Especificação da Camada de Persistência

1. Princípios e Estratégia

A camada de persistência no AxonAI atua como um **Adaptador de Saída (Driven Adapter)**. A sua única responsabilidade é implementar os `Output Ports` de repositório definidos pela Camada de Aplicação, traduzindo as operações sobre objetos de domínio puros em interações com o banco de dados.

- **Separação de Modelos:** Existe uma separação rigorosa entre os objetos do **Modelo de Domínio** e as **Entidades de Persistência (JPA)**. A camada de persistência opera exclusivamente com Entidades JPA internamente, e expõe apenas objetos de Domínio através dos `Ports`.
- **Implementação de Contratos:** Esta camada implementa as interfaces `Repository Port` (ex: `ProjectRepositoryPort`). Ela é um detalhe de infraestrutura, e o núcleo da aplicação não tem conhecimento sobre JPA, Hibernate ou SQL.
- **Responsabilidade Única:** A responsabilidade da camada de persistência é exclusivamente a **gestão do estado** (salvar, carregar, atualizar, deletar) dos agregados no banco de dados.

2. Estrutura de Implementação

O código da camada de persistência residirá no pacote `com.axonai.adapter.out.persistence`.

- **`...persistence.entity`**: Contém as classes de entidade JPA (`@Entity`). Estas classes representam a estrutura das tabelas do banco de dados e **não contêm lógica de negócio**.
- **`...persistence.repository`**: Contém as interfaces que estendem o `JpaRepository` do Spring Data. Elas operam sobre as Entidades JPA (ex: `ProjectJpaRepository extends JpaRepository<ProjectEntity, UUID>`).
- **`...persistence.mapper`**: Contém as classes de `Mapper` (ex: `ProjectPersistenceMapper`). A sua responsabilidade é a tradução bidirecional entre objetos de Domínio (ex: `Project`) e Entidades JPA (ex: `ProjectEntity`).
- **`...persistence.adapter`**: Contém as implementações dos `Output Ports` (ex: `ProjectPersistenceAdapter`). Esta é a fachada da camada de persistência.

3. Estratégias Operacionais

- **Gerenciamento de Transações:**
  - **Localização:** A anotação `@Transactional` do Spring permanece aplicada exclusivamente nos métodos públicos dos **Serviços da Camada de Aplicação** (ex: `CreateProjectService`).
  - **Justificativa:** O caso de uso define a unidade de trabalho do negócio. A transação deve abranger todo o caso de uso, que pode envolver múltiplas chamadas a diferentes repositórios. O `Adapter` de persistência executa dentro desta fronteira transacional.
- **Estratégia de Fetching e Prevenção de `LazyInitializationException`:**
  - **Padrão:** Todas as associações (`@OneToMany`, `@ManyToOne`, etc.) nas Entidades JPA são configuradas com `FetchType.LAZY` por padrão.
  - **Regra de Ouro:** A responsabilidade de carregar os dados necessários para um caso de uso é do `Adapter de Persistência`.
  - **Implementação:** O `Serviço de Aplicação` solicita um agregado através do `Repository Port` (ex: `projectRepositoryPort.findProjectWithTasksById(projectId)`). A implementação no

    `ProjectPersistenceAdapter` delega a chamada para um método no `JpaRepository` que utiliza `JOIN FETCH` ou `@EntityGraph` para garantir que o agregado seja carregado completamente dentro da transação. O

    `Mapper` então traduz a `Entity` totalmente inicializada para um objeto de Domínio.
- **Mapeamento de Exceções:**
  - **Estratégia:** O `Adapter de Persistência` é a fronteira que impede o vazamento de exceções de infraestrutura.
  - **Implementação:** Os métodos no `PersistenceAdapter` devem capturar exceções específicas do Spring/JPA (ex: `DataIntegrityViolationException`) e relançá-las como uma exceção da aplicação mais apropriada (ex: `ConflictException`), conforme definido na "Estratégia de Tratamento de Exceções".

4. Estratégia de Testes e Evolução de Schema (Mantida e Reforçada)

A estratégia de testes é crucial para garantir a robustez desta camada.

- **Migração de Schema com Flyway:**
  - **Ferramenta:** Flyway é mandatório para gerir a evolução do schema do banco de dados através de scripts SQL versionados, localizados em `src/main/resources/db/migration`.
- **Estratégia de Testes:**
  1. **Testes de Slice (`@DataJpaTest`):** Focados em testar o **mapeamento** das Entidades JPA e a lógica de **queries customizadas** dos `JpaRepositories`. Utilizam um banco de dados em memória (H2) para velocidade.
  2. **Testes de Integração (Testcontainers):** Focados em testar o `PersistenceAdapter` de ponta a ponta. Estes testes invocam os métodos do `Adapter` (que recebem e retornam objetos de Domínio) e verificam se o estado é corretamente persistido e recuperado. Utilizam Testcontainers para instanciar um container Docker do PostgreSQL real, garantindo que os testes rodem contra um schema idêntico ao de produção.

Ensino e Análise Final

Com esta especificação, a camada de persistência assume seu papel correto na Arquitetura Hexagonal: um "detalhe" de implementação. O núcleo da aplicação simplesmente diz: "persista este objeto de domínio", através de um `Port`. Como isso acontece – se é com JPA, JDBC, ou mesmo escrevendo em ficheiros – é problema do `Adapter`. Esta separação rigorosa, validada por uma estratégia de testes robusta com Testcontainers, garante que o componente mais propenso a mudanças e otimizações (a interação com o banco de dados) possa evoluir de forma independente, sem nunca colocar em risco a integridade da lógica de negócio.