TQS: Product specification report

Pedro Ponte [98059], Alexandre Regalado [124572], Miguel Soares Francisco [108304], Inês Ferreira [104415]

v2025-04-30

1 Introduction

1.1 Overview of the project

This project was created as part of the TQS course, where we were challenged to build an application while putting into practice everything we've learned, from planning and designing to testing and ensuring quality.

Our app is called **ChargeUnity**, and it's designed to make life easier for electric vehicle drivers. Instead of driving around hoping to find an available charger, users can check availability and book charging spots ahead of time. It's a simple, practical way to skip queues and charge when it fits their schedule. Station operators can also easily monitor and manage their charging stations through the app, giving them full visibility and control over bookings, availability, and user activity.

This aligns with TQS, as we'll be applying concepts we've learned throughout the course, including the different testing methods and quality assurance practices that will be implemented in our app.

1.2 Known limitations

Not all user stories were implemented, including payment operations using Stripe and IPMA weather calls.
The deployment worked earlier when testing, but when time came and we needed to deploy in production our code project wasn't aligned with what was expected. We couldn't figure out in time why our main branch was aligned with what was in develop.

1.3 References and resources

1. Swagger
2. SpringBoot
3. Grafana
4. SonarCloud
5. K6
6. Cucumber BDD
7. TDD with JUnit and Mockito
8. Course materials

2 Product concept and requirements

2.1 Vision statement

ChargeUnity aims to address on of the significant challenges in the electric vehicle (EV) charging ecosystem: the fragmentation of charging services across different networks. By providing a unified platform for EV drivers and station operators, ChargeUnity simplifies the process of finding, booking, and paying for charging services. The core problem the system solves is the inconvenience caused by the lack of interoperability between charging stations, improving the overall experience for EV drivers

while also enhancing operational efficiency for station operators.
ChargeUnity's platform delivers the following key features:

- **Charger Discovery**: Users can easily locate available charging stations in real time.
- **Slot Booking**: Drivers can reserve charging slots in advance, ensuring availability and reducing wait times.
- **Charging Management**: The system allows users to start, monitor, and finish charging sessions with real-time consumption tracking.
- **Payment Integration**: A flexible payment system that supports pay-per-use and subscription-based models.
- **Station Management**: Station operators can register, monitor, and maintain charging stations, optimizing availability and maintenance schedules.

ChargeUnity differentiates itself from existing solutions such as PlugShare or ChargeMap by offering a more integrated and user-friendly approach since these platforms focus on station discovery and availability, but ChargeUnity combines discovery with slot booking, payment integration, and station management into a single, streamlined platform. Additionally, ChargeUnity includes future-oriented features like CO2 savings visualization, making it a more holistic solution for both EV drivers and station operators.

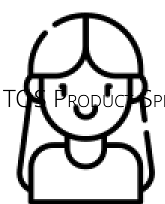
To gain deeper insights into our target market, we conducted interviews with EV owners to identify the key features they would want in a charging app. We focused on understanding the most convenient functionalities and their specific pain points. Additionally, we explored online communities where users frequently discussed their frustrations with existing EV map apps. This helped us uncover opportunities for innovation and areas where we could introduce improvements in our own app.

2.2 Personas and scenarios

To ensure ChargeUnity meets the needs of its diverse user base, we developed a set of detailed personas based on real-world user behaviors and pain points gathered through interviews, online discussions, and market research. These personas represent the different types of users who interact with EV charging system ranging from everyday drivers to infrastructure managers and business operators.

By exploring realistic usage scenarios for each persona, we were able to validate our feature set, prioritize development efforts, and design a platform that addresses both the technical and experiential challenges of EV charging. Each scenario illustrates how ChargeUnity simplifies daily routines, solves specific frustrations, and adds value to both individuals and organizations involved in the EV ecosystem.

Maria Agostinho

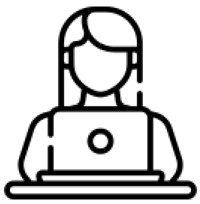


- **Age**: 24
- **Occupation**: Freelance Graphic Designer & Sustainability Blogger
- **Location**: Lisbon, Portugal
- **Electric Vehicle**: Peugeot e-208
- **Personality**: Eco-driven, tech-savvy, minimalist
- **Motivations**: Minimize her carbon footprint, stay productive on the move, and live by example for her followers
- **Pain Points**: Inconsistent pricing across chargers, stations hidden in hard-to-reach places, no quick stats for usage impact

João Ferreira

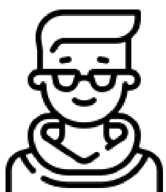
- **Age**: 45
- **Occupation**: Regional Sales Representative
- **Location**: Braga, Portugal
- **Electric Vehicle**: Tesla Model 3
- **Personality**: Pragmatic, efficient, always on the clock
- **Motivations**: Maximize uptime, plan ahead, simplify reimbursements
- **Pain Points**: Wasted time at full chargers, unpredictable station availability, clunky expense tracking

Inês Lopes



- **Age**: 38
- **Occupation**: EV Charging Infrastructure Manager
- **Location**: Porto, Portugal
- **Tools**: Operates 20 charging stations across Northern Portugal
- **Personality**: Detail-oriented, structured, loves dashboards and live data
- **Motivations**: Keep uptime high, improve charger usage, reduce user complaints
- **Pain Points**: Hard-to-track usage spikes, slow maintenance updates, lack of predictive tools

Alberto Correia



- **Age**: 25
- **Occupation**: Software Engineer
- **Location**: Aveiro, Portugal
- **Electric Vehicle**: VW ID.3
- **Personality**: Curious, analytical, quick to adopt new tech
- **Motivations**: Understand his car's behavior, optimize usage, experiment with charging times
- **Pain Points**: Learning curve of EV ownership, anxiety around charging logistics, lack of personalized tips

Helena Tavares

- **Age**: 52
- **Occupation**: Manager at Algarve EV Rentals
- **Location**: Faro, Portugal
- **Manages**: 30 EVs for tourism purposes
- **Personality**: Efficient, commanding, cool-headed under pressure
- **Motivations**: Fast turnover of vehicles, smooth logistics, customer satisfaction
- **Pain Points**: Tourist misuse of chargers, overlapping reservations, station bottlenecks in peak season

Scenarios

Maria's scenario 

It's Thursday morning, and Maria's planning to hit her yoga class at 10 AM before a client meeting at noon. Her car's at 37% battery, enough to get around but not enough for tomorrow's out-of-town trip.
She opens the ChargeUnity app and filters chargers near her yoga studio. A nice café across the street has a fast charger with a 25% off-peak discount.
Maria books the slot for 9:45–10:45, heads out, and drops her car off before class. She does her exercises, grabs a *galão* from the café, and returns to find her battery at 94%.
Later that evening, she posts a screenshot of her CO2 savings from the app's dashboard, she's now saved 1.2 tons this year.

