TQS: Product Specification Report

SportsLink

Sports Facility Rental Platform

Authors:

Paulo Cunha	Team Coordinator
Tomás Hilário	Product Owner
Rafael Ferreira	QA Engineer
Diogo Duarte	DevOps Master

University of Aveiro
Software Testing and Quality (TQS)

Version: December 2025

Contents

1 Introduction

1.1 Overview of the Project

This project was developed within the scope of the **Software Testing and Quality (TQS)** course, aiming to apply **Software Quality Assurance (SQA)** practices and **DevOps** methodologies in a realistic, medium-sized software development scenario. The team implemented a full-stack application using **Spring Boot**, applying agile methodologies with continuous integration, continuous testing, and continuous delivery pipelines.

What is SportsLink?

SportsLink is a digital marketplace platform that connects sports facility owners with sports enthusiasts looking to rent venues and equipment on-demand. The application addresses a critical inefficiency in facility usage: owners have idle capacity during off-peak hours, while potential users lack convenient access to affordable sports facilities.

1.1.1 Target Audience

- **For Facility Owners:** Monetize underutilized sports facilities through rentals, manage bookings and equipment inventory, track maintenance schedules, and monitor occupancy rates.
- **For Renters:** Discover nearby sports fields filtered by sport type, location, and availability; easily book fields for specific time slots; access quality assurance information through reviews and ratings.
- **For Platform Admins:** Monitor platform health, analyze usage metrics, manage user partnerships, and ensure compliance and safety standards.

Product Name	SportsLink
Domain	Sports facility and equipment rental
Initial Focus	Team sports (padel, futsal, tennis, basketball, volleyball)

1.2 Project Limitations and Known Issues

1.2.1 Current Limitations

Component	Status
Review System	Planned functionality but not fully implemented (Epic 7)
Community Features	Occupancy indicators and grouping (Epic 9) not implemented
Notification System	Email and SMS confirmation (Epic 10) not fully implemented
Admin Dashboard	Advanced analytics (Epic 8) partially implemented

1.2.2 Reasons for Limitations

- Prioritization of core functionalities (Epics 1-5) for the MVP.
- Academic semester time constraints.
- Focus on the quality and robustness of implemented features rather than quantity.

1.3 References and Resources

1.3.1 Frameworks and Libraries

Technology	Description
Spring Boot 3.5.7	Main application framework
Spring Data JPA	Data persistence
Spring Security	Authentication and authorization
Lombok	Boilerplate code reduction
JWT	Stateless authentication

1.3.2 Database and Storage

Technology	Description
PostgreSQL 16	Production database
H2 Database	Testing database
MinIO	Image storage (S3-compatible)

1.3.3 Testing Tools

Tool	Test Type
JUnit 5	Unit testing
Mockito	Mocking framework
Cucumber 7.14.0	BDD testing
Selenium WebDriver	Functional testing
Testcontainers	Integration testing with containers
RestAssured	API testing
k6	Performance testing

1.3.4 Quality and CI/CD

- **SonarQube Cloud** – Static code analysis
- **XRay** – Test case management
- **Jira** – Defect and backlog management
- **Docker** – Containerization

2 Product Concept and Requirements

2.1 Vision Statement

SportsLink aims to solve inefficiencies in sports facility utilization by creating a digital marketplace that:

1. **Connects Supply and Demand:** Allows facility owners to monetize idle capacity while users easily find available venues.
2. **Simplifies the Booking Process:** Replaces traditional methods (WhatsApp, phone calls) with a centralized platform featuring a real-time availability calendar.
3. **Integrated Equipment Management:** Allows renting equipment together with facilities in a single transaction.

2.1.1 Key Features

- Field discovery and search by location, sport type, and availability.
- Booking system with double-booking prevention.
- Facility and equipment management for owners.
- Dashboard with occupancy and revenue metrics.
- Secure authentication with role-based access control.

