# Summary for the videos

Clean architecture known formerly by names like onion, hexagonal architecture; is a domain centric approach to organizing dependencies. It is where concerns related to the dependencies on infrastructure is minimized. It is mainly focused on domain logic rather than architecture. Its complex business logic gives a highly testable architecture.

Basically there are two approaches into architecture. These are N-tier architecture and Clean Architecture. The N-tier architecture is the all in one architecture layered approach while the Clean Architecture is a domain logic focused layer independent architecture to most extent. But generally, there are 3 types of architectural styles. Each are listed below with their pros and cons.

* All in one architecture
* Easy and stable but hard to test and maintain.
* Layered Architecture
* Enforces solid principles and maintainable but dependent layers and logic is scattered
* Onion architecture
* Easily testable and maintainable but time consuming and harder to learn

The all in one architecture does not work for the extendable projects and customers deem a software good as long as it works while developers look more at structure and logic of the software to categorize it as good software. That is why Clean Architecture is superior to the other Architecture styles. But also it is vital to remember that Clean Architecture is not suited for small projects that are not bounded to maintainability and good logic as it will be waste of time and resources.

There are rules to be followed when implementing the clean architecture. They are discussed as follows.

* Model all business rules and entities in the core project
* All dependencies flow toward the core project
* Inner projects define interfaces; outer projects implement theme

There are two ways you can implement the clean architecture. The first one is using the provided Ardalis.CleanArchitecture.Template on NuGet package manager. It has already defined examples to get developers familiar with the usage. The other alternative is to create a blank solution and start from scratch populating the solution folder with required projects and folders.

Setting up the project solution folder using the latter method can be achieved by following this instructions.

* Create a blank solution and inside the solution create folders namely Core, API, infrastructure and UI.
* Create a new project called application inside the core folder. And inside the application project create new folder called persistence
* Inside persistence folder, create folder named Contracts. Inside Contracts Folder create a generic interface with Type T of class as repository with CRUD tasks. Then create repository interfaces that implement the generic interface with their class type.