ITD
Software Engineering Project
A.Y. 2022-2023

Marco Ronzani, Alessandro Sassi

December 2022

Contents

1 Links to the software

Follow the link to the source code and to a .README file that explains how to install and run the former.

- **Source Code**: link here
- **Installation and usage instructions**: link here

2 Introduction

2.1 Purpose

Electric vehicles are starting to grow in number, and their takeover of combustion engines is bound to happen, consequently to support such a thriving trend adequate easy access to charging stations is of utmost importance. In this landscape the goal of eMall is to allow owners of electric vehicles to easily know where charging stations are and carefully plan their charging process according to their schedules at any such station.

This document will follow up on the RASD and DD documents presenting the result of the development of a demonstrative viable prototype of the eMall software, please note that nice-to-have features have been disregarded when they were deemed not relevant to the overall functionalities to demonstrate.

2.2 Scope

This IT document start from what was layed down in the DD document and the requirements discussed in the RASD, presenting the result of the development of a prototype of eMall's software, complete with also the software of a CPO (a CPMS) which eMall relies upon to interact with CSs. Such software consists of a WebApp supported by a set of APIs which is itself reliant on APIs exposed by the software of CPMSs.

2.2.1 Acronyms

Acronym	Full Name
eMall	Electric Mobility for All
eMSP	Electric Mobility Service Provider
CPO	Charging Point Operator
CPMS	Charge Point Management System
CS	Charging Station
DSO	Distribution System Operator
STB	System-To-Be
OS	Operating system
DB	Database

2.3 Reference Documents

1. The provided document describing the project: *Assignment IT AY 2022-2023*.

2. The provided document describing the project: *Assignment RDD AY 2022-2023-v3*.

3. The Software Engineering 2 course held by Prof.s Camilli Matteo, Di Nitto Elisabetta and Rossi Matteo Giovanni.

4. *ISO/IEC/IEEE 29148:2011(E)* standard for Requirement Engineering.

5. Project of last year provided as an assignment.

3 Implemented Functionalities and Requirements

The development of the prototype focused on the set of feature that were demanded in the first reference document for a group of two people.
Any requirement for which a justification is not provided regarding its exclusion from the prototype can be assumed to have been disregarded because outside of the following features *(Those are marked with -)*.

An eMSP shall offer to end users the possibility to:

1. know about the charging stations nearby, their cost, any special offer they have.

2. book a charge in a specific charging station for a certain time frame.

3. start the charging process at a certain station.

4. notify the user when the charging process is finished.

A CPMS shall offer the following main functions:

1. know the location and "external" status of a charging station (number of charging sockets available, their type such as slow/fast/rapid, their cost, and, if all sockets of a certain type are occupied, the estimated amount of time until the first socket of that type is freed).

2. start charging a vehicle according to the amount of power supplied by the socket, and monitor the charging process to infer when the battery is full.

3.1 Functionalities

Refer to the RASD document for detailed descriptions under section 2.2 .

Id	Functionality	Status
1	User registration	Implemented
1	Search for CSs	Implemented
1	Book a recharge at a CS	Implemented
1	Perform a recharge at a CS	Implemented (mocked CS)
1	Acquire price and energy mix information from DSOs	-
1	Choose prices, energy sources and battery usage policies for a CS	-
1	Allow the CPMS to operate automatically	-
1	Allow the CPO to change the "Automatic Mode" policy used by the CPMS	-

3.2 Requirements

Refer to the DD document for detailed descriptions under section 4 .

Id	Requirement	Status
R1	The eMSP shall allow a registered user to login	Implemented
R2	The eMSP shall allow an unregistered user to register on the platform	Implemented
R3	The eMSP should report to the user when they sent invalid input data through a form	Implemented
R4	The eMSP should report to the user when they attempted to perform an unauthorized action	Implemented
R5	The eMSP should report to the user when the platform encountered an error while processing the action	Implemented
R6	The eMSP should report to the user any successful action performed	Implemented
R7	The eMSP shall allow a logged-in user to view charging stations nearby	Implemented
R8	The eMSP shall allow a logged-in user to know the cost of a recharge at a CS	Implemented
R9	The eMSP shall allow a logged-in user to know about ongoing special offers present at a CS	Implemented
R10	The eMSP shall allow a logged-in user to book a recharge at a free socket of a CS for a given time slot	Implemented
R11	The eMSP shall allow a logged-in user to cancel any of its booked recharges before their scheduled time	Implemented
R12	The eMSP shall not allow a user to book a recharge at a socket of a CS which has already been booked for the chosen time slot	Implemented
R13	The eMSP shall not allow a user to book multiple recharges with overlapping time slots	Implemented
R14	The eMSP shall allow a logged-in user to see their bookings	Implemented
R15	The eMSP shall allow a logged-in user to start the charging process for one of their booked recharges	Implemented
R16	The eMSP shall not allow a user without a booking to start the charging process	Implemented

