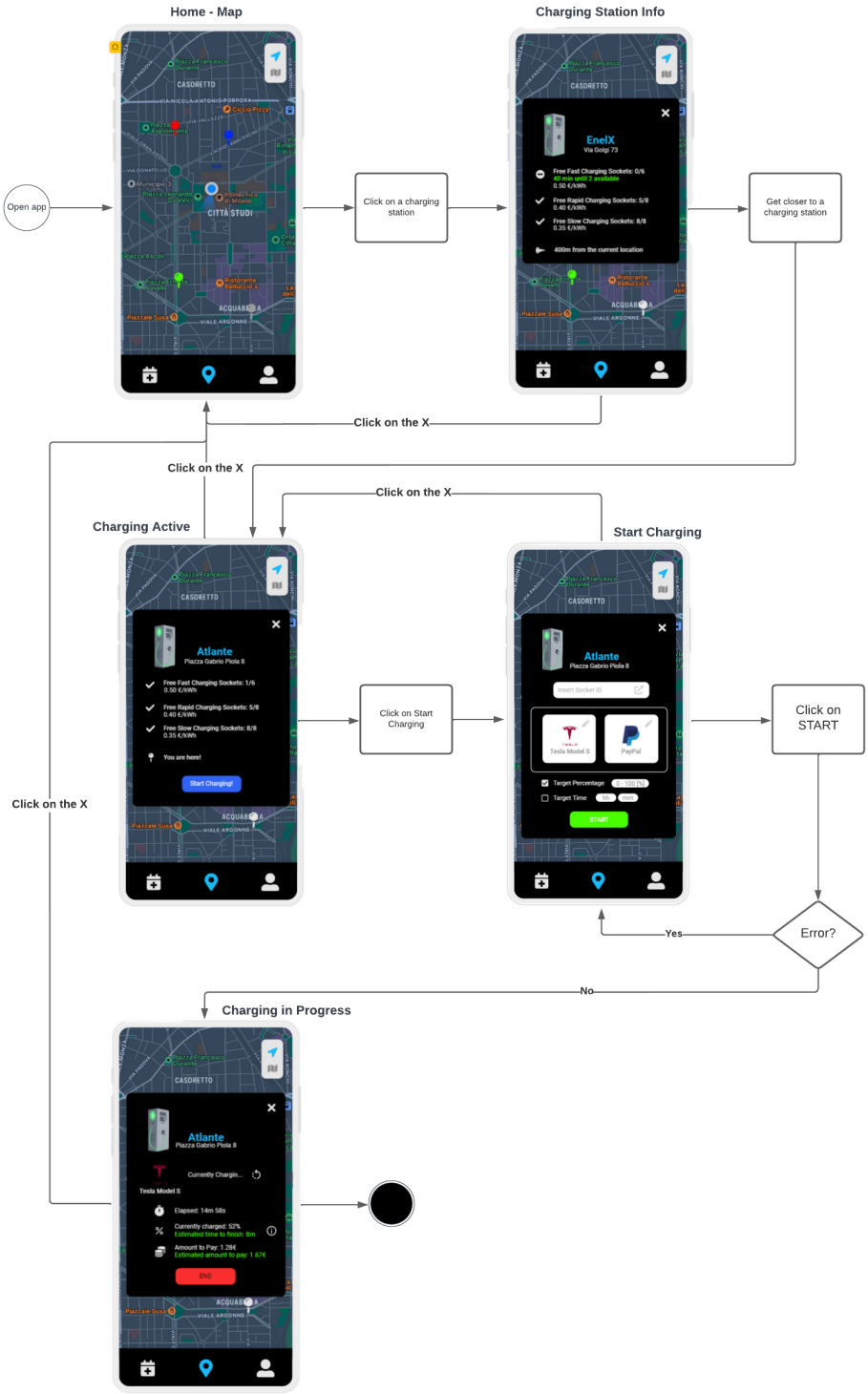


Figure 3: Charging process

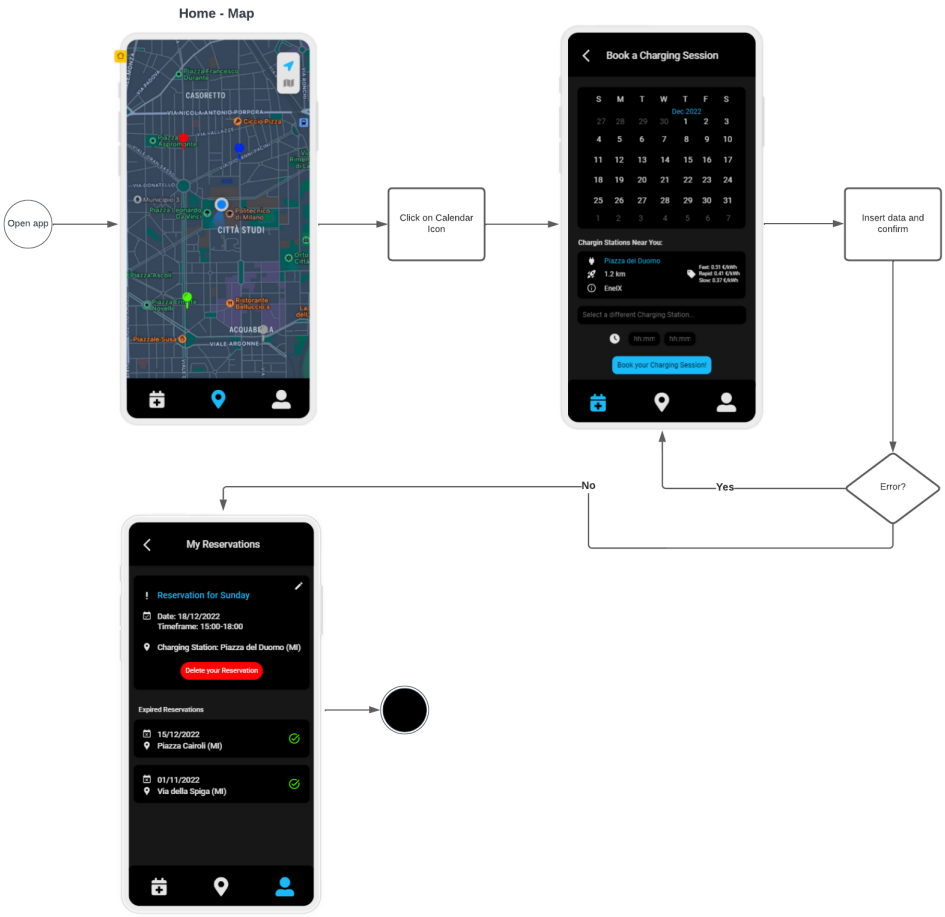


Figure 4: Book a session

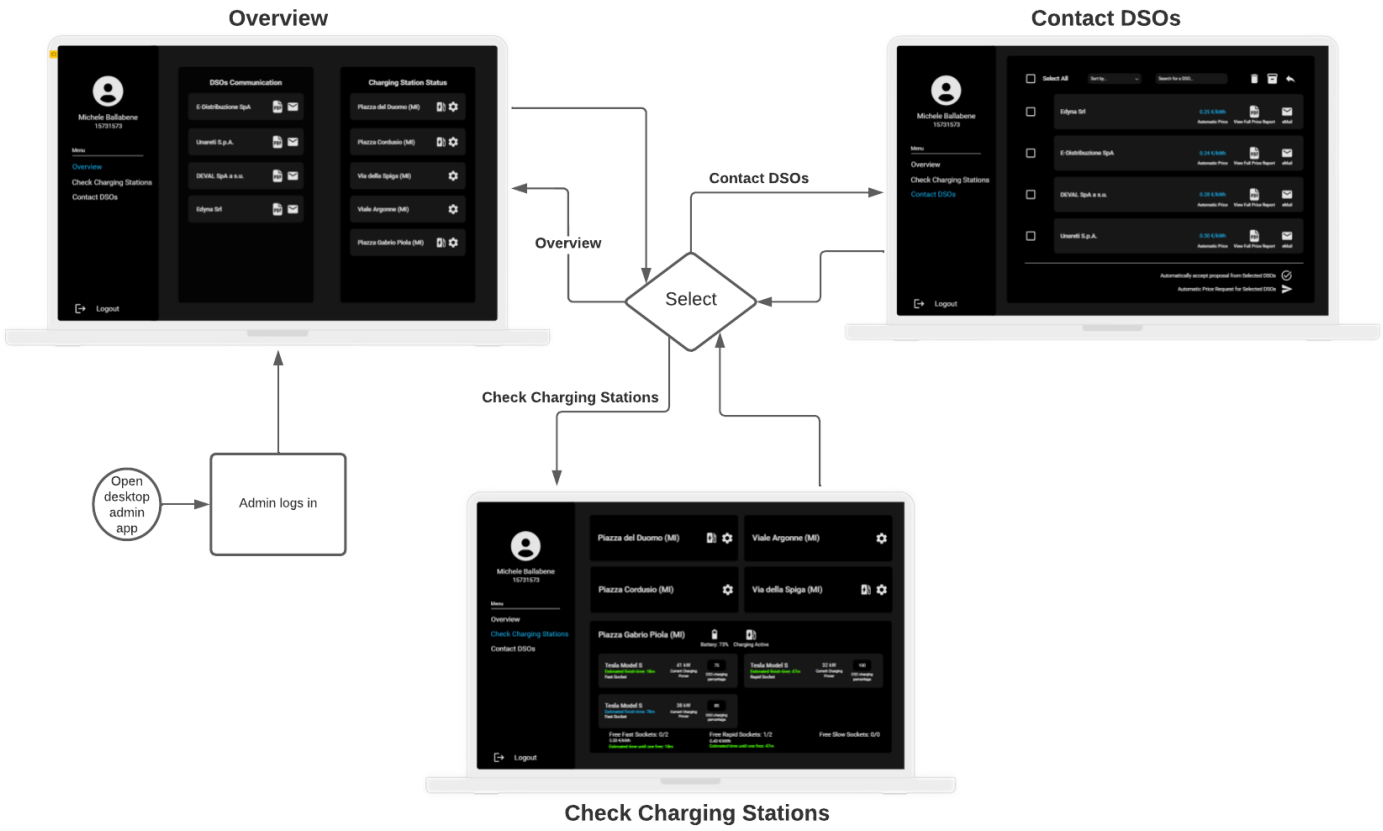


Figure 5: Admin desktop application

4 Requirements traceability

In this section, it will be explained how the requirements identified in the RASD translate to the design elements presented in the previous sections of this document. The following is a copy of the requirement table from the RASD, inserted here for reference.

Requirement	Description
R1	The system should allow an unregistered user to create a new account
R2	The system should allow (registered) end users to log in
R3	The system should allow end users to save information about their vehicle(s)
R4	The system should allow end users to save payment methods in their account
R5	The system should allow end users to see a map with information about nearby charging stations
R6	The system should allow end users to start/terminate the charging process at an available charging socket
R7	The system should allow end users to make/cancel a reservation for a charging socket for a certain time
R8	The system should allow end users to make payments for the services used
R9	The system should send end users special offers (when available)
R10	The system should be able to choose which DSO to buy energy from
R11	The system should be able to recharge the batteries of a charging station
R12	The system can send e-mails to the end user
R13	The system can store personal information about the end users
R14	The system can access the user's location
R15	The system can access the user's schedule
R16	The system can control (and show data about) the charging sockets owned directly by e-Mall

R17	The system can communicate with external CPMSs (owned by third-party entities) via specific APIs
R18	The system can communicate with DSOs via specific APIs
R19	The system can interact with human operators at eMall using a desktop application

