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= SRP (Single Responsibility Principle), que faz parte dos princípios SOLID de design de software. O SRP estabelece que uma classe deve ter apenas um motivo para mudar, ou seja, deve ter uma única responsabilidade.

O Princípio de Responsabilidade Única (SRP) afirma que uma classe deve ter apenas uma razão para mudar. Isso significa que uma classe deve ter uma única responsabilidade ou tarefa bem definida. Quando uma classe tem várias responsabilidades, ela se torna mais difícil de manter, testar e evoluir.

Para corrigir esse problema e aderir ao SRP, nós dividimos as responsabilidades em diferentes camadas ou classes especializadas. Separamos a logica de negocio das camadas de persistencia e apresentação. Usamos também a injeção de dependencia.

==========

Questão 2:
Começamos a dividir nosso projeto em camadas , são elas:
Domain, Data, Applicaion e API. Descreva qual o papel de cada uma delas.
R=

Domain: Esta camada é responsável por representar o núcleo do negócio. Ela contém as entidades principais, regras de negócio e lógica específica do domínio. É neste local que você modela os conceitos essenciais para o seu negócio, independentemente de como esses conceitos serão persistidos ou apresentados. A camada de domínio deve ser independente das tecnologias externas.

Data: A camada de dados é responsável por lidar com o armazenamento e recuperação de dados. Isso pode incluir acesso a banco de dados, leitura/gravação de arquivos ou integração com serviços externos. O objetivo é isolar toda a lógica de acesso a dados, garantindo que a camada de domínio não precise se preocupar com detalhes de armazenamento.

Application: Esta camada serve como uma camada intermédia entre as camadas de domínio e dados. Ela contém a lógica de aplicação que coordena a execução das regras de negócio e a manipulação dos dados. Aqui, você pode orquestrar diferentes componentes do sistema para atender às necessidades específicas da aplicação. A camada de aplicação também é um lugar comum para implementar serviços, como autenticação, autorização e transações.

API: A camada de API fornece uma interface externa para interação com o sistema. Pode ser uma API REST, GraphQL ou qualquer outro tipo de interface que permita a comunicação entre o seu sistema e outros sistemas, aplicativos ou usuários. Essa camada geralmente traduz solicitações externas em chamadas aos serviços da camada de aplicação e fornece as respostas adequadas.

==========

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= Basicamente a utilização de setters privados na camada de Domain traz benefícios significativos, como controle sobre a mutabilidade, validação centralizada, rastreabilidade de mudanças e a aplicação dos princípios de encapsulamento. Essas práticas contribuem para um código mais robusto, coeso e de fácil

manutenção.

==========

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 prática que promove desacoplamento, testabilidade, reusabilidade e flexibilidade no código, contribuindo para um design de software mais sustentável e de fácil manutenção.