Id	Requirement	Status
R17	The eMSP shall not allow a user to start the charging process as long as the socket is vacant	Implemented
R18	The eMSP should notify a logged-in user when their ongoing charging processes is complete	Implemented
R19	The eMSP should notify a logged-in user when their ongoing charging process is terminated due to the expiration of its booked time slot	Implemented
R20	The eMSP shall allow a logged-in user to terminate their charging process earlier	Implemented
R21	The eMSP should charge a user for everyone of his charging processes as soon as they finish	Implemented (mockup)
R22	The eMSP shall charge a user for a fee if they do not start a recharge during their booking's time slot	-
R23	The eMSP shall be able to offer its functionalities to multiple users at once	eMSP: Implemented
R24	The CPMS should be able to acquire the location from CSs	Implemented (mocked via a fake CS client, with the data present anyway in teh CPMS's DB)
R25	The CPMS should be able to acquire the internal status from CSs	-
R26	The CPMS should be able to acquire the external status from CSs	Implemented
R27	The CPMS should be able to start charging a vehicle connected to a CS socket	Implemented (mocked via a fake CS client)
R28	The CPMS should be able to monitor the charging process of a vehicle	Implemented (mocked via a fake CS client)
R29	The CPMS should be able to acquire from DSOs their current energy prices	-
R30	The CPMS should be able to acquire from DSOs their energy mix	-
R31	The CPMS shall allow its CPO to authenticate with it	-
R32	The CPMS shall allow its CPO to assign a DSO to a CS as its energy provider when operating in "Manual Mode"	-

Id	Requirement	Status
R33	The CPMS should be able to automatically assign a DSO to a CS as its energy provider when operating in "Automatic Mode"	-
R34	The CPMS shall allow the CPO to decide the energy source management policies when operating in "Manual Mode"	-
R35	The CPMS should be able to automatically choose the current energy source for a CS according to the given policy when operating in "Automatic Mode"	-
R36	The CPMS shall allow its CPO to assign a nominal-price, a user-price and an offer reset date to a CS when operating in "Manual Mode"	-
R37	The CPMS should be able to automatically assign a nominal-price, a user-price and an offer reset to a CS when operating in "Automatic Mode"	-
R38	The CPMS shall allow the CPO to choose its policy for operating in "Automatic Mode"	-
R39	The CPMS shall allow the CPO to choose whether it acts automatically or manually	-
R40	The CPMS shall never take a decision automatically when it is set in "Manual Mode"	-
R41	The CPMS should report to the user when they sent invalid input data	Implemented
R42	The CPMS should report to the user when they attempted to perform an unauthorized action	Implemented
R43	The CPMS should report to the user when the platform encountered an error while processing the action	Implemented
R44	The CPMS should be able to decide whether to start charging the internal batteries of a CS (if present)	-

4 Adopted development frameworks

As expected of every modern Web Application supported by a backend API, its software architecture is divided in a Model and Routes for the backend and a framework for the frontend which allows for both a logic and presentation layer in order to offer more advanced functionalities that a plain HTTP page. In the prototype here discussed keep in mind that there is a further distinction between the CPMS's backend and the eMSP's one, when neither is mentioned explicitly, what follows refers to both.

4.1 Adopted programming languages

4.1.1 TypeScript

Both backends and the frontend are primarily written in **TypeScript**, a superset of Javascript that adds to the former static typing and optional classes and interfaces, resulting in better type safety and less mandelbugs. TypesScript unfortunately offers a considerably smaller set of libraries compared to JavaScript, but nonetheless did not hinder the capabilities of the final prototype.

4.1.2 SQL

The interactions with the database that occur within the model are realized by preparing and executing SQL statements, such queries make extensive use of MySQL's dialect of SQL, requiring no redundant storage of data on the database and a simple way to recover from it only what is necessary.

4.1.3 HTML and CSS

4.2 Adopted middleware

4.2.1 Express.js as a Request Router for the backend

Express is a free and open-source web framework for building APIs with a comprehensive set of features. Its approach to middleware and routing, which involves adding functions to endpoint declarations, helps improve code scalability. The use of functions as middleware also promotes modularity in the application by allowing endpoints to be added through corresponding functions for each route.

4.2.2 Database connection with MySQL2

MySQL is a set of modules for working with the MySQL database, it is a typescript version of the popular MySQL package. This library was chosen for its excellent pooling system and the possibility of preparing statements before executing queries, where the query string is passed directly to the database and parameters are replaced there, preventing the risk of SQL injection that can result from string concatenation on the server.

4.2.3 Connection to the CS via WebSocket

The WebSocket library for JavaScript (ws) enables real-time, two-way communication between a client and a server, it provides a simple and efficient API for creating and managing connections, supporting both server-side and client-side use.

4.2.4 Http requests via the Axios library

Axios is JavaScript library for making HTTP requests in Node.js, it is promise-based and enables sending HTTP requests to access REST APIs and other web services. Axios abstracts away some of the complexities of making HTTP requests and provides a simple, straightforward API for handling requests and responses.

4.3 Utilized APIs (not present in the DD)

4.4 Platforms and Libraries

4.4.1 Node.js

Node is an open-source, cross-platform, back-end JavaScript (and by extension TypeScript) runtime environment which offers many stable and well known libraries (ex: Express). It allows the execution of JavaScript outside of a web browser and as act as a server, is well-suited for building responsive applications, such as eMall, due to its ability to handle multiple simultaneous connections with low latency. Its only drawback is its single-threaded model, resulting in poor performances when dealing with CPU-intensive tasks, but that is never the case with the present prototype.

