**🡺CLS (Common Language Specification)** is a subset of CTS. It defines a set of base rules and standards that all .NET languages must follow to ensure interoperability.

This means a class written in C# can be used in VB.NET if it follows CLS rules.

**CLR** runs the code, **CTS** defines type system, **FCL** is the library.

🡺

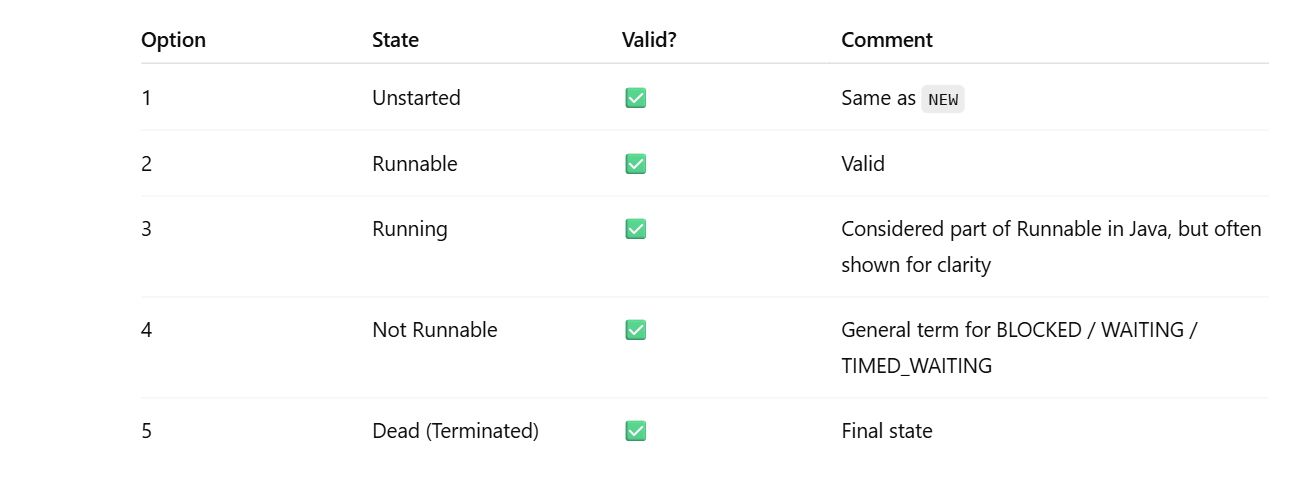
SqlConnection conn = new SqlConnection("Data Source=(local);Initial Catalog=Northwind; Integrated Security=SSPI");

🡺

# DotNet

* In C#, the keyword used to **overload user-defined types** (such as operator overloading for classes or structs) is:

**✅ operator**



SqlConnection in .net

SqlConnection conn = new SqlConnection("Data Source=local;Initial Catalog=Northwind;Integrated Security=SSPI");

# Introduction to ASP.NET MVC Core

**🔍 1. MVC Architecture**

**command creates a new ASP.NET MVC project in CLI🡺dotnet new mvc**

**the default port for ASP.NET Core dev server (Kestrel)🡺5000**

| **Component** | **Role** |
| --- | --- |
| **Model** | Contains business logic and application data |
| **View** | Displays the UI using Razor syntax |
| **Controller** | Handles user input, interacts with model, and selects a view |

**Model (The Data):** The Model is responsible for the data and business logic of your application. It knows how to retrieve data from a database, store it, and perform any calculations on it

**Controller (The Traffic Cop):** The Controller is the "brains" of the operation. It receives user requests (like a URL you type in your browser), decides which Model to use, and then chooses the correct View to display the result. The Controller orchestrates the entire process.**The Request Life Cycle**

When a user requests a page, here's the typical flow in an MVC application:

1. A request arrives at the **Controller**.
2. The Controller asks the **Model** to get or update some data.
3. The Model performs the necessary actions (e.g., fetching a list of products from a database) and returns the data to the Controller.
4. The Controller then selects an appropriate **View** and sends the data from the Model to it.
5. The View renders the data into HTML and sends it back to the user's browser as the final response.

**🔧 2. Folder Structure Overview (ASP.NET Core MVC)**

| **Folder/File** | **Purpose** |
| --- | --- |
| /Controllers/ | Contains controller classes like HomeController.cs |
| /Models/ | Data models or view models conatin business Logic&Application's data |
| /Views/ | Razor pages for UI (.cshtml) |
| /Views/Shared/ | Shared UI components like layout |
| Startup.cs | Middleware and routing config (older template) |
| Program.cs | Main entry point (new template in .NET 6/7/8) |
| appsettings.json | App configuration like DB connection strings |
| /wwwroot/ | Static files (CSS, JS, Images) |

**💡 3. Configuration Files in .NET Core**

* appsettings.json → type of file is .Json

Stores:

* + Connection strings
  + Logging settings
  + Custom config values
  + Databse Connection string

{

"ConnectionStrings": {

"DefaultConnection": "Server=.;Database=MyAppDb;Trusted\_Connection=True;"

}}

* Program.cs (ASP.NET Core 6+):

//used to buid the application in Program.cs

var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

Razor syntax to render a variable is 🡺@model.Name

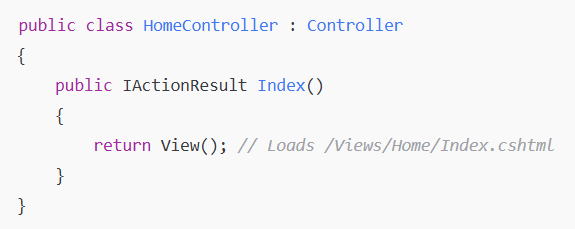
# Controllers and Actions in ASP.NET Core MVC

**1. What is a Controller?**

A **Controller** is a C# class that:

A Controller class must inherit from the **Microsoft.AspNetCore.Mvc.Controller** base class.

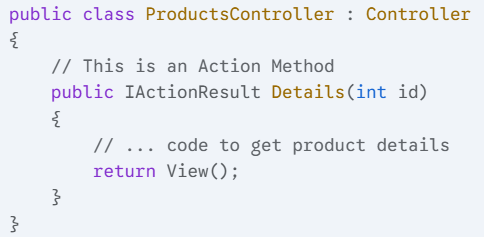
* Handles incoming HTTP requests
* Communicates with the Model (if needed)
* Returns a **View** or **JSON**, **Redirect**, etc.



**2. What is an Action Method?**

An **Action** (also known as an Action Method) is a public method within a Controller class. It is the specific method that is executed when a user's request matches a route. Only public methods can be Actions. 🡺default return type is ActionResult or View with same name as method(.cshtml file)

* The name of the method usually corresponds to a part of the URL (e.g., the Details() method for /Products/Details).



**3. Types of Action Attributes:**

| **Attribute** | **Purpose** |
| --- | --- |
| [HttpGet] | Accepts GET requests (default) |
| [HttpPost] | Accepts POST requests (form submits) |
| [HttpPut] | For updating via PUT |
| [HttpDelete] | For deleting data |
| [NonAction] | Marks a method not as an action method |

**[**NonAction]: This is a very important attribute. If you have a public helper method in your Controller that you do not want to be accessible via a URL, you must mark it with [NonAction]. This prevents the routing engine from treating it as an Action



**4. Returning Responses from Controller:**

The return type of an Action method is typically **IActionResult**. This is an interface that represents the result of the action

| **Return Type** | **Used For** |
| --- | --- |
| View() | Return a View (.cshtml) |
| Json() | Return JSON data/object (use for API or AJAX) |
| RedirectToAction() | Redirect to another action method |
| Content() | Return plain string |
| File() | Return file as download |

**5. Routing to Actions**

Default URL Pattern:

/{controller}/{action}/{id?}

The ASP.NET Core **routing engine** is responsible for matching the incoming URL to the correct Controller and Action Method.

