CENTRO UNIVERSITÁRIO FAMETRO
CURSO DE GRADUAÇÃO TECNOLÓGICA EM
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
PROJETO INTEGRADOR 4 - ENGENHARIA DE SOFTWARE

Jefferson Sant'ana Galvão
Victória Maria de Brito Costa

AV1 - Sistema CRUD para Gerenciamento de Consultas - SysAdm

2024.1

SUMÁRIO

1 INTRODUÇÃO

Este documento oferece uma visão aprofundada da implementação, configuração e desdobramento do sistema de agendamento de consultas, nomeado como SysAdm, enfatizando a arquitetura, tecnologias, e justificativas para escolhas de design.

2 VISÃO GERAL DO SISTEMA

O sistema oferecerá uma solução completa para o cadastro de usuários, autenticação, e gerenciamento de consultas. Para a primeira entrega, solicitou-se um sistema de administração de usuários, contendo autenticação, criação, recuperação e redefinição de senha e um sistema de CRUD (Criação, Leitura, Atualização e Remoção de Dados).

Optou-se por implementar o projeto através de conteinerização dos módulos utilizando Docker. A escolha pela conteinerização se deve a várias vantagens significativas:

- Isolamento de Ambiente: Cada contêiner opera independentemente, evitando conflitos entre dependências.
- Desdobramento Simplificado: Facilita a implantação em diferentes ambientes, mantendo consistência entre desenvolvimento, teste e produção.
- Eficiência de Recursos: Contêineres utilizam recursos de maneira otimizada, aumentando a eficiência do sistema.

O sistema de conteinerização é construído a partir dos diretórios locais ./site e ./api, e as características dos containers são definidas dentro do docker-compose.yml.

2.1 Estrutura do Docker Compose

Serviços: Define três serviços principais - postgres, api e site, cada um correspondendo a um componente do sistema.

Rede Docker Personalizada: Facilita a comunicação interna entre os contêineres, permitindo que se comuniquem usando nomes de serviço em vez de endereços IP.

3 ARQUITETURA DO SISTEMA

O sistema é dividido em duas partes principais: uma API back-end responsável pela lógica de negócio e persistência de dados; e um site front-end que serve como interface de usuário. A comunicação entre o front-end e o back-end é realizada por meio de chamadas API REST.

3.1 Detalhes de Implementação da API

3.1.1 Funcionalidades Principais

Gerenciamento de Usuários: Inclui criação, leitura, atualização e exclusão de usuários.

Autenticação e Autorização: Métodos para login e verificação de identidade dos usuários.

3.1.2 Estrutura e Tecnologias

A API foi desenvolvida utilizando Java com Spring Boot, organizada seguindo o padrão Maven. A implementação se deu a partir de um modelo de usuários (Usuario), controladores REST (como UsuarioController), e repositórios para interação com o banco de dados (UsuarioRepository).

3.1.2.1 Ficha técnica

Java: Linguagem de programação.
Spring Boot: Framework para desenvolvimento de aplicações Java com ênfase em microserviços.
Maven: Ferramenta de gerenciamento e compreensão de projeto.

Pacotes(Packages): A API está contida no pacote principal br.com.sysadm, seguindo uma arquitetura de pacotes organizada, separando claramente os componentes do sistema:

- Model: Define as entidades do sistema, como Usuario, representando os dados que serão persistidos no banco de dados.
- Repository: Camada de abstração que facilita a comunicação com o banco de dados, utilizando Spring Data JPA para simplificar as operações CRUD.
- Controller: Contém os controladores que expõem endpoints da API, recebendo requisições HTTP e retornando respostas. Utiliza anotações do Spring MVC para roteamento e manipulação de dados.

Persistência de Dados:

- Spring Data JPA: Utilizado para interagir com o banco de dados de forma abstrata, permitindo focar na lógica de negócio em vez de detalhes de implementação da persistência.
- Hibernate: Como implementação de JPA, o Hibernate é usado indiretamente pelo Spring Data JPA para mapeamento objeto-relacional (ORM), convertendo as entidades Java em registros de banco de dados e vice-versa.

Estrutura do código:

- ApiSysadmApplication.java: Ponto de entrada da aplicação Spring Boot, configurando a inicialização.
- CorsConfig.java: Configurações de CORS para permitir requisições cross-origin.