- R1: The system should allow an unregistered user to create a new account
  Client Application: to let the user directly interact with the system
  Registration Service: to let the unregistered user create a new account
  Personal Info Service: in case the user (once registered) wants to update personal information in their account, via the EditUserData Interface
  Users DBMS: to store the information provided by the user

- R2: The system should allow (registered) end users to log in
  Client Application: to let the user directly interact with the system
  Authorization Service: to let end users log in with their account by verifying their credentials.
  Users DBMS: to store information provided by the user

- R3: The system should allow end users to save information about their vehicles
  Client Application: to let the user directly interact with the system
  Personal Info Service: to let the user add information about their vehicles, via the Vehicles Interface.
  Authorization Service: to validate that the user is correctly authenticated in the system. This module verifies the user's credentials (see description of R2) and provides the client with a unique token that expires after a certain time. Every time the client sends a request using a certain interface, this module checks that it is still valid. To do so, it must communicate with all the other modules that interact directly with a registered end user, via a proper interface (AuthToken Interface).
  Users DBMS: to store the information provided by the user

- R4: The system should allow end users to save payment methods in their account
  Client Application: to let the user directly interact with the system
  Personal Info Service: to let the user add information about their payment methods, via the Payment Methods Interface.
  Authorization Service: to validate that the user is correctly authenticated in the system (see description of R3)
  Users DBMS: to store information provided by the user

- R5: The system should allow end users to see a map with information about nearby charging stations
  Client Application: to let the user directly interact with the system
  Charging Stations Service: to provide the user data about the position and status of stations nearby using the ChargingStation Interface and getNearbyStations() method.
  Authorization Service: to validate that the user is correctly authenticated in the system (see description of R3)
  Charging Stations Handler: to retrieve data from the Stations DBMS and pass them to the Charging Stations Service using the CS Interface
  Stations DBMS: to store persistently data about the charging stations (owned by eMall or a different CPO)

- R6: The system should allow end users to start/terminate the charging process at an available charging socket
  Client Application: to let the user directly interact with the system
  Charging Stations Service: to receive the start/terminate command via the startCharge() and endCharge() methods, respectively, and send the request to the Charging Stations Handler to update the status of the selected socket using the updateSocketStatus() method
  Authorization Service: to validate that the user is correctly authenticated in the system (see description of R3)
  Charging Stations Handler: to update the status of the requested socket in the Stations DBMS. If the socket if owned by a different CPO, the component sends a request to them using an appropriate API, if it is directly owned by eMall the request is sent to the eMall Stations Handler
  eMall Stations Handler: to update the status of the requested socket if the socket if directly owned by eMall. The stations are not con-

sidered as part of the system, and they each communicate with this component using an appropriate API
Stations DBMS: to store information about the current status of every socket

- R7: The system should allow end users to make/cancel a reservation for a charging socket for a certain time
  Client Application: to let the user directly interact with the system
  Reservations Service: to manage the request from the user of making/cancelling a reservation
  Authorization Service: to validate that the user is correctly authenticated in the system (see description of R3)
  Charging Stations Handler: to communicate to/from the Stations DBMS the changes of the reservations made by the user
  Stations DBMS: to store persistently every reservation the user has made

- R8: The system should allow end users to make payments for the services used
  Client Application: to let the user directly interact with the system
  Personal Info Service: to manage the request to make a payment from the client application, retrieving the information about payment methods needed from the Users CPMS
  Authorization Service: to validate that the user is correctly authenticated in the system (see description of R3)
  Payment Handler: to handle the communication with the various external payment service providers (e.g., PayPal)
  Charging Stations Handler: to automatically charge the sum due when a charging process is finished for the user
  Users DBMS: to store the payment methods inserted by the user