João's Scenario 

João's day starts at 6:30 AM with a strong espresso and a packed schedule. He's driving from Braga to Lisbon for back-to-back meetings.
He opens ChargeUnity and plans his route: one high-speed charger near Coimbra timed to keep him on the move.
He books the time slot and pre-pays using his card. The Coimbra stop is seamless, no waiting, automatic unlock when he arrives. While charging, he sends an email and checks the meeting agenda.

Inês' Scenario 

Inês starts her day by logging into the ChargeUnity worker dashboard. She notices one charging station marked as "Available" but with no recent usage.

Using the platform's **monitoring tools**, she checks the station status and sees a small issue reported. She uses the **maintenance feature** to change its status to **"Unavailable",** and schedules a visit.

After the technician resets the unit, Inês updates the status to "Available" again directly through the system.

Later, she checks the **map view** and sees the station is being used again.

Alberto's Scenario 

Alberto had planned to charge his EV at home overnight, but forgot to plug it in.
With only 12% left, he opens ChargeUnity and finds three chargers nearby, but one is slightly cheaper and near his favorite record store.
He books a slot and unlocks the charger when he gets there. While his ID.3 fills up, he flips through vinyls and grabs a new indie album.
Back in the car, the app sends him a little "Efficiency Tip": he could save 4% on average if he avoided AC use below 50% charge.

Helena's Scenario 

It's peak tourist season, and Helena's EV rental fleet is flying off the lot. But with 30 cars needing charging every night, she has to orchestrate this perfectly.
She uses ChargeUnity's scheduling tool to book charging windows across four stations, she staggers them to prevent queueing and ensures the fast chargers go to the larger vehicles.
Not a single customer missed their morning tour, and no one had to wait for a charger.

2.3 Project epics and priorities

Our project will be developed in weekly sprints, for making progress towards our end goal we defined a priority for the development of our epics, which encapsulate our different user stories. The priority of our epics is the following:

Station Discovery

1. Search for chargers
   - **As** an EV driver, **I want** to search for charging stations nearby or in a specific destination, **so that** I can find a convenient place to charge my car.

2. Register charging station
   - **As** a Station Operator, **I want** to register a new charging station with its specifications **so that** it becomes available for drivers to view and book.

Charging

1. Charge vehicle
   - **As** an EV driver, **I want** to start charging my vehicle once I arrive at the station, **so that** I can charge it quickly and easily.
2. Plan trips
   - **As** an EV driver, **I want** to plan a trip with charging stops along the route, **so that** I can complete the journey without running out of battery.

Slot booking & scheduling

1. Book time slots
   - **As** an EV driver, **I want** to book a time slot at an available charging station, **so that** I can ensure a charger is free when I arrive.
2. Receive maintenance alerts
   - **As** a Station Operator, **I want** to be notified when a charger fails or goes offline **so that** I can schedule maintenance and prevent service.
3. Update station availability
   - **As** a Station Operator, **I want** to change the availability status of chargers **so that** drivers cannot book chargers that are unavailable.

Payment integration

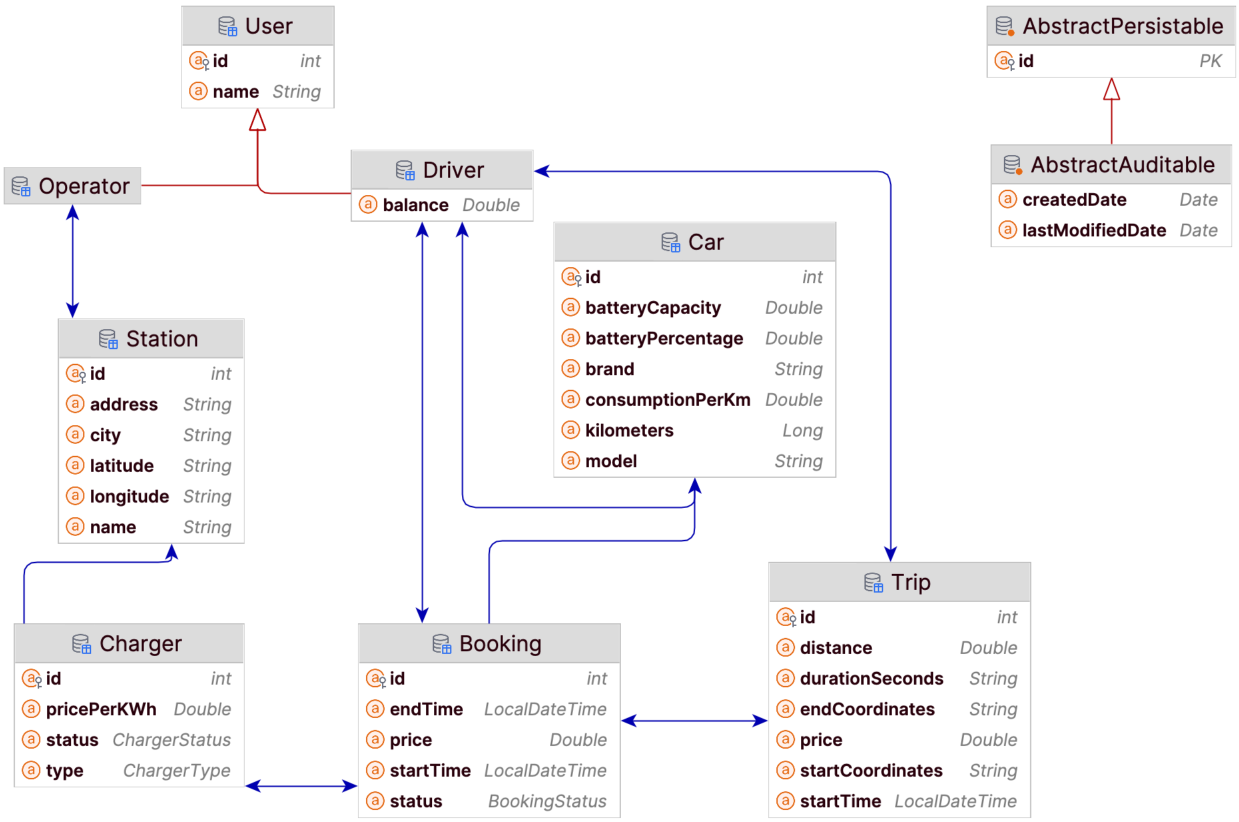
1. Make payment
   - **As** an EV driver, **I want** to pay for the charging session directly in the app, **so that** I don’t have to use physical or external payment methods.
2. Configure off-peak discounts
   - **As** a Station Operator, **I want** to define off-peak hours and apply discounts **so that** I can incentivize usage during those hours.