4.4.2 Authentication with JWT and BCrypt

JWT (JSON Web Token) is a javascript implementation of the JSON Web Token standard, which provides a secure and URL-safe method of transferring user information between two parties.

BCrypt is the Javascript implementation of the bcrypt hashing function based, a standard to hide passwords in a web environment. It provides a simple and secure way to hash passwords and compare them during authentication.

4.4.3 Testing with Mocha and Sinon-Chai

Mocha is a JavaScript testing framework that works with Node.js. It provides a simple, flexible, and fast environment for writing and executing test cases for applications and libraries.
Sinon-Chai is comprised of the two libraries Sinon and Chai, the first allows for testing via stubs and mocks of objects and functions, the second complements it with an easy way to write assertions.

4.4.4 Frontend with Vue

5 Source Code Structure

5.1 Code Structure

As mentioned in the DD document, the structure of the code is made to reflect the Model View Controller (MVC) pattern. Briefly:

- **Model**, it contains the logic that retrieves from the database the data that needs to be show to the user, as well as classes used to incapsulate the data retrieved. Ad-hoc versions of the model exist for the frontend and backend, when the former is present.
  More precisely there are 3 different models, one for the eMSP's backend, one for its frontend, and one for the CPMS's backend.

- **View**, its goal is to show the data of the model to the user, formatted in a human-friendly way. The view is realized with the above-mentioned Vue framework and is present only for the eMSP.

- **Controller**, logic in charge of using the Model to feed the View and respond to user inputs, it is comprised of the client-side modules used to asynchronously request and send data to the backend, but also the Server-side components handling all API routes.

Reflecting such structure is the hierarchy of directories within the source code, atleast for what concerns the backend of the CPMS and eMSP, with the added paths being for tests and helper functions:

- *src/routes* are the express routes realizing the aforementioned controller.

- *src/model* contains all the classes for data representation and retrieval from the database.

- *src/test* contains, following the same hierarchy as the **src** directory, the tests for the entire backend.

- *src/helper* consists of some miscellaneous utility functions, each commented in detail.

The frontend follows the a structure not so different from the two backends, with the notable additions of:

- *src/components* stores Vue components, which are reusable user interface elements.

- *src/controllers*, contains JavaScript files that define the behavior and logic of the web application, interacting with the Vue instance and the backend.

- *src/views* contains components that define the layout and structure of the frontend's web pages, those define the application's routing as well.

5.2 Components in the Source Code

What follows is a brief description of the realized components, refer to the DD for a more complete overview of the components and modules involved.
The only component present here, not previously discussed as such in the DD is the "bank" component, it simply simulates the "payment provider" extenal module mentioned in the DD.

- **The CPMS component** is comprised of the Request Router (Web Server) for the CPMS realized with Express and the CPMS Application Server, the latter containing every module realizing API endpoints and Data Access Models.
  The CPMS accepts connections via WebSockets from the CSClients and allows known eMSPs to interact with its charging stations, from simply searching for them to managing a charging process ongoing at a CS. To simulate a real CS, we realized a mockup which connects to the CPMS as expected, but instead of allowing a vehicle to be charged exposed a CLI to manually simulate such actions.

- **The EMSP component** is comprised of the Request Router (Web Server) for the EMSP realized with Express, the EMSP Application Server, the latter containing every module realizing API endpoints and Data Access Models, and the EMSP Web Application, which is the fronted of the component, seen by the end user. The EMSP allows users to register and login, as well as reserve and operate (start/end charge) bookings with any of the CSs known to the EMSP. To know about CSs and manage them on behalf of the user the EMSP knows about multiple CPMSs and reaches their APIs to fulfill its tasks. Furthermore the EMSP can reach the external BANK component to verify credit cards validity and charge users for the offered service.

- **The BANK component** was only mentioned in the DD as an external component, here it exists only as a mockup to allow a realistic simulation of how a payment would happen on the EMSP's end. It is comprised of a Request Router realized with Express and a simple Application Server, exposing a simple endpoint which allows to both verify the correctness of payment information and to actually charge the owner of those information for a certain amount. The component has neither be tested nor commented due to its extreme sempilicity as a mockup.

6   Testing

See the DD for a more detailed description of the following process.

6.1   Integration Testing

Following what was discussed in the DD document, integration testing has been performed during each phase of the development, which proceeded in parallel for both the CPMS and eMSP backends, with their final integration with one-another occurring only when both were already fully ready.
Internally each backend had its development start from the database, then data access modules (the previously discussed Models) followed by the rest of the control logic (the Routes). After the routes went through testing and were complemented with the authentication capabilities the final set of integration test could be done, before moving on to the frontend development and finally full integration tests.

In more detail every component's API has been tested with PostMan (https://www.postman.com/) and MySQL WorkBench (https://www.mysql.com/products/workbench/), in order:

1. After constructing of each component's database and Model, the utilized queries were tested with MySQL WorkBench by:

    - Running the queries directly from WorkBench, with dummy DB entries and query parameters.

