Questão 1:

Em sala no início da disciplina criamos e comentamos que nossa API estava com muitas responsabilidades (regras de negócio, validações, persistência), ferindo assim qual princípio? Descreva o princípio e o que a partir de então começamos a fazer para corrigir isso?

R: Princípio da responsabilidade única, o controller fazia tudo dentro da api.  
utilizamos uma estrutura e arquitetura de dados chamada DDD.

Questão 2:

Começamos a dividir nosso projeto em camadas, são elas:

Domain, Data, Application e API. Descreva qual o papel de cada uma delas.

R:

Domínio (Domain): Define as entidades e as regras de negócios específicas da aplicação.

Dados (Data): Responsável pelo gerenciamento do armazenamento e recuperação de informações dentro da aplicação.

Aplicação (Application): Coordena a execução das regras de negócios e responde às solicitações provenientes da camada de API.

API (Interface de Programação de Aplicativos): Fornecer uma interface para que os clientes interajam com a aplicação, processando solicitações HTTP e expondo serviços.

Questão 3:

Na camada de Domain criamos classes cujas propriedades são com set privado. Descreva, vantagem de usar dessa forma destacando como fizemos em sala com o produto.

R: Usar propriedades com métodos privados de definição (setters) em classes de domínio, como exemplificado com a classe Produto, proporciona um controle rigoroso sobre como os dados são acessados. Isso permite a aplicação de lógica de validação, previne efeitos colaterais indesejados, oferece flexibilidade para futuras modificações e facilita a criação de objetos imutáveis, assegurando, assim, a integridade dos dados.

Questão 4:

Na camada de applicattion na classe service (de serviço) fizemos o que chamamos de injeção de dependência, descreva por que utilizamos essa técnica e como isso pode ser uma vantagem?

R:

A injeção de dependência é uma técnica aplicada na camada de Aplicação com o propósito de desvincular as dependências de uma classe de serviço. Isso tem como objetivo aprimorar a testabilidade, fomentar a reutilização e aumentar a flexibilidade, tornando, assim, a manutenção do código mais fácil de realizar. Essa abordagem é amplamente adotada e adere aos princípios de design sólidos.