2.2 Personas and Scenarios

Persona 1: Francisco Garcia Gameiro – The Private Field Owner

Demographics:

- Age: 48
- Occupation: Sports Complex Owner (3 Padel courts + 2 Futsal fields)
- Location: Aveiro, Portugal
- Tech Proficiency: Moderate

Goals: Monetize idle capacity without hiring additional staff. Wants clear visibility on bookings, equipment maintenance, and revenue.
Pains: Managing bookings via WhatsApp and calls, lacking a global view of availability, no maintenance tracking system.
Scenario: Monday at 9 AM, Francisco opens SportsLink and accesses the Owner Dashboard. He views weekly occupancy (Padel 65%, Futsal 45%), checks equipment needing maintenance, schedules repairs, and consults revenue charts.

Persona 2: Maria Floro Silva – The Occasional Renter

Demographics:

- Age: 28
- Occupation: Marketing Specialist
- Location: Aveiro, Portugal
- Tech Proficiency: High

Goals: Book fields quickly when free time arises, without calls or lengthy searches. Values convenience, fair pricing, and transparency regarding quality.
Pains: Unaware of real-time availability, different processes for each facility, concerns about quality and maintenance.
Scenario: Thursday at 6 PM, Maria decides to play Padel. She opens SportsLink, filters by "Padel" and "Today, 19h-21h", finds 3 results with ratings and prices, reads reviews, selects a slot, and completes payment in 10 seconds.

Persona 3: Pedro Gustiani Albertino – The Community Builder

Demographics:

- Age: 24
- Occupation: Software Engineer (Remote)
- Location: Aveiro, Portugal
- Tech Proficiency: Very High

Goals: Find people to play with at specific times, build a sports community.
Pains: Difficult to find players for pickup games, disorganized WhatsApp groups.
Scenario: Pedro searches for Futsal for Wednesday at 7 PM, discovers occupancy indicators showing other players at the same time, starts a group chat, and meets new game partners.

Persona 4: Ana Bernolli Jacoco – Platform Administrator

Demographics:

- Age: 42
- Occupation: SportsLink Operations Manager
- Location: Lisbon (Remote)
- Tech Proficiency: Moderate

Goals: Monitor platform health, identify issues, generate reports for investors.
Scenario: Monday at 10 AM, Ana accesses the Admin Dashboard. She views KPIs (active users, bookings, revenue, uptime), identifies facilities with low ratings, contacts owners, and exports a weekly report.

Persona 5: Sofia Mendes – Casual Group Organizer

Demographics:

- Age: 35
- Occupation: HR Manager
- Location: Lisbon, Portugal
- Tech Proficiency: Moderate to High

Goals: Organize weekly volleyball games for colleagues, reserve field and equipment in a single transaction.
Scenario: Tuesday at 1 PM, Sofia searches for Volleyball for that night, finds "Lisbon Sports Arena", selects a slot, adds 2 volleyballs to the rental (€40 + €10 = €50), completes payment, and shares confirmation via WhatsApp.

2.3 Project Epics and Priorities

Epic	Name	Priority	Description
1	Field Discovery & Catalog	HIGH	Search and filter facilities by sport, location, price
2	Booking & Reservation System	HIGH	Calendar-based bookings with conflict prevention
3	Owner Field Management	HIGH	Facility registration, equipment tracking, dashboard
4	Payment & Transactions	HIGH	Stripe integration, secure payments
5	User Management & Auth	HIGH	Registration, login, role-based access control
6	Renter Dashboard & History	MEDIUM	Booking history, favorites, quick rebooking
7	Reviews & Trust System	MEDIUM	Ratings, reviews, reputation scoring
8	Admin Dashboard & Analytics	MEDIUM	KPI dashboard, metrics, and reports
9	Community Features	LOW	Occupancy indicators, profiles, group chat
10	Notifications & Communication	LOW	Email notifications, booking reminders

