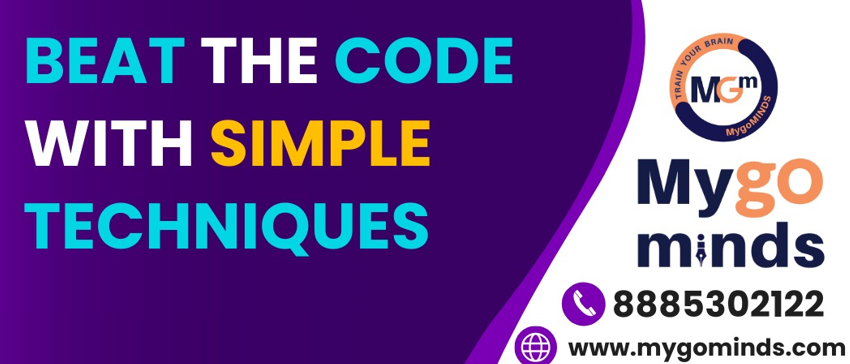


****

**Chapter-1**

**Introduction to .NET Core**

**📘 Introduction to .NET Core**

**🔑 What is .NET Core?**

.NET Core is a free, open-source, cross-platform framework developed by Microsoft for building modern, cloud-based, and internet-connected applications. It is a lightweight and modular platform used to develop applications that run on Windows, macOS, and Linux.

**🔑 Features of .NET Core**

* Cross-platform support
* High performance and scalability
* Flexible deployment (framework-dependent or self-contained)
* Command-line tools for development
* Modular architecture with NuGet packages
* Supports microservices and containerization (e.g., Docker)

**🔑 .NET Core vs .NET Framework**

| **Feature** | **.NET Core** | **.NET Framework** |
| --- | --- | --- |
| Platform Support | Cross-platform (Windows, macOS, Linux) | Windows only |
| Open Source | Yes | Partially |
| Deployment Model | Flexible (self-contained or framework) | Only framework-dependent |
| Microservices | Fully supported | Limited |
| Performance | High | Moderate |

**🔧 Installation and Setup of .NET Core SDK**

1. Go to: <https://dotnet.microsoft.com>
2. Download the latest .NET Core SDK.
3. Install it by running the installer.
4. Verify installation:

dotnet --version

**🔧 Creating First .NET Core Application**

1. **Open Command Prompt or Terminal**
2. **Create a new console project:**

dotnet new console -n HelloDotNetCore

1. **Navigate to project folder:**

cd HelloDotNetCore

1. **Run the application:**

dotnet run

📝 **Program.cs**

using System;

namespace HelloDotNetCore

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello, .NET Core!");

}

}

}

✅ **Multiple Choice Questions (MCQs)**

1. What is .NET Core?
   * A. A mobile OS
   * B. A database management system
   * C. A cross-platform development framework
   * D. A text editor  
     **Answer: C**
2. Which company developed .NET Core?
   * A. Google
   * B. Apple
   * C. Microsoft
   * D. IBM  
     **Answer: C**
3. Which of the following is a key feature of .NET Core?
   * A. Windows-only support
   * B. Closed source
   * C. Cross-platform
   * D. Manual memory management  
     **Answer: C**
4. Which command is used to check .NET Core version?
   * A. dotnet -v
   * B. dotnet version
   * C. dotnet --version
   * D. check .net  
     **Answer: C**
5. Which file contains the main logic of a console app?
   * A. main.cs
   * B. app.cs
   * C. Program.cs
   * D. index.cs  
     **Answer: C**
6. .NET Core applications can run on:
   * A. Windows
   * B. macOS
   * C. Linux
   * D. All of the above  
     **Answer: D**
7. Which of the following allows installing .NET Core packages?
   * A. GitHub
   * B. NuGet
   * C. npm
   * D. pip  
     **Answer: B**
8. What does SDK stand for in .NET Core?
   * A. System Development Kit
   * B. Software Deployment Kit
   * C. Software Development Kit
   * D. Source Development Kit  
     **Answer: C**
9. What is the extension of a .NET Core project file?
   * A. .proj
   * B. .coreproj
   * C. .csproj
   * D. .dotproj  
     **Answer: C**
10. Which method is the entry point in a .NET Core console application?
    * A. Init()
    * B. Start()
    * C. Main()
    * D. Begin()  
      **Answer: C**
11. What is the purpose of the dotnet new command?  
    o A. Run the application  
    o B. Restore packages  
    o C. Create a new project  
    o D. Publish the app  
    **Answer: C**
12. Which of the following template names is used to create a console app using CLI?  
    o A. mvc  
    o B. web  
    o C. console  
    o D. razor  
    **Answer: C**
13. Which command compiles the application in .NET Core?  
    o A. dotnet compile  
    o B. dotnet run  
    o C. dotnet build  
    o D. dotnet generate  
    **Answer: C**
14. What is the default web server used in .NET Core?  
    o A. IIS  
    o B. Tomcat  
    o C. Apache  
    o D. Kestrel  
    **Answer: D**
15. What is the purpose of appsettings.json in .NET Core projects?  
    o A. To store UI layout  
    o B. To manage source code  
    o C. To store configuration settings  
    o D. To write database queries  
    **Answer: C**
16. Which of the following is NOT a valid .NET Core CLI command?  
    o A. dotnet new  
    o B. dotnet build  
    o C. dotnet add  
    o D. dotnet code  
    **Answer: D**
17. Which package manager is used for managing dependencies in .NET Core?  
    o A. npm  
    o B. NuGet  
    o C. Maven  
    o D. Gradle  
    **Answer: B**
18. How do you run a .NET Core application from the terminal?  
    o A. dotnet execute  
    o B. dotnet build  
    o C. dotnet start  
    o D. dotnet run  
    **Answer: D**
19. Which command is used to create a web API project in .NET Core CLI?  
    o A. dotnet new mvc  
    o B. dotnet new webapi  
    o C. dotnet new web  
    o D. dotnet new restapi  
    **Answer: B**
20. Which runtime does .NET Core use?  
    o A. CLR  
    o B. JVM  
    o C. CoreCLR  
    o D. Mono  
    **Answer: C**

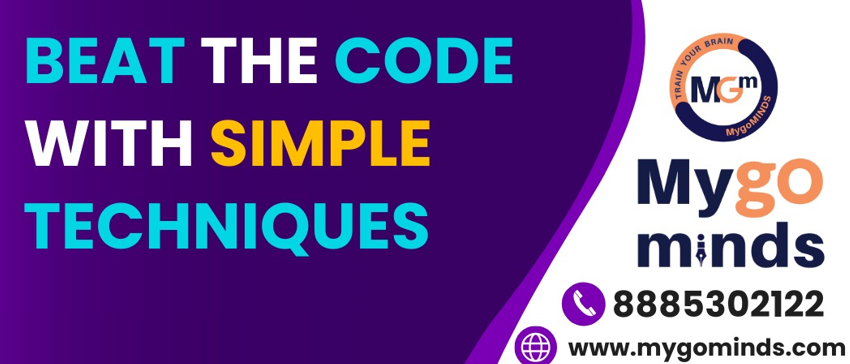
✍️ **Fill in the Blanks**

1. .NET Core is a \_\_\_\_\_\_\_\_\_\_ platform development framework.  
   **Answer: cross**
2. The command to create a new console project is dotnet new \_\_\_\_\_\_\_\_\_\_.  
   **Answer: console**
3. The main class file is named \_\_\_\_\_\_\_\_\_\_.  
   **Answer: Program.cs**
4. To run a project, the command is dotnet \_\_\_\_\_\_\_\_\_\_.  
   **Answer: run**
5. The \_\_\_\_\_\_\_\_\_\_ CLI is used to work with .NET Core projects.  
   **Answer: dotnet**
6. The project configuration file has a \_\_\_\_\_\_\_\_\_\_ extension.  
   **Answer: .csproj**
7. \_\_\_\_\_\_\_\_\_\_ is the package manager used in .NET Core.  
   **Answer: NuGet**
8. The Main method must be declared as \_\_\_\_\_\_\_\_\_\_.  
   **Answer: static**
9. .NET Core can be used to create \_\_\_\_\_\_\_\_\_\_-based applications.  
   **Answer: console**
10. The .NET Core SDK includes tools and \_\_\_\_\_\_\_\_\_\_ for development.  
    **Answer: libraries**

✅  **True or False**

1. .NET Core is a paid platform.  
   **Answer: False**
2. .NET Core applications can only run on Windows.  
   **Answer: False**
3. .NET Core supports Docker containers.  
   **Answer: True**
4. You can create web applications using .NET Core.  
   **Answer: True**
5. The file Program.cs contains the Main() method.  
   **Answer: True**
6. .NET Core doesn't support command-line tools.  
   **Answer: False**
7. .NET Core is open-source.  
   **Answer: True**
8. .NET Framework is newer than .NET Core.  
   **Answer: False**
9. .NET Core applications must always be deployed on IIS.  
   **Answer: False**
10. NuGet is used to manage dependencies in .NET Core.  
    **Answer: True**



****

**Chapter-2**

**Folder Structure of .Net Core**

**📘 Folder Structure of .NET Core Project**

The structure can vary slightly depending on the type of application (console, web API, MVC, etc.), but here’s a **typical structure for a .NET Core Web Application**:

MyDotNetApp/

│

├── Controllers/ --> Contains controller classes (MVC/Web API)

│

├── Models/ --> Holds domain or data model classes

│

├── Views/ --> Contains Razor views (MVC only)

│ ├── Shared/ --> Shared layout views (\_Layout.cshtml)

│ └── Home/ --> Views related to the HomeController

│

├── wwwroot/ --> Web root folder for static files (css, js, images)

│ ├── css/

│ ├── js/

│ └── images/

│

├── obj/ --> Temporary object files (auto-generated)

│

├── bin/ --> Binary output (compiled files)

│

├── Properties/ --> Contains launchSettings.json (for app config)

│

├── appsettings.json --> Application settings and configuration

│

├── Program.cs --> Entry point of the application

│

├── Startup.cs --> Configuration for services and middleware

│

└── MyDotNetApp.csproj --> Project file that defines dependencies and settings

**🔑 Key Folders Explained**

**🔧 Controllers/**

Holds all controller classes which handle incoming requests.

public class HomeController : Controller

{

public IActionResult Index()

{

return View();

}

}

**🔧 Models/**

Contains C# classes that represent the data or business logic.

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

}

**🔧 Views/**

Contains Razor (.cshtml) files used for UI rendering.

* Shared/: Common layout files like \_Layout.cshtml
* Home/: Views related to HomeController like Index.cshtml

**🔧 wwwroot/**

Public web assets like CSS, JS, images, etc. Only folder accessible to browser.