User stats

1. Monitor daily operations
    - **As** a Station Operator, **I want** to monitor current and historical charging activity per station **so that** I can ensure everything is running smoothly.

3 Domain model

This chapter describes the core domain of the ChargeUnity application, focusing on the main information concepts and the logical relationships between them. The domain is represented through a UML class diagram, which defines the structure of the system.



The system manages two main types of users: `Drivers` and `Operators`. Both inherit from a general `User` class, which includes shared attributes such as an identifier and a name. Drivers use the system to manage their vehicles, plan trips, and reserve charging stations. Operators are responsible for managing the infrastructure, including stations and chargers.

Each driver can register one or more electric vehicles. Each `Car` holds information such as battery capacity, current battery percentage, energy consumption per kilometre, brand, model, and total kilometres driven. This data supports both trip planning and charging logic.

`Trips` represent planned journeys made by drivers. A trip includes information such as start and end coordinates, estimated distance, duration, start time, and projected cost. When a trip is created, a booking associated with it by id can be made to accommodate said trip with a charging session along the way.

`Bookings` represent the reservation of a charging session. They include details such as start and end time, the total price, and the current booking status. A booking is associated with a **`driver`**, a **`charger`**, and optionally a **`trip`**.

`Chargers` are located at **`stations`** and are identified by id, type, price per kilowatt-hour, and current status (e.g., available, in use etc.).

`Stations` themselves include details like address, city, and geographic coordinates. Each station is managed by an operator and contains one or more chargers.

This structure ensures a clear separation of responsibilities while enabling flexible interactions between users and infrastructure.

4 Architecture notebook

4.1 Key requirements and constraints

Our architecture is shaped by a set of key requirements and constraints that aim to deliver an user-friendly platform for both EV drivers and station operators. These requirements will help us design and implement the system to ensure it meets users' needs.

Key Requirements

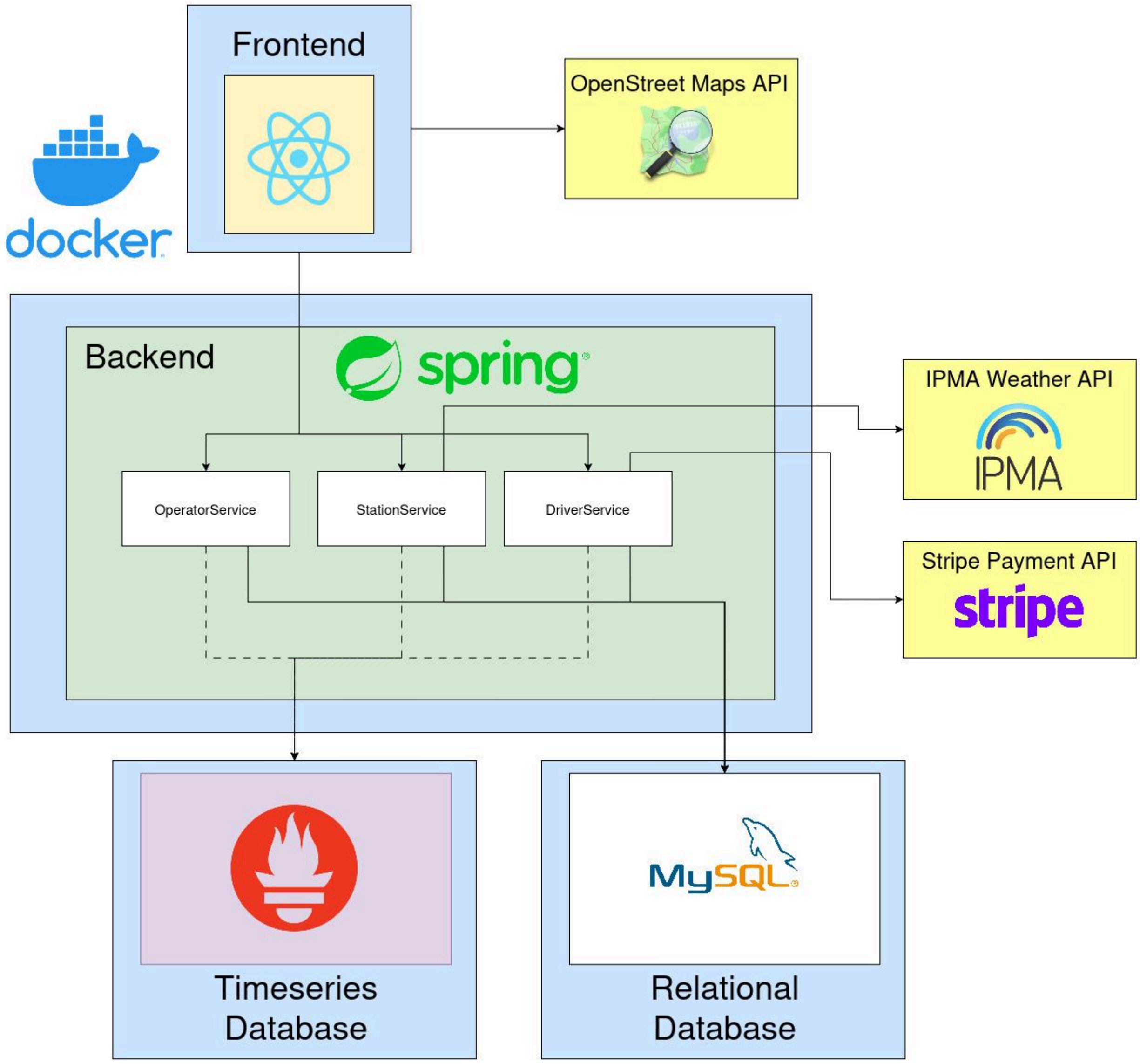
1. ChargeUnity will be available across multiple user-facing platforms, including web and mobile devices. This is achievable by creating responsive pages in *React*.
2. The platform must integrate with external systems, such as:
    - **Payment Gateways**: Stripe's sandbox for completing payments
    - **Forecast Services**: IPMA's API for showing forecast on destinations
    - **Navigation Systems**: Google Maps API for creating routes and showing maps
3. The application should function efficiently under unusual conditions, such as high traffic during peak hours or adverse network connectivity.
4. The architecture should allow for future enhancements, such as:
    - Adding new features like carbon footprint tracking or predictive analytics.
    - Supporting new types of chargers or payment methods.
5. The system should provide:
    - **Drivers**: Booking and payment features, personalized trips, and easy charging operations.
    - **Station Operators**: Informative dashboards to monitor station performance, manage bookings, and schedule maintenance.

