Especificação de Ports e Adapters

1. Princípios e Estratégia

Esta especificação detalha a implementação da Arquitetura Hexagonal (Ports and Adapters), que cria uma separação rigorosa entre o núcleo da aplicação (domínio e casos de uso) e os detalhes de infraestrutura (banco de dados, API REST, serviços de IA). Os princípios são:

- **Separação de Responsabilidades:** O núcleo da aplicação não deve ter conhecimento de como os dados são apresentados ao utilizador (HTTP/JSON) nem de como são armazenados (PostgreSQL/JPA).
- **Contratos Explícitos (Ports):** A comunicação entre o núcleo e a infraestrutura é definida por interfaces explícitas (Ports). Os `Input Ports` representam os casos de uso, e os `Output Ports` representam os requisitos de infraestrutura do núcleo (ex: "preciso de persistir este objeto").
- **Implementações Intercambiáveis (Adapters):** Os `Adapters` são as implementações concretas dos `Ports`. Eles traduzem os sinais da infraestrutura para chamadas no núcleo da aplicação (`Driving Adapters`) ou implementam os requisitos do núcleo usando uma tecnologia específica (`Driven Adapters`).

2. Estrutura de Pacotes

A estrutura de pacotes é organizada para refletir a arquitetura:

- `com.axonai.domain`: Contém o modelo de domínio puro.
- `com.axonai.application.port.in`: Contém as interfaces dos `Input Ports` (Casos de Uso) e seus `Commands` e `DTOs` de retorno.
- `com.axonai.application.port.out`: Contém as interfaces dos `Output Ports` para infraestrutura externa (ex: `ProjectRepositoryPort`, `AiModelPort`, `ApiKeyVault`).
- `com.axonai.application.service`: Contém as implementações dos Casos de Uso (dos `Input Ports`).
- `com.axonai.adapter.in.web`: Contém o `Driving Adapter` para a API REST (Controllers, DTOs da Web e Mappers).

- `com.axonai.adapter.out.persistence`: Contém o `Driven Adapter` para a persistência (Entidades JPA, Repositórios Spring Data, Mappers de persistência e a implementação do `RepositoryPort`).
- `com.axonai.adapter.out.ai`: Contém o `Driven Adapter` para a interação com LLMs (ex: `GeminiAdapter`).
- `com.axonai.adapter.out.security`: Contém `Driven Adapters` para componentes de segurança (ex: implementação do `ApiKeyVault`).

3. Detalhamento dos Ports (Contratos)

- **Input Ports (Casos de Uso):**
  - Definidos como interfaces em `...application.port.in`. Eles formam a API da camada de aplicação, definindo todas as funcionalidades que o sistema oferece.
  - Exemplos: `RegisterUserUseCase`, `CreateProjectUseCase`, `ProposeTasksForProjectUseCase`.
- **Output Ports (Requisitos de Infraestrutura):**
  - Definidos como interfaces em `...application.port.out`. A camada de aplicação depende destas interfaces, não das suas implementações.
  - **Port de Persistência:** `ProjectRepositoryPort` define métodos que operam sobre objetos de domínio puros, como `save(Project project)` e `findById(ProjectId id): Optional<Project>`.
  - **Port de IA:** `AiModelPort` define o contrato para interagir com um modelo de IA (`execute` e `executeStream`).
  - **Port de Segurança:** `ApiKeyVault` (anteriormente `ApiKeyStore`) define o contrato para armazenar e recuperar chaves de API de forma segura.

4. Detalhamento dos Adapters (Implementações)

- **Driving Adapter (Web - `...adapter.in.web`):**
  - **Componentes:** `@RestController`, `Request/Response DTOs` e `Mappers`.
  - **Responsabilidade:** Mapear requisições HTTP para `Commands` da camada de aplicação, invocar o `UseCase` correspondente e mapear os `DTOs` da

aplicação para `Responses HTTP`. Esta camada é responsável por toda a interação com o protocolo HTTP e a serialização JSON.

- **Driven Adapters (Saída):**
  - **Adaptador de Persistência (`...adapter.out.persistence`):**
    - **Componentes:** `ProjectPersistenceAdapter`, `Entidades JPA`, `Mappers de Persistência` e as interfaces `JpaRepository` do Spring Data.
    - **Responsabilidade:** A classe `ProjectPersistenceAdapter` implementa a interface `ProjectRepositoryPort`. Internamente, ela usa um `Mapper` para converter o objeto de domínio `Project` numa `ProjectEntity` JPA, e depois usa o `ProjectJpaRepository` para persistir a entidade no banco de dados.
  - **Adaptador de IA (`...adapter.out.ai`):**
    - **Componentes:** `GeminiAdapter`, `OpenAIAdapter`.
    - **Responsabilidade:** Implementar a interface `AiModelPort`, traduzindo o `AIRequest` genérico para a API específica do provedor e mapeando a resposta de volta.
  - **Adaptador de Segurança (`...adapter.out.security`):**
    - **Componentes:** `KmsApiKeyVault`.
    - **Responsabilidade:** Implementar a interface `ApiKeyVault`, orquestrando a encriptação e desencriptação das chaves de API com um KMS externo.

5. Fluxo de Exemplo Revisado: "Criar um Projeto"

Este fluxo ilustra a arquitetura com o desacoplamento completo:

1. **Requisição HTTP:** O cliente envia `POST /api/v1/projects` com um corpo JSON correspondente ao `CreateProjectRequest` DTO da camada web.
2. **Controller (`ProjectController`):** Recebe o `CreateProjectRequest`. O Spring Security valida o JWT e a identidade do utilizador é extraída.
3. **Mapeamento (Web):** O `WebMapper` é invocado para converter o `CreateProjectRequest` num `CreateProjectCommand` (o objeto esperado pela camada de aplicação).

4. **Invocação do Caso de Uso:** O controller invoca o método do `CreateProjectUseCase`, passando o `Command` e o `userId`.
5. **Serviço de Aplicação (`CreateProjectService`):**
    - Recebe o `Command`.
    - Executa a lógica de autorização.
    - Cria um novo objeto de domínio `Project` (puro, sem anotações JPA).
    - Invoca o port de persistência: `projectRepositoryPort.save(newProject)`.
    - Mapeia o objeto `Project` retornado para um `ProjectDTO` e o retorna.
6. **Adaptador de Persistência (`ProjectPersistenceAdapter`):**
    - Recebe a chamada do `save(newProject)`.
    - Usa o `PersistenceMapper` para converter o objeto de domínio `Project` numa `ProjectEntity` JPA.
    - Chama `projectJpaRepository.save(projectEntity)` para persistir no banco de dados.
    - Mapeia a entidade salva de volta para um objeto de domínio e o retorna.
7. **Mapeamento (Resposta):** De volta ao `Controller`, o `ProjectDTO` recebido do serviço é mapeado para um `ProjectResponse` DTO pelo `WebMapper`.
8. **Resposta HTTP:** O controller retorna o `ProjectResponse` no corpo da resposta JSON com o status `201 Created`.
