Especificação do Adaptador Web (API REST)

1. Princípios e Estratégia

A camada Web atua como o Adaptador de Entrada (Driving Adapter) principal do sistema. A sua responsabilidade é expor os casos de uso da Camada de Aplicação através de uma interface RESTful, seguindo os seguintes princípios:

1. **Tradução, Não Lógica:** Os Controllers são tradutores. A sua única função é validar, mapear e delegar. Nenhuma lógica de negócio reside nesta camada.
2. **API como Contrato Público Estável:** A estrutura dos endpoints, os corpos de requisição/resposta (DTOs da Web) e os códigos de status HTTP formam o contrato público da nossa API. Este contrato deve ser estável e evoluir de forma controlada para não quebrar clientes existentes.
3. **Stateless:** A API é completamente stateless. Cada requisição contém toda a informação necessária para ser processada, com a autenticação gerida via JWTs no cabeçalho `Authorization`.

2. Estrutura de Implementação

O código da Camada Web residirá no pacote `com.axonai.adapter.in.web`. A estrutura é desenhada para criar uma fronteira clara (Anti-Corruption Layer) entre o mundo externo e a camada de aplicação.

- `...web.controller`: Contém os Controllers (`@RestController`). Cada controller é focado num recurso de domínio.
- `...web.dto.request`: Contém classes que modelam os corpos das requisições (ex: `CreateProjectRequest`). Incluem anotações de validação (`@NotBlank`, `@Email`, etc.).
- `...web.dto.response`: Contém classes que modelam os corpos das respostas (ex: `ProjectResponse`). Estes DTOs definem a estrutura exata dos dados retornados pela API.

- `...web.mapper`: Contém os Mappers (ex: `ProjectWebMapper`) responsáveis pela tradução bidirecional entre os DTOs da Web e os Commands / DTOs da Camada de Aplicação.
- `...web.exception`: Contém o `GlobalExceptionHandler` (`@RestControllerAdvice`), responsável por traduzir exceções em respostas HTTP padronizadas.

3. Convenções da API REST (Mantidas)

Para garantir consistência e previsibilidade, a API seguirá as seguintes convenções padrão da indústria:

- **Versioning:** Todas as rotas terão o prefixo `/api/v1`.
- **Nomenclatura de Recursos:** Substantivos no plural (ex: `/projects`, `/tasks`).
- **Métodos HTTP:** Utilização semântica dos verbos HTTP (`POST`, `GET`, `PUT` / `PATCH`, `DELETE`).
- **Códigos de Status HTTP:** Utilização correta dos códigos de status (`201 Created`, `200 OK`, `204 No Content`, `400 Bad Request`, `401 Unauthorized`, `403 Forbidden`, `404 Not Found`).

> ## Endpoint Crítico: Ativação do "Modo Foco"
>
> A operação de focar em um item do checklist, central para a aplicação, é exposta da seguinte forma:
>
> - **Endpoint:** `PUT /api/v1/tasks/{taskId}/checklist-items/{checklistItemId}/focus`
> - **Método HTTP:** `PUT`. A escolha se deve à idempotência da operação: chamar a mesma requisição múltiplas vezes resulta no mesmo estado final (aquele item específico estará focado).
> - **Descrição:** Ativa o "Modo Foco" para um `ChecklistItem` específico dentro de uma `Task`. O `TaskAggregate` no domínio garantirá que qualquer outro item previamente focado seja desfocado.
> - **Corpo da Requisição:** Vazio.
> - **Resposta de Sucesso:** `200 OK` - Retorna o DTO (`ChecklistItemResponse`) do item que foi focado, refletindo seu novo estado.
> - **Respostas de Erro:**
>   - `403 Forbidden`: O usuário autenticado não tem permissão para modificar a tarefa.

- `404 Not Found`: A tarefa (`taskId`) ou o item de checklist (`checklistItemId`) não foram encontrados.

4. Exemplo de Fluxo Revisado: Criar um Projeto

Este fluxo ilustra a arquitetura desacoplada em ação:

1. **Requisição:** O cliente envia `POST /api/v1/projects` com um corpo JSON que corresponde à estrutura do DTO `CreateProjectRequest`. O `Access Token` JWT é enviado no cabeçalho `Authorization`.
2. **Controller:** O método no `ProjectController` é ativado. O Spring Security valida o JWT e a identidade do utilizador é extraída. O corpo da requisição é mapeado para o objeto `CreateProjectRequest`. A validação (`@Valid`) é acionada.
3. **Mapeamento para a Aplicação:** O `Controller` invoca o `WebMapper`, que traduz o DTO `CreateProjectRequest` para um `CreateProjectCommand`, o objeto esperado pela camada de aplicação.
4. **Delegação para o Caso de Uso:** O controller invoca o `CreateProjectUseCase` (o Input Port), passando o `CreateProjectCommand` preenchido e o `userId` extraído do Principal.
5. **Resposta da Aplicação:** O caso de uso processa a lógica e retorna um `ProjectDTO` da camada de aplicação.
6. **Mapeamento da Resposta:** O `Controller` recebe o `ProjectDTO` e invoca o `WebMapper` novamente para traduzi-lo num `ProjectResponse`, o DTO que define o contrato de resposta da API.
7. **Resposta HTTP:** O controller coloca o `ProjectResponse` dentro de uma `ResponseEntity` com status `201 Created` e o cabeçalho `Location`, e a retorna ao cliente.

5. Integração com a Camada de Segurança (Mantida)

- **Extração de Identidade:** Os `Controllers` são a única camada responsável por interagir com o `SecurityContext` do Spring para extrair o `userId` do `Principal` injetado.
- **Isolamento:** Nenhum artefacto do Spring Security (como o objeto `Principal`) é passado para a Camada de Aplicação. Apenas o

`userId` (um tipo primitivo ou UUID) é passado para os `Commands`, mantendo a camada de aplicação agnóstica ao framework de segurança.

6. Tratamento Global de Exceções (Mantido)

- **Centralização:** A classe `GlobalExceptionHandler` (`@RestControllerAdvice`) interceptará todas as exceções lançadas pelas camadas inferiores.
- **Tradução:** Ela conterá métodos (`@ExceptionHandler`) para mapear exceções específicas da aplicação para as respostas HTTP padronizadas e seguras, conforme a "Estratégia de Tratamento de Exceções".

Ensino e Análise Final

A reestruturação do Adaptador Web completa a implementação da Arquitetura Hexagonal no backend. A introdução da camada de DTOs e Mappers da Web pode parecer um boilerplate adicional, mas na verdade é a implementação de uma **Camada Anti-Corrupção (Anti-Corruption Layer)**. Esta camada protege o núcleo da sua aplicação das inevitáveis mudanças e da complexidade do mundo exterior (neste caso, o protocolo HTTP). Ela permite que o contrato da sua API evolua de forma independente da lógica de negócio interna, um fator crucial para a manutenibilidade e escalabilidade a longo prazo de qualquer sistema.