EVSYNC: Product specification report

Bernardo Borges [1035923], Gonçalo Monteiro [107758], Miguel Pinto [108481], Filipe Oliveira [114640]

1 Introduction

1.1 Overview of the project

The objective of this project within the scope of TQS course is to develop an application using Software Quality Assurance (SQA) principles and DevOps practices. The final goal is to deliver a Minimal Viable Product (MVP) with Spring Boot framework to facilitate tests and throughout the project we have to integrate Continuous Integration (CI) and Continuous Delivery (CD).
The application that we are going to create, **EVSync**, is designed to streamline electric vehicle charging experience providing a unified platform for EV drivers, station operators, and for users that care more about the environment. EVSync addresses the fragmented nature of charging services by offering real-time station discovery, slot booking, charging session management, and payment integration. It could also include features for station operators to manage availability and maintenance, as well as tools for users to track their environmental impact through CO2 savings metrics.

1.2 Known limitations

While the project meets its core functional and quality objectives, several limitations persist in the current iteration, primarily resulting from scope constraints and resource prioritization. These limitations are documented to inform future development and evaluation:

Simplified Wallet Implementation
The wallet subsystem was implemented in a minimal form, lacking integration with a real-world payment API. As a result, it does not support secure financial transactions or external payment gateways. This limitation stems from a deliberate decision to reduce complexity and focus on delivering a functional prototype within the available timeframe.

Lack of Robust Authentication Mechanisms
Although a basic authentication layer is present, it omits essential security practices such as password encryption, rate limiting, or protection against brute-force attacks. Advanced mechanisms like token expiration management or multi-factor authentication were excluded. These omissions were acknowledged during planning, with security features marked as secondary to functional deliverables.

Absence of Automated End-to-End Testing:
End-to-end (E2E) testing using Selenium or equivalent frameworks was initially considered but not implemented. The testing strategy prioritized unit, integration, and behavior-driven tests. Consequently, UI-level automation coverage is absent, increasing reliance on manual exploratory testing. The absence of E2E testing is recognized as a limitation in ensuring comprehensive regression validation.

These limitations reflect trade-offs made during development and serve as reference points for future improvements or extended project phases, particularly if the system is to be hardened for production-like environments.

1.3 References and resources

Some of the resources used in the project are: Spring Boot, a framework for backend development, including APIs REST and MySQL for the database, Next.js for frontend, OpenStreetMaps for location logic, Jira a tool to facilitate management of projects, backlogs, sprints, etc., SonarQube for static code analysis, k6 for performance testing, Grafana and Prometheus for monitoring.

2 Product concept and requirements

2.1 Vision statement

As a business, we intend to facilitate EV drivers to locate and use electric charging stations. A big problem within Portugal tends to be the long queues to access said stations, as well as the inability to preview possible malfunctions, putting unnecessary constraints on the drivers.
To solve this problem, EVSync not only shows locations and availability, but it also allows the user to make reservations for said charging stations, being able to arrive and just charge their vehicle.

2.2 Personas and scenarios

To further explain the main reason to use EVSync and other possible necessities, we created some personas as well as scenarios for each one.

1st Persona
Name – Joana Carmo
Occupation – TVDE Driver
Context – Uses an electric vehicle every day for both work and daily activities.
Objectives – Wishes to find free charging stations and quickly pay. Values speed and simplicity.

Scenario: Joana is finishing a shift as a ride-hailing driver. At the end of the day, she opens the app to find a charging station near her home. She books an available time slot at a nearby station, drives to the location, unlocks the charger, and starts charging. After the charging session, she quickly makes the payment through the app.

2nd Persona
Name – Carlos Simão
Occupation – Station Operator
Context – Works with around 15 charging stations
Objectives – Needs to monitor availability and mark any possible maintenances.

Scenario: Carlos logs into the management platform and notices that one of the charging stations is consistently inactive. He decides to mark the station as “under maintenance.” He updates the status in the app to prevent new reservations and schedules a technical visit.

3rd Persona
Name – Sara Marques
Occupation - Environmental Project Manager
Context – Is someone who makes changes to help with ecological changes
Objectives – Wants to follow her habits and environmental impact. Uses the app on occasion.