Key Constraints

1. The application must be developed until the end of the TQS classes.

2. The system relies on the availability and reliability of third-party services, such as payment and weather forecast APIs.
3. We need chargers that integrate with our service through their own APIs

4.2 Architecture view



Our architecture is designed to ensure modularity and a seamless integration with external services. The system is divided into three primary layers: **Frontend**, **Backend**, and **Data Storage**, with additional external API integrations to enhance functionality.

Frontend

Our *frontend* is built using *React*. It was chosen due to its very rich ecosystem, easy syntax and overall familiarity of everyone in the group. It communicates with the backend using our REST API.

Backend

The *backend* is implemented using **Spring Boot**, this was a mandatory requirement of the assignment, but one that also fits the group as everyone is familiar with this framework.

We'll follow a SOA with several modular services each handling a specific domain:

1. **DriverService**: Manages all operations related to EV drivers, such charging and handling cars
2. **StationService**: Handles station registrations, updates, and monitoring.
3. **OperatorService**: Deals with operator requests like changing station availability and configuring discounts.

Relational Database

The backend uses **MySQL** as the relational database to store and manage persistent data, including user information, booking details and charging station status.

External APIs

The system integrates external APIs to extend its functionalities:

- **Stripe Payment API**: Facilitates secure and reliable payment processing using a sandbox for a simulation of a real process.
- **IPMA Weather API**: Provides real-time weather updates and forecasts.
- **OpenStreet Maps API**: Enables map visualization.

TimeSeries Database

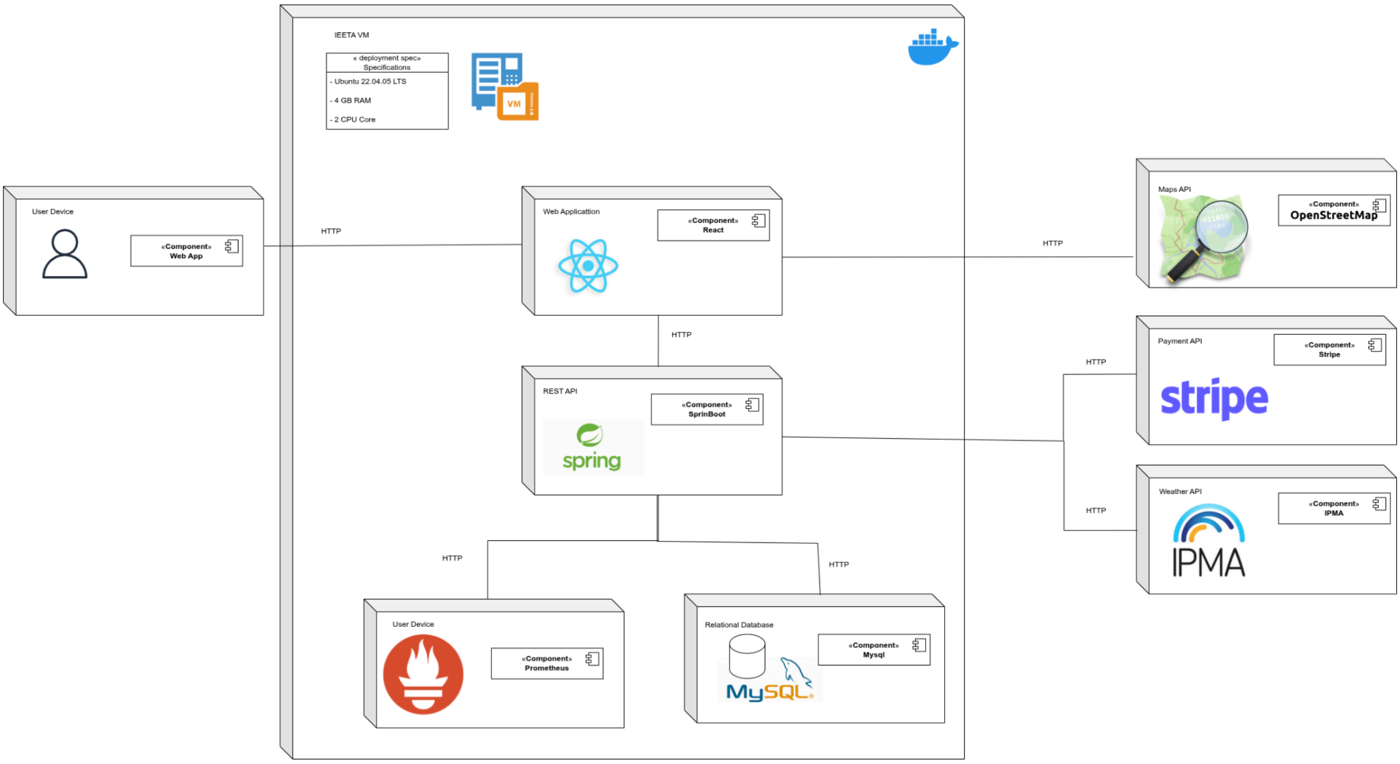
We also have a **Prometheus** TimeSeries Database to help monitor our API and collect metrics.

