Software Engineering 2 - Prof. Di Nitto Elisabetta
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

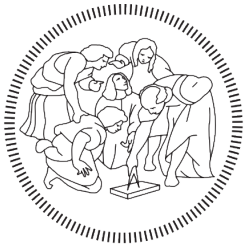
eMall - e-Mobility for All

ITD
Implementation and Testing Deliverable

February 5, 2023

Giovanni De Lucia (10700658)
Lorenzo Battiston (10618906)
Matteo Salvatore Currò (10940719)

Repository: `https://github.com/gio-del/BattistonDeLuciaCurro-swe2`



POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

This document outlines the implementation and testing procedures that have been followed to develop a functioning prototype of the service described in the "Requirements Analysis and Specification Document" and "Design Document". It is intended to serve as a reference for the development team, detailing the software, frameworks, and programming languages chosen.

1.2 Definitions, Acronyms and Abbreviations

1.2.1 Definitions

- EV Driver - Electric Vehicle Driver, people or entities who own an EV car and want to use the system for their charging needs. In this document they can be also referred to as "users".
- EVCP - Electric Vehicle Charging Pool, is a station with multiple CPs
- CP - a synonym of EVSE - is a single charging column with multiple connectors
- Connector (Socket) - charging socket that can be of different types (e.g. CCS2, Type2)
- Rate - the rate that the CPO decides to set for the CPs it manages. It contains a fixed part for parking and a variable part per kWh. Usually the rates are associated with a certain power (kW)

1.2.2 Acronyms

RASD	Requirement Analysis and Specification Document
DD	Design Document
API	Application Programming Interface
CPO	Charging Point Operator
DSO	Distribution System Operator

1.2.3 Abbreviations

$\mathbf{R}_x$	x-Functional Requirement

1.3 Revision history

Revised on	Version	Description
31-Jan-2023	1.0	Initial Release of the document

1.4 Reference Documents

- Requirement Analysis and Specification Document (referred to as "RASD" in the document)
- Design Document (referred to as DD in the document)
- Assignment document A.Y. 2022/2023 ("Requirement Engineering and Design Project: goal, schedule and rules")

1.5 Document Structure

This document is composed of six sections:

- Introduction

- Product Functions: presents the implemented and discarded functions of the prototype.
- Development Frameworks: presents the adopted programming languages and frameworks, justifying each choice.
- Code Structure: presents the structure of the code
- Testing: explains how and what has been tested
- Installation Guide: provides explanations on how to run, test and build the prototype

2 Product functions

In this section, we present the implemented and discarded functional requirements of the prototype.

2.1 CPO functions

- **R1** The system must allow unregistered CPO to register an account
  Implemented: Yes, in the form of a registration page
- **R2** The system must allow registered CPO to login
  Implemented: Yes, in the form of a login page
- **R3** The system must allow authenticated CPOs making a special offer on their CPs prices
  Implemented: Yes, it is possible to make a special offer on the price of an entire EVCP in the Rate tab of the CPO dashboard
- **R4** The system must allow authenticated CPOs monitoring the charging process to infer when the battery is full
  Implemented: No, it is not possible to monitor the charging process. Although we have implemented a simple socket simulator, it is not possible because it would need sensors to monitor the charging process
- **R5** The system must allow authenticated CPOs retrieving the amount of energy available in their EVCPs batteries
  Implemented: No, it is not possible to retrieve the amount of energy available in the batteries of the EVCPs. We made a mocked 'Energy API' that returns a random value given a key, but it is not possible to retrieve the actual value of the batteries
- **R6** The system must allow authenticated CPOs retrieving the number of vehicle being charged in their EVCPs and for each vehicle the amount of absorbed power
  Implemented: +/-, it is possible to retrieve the number of vehicle being charged in their EVCPs but for the same reasons as R4, it is not possible to retrieve the amount of energy being absorbed
- **R7** The system must allow authenticated CPOs retrieving the remaining charge time for each connected vehicle
  Implemented: +/-, it is possible to retrieve the end of a reservation but not an estimation of the remaining charge time, for the same reasons as R4
- **R8** The system must allow authenticated CPOs retrieving details on active and historical reservations on their EVCPs
  Implemented: Yes, in the reservations tab of the CPO dashboard
- **R9** The system must allow authenticated CPOs acquiring information from the DSOs about the current price of energy
  Implemented: Yes, via a mocked 'DSO API' in the Energy tab of the CPO dashboard
- **R10** The system must allow authenticated CPOs deciding from which DSO to acquire energy from
  Implemented: Yes, via a mocked 'DSO API' in the Energy tab of the CPO dashboard
- **R11** The system must dynamically decide where to get energy for charging (electrical grid, battery or a mixture)
  Implemented: No
- **R12** The system must allow authenticated CPOs statically deciding where to get energy for charging (electrical grid, battery or a mixture)
  Implemented: +/-, via the Energy API and the battery key it is possible to set an operating mode for the battery
- **R13** The system must allow authenticated CPOs adding, modifying and deleting CPs
  Implemented: Yes, in the Charging Points tab of the CPO dashboard it is possible to create EVCP, adding CP to them and then sockets to the CP
- **R14** The system must allow authenticated CPOs changing availability status of their CPs
  Implemented: No