**🔧 Program.cs**

Main entry point of the app, sets up the host and web server.

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

}

**🔧 Startup.cs**

Defines middleware, routing, and services used in the app.

public class Startup

{

public void ConfigureServices(IServiceCollection services) { }

public void Configure(IApplicationBuilder app, IWebHostEnvironment env) { }

}

**🔧 appsettings.json**

Stores configuration values like database connection strings and logging settings.

{

"ConnectionStrings": {

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

}

}

**🔧 .csproj**

Project file that includes metadata and NuGet package references.

<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>

<TargetFramework>netcoreapp3.1</TargetFramework>

</PropertyGroup>

</Project>

**📘 Middleware and Services in .NET Core**

**🔑 What is Middleware?**

Middleware is software that handles HTTP requests and responses in a pipeline. Each component (middleware) in the pipeline can:

* Process the request
* Call the next middleware in the pipeline
* Modify the response

🔧 Middleware runs in the order they are registered in the Configure() method.

**🔧 Where is Middleware Used?**

Middleware is used in the Startup.cs file, inside the Configure() method:

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseRouting(); // Routing middleware

app.UseAuthentication(); // Authentication middleware

app.UseAuthorization(); // Authorization middleware

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers(); // Maps controller routes

});

}

🧠 Example:

app.Use(async (context, next) =>

{

Console.WriteLine("Request received");

await next(); // Call the next middleware

Console.WriteLine("Response sent");

});

**🔑 What are Services in .NET Core?**

**Services** are reusable components or dependencies (like database contexts, logging, authentication, etc.) that are registered with the **Dependency Injection (DI) container**.

**🔧 ConfigureServices() Method**

* Located in Startup.cs
* Used to register **services** with the DI container
* Called once when the application starts

🧠 Example:

public void ConfigureServices(IServiceCollection services)

{

services.AddControllersWithViews(); // Add MVC

services.AddDbContext<AppDbContext>(); // Register DB context

services.AddAuthentication(); // Add authentication services

}

✅ Common Services:

* MVC (AddControllersWithViews())
* Razor Pages (AddRazorPages())
* DbContext (AddDbContext<>)
* Identity (AddIdentity())
* CORS (AddCors())

**🔧 Configure() Method**

* Also in Startup.cs
* Defines the **HTTP request processing pipeline** using middleware
* Called at runtime to set up how the app responds to HTTP requests

🧠 Example:

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseStaticFiles(); // Serves static files from wwwroot

app.UseRouting(); // Enables routing

app.UseAuthorization(); // Applies authorization rules

app.UseEndpoints(endpoints =>

{

endpoints.MapDefaultControllerRoute(); // Maps to default controller

});

}

**📝 Summary Table**

| **Method** | **Purpose** |
| --- | --- |
| ConfigureServices() | Register services (e.g., MVC, DbContext, Identity) |
| Configure() | Define request pipeline using middleware |

✅  **Multiple Choice Questions (MCQs)**

1. What is .NET Core?
   * A. A game engine
   * B. A cross-platform development framework
   * C. A scripting language
   * D. A database tool
2. Which company created .NET Core?
   * A. Google
   * B. IBM
   * C. Microsoft
   * D. Oracle
3. What is the extension of a .NET Core project file?
   * A. .proj
   * B. .coreproj
   * C. .csproj
   * D. .netproj
4. What does SDK stand for in .NET Core?
   * A. Software Deployment Kit
   * B. Software Development Kit
   * C. System Development Kit
   * D. Source Deployment Kit
5. Which command is used to create a new console application in .NET Core?
   * A. dotnet make console
   * B. dotnet new console
   * C. dotnet create app
   * D. new console
6. What folder is used to store static files in ASP.NET Core?
   * A. public
   * B. static
   * C. assets
   * D. wwwroot
7. Which file contains the entry point of a .NET Core app?
   * A. Startup.cs
   * B. Main.cs
   * C. Program.cs
   * D. App.cs
8. What is the use of Startup.cs file?
   * A. To write HTML
   * B. To configure services and middleware
   * C. To manage the database
   * D. To build the UI
9. Which method is used to register services in .NET Core?
   * A. Configure()
   * B. RegisterServices()
   * C. UseServices()
   * D. ConfigureServices()
10. Which method is used to configure middleware in .NET Core?
    * A. ConfigureMiddleware()
    * B. Setup()
    * C. Configure()
    * D. Init()
11. What is Middleware in .NET Core?
    * A. UI component
    * B. Request processing component
    * C. Data storage
    * D. None of the above
12. Which middleware serves static files like images or CSS?
    * A. UseRouting()
    * B. UseAuthorization()
    * C. UseStaticFiles()
    * D. UseEndpoints()
13. What is the correct order of methods in Startup.cs?
    * A. Configure(), ConfigureServices()
    * B. ConfigureServices(), Configure()
    * C. Main(), Configure()
    * D. None
14. Where is dependency injection configured?
    * A. Program.cs
    * B. appsettings.json
    * C. ConfigureServices()
    * D. Controller
15. What type of file is appsettings.json?
    * A. XML
    * B. JSON
    * C. YAML
    * D. INI
16. Which NuGet package manager command is used to add a package?
    * A. dotnet add package
    * B. nuget install
    * C. dotnet install
    * D. add nuget
17. What is the command to run a .NET Core application?
    * A. dotnet execute
    * B. dotnet go
    * C. dotnet run
    * D. run app
18. What is the purpose of UseRouting() middleware?
    * A. Configure authorization
    * B. Handle errors
    * C. Match requests to endpoints
    * D. Serve static files
19. Which method is called when the app starts?
    * A. StartApp()
    * B. Main()
    * C. Begin()
    * D. Run()
20. What file is used for environment settings in .NET Core?
    * A. env.config
    * B. settings.json
    * C. appsettings.json
    * D. web.config
21. In which folder are MVC Views stored?
    * A. Pages
    * B. Templates
    * C. Views
    * D. UI
22. Which of the following is a Razor View file?
    * A. .html
    * B. .cshtml
    * C. .razor
    * D. .xml
23. What is DI in .NET Core?
    * A. Data Integration
    * B. Direct Injection
    * C. Dependency Injection
    * D. Default Injection
24. What kind of method is Main()?
    * A. Static
    * B. Virtual
    * C. Async
    * D. Abstract
25. Which method maps controller endpoints in ASP.NET Core?
    * A. app.MapControllers()
    * B. app.UseRouting()
    * C. app.UseEndpoints()
    * D. app.MapRoutes()
26. What’s the default port for ASP.NET Core on Kestrel?
    * A. 80
    * B. 5000
    * C. 3000
    * D. 8080
27. Middleware components form a:
    * A. Database
    * B. Service Layer
    * C. Pipeline
    * D. View Engine
28. Which folder stores compiled binaries?
    * A. bin
    * B. obj
    * C. out
    * D. build
29. The host is created and run in which file?
    * A. Startup.cs
    * B. Program.cs
    * C. launchSettings.json
    * D. Project.cs
30. Which method is used to add MVC services?
    * A. AddMvc()
    * B. UseMvc()
    * C. RegisterMvc()
    * D. ConfigureMvc()
31. Which of these is NOT a middleware method?
    * A. UseRouting
    * B. UseEndpoints
    * C. UseStaticFiles
    * D. AddControllers
32. Which object is passed into ConfigureServices()?
    * A. IApplicationBuilder
    * B. IServiceCollection
    * C. IWebHostBuilder
    * D. IConfiguration
33. IApplicationBuilder is passed into:
    * A. Main()
    * B. Configure()
    * C. Program.cs
    * D. UseRouting()
34. What command lists installed SDKs?
    * A. dotnet list
    * B. dotnet --sdks
    * C. dotnet --list-sdks
    * D. dotnet get-sdks
35. Where is \_Layout.cshtml typically located?
    * A. Views/Home
    * B. Views/Shared
    * C. Views/Layout
    * D. Views/Templates
36. What is returned from a controller action?
    * A. IActionResult
    * B. void
    * C. string only
    * D. Task
37. What kind of application can .NET Core build?
    * A. Web Apps
    * B. Console Apps
    * C. APIs
    * D. All of the above
38. What command builds a .NET Core app?
    * A. dotnet build
    * B. dotnet compile
    * C. build
    * D. compile
39. Razor views are used for:
    * A. Backend logic
    * B. Database
    * C. HTML UI with C# code
    * D. Static files
40. Which one is used for connecting to a database?
    * A. Startup.cs
    * B. appsettings.json
    * C. DbContext
    * D. View
41. Which type is commonly injected into controllers?
    * A. Services
    * B. Repositories
    * C. DbContext
    * D. All of the above
42. What does UseDeveloperExceptionPage() do?
    * A. Shows custom error UI
    * B. Enables detailed error page in development
    * C. Disables error logging
    * D. Redirects to homepage
43. How are services injected into constructors?
    * A. Static method
    * B. By default
    * C. Constructor injection
    * D. Getter method
44. ASP.NET Core is built on which design pattern?
    * A. Factory
    * B. MVC
    * C. Singleton
    * D. Chain
45. Which JSON file contains launch profiles?
    * A. appsettings.json
    * B. launchSettings.json
    * C. web.config
    * D. profiles.json
46. What is the purpose of app.UseAuthorization()?
    * A. Serves files
    * B. Handles errors
    * C. Enforces access policies
    * D. None
47. Which folder contains temporary build files?
    * A. bin
    * B. obj
    * C. temp
    * D. wwwroot
48. AddDbContext() registers:
    * A. View Engine
    * B. Middleware
    * C. Database Context
    * D. Static Files
49. services.AddControllers() is used to:
    * A. Add controller routes
    * B. Register MVC controllers
    * C. Add static files
    * D. None
50. Which command restores project dependencies?
    * A. dotnet fetch
    * B. dotnet install
    * C. dotnet restore
    * D. dotnet setup

**✅ Answer Key Table**

| **Q** | **A** | **Q** | **A** | **Q** | **A** | **Q** | **A** | **Q** | **A** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | B | 2 | C | 3 | C | 4 | B | 5 | B |
| 6 | D | 7 | C | 8 | B | 9 | D | 10 | C |
| 11 | B | 12 | C | 13 | B | 14 | C | 15 | B |
| 16 | A | 17 | C | 18 | C | 19 | B | 20 | C |
| 21 | C | 22 | B | 23 | C | 24 | A | 25 | C |
| 26 | B | 27 | C | 28 | A | 29 | B | 30 | A |
| 31 | D | 32 | B | 33 | B | 34 | C | 35 | B |
| 36 | A | 37 | D | 38 | A | 39 | C | 40 | C |
| 41 | D | 42 | B | 43 | C | 44 | B | 45 | B |
| 46 | C | 47 | B | 48 | C | 49 | B | 50 | C |

