|  |  |
| --- | --- |
| Clean Architecture | Microsoft Word logo |

# **Clean Architecture**is an **architecture pattern** aimed to divide a big project into pieces (classes, entities, modules), and makes them independent of each other. This makes the project more maintainable, flexible and easy to work on together.

# Layers of Clean Architecture

1. **Domain Layer**: Contains the core business logic, entities, and domain-specific rules. This layer is independent of external concerns like databases or UI.

**Things you typically see in the Domain layer**

* Entities
* Value objects >> immutable objects (Records)
* Domain Events
* Domain Services
* Interfaces
* Exceptions
* Enums



1. **Application Layer**: Responsible for orchestrating domain logic. This means that the Application Layer coordinates the execution of business rules and operations defined in the Domain Layer. It acts as a mediator between the user interface (or other external clients) and the domain model, ensuring that domain entities and services are used correctly to achieve business goals.

### Key Responsibilities of the Application Layer

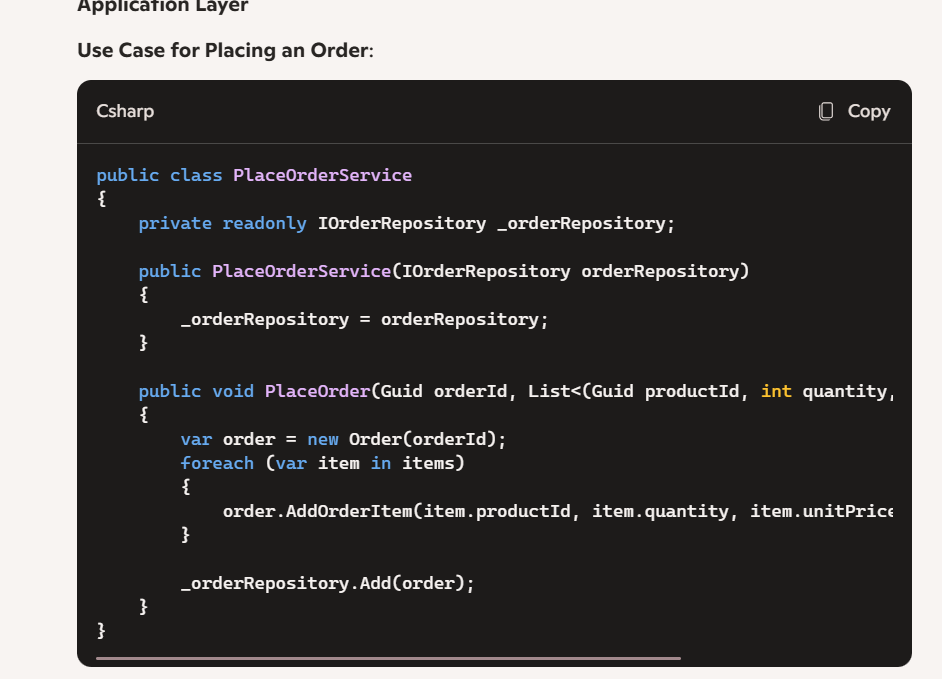
1. **Use Cases**: It defines and implements application-specific use cases or services that orchestrate the domain model's operations to fulfill user requests or system events. This is usually implemented using the CQRS Pattern
2. **Interactors/Services**: These are the core classes that contain the application logic, coordinating between different domain entities and services.
3. **Transaction Management**: The Application Layer can manage transactions, ensuring that multiple domain operations either complete successfully or roll back together.

Application Layer Contains:

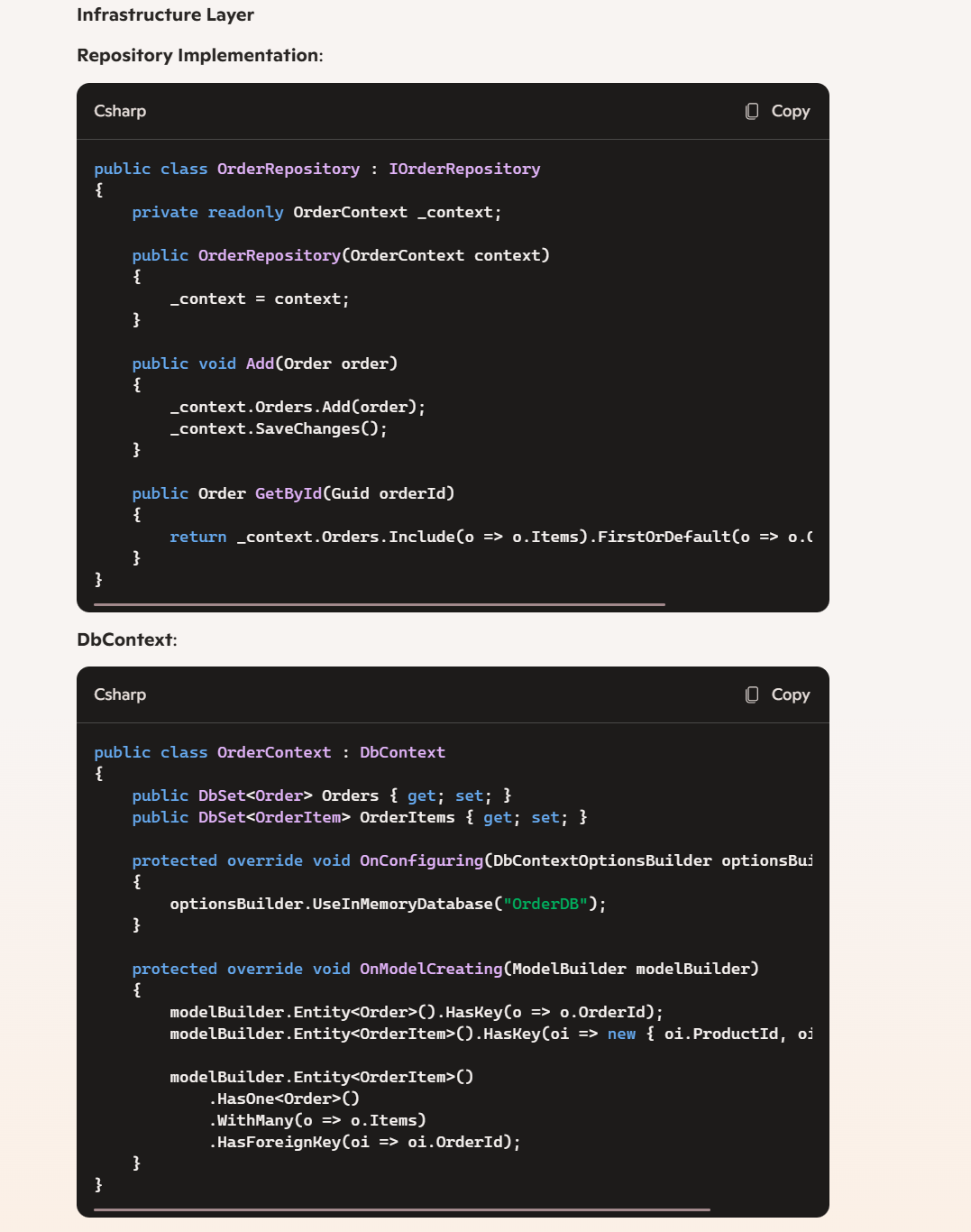
* CQRS
* Logging
* Validation
* Exceptions
* DI Configuration

### Example in C#

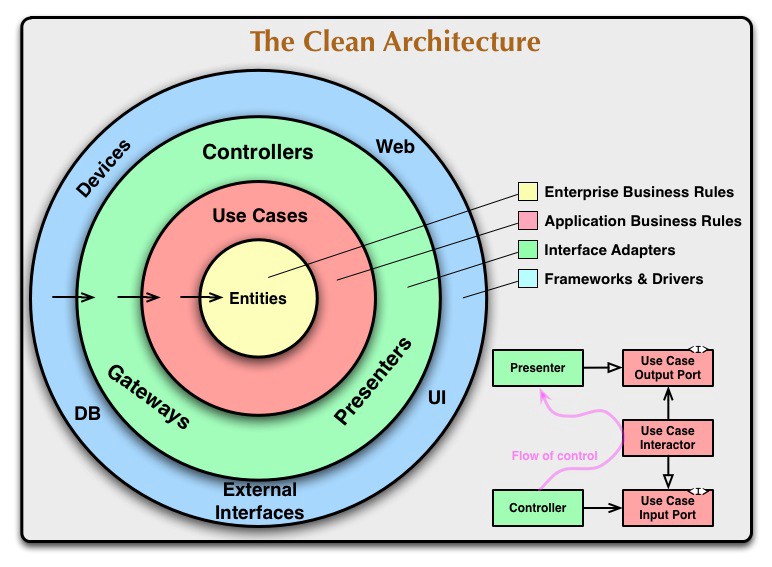
Let's consider an example where we have an online store with a use case for placing an order. The Application Layer will orchestrate the domain logic to handle this operation.

