Kenedy Alves Gondim - SP3132307

1. Descrição do Domínio e Arquitetura do Projeto

O projeto é uma aplicação monolítica, focada no domínio de **filmes**. Seu propósito principal é ilustrar o uso do Java Reflection e da biblioteca Jackson.

A arquitetura de design é simples, seguindo um padrão monolítico com a separação de responsabilidades em camadas:

- Models: Representam as entidades do banco de dados (`Filme`).
- DTOs**:** Objetos utilizados para transferência de dados entre o servidor e o cliente, expondo apenas as informações necessárias (`FilmeDto`).
- Controllers**:** Camada de API REST, responsável por receber requisições HTTP e retornar respostas.
- Configurations**:** Classes de configuração do Java Reflection e Jackson.
- Repositories: Responsáveis pela comunicação com o banco de dados

2. Limitações e Riscos da Java Reflection

Embora a Reflection seja uma ferramenta poderosa, seu uso indiscriminado introduz diversas limitações e riscos que devem ser considerados.

- ***Problemas de Performance***

A principal limitação da Reflection é o desempenho. A invocação de um método ou o acesso a um campo via Reflection é significativamente mais lento do que o acesso direto em tempo de compilação. Isso ocorre porque o Java Virtual Machine (JVM) não pode otimizar as chamadas de método refletivas da mesma forma que otimiza o código padrão.

No contexto do projeto: Para uma aplicação de demonstração que lida com poucos filmes e um baixo volume de requisições, essa perda de performance é insignificante. No entanto, em um sistema de alta performance, com milhares de requisições esse efeito pode ser sentido.

- ***Violação do Encapsulamento***

O uso de `setAccessible(true)` é a principal ferramenta da Reflection para ignorar modificadores de acesso (`private`, `protected`). Ao fazer isso, o código viola

diretamente o princípio de **encapsulamento**, permitindo que uma classe externa manipule o estado interno de outra, ignorando a intenção do design original.

- ***Dificuldade de Manutenção e Depuração***

O código que utiliza Reflection é menos legível e mais difícil de manter..

3. Mitigação de Riscos e Decisões de Design

Para mitigar os riscos e limitações da Reflection, o projeto implementou decisões de design conscientes:

- ***Estratégia de Mapeamento Seletivo***

O mapeamento de `Filme` para `FilmeDto` foi projetado para ser **seletivo**. O DTO expõe apenas os campos relevantes para o cliente, como `nome` e `ano_lancamento`. O campo de `senha` (se existisse) não seria incluído no DTO, o que naturalmente evita a necessidade de um `setAccessible(true)` para campos confidenciais. Essa é a principal forma de mitigar o risco de violação de segurança.

- **Alternativas Seguras**

Bibliotecas de Geração de Código: Ferramentas como o **MapStruct** geram o código de mapeamento em tempo de compilação. Isso elimina a sobrecarga da Reflection em tempo de execução e garante a segurança e o encapsulamento, pois o código gerado faz acessos diretos e seguros aos métodos `public` (getters e setters) e falha na compilação se a estrutura for alterada.