:

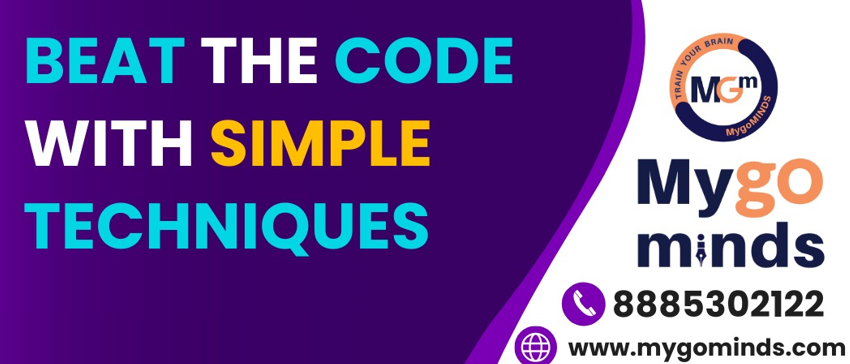
✍️ **Fill in the Blanks**

1. The entry point of a .NET Core application is the \_\_\_\_\_\_\_\_\_\_ method in Program.cs.  
   **Answer: Main**
2. The Startup.cs file contains the \_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_ methods.  
   **Answer: ConfigureServices, Configure**
3. Middleware components are added inside the \_\_\_\_\_\_\_\_\_\_ method in Startup.cs.  
   **Answer: Configure**
4. Static files like images, CSS, and JS are placed inside the \_\_\_\_\_\_\_\_\_\_ folder.  
   **Answer: wwwroot**
5. Services are registered into the dependency injection container using the \_\_\_\_\_\_\_\_\_\_ method.  
   **Answer: ConfigureServices**
6. The file \_\_\_\_\_\_\_\_\_\_ is used to configure application settings like connection strings.  
   **Answer: appsettings.json**
7. Razor view files have the extension \_\_\_\_\_\_\_\_\_\_.  
   **Answer: .cshtml**
8. In .NET Core, app.UseRouting() is used to enable the request routing to the correct \_\_\_\_\_\_\_\_\_\_.  
   **Answer: controller/action**
9. To add MVC services in ConfigureServices, we use the method services.\_\_\_\_\_\_\_\_\_\_();  
   **Answer: AddControllersWithViews**
10. Middleware follows the design pattern of a request-response \_\_\_\_\_\_\_\_\_\_.  
    **Answer: pipeline**

✅  **True or False**

1. .NET Core is a cross-platform and open-source framework.  
   **Answer: True**
2. The ConfigureServices() method is called multiple times during an application's lifetime.  
   **Answer: False**
3. Middleware can modify both the request and the response.  
   **Answer: True**
4. The folder bin contains compiled binaries of the project.  
   **Answer: True**
5. The Configure() method is used to register services.  
   **Answer: False**
6. Dependency Injection is built-in in .NET Core.  
   **Answer: True**
7. Razor pages use the .razor file extension in MVC.  
   **Answer: False**
8. The UseEndpoints() middleware must come after UseRouting().  
   **Answer: True**
9. Program.cs is responsible for building and running the host.  
   **Answer: True**
10. You must always create the Startup.cs file manually in a new .NET Core app.  
    **Answer: False**



****

**Chapter-2**

**Oops in C#**

**📘 1. Classes and Objects**

**Class** is a blueprint for creating objects.  
**Object** is an instance of a class.

🔧 **Example**:

public class Car

{

public string Color;

public void Drive() => Console.WriteLine("Driving the car");

}

Car myCar = new Car();

myCar.Color = "Red";

myCar.Drive();

**📘 2. Inheritance**

Inheritance allows a class to acquire properties and methods from another class using :.

🔧 **Example**:

public class Animal

{

public void Eat() => Console.WriteLine("Eating");

}

public class Dog : Animal

{

public void Bark() => Console.WriteLine("Barking");

}

Dog d = new Dog();

d.Eat();

d.Bark();

**📘 3. Polymorphism**

Polymorphism means many forms. It allows methods to behave differently based on object types.

🔧 **Example** (Method Overriding):

public class Animal

{

public virtual void Sound() => Console.WriteLine("Animal sound");

}

public class Cat : Animal

{

public override void Sound() => Console.WriteLine("Meow");

}

Animal a = new Cat();

a.Sound(); // Meow

**📘 4. Abstraction and Interfaces**

**Abstraction** hides implementation details and shows only essentials.  
**Interface** defines a contract without implementation.

🔧 **Example**:

public interface IShape

{

void Draw();

}

public class Circle : IShape

{

public void Draw() => Console.WriteLine("Drawing Circle");

}

**📘 5. Constructors and Destructors**

**Constructor** initializes an object.  
**Destructor** is used to perform cleanup before an object is destroyed.

🔧 **Example**:

public class Demo

{

public Demo() => Console.WriteLine("Constructor called");

~Demo() => Console.WriteLine("Destructor called");

}

✅  **Multiple Choice Questions (MCQs)**

1. **Which keyword is used for inheritance in C#?**  
   A. extends  
   B. base  
   C. inherits  
   D. :  
   **Answer: D**
2. **What is the default access modifier for class members?**  
   A. private  
   B. public  
   C. internal  
   D. protected  
   **Answer: A**
3. **What type of method can be overridden?**  
   A. static  
   B. virtual  
   C. sealed  
   D. abstract  
   **Answer: B**
4. **Which concept allows multiple classes to use the same method name but behave differently?**  
   A. Inheritance  
   B. Abstraction  
   C. Polymorphism  
   D. Encapsulation  
   **Answer: C**
5. **Which keyword is used to define a constructor?**  
   A. def  
   B. construct  
   C. new  
   D. None (constructor is same name as class)  
   **Answer: D**
6. **Interface members are:**  
   A. Public by default  
   B. Private  
   C. Protected  
   D. Internal  
   **Answer: A**
7. **Which keyword prevents a class from being inherited?**  
   A. virtual  
   B. abstract  
   C. sealed  
   D. static  
   **Answer: C**
8. **What is the purpose of a destructor?**  
   A. Initialize object  
   B. Destroy object memory  
   C. Clean up resources  
   D. Compile object  
   **Answer: C**
9. **How many classes can a class inherit in C#?**  
   A. One  
   B. Two  
   C. Multiple  
   D. None  
   **Answer: A**
10. **Which of the following is not part of OOP?**  
    A. Inheritance  
    B. Encapsulation  
    C. Compiling  
    D. Polymorphism  
    **Answer: C**
11. Which of the following is true about constructors in C#?  
    A. Can have a return type  
    B. Can be overloaded  
    C. Can be inherited  
    D. Must be static  
    **Answer: B**
12. What is the role of the base keyword in C#?  
    A. It creates a base class  
    B. It defines static methods  
    C. It is used to call the base class constructor  
    D. It defines an abstract method  
    **Answer: C**
13. What is method overloading?  
    A. Multiple methods with the same name and signature  
    B. Multiple methods with the same name but different parameters  
    C. Methods defined in an interface  
    D. A method calling another method  
    **Answer: B**
14. Which access modifier allows access within the same assembly and from derived classes?  
    A. public  
    B. protected  
    C. private  
    D. protected internal  
    **Answer: D**
15. Which keyword is used to inherit a class in C#?  
    A. derive  
    B. implement  
    C. :  
    D. extends  
    **Answer: C**
16. Which of the following supports runtime polymorphism?  
    A. Overloading  
    B. Static methods  
    C. Virtual methods  
    D. Private methods  
    **Answer: C**
17. What is encapsulation?  
    A. Hiding internal details  
    B. Inheriting properties  
    C. Using interfaces  
    D. Creating objects  
    **Answer: A**
18. Which class member is shared among all objects of a class?  
    A. readonly  
    B. const  
    C. static  
    D. protected  
    **Answer: C**
19. Which class cannot be instantiated?  
    A. Static  
    B. Sealed  
    C. Abstract  
    D. Interface  
    **Answer: C**
20. An interface can contain:  
    A. Constructors  
    B. Private methods  
    C. Constants and methods  
    D. Method declarations only
21. What is abstraction in OOP?  
    A. Making members public  
    B. Hiding unnecessary details  
    C. Using static methods  
    D. Overriding base methods  
    **Answer: B**
22. What is the purpose of the override keyword?  
    A. Hide base class method  
    B. Overload method  
    C. Replace virtual method implementation  
    D. Define a new method  
    **Answer: C**
23. Which class type supports multiple inheritance?  
    A. Abstract class  
    B. Sealed class  
    C. Interface  
    D. Partial class  
    **Answer: C**
24. A constructor with no parameters is called:  
    A. Static constructor  
    B. Overloaded constructor  
    C. Default constructor  
    D. Abstract constructor  
    **Answer: C**
25. Can a class inherit from multiple base classes in C#?  
    A. Yes  
    B. No  
    C. Only if abstract  
    D. Only if sealed  
    **Answer: B**
26. Which modifier makes a class member accessible from any other code?  
    A. private  
    B. protected  
    C. internal  
    D. public  
    **Answer: D**
27. Which type of class must override its abstract members?  
    A. Normal class  
    B. Sealed class  
    C. Static class  
    D. Derived class  
    **Answer: D**
28. Which feature allows different behavior for the same method in different classes?  
    A. Method hiding  
    B. Constructor overloading  
    C. Polymorphism  
    D. Abstraction  
    **Answer: C**
29. Which keyword hides a base class member in derived class?  
    A. base  
    B. sealed  
    C. override  
    D. new  
    **Answer: D**
30. Which of the following is true about static constructors?  
    A. Called manually  
    B. Can have parameters  
    C. Called automatically  
    D. Can be inherited  
    **Answer: C**
31. In C#, the object class is the:  
    A. Interface  
    B. Derived class  
    C. Root class  
    D. Abstract class  
    **Answer: C**
32. What is the output of polymorphism in OOP?  
    A. Memory leak  
    B. One method, multiple behaviors  
    C. Multiple classes  
    D. One constructor  
    **Answer: B**
33. What is the default base class of all classes in C#?  
    A. Object  
    B. Base  
    C. System  
    D. Class  
    **Answer: A**
34. Which keyword defines an abstract method?  
    A. sealed  
    B. override  
    C. virtual  
    D. abstract  
    **Answer: D**
35. Which access modifier allows access only within the class?  
    A. private  
    B. public  
    C. internal  
    D. protected  
    **Answer: A**
36. Can a class be both abstract and sealed?  
    A. Yes  
    B. No  
    C. Only in .NET Core  
    D. Only in interfaces  
    **Answer: B**
37. What is true about interfaces in C#?  
    A. They can have private members  
    B. They can have method bodies  
    C. They support multiple inheritance  
    D. They must be sealed  
    **Answer: C**
38. Which operator is used to create objects in C#?  
    A. create  
    B. this  
    C. new  
    D. class  
    **Answer: C**
39. Which of these keywords is used to inherit constructors explicitly?  
    A. base  
    B. this  
    C. override  
    D. sealed  
    **Answer: A**
40. Which method is used for cleanup before garbage collection?  
    A. Initialize  
    B. Dispose  
    C. Destructor  
    D. Finalize  
    **Answer: D**
41. What happens when you override a method in a derived class?  
    A. It hides the base version  
    B. It ignores base implementation  
    C. It replaces the base method behavior  
    D. It calls base method  
    **Answer: C**
42. Which of these cannot be used with an interface?  
    A. Properties  
    B. Events  
    C. Fields  
    D. Methods  
    **Answer: C**
43. What is true about a sealed method?  
    A. It can be overridden  
    B. It cannot be inherited  
    C. It prevents further overriding  
    D. It must be static  
    **Answer: C**
44. What is true about partial classes?  
    A. They can only be used in inheritance  
    B. Defined in a single file  
    C. Allows class split across files  
    D. Only used with interfaces  
    **Answer: C**
45. What is the benefit of encapsulation?  
    A. Code duplication  
    B. Better performance  
    C. Code security and maintenance  
    D. Complex code structure  
    **Answer: C**
46. Which OOP principle is used when a child class uses parent class methods?  
    A. Abstraction  
    B. Inheritance  
    C. Polymorphism  
    D. Encapsulation  
    **Answer: B**
47. What is the purpose of the virtual keyword?  
    A. Prevent method access  
    B. Make method abstract  
    C. Allow overriding  
    D. Prevent override  
    **Answer: C**
48. Which of the following can be declared as abstract?  
    A. Constructors  
    B. Properties  
    C. Static methods  
    D. Private methods  
    **Answer: B**
49. Which C# feature is used to hide complexity from users?  
    A. Inheritance  
    B. Overriding  
    C. Abstraction  
    D. Interfaces  
    **Answer: C**
50. Which of the following supports both fields and implementation?  
    A. Interface  
    B. Abstract class  
    C. Sealed class  
    D. Static class  
    **Answer: B**

