Three-Tier System Architecture

>Data Tier (Database): Manages storage and updates of the application's data in a database.

>Application Server Tier (Logic): Processes the business logic. Acts as the intermediary between the User Tier and the Data Tier.

>User Tier (Presentation): The front-end interface for users to interact with the system.

---

application server (tier) layered architecture

> Overview : design model that separates a system into distinct layers:

- **Infrastructure Layer**
- **Data Access Layer (DAL)**
- **Business Logic Layer (BLL)**
- **Presentation Layer**

> Definition:

- **Infrastructure Layer: Logs transactions, authenticates**

users, and handles API configurations.

- DAL: Handles database operations such as queries, inserts, updates, and deletes (Repositories).

- BLL: Processes the core application logic, validates data, and applies business rules (Services)

- Presentation Tier: Handles user interface and interactions(Controllers)

> Importance:

- Scalability: Each layer can scale independently to handle increased loads.

- Maintainability: Changes in one layer have minimal impact on others.

- Reusability: Components in one layer can be reused across multiple applications.

- Security: Better isolation.

Data transfer objects (DTO)

> Overview : simple objects designed to transfer data between layers of an application without exposing internal entities.

> Purpose:To encapsulate data and move it between application

layers (e.g., from Business Logic Layer to Presentation Layer).

> Importance:

- **Separation of Concerns: Keeps the database entities isolated from presentation logic.**
- **Improves security by hiding internal entity details.**
- **Facilitates loose coupling between layers.**
- **Optimizes data transfer by sending only necessary information.**

Repository pattern in .net and generic repo

> Overview : design pattern that provides a centralized interface for interacting with a data source

- **Repository Pattern: A pattern that creates a layer to isolate the application's business logic from the data access logic by using an interface to perform CRUD operations.**
- **Generic Repository: A reusable repository that operates on a specific type and can be reused across multiple entities.**

> Importance:

- **Separation of Concerns.**
- **Testability :Makes unit testing easier by mocking repository**

interfaces.

- Code Reusability

- Flexibility: Provides a single, consistent API for accessing data from multiple sources.

---

unit of work pattern in .net

> Overview : Combines repository operations and manages the database context's lifecycle.

> Definition:

- Acts as a wrapper around the database context.

- Groups multiple operations (e.g., Create, Update, Delete) into a single transaction.

- Ensures that either all operations succeed or none are applied (atomicity).

> Importance:

- Transaction Scope: Groups multiple repository operations under a single transaction.

- Code Organization: Simplifies business logic by delegating data access and transaction management to Unit of Work.

- Decoupling: Abstracts the database context from the business

layer.

- Flexibility: Makes the data access layer reusable and testable.

---

dependency inversion and dependency injection and object lifecycle

> Dependency Inversion (DIP) : A design principle where high-level modules and low-level modules depend on abstractions rather than each other.

- Abstractions > Depend on interfaces, not concrete implementations.

> Dependency Injection (DI): A technique where dependencies are provided externally to an object rather than being created within the object.

- Supply objects with dependencies instead of hard-coding them.

> Object Lifecycle: The stages an object goes through: creation, initialization, usage, and destruction.

- Ensure objects are created, initialized, and destroyed systematically and in sync with the application's needs.

Middleware in .net and middleware pipeline

> Overview : Middleware in .NET is a component in the request-response pipeline of an application. Each middleware processes incoming HTTP requests and decides whether to pass the request to the next middleware or handle it itself.

> Definition:

- Middleware: A function or component that handles HTTP requests and responses.

- Middleware Pipeline: A sequence of middleware components that are executed in a specific order for each HTTP request.

> Importance:

- Middleware components can validate, authenticate, or modify requests

- Response Processing: Middleware can modify or log responses before they are sent to the client.

- Extensibility: Developers can easily add or remove functionality by modifying the pipeline.

app setting .json

> Overview : The appsettings.json file is a configuration file used in ASP.NET Core applications to store key-value pairs for settings. It's a common way to manage application settings such as connection strings, logging configurations, or any custom configuration data.

- In an ASP.NET Core application, the configuration is loaded automatically during startup

> Importance:

flexible and powerful for managing application configurations.

Rest Api and HTTP protocol

> Overview :

- REST API (Representational State Transfer):

architectural style for building web services. REST uses standard HTTP methods for communication and relies on stateless interactions between clients and servers.

- HTTP Protocol (Hypertext Transfer Protocol):

The underlying protocol for sending and receiving requests between clients and servers, supporting the methods and

status codes used in RESTful APIs.

> HTTP Methods:

GET: Retrieve information from the server (e.g., get a list of users).

POST: Send new data to the server (e.g., create a new user).

PUT: Update existing data on the server (e.g., update a user's details).

DELETE: Remove data from the server (e.g., delete a user)

> REST API and HTTP Protocol Relationship

- REST defines how to structure the interactions (i.e., requests and responses) between the client and server, typically using HTTP as the communication protocol.

- HTTP provides the foundational methods (GET, POST, PUT, DELETE, etc.) for these interactions..

---

swagger API documentation

> Overview : A web-based interface that displays API documentation in an interactive and readable format.

> Importance:

- **Developers can test API endpoints directly from the Swagger UI.**
- **Ensures APIs are documented consistently.**
- **Can be automatically integrated with many programming languages and frameworks.**
- **Helps teams understand and interact with APIs efficiently.**

> Steps to Use Swagger:

1. **Add Swagger to the Project**
2. **Configure Swagger in Program.cs**
3. **Modify the SwaggerGen configuration**
4. **Define API Controllers**

> How Swagger Works:

1. **Swagger automatically scans controllers and their routes to discover endpoints.**
2. **Based on controller annotations, Swagger generates the API documentation.**
3. **Swagger UI allows users to test endpoints with live data.**

Routing in .net

> Overview : Routing in ASP.NET Core is a system that maps

incoming HTTP requests to specific endpoints in the application,

such as controllers. It also provides mechanisms to generate URLs dynamically.

> How Routing Works

1. Request Matching: Routing evaluates incoming HTTP requests and matches them to defined routes.

2. Route Execution: Once matched, the request is forwarded to the corresponding controller/action or middleware

3.URL Generation: Routing helps construct URLs based on route definitions for navigation or API endpoints.

Model binding and validation

> Overview : Model Binding and Validation are key features in ASP.NET Core that simplify the process of handling incoming data from HTTP requests and validating it before further processing.

> Model Binding : is the process of mapping incoming HTTP request data to action method parameters or objects.

How Model Binding Works

1. Request Data

2. Binding

3.Conversion

> **Model Validation:** checks if the incoming data meets the defined rules and constraints

Validation Process

1. Automatic Validation

2.Check

3.Handle Validation Errors