**COOPER VISION DOCUMENTATION**

**SOLID PRINCIPLES IN C#:**

**SOLID PRINCIPLES:**

It is a basic design principles in c#.

**Why We need this:**

It helps to maintain and debug the code easily. The solid principles provides the idea about how to design the software in a better way. If we had certain principles for designing the software then it is easy to debug.

**Definition:**

SOLID is an acronym for the following.

* S: Single Responsibility Principle (SRP)
* O: Open-closed Principle (OCP)
* L: Liskov substitution Principle (LSP)
* I: Interface Segregation Principle (ISP)
* D: Dependency Inversion Principle (DIP)

**1.S:Single Responsibility Principle(SRP):**

This means that every class in a code should have only one job.Putting too much of job’s in a single class lead’s to difficult in bug identification.

**2. O:Open-Closed Principle(OCP):**

The Open-Closed Principle emphasizes that a class should be open for extension but closed for modification.

Key points:

* Allows adding new functionality without changing existing code
* Encourages the use of inheritance and polymorphism

**3.Liskov Substitution Principle (LSP)**

The Liskov Substitution Principle states that objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program.

Key points:

* Ensures inheritance hierarchies make sense
* Promotes proper use of polymorphism

**4.Interface Segregation Principle (ISP)**

The Interface Segregation Principle states that a client should not be forced to implement interfaces it doesn't use .

Key points:

* Promotes smaller, client-specific interfaces.
* Reduces coupling between classes.
* Don’t force to implement more methods in a single interface

**5.Dependency Inversion Principle (DIP)**

The Dependency Inversion Principle advocates that high-level modules should not depend on low-level modules, but both should depend on abstractions.

Key points:

* Encourages loose coupling between components
* Makes code more flexible and easier to test

**CLEAN ARCHITECTURE**

**What is Clean Architecture?**

Clean Architecture is a software design approach that aims to separate an application into distinct layers based on responsibility.

**Key points about Clean Architecture:**

* Emphasizes separation of concerns
* Creates independent, testable components
* Allows swapping out infrastructure/UI implementations
* Makes core business logic independent of external factors

**It has four layers:**

1. **Domain Layer**: Contains core business objects and logic.
2. **Application Layer**: Implements use cases and interfaces.
3. **Infrastructure Layer**: Handles data access, logging, etc.
4. **Presentation Layer**: Contains controllers and API endpoints.

**1.Domain Layer**

The Domain Layer contains the core business objects and logic.Like variable decalarations.

Eg code:

public class User

{

public int Id { get; set; }

public string Name { get; set; }

public string Email { get; set; }

}

**2.Application Layer**

The Application Layer acts as an intermediary between the Domain Layer and the Presentation Layer. It contains use cases and interfaces that define how the domain objects will be manipulated

**Key characteristics:**

* Defines use cases and interfaces
* Implements business logic
* Acts as an abstraction layer between Domain and Infrastructure layers
* Does not depend on infrastructure or presentation layers

**3.Infrastructure Layer**

The Infrastructure Layer handles data access, logging, and other external interactions. It implements interfaces defined in the Application Layer [**2**](https://positiwise.com/blog/clean-architecture-net-core).

Key characteristics:

* Implements data access logic
* Handles external dependencies
* Provides concrete implementations of repositories and services
* Does not depend on Domain or Application layers

**4.Presentation Layer**

The Presentation Layer contains the UI components and API controllers. It depends on the Application layer through interfaces .

Key characteristics:

* Contains UI components and API endpoints
* Depends on Application layer through interfaces
* Should not depend on Infrastructure layer
* Handles incoming requests and sends responses