Relatório Técnico - Hairstyle - Sistema de Agendamento e Gerenciamento para Serviços de Beleza e Bem-Estar

Link do GitHub

GitHub: Hairstyle - Projeto

---

1. Documentação Técnica do Sistema e da API

1.1. Introdução

O sistema **Hairstyle** foi desenvolvido com o objetivo de gerenciar agendamentos para serviços de beleza e bem-estar, utilizando **Java 17**, **Spring Boot**, **JWT** para autenticação, **PostgreSQL** como banco de dados e **Docker** para facilitar o deploy e a execução em ambientes controlados.

Tecnologias principais:

- **Java 17** com **Spring Boot** para o backend.
- **Spring Security (JWT)** para autenticação e controle de acesso.
- **Swagger** para documentação da API.
- **PostgreSQL** para o armazenamento de dados relacionais.
- **Docker** para containerização da aplicação.

1.2. Arquitetura do Sistema

O sistema foi projetado com base na **Clean Architecture**, separando claramente as camadas de domínio, aplicação, adaptação e infraestrutura.

- **Domínio**: Contém as entidades que representam os conceitos do negócio, como `Usuario`, `Profissional`, `Agendamento`, etc.
- **Aplicação**: Camada de serviços e lógica de negócios, como `NotificacaoService` e `GoogleCalendarService`.
- **Adaptadores**: Responsáveis pela integração com sistemas externos (ex.: Google Calendar).
- **Frameworks e Externos**: Inclui dependências como Spring Boot para controle de injeção de dependências e Spring Security para autenticação.

1.3. Entidades do Domínio

As principais entidades do sistema são:

- **Usuario**: Representa os usuários do sistema, com informações como nome, email e senha.
- **Cliente**: Contém informações sobre os clientes que utilizam os serviços.
- **Profissional**: Contém dados sobre os profissionais de beleza e bem-estar.
- **Serviço**: Detalha os serviços disponíveis nos estabelecimentos.
- **Agendamento**: Representa os agendamentos realizados pelos clientes.
- **Estabelecimento**: Contém informações sobre os locais onde os serviços são prestados.
- **Avaliacao**: Representa as avaliações feitas pelos clientes sobre os serviços ou profissionais.
- **HorarioDisponivel**: Representa a disponibilidade de horários dos profissionais.

1.4. Serviços Principais

- **NotificacaoService**: Lógica de envio de notificações automáticas de confirmações e lembretes de agendamentos.
- **GoogleCalendarService**: Responsável pela integração com o Google Calendar para sincronização de agendamentos.

1.5. Descrição da API

A API REST foi documentada com o Swagger. Aqui estão alguns exemplos de endpoints:

- **POST /api/auth/login**: Realiza login e retorna um token JWT.
- **POST /api/usuarios**: Cria um novo usuário.
- **POST /api/estabelecimentos**: Cria um novo estabelecimento (com token JWT).
- **POST /api/agendamentos**: Cria um novo agendamento (com token JWT).
- **GET /api/estabelecimentos/filtros**: Realiza buscas avançadas de estabelecimentos.

A autenticação é feita via **JWT** (JSON Web Tokens), garantindo que apenas usuários autenticados possam acessar rotas protegidas.

---

2. Relatório Técnico

2.1. Tecnologias e Ferramentas Utilizadas

- **Java 17 e Spring Boot**: Usados para o desenvolvimento do backend com desempenho aprimorado e estrutura modular.
- **Spring Security (JWT)**: Implementação de segurança e autenticação utilizando tokens JWT para proteger endpoints.
- **Swagger**: Utilizado para gerar a documentação interativa da API.
- **Google Calendar API**: Para integração e sincronização de agendamentos com calendários externos.
- **PostgreSQL**: Banco de dados relacional usado para persistência de dados.

- **Docker**: Usado para containerizar a aplicação, facilitando o deploy em diferentes ambientes.

2.2. Desafios Técnicos e Soluções Adotadas

- **Sincronização com Google Calendar**: A integração com o Google Calendar envolveu o uso de credenciais OAuth e a criação de eventos no Google Calendar para refletir os agendamentos.
- **Segurança com JWT**: A implementação do JWT garante que apenas usuários com token válido possam acessar rotas protegidas, evitando o acesso não autorizado.
- **Validação de Horários**: A validação dos horários disponíveis dos profissionais foi implementada para evitar conflitos entre agendamentos.

2.3. Aplicação dos Princípios de Clean Architecture

A arquitetura limpa foi aplicada em várias camadas:

- **Entidades**: Definidas na camada de domínio, com foco em abstrair as regras de negócio.
- **Casos de Uso**: Serviços como `NotificacaoService` e `GoogleCalendarService` encapsulam a lógica de negócios específica.
- **Adaptadores de Entrada**: Controladores que fazem a ponte entre a interface externa (API) e a lógica interna do sistema.
- **Frameworks e Externos**: Usados para facilitar o desenvolvimento, testes e manutenção da aplicação.

2.4. Qualidade de Software

- **Testes Unitários e TDD**: O sistema foi desenvolvido com base em **Test-Driven Development (TDD)**, garantindo a cobertura de testes e a detecção precoce de falhas.
- **Testes de Integração e CI**: O uso de **Continuous Integration (CI)** permite a execução contínua de testes integrados, assegurando que o sistema funcione de maneira coesa.
- **Testes Não Funcionais**: Foram realizados testes de carga e performance para garantir que o sistema suporte um alto volume de agendamentos simultâneos.

---

3. Instruções para Rodar o Projeto

3.1. Pré-Requisitos

- **Java 17 ou superior**
- **Docker** instalado

3.2. Passos para Execução

1. Clone o repositório:

   ```
   bash
   Copiar código
   git clone https://github.com/MatheusFDS/hairstyle.git
   ```

2. Navegue até o diretório do projeto:

   ```
   bash
   Copiar código
   cd hairstyle
   ```

3. Construa as imagens do Docker:

   ```
   bash
   Copiar código
   docker-compose build
   ```

4. Inicie o ambiente com o Docker:

   ```
   bash
   Copiar código
   docker-compose up
   ```

5. Acesse a documentação da API no Swagger:

   ```
   bash
   Copiar código
   http://localhost:8080/swagger-ui/index.html#
   ```

3.3. Rodando os Testes

Para rodar os testes, execute o seguinte comando:

```
bash
Copiar código
mvn test
```

Lembre-se de que a aplicação principal deve estar em execução para que os testes de integração funcionem corretamente.