Scenario: Sara, interested in her environmental impact, opens the app to check her charging history. She reviews her consumption over the past few months and sees how much CO_2 emissions she has saved. She feels satisfied with the positive progress in her habits.

4th Persona
Name – Joel Martins
Occupation – EVSync Administrator
Context – Maintains the application as well as new charging stations.
Objectives – Maintains the application and is responsible for adding new charging stations to it.

Scenario: Carlos EVSync is expanding its charging network to a new zone of the city.

2.3 Project epics and priorities

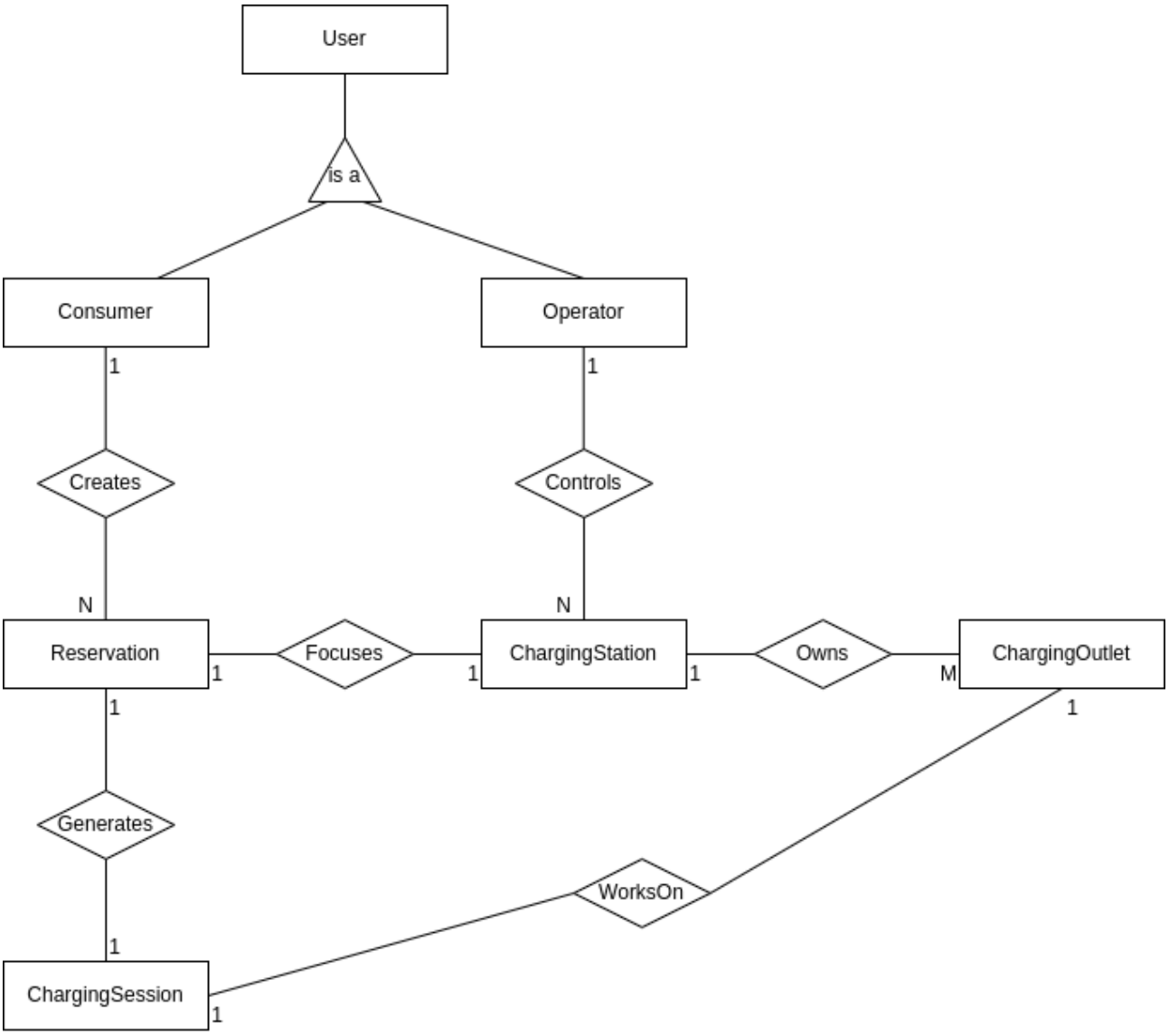
When it comes to epics, we have decided on an initial few. This list could increase as we see fit, as a way to introduce new features to EVSync.