* For a URL like /Products/Details/5, the routing engine sees:
  + Products -> It looks for ProductsController. 🡺Controller
  + Details -> It looks for the Details() **Action** method.
  + 5 -> It passes 5 as a parameter to the Details() method. 🡺id

**6.** Which return type sends a Razor view to the browser?  
A. ViewResult 🡺This is returned when a Razor .cshtml page is served.  
B. FileResult  
C. JsonResult  
D. IActionResult

**10.** Which of the following is NOT a valid return type in MVC?  
A. ViewResult  
B. RedirectResult  
C. HtmlResult 🡺 not return type  
D. JsonResult

**11.** What happens if you decorate a method with both [HttpGet] and [HttpPost]?  
A. Method runs twice  
B. Compilation error🡺You can't use [HttpGet] and [HttpPost] together on the same method **unless using [AcceptVerbs].**  
C. Method handles both GET and POST  
D. Only GET is executed

**13.** What is the use of **ModelState.IsValid**?  
A. Checks if View is working  
B. Validates form inputs🡺checks if model binding + data annotations succeeded.  
C. Checks database connection  
D. Redirects after validation

**16.** Can a controller method return a file download?  
A. No  
B. **Yes, using File()**🡺The File() method is used to return downloadable content (files, PDFs, images, etc.)  
C. Only in APIs  
D. Only if marked static

**19.** Can you overload action methods in the same controller?  
A. Yes, always  
B. No, C# doesn't support it  
C. Yes, but only with attribute routing🡺Overloading by parameter alone may cause routing ambiguity unless explicitly routed.  
D. No, MVC blocks it

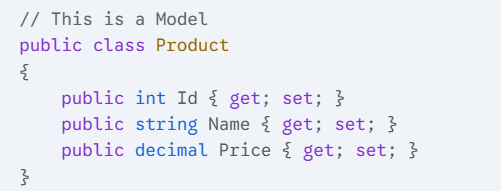
# ✅ Session 15: Understanding Views and Models

**1. What is a View in MVC?**

* A **View** is a .cshtml file used to generate UI (HTML).
* Views are returned from controller actions using return View();
* Razor syntax (@) allows C# and HTML to be mixed

**2. What is a Model in MVC?**

* A **Model** is a C# class that contains **application data and business logic**
* **Why use a ViewModel?** To shape data exactly as the View needs it
* It can be passed from controller → view using View(model)
* Strongly typed views use @model directive



**3. What is a ViewModel?**

* A **ViewModel** is a class that combines multiple models/data sources for a View

**4. Razor Syntax Examples:**

| **C# Code** | **Razor Code** |
| --- | --- |
| Display variable | @Model.Name |
| Condition | @if (Model.IsAvailable) |
| Loop | @foreach (var p in Model.Products) |

**5. Passing Data to Views:**

| **Method** | **Use Case** |
| --- | --- |
| ViewBag | Dynamic object for short term use |
| ViewData | Dictionary-based |
| TempData | Used across redirects |
| Model | Strongly typed object |

**6. Creating Views**

* Views are stored under /Views/ControllerName/ActionName.cshtml
* Shared views go to /Views/Shared/
* Layout views go to /Views/Shared/\_Layout.cshtml

**The ViewBag (and its problems)**

The ViewBag is an object used to pass data from a Controller to a View.

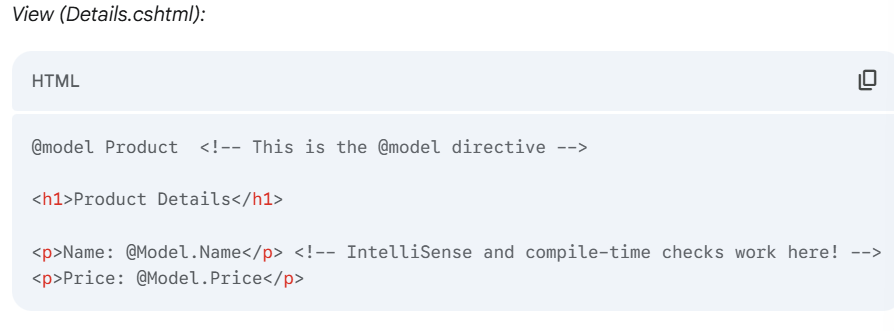
* **Key Characteristic:** It is **dynamic**. This means you can add any property to it without defining it first. (Mcq pucha hai )
* **Major Disadvantage (Very important for exams):** ViewBag is **not strongly-typed**. This leads to two big problems:
  1. There is **no IntelliSense** (can't provide its smart features—like **autocompletion**, **code hints**, and **real-time error checking**) support in the View.
  2. There are **no compile-time errors**. If you make a typo (e.g., ViewBag.Mesage instead of ViewBag.Message), your application will **crash at runtime, not during compilation.**

**Strongly-Typed Views (The Recommended Approach)**

A strongly-typed View is a View that is explicitly tied to a specific Model or ViewModel class. This is the **best practice** and solves all the problems of ViewBag.

* **How it works:** You declare the model type at the top of your view using the @model directive.
* **Major Advantages:**
  1. **Strongly-typed:** You get full IntelliSense for all model properties.
  2. **Compile-time checking:** If you make a typo in a property name, the project will fail to build, allowing you to catch errors early.





In a strongly typed Razor view, which directive is used at the top to specify the model type?

A. @ModelType Product  
B. @model Product 🡺ans  
C. @Model Product  
D. @using Product

🡺 **\_**Layout.cshtml and reference it using @layout

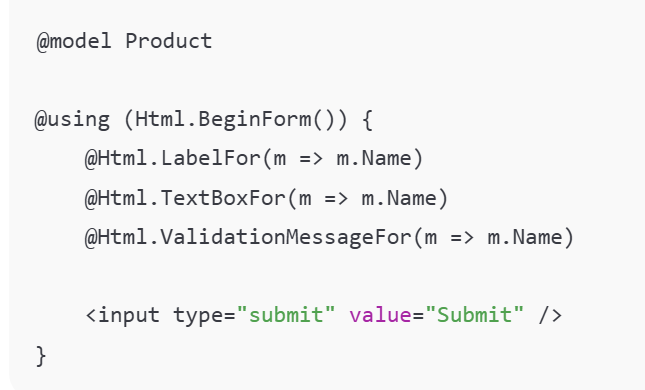
**HTML Helper Functions**

HTML Helpers are methods in Razor that help you generate HTML markup. They are very useful because they automatically use the metadata from your model (like property names and validation attributes).

* @Html.DisplayNameFor(model => model.Name): Creates a <label> tag for the Name property.
* @Html.EditorFor(model => model.Name): Creates an <input> field suitable for editing the Name property.
* @Html.ValidationMessageFor(model => model.Name): Creates a placeholder to display validation error messages for the Name property. We will cover validation in the next session.

| **Helper** | **Purpose** |
| --- | --- |
| @Html.TextBoxFor() | Input field for model property |
| @Html.LabelFor() | Label for model property |
| @Html.TextAreaFor() | Multiline textbox |
| @Html.EditorFor() | Automatically selects input type |
| @Html.ValidationMessageFor() | Show validation errors |

Example :



# Client-side and Server-side Validation using Data Annotations in ASP.NET Core MVC

**✅ 1. What is Validation?**

**Validation** means checking whether the user input is **correct, complete, and acceptable** before it goes into the system (like database).