Table 1: Project Epics and Priorities

Release Scope

- **MVP (I1-I4):** Epics 1-5
- **Future Work (FW):** Epics 6-10

3 Domain Model

3.1 Class Diagram

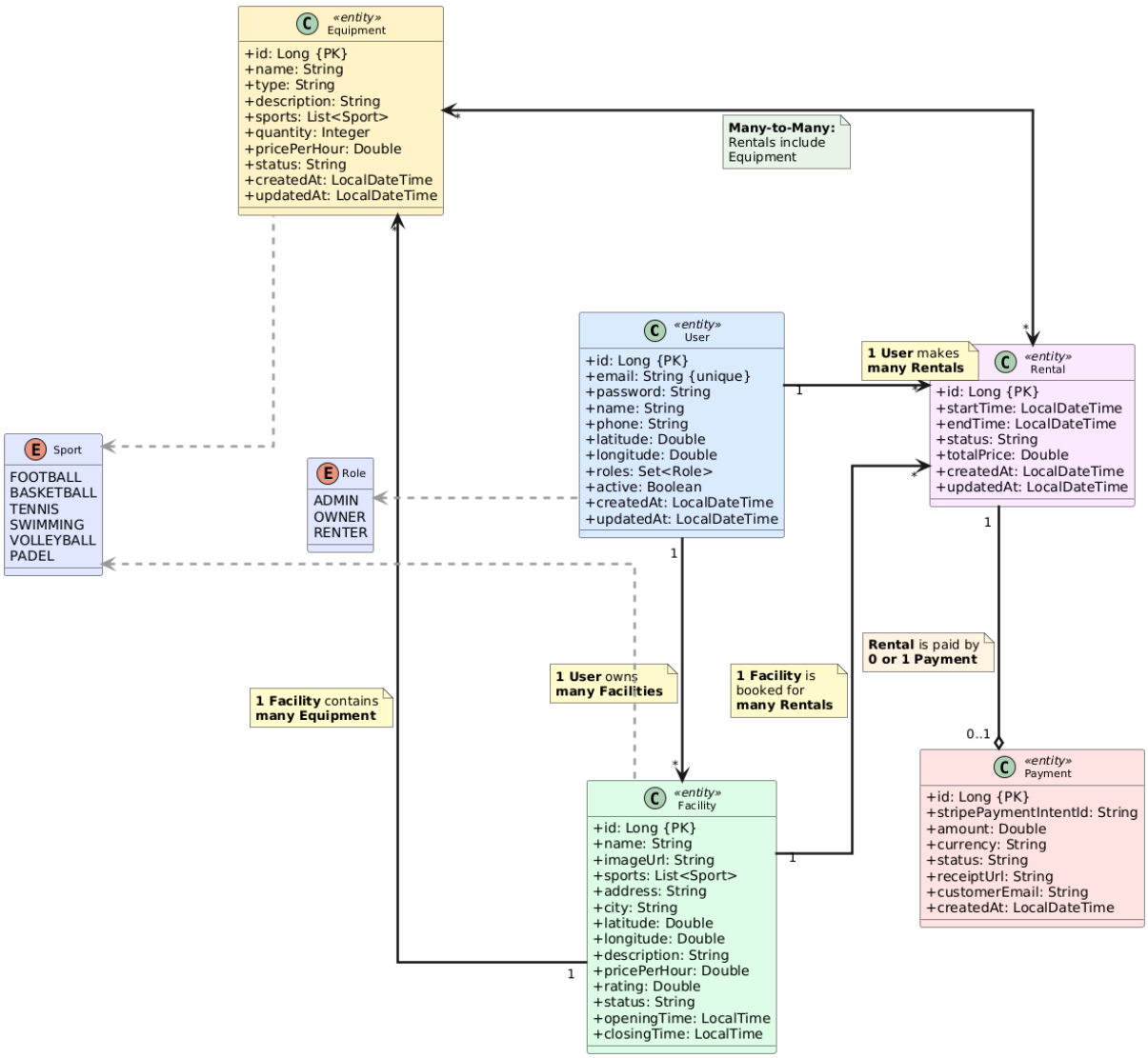

Figure 1: Domain Model

3.2 Entity Descriptions

Entity	Description
User	Represents all platform users. Can have multiple roles (ADMIN, OWNER, RENTER) and own facilities or make reservations.
Facility	Sports facility with information on location, price, supported sports, and operating hours. Belongs to an owner.
Equipment	Equipment available for rent (rackets, balls, nets, etc.). Associated with a specific facility.
Rental	Represents a reservation/rental with a time period, status, and total price. Links user, facility, and equipment.

