<u>Topics we are going to learn:</u>

- ✓ **What is C#? (History,Language,Features,Definition).**
- ✓ **Why is C#?**
- ✓ **Compilation and OS Interaction**
- ✓ **Runtimes;**
- ✓ **Frameworks;**
- ✓ **.Net Standard;**
- ✓ **.NET Other Topics (**Dependency Injection (DI), Entity Framework (EF),NuGet, Blazor);
  - ✓ **Multithreading & Parallel Programming;**
  - ✓ **Design Patterns in C#;**
  - ✓ **File Handling & Serialization;**
  - ✓ **Interoperability & Integration;**
  - ✓ **C# in Game Development;**
  - ✓ **Security in C#;**
  - ✓ **Testing in C#;**

1)What is C#?

- object-oriented;
- developed by Microsoft (Anders Hejlsberg);
- released by Microsoft in 2000 as part of the .NET framework;
- Initially designed to simplify the development of Windows applications;
- C# incorporated features from languages like C++ and Java.
  - "incorporated" means that C# has included or adopted certain features or characteristics from other programming languages like C++ and Java.
- type-safe programming language;
  - type-safe: It ensures that variable are used only in ways that are consistent with their declared types, preventing type errors at compile time.
  - এর মানে হল যে আপনি কোন ভেরিয়েবলে ভুল টাইপের মান বরাদ্দ করতে পারবেন না। অথবা এমনভাবে ব্যবহার করতে পারবেন না যা তার টাইপের সাথে অমিল হয়।.
  - C#-এ এটি কম্পাইলার দ্বারা প্রয়োগ করা হয়, which checks type compatibility before code is executed.
  - Example:

```
int number = 10;
string text = "Hello";

// নিম্নলিখিত লাইনে কম্পাইল-টাইম এরর হবে কারণ আমরা একটি string-কে int এ বরাদ্দ করতে পারি না।
number = text; // Error: Cannot implicitly convert type 'string' to 'int'
```

- part of .NET ecosystem;
  - **.NET ecosystem** is an extensive platform designed to support a wide range of application types (web, desktop, mobile, cloud, etc.)

with tools and frameworks for development, testing, deployment, and scalability

- **Deployment:** It refers to the process of making an application or software available for use by end-users. Ex- Web Hosting, Cloud Deployment, Containers (libraries, configurations, etc.)
    - **Cloud Deployment** refers to hosting applications in a cloud environment, like AWS, Azure, or Google Cloud. Both C# and Java applications can be deployed to the cloud to get cloud resources (e.g., compute power, databases, storage) for scalability and performance.
- **Scalability**: It refers to the ability of an application or system to handle an increasing amount of work.
    - Vertical Scalability (Scaling Up): This involves adding more resources (CPU, RAM, storage) to an existing server to increase its performance
    - Horizontal Scalability (Scaling Out): This involves adding more servers to distribute the load across multiple machines or instances.

Key Differences

- **C#** is a programming language.
- **.NET Framework** is a platform/runtime that supports running C# applications.

- designed to create a variety of application (Web and desktop applications to games and mobile applications);
- Language Features:

Features C# Shares with C++:

I. **(OOP):** Like C++, C# is an object-oriented language, supporting concepts like classes, objects, inheritance, polymorphism, and encapsulation.
II. **Strongly Typed:** Both C# and C++ are strongly typed, meaning types are checked at compile-time, ensuring type safety.
III. **Pointers (with Restrictions):** C# supports pointers only in unsafe code blocks. C++ allows pointers freely, while C# uses them more cautiously for memory safety.

Pointers can only be used in unsafe code blocks. unsafe is required because pointers directly manipulate memory.
Example:

```
using System;

class Program
{
    unsafe static void Main()
    {
        int number = 10;
        int* ptr = &number;  // Pointer to the variable 'number'

        Console.WriteLine($"Value of number: {number}");
        Console.WriteLine($"Address of number: {(ulong)ptr}"); // Display memory address
        Console.WriteLine($"Value using pointer: {*ptr}"); // Dereferencing pointer

        *ptr = 20; // Modify value through pointer
        Console.WriteLine($"Updated value of number: {number}");
    }
}
```

IV. **Exception Handling:** Both languages support try, catch, and finally blocks for exception handling.
V. **Namespaces:** C# and C++ use namespaces to organize code and avoid naming conflicts.

Features C# Shares with Java:

I. **Garbage Collection:** Like Java, C# uses automatic garbage collection to manage memory, which removes the need for manual memory management.

II. **Cross-Platform:** C# (via .NET Core and .NET 5+) and Java are designed to run on multiple platforms, making them cross-platform languages (they can run on Windows, Linux, and macOS)**.**

Why Are C# and Java Cross-Platform?

Feature	C# (.NET 6+)	Java (JVM)
Runtime	.NET Runtime	Java Virtual Machine
OS Support	Windows, Linux, macOS	Windows, Linux, macOS
Compilation	Intermediate Language (IL)	Bytecode
Execution	Runs on .NET runtime (Cross-Platform)	Runs on JVM (Cross-Platform)

Runtime:

- **C# (.NET 6+):** Uses the **.NET Runtime**, which is the environment responsible for running C# programs
- **Java (JVM)**: Java uses the **Java Virtual Machine (JVM)** to execute its code.

Compilation:

- **C#**: Compiles code into **Intermediate Language (IL)**, also known as CIL (Common Intermediate Language)**.**
- **Java**: Compiles code into bytecode by the JVM, which also uses JIT compilation to convert it into native machine code during execution.

