SmartPOD **Architecture**

# **Why Use Clean Architecture**

A modern, clean architecture template for .NET applications following domain-driven design principles and best practices. This template provides a solid foundation for building scalable, maintainable, and testable applications.

# **Architecture Overview**

## Core

### **Contains Enterprise Business Rules**

Domain Entities

Interfaces/Abstractions

Application Logic (Use Cases)

DTOs and Enums

**NOTE:** This layer **should not depend** on any other project. **Other layers depend on Core**

### **Infrastructure**

### **Implements Infrastructure Concerns**

Database Access (EF Core, Dapper)

File systems, external APIs

Repositories that implement interfaces from Core

**NOTE**: Depends on Core

### **Presentation**

**User Interface Layer**

ASP.NET Core Web API or MVC project

Controllers

Input/Output Models

**NOTE**: **Depends on Core and possibly Infrastructure** (but can be decoupled via Dependency Injection).

### **Shared**

**Cross-cutting Concerns**

Common utilities

Validation

Logging

Shared constants

**NOTE**: **Accessible by all other layers** without introducing tight coupling

### **Typical Implementation Flow**

Example user registration:

1. **Presentation** calls IUserService (defined in Core).
2. **Infrastructure** provides UserRepository implementing IUserRepository.
3. IUserService orchestrates domain logic.
4. Entities and DTOs defined in Core.
5. Results passed back to Presentation.

|  |
| --- |
|  |

|  |
| --- |
| C:\Users\Himanshu.Mehta.SDEICHD0\Downloads\mermaid.png |

### **Project Structure**

|  |
| --- |
| ├── Core  │ ├── App.Application # Application logic, use cases, DTOs, interfaces  │ └── App.Domain # Domain entities, business rules, domain interfaces  │  ├── Infrastructure  │ └── App.Infrastructure # Implementation of services, repositories, database access  │  ├── Presentation  │ └── App.Api # ASP.NET Core Web API project  │ ├── Controllers # API endpoints  │ ├── Middleware # Custom middleware (e.g., error handling, logging)  │ ├── Models # Request/response models  │ ├── Validation # Validation logic (FluentValidation, etc.)  │ ├── Program.cs # Entry point  │ ├── appsettings.json  │  ├── Shared  │ └── App.Common # Shared/common utilities, constants, helpers, extensions  │  └── Tests  └── Tests # Unit and integration test projec |

**Layer Details**

## App.Application

### **Purpose** This is the **Application Layer**

Use Cases / Application Services

* Classes that orchestrate workflows (e.g., UserService)
* They coordinate repositories and domain entities to fulfill specific operations (like “RegisterUser” or “UpdateUserStatus”)

DTO

* Data Transfer Objects to move data in/out of the application layer

Interfaces

* Contracts for repositories and services (e.g., IUserRepository, IUserService)

NOTE:

**Does Not:**

* Contain infrastructure details (like Entity Framework)
* Contain persistence logic

**Depends on:**

* **Domain Layer** (because it uses Entities and Enums defined there)

### **App.Domain**

### **Purpose** This is the **Application Layer**

Entities

* Domain models (e.g., User, Order)

Value Objects

* Immutable types (e.g., Money, EmailAddress)

Enum

* Enumerations like UserStatus, UserRole

Domain Logic

* Business rules embedded in entities (e.g., validations, invariants)

NOTE:

**Does Not:**

* Know anything about infrastructure
* Depend on Application Layer or anything else

**This is the purest layer of your system.**

|  |
| --- |
| In Clean Architecture:   * **Domain Layer** (App.Domain): the **innermost circle**—pure business rules * **Application Layer** (App.Application): **use cases**, which depend on Domain |

## App.Infrastructure

### **Purpose**

### This is the **Infrastructure Layer**

It contains all the **technical details** and **implementation classes** that connect your application to the outside world,

**DBContext**

This folder holds everything related to Entity Framework Core

**File Name: ApplicationDbContext.cs**

* Your EF Core DbContext.
* Defines DbSet<TEntity> properties to map entities to database tables.
* Configures model relationships.

**File Name: IUnitOfWork.cs**

* Concrete implementation of IUnitOfWork.
* Provides access to repositories and manages transaction scope.

**File Name: UnitOfWork.cs**

An interface defining a **Unit of Work pattern**, which:

* Groups multiple repository operations in a single transaction.
* Ensures all changes are committed atomically.

**Repository**

Implement data access pattern

**File Name: BaseRepository.cs**

* A generic repository with common CRUD methods.
* Typically operates on TEntity

**File Name: UserRepository.cs**

* Concrete implementation of IUserRepository interface (defined in Application layer).
* Contains User-specific queries and commands**.**

**Service**

**File Name: UserService.cs**

A technical service implementing IUserService (if you have it in Application layer).

Could contain business logic that requires infrastructure (e.g., sending emails, hashing passwords, background tasks).

NOTE:

**Does Not:**

* Contain infrastructure details (like Entity Framework)
* Contain persistence logic

**Depends on:**

* **Domain Layer** (because it uses Entities and Enums defined there)

### **App.Api**

### **Purpose** This is the **Presentation Layer**

This is your **API layer**—the part of the system responsible for

* Handling HTTP requests/responses
* Defining REST endpoints
* Configuring middleware
* Bootstrapping the application

**Controller**

Defines the endpoints for interacting with your system

**File Name: BaseController.cs**

* Typically an **abstract base class** all controllers inherit.
* Common features:
  + Access to the current user identity
  + Standardized response formatting
  + Error handling helper

**File Name: UserController.cs**

* Concrete controller exposing **User-related API endpoints**.
* Likely has methods such as:
  + GetUserById
  + CreateUser
  + UpdateUser

**Middleware**

Custom pipeline logic applied to all requests

**File Name: Program.cs**

* The **application entry point**.
* Configures:
  + Dependency Injection
  + Routing
  + Middleware
  + Application startup logic

**File Name: appsettings.json / appsettings.Development.json**

* Configuration files.
* Contain:
  + Connection strings
  + Logging settings
  + API keys
  + Other environment-specific settings

**File Name: launchSettings.json**

Defines how the API is launched locally (e.g., ports, profiles).