deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

TQS: Product specification report

Ana Rita Fernandes da Silva [114220], Ângela Maria Martins de Abreu Ribeiro [109061], Carolina Sofia Pereira da Silva [113475], Hugo Gonçalo Lopes Castro [113889]
v2025-04-25

1 Introduction

1.1 Overview of the project

This project was developed within the scope of the *Teste e Qualidade de Software (TQS)* course and aims to apply software testing principles and quality assurance techniques in the context of a real-world software product. The project challenges students to specify, design, and verify a software solution with attention to functionality, usability, maintainability, and testability.

The solution developed is called **NikCharge**, a digital platform that addresses common challenges in the electric vehicle (EV) charging ecosystem. It is designed to serve three main user types: EV drivers, station employees, and station managers. The product offers a centralized system to streamline the discovery, reservation, operation, and management of EV charging stations.

NikCharge enables users to locate and reserve charging stations in real-time, manage charging sessions, and simulate payments, while also providing dashboards for personal insights (for drivers) and operational tools (for staff). By leveraging a modern technology stack and aligning the needs of all stakeholders, NikCharge offers a seamless and efficient EV charging experience.

1.2 Known limitations

During the development of the NikCharge platform, some features initially planned were not implemented in the final version of the project, due to time constraints and development priorities. The main known limitations include:

Trip Planning:

The functionality that would allow users to plan routes with multiple stops and automatically reserve charging stations along the journey was not implemented. This feature would require integration with more advanced mapping services and additional scheduling logic, which was beyond the scope of the MVP.

Manager Dashboard:

Although operators can manage individual chargers, the functionality that would allow station managers to view the aggregate status of all stations, monitor real-time performance metrics, and make operational decisions based on analytical data was not developed. This feature would require the development of more complex visualizations and system-level data aggregation logic.

These limitations were considered secondary compared to the platform's essential flows, such as charger discovery, reservations, charging sessions, and basic operational management. The mentioned features remain as potential future evolutions of the platform.

1.3 References and resources

During the development of the NikCharge platform, we used several resources and tools that were fundamental to the success of the project. Below are the main libraries, services, and references that we recommend to other students:

- **React** (https://react.dev/): Used for frontend development, focusing on reusable components and responsive design. The official documentation was essential for resolving doubts and implementing best practices in component structuring.
- **Spring Boot** (https://spring.io/projects/spring-boot): The foundation of the application's backend. Its "convention over configuration" approach facilitated the rapid and modular development of REST APIs.
- **Stripe API** (https://stripe.com/docs/api): Used for payment simulation. The clear documentation and code examples made integration with the system much easier.
- **PostgreSQL** (https://www.postgresql.org/): The database system used. The official website and forums like Stack Overflow were helpful for questions about modeling and queries.
- **Swagger** (https://swagger.io/): An important resource for API documentation. It made it easier to organize and verify the API's compliance with REST best practices.
- **Docker** (https://docs.docker.com/): Used to ensure consistent development and deployment environments. The official documentation and online tutorials were very helpful in creating the Dockerfiles.

2 Product concept and requirements

2.1 Vision statement

Our platform delivers a seamless and centralized solution for managing the complete electric vehicle charging experience. It connects drivers, station employees, and station managers through a unified digital ecosystem that simplifies discovery, reservation, charging, and operations.

The platform addresses key challenges in EV charging, including unreliable access, lack of reservation guarantees, and inefficient station management. It enables real-time station discovery, accurate availability, reservation enforcement, and clear operational control.

For drivers, the platform offers a fast and intuitive experience. They can search for nearby stations, reserve chargers in advance, receive accurate estimates for cost and charging time, and track their usage and spending through a personal dashboard.

For station employees, it provides an easy-to-use dashboard to monitor charger status, mark chargers under maintenance, and quickly update availability. This reduces downtime and prevents customer disruptions.

For station managers, the platform supports oversight and optimization. It offers tools to monitor station performance, configure pricing and discounts, and analyze usage trends to improve operations and profitability.

Core features include:

- Real-time charger search with filtering by location, charger type, time, and pricing
- Smart reservation system with estimated session time and enforcement via unlock codes
- Charging session management with energy tracking and simulated payments
- User profiles with personalized preferences and access to full charging history
- Operational tools for employees to manage charger availability and maintenance status
- Management dashboards with reporting, performance insights, and pricing controls

By aligning the needs of EV drivers, station operators, and business managers, our platform creates a reliable, efficient, and user-friendly EV charging experience.