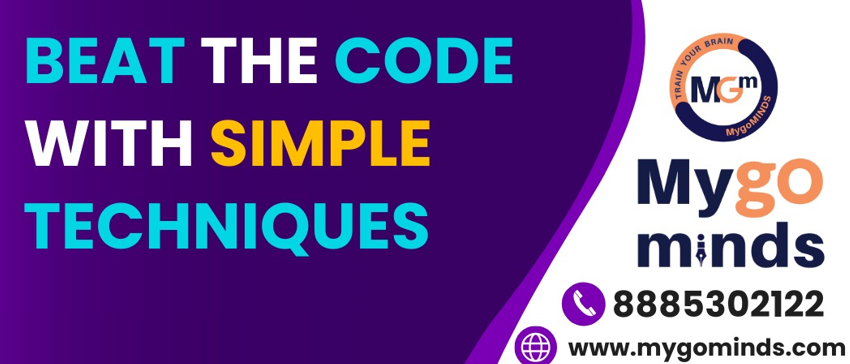
✍️ **Fill in the Blanks**

1. A class is a \_\_\_\_\_\_\_ for creating objects.  
   **Answer: blueprint**
2. A constructor has the same \_\_\_\_\_\_\_ as the class.  
   **Answer: name**
3. Inheritance is achieved using the \_\_\_\_\_\_\_ symbol.  
   **Answer: colon (:)**
4. A class can only inherit from \_\_\_\_\_\_\_ base class. **Answer: one**
5. The virtual keyword is used for method \_\_\_\_\_\_\_.  
   **Answer: overriding**
6. Destructor starts with a \_\_\_\_\_\_\_ character.  
   **Answer: tilde (~)**
7. Interfaces contain only method \_\_\_\_\_\_\_ without body.  
   **Answer: signatures**
8. Object is an \_\_\_\_\_\_\_ of a class.  
   **Answer: instance**
9. The override keyword is used in \_\_\_\_\_\_\_ class.  
   **Answer: derived**
10. The keyword sealed is used to \_\_\_\_\_\_\_ inheritance.  
    **Answer: prevent**

✅  **True or False**

1. A class can implement multiple interfaces in C#.  
   **Answer: True**
2. Constructors can be overloaded in C#.  
   **Answer: True**
3. Abstract classes cannot have method implementation.  
   **Answer: False**  
   (They can have both implemented and abstract methods.)
4. Interfaces can contain fields.  
   **Answer: False**  
   (Interfaces can only contain signatures, not fields.)
5. The destructor can be called manually.  
   **Answer: False**  
   (It is automatically called by the garbage collector.)
6. Polymorphism is achieved using method overriding.  
   **Answer: True**
7. You must define a constructor explicitly in every class.  
   **Answer: False**  
   (If not defined, a default constructor is provided automatically.)
8. Sealed classes cannot be inherited.  
   **Answer: True**
9. You can have multiple constructors in a class.  
   **Answer: True**
10. An abstract method must be implemented in a derived class.  
    **Answer: True**



****

**Chapter-3**

**Advanced topics in .Net Core**

## 📘 ****SOLID Principles in .NET Core (C#)****

**SOLID** is an acronym for five design principles that help developers write cleaner, more manageable, and scalable code.

### 🔑 ****1. S – Single Responsibility Principle (SRP)****

**Definition:** A class should have one, and only one, reason to change.

🔧 **Example**:

public class Invoice

{

public void CalculateTotal() { /\* logic \*/ }

}

public class InvoicePrinter

{

public void Print(Invoice invoice) { /\* logic \*/ }

}

Why? Logic and printing are separated — each class has a single responsibility.

### 🔑 ****2. O – Open/Closed Principle (OCP)****

**Definition:** Software entities should be open for extension but closed for modification.

🔧 **Example using interfaces**:

public interface IDiscount

{

double GetDiscount(double total);

}

public class RegularDiscount : IDiscount

{

public double GetDiscount(double total) => total \* 0.1;

}

public class PremiumDiscount : IDiscount

{

public double GetDiscount(double total) => total \* 0.2;

}

Why? You can add new discount types without modifying existing code.

### 🔑 ****3. L – Liskov Substitution Principle (LSP)****

**Definition:** Subtypes must be substitutable for their base types without breaking the application.

🔧 **Example**:

public abstract class Bird

{

public abstract void Fly();

}

public class Sparrow : Bird

{

public override void Fly() => Console.WriteLine("Flying...");

}

A duck shouldn’t inherit Bird if it can’t fly – use interfaces wisely.

### 🔑 ****4. I – Interface Segregation Principle (ISP)****

**Definition:** No client should be forced to depend on methods it does not use.

🔧 **Bad Example**:

public interface IWorker

{

void Work();

void Eat();

}

🔧 **Better**:

public interface IWorkable { void Work(); }

public interface IEatable { void Eat(); }

public class Robot : IWorkable { public void Work() { } }

public class Human : IWorkable, IEatable { public void Work() { } public void Eat() { } }

### 🔑 ****5. D – Dependency Inversion Principle (DIP)****

**Definition:** High-level modules should not depend on low-level modules. Both should depend on abstractions.

🔧 **Example**:

public interface IMessageService

{

void Send(string message);

}

public class EmailService : IMessageService

{

public void Send(string message) => Console.WriteLine($"Email: {message}");

}

public class Notification

{

private readonly IMessageService \_service;

public Notification(IMessageService service) => \_service = service;

public void Notify(string msg) => \_service.Send(msg);

}

Use constructor injection to follow DIP, as shown above.

Would you like me to continue with other **Advanced Concepts** like:

* Dependency Injection in .NET Core
* Extension Methods
* Async & Await Patterns
* Delegates & Events
* LINQ & Expression Trees
* Nullable Reference Types
* Reflection & Attributes

## 📘 ****1. Dependency Injection in .NET Core****

**Definition:** A design pattern that allows the removal of hard-coded dependencies and makes code more flexible and testable.

🔧 **Example:**

public interface IMessageService

{

void Send(string message);

}

public class EmailService : IMessageService

{

public void Send(string message) => Console.WriteLine("Email: " + message);

}

public class Notification

{

private readonly IMessageService \_messageService;

public Notification(IMessageService service)

{

\_messageService = service;

}

public void Notify(string message)

{

\_messageService.Send(message);

}

}

🛠️ **In Startup.cs**:

services.AddScoped<IMessageService, EmailService>();

## 📘 ****2. Extension Methods****

**Definition:** Allows adding new methods to existing types without modifying them.

🔧 **Example:**

public static class StringExtensions

{

public static bool IsLong(this string str)

{

return str.Length > 10;

}

}

string name = "HelloWorld!";

bool result = name.IsLong(); // True

## 📘 ****3. Async & Await Patterns****

**Definition:** Used for asynchronous programming. Improves performance and responsiveness.

### 🔑 ****Definition:****

**Async and Await** are C# keywords used to write asynchronous code in a simple and readable way. They allow non-blocking operations such as I/O, database access, or web requests without freezing the main thread (especially important in UI applications or web servers).

### 📝 ****Key Concepts:****

* **async**: Marks a method as asynchronous. It must return Task, Task<T>, or void (rarely used).
* **await**: Pauses the method execution until the awaited Task completes, without blocking the thread.
* Improves **responsiveness**, **scalability**, and **performance** of applications.