4.3 Deployment view

For deployment, we divided our system into 4 different containers:

- **Databases**: **MYSQL** and **Prometheus**, which run on ports 3306 and 9090 respectively
- **Backend**: **SpringBoot** which is on port 8080
- **Frontend**: **React** in port 5173
- **Observability**: **Grafana** running on 3000

Every service in our system communicates in HTTP and they're all connected to the same *docker network*.



5 API for developers

The **ChargeUnity REST API** is designed following best practices in resource-oriented REST architecture. Each core domain entity (such as *booking*, *car*, *driver*, *station*, etc.) is exposed through structured RESTful endpoints. Full API documentation is available via Swagger UI at:

`http://192.168.160.8:8080/swagger-ui/index.html`

The API is organized into three main types of controllers:

1. Entity Controllers

Responsible for CRUD operations on core entities:

- **Booking** → `/bookings`
- **Car** → `/cars`
- **Charger** → `/chargers`
- **Driver** → `/drivers`
- **Operator** → `/operators`
- **Station** → `/stations`

- **Trip** → `/trips`
- **User** → `/users`

Typical operations include:

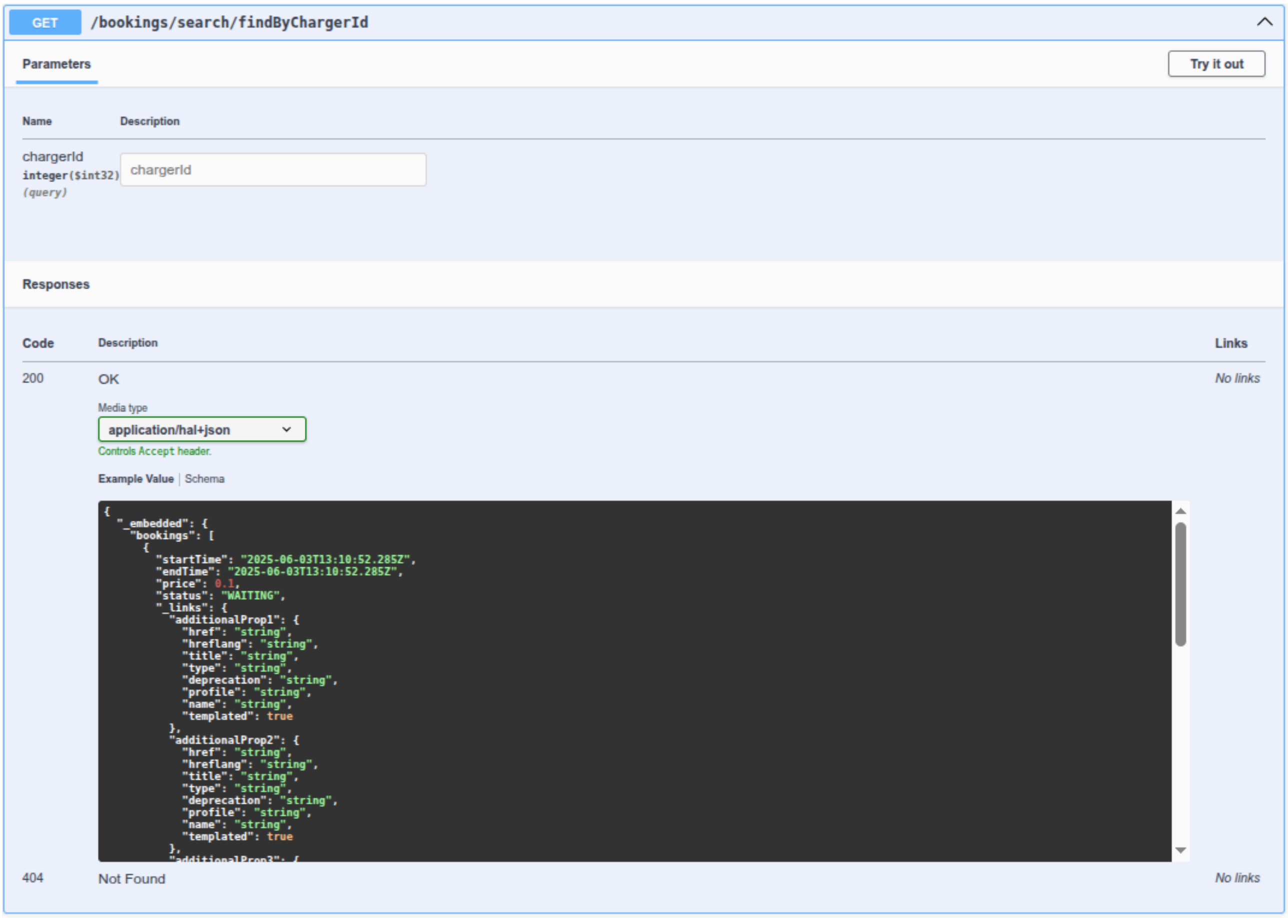
- `GET /{entity}` – list all
- `GET /{entity}/{id}` – retrieve by ID
- `POST /{entity}` – create new
- `PUT`, `PATCH`, `DELETE /{entity}/{id}` – update or delete