2.2 Personas and scenarios

Urban Commuter: Sofia Oliveira

"I need to find a nearby charger I can rely on during the week—quick, easy, and stress-free."

Name: Sofia Oliveira

Age: 34

Gender: Female

Occupation: Marketing Manager

Location: Lisbon, Portugal

Income: Middle-high income

Tech familiarity: Tech-savvy, frequent app user

Sofia is a busy marketing manager who relies on her EV for commuting and errands around Lisbon. She expects quick access to nearby chargers, real-time availability, and a fast, clear booking experience. She values the ability to track costs and energy usage over time through a personal dashboard.

Goals:
- Find and reserve nearby chargers based on real-time or planned availability
- Filter chargers by type and price (e.g. fast, discounted)
- View charging cost, duration, and history
- Track energy use and spending over time

Frustrations:
- Chargers being in use despite being shown as available
- Not being able to book in advance
- Confusing or slow reservation/payment flows
- Lack of visibility into charging trends or cost history

Behaviours:
- Opens the app just before leaving work or during errands
- Uses location and time filters frequently
- Manages her EV profile through the app settings when specs change

Needs:
- A fast and intuitive way to search for nearby chargers based on current location or a specific area
- Accurate, real-time information about charger availability
- The ability to filter chargers by type (e.g., fast or standard) and see any active discounts
- A simple reservation process that shows clear pricing, estimated time, and energy use based on her battery level
- The option to plan ahead by checking and booking chargers for a future time
- A secure way to save and manage her EV details (battery size, range, compatibility)
- Access to a personal dashboard where she can view past charging sessions
- Insight into her electricity usage and total spending over time
- A seamless way to start, stop, and pay for a session via the app

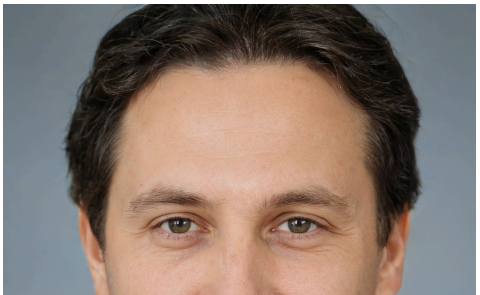
Scenario

It's Thursday evening, and Sofia is leaving work with only 25% battery left. While walking to her car, she opens the EV app to find a charger near her gym. She filters for fast chargers and selects a discounted station that's available in 15 minutes. She enters her current battery level, reviews the estimated time and cost, and confirms the reservation.

When she arrives, she unlocks the charger using the app, starts the session, and begins charging. While waiting, she checks her dashboard to view her total charging costs this month. Once done, she ends the session, pays, and receives a receipt along with a session summary.

Station Employee: João Ramos

"I just want an easy way to see what's working, flag what's not, and keep things running smoothly without complications."



Name: João Ramos

Age: 42

Gender: Male

Occupation: Employee at an EV charging station

Location: Porto, Portugal

Income: Middle income

Tech familiarity: Comfortable with basic business software; not highly technical

João is a station employee responsible for keeping chargers operational and available. He uses a desktop system to monitor charger status and relies on a simple, visual dashboard to mark chargers as under maintenance or available. He prefers tools that require minimal training and offer fast updates.

Goals:

- See real-time status of all chargers
- Mark chargers as under maintenance or available
- Prevent users from reserving broken chargers
- Minimize downtime by acting fast

Frustrations:

- Disjointed tools or poor UX
- Discovering issues too late
- No centralized view of charger statuses

Behaviours:

- Uses system mostly from a desktop
- Logs in at start of shift and during issue handling
- Occasionally uses mobile in the field

Needs:

- A clear dashboard showing the real-time status of all chargers at his station
- A simple method to mark chargers as under maintenance to avoid customer issues
- The ability to restore charger availability once they're repaired
- A straightforward interface that doesn't require technical knowledge
- Minimal steps to update charger status quickly during busy shifts
- A way to ensure that broken chargers aren't reserved by customers
- A consistent process for keeping charger information accurate throughout the day
- Basic access on mobile in case updates need to be made while away from the main desk

Scenario

João begins his morning shift at the Porto EV station by logging into the charger dashboard. He notices that one charger still shows as under maintenance from yesterday. After checking with the technician and confirming the repair, João updates the status to available.