    - Running the queries from the Model, ran withing Node,js, using dummy mains, follows an example of such tests for the "CS.ts" model file within the CPMS's backend:
      ```
      CS.getCSList([30, 140], 20, [2, 20]).then( async (result) => { console.log(result);
      await DBAccess.closePool();}
      ```
      This has been done together with active logging within the database instance to ensure the correct format of parameters inserted into prepared statements:
      ```
      SET GLOBAL log_output = 'TABLE';
      SET GLOBAL general_log = 'ON';
      SELECT * FROM mysql.general_log;
      ```

2. Each realized API interface has been tested during and after construction, together with the Model it relied upon, with PostMan, by sending to it HTTP requests and verifying both the correctness of responses and the tolerance of the API to malformed requests. Follow some examples of the performed tests:
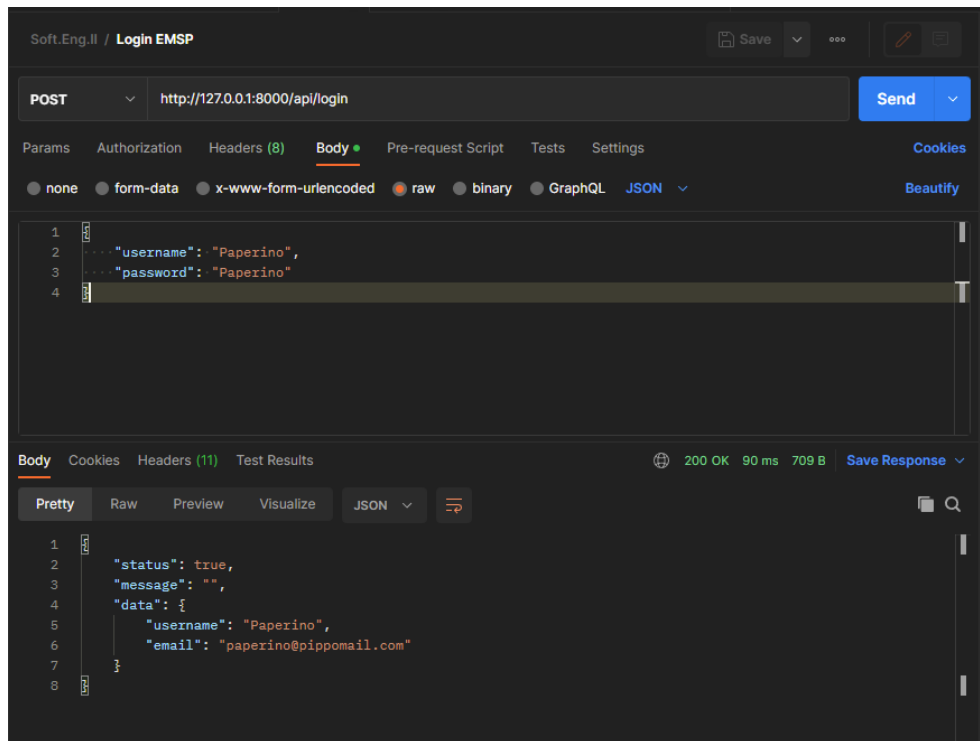
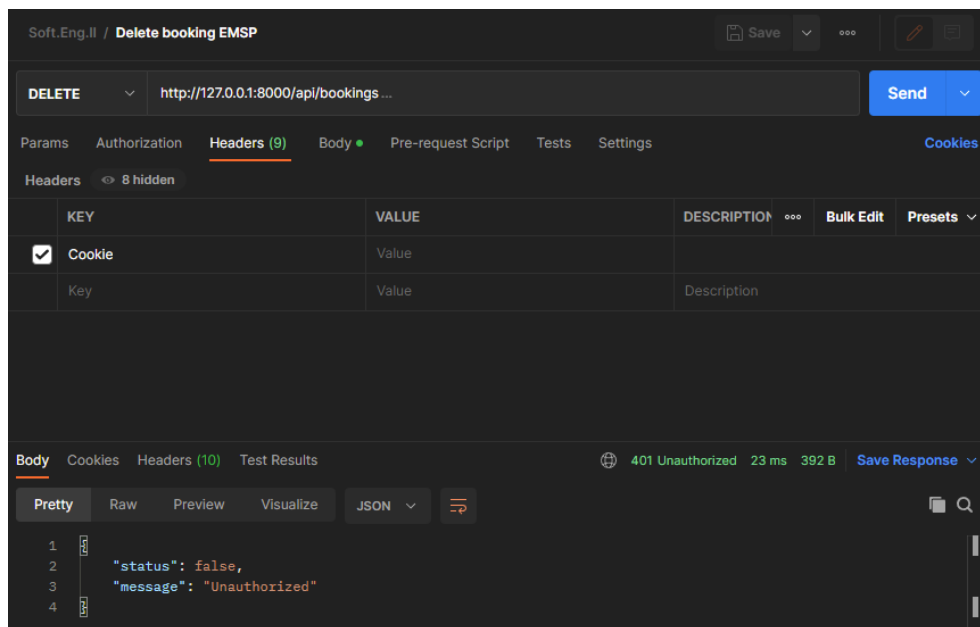Figure 1: Testing of the EMSP Login endpoint, resulting in a success