- Discovery of charging stations
- Reservation of charging stations
- Charging session management
- Payments
- User history and statistics
- Charging station management (back office/admin)
- Environmental impact management (optional)

We'll focus on the customer side of the app, focusing on both the discovery and the reservation of the charging stations, as well as payment possibilities. The ability to manage your own charging session is also important, as the necessity to cancel them could easily arise.
Eventually, we'll focus on making the user profile with history and statistics, which could include the individual environmental impact.
Once the user experience is completed, we'll turn to work on the admin interface, which can be used to add, alter and delete stations.

3 Domain model

The domain of EVSync follows user interaction with the app, mostly revolving around what each kind of person can do here.
Here follows a diagram to showcase the domain itself.



There are 2 types of users:

- Our common Customer, which simply cares about making reservations. These reservations are made within an available charging station with charging outlets with different power and costs.
- The charging station Operator, who is assigned an amount of Charging Stations. He can modify charging stations, changing their status to display whether it is in maintenance or is available for customers.

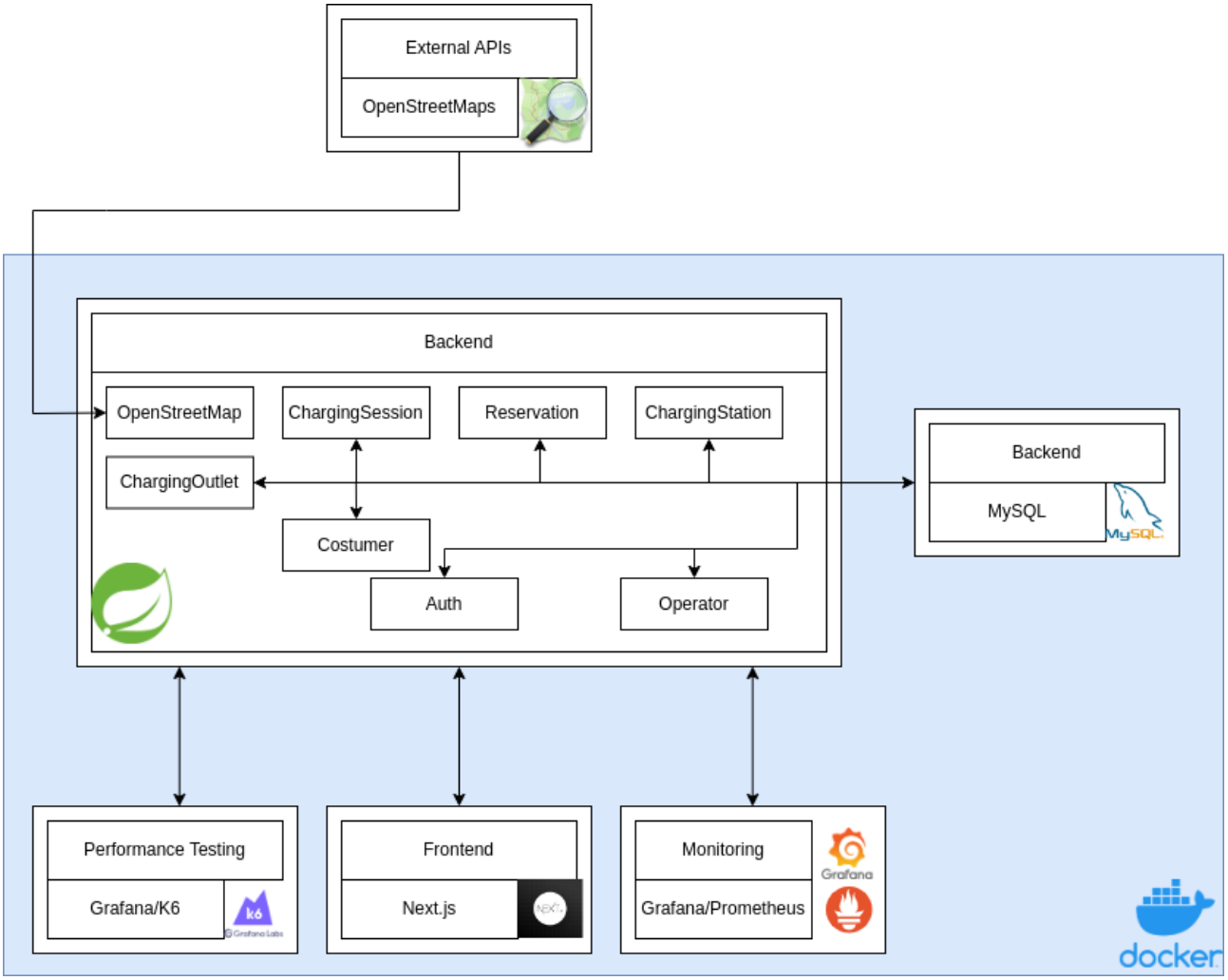
A Customer can create Charging Reservations on a Charging station. This reservation should provide a code that can be used by the customer in the terminal to initiate the Charging Session. This Reservation will be placed in a time frame, with an eventual cost being calculated for the time allocated for the user.