III. **JIT (Just-In-Time) Compilation:** Both languages compile code to intermediate bytecode (C# uses Common Intermediate Language, or CIL, and Java uses bytecode), which is then compiled to native code at runtime.

IV. **Multithreading:** Both languages have built-in support for multithreading, making it easier to write concurrent applications.

V. **Interfaces and Abstract Classes:** C# and Java both support interfaces and abstract classes for defining method signatures that classes can implement or inherit.

Unique Features of C#:

I. **LINQ (Language Integrated Query):** LINQ in C# is to work with data access from data sources such as objects, data sets, SQL Server, and XML etc. It is a data querying API (Library). Like Query syntaxes in SQL. (query operator like-From, select, where, Orderby)

Example:

```
var numbers = new List<int> { 1, 2, 3, 4, 5 };
var result = numbers.Where(n => n > 3);
Console.WriteLine(string.Join(", ", result));  // Output: 4, 5
```

More,

int[] age={12,13,21,22,45}

var a=from i in age where i>20 select i;

```
foreach(int item in a)
{
Console.WriteLine(item);
}
Console.ReadLine();
```

I. **Delegates and Events:** Delegates contains the reference to several methods and call them when needed. So, we need to create number of methods as we need and attach these to delegates.

   - A delegate is a class that that encapsulates a method signature
   - 3 types of delegates

II. **Properties:** C# has built-in support for properties, providing a clean syntax for getting and setting fields. This is more concise compared to Java's getter and setter methods.

- **C# (Built-in, Concise Syntax)**: In C#, properties are a built-in feature. You can access them just like fields but still apply encapsulation and logic behind the scenes.

  Example:

```csharp
public class Person
{
    private string name;

    // Property with get and set accessors
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}
```

You can access it like a field:

```csharp
Person person = new Person();
person.Name = "John";  // Setting
Console.WriteLine(person.Name);  // Getting
```

- **Java (Requires Getters/Setters)**: In Java, you have to explicitly define getter and setter methods for each field, making it more verbose.

Example:

```
public class Person {
    private String name;

    // Getter method
    public String getName() {
        return name;
    }

    // Setter method
    public void setName(String name) {
        this.name = name;
    }
}
```

To access the property, you call the getter and setter methods:

```java
Person person = new Person();
person.setName("John");  // Setting
System.out.println(person.getName());  // Getting
```

- **Auto-Implemented Properties**

  C# (Auto-Properties): C# allows you to use auto-implemented properties. This means that C# automatically generates the backing field for you, reducing the need to explicitly define it.

  Example:

```
public class Person
{
    // Auto-implemented property (C# handles the backing field)
    public string Name { get; set; }
}
```

- **Java (No Auto-Properties)**: Java does not have auto-properties, so you need to manually create getter and setter methods.
  Examples:

```
public class Person {
    private String name;

    // Getter method
    public String getName() {
        return name;
    }

    // Setter method
    public void setName(String name) {
        this.name = name;
    }
}
```

I. **Generics:** Generics in C# allow you to define classes, methods, and data structures with type parameters that are specified at compile-time, rather than at runtime. This provides type safety (ensuring that you are working with the correct type) and performance benefits because the compiler knows the type at compile time

II. **Async/Await:** In C#, **async** and **await** keywords simplify asynchronous programming by allowing you to write code that

works asynchronously but looks and behaves like synchronous code. Asynchronous programming is useful when performing operations that might take time, like I/O-bound tasks (e.g., file reading, database access, or web requests), allowing the program to continue doing other tasks without getting blocked.

- **What is Asynchronous Programming?**
  - In synchronous programming, the program executes each line of code one by one. If one operation takes time (e.g., reading a file), it blocks the program from continuing until that operation is finished.
  - In **asynchronous programming**, you can perform long-running operations without blocking the program. It allows the program to do other work while waiting for the time-consuming task to complete.

- **Basic Syntax**
  - **async**: A method is marked with the ‘async’ keyword to indicate that it contains asynchronous operations.
  - **await**: The ‘await’ keyword is used to pause the method execution until the asynchronous operation completes.

Example:

```
using System;
using System.Threading.Tasks;

public class AsyncExample
{
    // Asynchronous method that simulates a time-consuming task
    public static async Task<string> GetDataAsync()
    {
        // Simulate a delay (e.g., waiting for data from a web API)
        await Task.Delay(2000);  // Asynchronously wait for 2 seconds
        return "Data received!";
    }

    public static async Task Main(string[] args)
    {
        // Call the asynchronous method and wait for the result
        string result = await GetDataAsync();
        Console.WriteLine(result);  // Output after 2 seconds: "Data received!"
    }
}
```

III. **Nullable Types:** C# supports nullable types for value types (e.g., int?), allowing you to represent missing or undefined values.

IV. **Indexers and Collections:**

- Indexers in C# allow an object to be accessed like an array. They use the ‘this’ keyword.

Example:

```
class StudentRecords
{
    private string[] students = new string[3];

    // Indexer
    public string this[int index]
    {
        get { return students[index]; }
        set { students[index] = value; }
    }
}

class Program
{
    static void Main()
    {
        StudentRecords record = new StudentRecords();

        record[0] = "Alice";
        record[1] = "Bob";
        record[2] = "Charlie";

        Console.WriteLine(record[0]); // Output: Alice
        Console.WriteLine(record[1]); // Output: Bob
        Console.WriteLine(record[2]); // Output: Charlie
    }
}
```

- **Why Use Indexers?**

  - Allows class objects to be accessed like arrays.
  - Simplifies data retrieval and assignment.