Later, a customer reports a broken cable. João flags the affected charger as under maintenance in the system, instantly removing it from the booking schedule. He notifies his manager to schedule service and continues monitoring the station through the dashboard for the rest of his shift.

2.3 Project epics and priorities

Epics

EPIC 1: Charger Discovery

Allow users to find and compare charging stations using a simplified map or list. Default search uses current GPS location and time. Users can manually set a different location, choose a future time, and filter by charger type. Results show real-time or scheduled availability, with each station showing available chargers for the selected type and time. Map pins and list items display availability counts and discount tags when applicable. Tapping a station opens a detail view with more info and reservation options.

User Stories:

US1: View nearby chargers on map

As a user
I want to see available charging stations near me when I open the app
So that I can quickly check availability in my area
Acceptance Criteria:

- Default location is current GPS
- Default time is “now”
- Stations appear as icons on the map
- Tapping a station opens its detail view

US2: Set a custom location

As a user
I want to change the search location
So that I can explore availability in another area
Acceptance Criteria:

- Location input
- Map and list update to new location
- Distance values update relative to this new point
- Switching back to “this location” resets to current GPS location

US3: Set a custom time

As a user
I want to choose a future time
So that I can check availability when I plan to charge
Acceptance Criteria:

- Time picker (date + time) is available
- Results update for selected time
- Switching back to “now” resets to real-time availability

US4: Filter by charger type

As a user
I want to filter by charger type
So that I only see availability for the type my EV supports
Acceptance Criteria:

- Filter options shown clearly
- Stations only show availability count for selected type(s)
- Map/list updates in real time

US5: View detailed information for a selected station

As a user
I want to select a station and view more detailed information
So that I can decide if it suits my needs before reserving a charger
Acceptance Criteria:

- User can tap a station pin (map) or item (list)
- Detail view includes:
  - Station name and address
  - All the station available chargers
  - Price per kWh (default)
- Option to proceed to reservation from this screen

US6: Highlight discounted stations in search results

As a user
I want to see which stations have a discount for my selected time
So that I can take advantage of cheaper charging options
Acceptance Criteria:

- When a discount is active for a station and selected time:
- A label or tag (e.g., "15% off") is shown on the station card (list view) and pin (map view)
- Tag updates when user changes time or charger type filters
- Clicking the station still opens full detail view

EPIC 2: User Profile Management

Allow users to create, store, and update their EV-related profile (battery capacity, range, charger compatibility). Profile data is collected during sign-up, loaded on login, and editable via settings. This ensures accurate filtering and estimates throughout the app.

User Stories:

US7: Sign up logic

As a new user
I want to create an account and enter my EV details during sign-up
So that I can start searching and make reservations
Acceptance Criteria:

- Sign-up form includes:
  - Email and password
  - EV data: battery capacity, full range
- User is logged in after registration

US8: Log in logic

As a returning user
I want to log into my account
So that I can access my reservations
Acceptance Criteria:

- Login requires email and password
- On successful login:
  - User profile is loaded
- Incorrect credentials show an error

US9: Update EV information

As a user
I want to update my EV data
So that I can change settings or update my car specs
Acceptance Criteria:

- Profile can be updated from settings
- All future calculations reflect the updated data

EPIC 3: Charging Flow & History

Enable users to reserve, manage, and track charging sessions end-to-end. Users can select a station, future time, and input current battery level to get accurate estimates for duration, cost, and energy needs. Reservations only succeed if the charger is available for the full session. During the session, users start charging with a valid code, track simulated energy delivery, and complete payment. Afterward, users can view past sessions and monthly summaries through their dashboard.