Figure 2: Testing of the EMSP Bookings endpoint, a attempt to access any booking before being logged in fails
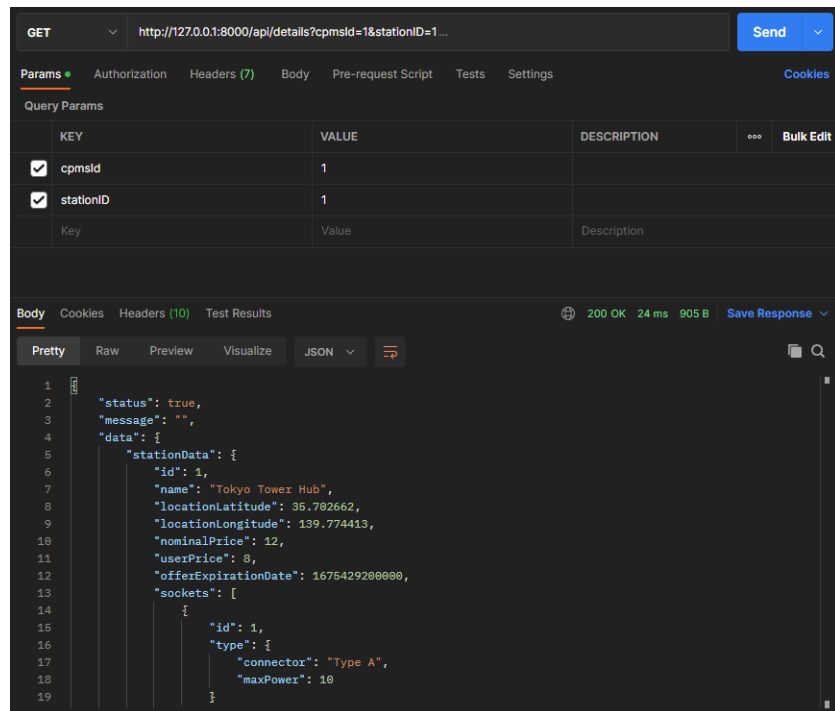
Figure 3: Testing of the EMSP Details endpoint, a logged in user received an object describing the requested CS
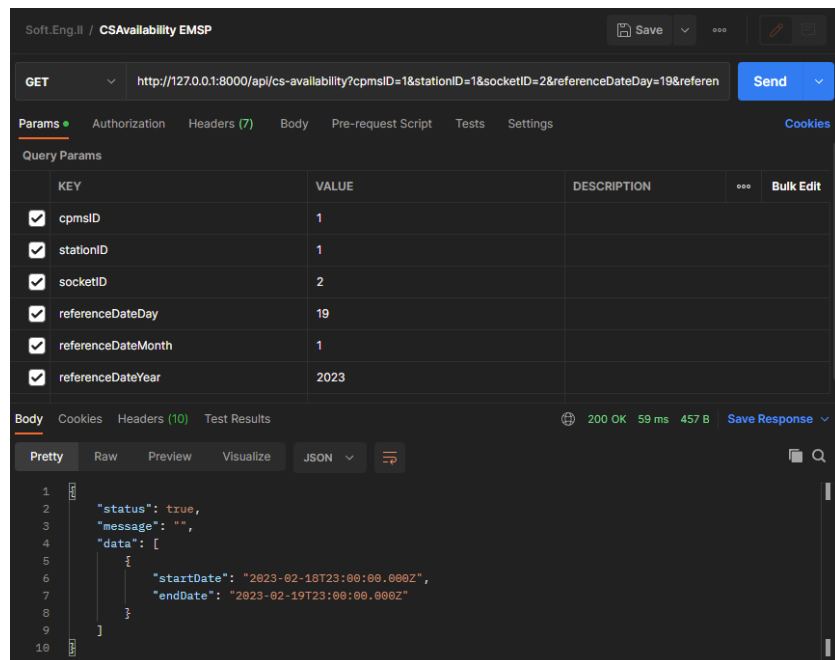

