1. Fale sobre os princípios SOLID e forneça um exemplo prático de como cada princípio pode ser aplicado em uma aplicação .NET?

R: Os princípios do SOLID consistem em um conjunto de orientações para desenvolver programas de forma simples e manter a organização do codigo. Esses princípios contribuem para a modularidade e legibilidade do codigo.

1-1. Princípio da Responsabilidade Única (SRP)

Princípio: Uma classe deve ter apenas uma responsabilidade.

1-2. Princípio Aberto/Fechado (OCP)

Princípio: O software deve estar aberto para extensão, mas fechado para modificação.

1-3. Princípio de substituição de Liskov (LSP)

Princípio: Os objetos de uma classe base deve ser substituídos por objetos de uma classe derivada sem alterar o comportamento correto do programa.

1-4. Princípio de compartilhamento de interface (ISP)

Princípio: Uma classe não deve ser forçada a implementar interfaces que não utiliza.

1-5. Princípio de Inversão de Dependência (DIP)

Princípio: Depende de abstrações, não de implementações concretas.

2. O que são Delegates em C# e como o tipo genérico Func pode ser utilizado. Forneça um exemplo de código onde um Func é utilizado para encapsular uma função anônima que calcula a soma de dois números?

R: O delegate é um tipo que representa referências a métodos com um tipo de parâmetro específico e retorna o um outro tipo específico de retorno.

Func<T1, T2, TResult>

3. Explique a diferença entre as classes Task e Thread no .NET. Quando usar uma sobre a outra? Forneça um exemplo prático de uso de Task.

R: thread consiste em conjunto pequenos de comandos em execução e o framework .NET disponibiliza classes específicas para threads no namespace System.Threading.

A task , elas podem ser usadas sempre que você quiser executar algo em paralelo. A implementação assíncrona é fácil de ser usada, usando as palavras chave "async" e "Task", e quando chamamos um outro serviço ou uma função que usa palavras chave "async" e "Task" usamos a palavra “await” na antes da função ou serviços.

Ex: public async Task<int> nome - com retorno

public async Task – sem retorno

4. O que é Dependency Injection? Explique como o .NET Core/6 implementa o padrão de injeção de dependência e forneça um exemplo de código

R: A injeção de dependência (DI) é um padrão de design partner muito utilizado, que faz uma classe ser independente das suas dependências. A implementação e muito simples, crie uma class que implementa uma outra class inteface, nesta interface teremos uma assinatura de um metodo ou função, que também vai implementa a class normal, com as suar funcionalidade. E configura na class "Program" e configura no contêiner de injeção de dependência.

EX: public class UsuarioCreateUsecases : IUsuarioCreateUsecases - Uma classe implementando uma irteface.

builder.Services.AddScoped<IUsuarioCreateUsecases, UsuarioCreateUsecases>(); configuração no contêiner na "Program"

```
public UsuarioController(
    IUsuarioCreateUsecases iUsuarioCreateUsecases,
    )
  {
     this.iUsuarioCreateUsecases = iUsuarioCreateUsecases;
      } -  resolvendo a depentendencia
```

5. Descreva o funcionamento do Entity Framework e explique as diferenças entre Code First, Database First e Model First. Qual a abordagem que você prefere e por quê?

R: Entity Framework e um ORM utilizado para conexão para vários banco de dados, que pode ser usada em três abordagem, descrita na pergunta acima "Code First, Database First e Model First".

Diferenças: Model First: ferramenta do visual (Entity Model Designer) para criar um diagrama de classes que servirá de base para a geração das classes e tabelas do banco de dados;

Code First: Criamos entidades e a partir delas o Entity Framework gera o banco de dados e suas respectivas tabelas.