User Stories:

US11: Reserve a charger with time and battery input

As a user
I want to select a charger, choose a time slot, and enter my battery level
So that I can get an accurate cost and duration estimate, and reserve the charger for the full time I'll need
Acceptance Criteria:

- User selects a station, date, and time
- User must input current battery %
- System uses profile + charger metadata to calculate:
  - Estimated duration
  - Cost
  - Expected end time
- Reservation is only possible if charger is free for full estimated duration
- Confirmation screen shows all values clearly before booking

US12: View upcoming reservations

As a user
I want to see my scheduled reservations
So that I can prepare for upcoming sessions
Acceptance Criteria:

- List shows station, start time, and estimated end time
- Tapping a reservation shows full info

US13: Cancel a reservation

As a user
I want to cancel a reservation
So that I don't hold a slot I no longer need
Acceptance Criteria:

- Can cancel any upcoming reservation
- Slot is freed in the schedule

US14: Simulate payment

As a user
I want to review my session and go through a simulated payment
So that I can complete the charging process like I would in a real-world experience
Acceptance Criteria:

- System shows:
  - Calculated cost
- User taps "Pay" to trigger simulated payment
- System shows confirmation screen
- Session is marked as paid and saved to history

US15: View list of past charging sessions

As a user
I want to see all my previous charging sessions
So that I can track how much I've charged and spent
Acceptance Criteria:

- Past sessions listed by most recent first
- Each session shows:
  - Date and time
  - Station name
  - Charging duration
  - Cost paid

EPIC 4: Station Operations

Enable station employees to monitor and manage the operational status of chargers. Staff can view a live dashboard showing each charger's availability, flag chargers as under maintenance to prevent reservations, and restore them once repaired to keep station data accurate and up to date.

User Stories:

US16: View charger status dashboard

As a station employee
I want to see the status of all chargers at my station
So that I can monitor operations and know which ones are available
Acceptance Criteria:

- Dashboard shows list of chargers
- Each charger shows its current status:
  - Available
  - In use
  - Under maintenance

US17: Flag a charger for maintenance

As a station employee
I want to mark a charger as under maintenance
So that I can prevent users from reserving it until it's repaired
Acceptance Criteria:

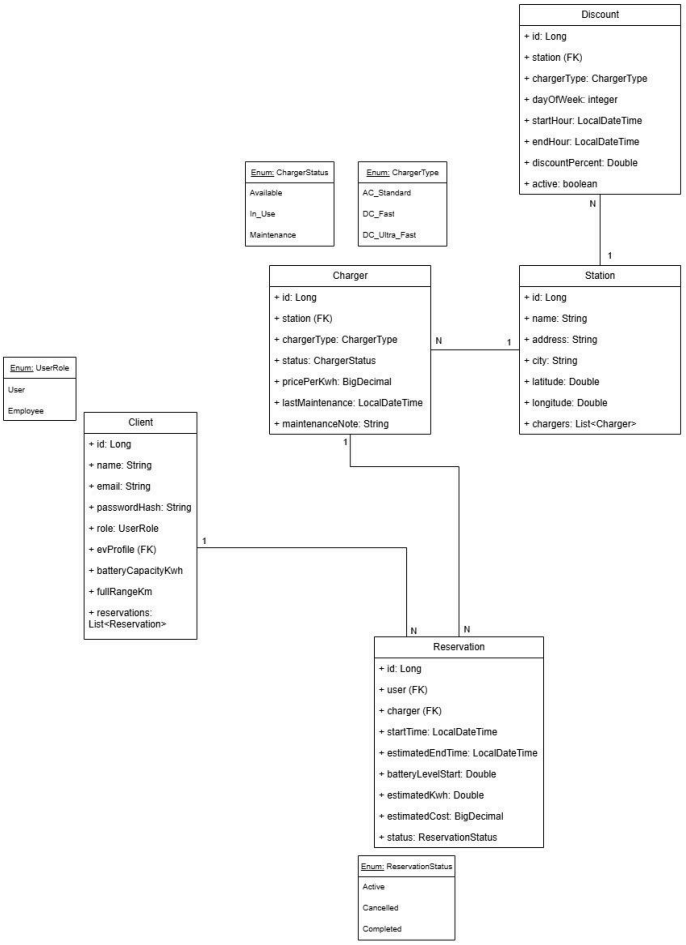
- Option to mark any charger as "under maintenance"
- Charger becomes unavailable for reservation immediately

US18: Unflag a repaired charger

As a station employee
I want to mark a charger as available again
So that I can restore it for user reservations
Acceptance Criteria:

- Option to change charger status from "under maintenance" to "available"
- Charger becomes reservable again

3 Domain model



The **domain model** for NikCharge organizes the essential business objects involved in managing EV charging stations. Clients (users) can create reservations for chargers at specific stations. Each station contains multiple chargers, which may be of different types (e.g., AC standard, DC fast) and have statuses indicating their availability. Reservations track who booked which charger, for when, and at what cost. The system also supports discounts, which can be set by station employees for particular stations, charger types, times, or days to offer dynamic pricing. All of this is linked by clearly defined relationships, ensuring that bookings, payments, station operations, and discounts are handled efficiently across the platform.