- R9: The system should send end users special offers (when available)
  Client Application: to let the user directly interact with the system and gain access to the user's schedule
  Suggestions Service: to send to the client application suggestions and special offers when they become available
  Authorization Service: to validate that the user is correctly authenticated in the system (see description of R3)
  Personal Info Service: to import appointments from the user's cal-

endar app
Suggestions Provider: to compute new suggestions for the end user, using their schedule and/or new energy prices available from a certain CPO
Suggestions DBMS: to store the information about suggestions computed for a certain user

- R10: The system should be able to choose which DSO to buy energy from
  Energy Price Handler: to retrieve current energy prices for every available DSO and choose the preferred one (automatically or not)
  Charging Stations Handler: to update the prices for the end users based on the choices made by the Energy Price Handler
  Operator Application: to manually modify (with a human operator) the choices made automatically by the Energy Price Handler
  Stations DBMS: to store the updated price of energy for end users

- R11: The system should be able to recharge the batteries of a charging station
  eMall Stations Handler: to update the status of the batteries available at certain charging stations

- R12: The system can send e-mails to the end user
  Client Application: to get the e-mail address with which the new user wants to sign up and show a message confirming the e-mail was successfully sent.
  Registration Service: to send an e-mail to a new user in the process of signing up to confirm the e-mail address inserted

- R13: The system can store personal information about the end users
  Users DBMS: to persistently store information provided by the end user

- R14: The system can access the user's location
  Client Application: to get access to the user's location using APIs provided by the operating system of the smartphone on which the

application is installed (after permission to do so has been granted)

- R15: The system can access the user's schedule (in the calendar app)
  Client Application: to get access to the user's calendar app on the smartphone on which the application is installed (after permission to do so has been granted)

- R16: The system can control the charging sockets owned directly by eMall
  eMall Stations Handler: to directly communicate with the sockets owned directly by eMall using appropriate APIs

- R17: The system can communicate with external CPMSs (owned by third-party entities) via specific APIs
  Charging Stations Handler: to communicate directly with the said entities using specific APIs
  Stations DBMS: to store data about the external CPMSs, like a list of all their sockets and corresponding status

- R18: The system can communicate with DSOs via specific APIs
  Energy Price Handler: to communicate directly with DSO using specific APIs

- R19: The system can interact with human operators at e-Mall using a desktop application
  Operator Application: to specifically serve this requirement

The following is a table that better show how every requirement map to at least one component in the system.

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19
CLI	•	•	•	•	•	•	•	•	•			•		•	•				
AUT		•	•	•	•	•	•	•	•										
REG	•											•							
PIS	•		•	•				•	•										
CSS					•	•													
RES							•												
SUG									•										
PAY								•											
UDB	•	•	•	•				•					•						
OPR										•									•
CSH					•	•	•	•		•							•		
EML						•					•					•			
EPH										•								•	
SUP									•										
SDB					•	•	•			•							•		
ODB									•										

Abbreviations used:

- **CLI**: Client Application
- **AUT**: Authorization Service
- **REG**: Registration Service
- **PIS**: Personal Info Service
- **CSS**: Charging Stations Service
- **RES**: Reservations Service
- **SUG**: Suggestions Service
- **PAY**: Payment Handler
- **UDB**: Users DBMS
- **OPR**: Operator Application
- **CSH**: Charging Stations Handler
- **EML**: eMall Stations Handler
- **EPH**: Energy Price Handler
- **SUP**: Suggestions Provider
- **SDB**: Stations DBMS
- **ODB**: Suggestions DBMS

5 Implementation, Integration and Test Plan

In this section will be described the plans that have to be followed in order to implement, integrate and test the various components of the system.

5.1 Implementation

The implementation of the system will follow a precise schedule that will end up on the final release's date. By the time the application will be launched on the marketplace, all the components will have to be fully completed and working.

Having chosen a microservice-based architecture where many services are offered by a single component and independently from the others (and, in general, rarely more than two components interact in order to provide a service), the development of the majority of the components can be carried out by separate teams independently one from another.