Figure 4: Testing of the EMSP Availability endpoint, it correctly responds to a logged in user with the intervals in which the specified CS is available for the provided date
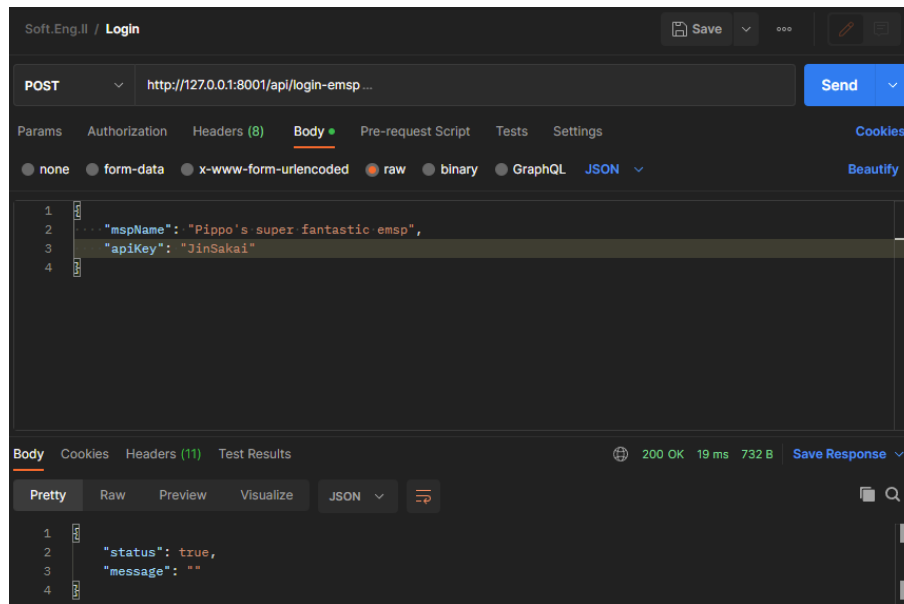
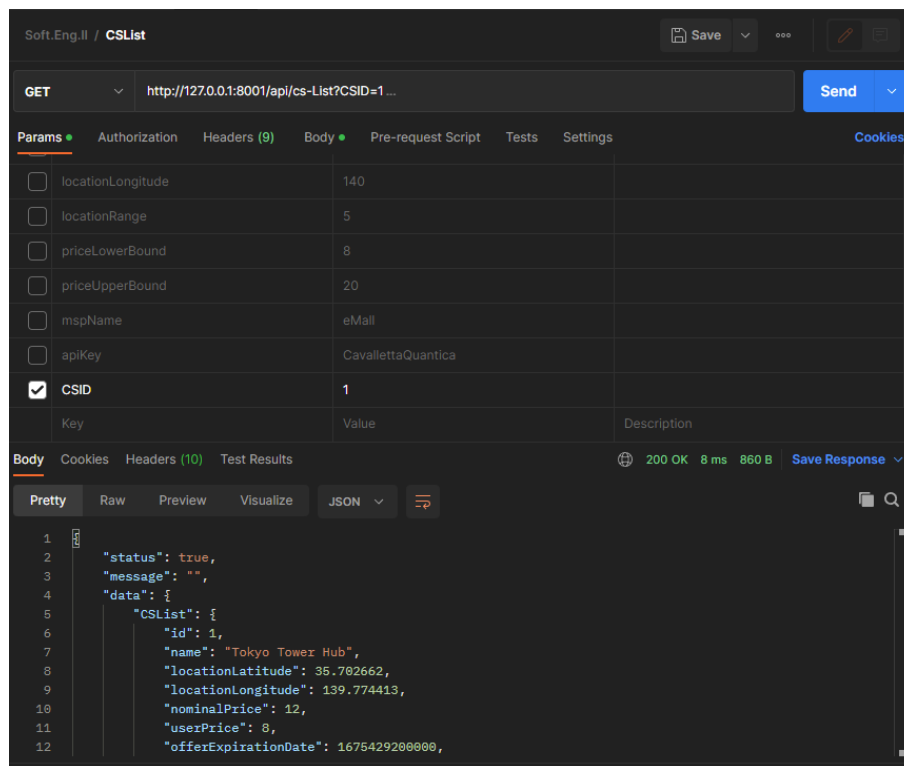Figure 5: Testing of the CPMS Login-emsp endpoint, resulting in a success

Figure 6: Testing of the CPMS CS-list endpoint, that returns to an authenticated EMSP a filtered list of charging stations information

6.2 Automated Testing

6.2.1 Backend Tests

Automated tests have been written for every route with the aid of Sinon-Chai, during those tests the interactions with the database are always stubbed with SinonStub instances that replace every connections to the DB, returning predefined results that consequently not only test the behaviour of the routes, but of the Model as well. Every route of the CPMS and EMSP has been fully tested, no tests are present for the bank as it is just a mockup.

To run the tests, assuming that you already followed the **Installation instructions** in the section below, you can follow those instructions:

1. **Run the CPMS's tests** by reaching its location, `YourProjectRoot/cpms/backend`, with a console and running the command: `npm run test`.

2. **Run the eMSP's tests** by reaching its location, `YourProjectRoot/emsp/backend`, with a console and running the command: `npm run test`.

These will run the npm scripts to execute the tests.
You can also produce coverage reports (a pre-evaluated instance is already included in the project) with the following commands:

1. **Evaluate the CPMS's coverage** by reaching its location, `YourProjectRoot/cpms/backend`, with a console and running the command: `npm run coverage`.

2. **Evaluate the eMSP's coverage** by reaching its location, `YourProjectRoot/emsp/backend`, with a console and running the command: `npm run coverage`.