- Model/Repository/Controller: Organização MVC tradicional, facilitando a separação de responsabilidades e manutenção do código.
- Métodos Disponíveis no UsuarioController:

  listarTodos(): Retorna uma lista de todos os usuários cadastrados na base de dados. Utiliza o método GET.

  cadastrarUsuario(Usuario usuario): Recebe um objeto Usuario via POST, verifica se o usuário já existe (pelo CPF), e, se não, salva o novo usuário na base de dados. Retorna um status CREATED se o usuário for cadastrado com sucesso, ou CONFLICT se já existir um usuário com o mesmo CPF.

  atualizarUsuario(String cpf, Usuario usuarioAtualizado): Atualiza os dados de um usuário existente, identificado pelo CPF. O método utiliza PUT e retorna o usuário atualizado se encontrado; caso contrário, retorna um status NOT FOUND.

  deletarUsuario(String cpf): Remove um usuário da base de dados, identificado pelo CPF, usando o método DELETE. Retorna um status NO CONTENT se a operação for bem-sucedida ou NOT FOUND se o usuário especificado não existir.
- Anotações e Funcionalidades do Spring Boot:

  @RestController: Indica que a classe é um controlador REST, e seus métodos retornam dados diretamente como resposta HTTP.

  @CrossOrigin: Permite requisições cross-origin para este controlador, essencial para aplicações web que fazem requisições de domínios diferentes.

  @RequestMapping("/usuario"): Define o caminho base para todos os métodos deste controlador. Assim, todas as operações relacionadas a usuários serão acessadas por esse caminho.

  @Autowired: Injeção de dependência do Spring para instanciar automaticamente o UsuarioRepository, permitindo operações no banco de dados relacionadas a usuários.

3.1.2.2 Estrutura de Diretórios

src/main/java: Código fonte da aplicação, incluindo modelos, controladores e repositórios.
src/test/java: Testes unitários e de integração gerados automaticamente pelo Spring Initializr.
src/main/resources: Definição de propriedades do Spring Boot.

3.1.3 Construção do Dockerfile

O Dockerfile da API utiliza uma abordagem de construção em duas etapas:

- Etapa de Compilação: Usa maven:3.8.4-openjdk-17 para compilar o código Java, aproveitando o Maven para gerenciamento de dependências.
- Etapa Final: Baseia-se em openjdk:17 para criar uma imagem leve contendo apenas o artefato JAR necessário para executar a aplicação.

3.2 Detalhes de Implementação do Site

O componente front-end do sistema SysAdm é desenvolvido com Flask, um microframework Python, e serve como a interface de usuário para interação com a API back-end. Este módulo é responsável por apresentar uma interface web para cadastro de usuários, autenticação e gerenciamento de consultas.

3.2.1 Funcionalidades Principais

Interface de Usuário: Facilita o cadastro de novos usuários, login, visualização e, futuramente, o gerenciamento das consultas agendadas.

Integração com a API: Comunica-se com a API back-end para todas operações relacionadas a usuários e consultas.

3.2.2 Estrutura e Tecnologias

O site foi desenvolvido utilizando Flask, juntamente com HTML, CSS e JavaScript para a construção do front-end.

3.2.2.1 Ficha Técnica

Python: Linguagem de programação utilizada para o desenvolvimento do back-end do serviço web.
Flask: Microframework Python que facilita a criação de aplicações web.
HTML/CSS/JavaScript: Tecnologias padrões da web utilizadas para o desenvolvimento do front-end.
Pacotes (Packages): O site está contido no diretório principal ./site, seguindo uma estrutura organizada que separa claramente os componentes do sistema:

- Templates: Armazena os arquivos HTML para renderização das páginas.
- Static: Contém arquivos estáticos como CSS para estilização e JavaScript para funcionalidades dinâmicas.

Persistência de Dados: Embora o site em si não armazene dados diretamente, ele interage com a API back-end para todas as operações de dados, que, por sua vez, utiliza Spring Data JPA e Hibernate para a persistência.

3.2.2.2 Estrutura de Diretórios

/app.py: Ponto de entrada da aplicação Flask, definindo rotas e lógica do negócio.
/templates: Armazena os arquivos HTML usados na renderização das interfaces de usuário.
/static: Contém recursos estáticos como imagens, arquivos CSS e JavaScript.

3.2.3 Construção do Dockerfile

O Dockerfile do site é construído com base em python:3.11-slim-bookworm, proporcionando um ambiente de execução leve e eficiente. Esse Dockerfile

especifica as etapas necessárias para a construção da imagem Docker do site, incluindo a instalação de dependências e a execução da aplicação Flask.