The general strategy of the implementation is bottom-up, with the purpose of encouraging the reusability of the various components and avoiding writing unnecessary stubs when it comes to testing.

At the end of the development process, as mentioned, there will be one final, big release at the end, that will be obviously followed by patches and updates with the aims of introducing new features, fixing bugs and meeting the needs of the users. This approach does not involve significative disadvantages for this case study: there's no need to schedule minor, intermediate releases, because eMall's business model can be pursued only if all the functionalities described in the RASD and in this document are implemented.

The first part of the system which has to be developed is the one which in the component diagram is marked as eMall CPMS backend, because it will surely be longer to build and more crucial than the other part in providing the desired functionalities of the S2B. Inside this "macro-component", the single components have different priorities and complexities: following the bottom-up approach, we can implement the DBMSs first, then, possibly in parallel, SuggestionsProvider and EnergyPriceHandler, followed by eMall-StationsHandler and eventually by StationsHandler, the core component of this group.

A similar schedule can be followed when developing the remaining part of the system: the DBMS first, then AuthorizationService and Payment Handler and, subsequently, all the other components in parallel.

The mobile application for the users will be developed by a team different from the ones that take care of the backend, since the technologies and the approaches involved are really different. The app can be implemented when the backend is almost completed: in fact, it could be used to further test the correctness of the functionalities offered by the system. The user experience should be one of the priorities in mind of the developers while building the application: to ensure this requirement is met, a beta testing campaign could be launched, in order to collect feedbacks before the official launch.

It's reasonable to implement the desktop app for eMall's operators only when the backend is ready, tested and fully working, because the system can work properly also without the human interaction and, most importantly, the development of this relatively simple application would require the creation of a lot of unnecessary stubs which fiction the components that are not ready yet.

5.2 Integration and Test Plan

The order in which components are integrated is the one which has been briefly explained in the previous paragraph. The testing strategy will be bottom-up, as the implementation: this way it's possible to test the correctness of every component if seen in isolation and, through the use of appropriate drivers, how the various components interact with each other. With this approach, at the end of the development, every component should be correctly tested both on its own and when working with other parts of the system with the goal of providing a service.

A component can be integrated into the system if and only if it offers all the functionalities defined in its representation within the component diagram and it successfully passes all the unit tests.

In the following pictures we'll show how the service related to suggestions is progressively integrated into the system. Note that the same procedure applies to all the other components presented in the component diagram.

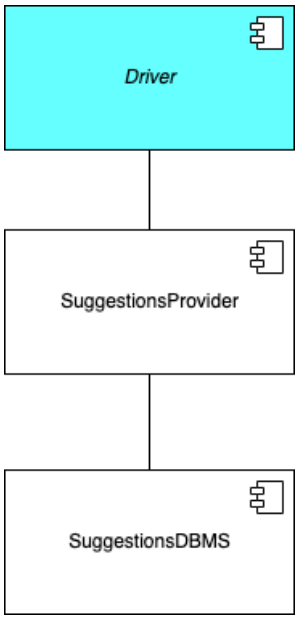


Figure 6: Integration test of SuggestionProvider – belonging to eMall CPMS's backend