🔧 **Example Code:**

using System;

using System.Net.Http;

using System.Threading.Tasks;

class Program

{

static async Task Main(string[] args)

{

string content = await GetWebPageAsync("https://example.com");

Console.WriteLine(content);

}

static async Task<string> GetWebPageAsync(string url)

{

HttpClient client = new HttpClient();

string result = await client.GetStringAsync(url);

return result;

}

}

🔍 **What’s happening?**

* GetWebPageAsync fetches data from the web.
* await tells the compiler to wait for GetStringAsync to finish without blocking the main thread.
* Control returns to the caller (Main) while waiting.

| **Feature** | **Benefit** |
| --- | --- |
| Non-blocking | UI remains responsive |
| Improves performance | Frees up threads for other work |
| Readable code | Looks like synchronous code but runs asynchronously |
| Handles latency | Ideal for I/O-bound operations (DB, Web APIs, File I/O) |

❗ Common Return Types

| **Return Type** | **Use Case** |
| --- | --- |
| Task | For methods with no return value |
| Task<T> | For methods that return a value (T) |
| void | Only for event handlers |

## 📘 ****4. Delegates & Events****

**Definition:** Delegates are type-safe function pointers. Events are a way to publish/subscribe to actions.

🔧 **Example (Delegate):**

public delegate void Notify(); // Declare delegate

public class Process

{

public Notify OnProcessCompleted; // Event using delegate

public void Start()

{

Console.WriteLine("Process Started");

OnProcessCompleted?.Invoke();

}

}

🔧 **Use:**

Process p = new Process();

p.OnProcessCompleted = () => Console.WriteLine("Process Completed");

p.Start();

## 📘 ****5. LINQ & Expression Trees****

**LINQ (Language Integrated Query):** Enables querying collections using C# syntax.

🔧 **Example:**

int[] numbers = { 1, 2, 3, 4, 5 };

var evenNumbers = numbers.Where(n => n % 2 == 0).ToList();

**Expression Trees:** Represent code in a tree-like structure.

🔧 **Example:**

Expression<Func<int, bool>> isEven = x => x % 2 == 0;

## 📘 ****6. Nullable Reference Types (C# 8.0+)****

**Definition:** Helps avoid null reference exceptions.

🔧 **Enable in project file (.csproj):**

<Nullable>enable</Nullable>

🔧 **Example:**

string? name = null; // Nullable reference type

string value = name ?? "default"; // Safe handling

## 📘 ****7. Reflection & Attributes****

**Reflection:** Allows inspection of metadata, types, methods, etc., at runtime.

🔧 **Example:**

Type type = typeof(string);

MethodInfo[] methods = type.GetMethods();

foreach (var m in methods)

Console.WriteLine(m.Name);

**Attributes:** Add metadata to code elements.

🔧 **Custom Attribute Example:**

[AttributeUsage(AttributeTargets.Class)]

public class InfoAttribute : Attribute

{

public string Description { get; }

public InfoAttribute(string description) => Description = description;

}

[Info("Sample Class")]

public class Sample { }

🔧 **Reading it via Reflection:**

var attr = typeof(Sample).GetCustomAttribute<InfoAttribute>();

Console.WriteLine(attr.Description);

✅  **Multiple Choice Questions (MCQs)**

### 1. What does Dependency Injection help achieve?

A) Hard-coded dependencies  
B) Loose coupling  
C) Code duplication  
D) Runtime errors

**Answer: B) Loose coupling**

### 2. What is required to define an extension method?

A) Static class  
B) Public interface  
C) Abstract class  
D) Sealed class

**Answer: A) Static class**

### 3. Which keyword is used to define an asynchronous method?

A) async  
B) wait  
C) task  
D) thread

**Answer: A) async**

### 4. What is the correct way to wait for a task to complete asynchronously?

A) wait Task  
B) task.Wait();  
C) await Task  
D) Task.Completed

**Answer: C) await Task**

### 5. Which keyword is used to define a delegate?

A) function  
B) delegate  
C) action  
D) lambda

**Answer: B) delegate**

### 6. LINQ allows you to:

A) Format code  
B) Query collections  
C) Debug applications  
D) Create threads

**Answer: B) Query collections**

### 7. What does Expression<Func<int, bool>> represent?

A) A LINQ query  
B) A compiled delegate  
C) An expression tree  
D) An anonymous method

**Answer: C) An expression tree**

### 8. Which file do you modify to enable nullable reference types?

A) Startup.cs  
B) Program.cs  
C) .csproj  
D) launchSettings.json

**Answer: C) .csproj**

### 9. What is the purpose of the AttributeUsage attribute?

A) To create properties  
B) To initialize an attribute  
C) To define how an attribute can be used  
D) To enable reflection

**Answer: C) To define how an attribute can be used**

### 10. Reflection is primarily used for:

A) Compiling code  
B) Creating interfaces  
C) Inspecting metadata at runtime  
D) Drawing UIs

**Answer: C) Inspecting metadata at runtime**

**11.** Which interface is commonly used in .NET Core for dependency injection?  
o A. IDependency  
o B. IServiceCollection  
o C. IApplicationBuilder  
o D. IHostEnvironment  
**Answer: B) IServiceCollection**

**12.** Which of the following is a valid extension method syntax?  
o A. public void Extend(this string str)  
o B. static void Extend(string str)  
o C. void Extend(this string str)  
o D. public static void Extend(this string str)  
**Answer: D) public static void Extend(this string str)**

**13.** Which return type is commonly used with async methods?  
o A. Task  
o B. Void  
o C. Thread  
o D. Action  
**Answer: A) Task**

**14.** What is the purpose of ConfigureServices in Startup.cs?  
o A. Configure UI elements  
o B. Register services for dependency injection  
o C. Handle exceptions  
o D. Map API routes  
**Answer: B) Register services for dependency injection**

**15.** Which of the following can be used to define a delegate with parameters?  
o A. delegate void MyDelegate();  
o B. delegate void MyDelegate(int x);  
o C. delegate MyDelegate(int x);  
o D. action MyDelegate(int x);  
**Answer: B) delegate void MyDelegate(int x);**

**16.** What is the result of list.Where(x => x > 5).ToList();?  
o A. Filters values greater than 5 and returns a list  
o B. Sorts the list in ascending order  
o C. Maps values to a new type  
o D. Adds 5 to each element  
**Answer: A) Filters values greater than 5 and returns a list**

**17.** Which type can be used to build LINQ providers?  
o A. Func<T>  
o B. IQueryable<T>  
o C. IEnumerable<T>  
o D. Predicate<T>  
**Answer: B) IQueryable<T>**

**18.** What is the purpose of Nullable.Enable in a C# project?  
o A. It disables reference types  
o B. It treats reference types as nullable by default  
o C. It enables strict null-safety checks  
o D. It allows primitive types to be nullable  
**Answer: C) It enables strict null-safety checks**

**19.** Which method retrieves all properties of a class at runtime using reflection?  
o A. GetFields()  
o B. GetProperties()  
o C. GetMethods()  
o D. GetMembers()  
**Answer: B) GetProperties()**

**20.** What does the [Obsolete] attribute indicate in C#?  
o A. A method is newly added  
o B. A method is not thread-safe  
o C. A method should not be used anymore  
o D. A method is extension-only  
**Answer: C) A method should not be used anymore**

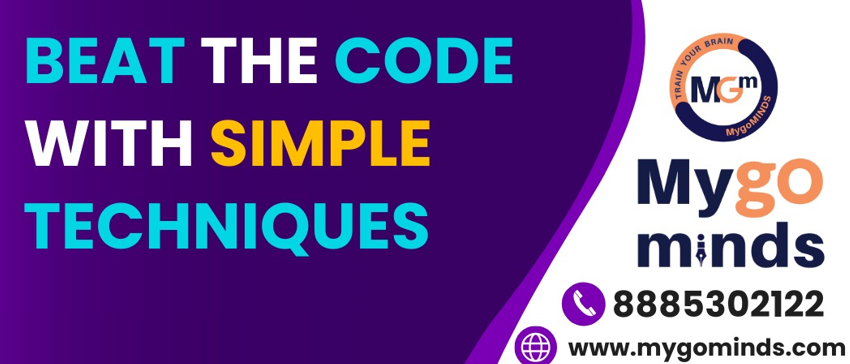
✍️ **Fill in the Blanks**

1. In .NET Core, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ helps in achieving loose coupling between classes.  
   **Answer: Dependency Injection**
2. An \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ method allows you to add new methods to existing types without modifying them.  
   **Answer: Extension**
3. The keyword used to define an asynchronous method is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: async**
4. Delegates in C# are \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ function pointers.  
   **Answer: type-safe**
5. The await keyword can only be used inside a method marked with \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: async**
6. LINQ stands for \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: Language Integrated Query**
7. Expression trees are defined using the type \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: Expression<Func<...>>**
8. Nullable reference types are enabled using the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ element in the .csproj file.  
   **Answer: <Nullable>enable</Nullable>**
9. Reflection allows you to inspect \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of types at runtime.  
   **Answer: metadata**
10. Custom attributes are derived from the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ class.  
    **Answer: Attribute**

✅  **True or False**

1. Dependency Injection is used to inject low-level modules directly into high-level modules.  
   **Answer: False**
2. Extension methods must be defined in a static class.  
   **Answer: True**
3. The await keyword can be used without the async keyword in a method.  
   **Answer: False**
4. Delegates are used to store references to methods.  
   **Answer: True**
5. Events in C# use delegates under the hood.  
   **Answer: True**
6. LINQ can be used only with arrays.  
   **Answer: False**
7. Expression trees are evaluated immediately like regular code.  
   **Answer: False**
8. Nullable reference types are available by default in all versions of C#.  
   **Answer: False**
9. Reflection can be used to dynamically invoke methods.  
   **Answer: True**
10. Custom attributes cannot have constructor parameters.  
    **Answer: False**



****

**Chapter-4**

**.Net Core CLI and Project Structure**

**📘 .NET Core CLI and Project Structure**

**🔑 1. dotnet CLI Commands**

**Definition:** The .NET CLI (Command Line Interface) is a cross-platform tool for building, running, and managing .NET applications.

🔧 **Common Commands:**

| **Command** | **Description** |
| --- | --- |
| dotnet new console | Creates a new console app |
| dotnet build | Builds the application |
| dotnet run | Runs the application |
| dotnet restore | Restores dependencies from NuGet |
| dotnet test | Runs unit tests |
| dotnet publish | Prepares app for deployment |

**🔑 2. Understanding csproj files**

**Definition:** A .csproj file defines project settings and dependencies in XML format.

🔧 **Sample:**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net6.0</TargetFramework>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Newtonsoft.Json" Version="13.0.1" />

</ItemGroup>

</Project>

* OutputType: Type of app (Exe or Library)
* TargetFramework: The .NET version
* PackageReference: External packages used

**🔑 3. Project Folder Structure**

🔧 **Default Structure (Console App):**

MyApp/

├── Program.cs

├── MyApp.csproj

├── bin/

├── obj/

* Program.cs: Entry point of the app
* bin/: Compiled binaries
* obj/: Intermediate files
* .csproj: Project configuration

**🔑 4. Adding Dependencies via NuGet**

**Definition:** NuGet is the package manager for .NET. You can install libraries using the CLI or manually via .csproj.

🔧 **CLI Example:**

dotnet add package Newtonsoft.Json

🔧 **Manual in .csproj:**

<PackageReference Include="Newtonsoft.Json" Version="13.0.1" />

✅  **Multiple Choice Questions (MCQs)**

1.Which command is used to create a new console application?

A) dotnet console  
B) dotnet create console  
C) dotnet new console  
D) dotnet init console

**Answer: C) dotnet new console**

2.Which command compiles the application?

A) dotnet compile  
B) dotnet make  
C) dotnet run  
D) dotnet build

**Answer: D) dotnet build**

3.Which file holds project configuration in a .NET Core app?

