Estratégia de Tratamento de Exceções

1. Filosofia e Objetivos

O tratamento de exceções no AxonAI segue três princípios fundamentais para garantir uma aplicação robusta e operável:

1. **Previsibilidade:** Clientes da API, como o frontend, devem receber uma estrutura de erro consistente e previsível para todas as falhas, permitindo um tratamento de erros padronizado na interface do utilizador.
2. **Segurança:** Nenhuma informação sensível ou detalhes de implementação (como *stack traces*, mensagens de exceções de banco de dados) deve ser exposta ao utilizador final.
3. **Separação de Responsabilidades:** Cada camada da arquitetura tem uma responsabilidade distinta no tratamento de erros, garantindo que detalhes de infraestrutura não vazem para o núcleo da aplicação.

2. Hierarquia de Exceções da Aplicação

A hierarquia de exceções customizadas é desenhada para ser explícita e orientada à intenção, comunicando claramente o que deu errado do ponto de vista do negócio. Todas as exceções controladas da aplicação herdam de uma classe base `AxonAiException`.

- `AxonAiException` (Classe Base): A raiz de todas as exceções controladas.
- **Exceções Baseadas na Intenção:**
  - `NotFoundException`: Lançada quando um recurso específico não pode ser encontrado (ex: projeto, tarefa). Mapeia para **HTTP 404 Not Found**.
  - `ValidationException`: Lançada para erros de validação de input ou violações de regras de negócio que podem ser corrigidas pelo utilizador (ex: título de tarefa em branco). Mapeia para **HTTP 400 Bad Request**.
  - `ForbiddenException`: Lançada quando um utilizador autenticado tenta executar uma ação para a qual não tem permissão. Mapeia para **HTTP 403 Forbidden**.

- `ConflictException`: Lançada quando uma ação não pode ser concluída devido a um conflito com o estado atual do recurso (ex: tentar criar um recurso que já existe com um identificador único). Mapeia para **HTTP 409 Conflict**.

3. Estratégia de Tratamento por Camada

- **Camada de Aplicação (`application.service`):**
  - É a principal fonte das exceções da aplicação (`NotFoundException`, `ForbiddenException`, etc.). Os serviços validam as condições de negócio e de autorização e lançam a exceção apropriada.
- **Adaptadores de Saída (`adapter.out.*`):**
  - **Responsabilidade Crítica:** Esta camada é a fronteira que protege o núcleo da aplicação de exceções específicas da infraestrutura.
  - **Implementação:** Os `Adapters` (ex: `PersistenceAdapter`, `GeminiAdapter`) devem capturar exceções da tecnologia que utilizam (ex: `SQLException`, `DataIntegrityViolationException` do Spring Data, `HttpTimeoutException` do cliente HTTP da IA) e **traduzi-las** para uma exceção da aplicação mais apropriada (ex: `ConflictException` ou uma `InfrastructureException` genérica que resultará num erro 500). Isso impede que o núcleo da aplicação tenha conhecimento sobre `JPA`, `HTTP`, etc.
- **Adaptador de Entrada (`adapter.in.web`):**
  - **Responsabilidade:** Atua como a fronteira final que traduz todas as exceções (tanto as da aplicação quanto as inesperadas) em respostas HTTP padronizadas.
  - **Implementação:** Um `GlobalExceptionHandler` (`@RestControllerAdvice`) centraliza toda a lógica de mapeamento de exceções para respostas HTTP.

4. Formato Padrão da Resposta de Erro da API (Enriquecido)

Toda resposta de erro da API seguirá um formato JSON padronizado e rico em informações, agora incluindo um `traceId` para correlação com os logs.

JSON

```
{
 "timestamp": "2025-09-02T18:30:00.123Z",
 "status": 404,
 "error": "NOT_FOUND",
 "message": "O projeto com o ID '123e4567-e89b-12d3-a456-426614174000' não foi encontrado.",
```

```
  "path": "/api/v1/projects/123e4567-e89b-12d3-a456-426614174000",
  "traceId": "a7b3c1d9-e8f0-4a2b-8c7d-6e5f4g3h2i1j"
}
```

- `traceId`: Um identificador único gerado para cada requisição. Este ID será incluído em **todos os logs** gerados durante o processamento daquela requisição, permitindo rastrear o fluxo completo de uma operação e correlacionar um erro específico reportado pelo cliente com os logs detalhados no backend.

5. Mapeamento no `GlobalExceptionHandler`

O `GlobalExceptionHandler` irá mapear a hierarquia de exceções para os códigos de status HTTP apropriados:

- `@ExceptionHandler(NotFoundException.class)` → **HTTP 404 Not Found**
- `@ExceptionHandler(ForbiddenException.class)` → **HTTP 403 Forbidden**
- `@ExceptionHandler(ValidationException.class)` → **HTTP 400 Bad Request**
- `@ExceptionHandler(ConflictException.class)` → **HTTP 409 Conflict**
- `@ExceptionHandler({MethodArgumentNotValidException.class, ConstraintViolationException.class})` → **HTTP 400 Bad Request**
- `@ExceptionHandler(AIError.class)` → Mapeia o código de erro interno do `AIError` para o status HTTP apropriado.
- `@ExceptionHandler(Exception.class)` → **HTTP 500 Internal Server Error** (catch-all para erros inesperados).

6. Estratégia de Logging

A estratégia de logging é fundamental para a depuração e monitorização.

- **Formato Mandatório:** Todos os logs devem ser emitidos em **JSON estruturado**. Isso torna os logs pesquisáveis e fáceis de analisar por ferramentas de observabilidade (Datadog, ELK Stack, etc.). Cada entrada de log **deve** conter o `traceId`.
- **Nível `WARN`:** Para exceções controladas que resultam em respostas `4xx` (erros do cliente). Estas indicam um problema com a requisição do cliente, não uma falha no servidor. O log deve conter a mensagem de erro e o contexto da requisição.

- **Nível `ERROR`:** Para exceções inesperadas que resultam em respostas `5xx`. Estas indicam uma falha no servidor. O log deve **sempre** incluir o *stack trace* completo para facilitar a depuração.