There are **two types of validation**:

 **Client-Side Validation:** This validation happens in the user's browser using JavaScript. The Data Annotation attributes are automatically translated into HTML5 attributes (e.g., data-val-required), and JavaScript libraries (like jQuery Validation) handle the validation.

* **Benefit:** Provides instant feedback to the user, improving the user experience.
* **Drawback:** It is **not secure**. Malicious users can easily bypass or disable client-side validation.

 **Server-Side Validation:** This is the ultimate, **required** validation. It happens in the Controller on the server, after the form has been submitted.

* **Benefit:** It is **secure**. No matter how the user submits data, it will be validated on the server.
* **How it works:** You check the \*\*ModelState.IsValid\*\* property in your [HttpPost] action.

## ✅ 2. What Are Data Annotations?

**Data Annotations** are **C# attributes** (like [Required], [StringLength], etc.) you apply on your model class to define validation rules.

You must remember to include the System.ComponentModel.DataAnnotations namespace at the top of your model file.

**✅ 3. Most Common Validation Attributes**

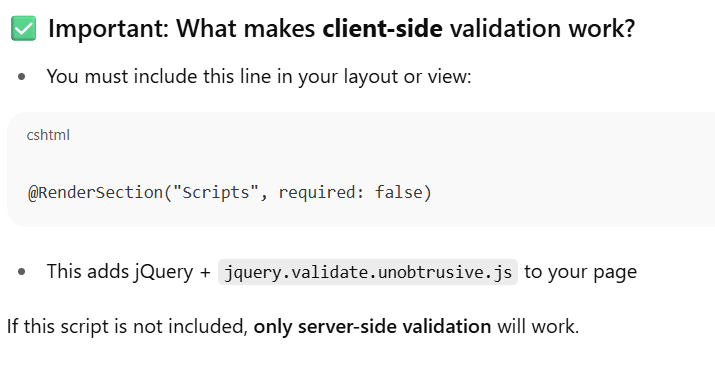
| **Attribute** | **Purpose** |
| --- | --- |
| [Required] | Field must not be null/empty |
| [StringLength(n)] | Limits length of string |
| [Range(min, max)] | Validates numeric range |
| [EmailAddress] | Validates email format |
| [Compare("Other")] | Confirms password matches |
| [RegularExpression("pattern")] | Match pattern like phone numbers |

✅ 4. How to Use It – Full Example











A developer uses the [Required] attribute on a model property. On a web page, where is this validation **first** performed to provide the user with immediate feedback?

A. In the Controller's if (ModelState.IsValid) check.

B. On the server, before the database operation.

C. In the user's browser via JavaScript. 🡺 Client-side validation happens first.

D. In the startup.cs file.

**Scaffold Templates in MVC**

**✨ What is Scaffolding?**

**Scaffolding** is the process where **Visual Studio or CLI automatically generates code** for CRUD (Create, Read, Update, Delete) based on a model.

You get:

* Controller with full actions (Index, Create, Edit, Delete)
* Views for each action
* Razor UI pre-built

**🧰 How to Scaffold in Visual Studio:**

1. Right-click on **Controllers** → Add → Controller
2. Select **MVC Controller with views, using Entity Framework**
3. Choose your model and DB context
4. Click **Add**

✔ This creates:

* ProductController.cs
* Views like Index.cshtml, Create.cshtml, etc.

**📘 Scaffold Templates Generated:**

| **View Name** | **Purpose** |
| --- | --- |
| Index | List all records |
| Create | Add new item |
| Edit | Modify existing |
| Delete | Confirm & delete |
| Details | View record details |

**🔹 MCQ 13:**

What interface should be implemented for a model to perform custom validation logic?

A. IModelValidator  
B. ICustomValidation  
C. IValidatableObject  
D. IValidateModel

✅ **Answer: C. IValidatableObject**  
**Explanation:** This interface has the Validate() method to return custom ValidationResults.

**15.** What method must be implemented when using IValidatableObject?

A. IsValid()  
B. ValidateData()  
C. Validate()  
D. CheckModel()

✅ **Answer: C. Validate()**  
**Explanation:** That method returns a list of validation results based on your custom logic.

# ✅ Session 16: MVC State Management in ASP.NET Core

**🔍 What is State Management?**

In web apps, **"state" means data that should persist between HTTP requests**.

👉 But HTTP is **stateless**, meaning:

* Every request is **independent**
* Server doesn’t remember previous data unless we **manually manage state**

So, **state management** is about storing data across requests:

* For current request (short)
* Between requests (medium)
* Across users/sessions (longer)

**✅Category 1: Per-Request State**

**This data only lives for the duration of a single HTTP request.**

* **🡺Once the response is sent back to the client, the data is gone**

**1. ViewBag and ViewData**

**🔸 ViewBag:**

* **Dynamic** object to pass data from **controller to view**
* Exists **only during the current request** Not accessible across redirects

**csharp**

ViewBag.Message = "Hello";

**cshtml**

<h3>@ViewBag.Message</h3>

**🔸 ViewData:**

* Dictionary-based (key-value pairs)
* Also available only for current request

**csharp**

ViewData["Username"] = "Shivam";

**cshtml**

<p>Hello @ViewData["Username"]</p>

**✅ Category 2: Short-Term, Per-Redirect State**

**This data is meant to be passed between two sequential requests, typically during a redirect.**

**TempData**

**TempData is a dictionary object (TempData["key"] = value)**

**It persists data for only one request after a redirect**

**Stores TempData in Session**

**But it clears it automatically after it’s read once**

**If you need to keep it again, you must use TempData.Keep() method**

** A user successfully submits a form to create a new product and is then redirected to the product list page. To display a message like "Product added successfully!" on the list page, which state management mechanism is the most suitable?**

**A. Session**

**B. ViewBag**

**C. TempData**

**D. QueryString**

**TempData is the perfect solution for this scenario. It is designed to pass data from one request to the very next request, which is exactly what happens during a redirect. ViewBag would not work because it's only for a single request, and Session would be overkill and require manual cleanup.**



**✅Category 3: Per-User Session State**

**This data is specific to a single user and persists across multiple requests from that user until their session expires or is manually cleared.**

**🡺Session state is specific to a single user, while Application state is shared across all users.**

**Session 🡺This is a dictionary-like object that stores data on the server-side. Each user has their own separate session.**

* **Stored on server** per user
* Available across multiple pages
* Needs to be enabled in Startup.cs:/program.cs

csharp

app.UseSession();

services.AddSession();

csharp

HttpContext.Session.SetString("User", "Shivam");

csharp

string user = HttpContext.Session.GetString("User");

**✅ Category 4: Per-Application State**

**This data is available to all users across the entire application's lifetime.**

* **Application: This is a dictionary-like object that stores data on the server-side and is shared among all users.**