A) appsettings.json  
B) startup.cs  
C) launchSettings.json  
D) .csproj

**Answer: D) .csproj**

4.What does dotnet restore do?

A) Compiles the app  
B) Installs NuGet packages  
C) Runs unit tests  
D) Publishes the app

**Answer: B) Installs NuGet packages**

5.What is the default entry file for a console app?

A) main.cs  
B) program.cs  
C) startup.cs  
D) init.cs

**Answer: B) program.cs**

6.Which folder contains compiled outputs?

A) src/  
B) lib/  
C) bin/  
D) build/

**Answer: C) bin/**

7.How can you add a NuGet package via CLI?

A) dotnet add reference  
B) dotnet add project  
C) dotnet add package  
D) dotnet install package

**Answer: C) dotnet add package**

8.What does <OutputType>Exe</OutputType> mean in .csproj?

A) The app is a web service  
B) The app is a class library  
C) The app will generate an executable  
D) The app is a test project

**Answer: C) The app will generate an executable**

9.Which XML element in .csproj specifies the .NET version?

A) <Platform>  
B) <Runtime>  
C) <Framework>  
D) <TargetFramework>

**Answer: D) <TargetFramework>**

10.Which directory is created when the app is built?

A) env/  
B) cache/  
C) obj/  
D) logs/

**Answer: C) obj/**

**11.** Which command runs the application in .NET Core?  
o A) dotnet execute  
o B) dotnet build  
o C) dotnet run  
o D) dotnet start  
**Answer: C) dotnet run**

**12.** Which directory typically stores user secrets and environment configs during development?  
o A) .vs/  
o B) secrets/  
o C) env/  
o D) Properties/  
**Answer: A) .vs/**

**13.** Which command is used to publish a .NET Core app for deployment?  
o A) dotnet export  
o B) dotnet pack  
o C) dotnet deploy  
o D) dotnet publish  
**Answer: D) dotnet publish**

**14.** What does dotnet clean do?  
o A) Deletes source code  
o B) Restores packages  
o C) Removes build outputs  
o D) Removes NuGet cache  
**Answer: C) Removes build outputs**

**15.** Which file in a .NET Core project typically contains environment-specific settings?  
o A) global.json  
o B) appsettings.Development.json  
o C) launchSettings.json  
o D) runtimeconfig.json  
**Answer: B) appsettings.Development.json**

**16.** Which tool is required to use the dotnet command?  
o A) Node.js  
o B) Visual Studio Code  
o C) .NET SDK  
o D) IIS Server  
**Answer: C) .NET SDK**

**17.** What does the --force option do in some dotnet CLI commands?  
o A) Forces an application shutdown  
o B) Skips all errors  
o C) Overwrites files or skips prompts  
o D) Installs all dependencies  
**Answer: C) Overwrites files or skips prompts**

**18.** Which file stores the command-line arguments and profiles for debugging?  
o A) settings.json  
o B) global.json  
o C) launchSettings.json  
o D) debug.config  
**Answer: C) launchSettings.json**

**19.** What is the purpose of the dotnet tool command?  
o A) Create APIs  
o B) Manage global/local .NET CLI tools  
o C) Launch web apps  
o D) Migrate projects  
**Answer: B) Manage global/local .NET CLI tools**

**20.** What is the name of the hidden folder that contains NuGet package cache?  
o A) .bin/  
o B) .nuget/  
o C) packages/  
o D) .tools/  
**Answer: B) .nuget/**

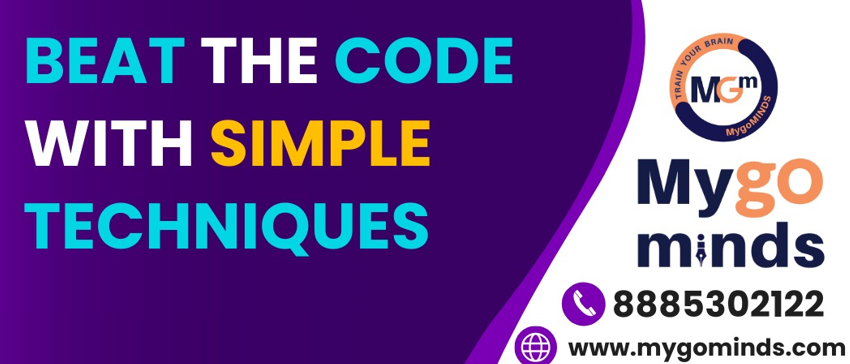
✍️ **Fill in the Blanks**

1. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ command creates a new .NET Core console application.  
   **Answer: dotnet new console**
2. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ file defines dependencies and settings for a project.  
   **Answer: .csproj**
3. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ folder contains compiled application binaries.  
   **Answer: bin**
4. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ command installs all dependencies from NuGet.  
   **Answer: dotnet restore**
5. The XML tag \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ in .csproj sets the .NET version.  
   **Answer: <TargetFramework>**
6. The default file that contains the Main method is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: Program.cs**
7. NuGet packages are added via the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ command.  
   **Answer: dotnet add package**
8. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ folder holds intermediate object files.  
   **Answer: obj**
9. The command \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is used to run the application.  
   **Answer: dotnet run**
10. Each project in .NET Core has exactly one \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ file.  
    **Answer: .csproj**

✅  **True or False**

1. dotnet build restores NuGet packages automatically.  
   **Answer: True**
2. The .csproj file is written in JSON format.  
   **Answer: False**
3. The obj folder is created when you publish your application.  
   **Answer: False**
4. dotnet publish creates a self-contained deployable output.  
   **Answer: True**
5. The bin folder contains debug and release builds.  
   **Answer: True**
6. You cannot edit the .csproj file manually.  
   **Answer: False**
7. The command dotnet add package can be used without editing .csproj.  
   **Answer: True**
8. NuGet is used to manage frontend packages in .NET Core.  
   **Answer: False**
9. The CLI is available only on Windows.  
   **Answer: False**
10. The <PackageReference> tag in .csproj is used to add NuGet packages.  
    **Answer: True**



****

**Chapter-5**

**Dependency Injection and Middleware**

**📘 Dependency Injection and Middleware in ASP.NET Core**

**✅Dependency Injection (DI)?**

* **Dependency Injection is a mechanism of injecting the objects within the application**
* **Dependency Injection (DI) is a design pattern used to achieve Inversion of Control (IoC) between classes and their dependencies.**
* **Instead of a class creating its own dependencies (like services, repositories), they are provided (injected) by an external entity — typically a DI container.**
* public class CowMilk
* {
* public string GetMilk()
* {
* return "Cow Milk";
* }
* }
* public class BrownSugar
* {
* public string GetSugar()
* {
* return "Brown Sugar";
* }
* }
* public class CoffeeMachine
* {
* private CowMilk \_milk = new CowMilk();
* private BrownSugar \_sugar = new BrownSugar();
* public void MakeCoffee()
* {
* Console.WriteLine($"Making coffee with {\_milk.GetMilk()} and {\_sugar.GetSugar()}");
* }
* }
* class Program
* {
* static void Main(string[] args)
* {
* CoffeeMachine machine = new CoffeeMachine();
* machine.MakeCoffee();
* }
* }
* }

**🔍 Why this is tightly coupled:**

* CoffeeMachine controls which milk and sugar classes to use.
* You cannot easily replace CowMilk with AlmondMilk or BrownSugar with WhiteSugar without changing the internal code of CoffeeMachine.

**Define Interfaces**

* public interface IMilk
* {
* string GetMilk();
* }
* public interface ISugar
* {
* string GetSugar();
* }

**Create Implementations:**

* public class CowMilk : IMilk
* {
* public string GetMilk() => "Cow Milk";
* }
* public class BrownSugar : ISugar
* {
* public string GetSugar() => "Brown Sugar";
* }

**CoffeeMachine class using DI**

* public class CoffeeMachine
* {
* private readonly IMilk \_milk;
* private readonly ISugar \_sugar;
* public CoffeeMachine(IMilk milk, ISugar sugar)
* {
* \_milk = milk;
* \_sugar = sugar;
* }
* public void MakeCoffee()
* {
* Console.WriteLine($"Making coffee with {\_milk.GetMilk()} and {\_sugar.GetSugar()}");
* }
* }

**Setup in Program.cs (Console App - .NET Core 8)**

**using Microsoft.Extensions.DependencyInjection;**

class Program

{

static void Main(string[] args)

{

var services = new ServiceCollection();

// Register services

services.AddTransient<IMilk, CowMilk>();

services.AddTransient<ISugar, BrownSugar>();

services.AddTransient<CoffeeMachine>();

var provider = services.BuildServiceProvider();

var machine = provider.GetRequiredService<CoffeeMachine>();

machine.MakeCoffee();

}

}

**✅ MMake Coffee using Dependency Injection**

1. Start Program
2. Create a new service collection
3. Initialize a Service Collection object to register services.
4. Register required services

a.Register IMilk interface with CowMilk implementation.

b.Register ISugar interface with BrownSugar implementation.

c.Register the CoffeeMachine class.

5. Build the service provider

6. Use BuildServiceProvider() to create a service provider from the registered services.

7. Resolve the CoffeeMachine service

8. Get an instance of CoffeeMachine using GetRequiredService.

9. Dependencies (IMilk, ISugar) are automatically injected by the service provider.

10. Call the MakeCoffee() method

11. Invoke the method to simulate coffee making.

12. End Program

**🔧 Why Dependency Injection**

* Imagine you're making coffee.
* You (the coffee drinker) need coffee.
* To get it, you need a coffee machine.
* The coffee machine needs coffee powder, milk, and sugar to work.
* Now, instead of you going out and getting all these things (milk, sugar, powder) and fixing the machine every time...