Table 2: Domain Entity Descriptions

3.3 Relationships

- **User → Facility:** A user (OWNER) can own multiple facilities (1:N).
- **User → Rental:** A user (RENTER) can have multiple reservations (1:N).
- **Facility → Equipment:** A facility contains multiple pieces of equipment (1:N).
- **Facility → Rental:** A facility can have multiple reservations (1:N).
- **Rental ↔ Equipment:** A reservation can include multiple pieces of equipment (N:M).

4 Architecture Notebook

4.1 Key Requirements and Constraints

4.1.1 Non-Functional Requirements

Requirement	Goal	Description
Performance	< 1s	Search response time for 100+ facilities
Performance	< 3s	Reservation processing time
Availability	$> 99.5\%$	System uptime
Security	JWT 24h	Authentication with 24-hour expiration
Security	Bcrypt	Password hashing
Scalability	Stateless	Support for concurrent users
Integrity	DB-level	Double-booking prevention via constraints

Table 3: Non-Functional Requirements

4.1.2 Technical Dependencies

- **Java 21** – Java LTS Version
- **Spring Boot 3.5.7** – Application Framework
- **PostgreSQL 16** – Relational Database
- **MinIO** – S3-compatible Object Storage
- **Docker** – Containerization

4.1.3 External Integrations

- **Stripe API** – Payment processing (PaymentIntents + Webhooks)
- **MinIO** – Facility image storage

4.1.4 Stripe Integration (Payments)

The payment system is implemented with full Stripe integration. Figure 2 illustrates the complete payment flow.

