1 - Nossa API estava violando o princípio da "Separação de Responsabilidades", também conhecido como o princípio SRP (Single Responsibility Principle). Esse princípio afirma que uma classe ou módulo deve ter apenas uma razão para mudar, ou seja, deve ter uma única responsabilidade bem definida.

Para corrigir essa violação, começamos a refatorar nossa API para separar claramente as responsabilidades. Isso envolveu a reorganização do código, de modo que as regras de negócio, as validações e a persistência de dados fossem tratadas em módulos separados. Dessa forma, cada componente da nossa API passou a ter uma responsabilidade específica e não houve mais a mistura de lógica de negócios com código de persistência ou validação.

Essa abordagem melhorou a manutenção do código, facilitou a identificação de erros e tornou o código mais modular e coeso, seguindo os princípios de design sólido, como o SRP, para criar um sistema mais robusto e de fácil evolução.

2 - Domain (Domínio): Nesta camada, definimos as regras de negócios essenciais para a aplicação. Ela contém as entidades do negócio, como classes que representam objetos importantes para o domínio, bem como os serviços que manipulam essas entidades. A camada de domínio é independente das outras camadas e não deve conter código relacionado a persistência de dados, validações de entrada ou detalhes de implementação.

Data (Dados): A camada de dados é responsável pela interação com a fonte de dados, como um banco de dados, sistema de arquivos ou serviços externos. Ela contém as classes e métodos que lidam com a persistência e recuperação de dados. Essa camada deve traduzir os objetos do domínio em estruturas de dados adequadas para armazenamento e vice-versa.

Application (Aplicação): A camada de aplicação é onde a lógica de negócios da aplicação é orquestrada. Ela utiliza os serviços e entidades definidos na camada de domínio e coordena as operações para atender aos requisitos do caso de uso. Além disso, pode conter lógica relacionada a fluxo de trabalho e regras de aplicação específicas. Essa camada não deve se preocupar com detalhes de como os dados são armazenados ou recuperados.

API (Interface de Programação de Aplicação): A camada de API é responsável por expor as funcionalidades da aplicação para os clientes, que podem ser aplicativos externos, front-ends da web, dispositivos móveis, etc. Ela define os pontos de extremidade (endpoints) e os contratos de comunicação, como APIs RESTful ou GraphQL. Essa camada lida com solicitações HTTP, validações de entrada e saída de dados e geralmente faz a ponte entre os clientes e a camada de aplicação.

Essa divisão em camadas ajuda a manter a aplicação organizada, facilita a manutenção, promove a reutilização de código e permite que as partes da aplicação evoluam de forma independente umas das outras, seguindo os princípios do design de software bem estruturado.  
  
3 - Isso significa que as propriedades de um objeto, como "Produto", não podem ser diretamente modificadas fora da classe. Em vez disso, a classe fornece métodos públicos para acessar e modificar essas propriedades, garantindo que qualquer mudança no estado do objeto seja feita de acordo com as regras de negócios e validações definidas na própria classe.

Essa abordagem proporciona as seguintes vantagens:

Controle de Acesso: A classe tem controle total sobre quem pode alterar seus atributos, o que ajuda a prevenir alterações indevidas ou inconsistentes no estado do objeto.

Validação Centralizada: Qualquer lógica de validação necessária para os atributos pode ser centralizada nos métodos de definição (setters) privados, garantindo que os objetos permaneçam em um estado válido.

Encapsulamento: A abstração do estado interno do objeto é preservada, permitindo que a implementação interna da classe seja alterada sem afetar os clientes que a utilizam.

Manutenção Simplificada: Se houver necessidade de alterar a lógica de validação ou adicionar comportamento específico ao modificar uma propriedade, isso pode ser feito em um único local na classe, tornando a manutenção mais simples e segura.

No caso do "Produto", ao manter as propriedades com setters privados, garantimos que qualquer alteração no preço, nome ou outros atributos do produto seja realizada de acordo com as regras definidas na classe, mantendo a integridade dos objetos e a consistência do domínio da aplicação.

4 - A técnica de injeção de dependência na camada de Application é utilizada para desacoplar as classes de serviço de suas dependências, proporcionando as seguintes vantagens:

Desacoplamento: Permite que as classes de serviço não saibam como suas dependências são implementadas.

Testabilidade: Facilita a criação de testes unitários, pois as dependências podem ser facilmente substituídas por implementações de teste.

Reutilização de Código: Promove a reutilização de código, pois as dependências podem ser compartilhadas entre várias classes de serviço.

Flexibilidade: Permite a troca fácil de implementações de dependência sem afetar as classes de serviço.

Essa abordagem torna o código mais flexível, testável e fácil de manter.