Boas Práticas e Padrões de Projeto

Sumário:

1. Arquitetura de Microsserviços

O sistema adota arquitetura de microsserviços com três serviços independentes: Cadastro (PostgreSQL), Agendamento (MongoDB) e Estoque. Cada serviço possui responsabilidade única e pode escalar de forma independente, promovendo desacoplamento e flexibilidade tecnológica.

2. Clean Architecture e Separação em Camadas

A estrutura segue os princípios de Clean Architecture com camadas bem definidas:

- Domain: Entidades de negócio (Pessoa, Agendamento, Item) com herança e polimorfismo
- Application: Use Cases, Services e DTOs isolando regras de negócio
- Infrastructure: Repositories e integrações externas
- Web: Controllers REST exposindo APIs

Esta separação garante testabilidade, manutenibilidade e conformidade com o Princípio da

Responsabilidade Única (SRP).

3. Padrões de Projeto Implementados

3.1 Repository Pattern

Abstrações (JpaRepository, MongoRepository) encapsulam acesso a dados, permitindo trocar implementações sem impactar camadas superiores.

3.2 Use Case Pattern
Classes específicas (AgendarConsultaUseCase, CadastrarNovoPacienteUseCase) encapsulam fluxos de negócio completos, facilitando testes unitários e segregação de responsabilidades.

3.3 DTO Pattern (Data Transfer Object)

DTOs (PacienteDTO, AgendarConsultaDTO) desacoplam a representação externa das entidades internas, evitando exposição de detalhes de implementação.

3.4 Strategy Pattern (Herança)

Hierarquias de classes (Pessoa → Usuario/Paciente/Medico, Agendamento → Consulta/Exame/Procedimento) aplicam polimorfismo, permitindo tratamento específico por tipo.

3.5 Dependency Injection

Uso extensivo de injeção de dependência via construtor (@RequiredArgsConstructor do Lombok) promove baixo acoplamento e facilita testes com mocks.

4. Princípios SOLID Aplicados

S - Single Responsibility: Cada classe tem uma única responsabilidade (ex: PacienteService gerencia apenas pacientes)
O - Open/Closed: Entidades abstratas (Pessoa, Agendamento) permitem extensão sem modificação
L - Liskov Substitution: Subclasses podem substituir superclasses sem quebrar funcionalidade
I - Interface Segregation: Repositories específicos (PacienteRepository) em vez de interfaces genéricas gigantes
D - Dependency Inversion: Camadas superiores dependem de abstrações (interfaces Repository) não de implementações

5. Boas Práticas de Código

A. Transações (@Transactional) garantindo consistência de dados
B. Validações centralizadas (Bean Validation, exceções customizadas)
C. Exception Handling global (GlobalExceptionHandler) com @RestControllerAdvice
D. Imutabilidade parcial com Lombok (@Data, @Builder)
E. Nomenclatura clara e expressiva seguindo convenções Java
F. Uso de Enums para valores controlados (StatusAgendamentoEnum, ModalidadeEnum)

6. Integração e Comunicação

Microsserviços mantêm referências fracas (strings: CPF, CRM) evitando acoplamento de bancos. Sistema preparado para evolução com eventos assíncronos e mensageria.

7. Testabilidade

Estrutura facilita testes em múltiplos níveis: unitários (Use Cases isolados), integração (repositories) e E2E (controllers). Evidências em RELATORIO_TESTES.md e TESTES_CRIADOS.md.