Análise da Estrutura Básica do Projeto Backend

Visão Geral

O projeto analisado é um sistema de gestão de pedidos de serviços municipais chamado "Simple", desenvolvido em Java utilizando o framework Spring Boot. O sistema permite o gerenciamento de pedidos de serviços municipais, com funcionalidades para cidadãos, atendentes, gestores e técnicos.

Tecnologias Utilizadas

- **Linguagem de Programação**: Java 17
- **Framework**: Spring Boot 3.2.0
- **Gerenciador de Dependências**: Maven
- **Banco de Dados**: PostgreSQL
- **Segurança**: Spring Security com JWT (JSON Web Token)
- **Documentação da API**: SpringDoc OpenAPI (Swagger)
- **Persistência**: Spring Data JPA / Hibernate
- **Containerização**: Docker

Estrutura de Pacotes

O projeto segue uma arquitetura em camadas bem definida:

Pacotes Principais

1. **config**: Contém classes de configuração do Spring Boot
   - ApplicationConfig.java
   - CorsConfig.java
   - OpenApiConfig.java
   - SecurityConfig.java
2. **controller**: Controladores REST que definem os endpoints da API
   - AuthController.java
   - CidadaoController.java
   - ConfiguracaoController.java
   - FavoritoController.java
   - PedidoController.java
   - TipoServicoController.java
   - UserController.java
3. **dto**: Objetos de Transferência de Dados (Data Transfer Objects)
   - Requests: Classes para receber dados de entrada
   - Responses: Classes para retornar dados de saída
4. **entity**: Entidades JPA que representam as tabelas do banco de dados
   - CategoriaServico.java
   - Cidadao.java
   - EtapaProcesso.java
   - Favorito.java

- Pedido.java
- Perfil.java
- StatusPedido.java
- TipoServico.java
- Usuario.java

5. **repository**: Interfaces de repositório para acesso ao banco de dados
   - CategoriaServicoRepository.java
   - CidadaoRepository.java
   - FavoritoRepository.java
   - PedidoRepository.java
   - PerfilRepository.java
   - StatusPedidoRepository.java
   - TipoServicoRepository.java
   - UsuarioRepository.java
6. **service**: Camada de serviço que implementa a lógica de negócio
   - AuthService.java
   - CidadaoService.java
   - ConfiguracaoService.java
   - FavoritoService.java
   - PedidoService.java
   - TipoServicoService.java
   - UsuarioService.java
7. **security**: Classes relacionadas à segurança da aplicação
   - JwtAuthenticationFilter.java
   - JwtTokenProvider.java
8. **exception**: Classes para tratamento de exceções
   - GlobalExceptionHandler.java
   - ResourceNotFoundException.java
9. **util**: Classes utilitárias
   - CodigoAcompanhamentoUtil.java
   - StatusPedidoEnum.java
   - TipoProcessoEnum.java

Arquivos de Configuração

application.yml

- Configuração do banco de dados PostgreSQL
- Configuração do Hibernate/JPA
- Configuração de segurança JWT
- Configuração de logging
- Porta do servidor: 8080
- Context path da API: /api

pom.xml

Principais dependências: - spring-boot-starter-data-jpa - spring-boot-starter-security - spring-boot-starter-validation - spring-boot-starter-web - io.jsonwebtoken (JWT) - springdoc-openapi (Swagger) - postgresql - lombok - hypersistence-utils-hibernate

Modelo de Dados

O sistema possui várias entidades principais:

1. **Usuario**: Representa os usuários do sistema com diferentes perfis
2. **Cidadao**: Representa os cidadãos que solicitam serviços
3. **Pedido**: Representa as solicitações de serviços
4. **TipoServico**: Categoriza os diferentes tipos de serviços oferecidos
5. **StatusPedido**: Representa os diferentes estados de um pedido
6. **EtapaProcesso**: Representa as etapas do fluxo de processamento de um pedido
7. **Favorito**: Permite aos usuários marcar serviços como favoritos

Segurança

O sistema utiliza Spring Security com autenticação baseada em JWT: - Filtro de autenticação JWT personalizado - Endpoints públicos: `/auth/**`, `/v3/api-docs/**`, `/swagger-ui/**`, `/pedidos/codigo/**`, `/configuracoes` - Controle de acesso baseado em perfis (ADMINISTRADOR, ATENDENTE, GESTOR, TECNICO, FISCAL) - Sessões stateless

Containerização

O projeto inclui um Dockerfile para containerização da aplicação: - Baseado na imagem openjdk:17-jdk-slim - Compila a aplicação com Maven - Expõe a porta 8080

Scripts de Automação

- **build.sh**: Script para compilar o projeto
- **run.sh**: Script para executar a aplicação, verificando a versão do Java e compilando se necessário

Ponto de Entrada

A classe principal da aplicação é `SimpleApplication.java`, que utiliza as seguintes anotações: - `@SpringBootApplication` - `@EnableJpaAuditing` - `@EnableCaching`

Conclusão

O projeto segue uma arquitetura bem estruturada e modular, utilizando as melhores práticas de desenvolvimento com Spring Boot. A separação clara entre as camadas (controller, service, repository) facilita a manutenção e evolução do sistema. A implementação de segurança com JWT e controle de acesso baseado em perfis demonstra preocupação com a proteção dos dados e funcionalidades do sistema.

O uso de Docker facilita a implantação e garante consistência entre ambientes. A documentação da API com OpenAPI/Swagger facilita o entendimento e teste dos endpoints disponíveis.