4 Architecture notebook

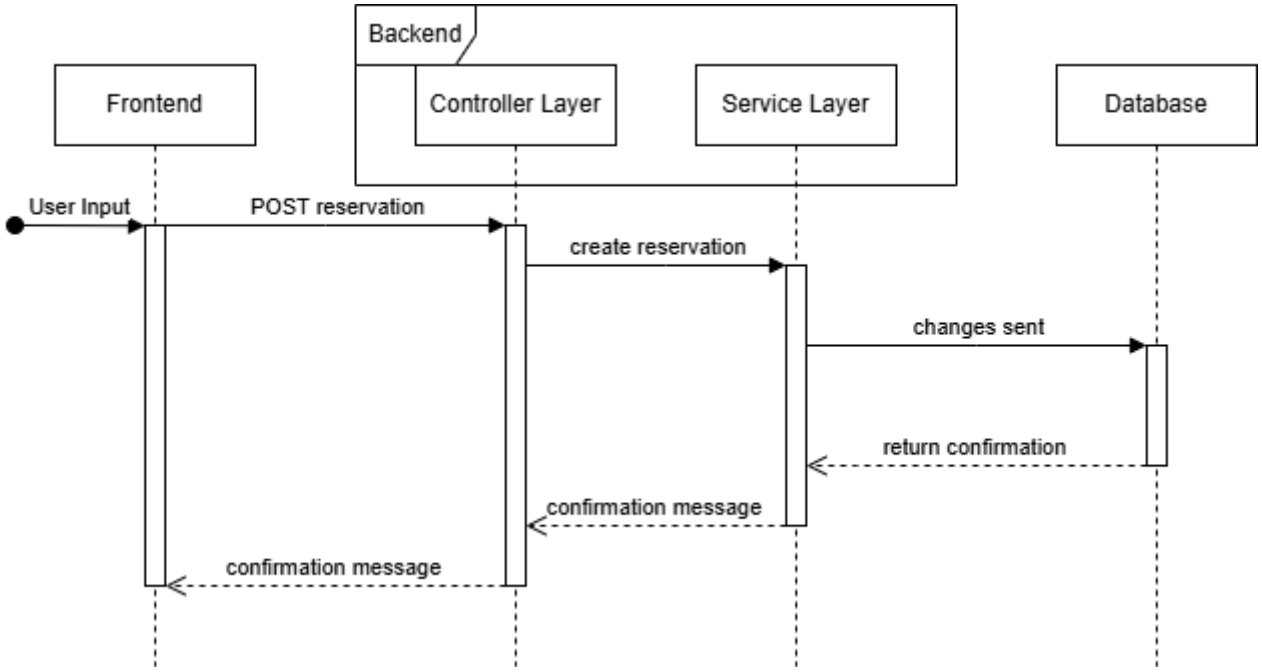
4.1 Key requirements and constrains

Regarding requirements and constraints, there are a few. To begin with, these should be able to connect to the EVSync machines located around Portugal. These should be able to read the codes generated by the reservations to allow for charging. The whole logic behind the app is around pre-paying, so even if you don't have a reservation, you should be able to generate a code for an instant charging instance.
In addition, we are using a map API to display the EV charging stations. We require that that API is always active, since it is vital for an important feature of the app.
The system should provide safety in all payments, as any possible mistakes could prove harmful to the customer. In a real scenario, there should be a fail-safe implemented into the system that would allow for managing possible errors in transactions.
The architecture itself is a layered architecture, meaning that adding extra features from outside could prove difficult.