V. **Dynamic Type (Dynamic):** The dynamic type in C# allows you to store any data type without specifying its type at compile time. The type is resolved at runtime, meaning the compiler does not check for errors until the program runs.

Example:

```
using System;

class Program
{
    static void Main()
    {
        dynamic value = 10;  // Initially an integer
        Console.WriteLine(value); // Output: 10

        value = "Hello, World!"; // Now it's a string
        Console.WriteLine(value); // Output: Hello, World!

        value = 3.14; // Now it's a double
        Console.WriteLine(value); // Output: 3.14
    }
}
```

Comparison: `dynamic` vs `var`

Feature	`dynamic`	`var`
Type Checking	At runtime	At compile time
Flexibility	Can change type	Cannot change type
IntelliSense Support	Limited	Fully supported

Example:

csharp Copy Edit

```csharp
var num = 10; // Type is fixed as int
// num = "Hello"; // ❌ Error: Cannot change type

dynamic x = 10; // Type is int
x = "Hello";  // ✅ Allowed: Type changes to string
```

VI. **Extension Methods:** C# allows adding methods to existing types (including third-party libraries) without modifying them directly, enabling richer functionality.

VII. **Nullable Reference Types (C# 8.0+):** This feature helps identify and handle potential null reference exceptions at compile time, making C# safer and less prone to null-related bugs.

2) Why is C#?

C# is widely used because:

- **Powerful and Simple**: Combines the power of C/C++ and the simplicity of Visual Basic.
- **High Productivity**: C# comes with modern tools like Visual Studio**,** IntelliSense, and LINQ, which increase developer productivity.
- **Cross-Platform**: With .NET Core, C# can target multiple platforms.
- **Strong Typing**: Minimizes errors by checking types at compile time.
- **Integration with .NET Ecosystem**: C# integrates seamlessly with other Microsoft technologies, like Azure and Visual Studio.

3)Compilation and OS Interaction

Compilation:

C# source code is compiled into Intermediate Language (IL) code by the C# compiler (Roslyn). The IL is platform-independent and stored in an assembly (DLL or EXE).

Roslyn (csc.exe**)** – The official C# compiler developed by Microsoft.

How to Compile C# Code Using `csc.exe`?

If you have installed the **.NET SDK**, you can manually compile a C# program using:

sh Copy Edit

```sh
csc HelloWorld.cs
```

This will generate an **executable file** (`HelloWorld.exe`).

Alternative C# Compilers:

1. **Mono (mcs)** – Open-source C# compiler for cross-platform development.
2. **dotnet CLI (dotnet build)** – The official .NET CLI tool to compile and run .NET applications.
3. **Rider, Visual Studio, VS Code** – These IDEs use Roslyn under the hood.

C# code--→C# Code Compilation (Roslyn - csc.exe)----→ Intermediate Language(.dll/ .exe)----→ The .NET Runtime (CLR - Common Language Runtime) loads the IL code.---→ The JIT Compiler (Just-In-Time Compiler) converts IL into native machine code at runtime.-----→Machine code

Example:

```
public class Example
{
   public static void Main()
   {
      Console.WriteLine("Hello, C#!");
   }
}
```

This code will be compiled into IL.

Runtime:

The **Common Language Runtime (CLR)** handles the execution of IL code and converts it into native machine code for the specific platform.

OS Interaction: The .NET Runtime (CLR) interacts with the Operating System in several ways:

1. Memory Management

- The **Garbage Collector (GC)** automatically manages memory, preventing memory leaks.
- It interacts with the OS to allocate and free memory as needed.

2. Exception Handling & OS Signals

- CLR handles exceptions **cross-platform**, interacting with the OS to catch **system-level errors** (e.g., OutOfMemoryException, AccessViolationException).

3. File System Access

- CLR interacts with the **OS File System API** to read/write files (System.IO).
- It uses OS-specific paths (C:\ in Windows, /home/ in Linux).

4. Networking & Sockets

- The **System.Net** namespace interacts with OS networking APIs for HTTP, TCP, UDP communication.
- On Windows, it uses **Winsock**, while Linux/macOS use **BSD sockets**.
- Both **Winsock (Windows Sockets)** and **BSD Sockets** are networking APIs used for handling **TCP/IP communication** in operating systems.

4. **Interoperability (P/Invoke & COM)**

- .NET applications normally use **C# code**, but sometimes we need to call functions from **native C/C++ libraries** or interact with **Windows COM components**. This is where **P/Invoke and COM Interop** come in.
- **P/Invoke (Platform Invocation)** allows .NET code (C#) to **call functions from native C/C++ libraries** (like kernel32.dll on Windows or libc.so on Linux).

Why Use P/Invoke?

- To access **OS-level features** that .NET does not provide.
- To use **existing C libraries** instead of rewriting them in C#.
- For **better performance** in system-level tasks.

Example:

```
using System;

using System.Runtime.InteropServices;



class Program

{

    // Import MessageBox function from user32.dll

    [DllImport("user32.dll", CharSet = CharSet.Auto)]

    public static extern int MessageBox(IntPtr hWnd, string text, string
caption, int type);



    static void Main()
```

```
    {
        MessageBox(IntPtr.Zero, "Hello from P/Invoke!", "P/Invoke Example", 0);
    }
}
```

7. Security & OS Permissions

- Uses **Windows API (ACL, UAC)** and **Linux/macOS permissions** for security.
- Handles cryptography using **OS libraries** (bcrypt.dll on Windows, OpenSSL on Linux).