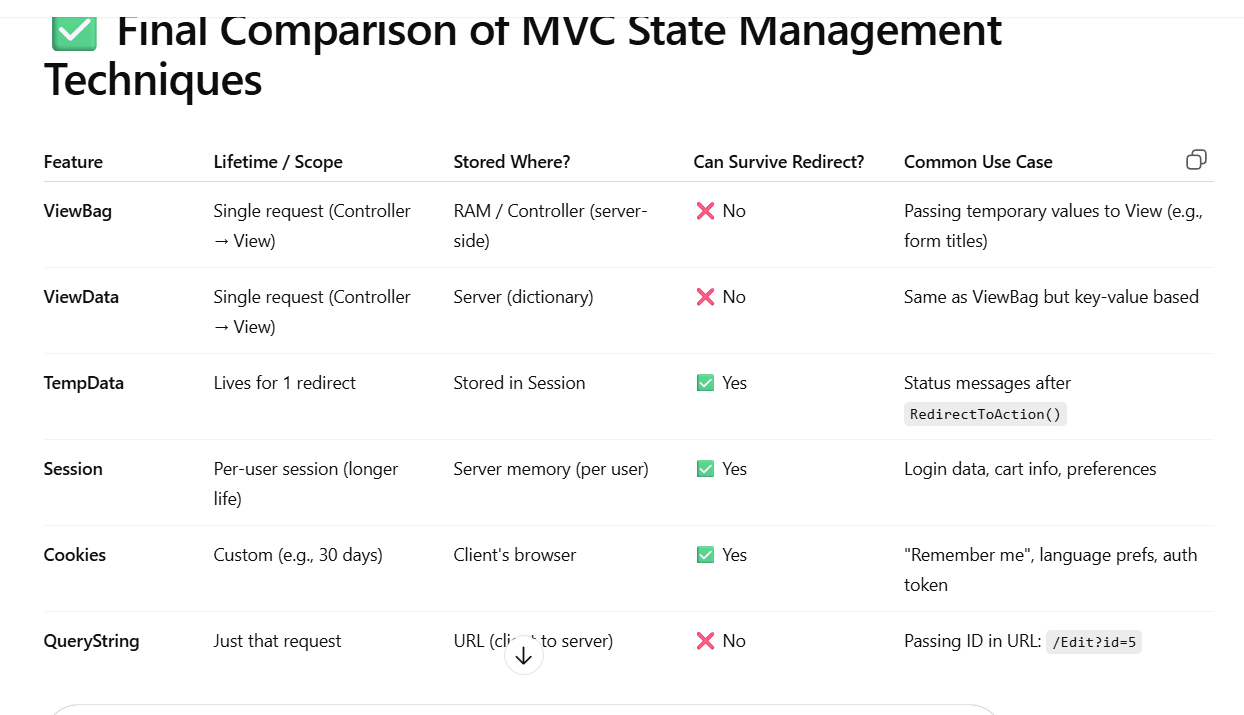
**Application (not used in ASP.NET Core)**

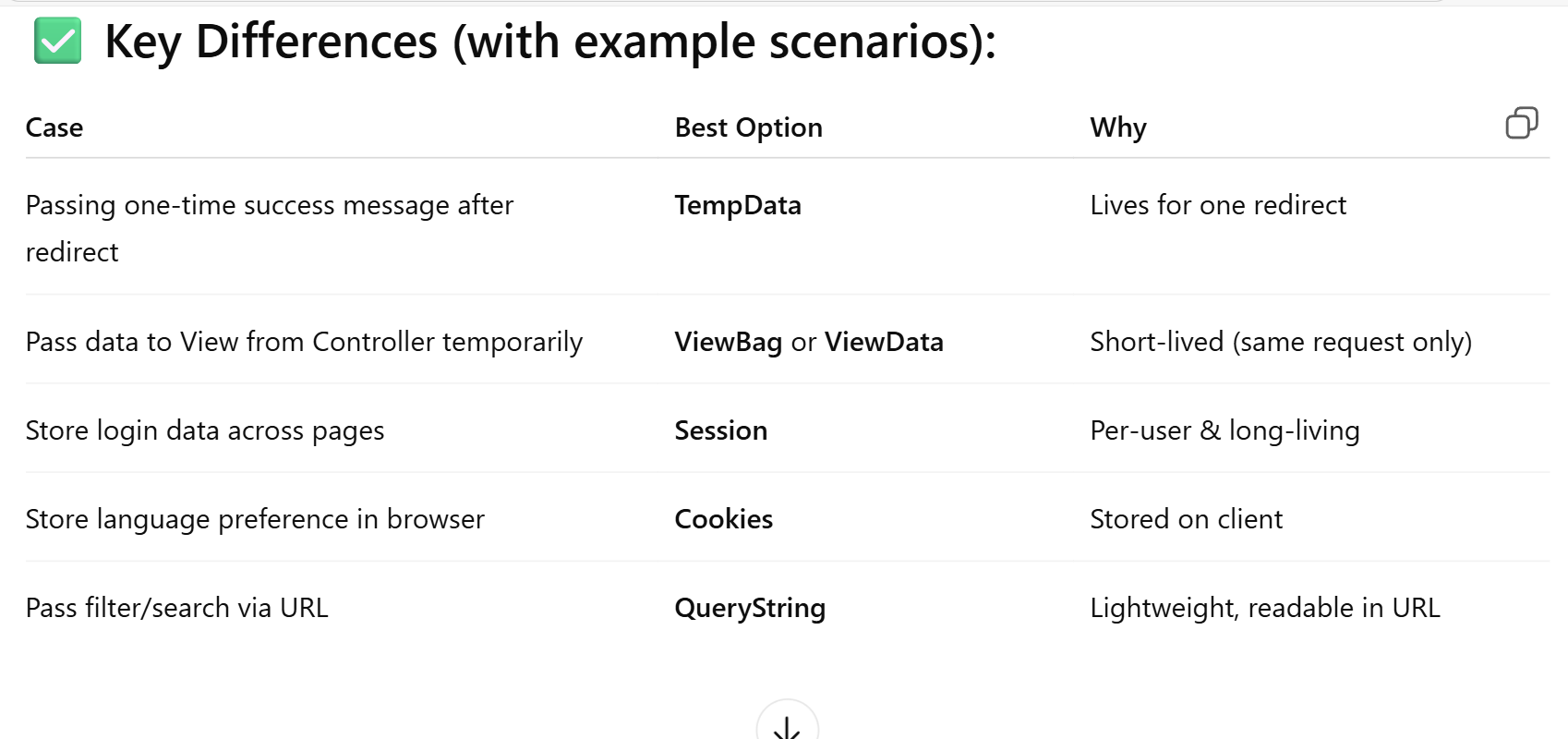
* In classic ASP.NET, Application["key"] was shared globally across all users
* Not available in Core — use Singleton service instead

**Category 5: Client-Side State**

**This data is stored on the client (the user's browser).**

* **Cookies: Small pieces of text data that the server sends to the client's browser. The browser stores it and sends it back with every subsequent request.**
* **QueryString: This is data appended directly to the URL (e.g., ?id=123&sort=name).**
* **Advantages:**
  + **No server memory is used.**
* **Disadvantages (Very important for exams):**
  + **Insecure: Data is visible to the user and can be tampered with.**
  + **QueryString has a limited length.**





**✅ What is ViewBag?**

* Used to pass **data from Controller to View**
* Data exists **only during that single request**
* **Lost if you redirect** to another action
* @ViewBag.PropertyName is used inside .cshtml views to print data from controller.