2. Search Controllers

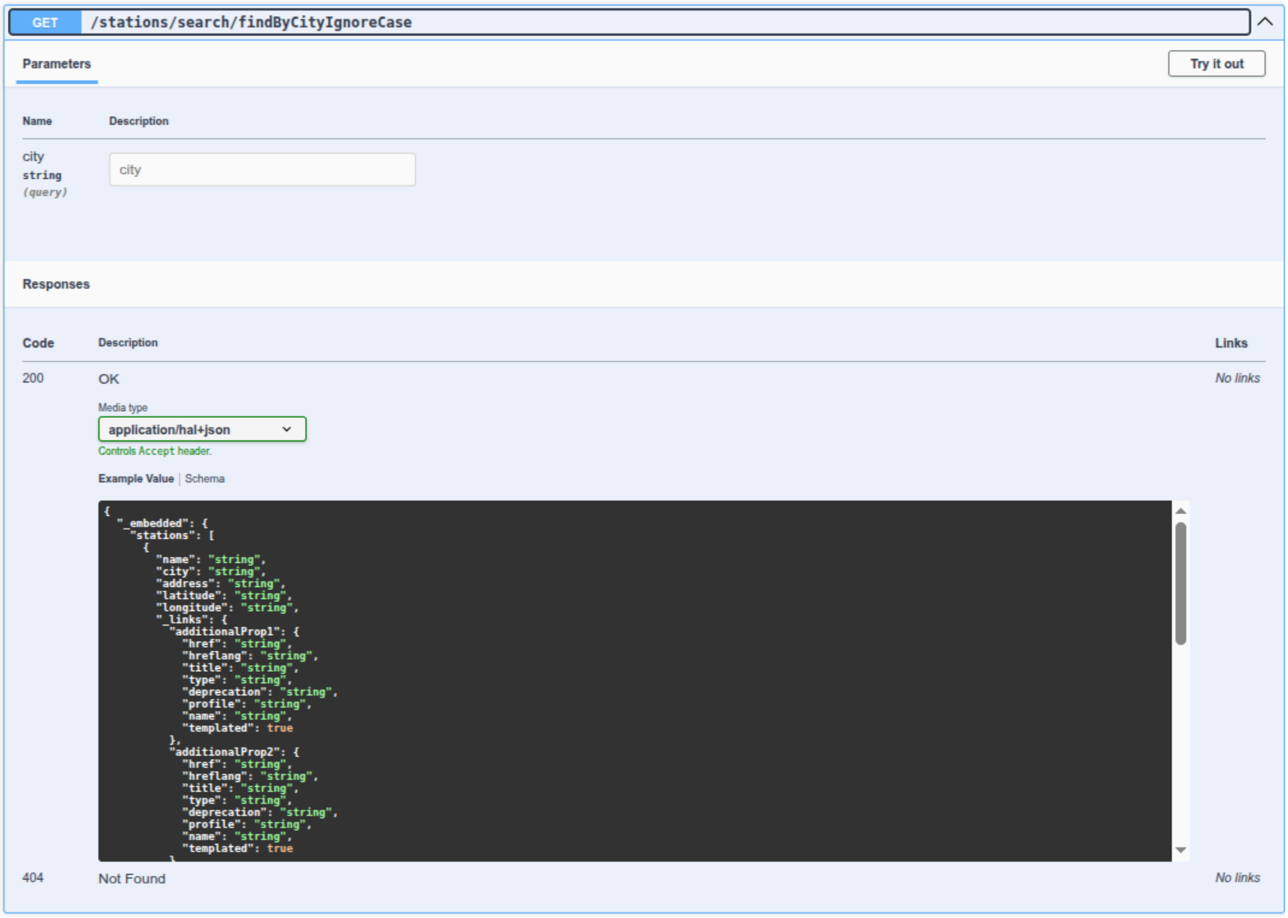
These expose advanced querying capabilities using custom filters:

Examples:

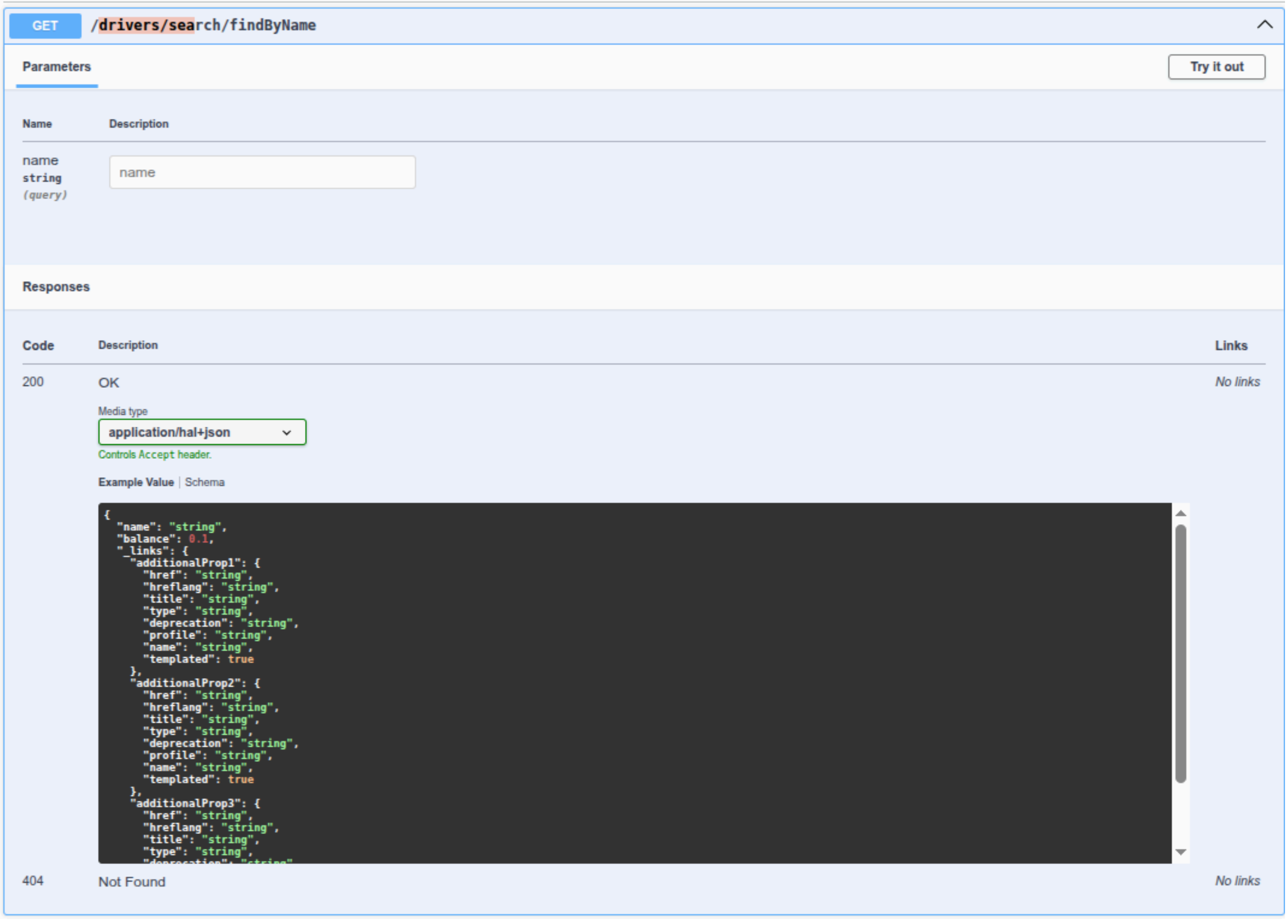
- `GET /bookings/search/findByChargerId`