8. Process & Environment Interaction

- Uses System.Diagnostics.Process to start/manage OS processes.
- Reads environment variables via System.Environment.

4) Frameworks

◆ What Is a Framework?

A **framework** is **not just a shorter version of a language** — it's more like a **pre-defined structure and tools** that help you build applications **faster, more consistently, and with less code duplication**.

Think of it like a **template with rules and helper tools** that guide how your code should be written and structured.

🔸 Example 1: Tailwind CSS vs Raw CSS

- **Raw CSS**: You write every style manually:

```css
.btn {
  background-color: blue;
  color: white;
  padding: 10px;
  border-radius: 5px;
}
```

- **Tailwind CSS**: You use utility classes directly in HTML:

```html
<button class="bg-blue-500 text-white px-4 py-2 rounded">Click</button>
```

✔️ Why it's shorter?

Tailwind is a utility-first **framework** that provides **predefined CSS classes** to avoid repeating styles manually. It doesn't reduce CSS as a language but gives you tools to write it efficiently.

🔸 Example 2: .NET vs C#

- **C#**: The **programming language** you write code in (like class, if, foreach, etc.).
- **.NET Framework / .NET Core / .NET 6+**: A **framework** that includes:
  - Libraries (like Entity Framework, ASP.NET, etc.)
  - Runtime environment

    - Base Class Libraries (collections, threading, I/O, etc.)

✔️ **Why it's not a short version of C#?**
Because **.NET is not about reducing code length**, it's about **providing tools, patterns, and infrastructure** to make your application scalable and maintainable. You still write C# code, but .NET gives you everything else needed to run your C# code effectively.

🧠 **Summary: Language vs Framework vs Library**

Concept	Example	What it Does
Language	C#, CSS	The syntax and rules to write code
Framework	.NET, Tailwind CSS	Gives structure, tools, and pre-built logic
Library	jQuery, Entity Framework	Pre-written code you can use when needed

---

🛠️ **In Your Project: SkillExchangeMVC**

- **C#** is the language you use for:
    - Writing controllers (CourseController.cs)
    - Writing models (Course.cs)
    - Writing business logic
- **.NET MVC Framework** gives:
    - Model-View-Controller architecture
    - Routing system
    - Form validation
    - Authentication system
    - Built-in helpers like @Html.LabelFor, @Html.TextBoxFor
- **Tailwind CSS** (if you use it) gives:

    - Utility classes like text-xl, bg-blue-500, rounded, etc.
    - Less need to write custom CSS classes

◇ 1. What is .NET?

- **.NET** is a **developer platform** (a big toolbox) made by Microsoft.
- It includes **languages** (like C#), **libraries**, and **runtimes**.
- It's used to build apps for:
    - Web
    - Desktop
    - Mobile
    - Cloud
    - Games
    - IoT

🧩 So What Are These "Sub-Frameworks"?

Each item you listed is a **specific framework within the .NET ecosystem**, made for different kinds of apps:

☑ .NET Framework

- 📅 Oldest one (since 2002)
- 🖥️ **Windows-only**
- 🛠️ Used for:
    - WinForms, WPF apps
    - ASP.NET (not Core)
- ❌ Not cross-platform
- 🔚 Microsoft is no longer adding big new features (maintenance only)

☑ .NET Core / .NET 5+

- 🚀 New, modern, fast, **cross-platform**
- 🔁 Replaced .NET Framework starting with **.NET 5 (2020)** and beyond
- ☑ Runs on Windows, Linux, macOS
- 🛠 Used for:
  - Console apps
  - Web apps
  - APIs
  - Cloud apps

📌 You are using this in **SkillExchangeMVC** because you said you're using **.NET 6**.

☑ ASP.NET Core

- 🌐 A web framework **built on top of .NET Core**
- 🛠 Used to build:
  - Websites (like Razor Pages or MVC)
  - APIs
  - Real-time apps (with SignalR)

✔ Your project **SkillExchangeMVC** is built with **ASP.NET Core + MVC + Razor Pages**, so this is what you're directly using!

☑ Xamarin

- 📱 Used to build **mobile apps**

- Write code in C#, and run it on **Android & iOS**
- Part of .NET ecosystem
- ☑ Good if you want to build apps for phones with .NET skills

ℹ Note: Xamarin is being replaced by **.NET MAUI**, which is even more powerful and supports desktop too.

🧠 Summary Table:

Framework	Purpose	Platform	Your Use?
.NET Framework	Windows apps only (legacy)	Windows	❌ Not used
.NET Core / .NET 6+	Cross-platform, modern .NET	Windows, Linux, macOS	☑ Yes
ASP.NET Core	Web apps, APIs	Cross-platform	☑ Yes (MVC/Razor)
Xamarin	Mobile apps (Android, iOS)	Cross-platform	❌ Not used

☑ Final Answer for You

So:

- **.NET is the main platform**
- **Each framework under it** is like a specialized toolkit:
  - ASP.NET Core → web
  - Xamarin → mobile
  - .NET Core → console, web, etc.
  - .NET Framework → old Windows-only apps

You are using:
☑ **ASP.NET Core + .NET 6** (part of .NET Core evolution)
❌ Not using Xamarin or legacy .NET Framework

Comparison of .NET Framework, .NET Core, ASP.NET Core, and Xamarin:

1. Platform Support (Where They Run)

.NET Framework	.NET Core / .NET 5+	ASP.NET Core	Xamarin
Windows-only	Cross-platform (Windows, Linux, macOS)	Cross-platform	Cross-platform (iOS, Android, Windows)

Explanation:

- **.NET Framework** runs **only on Windows**.
- **.NET Core / .NET 5+** is **cross-platform**, meaning it runs on **Windows, Linux, and macOS**.
- **ASP.NET Core** is also cross-platform, so you can **deploy web applications on Windows, Linux, or macOS**.
- **Xamarin** allows you to build **mobile applications** that work on **iOS, Android, and Windows** using **C#**.

