**Tech Challenge** - **FASE 03**

**Sistema de Gerenciamento Hospitalar**

**Equipe:**

Rafael Detilio - RM362570

Bruna Cristina - RM358934

Vinícius Guimarães – RM361304

Leticia Lucena - RM364673

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

**Data:** 13 de setembro de 2025

**Índice**

1. [Visão Geral do Projeto](file:///C:\Users\rafael.detilio\Desktop\tc-fase-3\doc.html#s1)
2. [Arquitetura da Solução](file:///C:\Users\rafael.detilio\Desktop\tc-fase-3\doc.html#s2)
3. [Tecnologias Utilizadas](file:///C:\Users\rafael.detilio\Desktop\tc-fase-3\doc.html#s3)
4. [Como Executar o Projeto](file:///C:\Users\rafael.detilio\Desktop\tc-fase-3\doc.html#s4)
5. [Documentação da API (GraphQL)](file:///C:\Users\rafael.detilio\Desktop\tc-fase-3\doc.html#s5)
6. [Segurança: Autenticação e Autorização](file:///C:\Users\rafael.detilio\Desktop\tc-fase-3\doc.html#s6)
7. [Comunicação Assíncrona com RabbitMQ](file:///C:\Users\rafael.detilio\Desktop\tc-fase-3\doc.html#s7)
8. [Como Testar](file:///C:\Users\rafael.detilio\Desktop\tc-fase-3\doc.html#s8)
9. **Visão Geral do Projeto**

Este projeto consiste no desenvolvimento de um backend modular e seguro para um sistema de gerenciamento de consultas hospitalares. A solução foi projetada utilizando uma arquitetura de microsserviços, com comunicação assíncrona para o envio de notificações e uma API GraphQL para a manipulação de dados, garantindo flexibilidade e escalabilidade.

O sistema atende a diferentes perfis de usuários (identificados como ADMIN e USER), cada um com níveis de acesso distintos, controlados por um sistema de autenticação e autorização robusto implementado com Spring Security e tokens JWT.

1. **Arquitetura da Solução**

A solução foi desenvolvida seguindo uma arquitetura de microsserviços para garantir a separação de responsabilidades.

**Serviços Principais**

* agendamento-service: Serviço principal responsável pelo CRUD de Consultas e Usuários, exposição da API GraphQL e publicação de mensagens no RabbitMQ.
* notification-service: Serviço secundário responsável por consumir mensagens da fila do RabbitMQ e simular o envio de lembretes.

**Infraestrutura**

* MySQL: Banco de dados relacional para persistência dos dados.
* RabbitMQ: Broker de mensageria para a comunicação assíncrona.

1. **Tecnologias Utilizadas**

* Linguagem: Java 17
* Framework: Spring Boot 3.2.7
* Segurança: Spring Security, JWT (Java Web Token)
* API: Spring for GraphQL
* Banco de Dados: MySQL
* Mensageria: RabbitMQ (Spring AMQP)
* Containerização: Docker e Docker Compose
* Build Tool: Maven

1. **Como Executar o Projeto**
   1. Pré-requisitos

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

* 1. Executando com Docker Compose

A maneira mais simples de executar todo o ambiente é com o Docker Compose. Na raiz do projeto, execute o seguinte comando no seu terminal:

docker-compose up --build

O serviço principal estará disponível em:

<http://localhost:8080>

1. **Documentação da API (GraphQL)**
   1. Endpoint: http://localhost:8080/graphql

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

* 1. Exemplos de Queries (Consultas)

Buscar a consultas por id de paciente:

query {

{

consultationsByPatient(patientId: 2) {

id

observations

date

patientId

doctorId

}

}

}

Buscar a consultas futuras por id de paciente:

{

  futureConsultationsByPatient(patientId: 1) {

    id

    observations

    date

    patientId

    doctorId

  }

}

* 1. Exemplos de Mutations (Modificações)

Criar uma nova consulta (requer autenticação):

Query:

mutation CreateNewConsultation($input: ConsultationInput!) {

  createConsultation(input: $input) {

    date

    observations

    patientId

    doctorId

  }

}

Variables:

{

  "input": {

    "patientId": "2",

    "doctorId": "1",

    "date": "2025-10-24T10:00:00",

    "observations": "Paciente relata dor de cabeça."

  }

}

Alterar uma consulta (Requer autenticação):

Query:

mutation UpdateExistingConsultation($id: ID!, $input: ConsultationInput!) {

  updateConsultation(id: $id, input: $input) {

    id

    date

    observations

    patientId

    doctorId

  }

}

Variables:

{

  "id": 2,

  "input": {

    "patientId": "2",

    "doctorId": "1",

    "date": "2025-10-24T10:00:00",

    "observations": "Paciente relata dor de cabeça."