2.2 Driver functions

- **R15** The system must allow unregistered users to register an account
  Implemented: Yes, in the form of a registration page
- **R16** The system must allow registered users to login
  Implemented: Yes, in the form of a login page
- **R17** The system must allow authenticated users to personalize their experience by providing information of their EV
  Implemented: No, it was not requested, and it would have slowed down the development process
- **R18** The system must allow users to search for CPs in the map
  Implemented: Yes, in the map tab of the driver
- **R19** The system must show to the users CPs nearby their current position
  Implemented: Yes, in the map tab of the driver
- **R20** The system must allow retrieving details on a given CP regarding connector types supported and cost of the charge
  Implemented: Yes, in the map tab of the driver by clicking on a marker representing an EVCP
- **R21** The system must allow authenticated user to book a CP for a certain time interval
  Implemented: Yes, in the map tab of the driver by clicking on a marker representing an EVCP and then on the book button
- **R22** The system must allow booking a CP if and only if it is free for the specified time interval
  Implemented: Yes, via a check on the availability of the EVCP in the backend
- **R23** The system must notify users when the charging shift is about to start
  Implemented: No, it was not requested, and it would have slowed down the development process
- **R24** The system must allow authenticated users to start the charge
  Implemented: Yes, in the reservation tab of the driver
- **R25** The system must suggest users when and where to charge based on daily schedule, special offers and availability
  Implemented: No, it was not requested
- **R26** The system must allow authenticated users to monitor the charging status
  Implemented: +/-, it is possible to monitor the charging status but not the actual energy being absorbed, for the same reasons as R4
- **R27** The system must notify authenticated users when the charging process is completed
  Implemented: Yes, via Firebase Cloud Messaging and a cron job that checks the status of the reservation every 5 minutes
- **R28** The system must allow authenticated users to pay for the charge
  Implemented: No, it was not requested
- **R29** The system must allow authenticated users to delete a reservation
  Implemented: No, it could have been implemented, but it was not requested, and it would have slowed down the development process
- **R30** The system must allow authenticated users to view historical reservations
  Implemented: Yes, in the reservation tab of the driver

3 Development Frameworks

3.1 Programming Languages

3.1.1 JavaScript

The programming language JavaScript was used for both the backend and frontend of the project. The choice was made due to its widespread use in the industry and our familiarity with the language.

JavaScript is a multi-paradigm language that is easy to learn, making it suitable for prototypes and small projects. However, it may not be the best choice for large projects or performance-intensive applications and does not offer type safety.

Pros:

- Widely used in the industry
- Easy to learn
- Multi-paradigm language
- Good for small projects
- Good for prototyping

Cons:

- Not suitable for large projects
- Not ideal for performance-intensive applications
- Not type-safe

Although **TypeScript** offers improved type safety compared to JavaScript, it was not used in this project. The main reason was that it is more verbose, which slows down the development process. The biggest disadvantage of JavaScript is its lack of type safety, making TypeScript a better choice for large projects. However, the trade-off of a slower development process outweighed the benefits for this project.

3.2 Framework and Libraries

3.2.1 Node.js

Node.js is a runtime environment that allows the execution of JavaScript code on the server-side. We adopted it as it is a popular and widely used framework with a large community of developers who provide useful packages and libraries through `https://www.npmjs.com`.

3.2.2 Vite

Vite is a fast build tool used for bundling our code and serving it to the browser. It is a good fit for our project as it is faster than alternatives such as Webpack and comes with a development server that features real-time updates thanks to its HMR (Hot Module Replacement) capability.

3.2.3 React

React is a widely used JavaScript library for developing user interfaces. It is ideal for creating prototypes as it enables the creation of fast and responsive single-page applications (SPAs). React's modular design makes it easy to create reusable components, contributing to clean and maintainable code.

3.2.4 Tailwind CSS

Tailwind CSS is a utility-first CSS framework that supports mobile-first, responsive design. It facilitates fast styling prototyping for the application. However, it is not a framework and does not provide reusable components. Nevertheless, it can be combined with React to create styled and reusable components.

3.2.5 Jest

Jest is a testing framework that enables us to test our code. It can be used to test both the backend and frontend, but we focus on testing the main functionalities of the backend. Jest also provides useful reports, including line and branch coverage.