The coverage reports can be view by opening `YourProjectRoot/cpms/backend/coverage/index.html` and `YourProjectRoot/emsp/backend/coverage/index.html`.
At the time of writing the coverage results are as follow:

src		65.3%	64/98	43.33%	13/30	50%	9/18	65.62%	63/96
src/helper		55.37%	103/186	33.73%	28/83	50%	15/30	55.67%	103/185
src/model		47.32%	106/224	33.8%	24/71	43.07%	28/65	49.29%	105/213
src/routes		97.58%	121/124	89.74%	70/78	100%	8/8	97.47%	116/119
src/routes/eMSP Authentication		100%	25/25	100%	6/6	100%	4/4	100%	24/24

Figure 7: Coverage of the CPMS

src		77.14%	54/70	50%	12/24	63.63%	7/11	76.81%	53/69
src/helper		74.41%	128/172	56.32%	49/87	76.92%	20/26	74.7%	127/170
src/model		82.89%	126/152	80.76%	42/52	66.66%	22/33	84.56%	126/149
src/routes		89.85%	301/335	80.1%	149/186	100%	18/18	90.27%	297/329
src/routes/authentication		95.77%	68/71	80.76%	21/26	100%	6/6	95.65%	66/69

Figure 8: Coverage of the EMSP

For a more detailed look at the coverage we suggest opening the index.html files directly and navigating the report.

Note that the coverage of the CPMS could not be improved due to the presence of the charging station mockup and its materialization in the CPMS's backend, CSConnection, both of which cannot be

tested due to their complete reliance on WebSockets and a live connection for their functionalities, hence any test would have just consisted of stubs interacting with one-another.

6.2.2 Frontend Tests

Some automated tests were also written for the frontend, in order to verify the correct behaviour of Vue controllers.

Assuming that you have already followed the **Installation instructions** in the section below, you can run those tests as well as produce a coverage report for the fronted with them with the following steps:

- **Run the fronted's tests** by reaching its location, `YourProjectRoot/emsp/fronted`, with a console and running the command: `npm run test`.

- **Evaluate the frontend's coverage** by reaching its location, `YourProjectRoot/emsp/fronted`, with a console and running the command: `npm run coverage`.

Npm scripts will then take care of running the tests and the coverage reports can be view by opening `YourProjectRoot/emsp/frontend/coverage/index.html`.
At the time of writing the coverage results are as follow:

Figure 9: Coverage of the CPMS

7 Installation instructions

7.1 Prerequisites

- **MySQL Server** must be installed on your system, you can download it from here:
  https://dev.mysql.com/downloads/installer/.
  For installation instructions for Windows you can refer to this guide:
  https://dev.mysql.com/doc/refman/8.0/en/windows-installation.html.
  Make sure to take note of your root password during installation, it will be useful later.

- **MySQL WorkBench** is not necessary, but recommended to allow you to import the provided database dump into your instance of MySQL Server (which is needed in order to properly test the behavior of the system). You can download it from here:
  https://dev.mysql.com/downloads/workbench/
  After installing WorkBench, connect it to your local instance of MySQL Server (which must be up and running for this to work): first choose to create a new connection (+ button near "My Connections"), then set its host to "localhost" and port to "3306". Your username for the connection is recommended to be "root", and its password the one chosen during the installation of MySQL Server. Note that you can use other user accounts as long as you give them high enough privileges to import a schema and operate on it at will.

- **Node.js** must be present on your system. The project was built and tested using version 18.12.0, which you can download here:
  https://nodejs.org/download/release/v18.12.0/
  Make sure that node is present in your system path.

7.2 Preparing the Project

1. **Download the Project**: download the ".zip" file provided with the project (It is located under "/DeliveryFolder" on GitHub) and extract it in a known location. As an alternative, you can also download the entire project repository using the "Clone" button provided by GitHub.

2. **Import the schemas in the DB**: open MySQL Workbench and open the DB connection you previously configured. In the left-side menu select "Administration" and then "Import Data/Restore". The screen to import database dumps will open and you will need to select "Import from Self-Contained File" and choose as the file to import `YourProjectRoot/DBs_dump.sql`, and then choose to "Start the Import".
   This will create in your MySQL local instance of the `emsp_db`, `cpms_db` and `bank_db` schemas.
   If you can't access the newly imported schemas after "Schemas" on the left menu, close and reopen WorkBench.

3. **Run NPM** to download all the node packages required by the project. To do this open a terminal and reach each of the following paths within the extracted project folder, then in each path run:
   `npm install`
   List of paths:

   - `YourProjectRoot/cpms/backend`

   - `YourProjectRoot/bank/backend`

   - `YourProjectRoot/emsp/backend`

   - `YourProjectRoot/emsp/frontend`

This should fetch all node modules listed in "package.json" for each of the 4 directories containing the CPMS's backend, the bank's backend, the eMSP's backend and the eMSP's frontend.

4. **Configure the ".env" files**, there are 3 ".env" files you need to configure to reflect your MySQL installation, they are located in:

   - `YourProjectRoot/cpms/backend`

   - `YourProjectRoot/bank/backend`

   - `YourProjectRoot/emsp/backend`

   The only needed change is to the field "DB_PASSWORD", which should be set to the root password you chose while installing MySQL Server.
   If you also are using a different user than "root", you can account for it by changing the "DB_HOST" accordingly field too.
   Note: not quotes required.