☕ Someone comes and sets up the machine for you, adds the milk, sugar, and powder, and hands you a ready-made coffee machine

**🔧 Program.cs**

You collect all ingredients 👜

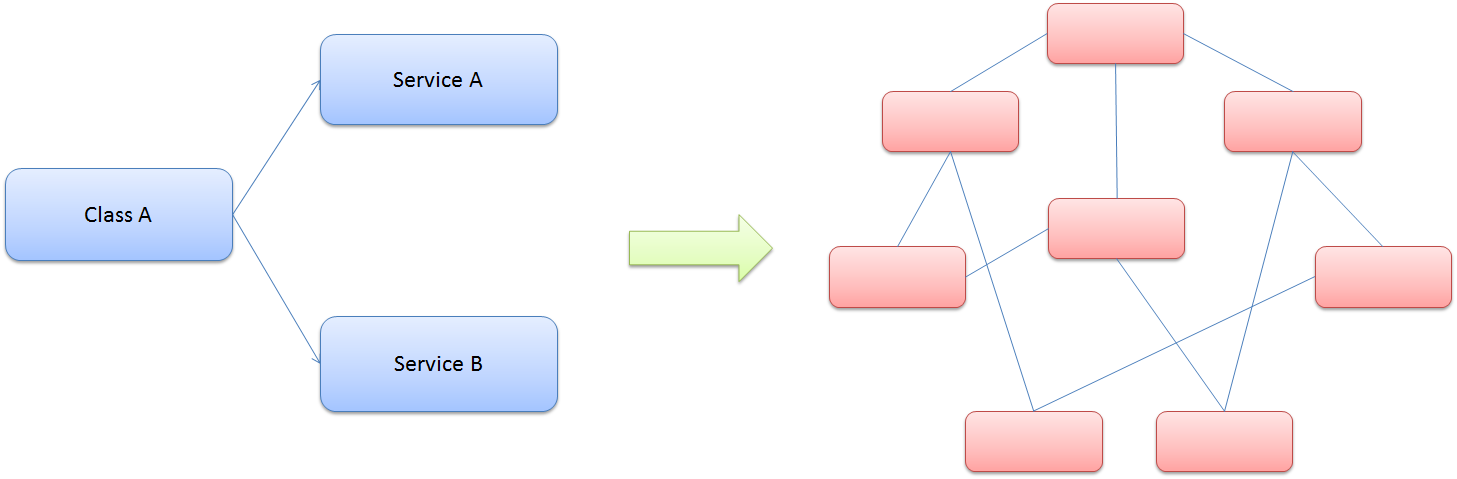
You put Milk, Sugar, etc. in it

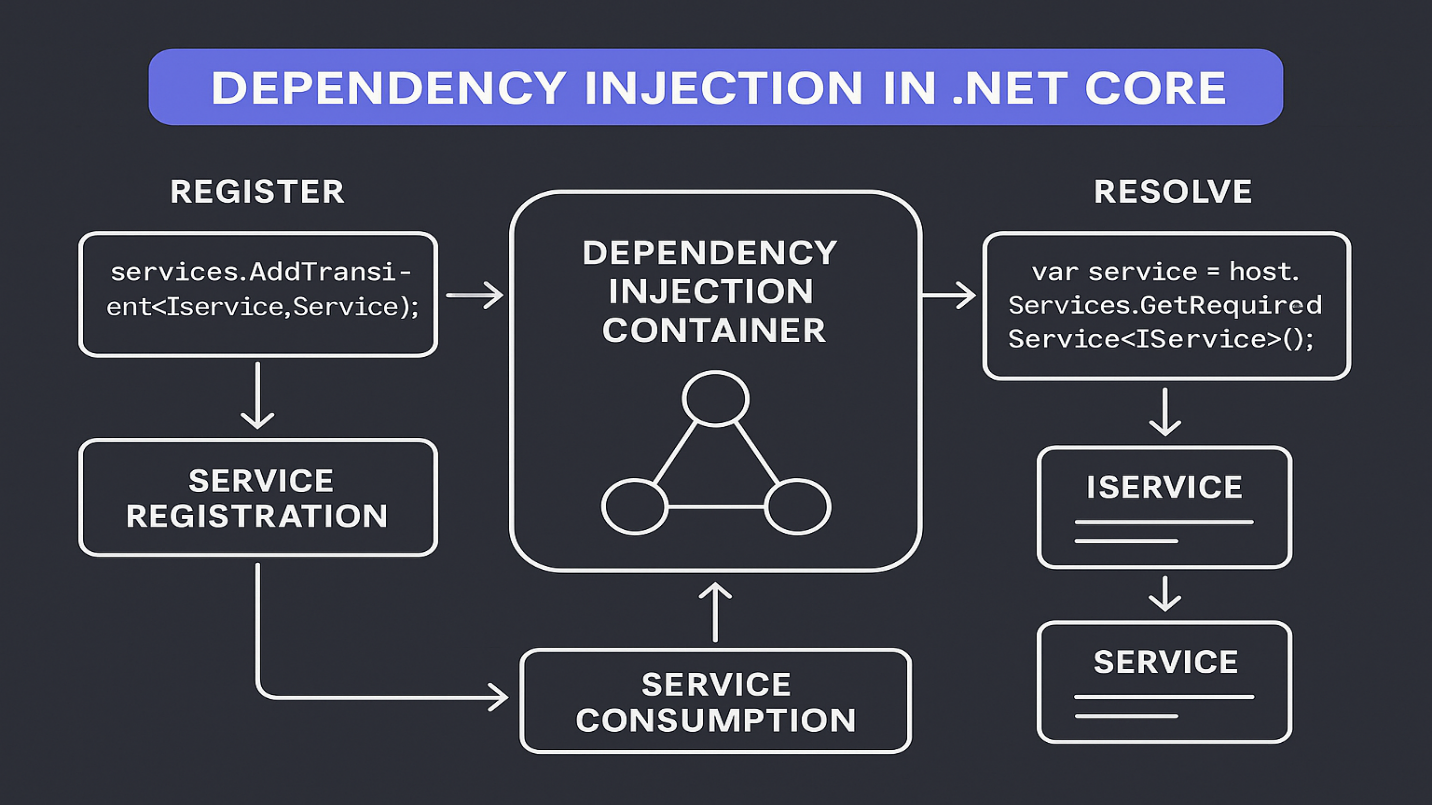
You seal the bag

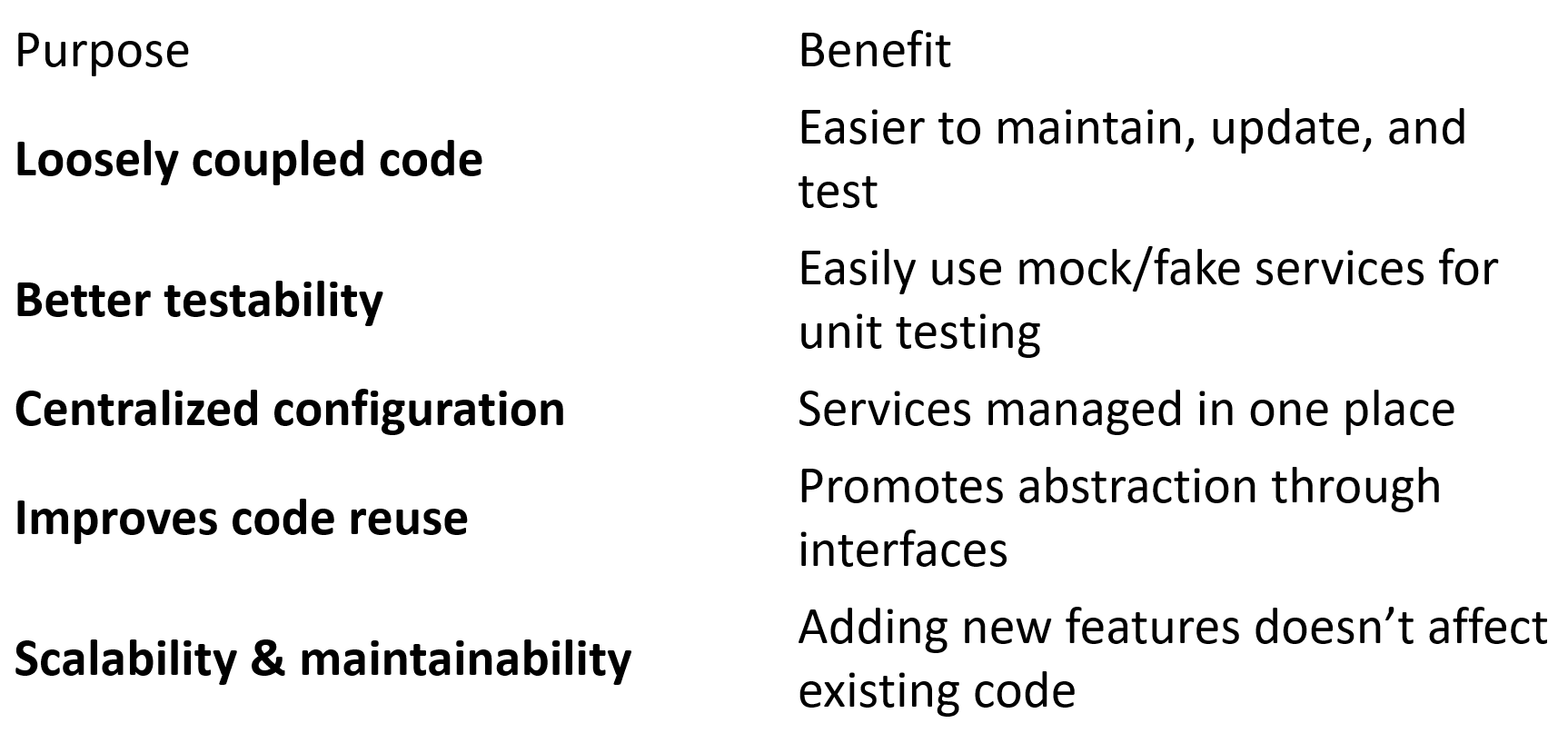
You open the bag to make coffee

You make coffee ☕

* ServiceCollection sc = new ServiceCollection();
* sc.AddTransient<IMilk,CowMilk>(); etc.
* var provider = sc.BuildServiceProvider();
* var machine = provider.GetRequiredService<CoffeeMachine>();
* machine.MakeCoffee();

****

****

**🔧 Purpose of Dependency Injection**

**How DI Works in .NET Core**

**.NET Core has built-in dependency injection support via**

**Microsoft.Extensions.DependencyInjection.**

**IMessageService.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Mgm1

{

public interface IMessageService

{

void SendMessage(string message);

}

}

**ConsoleMessageService.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Mgm1

{

public class ConsoleMessageService : IMessageService

{

public void SendMessage(string message)

{

Console.WriteLine($"Message: {message}");