3.3 API

3.3.1 Firebase Cloud Messaging (FCM)

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that allows us to send push notifications to our users. We use it to notify the driver when a charge has been completed.

3.3.2 Other APIs

Other APIs have been mocked to simulate the system's real behavior. For example, the API that retrieves the list of available DSOs for CPOs or the SMS and email API. This approach was taken as some of these APIs are not freely available, while others are unavailable altogether.

3.4 Other Tools

3.4.1 Postman

Postman is a comprehensive tool for testing and documenting APIs. It allows us to easily test our backend APIs without the need to write any client code, making it an efficient tool for development and testing purposes.

3.4.2 Docker

Docker is a powerful tool for creating and managing containers, which can be used to run applications in a production environment. We have chosen to use Docker in our project as it provides an easy and reliable way to deploy the application, while also reducing inconsistencies between different operating systems. With Docker, it is possible to easily manage the backend and frontend of our application in a containerized environment.

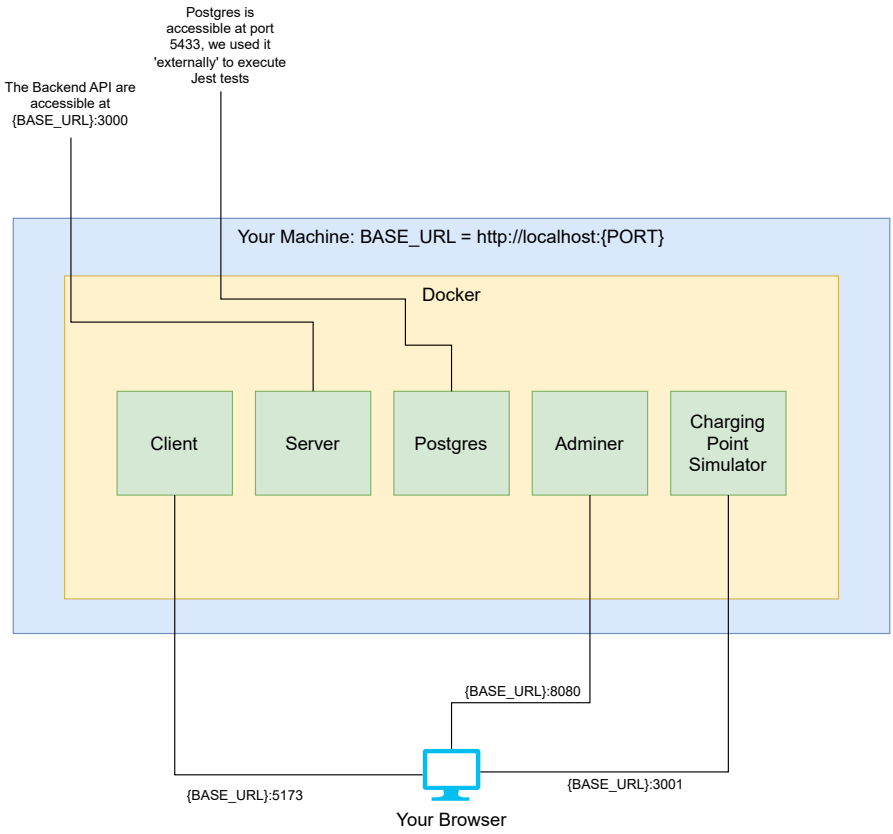


Figure 1: Internal structure of the container.

3.4.3 Jsdoc

Jsdoc is a tool used to document our JavaScript code. It helps to generate clear, comprehensive and well-organized documentation of the backend of our application. This ensures that the code is easy to understand and maintain, especially for other developers who may work on the project in the future. With Jsdoc, we can also provide information such as function signatures, parameter descriptions, and return types, making it easier to use and debug the code.
The generated documentation is in the /docs folder of the server.

4 Source Code Structure

In this section we describe the structure of the source code of the application. To be more clear, we will describe only the most important parts of the code and the relevant files.

