**SOLID DESIGN PRINCIPLES – REQUIRED FOR BACKEND CODE**

**S –** Single Responsibility Principle

**O –** Open-Closed Principle

**L –** Liskov Substitution Principle

**I –** Interface Segregation Principle

**D –** Dependency Inversion Principle

Solid Design is required for every programmer to know. Although this principle can be hard and complex as well as may take a lot of time to implement but these efforts are worth it. This will help increasing the code readability, reusability, and will be easier for maintenance.

**SINGLE RESPONSIBILITY PRINCIPLE**

This principle states that each and every class should have only one task to do. Each class should handle only one responsibility. So, if we have 100 different tasks at hand, then for each of the 100 tasks, we have to create 100 different classes to handle it.

**OPEN-CLOSED PRINCIPLE**

This principle states that each and every class should be created in such a way that it’s behavior should be extendable without modifying the contents. Open means open for extension and Closed means closed for modification.

**LISKOV SUBSTITUTION PRINCIPLE**

This principle ensures that derived classes should be able to extend the base class without changing behavior. Following this principle helps us to avoid unexpected consequences of changes and avoids having to open a closed class in order to make changes.

**INTERFACE SEGREGATION PRINCIPLE**

This principle states that it is better to have many smaller interfaces rather than having few larger interfaces. Smaller interfaces mean developers should have a preference for composition over inheritance and for decoupling over coupling.

**DEPENDENCY INVERSION PRINCIPLE**

This principle states that abstraction should not depend upon details, but details should depend upon abstraction. High level modules should not depend on low level modules but low level modules should depend upon high level modules.