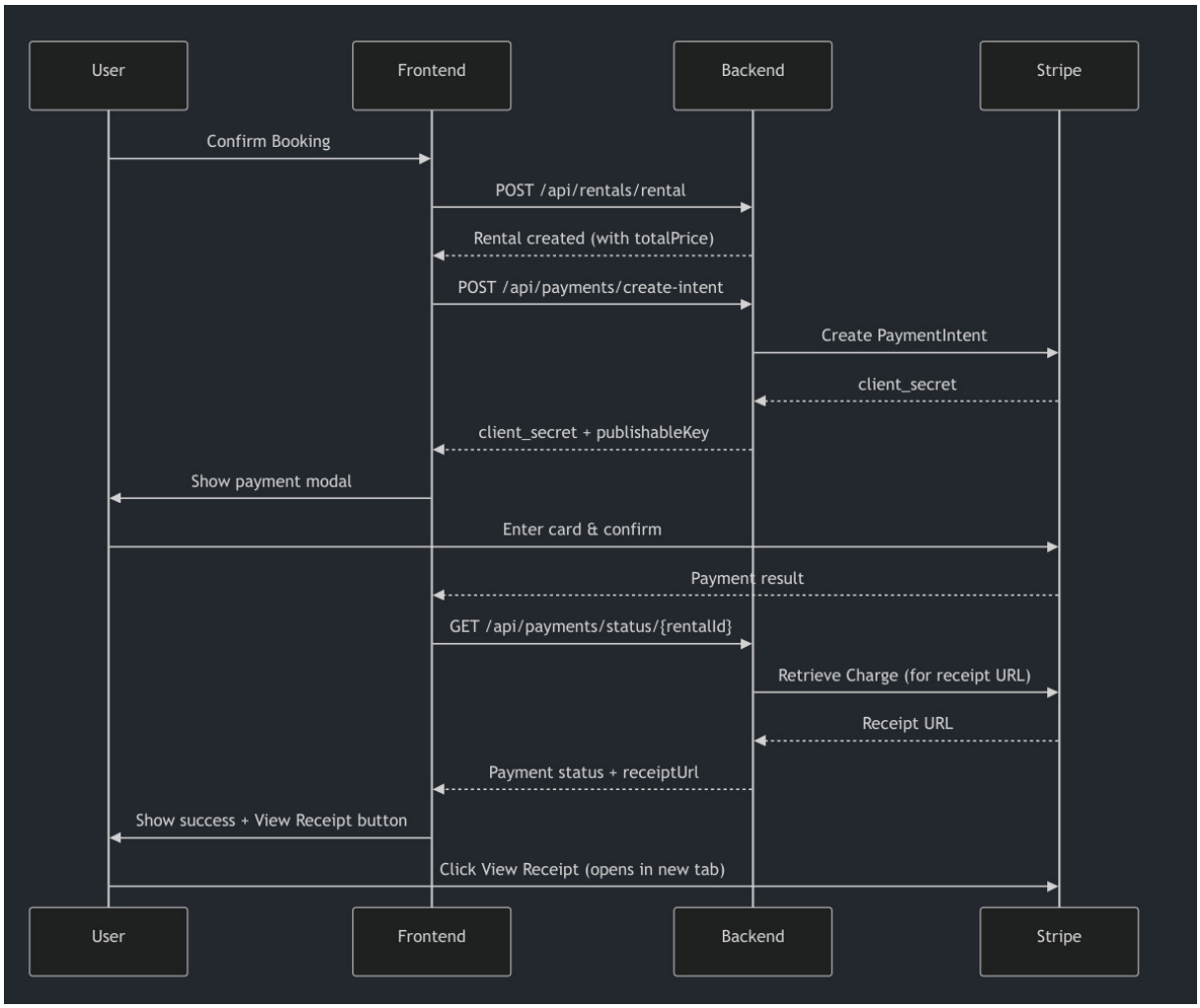

Figure 2: Sequence Diagram of the Payment Flow

Payment Flow:

1. **Booking Confirmation:** User confirms booking on frontend.
2. **Rental Creation:** Frontend sends `POST /api/rentals/rental` and receives the created rental with `totalPrice`.
3. **PaymentIntent Creation:** Frontend requests `POST /api/payments/create-intent/{rentalI`
4. **Stripe Communication:** Backend creates PaymentIntent on Stripe API.
5. **Client Secret Return:** Stripe returns `client_secret`, which backend sends to frontend along with `publishableKey`.
6. **Payment Modal:** Frontend shows Stripe.js payment form.
7. **Card Confirmation:** User enters card details and confirms.
8. **Payment Result:** Stripe processes and returns result to frontend.
9. **Status Verification:** Frontend polls via `GET /api/payments/status/{rentalId}`.
10. **Receipt Retrieval:** Backend queries Stripe to get `receiptUrl`.
11. **Final Confirmation:** Frontend shows success message and button to view receipt.

Note: The system also supports webhooks (`POST /api/payments/webhook`) for asynchronous Stripe notifications, ensuring consistency even in cases of client network failure.

