Projeto: Tech Challenge Fase 3

Equipe: Guilherme de Oliveira Vicente (rm360802) e Richard de Oliveira Lopes (rm360801)

1. Descrição do Problema

Em um ambiente hospitalar, é fundamental dispor de sistemas que assegurem o agendamento eficiente de consultas, a gestão do histórico dos pacientes e o envio de lembretes automáticos para garantir a assiduidade nas consultas. Este sistema deve ser acessível a diversos perfis de usuários (médicos, enfermeiros e pacientes), com controle de acesso rigoroso e funcionalidades adaptadas a cada perfil.Objetivo do Projeto

O objetivo é criar um backend simplificado e modular, com ênfase em segurança e comunicação assíncrona. Isso garantirá um sistema escalável e seguro, aplicando as melhores práticas em autenticação, autorização e comunicação interserviços.

2. Arquitetura do Sistema

O projeto segue uma arquitetura de microsserviços com Clean Architecture, utilizando o framework Spring Boot para facilitar o desenvolvimento de aplicações Java. O sistema está organizado em dois microsserviços independentes que se comunicam de forma assíncrona através do Apache Kafka.Camadas do Sistema

- **Domain (Domínio)**
    - **Entities:** Classes que representam as entidades de negócio como User e Consultation, contendo validações e regras de domínio.
    - **Enums:** Tipos específicos do negócio como UserType (MÉDICO, ENFERMEIRO, PACIENTE) e ConsultationStatus (AGENDADA, CONCLUÍDA, CANCELADA).
    - **Events:** Eventos de domínio como ConsultationCreatedEvent, ConsultationRescheduledEvent e ConsultationCancelledEvent para comunicação assíncrona.
    - **Repositories:** Interfaces que definem contratos para persistência de dados, sem dependência de implementações específicas.
- **Application (Aplicação)**
    - **Use Cases:** Classes como CreateConsultationUseCase, UpdateConsultationUseCase e CancelConsultationUseCase que coordenam as operações de negócio e publicam eventos quando necessário.
    - **DTOs:** Records Java para transferência de dados como ConsultationRequestDTO, UserResponseDTO e LoginRequestDTO, garantindo imutabilidade e validação de entrada.
- **Infrastructure (Infraestrutura)**
    - **Config:** Configurações do sistema como SecurityConfig para autenticação JWT, KafkaConfig para produção/consumo de mensagens e GraphQLConfig para APIs flexíveis.
    - **Persistence:** Implementações dos repositórios como ConsultationRepositoryImpl e UserRepositoryImpl que utilizam Spring Data JPA para interação com o banco H2.
    - **Security:** Componentes de segurança como JwtTokenUtil, JwtRequestFilter e UserDetailsServiceImpl para autenticação e autorização baseada em roles.

    - **Events:** EventPublisher para publicação de eventos de domínio no Kafka.
- **Presentation**
    - **Controllers:** Classes como ConsultationController, AuthController e UserController que recebem requisições HTTP REST, processam os dados de entrada e delegam operações para os casos de uso.
    - **GraphQL:** Resolvers GraphQL que fornecem uma API flexível para consultas complexas, permitindo que clientes solicitem apenas os dados necessários.

Microsserviços

- **scheduling-service (Producer):** Serviço principal responsável pelo gerenciamento de consultas e usuários.
- **notification-service (Consumer):** Serviço que consome as mensagens enviadas para envio de mensagem para o cliente.

Segurança e Autenticação

- **JWT (JSON Web Tokens)**
    - **Autenticação:** Tokens JWT com expiração configurável para sessões de usuário.
    - **Autorização:** Controle de acesso baseado em roles (MÉDICO, ENFERMEIRO, PACIENTE).
    - **Validação:** Interceptação de requisições através de filtros Spring Security.
- **Controle de Acesso**
    - **MÉDICO:** Acesso completo a todas as operações de consulta.
    - **ENFERMEIRO:** Pode criar e cancelar consultas, mas não editar.
    - **PACIENTE:** Pode apenas visualizar suas próprias consultas.

