Etapa 5 — Estratégias de Segurança e Qualidade

Introdução

Nesta etapa, o foco está na qualidade estrutural e organizacional do sistema, assegurada pela aplicação dos princípios SOLID e pela separação clara de responsabilidades entre os módulos da aplicação.

Essas práticas visam tornar o código modular, coeso e facilmente extensível, reduzindo o acoplamento e facilitando futuras manutenções e integrações.

Aplicação dos Princípios SOLID

Os princípios SOLID foram aplicados ao longo do desenvolvimento para garantir um design limpo, modular e sustentável. A seguir, cada princípio é descrito junto com sua aplicação prática no projeto.

Princípio	Descrição	Aplicação no Projeto
S — Single Responsibility Principle (Responsabilidade Única)	Cada classe deve ter apenas uma responsabilidade.	Separação clara entre as camadas: entidades cuidam do modelo de dados, repositórios da persistência, serviços da lógica de negócio e controladores da comunicação com o cliente. Exemplo: a classe RequisicaoService é responsável apenas pela lógica de requisições, sem acessar diretamente o banco.
O — Open/Closed Principle (Aberto para Extensão, Fechado para Modificação)	O código deve permitir novas funcionalidades sem alterar o existente.	O uso de interfaces e DTOs facilita a extensão de funcionalidades, como adicionar novos tipos de requisição ou material, sem alterar classes base.
L — Liskov Substitution Principle (Substituição de Liskov)	Classes derivadas devem poder substituir suas classes base sem quebrar o sistema.	Aplicado em hierarquias de classes e uso de heranças em entidades e serviços, garantindo compatibilidade entre tipos concretos e abstratos.
I — Interface Segregation Principle (Segregação de Interface)	Interfaces específicas são melhores que interfaces grandes e genéricas.	Cada repositório herda apenas de JpaRepository com seu tipo específico (ex: MaterialRepository, RequisicaoRepository), evitando métodos genéricos desnecessários.

D — Dependency Inversion Principle (Inversão de Dependência) Classes de alto nível não devem depender de classes de baixo nível, mas de abstrações.

Os controladores dependem de interfaces de serviço (como RequisicaoService), e não de implementações diretas, o que permite troca e teste isolado de componentes.

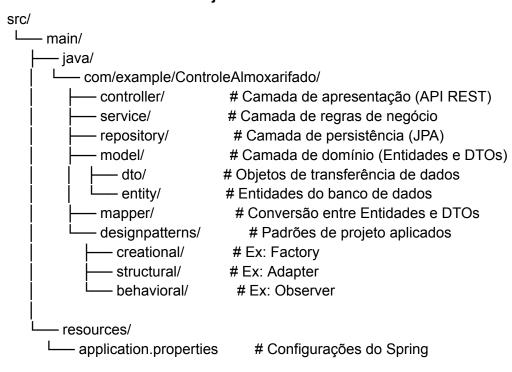
Esses princípios foram aplicados de forma contínua para garantir coerência arquitetural, reusabilidade de código e testabilidade.

Separação Clara de Responsabilidades entre Módulos

O projeto segue uma arquitetura em camadas, com separação rigorosa entre camadas de domínio, persistência, lógica e apresentação.

Isso garante uma estrutura organizada, fácil de manter e adequada ao uso de frameworks como Spring Boot e JPA.

Estrutura Geral do Projeto



Responsabilidade de Cada Módulo

- Controller/: Camada responsável por receber as requisições HTTP e responder com os dados apropriados, utilizando controladores REST.
- **Service**/: Contém as regras de negócio e lógica do sistema. Cada serviço deve ser responsável por um conjunto de operações que não envolvem diretamente a

manipulação de dados no banco.

- **Repository**/: Camada de persistência de dados via Spring Data JPA, onde as interfaces de repositório gerenciam a comunicação com o banco de dados.
- Model/: Onde as entidades JPA são definidas, representando as tabelas do banco de dados. Também contém os DTOs usados para a comunicação entre o frontend e o backend.
- **Mapper/**: Contém as classes que realizam a conversão entre entidades JPA e DTOs, evitando duplicação de lógica.
- Designpatterns/: Implementação dos padrões de projeto (creacionais, estruturais, comportamentais) utilizados no sistema para garantir flexibilidade e manutenibilidade.
- Resources/: Pasta contendo o arquivo de configuração do Spring (application.properties).

Benefícios Obtidos

- Facilidade de manutenção: cada módulo pode ser alterado ou estendido sem afetar os demais.
- Alta coesão e baixo acoplamento: cada classe cumpre apenas um propósito bem definido.
- Facilidade de testes: serviços e controladores podem ser testados isoladamente.
- Clareza estrutural: a organização por camadas facilita o entendimento e evolução do sistema.

Conclusão

A aplicação dos princípios SOLID e da separação clara de responsabilidades resultou em uma base de código limpa, modular e escalável.

Essas práticas fortalecem a qualidade da arquitetura e garantem que o sistema possa crescer de forma sustentável, mantendo a consistência entre os componentes e a integridade do domínio.