📌 **Example:**
If you want to run a .NET app on **Linux**, you **cannot use .NET Framework**. Instead, you should use **.NET Core / .NET 6/7**.

---

2. Use Case (What It's Used For)

.NET Framework	.NET Core / .NET 5+	ASP.NET Core	Xamarin
Windows desktop, old ASP.NET web apps	Modern desktop, cloud, microservices	Web apps & APIs	Mobile apps (iOS & Android)

Explanation:

- **.NET Framework** is good for **legacy applications** like **WinForms, WPF, and old ASP.NET apps.**
- **.NET Core / .NET 5+** is used for **modern applications, cloud apps, and microservices.**
- **ASP.NET Core** is the best choice for **web apps, APIs, and Blazor applications.**
- **Xamarin** is used to build **mobile apps** for **iOS and Android** with **C#**, instead of using Swift/Java.

📌 **Example:**
If you want to build a **high-performance web API**, use **ASP.NET Core**, not .NET Framework.
If you want to create an **Android/iOS app**, use **Xamarin**, not .NET Core.

---

3. Performance

.NET Framework	.NET Core / .NET 5+	ASP.NET Core	Xamarin
Slower (older architecture)	Faster (optimized runtime)	High-performance (better than .NET Framework)	Near-native performance

Explanation:

- **.NET Framework** is **slower** because it carries **legacy code** and is **not optimized** for modern hardware.
- **.NET Core / .NET 5+** is designed for **speed** and **efficiency**, using a new **runtime (CoreCLR)**.
- **ASP.NET Core** is **significantly faster** than the old ASP.NET because of **optimized request handling**.
- **Xamarin** compiles apps to **native code**, so performance is **almost as good as Swift (iOS) and Java/Kotlin (Android)**.

📌 **Example:**
A web API built with **ASP.NET Core** will be **much faster** than one built with **.NET Framework**.

---

4. Support for Modern Development

.NET Framework	.NET Core / .NET 5+	ASP.NET Core	Xamarin
Limited (old technology)	Fully supports modern development	Fully supports modern development	Fully supports modern development

Explanation:

- **.NET Framework** is **outdated** and does not support **new features** like **containerization (Docker, Kubernetes).**
- **.NET Core / .NET 5+** supports **modern cloud development**, **microservices**, and **Docker.**
- **ASP.NET Core** is a **modern framework** that supports **Blazor, APIs, and cloud.**
- **Xamarin** supports **modern mobile development** and integrates with **.NET MAUI (next-gen Xamarin).**

📌 **Example:**
If you are starting a **new project**, you should **not use .NET Framework.** Instead, choose **.NET 6/7** or **ASP.NET Core** for web applications.

---

5. Deployment (How Easy It Is to Deploy)

.NET Framework	.NET Core / .NET 5+	ASP.NET Core	Xamarin
Requires full .NET installation	Self-contained, lightweight	Self-contained, lightweight	Compiles to native apps

Explanation:

- **.NET Framework** requires a **full installation** of .NET on the target machine.
- **.NET Core / .NET 5+** allows **self-contained deployment**, meaning you can ship your app **without requiring .NET to be installed.**
- **ASP.NET Core** web apps can run as **self-contained applications** on Windows/Linux.
- **Xamarin** compiles directly to **native iOS and Android apps**, so there's **no dependency** on .NET at runtime.

📌 **Example:**
If you want to **deploy a .NET app to Linux**, use **.NET Core / .NET 6+**, **not .NET Framework.**

---

6. Future of Development

.NET Framework	.NET Core / .NET 5+	ASP.NET Core	Xamarin
Deprecated (No new updates)	Recommended for all new development	Recommended for web apps	Recommended for mobile apps

Explanation:

- **.NET Framework** is **deprecated**—Microsoft **will not release new versions** (last version is **.NET Framework 4.8**).
- **.NET Core / .NET 5+** is the **future** of .NET. All new projects should use **.NET 6 or later.**
- **ASP.NET Core** is the **preferred web framework** for all new projects.
- **Xamarin** is **being replaced by .NET MAUI** (the next generation of cross-platform mobile development).

📌 **Example:**
If you're starting a new **Windows desktop app**, use **.NET 6/7, not .NET Framework.**
For new **mobile apps**, consider **.NET MAUI** instead of Xamarin.

---

7. Example Use Cases

.NET Framework	.NET Core / .NET 5+	ASP.NET Core	Xamarin
WinForms, WPF, old ASP.NET apps	.NET 6/7 apps, cloud, microservices	Blazor, REST APIs, Web Apps	iOS & Android apps with C#

Explanation:

- Use **.NET Framework** if you are **maintaining old software** that cannot be migrated.
- Use **.NET Core / .NET 5+** for **modern desktop, cloud, or enterprise applications.**
- Use **ASP.NET Core** for **fast web applications and APIs.**
- Use **Xamarin** (or **.NET MAUI**) for **mobile development.**

📌 **Example:**
If you are building a **real-time web API**, choose **ASP.NET Core**, not .NET Framework.
If you want to build a **Windows desktop app**, use **.NET 6/7 (WinForms/WPF)**, not .NET Framework.