4.2 Architecture View

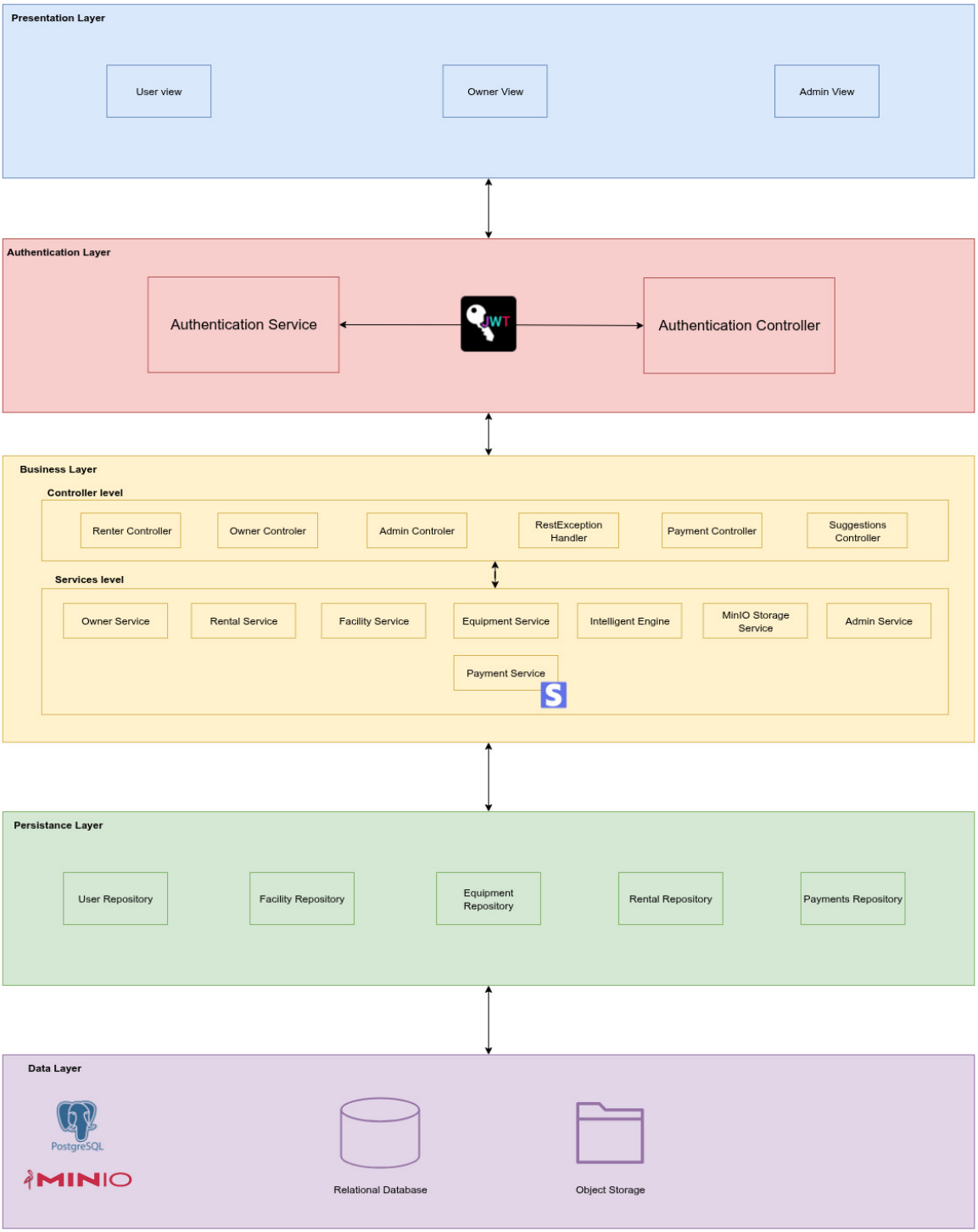

Figure 3: Logical System Architecture (Layered Architecture)

4.2.1 Modules and Responsibilities

Module	Responsibility
`boundary/`	REST Controllers – API endpoints
`service/`	Business logic and orchestration
`data/`	JPA Repositories and domain models
`dto/`	Data Transfer Objects for request/response
`config/`	Configurations (Security, JWT, MinIO)
`util/`	Utilities and helpers

Table 4: System Modules

4.3 Deployment View

The application is deployed using Docker Compose, orchestrating multiple containers in an isolated bridge network. Figure 4 illustrates the complete deployment configuration.

