Os princípios SOLID são um conjunto de cinco princípios de design de software que visam tornar o código mais compreensível, flexível e fácil de manter. Os princípios são:

* **S**ingle Responsibility Principle (Princípio da Responsabilidade Única)
* **O**pen/Closed Principle (Princípio do Aberto/Fechado)
* **L**iskov Substitution Principle (Princípio da Substituição de Liskov)
* **I**nterface Segregation Principle (Princípio da Segregação de Interface)
* **D**ependency Inversion Principle (Princípio da Inversão de Dependência)

##### Single Responsibility Principle (SRP)

Cada classe deve ter uma única responsabilidade ou razão para mudar.

##### Open/Closed Principle (OCP)

Classes devem estar abertas para extensão, mas fechadas para modificação.

**Liskov Substitution Principle (LSP)**

Objetos de uma classe base devem poder ser substituídos por objetos de uma classe derivada sem afetar a funcionalidade do programa.

##### Interface Segregation Principle (ISP)

Muitas interfaces específicas são melhores do que uma interface única e abrangente.

##### Dependency Inversion Principle (DIP)

Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações.

#### 2. Delegates e Func em C#

Delegates são tipos que representam referências a métodos com uma lista específica de parâmetros e tipo de retorno. Eles são usados para passar métodos como argumentos para outros métodos.

O tipo genérico Func é um delegate predefinido no .NET que pode encapsular um método que retorna um valor e recebe zero ou mais parâmetros.

#### 3. Task vs Thread no .NET

* **Thread**: Representa uma unidade de execução. Manipular diretamente threads pode ser complexo, especialmente quando se trata de sincronização e gerenciamento de recursos.
* **Task**: Introduzido no .NET Framework 4.0, Task representa uma operação assíncrona que pode ser executada de forma paralela. Task é mais eficiente e fácil de usar para operações assíncronas e paralelas do que Thread.

#### 4. Dependency Injection no .NET Core/6

Dependency Injection (DI) é um padrão de design que permite a injeção de dependências em uma classe em vez de a própria classe criar as dependências.

O .NET Core/6 tem suporte nativo para DI através do IServiceCollection e IServiceProvider.

#### 5. Entity Framework: Code First, Database First e Model First

O Entity Framework (EF) é uma ferramenta ORM (Object-Relational Mapper) que permite que os desenvolvedores trabalhem com um banco de dados usando objetos .NET.

* **Code First**: O modelo é definido através de classes C# e o banco de dados é gerado a partir dessas classes.
* **Database First**: O modelo é gerado a partir de um banco de dados existente.
* **Model First**: Um modelo é criado usando o designer do Entity Framework, e o banco de dados é gerado a partir desse modelo.