Feature 3: Modularização e design de APIs com DTOs

Esta feature visa aprofundar a compreensão e aplicação de arquiteturas multi-módulos e o uso estratégico de Data Transfer Objects (DTOs). O objetivo é capacitar na construir sistemas que não apenas se integram com fontes externas, mas que também apresentam dados de forma inteligente e são desenhados para atender às necessidades específicas de seus consumidores.

Objetivo:

Aplicar os conceitos de arquitetura multi-módulos, DTOs e orquestração de dados para:

1. **Estruturar aplicações de forma modular**, garantindo a separação de responsabilidades.
2. **Proteger os modelos de domínio** e gerenciar validações de entrada/saída.
3. **Projetar APIs flexíveis e orientadas ao cliente**, capazes de enriquecer e consolidar informações de diferentes fontes.

Desafio/Tarefas:

O aluno terá duas opções principais para aplicar os conceitos, e ambas devem incorporar DTOs e uma estrutura multi-módulos.

Opção 1: Refatoração de projeto existente

Utilize o projeto desenvolvido nas Features 1 e 2 como base. O desafio é refatorar este projeto para uma estrutura multi-módulos, seguindo a seguinte divisão lógica:

1. `common-domain` **(Módulo de domínio compartilhado):**
    - **Responsabilidade:** conter as classes de domínio (entidades JPA) e enums que são fundamentais para o negócio e precisam ser compartilhadas por múltiplos módulos. Estas classes representam o "vocabulário comum" do seu sistema.
    - **O que colocar:** entidades como `Vendedor`, `Endereco`, `Produto`, `Pedido` (as entidades principais, sem anotações específicas de API ou de validação de request).
    - **Importância:** garante consistência e reusabilidade das definições de negócio em todo o projeto.
2. `external-api-client` **(Módulo de cliente de APIs externas):**
    - **Responsabilidade:** encapsular a lógica de comunicação com APIs externas que o seu projeto consome.
    - **O que colocar:** Feign Clients e os DTOs específicos da *resposta* dessas APIs externas (ex: `ViaCepClient` e `ViaCepResponse`).
    - **Importância:** isola a complexidade da integração com terceiros, protege o restante da aplicação de mudanças em APIs externas e facilita a substituição de serviços externos, se necessário.
3. `main-application` **(Módulo da aplicação principal):**
    - **Responsabilidade:** conter a lógica de negócio principal, os repositórios para persistência de dados e os controladores REST que expõem a API do seu sistema.
    - **Integração:** este módulo dependerá do `common-domain` para suas entidades e do `external-api-client` para consumir serviços externos.
    - **Uso de DTOs:** **aqui é onde os DTOs da sua própria API (Request DTOs e Response DTOs) serão definidos e utilizados.**

        - **Request DTOs:** utilizados para receber dados das requisições HTTP (`POST`, `PUT`). As validações (`@NotBlank`, `@Pattern`, `@Email`, `@Min`, etc.) devem residir nesses DTOs, protegendo sua entidade de domínio de validações externas. Ex: `VendedorRequestDTO`.
        - **Response DTOs:** utilizados para enviar dados nas respostas HTTP (`GET`, `POST`, `PUT`). Eles devem expor apenas os dados relevantes e formatados para o cliente da API, ocultando detalhes internos e enriquecendo dados. Ex: `VendedorResponseDTO`, que pode incluir dados combinados do `Vendedor` e do `Endereco` externo.
    - **Mapeamento:** a camada de serviço será responsável por mapear os `Request DTOs` para as entidades de domínio, realizar a lógica de negócio (possivelmente chamando o `external-api-client` para orquestrar dados) e, finalmente, mapear as entidades de domínio para os `Response DTOs` antes de retornar ao controlador.
    - **Persistência:** contém a configuração do Spring Data JPA e os repositórios (ex: `VendedorRepository`).

Opção 2: Aplicação enxuta multi-módulos com DTOs

Crie uma nova aplicação Spring Boot com uma estrutura multi-módulos (seguindo a lógica acima: `common-domain`, `external-api-client`, `main-application`). Esta aplicação deve focar em **uma única classe de negócio** que você já utilizou em algum projeto anterior (ex: `Produto`, `Cliente`, `Tarefa`).

- **Reaproveitamento:** a entidade (`Produto`, `Cliente` etc) deve ser a mesma já criada, mas agora residindo no módulo `common-domain`.
- **Foco:** o objetivo é demonstrar claramente a separação entre:
    - A entidade de domínio (no `common-domain`).
    - Os DTOs de requisição e resposta para essa entidade (no `main-application`), com validações e projeções de dados.
    - A lógica de negócio e persistência para essa entidade (no `main-application`).
    - A comunicação com uma API externa, se aplicável, encapsulada no `external-api-client`, e como os dados dessa API externa são orquestrados e combinados com os dados internos em um `Response DTO`.

Requisitos e pontos de avaliação para ambas as opções:

1. **Estrutura multi-módulos:** o projeto deve estar organizado em múltiplos módulos, seguindo a lógica de `common-domain`, `external-api-client` e `main-application`.
2. **Aplicação de DTOs:**
    - **Request DTOs:** para *todas* as operações de `POST` e `PUT` da sua API. As validações (`@NotBlank`, `@Email`, `@Min`, etc.) devem residir nesses DTOs.
    - **Response DTOs:** para *todas* as operações de `GET` e para as respostas de `POST`/`PUT`. Eles devem expor apenas os dados relevantes e formatados para o cliente da API, demonstrando como os dados são orquestrados e apresentados de forma otimizada.
    - **Mapeamento:** demonstre claramente o mapeamento entre DTOs e entidades de domínio (e vice-versa) na camada de serviço.
3. **Encapsulamento do domínio:** as entidades de domínio (do `common-domain`) não devem conter anotações de validação específicas de requisições HTTP (`@NotBlank`, `@Email` etc) ou campos transitórios (`@Transient`) para entrada de dados da API. Elas devem focar em sua representação e regras de negócio.
4. **Orquestração de dados:** se o projeto usar uma API externa, demonstre como os dados do `external-api-client` são combinados com os dados do `common-domain` (ou criados internamente) para gerar um `Response DTO` mais completo e útil para o cliente final.
5. **Relatório/README:** explique no `README.md`.
    - A estrutura multi-módulos escolhida e a responsabilidade de cada módulo.
    - Como os DTOs foram aplicados para input e output, justificando as escolhas e demonstrando a **orientação ao cliente** na modelagem dos dados de resposta.
    - A relação entre as entidades de domínio e os DTOs, e como a **orquestração de dados** foi implementada para construir os `Response DTOs`.