🚀 Conclusion: Which One Should You Use?

✅ For Windows desktop apps → Use .NET 6/7 (WinForms/WPF)
✅ For cross-platform apps (Windows, Linux, macOS) → Use .NET 6/7
✅ For web applications & APIs → Use ASP.NET Core
✅ For mobile apps (iOS, Android) → Use Xamarin (or .NET MAUI for the future)

🧩 When Did You "Install" the Framework?

☑ The framework is installed when you install the .NET SDK, not when you start a project or add NuGet packages.

📌 **Here's the breakdown:**

🟦 **1. When You Installed .NET SDK**

- The .NET Framework or .NET 6/7 SDK is installed system-wide.
- It includes:
  - The runtime (to run .NET apps)
  - Base class libraries
  - Project templates (like ASP.NET MVC, Razor, etc.)
- This is the **main framework** your project will use.

🧠 **This happens BEFORE you even open Visual Studio**.

🟧 **2. When You Created the Project in Visual Studio**

- You probably selected:
  ☑ "ASP.NET Core Web App (Model-View-Controller)" or
  ☑ "ASP.NET Web Application (.NET Framework)"

🔧 At this point:

- Visual Studio **uses the installed .NET SDK**
- It creates a project with:
  - Controller folder
  - Views folder
  - wwwroot for static files
  - A Program.cs or Startup.cs
  - A .csproj file that references the framework

📁 Example in .csproj:

xml

CopyEdit
<TargetFramework>net6.0</TargetFramework>

So you're **using** the framework here, but not "installing" it — because it was already installed with the SDK.

🟩 3. When You Install NuGet Packages

- NuGet packages **are not the framework itself**, but **additional libraries** or tools.
- You use them to:
  - Add support for EF Core, Identity, SendGrid, etc.
  - Extend the functionality of your app

📦 Examples:

- Microsoft.EntityFrameworkCore
- Microsoft.AspNetCore.Identity.EntityFrameworkCore
- AutoMapper
- MailKit

🧠 So these **add features**, but don't install the base framework.

☑️ Final Summary

Action	What Happens
🔵 Install .NET SDK	☑️ Installs the .NET Framework (runtime, libraries, tools)
🟡 Create project in Visual Studio	☑️ Uses the installed framework to generate boilerplate

Action	What Happens
🟢 Install NuGet packages	☑ Adds optional tools/libraries (not the framework itself)

Want to Check Which Framework You're Using?

1. Open your `.csproj` file:

```xml
<TargetFramework>net6.0</TargetFramework>
```

This tells you the .NET version your app uses.

2. Use terminal or PowerShell:

```bash
dotnet --list-sdks
```

This shows all installed .NET versions.

5) .NET Standard

.NET Standard defines a set of APIs that all .NET implementations must support. It allows code sharing between different .NET platforms (e.g., .NET Framework, .NET Core, Xamarin).

This is the power of **.NET Standard**: you write your code once and use it across multiple .NET platforms, making code-sharing much easier and more efficient.

6) Runtimes

🧠 What is a Runtime?

- The **runtime** is the engine that **executes your .NET code**.
- It includes:
  - **Base class libraries**
  - **Memory management**
  - **Garbage collection**
  - **JIT compilation**

When you run your app, this runtime is what powers it under the hood.

☑️ Step 1: Check the .csproj File

Open your project's .csproj file (usually named like SkillExchangeMVC.csproj) — then look for this line:

xml
CopyEdit
<TargetFramework>net6.0</TargetFramework>

This line tells you:

- ☑️ You're using **.NET 6 Runtime**
- 🔁 Which is based on **.NET Core**, not the old .NET Framework

---

☑️ Step 2: CLI Confirmation (Optional)

If you're using Visual Studio or a terminal, you can run:

dotnet --list-runtimes

This will list all installed runtimes on your PC, e.g.:

Microsoft.NETCore.App 6.0.25 [C:\Program Files\dotnet\shared\Microsoft.NETCore.App]

Microsoft.AspNetCore.App 6.0.25 [C:\Program Files\dotnet\shared\Microsoft.AspNetCore.App]

That confirms:

- ☑ You're using .NET 6 Runtime
- ☑ And also using **ASP.NET Core Runtime**

- **CLR (Common Language Runtime)**: Used with the .NET Framework.
- **CoreCLR**: Used with .NET Core for cross-platform execution.
- **Mono**: A runtime used for mobile and other cross-platform development.
- **.NET Core Runtime**: This is the entire environment that includes (CoreCLR and CoreFx).

7) .NET Other Topics

Dependency Injection (DI):

What it is:

Dependency Injection is a design pattern used to achieve loose coupling between classes by injecting the required objects (dependencies) from outside rather than creating them inside the class.

Why it's useful:
It makes your code more flexible, easier to test, and easier to maintain. You don't need to change a class whenever you need a new version of its dependencies.

Example:

Without DI:

```
public class CategoryController : Controller
 {
     Var Context = new ApplicationDbContext();

 }
```

With DI:

```
public class CategoryController : Controller
 {
     private readonly ApplicationDbContext _context;
     public CategoryController(ApplicationDbContext context)
     {
         _context = context;

     }

}
```

<u>Entity Framework (EF):</u>

- It is an Object-Relational Mapping (ORM) tool that simplifies database operations.
- It allows you to interact with a database using C# objects, instead of writing SQL queries directly. EF automatically translates between your C# classes and database tables.