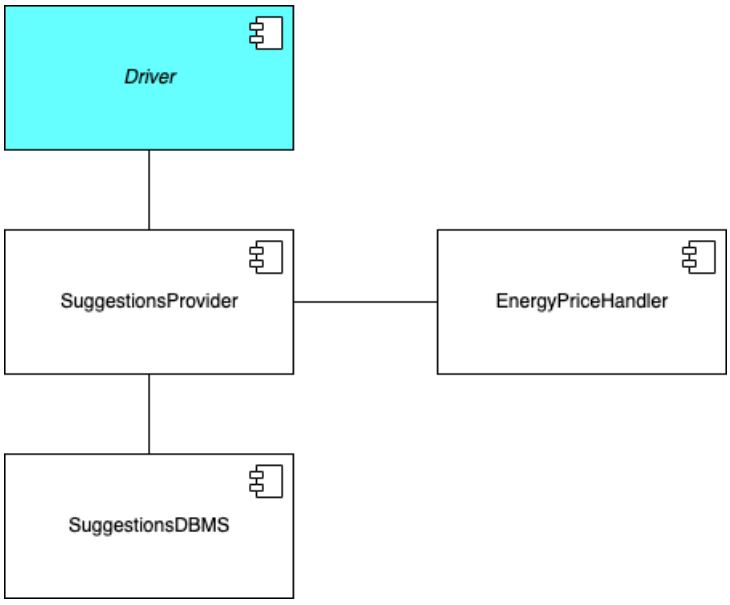


Figure 7: Integration test of EnergyPriceHandler

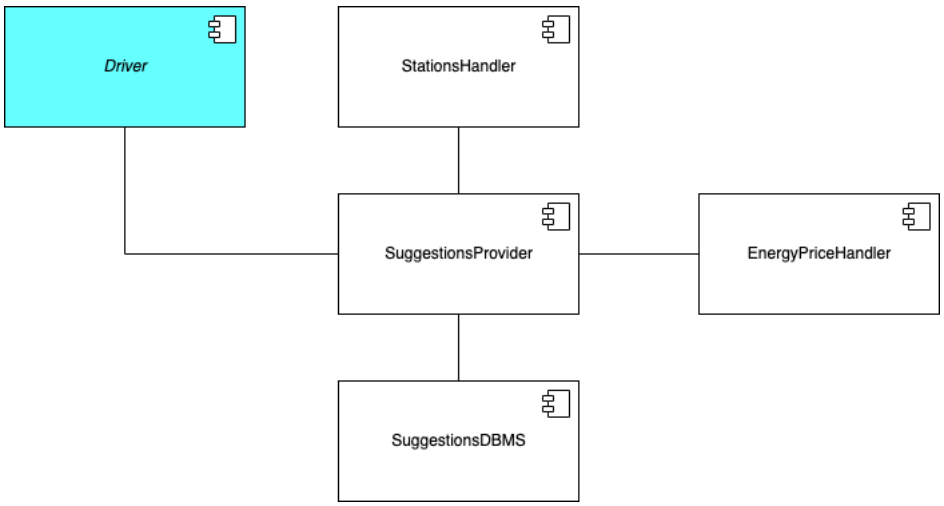


Figure 8: Integration test of StationsHandler – the main component of CPMS backend

