**What is Ntier architecture**

**N-Tier Architecture** (also known as multi-tier architecture) is a software design pattern that organizes an application into multiple logical layers or tiers, each responsible for a specific set of functionalities. The "N" in N-Tier refers to the number of tiers, which can vary depending on the complexity of the system, but the most common implementation is the 3-Tier Architecture (Presentation, Business Logic, and Data Access). It’s widely used in backend development, including .NET projects, to achieve separation of concerns, scalability, and maintainability.

**Core Concept**

In N-Tier architecture, the system is divided into tiers that are logically and often physically separated. Each tier interacts only with the tier directly above or below it, promoting modularity and reducing tight coupling. These tiers typically run on different components (e.g., servers, databases) or within the same process, depending on the deployment model.

**Common Tiers in N-Tier Architecture**

**1. Presentation Tier (UI Layer)**

Purpose: Handles user interaction and displays data to the user.

Responsibilities:

Rendering UI (e.g., web pages, desktop forms, mobile screens).

Capturing user input and sending it to the next tier.

**Technologies in .NET:**

ASP.NET Core MVC (web apps).

Razor Pages.

WPF or WinForms (desktop apps).

Blazor (web or hybrid).

Example: A web form where a user enters their order details in an e-commerce app.

**2. Business Logic Tier (Application Layer)**

Purpose: Contains the core functionality and rules of the application.

Responsibilities:

Processing business rules (e.g., calculating discounts, validating data).

Coordinating workflows between the presentation and data tiers.

Enforcing application logic (e.g., user permissions, transaction management).

Technologies in .NET:

C# classes and services.

Dependency Injection (e.g., in ASP.NET Core).

Domain-Driven Design (DDD) entities or use cases.

Example: A service that calculates the total price of an order, including taxes and discounts.

**3. Data Tier (Data Access Layer)**

**Purpose**: Manages data storage, retrieval, and persistence.

Responsibilities:

Interacting with databases, file systems, or external APIs.

Performing CRUD operations (Create, Read, Update, Delete).

Ensuring data integrity and consistency.

**Technologies in .NET:**

Entity Framework Core (ORM).

ADO.NET (raw database access).

Dapper (lightweight ORM).

**Advantages of N-Tier Architecture**

* **Separation of Concerns**: Each tier focuses on its own responsibility, improving code organization.
* **Scalability**: Tiers can be scaled independently (e.g., add more database servers without touching the UI).
* **Maintainability**: Changes in one tier (e.g., switching from SQL Server to PostgreSQL) don’t affect others if interfaces remain consistent.
* **Reusability**: Business logic or data access components can be reused across different presentation layers (e.g., web and mobile).
* **Testability**: Isolated tiers are easier to unit test (e.g., mock the data tier to test business logic).

**Disadvantages of N-Tier Architecture**

* **Complexity**: Adds overhead, especially for small projects where a simpler design might suffice.
* **Performance**: Inter-tier communication (e.g., across servers) can introduce latency, especially in distributed systems.
* **Development Time**: Requires more upfront planning and design compared to a monolithic approach.