Example: Let's say you have a Student class and a Student table in your database

```csharp
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Using EF, you can easily query the database like this:

```csharp
using (var context = new SchoolContext())
{
    var students = context.Students.ToList();  // No SQL needed!
    foreach (var student in students)
    {
        Console.WriteLine(student.Name);
    }
}
```

NuGet:

- A package manager for .NET libraries.
- It helps you download and manage libraries (pre-written code) that you can use in your own projects.
- These libraries can save you time by providing ready-made solutions for common tasks.

Blazor:

- A UI framework.
- A framework for building interactive web UIs using C# instead of JavaScript.

- It enables .NET developers to use their existing C# skills to build modern web applications without needing to learn JavaScript.

Example: Here’s a simple Blazor component that displays a button and a counter:

```
@page "/counter"

<h3>Counter</h3>

<p>Current count: @count</p>

<button @onclick="IncrementCount">Increment</button>

@code {
    private int count = 0;

    private void IncrementCount()
    {
        count++;
    }
}
```

In this example:

- You use C# (@code block) to manage the state and logic.
- Blazor handles rendering the UI and user interaction (like clicking the button) directly in the browser.

Key Differences:

Feature	Razor Pages	Blazor
Type	Templating Engine	UI Framework
Used in	ASP.NET MVC & Razor Pages	Blazor Server & Blazor WebAssembly
Language	C# & HTML	C# & HTML
Rendering	Server-Side (Generates HTML)	Client-Side (SPA-like) or Server-Side
Interactivity	Uses JavaScript for dynamic behavior	Uses C# for interactivity

Which One to Use?

- Use **Razor Pages** if you need a traditional request/response model (like MVC).
- Use **Blazor** if you want a **modern, interactive UI** with C# instead of JavaScript.

Since you're working on **ASP.NET with Razor Pages**, you're using Razor right now. But if you want to build **SPA-like experiences**, you can consider using **Blazor** in the future.

◇ What is Middleware?

☑ Definition:

Middleware is a piece of code that sits in the request pipeline and processes HTTP requests and responses **before they reach your controller** or **after the controller processes them**.

🧠 Think of it as:

A **security guard or checkpoint** that inspects or modifies the request **on its way in** and the response **on its way out**.

Browser → Middleware (e.g., Authentication) → Routing → Controller → Action → View → Middleware → Response to Browser

🛠️ Common Middleware in SkillExchangeMVC:

In `Program.cs` or `Startup.cs`, you may see middleware setup like this:

csharp Copy Edit

```csharp
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
app.UseStaticFiles();
app.UseEndpoints(endpoints => {
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

These do things like:

- `UseRouting()` – sets up routing for your app.
- `UseAuthentication()` – checks if the user is logged in.
- `UseAuthorization()` – checks if the user has permission.
- `UseStaticFiles()` – allows serving CSS, JS, images.
- Custom Middleware (optional) – e.g., logging, error handling, IP filtering.

◇ What is Routing?

☑ Definition:

Routing is how ASP.NET MVC **maps a URL to a specific controller and action**.

📌 For Example:

When you visit:

https://localhost:5001/Course/ViewCourses

ASP.NET MVC uses routing to:

- Call the CourseController
- Run the ViewCourses action method
- Render the ViewCourses.cshtml page

✨ Route Pattern in Your Project

In `Program.cs` or `Startup.cs`:

csharp Copy Edit

```csharp
endpoints.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

🔍 This pattern means:

- Default controller = `Home`
- Default action = `Index`
- Optional `id` parameter (like a course ID)

So:

URL	Maps to
`/`	`HomeController.Index()`
`/Course/AddCourse`	`CourseController.AddCourse()`
`/Account/Register`	`AccountController.Register()`

Feature	Middleware	Routing
Purpose	Handle request/response in the pipeline	Direct the request to the correct controller/action
Example	Authentication, logging, error handling	/Course/AddCourse → CourseController.AddCourse()
Where	Program.cs or Startup.cs	Also in Program.cs / Startup.cs
Runs	Before/after controller	Before controller is selected

◇ 1. What is an API? (Simple Answer)

API stands for **Application Programming Interface**.

It's like a **waiter in a restaurant** — you (the customer) place an order (a request), and the waiter (API) brings your food (the response) from the kitchen (the server).

☑ In technical terms:

An **API** lets two applications **talk to each other**.

- You → send a request to an API
- API → gives you back data or performs an action

🔵 2. Real Life Examples of APIs

Real Life Task	API Used Behind the Scenes
🌤️ Show weather on a website	Uses **Weather API** (e.g., OpenWeatherMap)
💳 Payment on an e-learning site	Uses **Payment API** (e.g., SSLCommerz, Stripe)
✉️ Send password reset email	Uses **Email API** (e.g., SendGrid, MailKit)
🔍 Search in a course catalog	May use **internal API** returning course data
📱 Mobile app connects to server	Uses **Web API** to fetch/post data

---

🔵 3. How to Use an API? (Simple Steps)

☑️ A. If You're Calling an API (Client-side)

1. **Find the API URL** (e.g., https://api.weather.com/current)
2. **Send a request** using HttpClient in C# or fetch() in JavaScript
3. **Get the response** (usually JSON)
4. **Use the data** in your app

💻 Example (C#):