**✅ What is TempData?**

* Used to pass data from one **action to another**, especially **after redirects**
* Lives **for 1 more request**
* Useful in **Post-Redirect-Get pattern (PRG)**

# ✅ Session 17: Partial Views & Child Actions

**🔍 What is a Partial View?**

A **Partial View** is like a **sub-view**:  
🔸 It **doesn't include layout**  
🔸 It’s **reused** across multiple views (like headers, menus, footers, cards, etc.)

Think of a partial view as a component like a \_LoginPartial.cshtml for a user login form, or a \_ProductCard.cshtml that is used to display a single product in a list. This allows you to follow the "Don't Repeat Yourself" (DRY) principle.

**✅ What is a Child Action?**

⚠️ In **ASP.NET MVC 5** (not in ASP.NET Core), you could create a **child action method** and call it inside a view using:

@Html.Action("ChildActionName", "ControllerName")

✅ In **ASP.NET Core**, **child actions are replaced** by:

* **View Components**

**Action Method and Child Action**: An Action Method can be called from within a view. This is often referred to as a "child action." It allows a partial view to have its own logic to get data from a controller. You use the @Html.Action() or @Html.RenderAction() helper functions to call a child action from a view.

Which method is used to insert a partial view **within a Razor block (@{})**?

A. @Html.Partial()  
B. Html.RenderPartial()  
C. @RenderView()  
D. Html.PartialView()

✅ **Answer: B. Html.RenderPartial()**  
**Explanation:** RenderPartial() writes directly to the response output stream, so it's called in @{ }.

**Why Is RenderPartial() Faster?**

* Html.Partial() returns a string → MVC engine **buffers the output**, then writes it
* Html.RenderPartial() sends HTML **directly to the browser** (no buffering)

That’s why **option C** is correct:

✅ RenderPartial() writes directly to output and has better performance

# Data Management with ADO.NET

**🎯 First: What is ADO.NET?**

**ADO.NET** (ActiveX Data Objects for .NET) is a **.NET library** that provides **low-level, fast, and flexible** access to relational databases like **SQL Server**, **MySQL**, and others.

🧠 It is part of **System.Data** namespace.

**✅ ADO.NET Object Summary with Analogy**

| **ADO.NET Object** | **Primary Purpose** | **Real-World Analogy** |
| --- | --- | --- |
| SqlConnection | Manages the connection to the database | ☎️ Phone line to the database |
| SqlCommand | Represents and sends SQL queries or commands to **database** | 📨 Message or instruction sent over the line |
| SqlDataReader | Fast, forward-only reading of result rows | 📖 Reading a letter one line at a time |
| SqlDataAdapter | Transfers data between DB and memory (DataSet/DataTable) | 📦 Mailman delivering data to/from database |
| DataSet / DataTable | In-memory, disconnected storage for tabular data(Stores data in-memory for disconnected use) | 📊 A spreadsheet or notebook in memory |

**✅ Other Commands:**

| **Task** | **Command** |
| --- | --- |
| SELECT | ExecuteReader() |
| INSERT / UPDATE | ExecuteNonQuery() |
| SCALAR value (1 cell) | ExecuteScalar() |

What does ExecuteNonQuery() return?

A. The first row of result  
B. The number of rows affected  
C. The result set as a table  
D. A connection string

✅ **Answer: B. The number of rows affected**

Which class in ADO.NET is used to send a SQL command to the database?

A. SqlAdapter  
B. SqlCommand  
C. SqlReader  
D. SqlRunner

✅ **Answer: B. SqlCommand**

# ✅ SqlDataAdapter and DataSet / DataTable

# the *Disconnected Architecture*

**🔍 Why Disconnected?**

Sometimes, you don’t want to keep your database connection open while reading/writing data.  
Instead, you want to:

1. **Fetch data**
2. **Work with it in memory**
3. **Update back to DB when needed**

This is called the **disconnected model**,

**✅ ADO.NET Disconnected Architecture Summary**

| **Component** | **Purpose** | **Analogy** |
| --- | --- | --- |
| SqlDataAdapter | Transfers data between DB and memory (fills DataTable) I,e **bridge btw db and DataTable** | 📦 Mailman: fetches & delivers data |
| DataTable | Holds one table's data (in memory) | 📊 Excel-like table in RAM |
| DataSet | Container for multiple DataTables (like a mini database) | 📚 A workbook with many sheets |
| Fill() | Loads DB data into DataTable/DataSet | ⬇️ Download data from DB |
| Update() | Pushes changes in memory back to DB | ⬆️ Upload modified data to DB |
| SqlCommandBuilder | Auto-generates insert/update/delete queries for adapter | 🛠️ Auto-writes SQL for you |

**🎯 MCQ Pointers:**

**Object for fast, forward-only reading: ✅ SqlDataReader**

**Bridge for DataSet and DB: ✅ SqlDataAdapter**

**Disconnected in-memory storage: ✅ DataSet**

**Connection object: ✅ SqlConnection**

**Can DataSet store multiple tables? ✅ Yes**

**✅ Disconnected Model Flow:**

1. Create SqlDataAdapter
2. Call Fill() → loads data into memory
3. Modify DataTable locally
4. Call Update() to save changes

This model is great for:

* Offline work
* Batch processing
* Grid views or form-bound UIs

# ✅ Asynchronous Data Access in ADO.NET

# (AKA: Making your DB operations non-blocking and faster)

**🎯 Why Use Async in ADO.NET?**

In a web app, **synchronous DB access blocks the thread** — meaning:

* While waiting for the DB to respond, the server can’t do anything else
* This reduces performance and scalability of your app

✅ With **asynchronous ADO.NET**, the thread is **freed** while the DB is processing.

**✅ Common Asynchronous ADO.NET Methods**

| **Method** | **Purpose** |
| --- | --- |
| OpenAsync() | Opens a database connection asynchronously |
| ExecuteReaderAsync() | Reads result set (SELECT) async |
| ExecuteNonQueryAsync() | Executes commands like INSERT/UPDATE |
| ExecuteScalarAsync() | Retrieves single value (first row/col) |

**✅ Where Is This Useful?**

| **Scenario** | **Async Recommended?** |
| --- | --- |
| ASP.NET MVC / Web API | ✅ Yes |
| Long-running DB operations | ✅ Yes |
| Console apps or quick scripts | ❌ Not needed |

**33.** Which keyword is required in the method signature for using await?

A. static  
B. void  
C. async  
D. awaitable

✅ **Answer: C. async**

# Session 19: Routing and Request Life Cycle

**✅ Part 1: What is Routing?**

**Routing is the process of mapping an incoming HTTP request URL to a specific Controller and Action method**

**Routing Engine & Routing Table:** When a request comes in, the Routing Engine compares the URL against a set of predefined patterns, called the Routing Table. When it finds a match, it knows which Controller and Action to execute**.**

**Default Routing Pattern (Conventional Routing)**

By default, ASP.NET Core MVC uses a convention-based routing pattern. The most common pattern is: {controller}/{action}/{id?}

* {controller}: A placeholder for the name of the controller (e.g., Home, Products).
* {action}: A placeholder for the name of the action method (e.g., Index, Details).
* {id?}: A placeholder for an optional parameter, typically used for a record's ID. The ? makes it optional.

Example: A URL like /Products/Details/5 would be mapped to the Details() action in the ProductsController, with 5 passed as the id parameter.

**✅ Part 2: Attribute Routing (Advanced)**

This is an alternative to the default pattern. You define the route directly on the controller and action methods using the [Route] attribute. This gives you more control over your URLs and is often used to create RESTful URLs.}

🔸 Attribute routing gives you **full control** over URL structure.