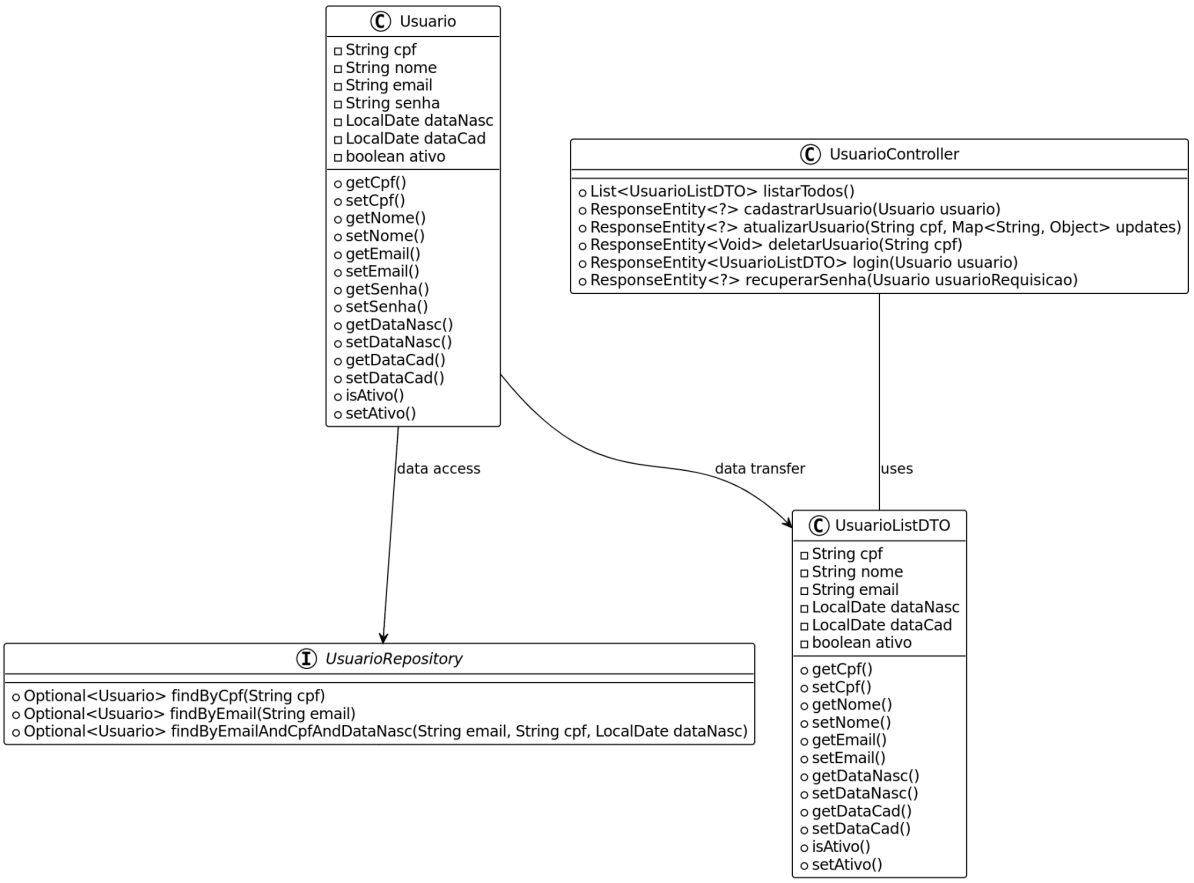
- Etapas de Construção:

Base: A imagem python:3.11-slim-bookworm é utilizada como ponto de partida para garantir um ambiente Python otimizado e leve.

Dependências: Utiliza o pip para instalar as dependências especificadas no arquivo requirements.txt.

- Execução da Aplicação: Define o comando flask run para iniciar o servidor Flask, configurado para aceitar conexões de qualquer endereço IP.

3.2.3 Diagrama de Classes



4 INSTRUÇÕES DE UTILIZAÇÃO

4.1 Construir os contêineres

Primeiramente, deve-se clonar o repositório através do terminal (Git requerido):

```
git clone https://github.com/hudjinn/SysAdm.git
```

Ou baixar e descompactar o zip do repositório disponível no link:
https://github.com/hudjinn/SysAdm

Para iniciar os contêineres, é necessário ter o Docker Desktop instalado, com o engine rodando. Entre na pasta SysAdm e execute o comando:

```
docker compose up -d --build
```

Este comando construirá e levantará os contêineres em segundo plano, conforme definido no arquivo `docker-compose.yml`.

4.2 Abrir plataforma

Após a construção e inicialização dos contêineres, a plataforma SysAdm estará acessível através dos seguintes endereços no navegador:

- Front-end (Site): http://localhost:5000
  Esta URL levará à interface de usuário do sistema, onde é possível realizar o cadastro de usuários, login, e interface de CRUD.
- API (Back-end): http://localhost:19000
  Embora a API geralmente seja acessada indiretamente pelo front-end, essa URL pode ser usada para acessar diretamente a API, por exemplo, através de ferramentas como Postman para desenvolvimento e teste de endpoints.

Certifique-se de que todas as dependências necessárias estejam corretamente instaladas e que não haja conflitos de porta com outros serviços que possam estar rodando em seu sistema. Em caso de problemas para acessar a plataforma, verifique os logs do Docker Compose para identificar possíveis erros durante o processo de construção ou inicialização dos contêineres. Se preferir visualizar os logs no terminal, levante os contêineres sem o argumento -d (daemon) através do comando `docker compose up --build`.