Persistência de Dados

- **Spring Data JPA**
    - **Banco:** H2 em memória para desenvolvimento.
    - **ORM:** Hibernate para mapeamento objeto-relacional.
    - **Repositories:** Implementações que estendem interfaces Spring Data para operações CRUD.
    - **Transações:** Gerenciamento automático através de anotações @Transactional.

3. Descrição dos Endpoints da API

Autenticação

- **POST** `/api/auth/login` - Fazer login no sistema.
- **POST** `/api/auth/register` - Registrar novo usuário.

Usuários

- **GET** `/api/users` - Listar todos os usuários ativos.
- **GET** `/api/users/{id}` - Buscar usuário por ID.

Consultas

- **POST** `/api/consultations` - Criar nova consulta.
- **GET** `/api/consultations/{id}` - Buscar consulta por ID.
- **GET** `/api/consultations` - Listar todas as consultas.
- **PUT** `/api/consultations/{id}` - Atualizar consulta.
- **DELETE** `/api/consultations/{id}` - Cancelar consulta.
- **GET** `/api/consultations/patient/{patientId}` - Listar consultas de um paciente.
- **GRAPHQL** `/graphql` - Consultas dinâmicas:
    a. **Usuários**
        i. users: [User!]!
        ii. user(id: ID!): User
    b. **Consultas**
        i. consultations: [Consultation!]!
        ii. consultation(id: ID!): Consultation
        iii. patientConsultations(patientId: ID!): [Consultation!]!
    c. **Histórico e consultas flexíveis do paciente**
        i. patientHistory(patientId: ID!): [Consultation!]!
        ii. patientUpcomingConsultations(patientId: ID!): [Consultation!]!
        iii. patientConsultationsByStatus(patientId: ID!, status: ConsultationStatus!): [Consultation!]!

4. Arquitetura do Sistema

O projeto implementa uma arquitetura de microsserviços seguindo os princípios da Clean Architecture com:Módulos Principais

- **shared-domain:** Domínio compartilhado contendo entidades, eventos e interfaces comuns.
- **scheduling-service:** Serviço responsável pelo agendamento de consultas (API principal).
- **notification-service:** Serviço responsável pelo envio de notificações via email.

Tecnologias Principais

- **Java 22:** Linguagem de programação com features modernas.
- **Spring Boot 3.2:** Framework principal para desenvolvimento dos microsserviços.
- **Spring Security:** Autenticação JWT e autorização baseada em roles.
- **Spring Data JPA:** Persistência com banco H2 (desenvolvimento).
- **GraphQL:** API flexível para consultas complexas.
- **Apache Kafka:** Message broker para comunicação assíncrona.
- **Maven:** Gerenciamento de dependências e build.

Padrões Arquiteturais Implementados

- **Clean Architecture (Camadas)**
- **Event-Driven Architecture**
    - **Producer:** `scheduling-service` publica eventos quando consultas são criadas/alteradas.
    - **Consumer:** `notification-service` escuta eventos e envia notificações por email.
    - **Message Broker:** Kafka gerencia a comunicação assíncrona entre

serviços.

Funcionalidades do Sistema

- **MÉDICO:** Pode criar, visualizar, editar e cancelar consultas.
- **ENFERMEIRO:** Pode criar, visualizar e cancelar consultas (não pode editar).
- **PACIENTE:** Pode apenas visualizar suas próprias consultas.

Fluxo de Agendamento

1. Usuário autenticado cria/edita/cancela consulta via API REST.
2. `scheduling-service` valida dados e persiste no banco H2.
3. Sistema publica evento no tópico Kafka `consultation-events`.
4. `notification-service` consome o evento automaticamente.
5. Email de notificação é enviado para paciente e médico.

Segurança Implementada

- **Autenticação:** JWT tokens com expiração configurável.
- **Autorização:** Controle de acesso baseado em roles usando Spring Security.
- **Validação:** Bean Validation para entrada de dados.
- **Criptografia:** Senhas criptografadas com BCrypt.

5. Testes Unitários

Cobertura de testes realizada utilizando as bibliotecas de JUnit, Jacoco e Mockito.

6. Collections para Teste

Collection Postman

7. Repositório do Código

Github projeto