  }

}

1. **Segurança: Autenticação e Autorização**

A segurança é gerenciada pelo Spring Security com Basic Auth utilizando o username e pasword cadastrados no banco de dados.

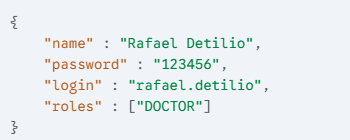
* **Endpoint de criação de usuário**

**Descrição:** Cria um novo usuário no sistema. Este endpoint é público e não requer autenticação.

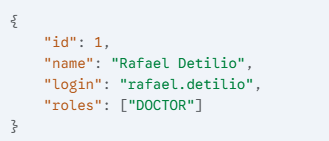
**Método:** POST

**URL:** http://localhost:8080/users

**Corpo da Requisição (Body):**



Resposta de Sucesso (201 Created):



Resposta de Erro (409 Conflict): Retornado caso o login já esteja em uso.

* **Endpoint de Listagem de Usuários**

**Descrição:** Lista os usuários cadastrados. Pode ser filtrado por role.

**Método:** GET

**URL:** http://localhost:8080/users

**Parâmetros de Query (Query Params):** role (Opcional): Filtra usuários pela role especificada.

Exemplo: <http://localhost:8080/users?role=PATIENT>

**Resposta de Sucesso (200 OK):**

****

**Resposta de Erro (403 Forbidden):** Retornado caso o usuário autenticado não tenha a permissão necessária.

* **Endpoint de Atualização de Usuário**

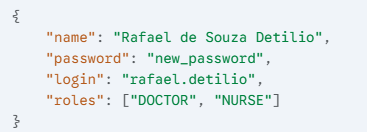
**Descrição:** Atualiza os dados de um usuário existente.

**Método:** PUT

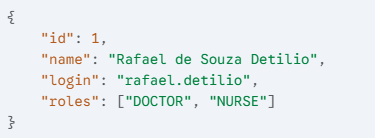
**URL:** [http://localhost:8080/users/{id}](http://localhost:8080/users/%7bid%7d)

**Parâmetros de URL (Path Variables):** id (Obrigatório): O ID do usuário a ser atualizado.

**Corpo da Requisição (Body):**



**Resposta de sucesso (200 OK):**



**Respostas de Erro:**

403 Forbidden: Usuário sem permissão.

404 Not Found: Usuário com o id especificado não encontrado.

409 Conflict: O novo login já está em uso por outro usuário.

* **Endpoint de Exclusão de Usuário**

**Descrição:** Exclui um usuário do sistema. A operação falhará se o usuário possuir consultas vinculadas a ele (seja como médico ou paciente).

**Autorização:** Requer autenticação Basic Auth com perfil de DOCTOR ou NURSE.

**Método:** DELETE

**URL:** [http://localhost:8080/users/{id}](http://localhost:8080/users/%7bid%7d)

**Parâmetros de URL (Path Variables):** id (Obrigatório): O ID do usuário a ser excluído.

**Resposta de Sucesso (204 No Content):** Nenhum corpo na resposta, apenas o status HTTP indica sucesso.

**Respostas de Erro:**

403 Forbidden: Usuário sem permissão.

404 Not Found: Usuário com o id especificado não encontrado.

409 Conflict (ou similar): Retornado caso o usuário possua consultas vinculadas, impedindo a exclusão.

* **Níveis de Acesso (Papéis - Roles)**

| **Papel** | **Permissões** |
| --- | --- |
| **DOCTOR** | Pode visualizar e editar o histórico de consultas de qualquer paciente. |
| **NURSE** | Pode registrar novas consultas e visualizar o histórico de qualquer paciente. |
| **PATIENT** | Pode visualizar apenas as suas próprias consultas e histórico. |

1. **Comunicação Assíncrona com RabbitMQ**

Fila: queue.consulta.criada

Quando uma consulta é criada, uma mensagem com os detalhes é enviada para a fila, visível através do console:

****

1. **Como Testar**

Recomenda-se o uso do Postman ou similar:

* Use o endpoint <http://localhost:8080/auth/register> para cadastrar um usuário.
* Utilize o usuário e senha cadastrados no Authorization como Basic Auth.
* Execute as queries de criação de consultas, visualização de consutas agendadas e alterações de consultas com GraphQL através do endpoint: <http://localhost:8080/graphql>.
* Ao criar uma consulta, verifique os logs do contêiner do notification-service para confirmar o recebimento da mensagem.