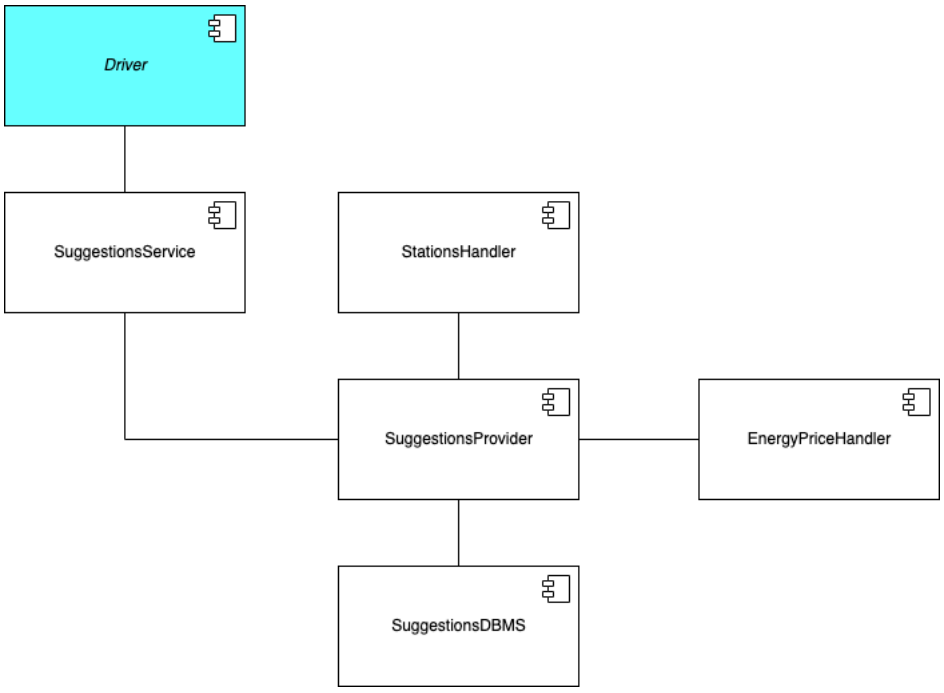


Figure 9: Integration test of SuggestionsService – one component away from the integration of the client application

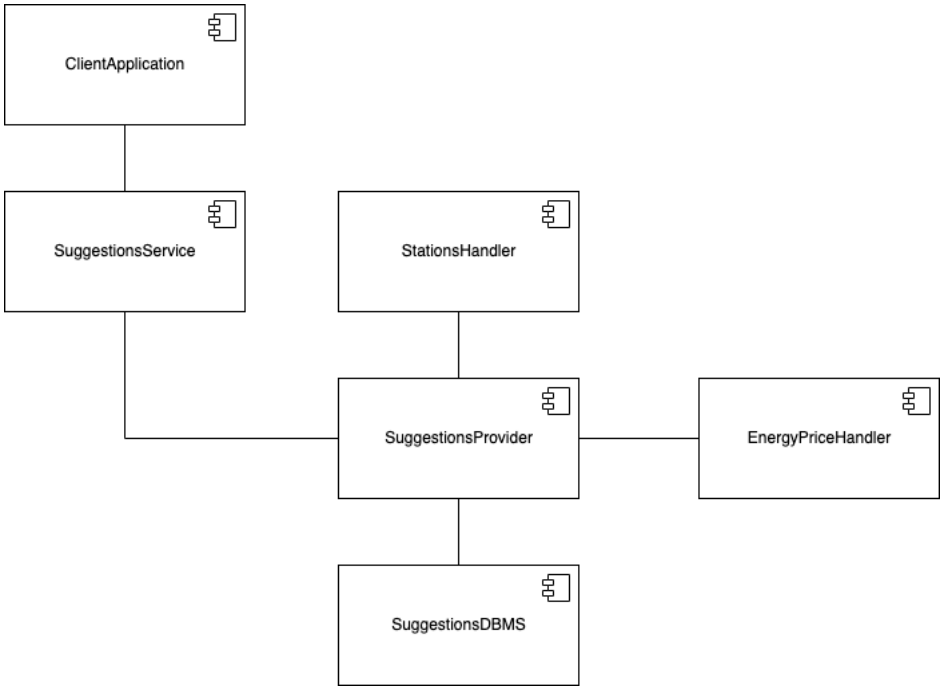


Figure 10: Final view of the components: ClientApplication is integrated into the system, replacing the last existing driver

When all the components have been integrated, a campaign of full testing on the system can be started, with the aim of making sure it fulfills all the functional and non-functional requirements specified in the RASD. As mentioned before, a beta testing campaign - to ensure that the user experience is bug-free and enjoyable - can be launched too.

6 Effort Spent

Student	Sect. 1	Sect. 2	Sect. 3	Sect. 4	Sect. 5
Claudio Arione	0h	18h	0h	2h	4h
Riccardo Begliomini	1h	11h	4h	2h	2h
Niccolò Bindi	1h	15h	1h	7h	2h

7 References

- https://www.europarl.europa.eu/news/en/headlines/society/20190313STO31218/co2-emissions-from-cars-facts-and-figures-infographics
- https://developer.ibm.com/articles/the-sequence-diagram/
- https://www.uml-diagrams.org
- https://restfulapi.net
- A Comparative Review of Microservices and Monolithic Architectures - 18th IEEE International Symposium on Computational Intelligence and Informatics