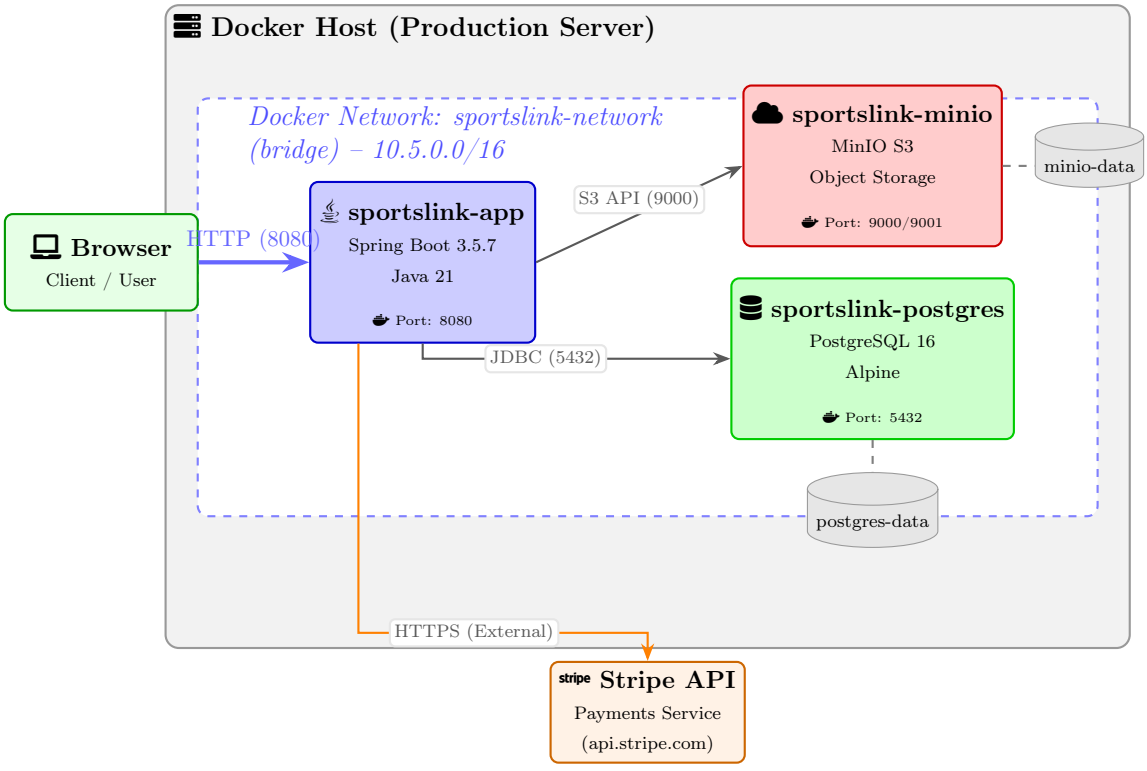

Figure 4: Deployment View - Arquitetura de Containers Docker

4.3.1 Component Descriptions

Component	Description
sportslink-app	Main container with the Spring Boot application. Exposes the REST API on port 8080 and serves static frontend files.
sportslink-postgres	PostgreSQL 16 database for data persistence. Communicates internally via port 5432.
sportslink-minio	S3-compatible object storage for facility images. API on port 9000, console on 9001.
Stripe API	External service for payment processing. Communication via HTTPS.

Table 5: Deployment Component Descriptions

4.3.2 Services and Configurations

Service	Image	Port(s)	Function
app	Build local	8080	Spring Boot Application
postgres	postgres:16-alpine	5432	PostgreSQL Database
minio	minio/minio	9000, 9001	Object storage for images
createbuckets	minio/mc	–	Bucket initialization

Table 6: Docker Services

4.3.3 Persistent Volumes

- **postgres-data:** PostgreSQL data
- **minio-data:** Image files

5 API for Developers

5.1 General Description

The **SportsLink** REST API is organized by functional domains and follows REST principles. All protected routes require JWT authentication via the `Authorization: Bearer <token>` header.

API Reference (Swagger UI) The complete and interactive documentation for all API endpoints is available through the Swagger UI, accessible in the development environment at: `http://192.168.160.31:8080/swagger-ui/index.html`. This interface displays all resources grouped by functional domain, including supported methods, parameters, data models, and example requests and responses. It also allows developers to test each endpoint directly, simplifying the integration and validation process.