```csharp
HttpClient client = new HttpClient();
var response = await client.GetAsync("https://api.weatherapi.com/current");
string data = await response.Content.ReadAsStringAsync();
// Use the data (usually JSON)
```

✅ B. If You're **Creating** an API (Server-side in your app)

1. Create a controller:

```csharp
[ApiController]
[Route("api/[controller]")]
public class CourseApiController : ControllerBase
{
    [HttpGet]
    public IActionResult GetCourses()
    {
        var courses = courseService.GetAll();
        return Ok(courses);
    }
}
```

2. Call it from frontend:

```js
fetch("/api/course")
  .then(res => res.json())
  .then(data => console.log(data));
```

🟦 4. In Your Project (SkillExchangeMVC)

You **might use an API** for:

- Allowing mobile apps to fetch course lists

- JavaScript to dynamically load course data
- Integrating payment (SSLCommerz API)
- Sending email for forgot password (MailKit or SMTP)

☑ Summary (Easy Table)

Question	Answer
❓ What is API?	A way for apps to talk to each other
🧠 Real use?	Weather, email, payment, data from database
🛠️ How to use it?	Use HttpClient to call an API / Build your own with [ApiController]
💡 In your project?	You **can** use APIs for login, course data, email, etc.

🔸 ViewData, ViewBag, TempData:

These are used in ASP.NET Core MVC to **store and pass data from controller to view**, or sometimes between **requests or controllers**.

☑ ViewBag:

- ViewBag is an object used to store data using key-value pairs.
- It **contains both the data and its data type**, thanks to dynamic typing.
- You **don't need to cast** the data while retrieving.
- It is used to **pass data from controller to view** within the **same request**.
- It only works for: **Same controller to same view** (i.e., it does **not** persist across redirects).

🔷 ViewBag দিয়ে ডেটা পাঠানো ও দেখানো

✅ Step 1: Controller এ ViewBag ব্যবহার

csharp Copy Edit

```csharp
public IActionResult IndexCategory()
{
    var categories = _context.Categories.ToList();

    ViewBag.Message = "ViewBag দিয়ে পাঠানো মেসেজ!";

    return View(categories);
}
```

✅ Step 2: View এ ViewBag থেকে ডেটা দেখা

cshtml Copy Edit

```cshtml
<p class="text-success">@ViewBag.Message</p>
```

> 🔹 ViewBag internally ViewData ই ব্যবহার করে, তবে dynamic টাইপ হওয়ায় cast করা লাগে না।

☑ ViewData

- ViewData is similar to ViewBag but has a few differences.
- It also stores data as key-value pairs.
- However, **it stores only the data**, and **not the data type** — so you often need to **cast** the data when using it.
- Like ViewBag, it also works **within the same request**, from **controller to view**.

Controller থেকে ViewData দিয়ে কিছু ডেটা পাঠাবো → View-তে সেটা দেখাবো।

Step 1: Controller-এ ViewData ব্যবহার

csharp

Copy Edit

```csharp
public IActionResult IndexCategory()
{
    var categories = _context.Categories.ToList();

    // ViewData দিয়ে মেসেজ পাঠাচ্ছি
    ViewData["Message"] = "এই পেজে সব ক্যাটেগরি দেখা যাবে।";

    return View(categories);
}
```

Step 2: View (IndexCategory.cshtml)-এ ViewData থেকে ডেটা নেয়া

c

Copy Edit

```c
@model IEnumerable<Practice.Models.Category>

@{
    ViewData["Title"] = "View Categories";
}

<h2>@ViewData["Title"]</h2>

<!-- ViewData দিয়ে পাঠানো মেসেজ দেখানো -->
<p class="text-info">@ViewData["Message"]</p>

<p>
    <a asp-action="CreateCategory" class="btn btn-primary">Create New Category</a>
</p>

<table class="table table-striped">
    <thead>
```

❗ Bonus:

যেহেতু ViewData টাইপ-নির্ভর না (non-generic object), তাই যদি তুমি string না পাঠাও, তাহলে View-তে ব্যবহার করার সময় type casting করতে হতে পারে।

🧪 যদি int পাঠাও:

Controller:

```csharp
ViewData["Total"] = 10;
```

View:

```cs
<p>মোট ক্যাটেগরি: @(int)ViewData["Total"]</p>
```

☑ TempData

- TempData is used to **store data between requests**, and even **across different controllers**.
- It's useful when you need to **redirect** and still pass data.
- TempData is internally backed by **Session**, so it can **persist across a single redirect**.
- It **retains both key and value**, but like ViewData, you need to **cast** the data when retrieving.

🔷 TempData দিয়ে ডেটা পাঠানো ও দেখানো

✅ Step 1: Controller থেকে TempData সেট করো

csharp Copy Edit

```csharp
[HttpPost]
public IActionResult CreateCategory(Category category)
{
    _context.Categories.Add(category);
    _context.SaveChanges();

    TempData["Success"] = "নতুন ক্যাটেগরি সফলভাবে যুক্ত হয়েছে!";

    return RedirectToAction("IndexCategory");
}
```

✅ Step 2: Target View (IndexCategory.cshtml) এ TempData থেকে ডেটা দেখাও

cshtml Copy Edit

```cshtml
@if (TempData["Success"] != null)
{
    <div class="alert alert-success">
        @TempData["Success"]
    </div>
}
```

> - TempData একবার ব্যবহার করার পর অটো রিমুভ হয়ে যায়।
> - এটা redirect করার পরেও data ধরে রাখে (unlike ViewBag/ViewData).