7.3   Running the project

Run each of the following programs, in the specified order (not a must, but highly suggested).

1. **Run the CPMS's backend** by reaching its location, `YourProjectRoot/cpms/backend`, with a console and running the command: `npm run run`.

2. **Run the CS mockup**, for any CS you wish to emulate of the 9 available ones, by reaching its location, `YourProjectRoot/cpms/backend`, with a console and running the command:
   `npm run run-cs [ID]`, where "ID" is a number from 1 to 9 (extremes included) indicating the charging station ID. Refer to the following figure for which id is of which charging station.

id	name	locationLatitude	locationLongitude	nominalPrice	userPrice	offerExpirationDate	imageURL
1	Tokyo Tower Hub	35.702662	139.774413	12.00	8.00	1675429200000	http://www.widest.com/wp-content/uploads/Tokyo-Tower-in-Tokyo-Japan.
2	Akiba Bolt	35.702662	139.776447	11.00	11.00	<null>	https://www.japan-guide.com/g18/3003_01.jpg
3	Deib	45.478611	9.232778	5.00	5.00	<null>	https://lh3.googleusercontent.com/p/AF1Qip0YluQw6pCBFtNuBFmJ70r_0-hh
4	Building 26	45.475833	9.234444	5.50	5.50	<null>	https://lh3.googleusercontent.com/p/AF1Qip0-0rEIQ-aLw_Q327T_g07jHcuo
5	Linate 1	45.460278	9.278611	8.00	8.00	<null>	https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fcdn.busi
6	Linate 2	45.458056	9.282222	8.00	8.00	<null>	https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fwww.want
7	Duomo	45.464167	9.191667	3.00	3.00	<null>	https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Flive.sta
8	CityLife	45.477778	9.155833	6.00	6.00	<null>	https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fmedia-cd
9	Sempione	45.475833	9.172222	6.00	6.00	<null>	https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fak.jogur

   You can run multiple mockups by opening multiple console instances and repeating the above procedure, however do not run two mockups on the same charging station ID.

3. **Run the bank's backend** by reaching its location, `YourProjectRoot/bank/backend`, with a console and running the command: `npm run run`.

4. **Run the eMSP's backend** by reaching its location, `YourProjectRoot/emsp/backend`, with a console and running the command: `npm run run`.

5. **Run the eMSP's frontend** by reaching its location, `YourProjectRoot/emsp/frontend`, with a console and running the command: `npm run dev`.

7.4   Intercting with the running project

1. **Reach the WebApp** by opening the URL printed during the frontend's startup, which should be `http://localhost:5173/`.

2. **Register an account** by following the register link on the login page, you will be asked with the usual information, plus payment details, a valid set of such details is:

   - **Card Number**: 4365875436666669

- **Owner**: MARIO LUIGI

- **Expiration Date**: 11/22

- **CVV**: 123

Those are obviously stand-ins allowed by the "bank" we simulated, more can be added by interacting with the "bank_db" schema in WorkBench.
For sake of semplicity a single bank account can be used for any number of accounts.

3. **Login** by simply going back to the login page and inserting the credentials you just created.

4. **Navigate the WebApp**, you can now freely navigate the WebApp by using the menu on the top right, comprising of:

   - **The Bookings Tab**, where you can see all your current bookings and delete some, as well as manage any booking that is currently active (start/stop charge). Some of those actions can be directly performed from the home page to improve accessibility.

   - **The Charging Stations Tab** allows you to search for CSs with the aid of a map, click on any CS available in the shown list to see its details and book it if you desire.

5. **Manually operate the a CS mockup** by utilizing the following commands:

   - `connectCar [socketId]` : simulates a car connecting to the given socket.

   - `disconnectCar [socketId]` : simulates a car disconnecting from the given socket.

   - `fullyCharge [socketId]` : simulates a car charging completely over the given socket.

   - `quit` : closes the program.

   - `help` : to print those instructions in the program.

8 Effort Spent

8.1 Ronzani Marco - mat: 224578

Task	Time spent
Backend (APIs and Models)	$25h$
Frontend	$0h$
Tests	$16h$
Documentation	$4h$
This document	$4h$
Total	$49h$

8.2 Sassi Alessandro - mat: 220837

Task	Time spent
Backend (APIs and Models)	xh
Frontend	xh
Tests	xh
Documentation	xh
This document	xh
Total	xh

9 References

1. ChargeLab - operating system for EV charges

2. Platform for Electromobility. EV Charging: How to tap in the grid smartly?.

3. F. Campos, L. Marques, and K. Kotsalos, Electric Vehicle CPMS and Secondary Substation Management.

4. Shu Su, Hui Yan, and Ning Ding. 2018. Machine Learning-Based Charging Network Operation Service Platform Reservation Charging Service System. In Proceedings of the 2018 International Conference on Signal Processing and Machine Learning (SPML '18). Association for Computing Machinery, New York, NY, USA, 1–5.

5. Jan Mrkos, Antonín Komenda, and Michal Jakob. 2018. Revenue Maximization for Electric Vehicle Charging Service Providers Using Sequential Dynamic Pricing. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '18). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 832–840.