

Tech Challenge FASE 03

Sistema de Gerenciamento Hospitalar

Rafael Detilio - RM362570

Bruna Cristina - RM358934

Vinícius Guimarães – RM361304

Leticia Lucena - RM364673

GitHub: <https://github.com/Detilio/tc-fase-3-v2>

Collections encontram-se no repositório GitHub acima.

Outubro de 2025

1. VISÃO GERAL	4
1.1 Objetivo Geral	4
1.1.1 Escopo	4
1.1.2 Descrição do Projeto	4
1.2 Requisitos Funcionais (alto nível)	4
2. ARQUITETURA	5
2.1 Diagrama de Contexto (C4-L1)	5
2.2 Contêineres/Componentes (C4-L2/L3)	5
2.3 Decisões de Arquitetura (ADRs)	5
3. DOMÍNIO E MODELO DE DADOS	6
3.1 Entidades	6
3.2 Regras de Negócio (principais)	6
3.3 Índices & Integridade (recomendado)	6
4. API HTTP (REST)	7
4.1 Convenções	7
4.2 Endpoints de Usuários (/users)	7
4.3 Endpoints de Consultas (/consultas)	8
4.4 Códigos de Erro (mapeamento)	9
5. API GRAPHQL	9
5.1 Schema (SDL)	9
5.2 Operações & Exemplos	9
5.3 Autorização GraphQL	10
6. SEGURANÇA	10
6.1 Autenticação	10
6.2 Autorização (RBAC)	10
6.3 Proteções e Boas Práticas	10
7. EXECUÇÃO & DEPLOY	10

7.1 Pré-requisitos	10
7. 2 Variáveis de Ambiente (exemplos)	11
7.3 Dockerfile (multi-stage)	11
7.4 docker-compose (dev)	11
7.5 Execução	12
8. QUALIDADE E TESTES	12
9. RESUMO TECNOLOGIAS UTILIZADAS	13

1. Visão Geral

1.1 Objetivo Geral

Centralizar autenticação de usuários e o ciclo de vida de consultas médicas (criação, edição e leitura), com histórico acessível por GraphQL e notificações assíncronas via mensageria.

1.1.1 Escopo

Backend único em Spring Boot 3 (Java 17) que expõe **REST** e **GraphQL**, persiste em **MySQL 8**, autentica com **Basic Auth + BCrypt**, e integra **RabbitMQ** para notificações de agendamentos. Projeto empacotado com **Docker** (multi-stage) e orquestrável via **docker-compose**. Build com **Maven Wrapper**.

1.1.2 Descrição do Projeto

Este projeto tem como objetivo o desenvolvimento de um backend modular, escalável e seguro para um sistema de gerenciamento de consultas hospitalares. A solução foi concebida a partir de uma arquitetura modular monolítica, com componentes bem desacoplados e comunicação assíncrona via RabbitMQ, explorando comunicação assíncrona para o envio de notificações e a utilização de uma API GraphQL para manipulação de dados, de modo a garantir maior flexibilidade e eficiência na integração entre componentes.

O sistema contempla diferentes perfis de usuários, classificados como **DOCTOR**, **NURSE** e **PATIENT**, cada um com permissões específicas definidas por nível de acesso distinto, cada qual com permissões específicas definidas por níveis de acesso distintos. Esses mecanismos de controle são assegurados por um modelo robusto de autenticação e autorização, implementado com Spring Security utilizando autenticação HTTP Basic (usuário e senha cadastrados no banco de dados). As credenciais são verificadas pelo CustomUserDetailsService, e as senhas são armazenadas de forma segura com o algoritmo BCrypt, o que proporciona confiabilidade, proteção e governança sobre as operações realizadas na aplicação.

1.2 Requisitos Funcionais (alto nível)

- CRUD de **Usuários** (cadastro aberto; demais operações restritas por papel).
- **Agendamentos de consultas** com validação de conflitos (paciente/doutor x data/hora).
- **Listagem de consultas** por paciente, incluindo **futuras**.
- **GraphQL** para consultas de histórico e mutações equivalentes às operações REST.
- **Notificações** em fila quando uma consulta é criada/alterada.

2. Arquitetura

2.1 Diagrama de Contexto (C4-L1)

flowchart LR

```
user[Usuário/Cliente] -->|HTTP (REST/GraphQL)| api[(Hospital API - Spring Boot)]
api --> db[(MySQL 8)]
api --> mq{{RabbitMQ}}
```

2.2 Contêineres/Componentes (C4-L2/L3)

- Camada Web:
 - REST: UserController, ConsultationController.
 - GraphQL: ConsultationHistoryGraphqlController (Queries/Mutations para Consultation).
- **Serviços:** UserService, ConsultationService, NotificationService.
- **Persistência:** UserRepository, ConsultationRepository (Spring Data JPA).
- **Segurança:** SecurityConfig, CustomUserDetailsService, UserPrincipal.
- **Mensageria:** RabbitMQConfig (AmqpTemplate, conversor JSON), NotificationService com @RabbitListener.