4.2 Architecture view



When it comes to the system's architecture, it's made in a very simple layered architectural structure. The internal parts of the app are placed together using Docker, allowing for seamless communication between them.
The front end is done using React alongside JavaScript for any possible operations made by the user (ex. login in, making a reservation, etc...).
The backend functions as the middle ground between the frontend and the backend, helping with any alterations made by the users. It alters the database, which is done using MySQL, and communicates with any external APIs needed to function properly. In this case, we utilize *OpenStreetMap* to obtain a map of the area needed. It's written in java and uses SpringBoot and maven to work. We are also considering using a mocking API for payments, making it more realistic when the user needs to add funds to their wallet.
Each module works independently, only communicating with others when needed. Here follows a simple interaction between the user and the app - creating a reservation:



When the user sends the information through the frontend, it sends a POST request to the backend through an endpoint located in the Controller Layer. The controller then sends the information to the correct Service, in this case the Reservation Service, where it creates the reservation and attempts to send it to the database. In a successful change, the database returns the confirmation to the backend, where it's eventually communicated to the user through the frontend.

4.3 Deployment view

The application will be deployed on a virtual machine (VM) provided by the professors. This VM will host the production version of the application using Docker containers, ensuring a clean, isolated, and reproducible environment.

VM Access

- VM IP address (within eduroam): 192.168.160.14
- Service access: Each service is exposed through a specific port using Docker's port mapping.

Services and Containers

The solution consists of six Docker containers, organized as follows:

Service	Port (Host:Container)	Technology Used	Purpose
Backend	8080:8080	Java / Spring Boot	Main API and business logic
Frontend	3000:3000	Next.js	Web user interface
MySQL	3306:3306	MySQL	Relational database
Grafana	3001:3000	Grafana	Visualization of metrics and logs
Prometheus	9090:9090	Prometheus	Collects metrics from monitored services
k6 (via Grafana)	*(no fixed port)*	k6 + Grafana plugin	Load testing tool, triggered manually when needed

Note: The **k6** container is not always running. It is triggered manually during performance testing sessions.

5 API for developers

The application exposes a RESTful API, following best practices and principles of resource-oriented design. This means the API is designed around entities (resources) rather than actions, and it uses standard HTTP methods (GET, POST, PUT, DELETE, etc.) to perform operations on these resources. The API is organized into logical collections that represent the main domain entities of the application. Each collection is accessible via a predictable, hierarchical URL structure and returns standardized HTTP status codes.

Session – GET /api/v1/sessions
GET /api/v1/sessions/{id}
PUT /api/v1/sessions/{id}/end
POST /api/v1/sessions/{id}/start
DELETE /api/v1/sessions/{id}

Operator – GET /api/operators/{id}
GET /api/operators
GET /api/operators/{id}/charging-stations
GET /api/operators/email/{email}
PUT /api/operators/{id}
POST /api/operators
DELETE /api/operators/{id}

Consumer – GET /api/consumer/{id}
GET /api/consumer/{id}/wallet
GET /api/consumer
GET /api/consumer/email/{email}
PUT /api/consumer/{id}
PUT /api/consumer/{id}/wallet
POST /api/consumer

DELETE /api/consumer/{id}

Reservation – GET /api/reservations
POST /api/reservations/{id}
POST /api/reservations/{id}/cancel
POST /api/reservations/{id}/confirm

Charging Outlet – GET /api/outlets
POST /api/outlets/{stationId}

Note: Detailed documentation of endpoints (parameters, responses, examples) will be hosted on a dedicated platform using Swagger UI in the following link - http://192.168.160.14:8080/swagger-ui.html