**404 Error: A 404 "Not Found" error occurs when the routing engine cannot find any matching route pattern for the requested URL. This means the URL does not correspond to any valid controller or action in your application.**

**✅ Part 3: Request Life Cycle in ASP.NET Core MVC**

**🔁 Request Flow:**

1. **Browser sends HTTP request**
2. **Middleware pipeline is triggered**
3. **Routing Middleware** matches URL → controller + action
4. **Controller** processes logic
5. **View is selected** using Razor
6. **Response is sent back to the browser**

## Common Middleware in MVC Apps:

| **Middleware** | **Purpose** |
| --- | --- |
| UseRouting() | Activates routing |
| UseEndpoints() | Maps routes to controllers |
| UseAuthorization() | Handles access checks |
| UseStaticFiles() | Serves static files like .css, .js |
| UseSession() | Enables Session support |

Given the default route pattern {controller}/{action}/{id?}, which URL will most likely invoke the Index() action in the HomeController?

A. /Index

B. /Home/Index

C. /home/index/1

D. Both B and C 🡺ans

# Layouts, Bundling & Minification

**Part 1: Layouts**

**What is a Layout Page?** A **Layout Page** is a shared template that provides a consistent structure and design for all the views in your application. It typically contains the common elements of your website, such as the header, footer, navigation bar, and links to CSS and JavaScript files.

* **File Naming:** By convention, the main layout page is named \_Layout.cshtml. The underscore (\_) at the beginning indicates that it's a partial or shared file.
* **Location:** It is usually placed in the Views/Shared folder.

**The Key Directives (@RenderBody() & @RenderSection()):**

* **@RenderBody()**: This is the most important directive in a layout page. It acts as a placeholder where the content of the individual view will be rendered. Every view that uses this layout will have its content placed in this spot.
* **@RenderSection("SectionName", required: false)**: This is used to define named content sections. It allows an individual view to provide specific content (like a JavaScript script for a single page) that will be rendered at that location in the layout. The required parameter lets you specify if the section must be provided by the view.

**Benefits:**

* **Consistency:** Ensures a uniform look and feel across all pages.
* **Reduced Repetition:** You avoid writing the same HTML for the header, footer, etc., in every view file. This follows the "Don't Repeat Yourself" (DRY) principle.

**Part 2: Bundling and Minification**

This is a very important topic for your exam, as it is directly related to application performance.

**How to Add Bundles (in BundleConfig.cs if using legacy or tools like Gulp/Grunt/WebOptimizer)**

* **What is Bundling?** Bundling is the process of combining multiple CSS or JavaScript files into a single file.
  + **Example:** Instead of having three separate requests for site.css, buttons.css, and forms.css, bundling combines them into one file, e.g., styles.css.
  + **Benefit:** Reduces the number of HTTP requests a browser has to make to load a page, which speeds up the page load time.
* **What is Minification?** Minification is the process of removing unnecessary characters (like whitespace, comments, and line breaks) from a file to reduce its size.
  + **Example:** A 10 KB JavaScript file might become an 8 KB file after minification.
  + **Benefit:** Reduces the overall file size that needs to be transferred over the network, which also speeds up page load time.
* **Combined Purpose:** The combined purpose of bundling and minification is to **improve the performance** of your web application.

### 🔸 How to Enable Minification

Set this in web.config or during bundling:

BundleTable.EnableOptimizations = true;

Or in ASP.NET Core, **set:**

environment.IsDevelopment() // false → enables minify

**✅ Summary Table**

| **Concept** | **Purpose** | **Tools/API Used** |
| --- | --- | --- |
| Layouts | Master page for shared layout (header, footer) | \_Layout.cshtml + @RenderBody() |
| Bundling | Combines multiple files into one | WebOptimizer / Gulp |
| Minification | Shrinks files by removing spaces/comments | EnableOptimizations / prod env |

Which Razor function is used in a layout file to inject the view content?

A. @RenderView()  
B. @RenderContent()  
C. @RenderBody()  
D. @RenderSection()

✅ **Answer: C. @RenderBody() or**

In a Razor \_Layout.cshtml file, which directive acts as a placeholder for the content of an individual view? A**. @RenderBody()** B. @RenderPage() C. @ViewBag.Body D. @RenderSection()

**39.** How do you control whether bundling/minification is active?

A. Change layout file  
B. Set BundleTable.EnableOptimizations  
C. Use ViewBag.Minify  
D. Use TempData

✅ **Answer: B. Set BundleTable.EnableOptimizations**

# MVC Security in ASP.NET

**🔐 PART 1: Authentication vs Authorization**

**✅ Authentication**

**Authentication** means:

"Are you a real user? Who are you?"

It checks **who the user is**.

* You log in using a username and password.
* ASP.NET Core **creates a cookie** to remember you.
* That cookie is stored on your browser and sent with every request.

**✅ Authorization**

**Authorization** means:

"Now that I know who you are, are you allowed to do this?"

It checks **what you’re allowed to do**.

**🔹 Example: code**

**[Authorize]**

public IActionResult Dashboard() { ... }

* If you're **not logged in**, you’ll be **redirected to login page**.
* If you're logged in: ✅ allowed.

### 🔹 Allow Public Access

To allow **unauthenticated (public)** users to access something, use:

[AllowAnonymous]

public IActionResult HomePage() { ... }

So you use [Authorize] to protect sensitive areas  
and [AllowAnonymous] to open public pages.

## 🔐 PART 2: **Anti-Forgery Token (CSRF Protection)**

### ❓ What is CSRF?

**CSRF = Cross Site Request Forgery**

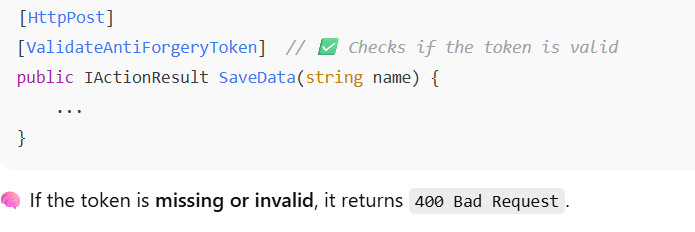
🛑 **Danger**: Imagine you're logged into your bank. A hacker sends a fake form that hits your bank’s endpoint and **steals your money**, using your session.

### ✅ Solution in ASP.NET Core:

ASP.NET Core **automatically protects** you by requiring a **token** for every form submission.

🔹 How To Use It:

* 1. In your Razor view: 

2. In your Controller: 

**🔐 PART 3: XSS (Cross Site Scripting) Protection**

**❓ What is XSS?**

**XSS = Injecting malicious JavaScript into your app**.

🛑 Example:  
User enters:

html

CopyEdit

<script>alert('hacked')</script>

If you **print that directly**, it executes in other users’ browsers = BAD.

