Especificação da Camada de Aplicação

1. Princípios e Estratégia

A Camada de Aplicação é o coração da orquestração dos casos de uso de negócio. Os seus princípios fundamentais são:

1. **Orquestração, Não Lógica de Negócio:** Os serviços de aplicação coordenam o fluxo de trabalho. Eles invocam os `Output Ports` para interagir com a infraestrutura e utilizam os objetos de Domínio para executar a lógica de negócio. A lógica de negócio crítica e as regras de estado residem exclusivamente no Modelo de Domínio.
2. **Agnóstica à Entrega e à Persistência:** A camada de aplicação não tem conhecimento de HTTP, JSON, JPA ou SQL. Ela opera com `Commands` e `DTOs` puros e interage com o mundo exterior através das interfaces dos `Ports`, garantindo a sua total independência da infraestrutura.
3. **Implementação dos Input Ports:** Os serviços de aplicação são as implementações concretas dos `Input Ports` (Casos de Uso), definindo as funcionalidades que o sistema oferece e formando a sua API interna.

2. Estrutura de Implementação (Mantida)

A estrutura de pacotes é mantida por sua clareza e excelente separação de responsabilidades:

- `...application.port.in`: Contém as interfaces dos `Input Ports` (Casos de Uso) e seus `Commands` e `DTOs`.
- `...application.service`: Contém as implementações concretas dos casos de uso (anotadas com `@Service`).

3. Fluxo de Trabalho de um Caso de Uso

O fluxo de trabalho de um caso de uso agora reflete a interação com os `Ports`, desacoplando completamente a aplicação da infraestrutura. Usando o caso de uso "Criar um Novo Projeto" como exemplo:

1. **Contrato do Caso de Uso (Input Port):** Uma interface `CreateProjectUseCase` define o contrato, aceitando um `CreateProjectCommand` e retornando um

`ProjectDTO`.

2. **Comando (Command):** A classe `CreateProjectCommand` encapsula os dados de entrada e inclui validações (`jakarta.validation`).
3. **Serviço de Aplicação (`CreateProjectService`):** A implementação concreta (`@Service`) orquestra o fluxo:
    - **Passo 1: Demarcação da Transação:** O método público do serviço é anotado com `@Transactional`. Esta é a fronteira que garante a atomicidade da operação.
    - **Passo 2: Receber o Command:** O serviço recebe o `CreateProjectCommand`.
    - **Passo 3: Carregar Agregados e Validar Permissões:** O serviço utiliza os `Output Ports` (injetados via construtor, ex: `WorkspaceRepositoryPort`) para carregar os agregados de domínio necessários (ex: `workspaceRepositoryPort.findById(...)`). Em seguida, realiza as verificações de autorização. Se a permissão for negada, lança uma `ForbiddenException`.
    - **Passo 4: Executar a Lógica de Negócio:** O serviço invoca os métodos do agregado de domínio (ex: `workspace.createProject(...)`) para aplicar as regras de negócio. O agregado retorna um novo objeto de domínio `Project`.
    - **Passo 5: Persistir o Novo Estado:** O serviço chama o `Output Port` de persistência correspondente (ex: `projectRepositoryPort.save(newProject)`).
    - **Passo 6: Mapear e Retornar o DTO:** O serviço mapeia o objeto de domínio `Project` retornado para um `ProjectDTO` e o retorna.

4. Responsabilidades Fundamentais da Camada de Aplicação

Com a arquitetura limpa, as responsabilidades da Camada de Aplicação são refinadas e focadas:

- **Gerenciamento de Transações:** É a **única** camada que deve conter a anotação `@Transactional`, definindo as fronteiras das unidades de trabalho do sistema.
- **Tratamento de Autorização:** É a responsabilidade fundamental da Camada de Aplicação garantir que um utilizador tenha as permissões necessárias para executar uma ação. Ela orquestra o carregamento dos dados necessários para a decisão de autorização.

- **Orquestração de Casos de Uso:** Coordena a interação entre o domínio e a infraestrutura (através dos `Ports`), garantindo que a lógica de negócio seja executada na ordem correta e dentro de uma transação consistente.

Ensino e Análise Final

A camada de aplicação, numa arquitetura limpa, é como um maestro de uma orquestra. O maestro (`Application Service`) não toca nenhum instrumento (`lógica de negócio`), nem construiu o teatro (`infraestrutura`). Ele lê a partitura (`UseCase`) e coordena os músicos (`Domain Objects`) e a equipa de palco (`Adapters` via `Ports`) para produzir a sinfonia. Esta reestruturação posiciona a Camada de Aplicação exatamente nesse papel: um orquestrador puro. Ao depender apenas de interfaces (`Ports`), ela se torna extremamente fácil de testar. Podemos "simular" (`mock`) os `Ports` para verificar a lógica de orquestração do serviço de forma isolada, sem precisar de um banco de dados ou de qualquer outra infraestrutura real. Esta testabilidade e clareza de propósito são os principais benefícios desta abordagem.