ADJT - BB - Tech Challenge - Fase 3

Gabriel de Alcantara Rodrigues Soares - RM363700

September 9th, 2025

3. Project Documentation

This document provides a clear overview of the **Hospital App** microservices system. It walks through the architecture, APIs, operational aspects, and step-by-step setup instructions. The goal is to give anyone a **straightforward understanding** of the system and a **quick starting point** for exploring and testing its features using the base admin user.

References to supplementary material within this repository are provided throughout, including:

- **Root README.md** – Implementation and operations quick-start
- **Module-specific READMEs** – Detailed service documentation

The official repository is hosted at: https://github.com/gabriel-dears/hospital_app

3.1 Executive Summary

The **Hospital App** is a microservices-based system designed to manage hospital operations related to users and appointments. It comprises distinct services responsible for:

- **Identity and authentication**
- **Appointment creation and updates**
- **Notification delivery via email**
- **Appointment history retrieval via GraphQL**

Services communicate through **REST**, **gRPC** (with mutual TLS), and **asynchronous messaging** (RabbitMQ over TLS). Security is enforced using **JWT-based resource servers** and **role-based access control**.

A **base admin user** is automatically created on startup using environment variables. This user allows immediate interaction with the system, demonstrating core functionalities such as **login**, **user creation**, and **role-based operations**.

3.2 System Overview

System name: `hospital_app`

Programming language and platform: Java 21, Spring Boot 3.5

Deployment: Docker Compose for local and development environments

Inter-service communication: - **REST (HTTP):** with OpenAPI documentation (Swagger UI) - **gRPC:** with mutual TLS (mTLS) for service-to-service validation - **Asynchronous messaging:** via RabbitMQ (TLS-enabled)

Security:
OAuth2 resource servers with JWT validation using a shared public key; role claims mapped to Spring authorities.

Primary modules/services:

- **user_service:** Identity management, authentication (JWT issuance), user CRUD; exposes REST and gRPC server.
- **appointment_service:** Appointment creation/update; validates user identities via gRPC; publishes appointment events to RabbitMQ.
- **appointment_history_service:** GraphQL queries for the historical timeline of appointments; consumes appointment events.
- **notification_service:** Consumes appointment events and sends emails using template-based HTML content and retry semantics.
- **common:** Shared utilities, DTOs, error-handling helpers, and messaging configuration.
- **jwt_security_common:** Shared JWT resource server configuration and key utilities.
- **proto_repo:** Protobuf definitions and generated gRPC artifacts for inter-service contracts.

Practical note for evaluation:
The system can be explored immediately by logging in with the base admin user created via environment variables, which allows access to all functionalities and operations.

3.3 Architecture

3.3.1 Architectural Style

The system follows a **microservices architecture** with a **hexagonal (ports and adapters) internal organization** for each service.

There is a clear separation of:

- **Inbound adapters**: REST controllers, GraphQL resolvers, message listeners
- **Outbound adapters**: Repositories, gRPC clients, mail senders, message producers

3.3.2 Components and Responsibilities

user_service - REST API for user CRUD (`/user`), authentication (`/login`), and existence checks (`/doctor/{id}/exists`, `/patient/{id}/exists`) - gRPC server for identity verification - Security: JWT resource server for REST; TLS/mTLS for gRPC - Persistence: PostgreSQL

appointment_service - REST API for creating/updating appointments - Outbound gRPC client to `user_service` for validation - Publishes `AppointmentMessage` events to RabbitMQ fanout exchange - Persistence: PostgreSQL

appointment_history_service - GraphQL API for querying appointment histories with filtering and pagination - Consumes `AppointmentMessage` events and persists historical snapshots - Security: JWT resource server; patient scoping for `PATIENT` role - Persistence: PostgreSQL

notification_service - Consumes `AppointmentMessage` events to send HTML emails - Templates rendered with Thymeleaf; retry with Resilience4j - Daily cleanup job for obsolete records - Persistence: PostgreSQL

common: DTOs, error response factories, pagination models, Jackson-based RabbitMQ JSON converter

jwt_security_common: JWT validation utilities and resource server configuration

proto_repo: Protobuf contracts and generated Java classes for gRPC

3.3.3 Cross-Cutting Concerns

- **Authentication and Authorization**: JWT tokens (RS256) issued by `user_service`; verified by `jwt_security_common`. Role claims mapped to Spring authorities (`ADMIN`, `DOCTOR`, `NURSE`, `PATIENT`).
- **Transport Security**: gRPC uses mTLS; RabbitMQ connections over TLS.
- **Observability and Operations**: Docker Compose with health checks; logs via console/docker logs.

3.3.4 Messaging Topology

- **Exchange**: `appointment.exchange` (fanout)
- **Producers**: `appointment_service`
- **Consumers**: `notification_service`, `appointment_history_service`
- **Serialization**: JSON via `Jackson2JsonMessageConverter`

3.4 Interfaces and API Endpoints

Base Admin User - Automatically created from `.env` variables: `INITIAL_ADMIN_USERNAME`, `INITIAL_ADMIN_PASSWORD`, etc. - Allows immediate login and exploration of API functionalities - Acts as starting point for role-based operations and CRUD testing

Base paths and endpoints

- **user_service (REST)**: `/login`, `/user`, `/doctor`, `/patient`
- **appointment_service (REST)**: `/appointment` (create/update)
- **appointment_history_service (GraphQL)**: `/graphql`, `/graphiql`, `/voyager`
- **notification_service**: Message-driven, no public API

> See `GRAPHQL_README.md` for schema and queries. Swagger UIs are available via local proxy or direct service ports.

3.5 Setup and Execution

- Prerequisites: Docker, Docker Compose, JDK 21, Maven
- Environment Configuration: Copy .env.example → .env; adjust credentials and paths.
- Admin User: Base admin user allows immediate login and testing of service functionalities.
- Certificates and Keys: Generate JWT key pair and TLS materials for gRPC and RabbitMQ.
- Build: mvn -q -DskipTests package from repo root
- Run: docker compose up -d (or selectively start services)
- Post-Startup: Verify via Swagger UI, GraphiQL, and RabbitMQ management UI.

3.6 Security Model

- JWT RS256 tokens for authentication
- Role-based access control mapped from token claims
- Transport security via TLS/mTLS

3.7 Data and Persistence

- Separate PostgreSQL databases per service
- Schema managed via Spring Data JPA
- appointment_history_service maintains versioned snapshots

3.8 Operations and Maintenance

- Logs via docker logs
- Credentials stored in .env (SMTP, DB, TLS)
- Certificate rotation handled via scripts and rebuild

3.9 Troubleshooting

- 401/403: verify JWT, roles, key pair
- gRPC mTLS failures: check certificates and client-auth
- RabbitMQ TLS: check keystore/truststore and .env paths
- DB connectivity: ensure containers healthy and credentials correct

3.10 References

- Root operations and quick-start: README.md
- GraphQL schema: GRAPHQL_README.md
- Module READMEs: user_service, appointment_service, appointment_history_service, notification_service, common, jwt_security_common, proto_repo
- Repository: https://github.com/gabriel-dears/hospital_app
- Linkedin: https://www.linkedin.com/in/gabriel-soares-665a18221/