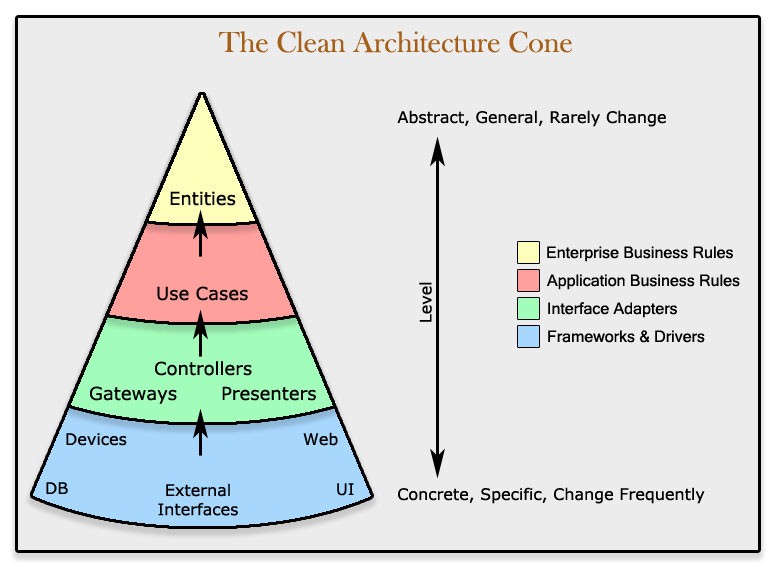


1. **Infrastructure Layer**: Deals with external concerns like databases, file systems, APIs, etc. It implements data access (repositories), external services, and other infrastructure-specific details.



1. **Presentation Layer**: Contains the user interface components (e.g., MVC, API Controllers, Blazor components). It communicates with the application layer and presents data to the user1.

[https://www.freecodecamp.org/news/a-quick-introduction-to-clean-architecture-990c014448d2/](https://www.freecodecamp.org/news/a-quick-introduction-to-clean-architecture-990c014448d2/)



Why Should you use it?

* Domain Driven Design
* Complex Business Logic
* Highly testable projects because you are using SRP
* Architecture can help enforce design policies

# 

# Project Folder Structure

* **Chinook**
* **├── src**
* **│ ├── Chinook.Core**
* **│ │ ├── Domain**
* **│ │ │ ├── Entities**
* **│ │ │ ├── ValueObjects**
* **│ │ │ └── Aggregates**
* **│ │ ├── Interfaces**
* **│ │ │ ├── Repositories**
* **│ │ │ └── Services**
* **│ │ ├── UseCases**
* **│ │ │ ├── Commands**
* **│ │ │ │ ├── Create**
* **│ │ │ │ └── Update**
* **│ │ │ ├── Queries**
* **│ │ │ │ ├── Get**
* **│ │ │ │ └── List**
* **│ │ │ └── Handlers**
* **│ │ └── Mappers**
* **│ │ └── MappingProfiles**
* **│ ├── Chinook.Infrastructure**
* **│ │ ├── Data**
* **│ │ │ ├── Migrations**
* **│ │ │ └── Seed**
* **│ │ ├── Repositories**
* **│ │ ├── Services**
* **│ │ └── Configuration**
* **│ ├── Chinook.Application**
* **│ │ ├── Dtos**
* **│ │ ├── Services**
* **│ │ ├── Validators**
* **│ │ └── EventHandlers**
* **│ └── Chinook.Web**
* **│ ├── Controllers**
* **│ ├── Views**
* **│ ├── wwwroot**
* **│ ├── Program.cs**
* **│ └── Startup.cs**
* **├── tests**
* **│ ├── Chinook.UnitTests**
* **│ │ ├── Domain**
* **│ │ ├── Application**
* **│ │ ├── Infrastructure**
* **│ │ └── Web**
* **│ └── Chinook.IntegrationTests**
* **│ ├── API**
* **│ ├── Repositories**
* **│ └── Services**
* **└── docs**
* **├── Architecture**
* **├── Requirements**
* **└── UserGuide**

# Explanation:

* **Chinook.Core**: This contains the core business logic, domain entities, value objects, aggregates, and interfaces.
  + **Domain**: Business entities, value objects, aggregates.
  + **Interfaces**: Repository and service interfaces.
  + **UseCases**: Commands, queries, and their handlers.
  + **Mappers**: Mapping profiles for DTOs and domain objects.
* **Chinook.Infrastructure**: Handles data access, repositories, external services, and configurations.
  + **Data**: Database context, migrations, seeding data.
  + **Repositories**: Implementations of repository interfaces.
  + **Services**: Implementations of external services.
  + **Configuration**: Infrastructure configurations.
* **Chinook.Application**: Contains application-level services, DTOs, validators, and event handlers.
  + **Dtos**: Data transfer objects.
  + **Services**: Application-specific services.
  + **Validators**: Validation logic for DTOs.
  + **EventHandlers**: Handles application events.
* **Chinook.Web**: The presentation layer with controllers, views, and static files.
  + **Controllers**: API or MVC controllers.
  + **Views**: Razor views (if using MVC).
  + **wwwroot**: Static files like CSS, JS, images.
  + **Program.cs and Startup.cs**: Setup and configuration of the ASP.NET Core application.
* **tests**: Unit and integration tests for different layers.
  + **Chinook.UnitTests**: Unit tests for domain, application, infrastructure, and web.
  + **Chinook.IntegrationTests**: Integration tests for APIs, repositories, and services.
* **docs**: Documentation related to the project architecture, requirements, and user guides.

# 