4 Architecture notebook

4.1 Key requirements and constrains

The architecture of the NikCharge system is the result of an in-depth analysis of both functional needs and broader architectural characteristics. These requirements guide the selection of technologies and the organization of system components to ensure performance, maintainability, scalability, and adaptability within the context of this course.

This section identifies key issues, constraints, and design drivers that shape the proposed architecture.

4.1.1 Functional Requirements

These are the primary capabilities the system must provide to end users in order to fulfill the business and operational goals defined in the case study:

1. **Multi-role user management**: The system must support different user types (e.g., EV drivers and station operators) with tailored access and functionality.
2. **Charging station search and discovery**: Users must be able to locate available charging stations, using filters such as location, real-time availability, and charger type.
3. **Slot booking and availability management**: Booking must be handled to prevent overlaps or overbooking, with real-time updates.
4. **Initiation and monitoring of charging sessions**: The system must allow users to unlock a charger and monitor electricity consumption during the session.
5. **Payment support**: Simulated integration with a payment service must support one-time payments.
6. **Operator tools (backoffice)**: Operators must be able to manage stations (add, update, configure schedules and pricing).
7. **User dashboard**: Drivers should access their consumption history and charging behavior in an informative and user-friendly way.
8. **Map and data visualization**: Real-time map integration must display current station data, usage trends, and historical insights.

4.1.2 Non-Functional Requirements

To ensure a robust, adaptable, and production-ready system, several architectural characteristics must be considered. These will guide key design decisions, support future scalability, and ensure the system performs reliably across different environments and use cases:

1. **Scalability:** The system must handle an increasing number of users and stations. APIs and databases must support concurrent read/write operations with consistent performance.
2. **Performance and Responsiveness:** Real-time updates are critical for operations like booking and status updates. The system must remain performant even under high user load or peak access times.
3. **Platform Support (User Interface):** The frontend must be responsive and accessible across desktop web browsers, mobile web interfaces, and optionally adaptable to larger screens.

4. **Integration with External Systems:** The backend integrates with a map API, a mock payment gateway, and third-party services. These must be abstracted for testability and loose coupling.
5. **Hardware Abstraction:** While the MVP does not directly control hardware, the architecture must isolate hardware-related logic to accommodate potential future IoT integration.
6. **Fault Tolerance and Resilience:** The system must implement retry strategies, timeouts, and circuit breakers to handle external API failures gracefully.
7. **Security:** Authentication and role-based authorization must be implemented, with secure communication and protection of sensitive data.
8. **Maintainability and Testability:** The architecture must support clear module separation, automated testing pipelines, and CI/CD practices.
9. **Observability:** Logging, monitoring, and alerting mechanisms should be in place to support debugging and performance tracking.
10. **Portability:** The system will be Dockerized to ensure consistent deployment across different environments.

4.1.3 Architectural Design Drivers

The following issues have particularly influenced our architecture:

Driver	Architectural Decision
Need for responsive UI on web/mobile	Use of React for frontend with responsive design principles
Integration with multiple external APIs	Use of Spring Boot to modularize and abstract API clients
Potential physical device integration in the future	Separation of hardware-facing logic into isolated service components
Scalability and extensibility needs	Modular monolith with potential for microservice evolution
Need for user dashboards and analytics	Use of stateful backend services that track session and consumption data
Development speed with future maintainability	Use of CI/CD tools and Docker-based development environments

This collection of architectural characteristics ensures that the system not only meets its immediate functional goals but is also technically sustainable for future extensions and testing, aligned with the pedagogical objectives of the TQS course.

4.2 Architecture view

The architecture for NikCharge follows a layered architectural style, organizing the software into distinct functional blocks that interact through well-defined interfaces. The main building blocks are grouped into packages or modules according to their business responsibilities.