5.2 Main Endpoints

5.2.1 Authentication (`/api/auth`)

Method	Endpoint	Description	Auth
POST	`/api/auth/register`	Register new user	No
POST	`/api/auth/login`	Login and obtain JWT	No

5.2.2 Rentals (`/api/rentals`)

Method	Endpoint	Description	Auth
GET	`/api/rentals/search`	Search facilities	No
GET	`/api/rentals/sports`	List available sports	No
GET	`/api/rentals/facility/{id}/equipments`	Facility equipment	No
POST	`/api/rentals/rental`	Create reservation	RENTER
PUT	`/api/rentals/rental/{id}/cancel`	Cancel reservation	RENTER
PUT	`/api/rentals/rental/{id}/update`	Update reservation	RENTER
GET	`/api/rentals/rental/{id}/status`	Reservation status	RENTER
GET	`/api/rentals/history`	Reservation history	RENTER

5.2.3 Owners (/api/owner)

Method	Endpoint	Description	Auth
POST	/api/owner/{ownerId}/facilities	Create facility	OWNER
GET	/api/owner/{ownerId}/facilities	List facilities	OWNER
PUT	/api/owner/.../facilities/{facilityId}	Update facility	OWNER
DELETE	/api/owner/.../facilities/{facilityId}	Delete facility	OWNER
POST	/api/owner/.../equipments	Add equipment	OWNER
PUT	/api/owner/.../equipments/{equipmentId}	Update equipment	OWNER
DELETE	/api/owner/.../equipments/{equipmentId}	Delete equipment	OWNER

5.2.4 Administration (/api/admin)

Method	Endpoint	Description	Auth
GET	/api/admin/users	List users	ADMIN
GET	/api/admin/facilities	List all facilities	ADMIN
GET	/api/admin/rentals	List all reservations	ADMIN

5.2.5 Payments (/api/payments)

Method	Endpoint	Description	Auth
POST	/api/payments/create-intent/{rentalId}	Create PaymentIntent	Yes
POST	/api/payments/webhook	Stripe Webhook	Stripe Sig
GET	/api/payments/status/{rentalId}	Payment status	Yes
GET	/api/payments/config	Stripe public key	No

5.3 Applied REST Best Practices

- **Correct HTTP Verbs:** GET for reading, POST for creation, PUT for updates, DELETE for removal.
- **Appropriate Status Codes:** 200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found.
- **Descriptive URLs:** Resources clearly identified in paths.
- **Stateless Authentication:** JWT tokens in Authorization header.
- **Input Validation:** `@Valid` annotations in request DTOs.
- **Separated DTOs:** Distinct Request and Response DTOs for each operation.

A Project Structure

```
sportslink/
|-- src/
|   |-- main/
|   |   |-- java/tqs/sportslink/
|   |   |   |-- SportslinkApplication.java
|   |   |   |-- boundary/       # REST Controllers
|   |   |   |-- config/         # Security, JWT, MinIO
|   |   |   |-- data/           # Repositories + Models
|   |   |   |-- dto/            # Data Transfer Objects
|   |   |   |-- service/        # Business Logic
|   |   |   |-- util/           # Utilities
|   |   |-- resources/
|   |       |-- application.properties
|   |       |-- static/         # Frontend files
|   |-- test/                   # Test classes
|   |   |-- java/tqs/sportslink/
|   |   |   |-- SportslinkApplicationTests.java
|   |   |   |-- A_Tests_repository/
|   |   |   |-- B_Tests_unit/
|   |   |   |-- C_Tests_controller/
|   |   |   |-- D_Tests_integration/
|   |   |   |-- functionals/
|   |   |   |-- non_functionals/
|-- docker-compose.yml
|-- Dockerfile
|-- pom.xml
```

B QA Tools

Category	**Tool**	**Function**
Unit Testing	JUnit 5, Mockito	Isolated logic validation
Integration Testing	RestAssured, Testcontainers	API and DB validation
Functional Testing	Cucumber, Selenium	E2E flow validation
Performance Testing	k6	Load validation
Static Analysis	SonarQube Cloud	Code quality and security
Test Management	XRay	Test case tracking
Project Management	Jira	Backlog and defects

Table 7: Quality Assurance Tools

C Definition of Done

A user story is considered "Done" when:

- ✓ All acceptance criteria are validated.
- ✓ Unit tests are implemented and passing.
- ✓ Minimum coverage target reached (60%).
- ✓ No critical findings in static analysis.
- ✓ Documentation updated (if necessary).
- ✓ Code review approved.