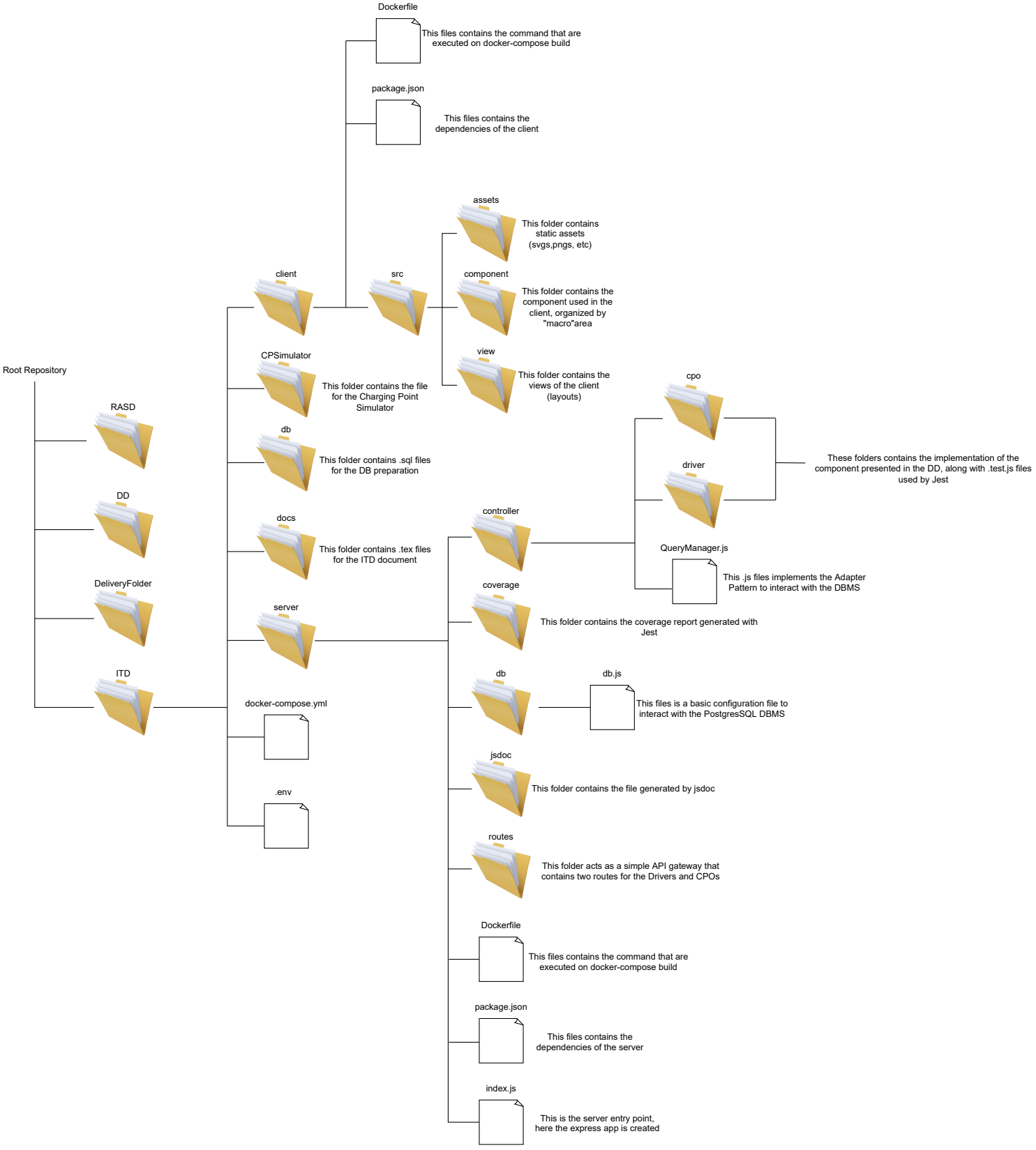


Figure 2: Source code structure of the application.

5 Testing

The project was tested using the **Jest** framework, a testing tool that enabled us to test the backend. Our testing approach was based on the **DD** as much as possible, given the time constraints.

First, we tested the most critical component, the **Query Manager**, through indirect testing, with a coverage rate of nearly 100%. Other components, which mostly serve as wrappers for the Query Manager, were tested using modular tests provided by Jest.

To ensure that our tests didn't affect the data in the database, we created a Query Manager method that executes a callback function and, regardless of its outcome, rolls back changes using a **ROLLBACK SQL** operation.

5.1 Relevant Test Cases

The following relevant test cases were implemented:

- User creation for CPO and Drivers, including sign-up, code verification, and login
- EVCP creation and deletion by the CPO, along with the creation of new CP and related sockets
- Driver reservation creation
- Driver search for EVCP
- CPO creation of rates and special offers
- CPO selection of a DSO, including searching for available DSOs and choosing one

5.2 Outcomes

A summary of the testing outcomes can be found in the Testing Report generated by Jest and located in the /coverage folder of the project. Detailed information is available in the report.

All files

84.56% Statements 1633/1931 **73.75%** Branches 163/221 **87.5%** Functions 84/96 **84.56%** Lines 1633/1931

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
controller		95.42%	709/743	69.56%	64/92	88.46%	46/52	95.42%	709/743
controller/cpo		75.82%	624/823	77.01%	67/87	80.64%	25/31	75.82%	624/823
controller/driver		81.26%	282/347	76.19%	32/42	100%	13/13	81.26%	282/347
db		100%	18/18	100%	0/0	100%	0/0	100%	18/18

Figure 3: Summary Coverage Report

6 Installation Guide

An exhaustive installation and usage guide can be found here, in the wiki of the project repository.