**✅ Solution: Razor HTML Encoding**

ASP.NET Core Razor views automatically protect against XSS:

cshtml

<p>@Model.Comment</p> // ✅ Safe

Razor **HTML-encodes** the output:

html

<script> → &lt;script&gt;

**⚠️ If You Use @Html.Raw(...)**

* **@Model.Comment → Razor automatically HTML-encodes this output**
* **@Html.Encode(Model.Comment) → Explicitly encodes it (does the same thing manually)**

**🔴 But @Html.Raw(Model.Comment) disables encoding, and is dangerous unless the data is 100% tru**

**🔐 PART 4: Security Attributes Summary**

| **Attribute / Tag** | **What It Does** |
| --- | --- |
| [Authorize] | Only logged-in users can access this action/controller |
| [AllowAnonymous] | Allows anyone (even unauthenticated) |
| [ValidateAntiForgeryToken] | Protects POST forms against CSRF |
| @Html.AntiForgeryToken() | Inserts a CSRF token inside the HTML form |

**✅ Final Summary Table**

| **Concept** | **How to Use / Protect** |
| --- | --- |
| Authentication | Login forms, cookies, [Authorize] |
| Authorization | Role-based access using [Authorize(Roles="...")] |
| CSRF Protection | @Html.AntiForgeryToken() + [ValidateAntiForgeryToken] |
| XSS Protection | Razor auto-encodes HTML, avoid @Html.Raw() |
| Allow anonymous access | Use [AllowAnonymous] |

🔐 What prevents a malicious form submission in ASP.NET Core MVC?  
✅ Anti-Forgery Token

Which tag is used in Razor views to **inject an Anti-Forgery Token** into a form?

A. @Html.ValidateToken()  
**B. @Html.AntiForgeryToken()**  
C. @Token.AntiForgery()  
D. @AntiForgery.Insert()

**Explanation:**

* @Html.AntiForgeryToken() is the Razor helper that:
  + Inserts a **hidden input field** in your form.
  + The value of that field is a **CSRF token** that is verified during form submission.
* It's used along with [ValidateAntiForgeryToken] in your controller to prevent CSRF attacks.

 **Razor automatically HTML-encodes** any variables you output using @Model.Property.

* This prevents malicious JavaScript (like <script>alert(1)</script>) from executing.

 Razor does **not inject JavaScript** (Option B is incorrect).

 XSS affects the **browser**, not the database (C is incorrect).

 XSS is very much a **real threat** in views if you don't sanitize properly (

 The combination of @Html.AntiForgeryToken() (in the form) and [ValidateAntiForgeryToken] (on the action) **protects your app against CSRF attacks**.

 **CSRF** = An attacker tricks an authenticated user into unknowingly submitting a malicious request.

 The anti-forgery token ensures that the **form request came from your own site**, not a third-party site.

# Entity Framework (EF) – Code First Approach in ASP.NET Core MVC

**✅ What is Entity Framework?**

**Entity Framework (EF)** is an **ORM (Object-Relational Mapper)**:

* It maps **C# classes to database tables**
* You can write **LINQ queries** instead of SQL
* It handles all DB interactions (create, insert, update, delete)

**🔹 1. Code First Basics**

* **Code First means you define the database structure using C# classes, and EF Core creates the database automatically.**
* **Main advantage of Code First: Full schema control in code, with automatic DB creation via migrations.**
* **MCQs asked:**
  + **Definition of Code First**
  + **Primary benefit of using Code First approach**

**🔹 2. DbContext & DbSet**

* **DbContext is the core class used to interact with the database.**
* **DbSet<T> is used inside DbContext to represent a table in the database.**
* **You access and query data through DbSet properties like db.Students.**

**✅ Code First Approach**

**💡 What it means:**

You **write your classes first**, and EF **creates the database automatically**.

**Step 1: Create a Model (POCO Class)**

public class Student

{

public int Id { get; set; } // Primary key (by convention)

public string Name { get; set; }

public int Age { get; set; }

}

🡺 ✅ EF will create a **Students** table with columns: Id, Name, Age

**🔸 Step 2: Create a DbContext**

public class AppDbContext : DbContext

{

public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

public DbSet<Student> Students { get; set; } // Table

}

**🔸 Step 3: Register DbContext in Program.cs**

builder.Services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer("Your\_Connection\_String"));

**🔸 Step 4: Create a Migration**

dotnet ef migrations add InitialCreate

This command:

* Scans your model
* Generates SQL to build the schema

### 🔸 Step 5: Apply Migration to DB

dotnet ef database update

EF will create your **database** and the Students table.

**🔷 Key Components**

| **Component** | **Purpose** |
| --- | --- |
| Model class | C# class (POCO) that defines table structure |
| DbContext | EF Core class that manages DB connection & CRUD |
| DbSet<T> | Represents a **table** for type T |
| Migrations | Track and apply schema changes to the database |

**🡺Summary**

**3.** Migrations & Database Creation

* Use dotnet ef migrations add <Name> to generate migration files.
* Use dotnet ef database update to apply migrations and create/update DB.
* SaveChanges() is called after Add(), Update(), or Remove() to persist changes.
* MCQs asked:
  + How to generate and apply migrations
  + Purpose of SaveChanges()

**✅ CRUD with EF (LINQ Queries)**

**🔹 Create**

db.Students.Add(new Student { Name = "Shivam", Age = 22 });

db.SaveChanges();

**🔹 Read**

var list = db.Students.ToList();

**🔹 Update**

var std = db.Students.First(x => x.Id == 1);

std.Age = 23;

db.SaveChanges();

**🔹 Delete**

var std = db.Students.Find(1);

db.Students.Remove(std);

db.SaveChanges();

**🔷 CRUD Operations in EF Core**

| **Operation** | **Code Example** |
| --- | --- |
| **Create** | db.Students.Add(new Student {...}); db.SaveChanges(); |
| **Read** | var list = db.Students.ToList(); |
| **Update** | var s = db.Students.Find(1); s.Name = "Updated"; db.SaveChanges(); |
| **Delete** | var s = db.Students.Find(1); db.Students.Remove(s); db.SaveChanges(); |

**5. Data Annotations & Configuration**

* [Key] is used to mark the primary key manually.
* [Required] makes a field **NOT NULL** in the database.
* [MaxLength(n)] limits the string length in the DB schema.
* [Table("Name")] lets you customize the table name. or remane
* [ForeignKey("...")] defines foreign key relationships between entities.
* Nullable types (string?, int?) make a column optional.

**6. Relationships and Navigation Properties**

* Navigation properties let EF **understand relationships** (1-to-many, many-to-many).
* Foreign keys can be defined explicitly using [ForeignKey("PropertyName")].
* POCO (Plain Old CLR Object) classes are used to define models in EF

**🔹 7. EF Core Packages and Conventions**

* Default table name = same as the class name (e.g., Student → Students).
* EF Core for SQL Server requires:  
  Microsoft.EntityFrameworkCore.SqlServer

# Introduction to Web APIs

**What is a Web API?**

A **Web API** (Application Programming Interface) is a type of web application that is designed to expose data and functionality to other client applications. Unlike a traditional ASP.NET Core MVC application that serves HTML pages to a web browser, a Web API serves **data** (typically in **JSON** or XML format) to clients.

🡺 Typically used to **send and receive JSON**

**Who uses a Web API?**

* **Mobile Apps:** An iPhone or Android app might call an API to get user data.
* **Single-Page Applications (SPAs):** A web application built with frameworks like React or Angular uses APIs to get the data it needs to display.
* **Other Services:** Another web service might call your API to perform a specific task.