The main logical packages are:

- **User Management** (handling registration, authentication, and profile management)
- **Station & Charger Management** (managing stations, their chargers, and their statuses)
- **Reservation Management** (handling creation and cancellation of reservations)
- **Discount Management** (enabling station managers to configure and apply discounts)
- **Payment Integration** (coordinating payments with an external service, Stripe)

All communication between the modules is typically via standard HTTP protocols, with REST conventions applied for API endpoints. The core system follows separation of concerns, where the API layer never contains business logic and the business logic never directly interacts with the database. Repositories are responsible for that.

Interactions Between Modules:
For example, when a user makes a reservation:

1. The API receives the reservation request and forwards it to the Reservation Service.

2. The Reservation Service validates availability, calculates cost (possibly applying discounts), and updates the relevant Charger and Reservation entities.

3. Since payment is required, the Reservation Service communicates with the Payment Integration module, which in turn talks to the external payment provider.

4. All changes are persisted through the appropriate repositories.

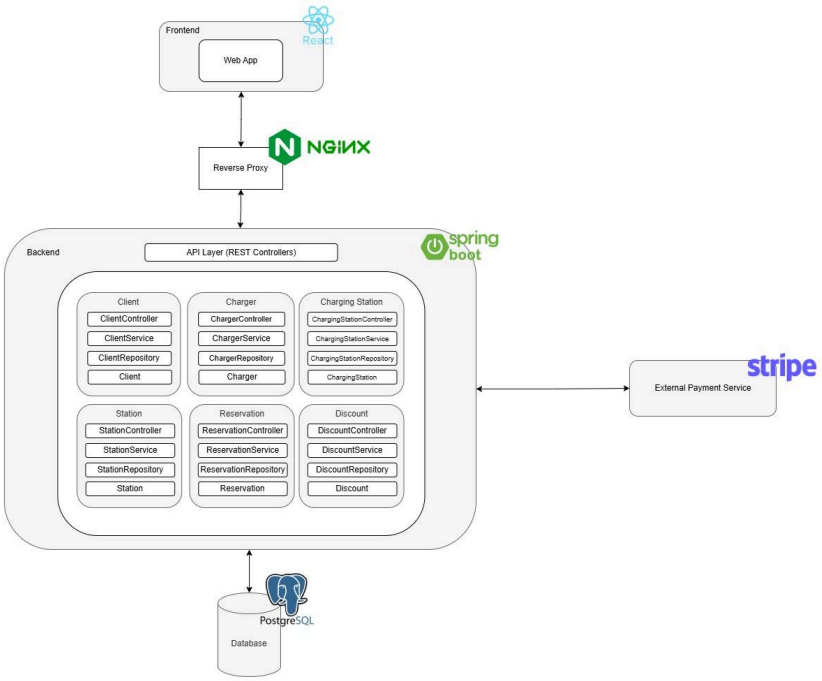
Advanced Design Considerations:

- **External Service Integration:** The system integrates with third-party services (Stripe, for payment processing), using secure HTTP protocols and standardized APIs.

- **Data Synchronization:** Consistency is maintained using transactional updates within the service layer. For real-time availability, the architecture supports near real-time updates, but global distributed synchronization is not required for this MVP.

- **Distributed Workflows:** While the main application is monolithic at this stage, the logical separation allows for future migration to a microservices architecture, where services like payments or trip planning could run as independent modules.

4.3 Deployment view



This diagram provides an overview of the high-level architecture of your NikCharge platform. It illustrates how the various technological layers and core components interact to deliver a seamless experience for users seeking to locate, reserve, and pay for EV charging services.

At the top of the stack, we have the **Frontend Web Application**, built with React. This is the user-facing part of your system, enabling customers and station employees to interact with the platform, from searching for charging stations, making reservations, and viewing their profiles, to accessing dashboards.

To manage incoming HTTP requests and to serve static content efficiently, we've placed an **Nginx Reverse Proxy** between the frontend and backend.

The **Backend** is implemented using Spring Boot. The backend is organized into modular domains (Client, Charger, Charging Station, Reservation, Discount, and Station) each following a typical layered architecture (Controller, Service, Repository, and Model). This structure ensures clear separation of concerns, maintainability, and scalability. All interactions between the frontend and backend occur via a well-defined REST API.