- `GET /stations/search/findByCityIgnoreCase`



- `GET /drivers/search/findByName`

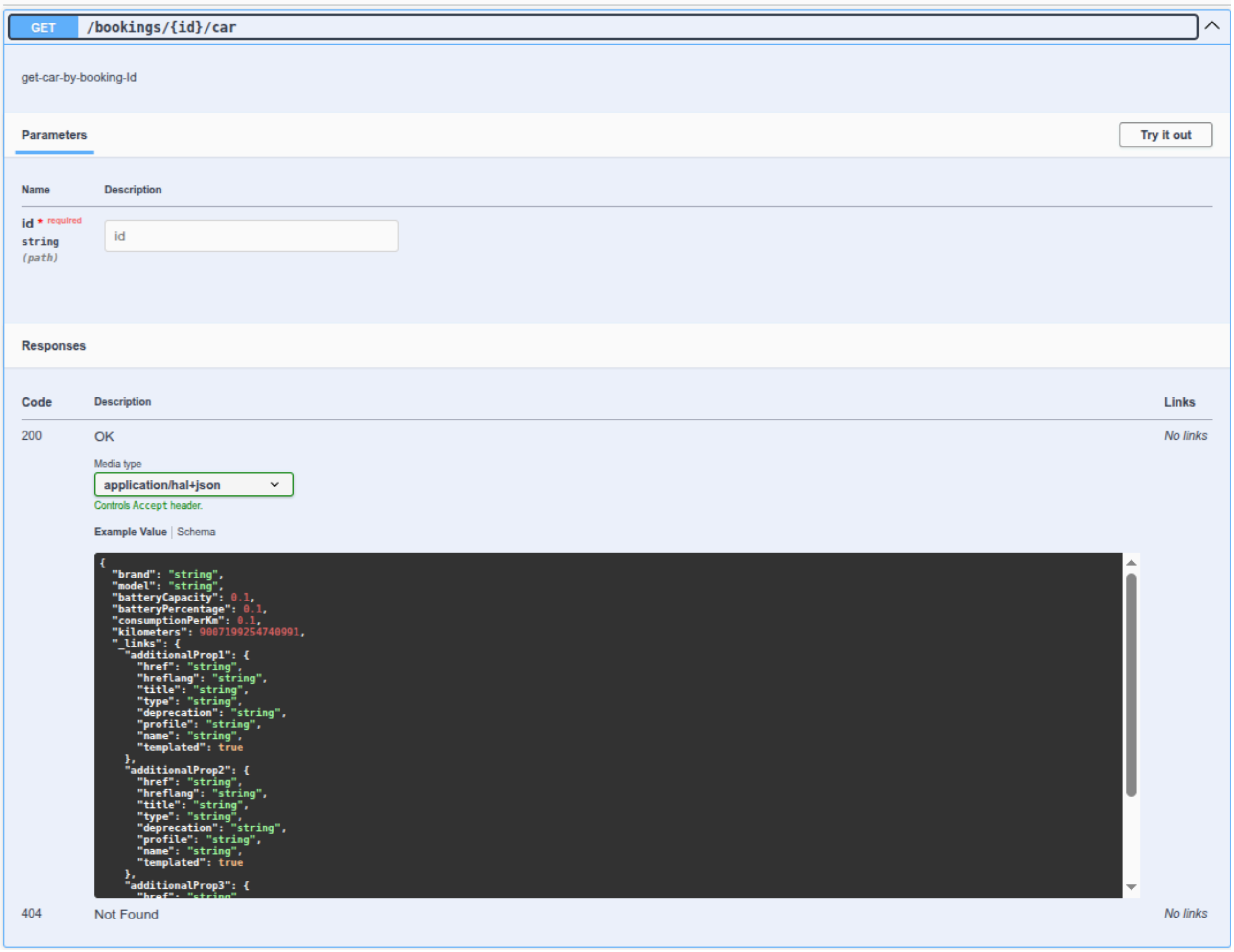


3. Property Reference Controllers

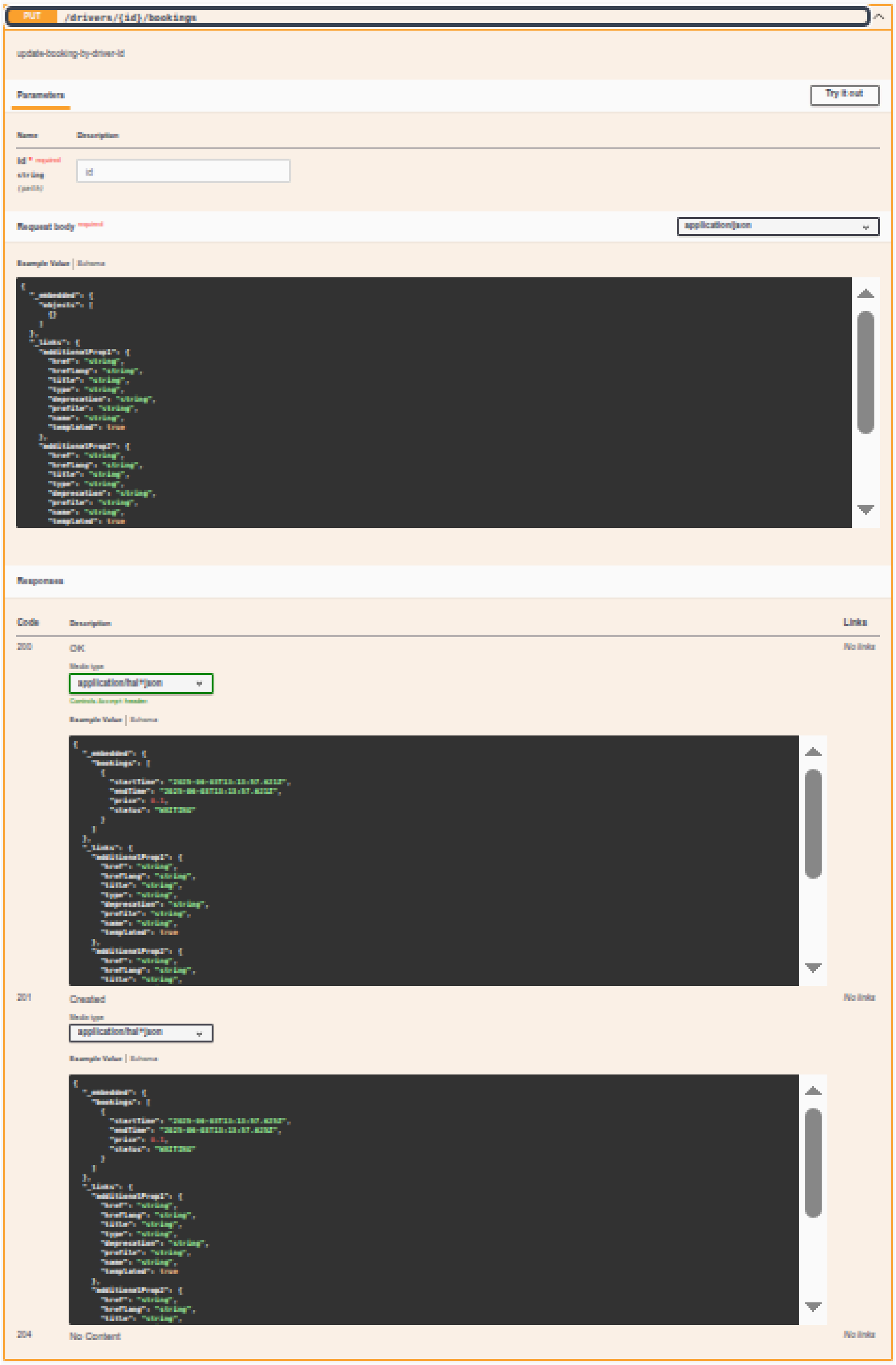
Used to manage relationships between entities (references and associations):

Examples:

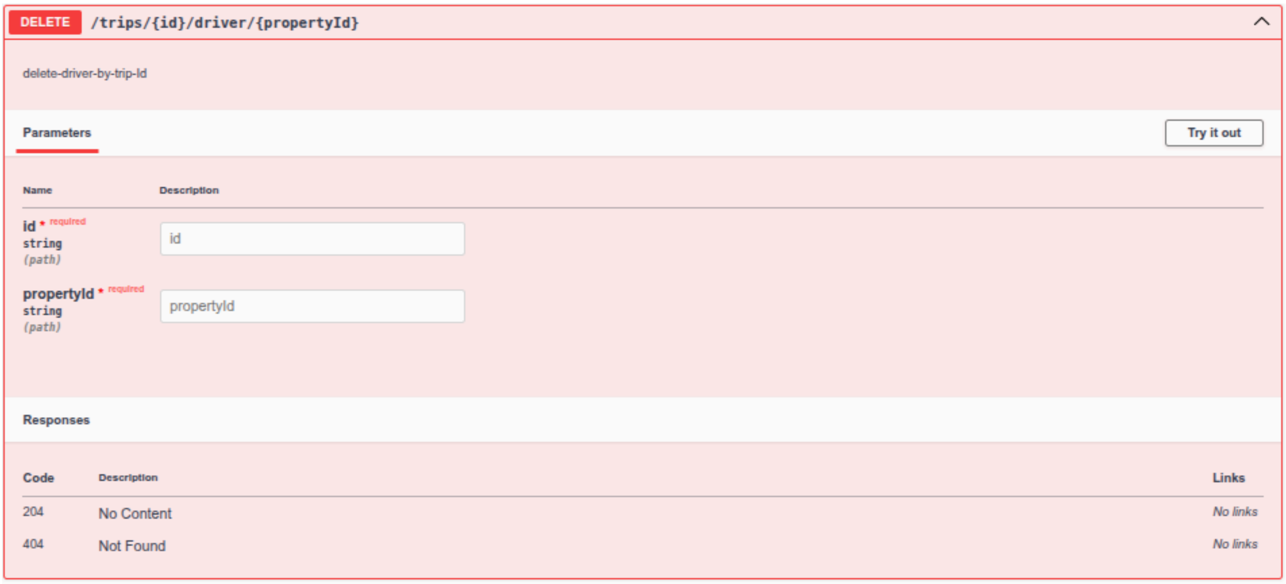
- `GET /bookings/{id}/car`



- `PUT /drivers/{id}/cars`



- `DELETE /trips/{id}/driver/{propertyId}`



4. V1 Simplified Version (Custom Controllers)

To simplify frontend integration, more focused and user-friendly endpoints are provided:

- `/api/v1/station`, `/api/v1/driver`, `/api/v1/booking`, etc.
- Additional logic includes:
  - `PATCH /api/v1/bookings/{id}/start`
  - `GET /api/v1/station/coordinates/{latitude}/{longitude}/{radius}`

6 References and resources

Project Resources

Resource	URL / Location
Git Repository	GitHub
Jira	Backlog
Video Demo	Youtube
QA Dashboard	Grafana dashboard (user:admin/pass:admin)
CI/CD pipeline	Workflows