## 🔶 MVC vs Web API

| **Feature** | **MVC (Controller)** | **Web API (ApiController)** |
| --- | --- | --- |
| Returns | Views (HTML) | Data (JSON, XML) |
| Used by | Browser | Any client (JS, mobile) |
| Decorator | [Controller] | [ApiController] |
| Return type | IActionResult → View | ActionResult<T> → JSON |
| Routing | /Controller/Action | /api/[controller] |

**Core Concept: The RESTful API**

Most modern Web APIs are designed using **REST (Representational State Transfer)** architectural principles. For your exam, you need to know these key concepts:

1. **Stateless:** This is the most important principle. Each request from a client to the server must contain all the information the server needs to understand and fulfill it. The server should not rely on any information from a previous request.
   * **Exam-Relevant:** This makes APIs scalable, because any server can handle any request.
2. **Resources:** Data is exposed as a "resource." A resource is an object with a type, associated data, and relationships to other resources.
   * **Example:** In a RESTful API, a user is a resource, a product is a resource, and an order is a resource. You identify them with URLs (e.g., /api/products).
3. **Standard HTTP Verbs:** REST uses standard HTTP verbs (methods) to perform actions on resources. This is a very common topic for MCQs.
   * **GET**: Used to **retrieve** a resource or a collection of resources. It's a read-only operation.
   * **POST**: Used to **create** a new resource.
   * **PUT**: Used to **update or replace** an existing resource.
   * **PATCH**: Used to **partially update** an existing resource.
   * **DELETE**: Used to **delete** a resource.

**Creating a Web API Controller**

In ASP.NET Core, an API controller is a class that inherits from ControllerBase. It is usually decorated with the [ApiController] and [Route] attributes.

* [ApiController] attribute: This attribute enables a number of API-specific behaviors like automatic HTTP 400 responses on model validation errors.
* [Route("api/[controller]")] attribute: This sets the base URL for all actions in the controller. [controller] is a token that will be replaced with the name of the controller (e.g., ProductsController becomes api/products).

According to REST principles, what does it mean for an API to be "stateless"?

A. The API does not have a database to store information.

**B. Each request from a client contains all the information needed, with no server-side context saved between requests.**

C. All API requests must be secured with a user token.

D. The API does not use cookies.

**✅ What is CORS?**

CORS = **Cross-Origin Resource Sharing**

* Allows your API to be called from **a different domain** (like localhost:3000 React app)
* CORS allows a frontend (e.g., React on port 3000) to access an API (e.g., backend on port 5000).
* You must **enable CORS** in Program.cs:

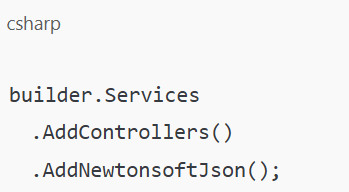


**🧠 MCQ Summary (Web API)**

1. Web API returns:
   * Answer: JSON or XML, not HTML
2. [ApiController] is used:
   * Answer: To make controller return data and use model validation
3. Default route for Web API:
   * Answer: api/[controller]
4. What is CORS used for?
   * Answer: To allow frontend apps on different domains to call the API

**🔷 4. JSON Serialization**

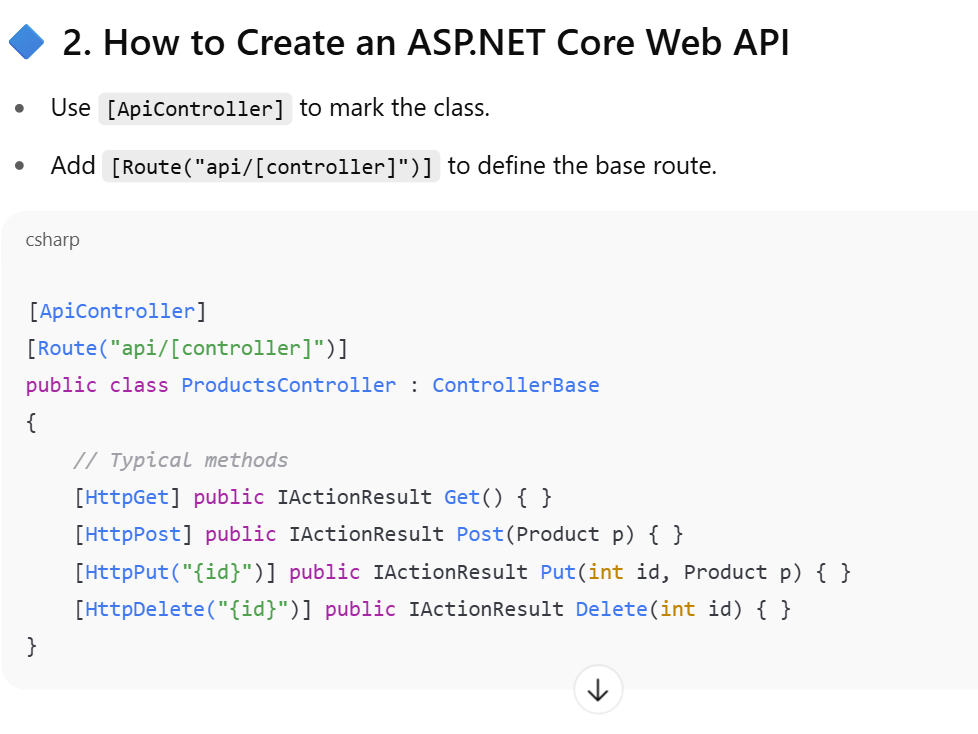
* ASP.NET Core uses **System.Text.Json** by default.
* You can use **Newtonsoft.Json** (add NuGet: Microsoft.AspNetCore.Mvc.NewtonsoftJson):



* Newtonsoft supports advanced features like:
  + JsonIgnore
  + ReferenceLoopHandling
  + Custom converters

**🔷 3. Supported HTTP Verbs**

| **HTTP Verb** | **Purpose** | **Example API Route** |
| --- | --- | --- |
| GET | Read data | GET /api/products |
| POST | Create data | POST /api/products |
| PUT | Update data | PUT /api/products/5 |
| DELETE | Delete data | DELETE /api/products/5 |



 **When a client sends a new resource to a Web API via a POST request, what should the Content-Type header of the request typically be set to?**

* **Correct Answer: C. application/json**
* **Explanation:** This is a crucial part of a POST or PUT request to a modern API. The Content-Type header tells the server the format of the data being sent in the request body. Since most Web APIs exchange data in JSON format, application/json is the correct value.

 **In a modern JavaScript application, which built-in browser API is commonly used to make HTTP requests to a Web API?**

* **Correct Answer: C. fetch**
* **Explanation:** The fetch API is the modern, standard method in JavaScript for making network requests. It uses promises, which makes asynchronous code easier to write and read than older technologies like XMLHttpRequest.

 **What is the purpose of the JSON.stringify() method in a client-side fetch request?**

* **Correct Answer: C. To convert a JavaScript object into a JSON string for the request body.**
* **Explanation:** While JavaScript works with objects, APIs expect to receive a raw string of data. The JSON.stringify() method takes a JavaScript object and converts it into a JSON formatted string, which can then be sent in the request body.

**Q11. How do you manage states in ASP.NET MVC application?**

**Options:**  
A. Application  
B. Session  
C. ViewBag  
D. All of the above

**✅ Correct Answer:** **D. All of the above**

**✅ Q14. How do you get information from a form that is submitted using the "post" method?**

**Options:**  
A. Request.QueryString  
B. Request.Form  
C. Response.Write  
D. Response.Writeln

**✅ Correct Answer:** **B. Request.Form**

💡 **Explanation:**

* Request.Form["fieldName"] is used to retrieve form data sent using POST.
* QueryString is used for GET.
* Response.Write just outputs data to the browser, not for retrieving form input.