A **PostgreSQL database** serves as the persistent storage layer, holding all business data: user accounts, reservations, charging station information, charger status, discount configurations, and more. The backend communicates with the database to read and write information as needed for each use case.

A key feature of this platform is secure, real-world payment processing. To handle this, the backend integrates with **Stripe**, an external payment service. In order for a user to complete a reservation, the backend communicates directly with Stripe's API to process payments securely and reliably.

Together, these components create a robust, modular, and extensible system architecture that supports all the core workflows outlined in our project vision, from real-time search and reservations to secure payments and administrative management.

Deployment Configuration and Production Setup
The planned deployment of the NikCharge solution is fully containerized using Docker, leveraging a multi-container architecture to ensure modularity, scalability, and ease of management. The primary components of the system are organized into separate services, each running in its own container:

- **Frontend:** Runs the React-based user interface, built and served via a Node.js environment. It communicates with the backend through HTTP calls and is exposed internally to other containers via the Docker network.

- **Backend:** Powered by Spring Boot, this service encapsulates all business logic, REST APIs, integration with the database, and connection to external services such as Stripe for payment processing. The backend is built using the configuration and listens on port 8080.

- **Database:** The system relies on a PostgreSQL instance to persist all domain data, including users, reservations, stations, and related entities. Credentials and connection parameters are specified in the .env file.

- **Nginx Reverse Proxy:** An Nginx container sits in front of the frontend and backend, acting as a reverse proxy to route incoming HTTP requests. Nginx listens on port 80 and forwards /api/ and /actuator/ requests to the backend, while all other traffic is routed to the frontend.

- **Stripe (External Service)**: Payment processing is handled via the Stripe API. The backend communicates with Stripe over the internet using the official Stripe SDK. Stripe itself is not deployed as a container, but its integration is essential to the solution.

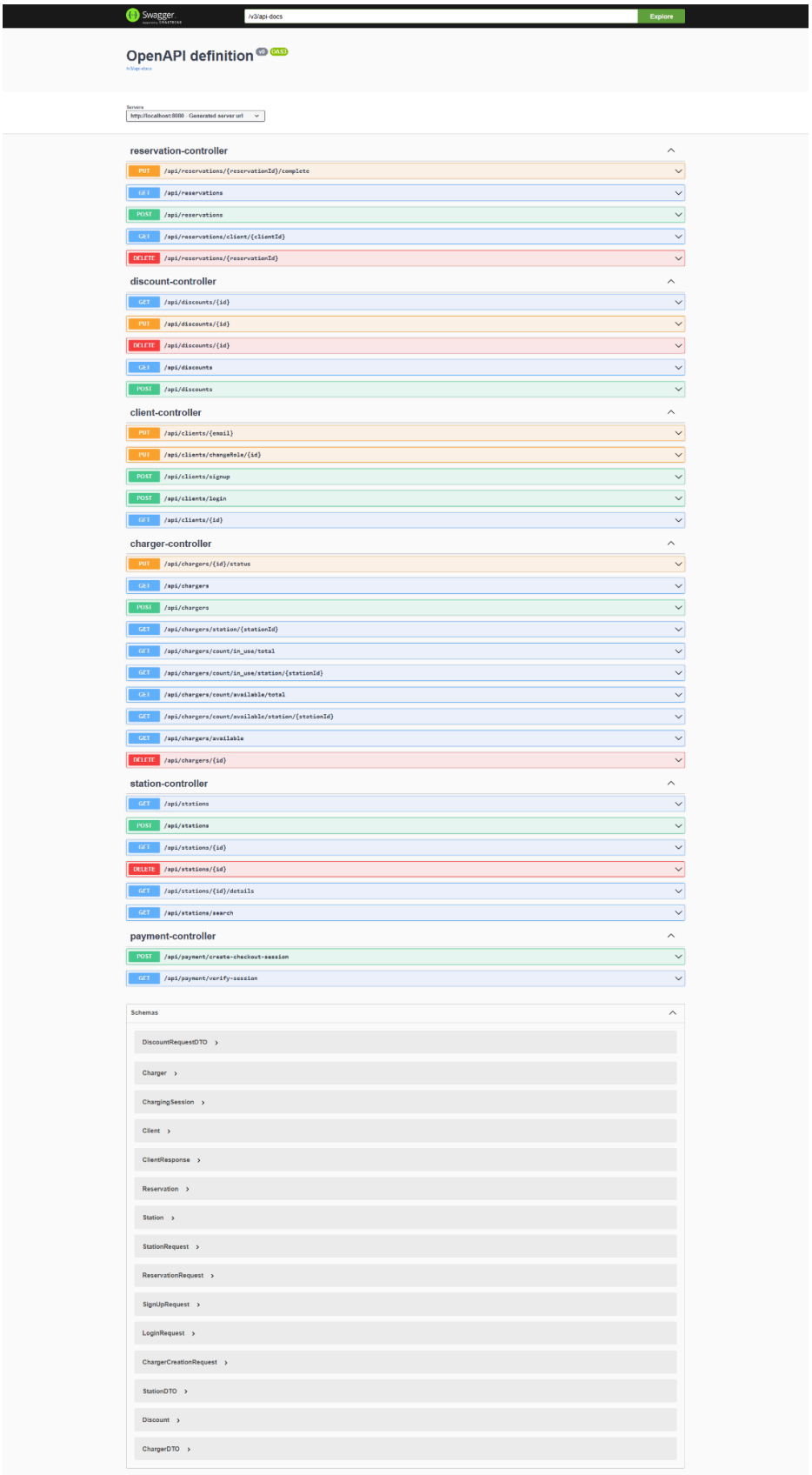
Network and Container Configuration
Docker Compose is used to orchestrate the deployment. Each major component (frontend, backend, database, nginx) is defined as a service within docker-compose.yml, with appropriate network aliases.