2.3 Decisões de Arquitetura (ADRs)

- **ADR-001 — REST + GraphQL:** REST para usuários/consultas; GraphQL para consultas/histórico (flexibilidade de leitura e agregação).
- **ADR-002 — Auth (Basic + BCrypt):** Simples para ambiente acadêmico; documentada migração para JWT (backlog) se necessário.
- **ADR-003 — Persistência (MySQL 8):** compatibilidade, SQL conhecido e suporte a índices compostos.
- **ADR-004 — Mensageria (RabbitMQ):** acoplamento fraco entre agendamento e notificação; troca do tipo *direct*.
- **ADR-005 — Build/Deploy:** Maven Wrapper; Docker multi-stage; compose para dev.

3. Domínio e Modelo de Dados

3.1 Entidades

User (users)

- **id** (*PK, bigint, auto-inc*)
- **name** (*varchar*)
- **email** (*varchar*)
- **login** (*varchar UNIQUE*)
- **password** (*varchar hash BCrypt*)
- **roles** (*set de Role via @ElementCollection*)
- **createdAt, updatedAt** (*timestamps*)

Consultation (consultations)

- **id** (*PK, bigint, auto-inc*)
- **patientId** (*bigint*)
- **doctorId** (*bigint*)
- **date** (*datetime*)
- **observations** (*text*)

3.2 Regras de Negócio (principais)

- **Conflitos de agenda:** negar criação/edição se já houver consulta para **mesmo paciente e mesmo doutor** na mesma date.
- **Listagem do paciente:** roles NURSE/DOCTOR podem listar qualquer paciente; PATIENT só lista seu próprio patientId (checagem no `@PreAuthorize`).
- **Futuras:** *cutoff* em `LocalDateTime.now()`.

3.3 Índices & Integridade (recomendado)

- UNIQUE (login) em users (já aplicado pelo mapeamento JPA).
- UNIQUE (doctor_id, date) e UNIQUE (patient_id, date) em consultations para reforçar regras de negócio no banco.

Nota: IDs de paciente/médico são representados como Long simples; integração com cadastro de pessoas/médicos é tratada fora do escopo deste serviço.

4. API HTTP (REST)

4.1 Convenções

- Base URL: http://<host>:8080.
- Content-Type: application/json (UTF-8).
- Autenticação: **Basic Auth** (exceto cadastro de usuário). Senhas sempre **BCrypt** no backend.
- Erros no formato ErrorResponse { message: string, status: number }.

4.2 Endpoints de Usuários (/users)

Método	Caminho	Auth	Descrição	Sucessos	Erros
POST	/users	Público	Cadastra Usuário	201 Created + UserResponse	409 (duplicidade), 400 (validação)
GET	/users	DOCTOR/NURSE	Lista Usuários	200 OK + [UserResponse]	401, 403
PUT	/users/{id}	DOCTOR/NURSE	Atualiza usuário	200 OK + UserResponse	404 (id), 409 (duplicidade), 400
DELETE	/users/{id}	DOCTOR/NURSE	Remove usuário	204 No Content	404

UserRequest (exemplo)

```
{  
  "name": "Maria Souza",  
  "email": "maria@exemplo.com",  
  "login": "maria",  
  "password": "<senha_clara_para_envio>",  
  "roles": ["PATIENT"]  
}
```

4.3 Endpoints de Consultas (/consultas)

Método	Caminho	Auth	Descrição	Sucessos	Erros
POST	/consultas	NURSE/DOCTOR	Cria consulta	201 Created + ConsultationResponse	404, (paciente/doutor), 409 (conflito)
PUT	/consultas/{id}	DOCTOR	Edita consulta	200 OK + ConsultationResponse	404, 409
GET	/consultas/paciente/{id}	NURSE/DOCTOR ou PATIENT	Lista consultas do paciente	200 OK + [ConsultationResponse]	401, 403
GET	/consultas/paciente/{id}/futuras	NURSE/DOCTOR ou PATIENT	Lista futuras do paciente	200 OK + [ConsultationResponse]	401, 403

ConsultationRequest (exemplo)

```
{  
    "patientId": 4,  
    "doctorId": 1,  
    "date": "2025-10-24T10:00:00",  
    "observations": "Consulta de rotina"  
}
```

ConsultationResponse (exemplo)

```
{  
    "id": 11,  
    "patientId": 4,  
    "doctorId": 1,  
    "date": "2025-10-24T10:00:00",  
    "observations": "Consulta de rotina"  
}
```

4.4 Códigos de Erro (mapeamento)

- 409 Conflict → UserAlreadyExistsException, ValidateConsultationException (conflitos de agenda)
- 404 Not Found → ResourceNotFoundException
- 401 Unauthorized → ausência/falha de credenciais Basic
- 403 Forbidden → AuthorizationDeniedException (sem escopo/role)

5. API GraphQL

- **Endpoint:** /graphql
- **Console:** /graphiql (habilitado em dev)

Nota: Para autenticação, é utilizado o **Basic Auth**, utilizando o **username** e **password** criados do banco de dados.

