ASP.NET Core MVC - Detailed Notes for Students

# 1. MVC (Model-View-Controller)

Definition: MVC is a design pattern that separates the application into 3 main parts:  
- Model: Data and business logic  
- View: What the user sees  
- Controller: Handles user input and connects Model + View

Real-Life Example: Online food delivery app  
- Model = Menu items, prices  
- View = App interface with food list  
- Controller = Handles clicks like 'Order Pizza' and fetches data

Why use it?  
- Code is organized  
- Easy to maintain  
- Separation of concerns

# 2. AutoMapper

Definition: AutoMapper copies data from one object to another automatically when property names are the same.

Why? In real apps, we use DTOs (simplified objects) instead of full Models to avoid sending unnecessary data.

**Real-Life Example**:  
You are a teacher. Your school database has full details of students:  
Name, Age, Address, Phone, Parent Info, Marks, etc.  
But to publish a topper list on the website, you only need:  
Name + Marks.

Instead of writing code like:  
dto.Name = student.Name; dto.Marks = student.Marks,  
AutoMapper does this **automatically**.

Benefit:  
- Saves time  
- Reduces mistakes  
- Makes code short and clean

# 3. Dependency Injection (DI)

Definition: DI provides required objects to a class from outside instead of letting the class create it.

Real-Life Example: A student uses a calculator given by the teacher instead of building one. Similarly, classes get objects provided.

Why use it?  
- Easy to test code  
- Easy to change logic  
- Loose coupling (less dependency)

# 4. Repository Pattern

**Definition**:  
It’s a pattern where we keep all **database code** (Insert, Update, Delete, Get) inside a **separate class called Repository**.

**Real-Life Example**:  
Imagine a restaurant:

* Waiter takes your order (Controller).
* Chef cooks food (Service).
* Inventory manager checks ingredients and tools (Repository).

**Why use Repository?**

* Keeps code **clean and reusable**.
* Database logic is in one place.
* Easy to change DB without affecting other parts.

# 5. Service Layer

**Definition**:  
A Service Layer contains **business logic** – rules or decisions of the application. It sits between the **Controller** and **Repository**.

**Real-Life Example**:  
You apply for a loan:

* Customer (User) fills the form (View).
* Officer (Controller) sends request.
* Bank manager (Service) checks if the customer is eligible (has business logic).
* Then asks staff (Repository) to access the customer database.

**Why use Service Layer?**

* Keeps Controller simple (only sends/receives).
* Puts business rules in one place.
* Makes app easier to test and modify.

# Summary Flow (Real-World Analogy)

View (User fills form)

↓

Controller (Receives request)

↓

Service (Applies logic: Check eligibility, calculate tax)

↓

Repository (Gets/Saves data in database)

↓

Database