- Ports:
  - Nginx listens on port 80 and acts as the single public entrypoint.
  - Backend is exposed internally on port 8080.
  - Frontend is exposed internally on port 80 (served via Node/NGINX in the container).
  - PostgreSQL is exposed internally on port 5432.

Environment Variables are used to inject secrets (like DB credentials, Stripe keys) and runtime configuration.

IPs and Hostnames
Within the Docker network, services are accessible via their service names:
- frontend
- backend
- db (PostgreSQL)
- nginx (as entrypoint)
- Nginx is mapped to the host's port 80, exposing the entire application at http://<server-ip>/.

5 API for developers



The backend of our EV charging platform exposes a RESTful API, following best practices for resource-oriented design. The API is structured around the key business entities of the system, each managed by its own controller. All endpoints are documented and can be explored and tested interactively via Swagger UI at /swagger-ui/index.html.

Main API Collections

- **Clients**: Endpoints under /api/clients allow for client registration (signup), authentication (login), updating client information, and role management. Clients can also be retrieved by ID.

- **Reservations**: Endpoints under /api/reservations manage reservation creation, completion, retrieval (all, or by client), and cancellation. This allows users to book, view, and cancel charging slots.

- **Stations**: Endpoints under /api/stations support creation, retrieval, search, and deletion of charging stations. Additional endpoints are provided for detailed station data and filtered station search based on parameters such as charger type, availability, and time slot.

- **Chargers**: Endpoints under /api/chargers manage charger entities. This includes creating chargers, updating their operational status (available, in use, under maintenance), counting chargers by status (globally or per station), and listing chargers available at a given time or station.

- **Discounts**: Endpoints under /api/discounts allow for the creation, retrieval, update, and removal of discounts.

- **Payments**: Endpoints under /api/payment manage payment flows via Stripe integration. For example, /api/payment/create-checkout-session creates a new payment session and /api/payment/verify-session verifies its status.

Resource-Oriented Approach

All API endpoints are structured following RESTful conventions:

- Resource names are plural nouns (e.g., /clients, /reservations, /stations).
- HTTP methods map to actions on resources (e.g., GET to retrieve, POST to create, PUT to update, DELETE to remove).
- Resource identifiers are used in path parameters for specific actions (e.g., /api/reservations/{reservationId}).

API Documentation

The complete, always up-to-date documentation of all available endpoints, schemas (DTOs), and request/response models is provided through Swagger UI, accessible at /swagger-ui/index.html. This allows both developers and stakeholders to explore the API, understand expected inputs and outputs, and even test API calls directly from the browser.