5.1 Schema (SDL)

TYPEDEF → O *SDL acima* é espelhado do *schema.graphqls* do projeto.

5.2 Operações & Exemplos

Query — consultas por paciente

```
query($patientId: ID!){  
  consultationsByPatient(patientId: $patientId){ id date patientId doctorId observations }  
}
```

Query — futuras por paciente

```
query($patientId: ID!){  
  futureConsultationsByPatient(patientId: $patientId){ id date }  
}
```

Mutation — criar consulta

```
mutation($input: ConsultationInput!){  
  createConsultation(input: $input){ id date patientId doctorId observations }  
}
```

Mutation — editar consulta

```
mutation($id: ID!, $input: ConsultationInput!){  
  updateConsultation(id: $id, input: $input){ id date observations }  
}
```

5.3 Autorização GraphQL

- Queries por paciente: NURSE/DOCTOR ou PATIENT se patientId == principal.id.
- Mutations: criar = NURSE/DOCTOR; editar = DOCTOR.

6. Segurança

6.1 Autenticação

- **HTTP Basic** em SecurityFilterChain com httpBasic().
- CustomUserDetailsService carrega usuário por login (campo único) e monta UserPrincipal.
- Senhas armazenadas com BCryptPasswordEncoder.

6.2 Autorização (RBAC)

Papeis: DOCTOR, NURSE, PATIENT

Matriz (principais recursos):

Recurso/Ação	DOCTOR	NURSE	PATIENT
Criar consulta	✓	✓	✗
Editar consulta	✓	✗	✗
Listar consultas de um paciente	✓	✓	✓
Listar futuras de um paciente	✓	✓	✓
Criar usuário	✓	✓	✓
Listar usuários	✓	✓	✗
Atualizar/Remover usuário	✓	✓	✗

6.3 Proteções e Boas Práticas

- **CSRF:** desabilitado para APIs stateless.
- **CORS:** configurar conforme domínios do cliente.
- **Rate limiting** e **account lockout** (backlog) para mitigar brute-force.
- **Segredos por ambiente** (nunca commitar senhas reais; usar variáveis de ambiente/Secrets Manager).

7. Execução & Deploy

7.1 Pré-requisitos

Git, JDK 17, Maven 3.8+, Docker e Docker Compose.

7. 2 Variáveis de Ambiente (exemplos)

```
SPRING_RABBITMQ_HOST=rabbitmq  
SPRING_DATASOURCE_URL=  
jdbc:mysql://mysql:3306/hospital?useSSL=false&serverTimezone=UTC  
SPRING_DATASOURCE_USERNAME=user-hospital  
SPRING_DATASOURCE_PASSWORD=<secret>  
SPRING_JPA_HIBERNATE_DDL_AUTO=update  
SERVER_ADDRESS=0.0.0.0  
SERVER_PORT=8080
```

7.3 Dockerfile (multi-stage)

- **Stage build:** maven:3.8.5-openjdk-17 → mvn -DskipTests package.
- **Stage runtime:** openjdk:17-jdk-slim; copia app.jar; expõe 8080.

7.4 docker-compose (dev)

Serviços mínimos:

- **mysql** porta 3306, usuário user-hospital, db hospital.
- **rabbitmq** imagem oficial com *management* em 15672 — recomendado.
- **app** build do Dockerfile; depender de mysql/rabbitmq; variáveis conforme §10.2.

O arquivo compose.yaml do projeto já declara o **mysql** e as variáveis da aplicação; recomenda-se adicionar explicitamente o serviço **rabbitmq** caso não esteja presente no ambiente.

7.5 Execução

```
# build local com wrapper  
./mvnw clean package -DskipTests  
  
# docker build  
docker build -t hospital-app:local .  
  
# docker compose (subindo mysql + rabbitmq + app)  
docker compose up -d  
  
# logs da aplicação  
docker logs -f <container_app>
```

Nota: O serviço principal estará disponível em: <http://localhost:8080>

8. Qualidade e Testes

Smoke de contexto: @SpringBootTest valida *ApplicationContext*.

Unitários (JUnit5/Mockito): UserService, ConsultationService (hash, duplicidade, conflitos de agenda).

Integração (Testcontainers):

- MySQL: repositórios UserRepository, ConsultationRepository (queries e constraints UNIQUE).
- RabbitMQ: publicar/consumir mensagem (@RabbitListener).

Contrato:

- REST com MockMvc (caminhos, status, erros mapeados por GlobalExceptionHandler).
- GraphQL com GraphQITester (queries/mutations e regras de autorização).

Cobertura: JaCoCo (alvos por módulo). **Mutação:** PITest (serviços principais).

9. Resumo Tecnologias Utilizadas

- **Linguagem:** Java 17
- **Framework:** Spring Boot 3.2.7
- **Segurança:** Spring Security, autenticação Basic (*usuário/senha + BCrypt*)
- **API:** Spring for GraphQL
- **Banco de Dados:** MySQL
- **Mensageria:** RabbitMQ (*Spring AMQP*)
- **Containerização:** Docker e Docker Compose
- **Build Tool:** Maven