}

}

}

**Program.cs**

using Mgm1;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

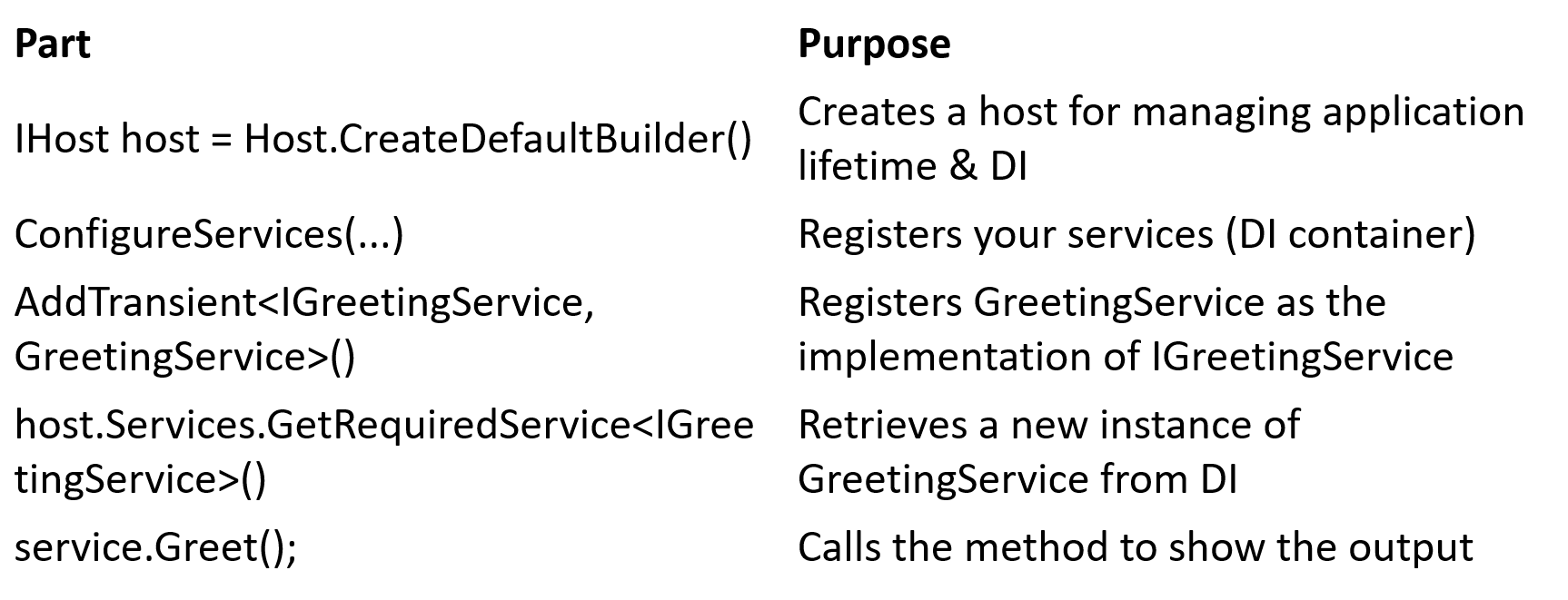
var host = Host.CreateDefaultBuilder(args)

.ConfigureServices(services =>

{

services.AddTransient<IMessageService, ConsoleMessageService>();

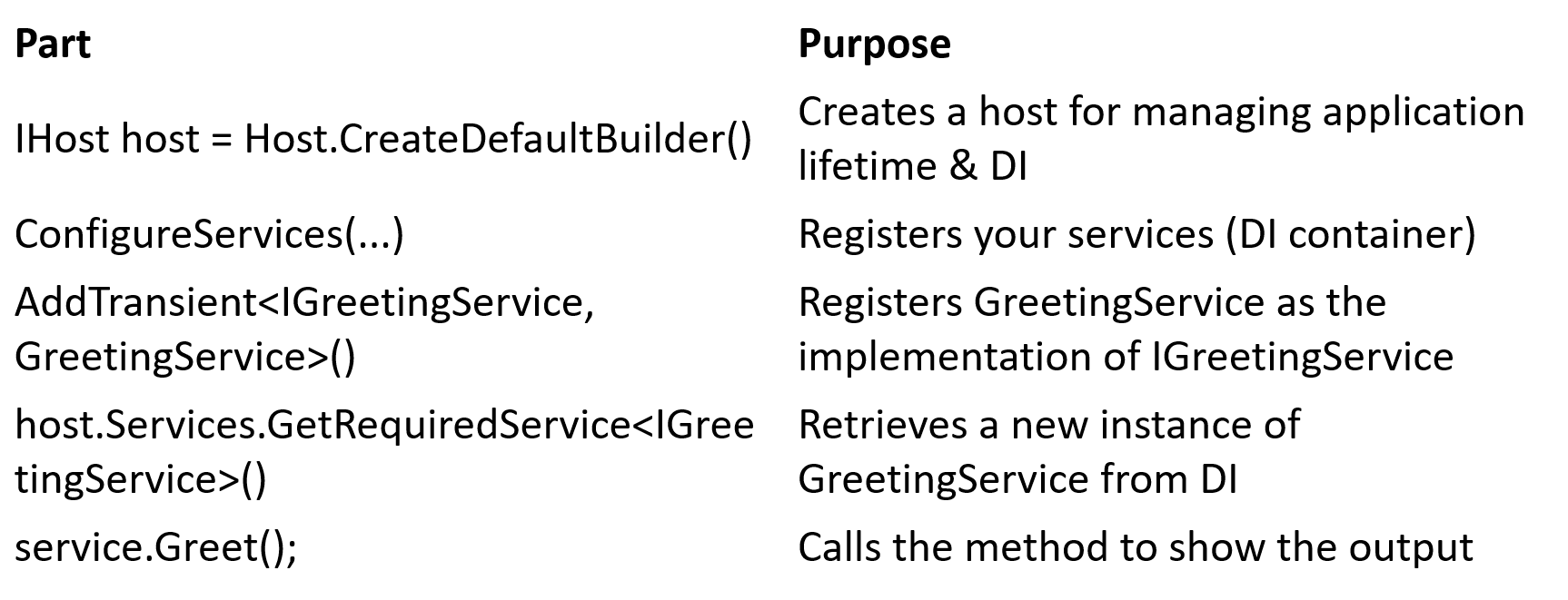
})

 .Build();

// Resolve the service

var messageService = host.Services.GetRequiredService<IMessageService>();

messageService.SendMessage("Hello from .NET 8 DI!");

****

**📘 Middleware Pipeline**

Middleware are components that process HTTP requests and responses in a pipeline fashion.

📌 Middleware can:

* Handle requests
* Modify requests/responses
* Short-circuit pipeline

**🔧 Common Middleware in ASP.NET Core**

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints => { endpoints.MapControllers(); });

Order matters in middleware pipeline!

**🔧 Custom Middleware**

**📘 What is Custom Middleware?**

A custom middleware allows you to add your own logic into the HTTP request processing pipeline (e.g., logging, error handling).

**🔧 Example: Logging Middleware**

**Create Middleware Class**

public class RequestLoggingMiddleware

{

private readonly RequestDelegate \_next;

public RequestLoggingMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext context)

{

Console.WriteLine($"Request: {context.Request.Method} {context.Request.Path}");

await \_next(context); // Call the next middleware

Console.WriteLine("Response completed");

}

}

**Register in Program.cs**

app.UseMiddleware<RequestLoggingMiddleware>();

**📝 Code Explanation**

| **Part** | **Explanation** |
| --- | --- |
| RequestDelegate \_next | Refers to the next middleware component in the pipeline |
| InvokeAsync() | Required method to execute the middleware logic |
| Console.WriteLine() | Logs request info before and after the next middleware |
| await \_next(context) | Passes control to the next component in the pipeline |

**✅ Summary**

| **Feature** | **Details** |
| --- | --- |
| Built-in DI | Automatically supported in ASP.NET Core |
| Register Services | Using AddTransient, AddScoped, or AddSingleton |
| Middleware Pipeline | Executes middleware in the order defined in Program.cs |
| Custom Middleware | You can define and inject your own logic like logging or error handling |

✅  **Multiple Choice Questions (MCQs)**

1.What is the default DI container in ASP.NET Core?

A) Ninject  
B) Autofac  
C) Built-in .NET Core Container  
D) StructureMap

**Answer: C) Built-in .NET Core Container**

2.Which method registers a service with one instance for the whole application?

A) AddTransient  
B) AddScoped  
C) AddSingleton  
D) AddInstance

**Answer: C) AddSingleton**

3.Middleware in ASP.NET Core is executed in:

A) Random order  
B) Reverse order  
C) Sequential order  
D) Parallel threads

**Answer: C) Sequential order**

4.Which DI lifetime creates a new instance every time?

A) Singleton  
B) Transient  
C) Scoped  
D) Static

**Answer: B) Transient**

5.What method is used to register custom middleware?

A) UseMiddleware  
B) AddMiddleware  
C) InjectMiddleware  
D) RegisterMiddleware

**Answer: A) UseMiddleware**

6.Which middleware is executed first?

A) Last one registered  
B) First one registered  
C) Depends on service lifetime  
D) Random

**Answer: B) First one registered**

7.Where do you register services in ASP.NET Core?

A) Configure()  
B) Program.cs (builder.Services)  
C) Startup.cs - Configure  
D) Views.cshtml

**Answer: B) Program.cs (builder.Services)**

8.What interface is required for custom services?

A) IMyService  
B) IService  
C) CustomService  
D) No specific interface required

**Answer: D) No specific interface required**

9.Which of the following can short-circuit the middleware pipeline?

A) Singleton service  
B) Middleware  
C) Controller  
D) Logger

**Answer: B) Middleware**

10.What type of service is most suitable for lightweight, short-lived objects?

A) Singleton  
B) Static  
C) Transient  
D) Scoped

**Answer: C) Transient**

**11. What is the default dependency injection technique used in ASP.NET Core?**  
A) Property Injection  
B) Method Injection  
C) Constructor Injection  
D) Field Injection  
**Answer:** C) Constructor Injection

**12. Which lifetime creates a new instance every time it is requested?**  
A) Scoped  
B) Singleton  
C) Transient  
D) Static  
**Answer:** C) Transient

**13. What is the correct method to register a service with Scoped lifetime?**  
A) AddSingleton<T>()  
B) AddTransient<T>()  
C) AddScoped<T>()  
D) Register<T>()  
**Answer:** C) AddScoped<T>()

**14. Middleware in ASP.NET Core is executed in which order?**  
A) Random  
B) Reverse registration order  
C) Based on priority  
D) In the order it is registered  
**Answer:** D) In the order it is registered

**15. What method must a custom middleware class implement?**  
A) Execute()  
B) Run()  
C) InvokeAsync()  
D) ProcessRequest()  
**Answer:** C) InvokeAsync()

**16. What is RequestDelegate in a middleware class?**  
A) A class to handle UI events  
B) A delegate to handle HTTP requests  
C) A controller class  
D) A startup configuration  
**Answer:** B) A delegate to handle HTTP requests

**17. Which DI lifetime is best for lightweight, stateless services?**  
A) Singleton  
B) Scoped  
C) Transient  
D) Static  
**Answer:** C) Transient

**18. In which file is the middleware pipeline defined in ASP.NET Core 6?**  
A) Startup.cs  
B) appsettings.json  
C) Program.cs  
D) web.config  
**Answer:** C) Program.cs

**19. Where should you register custom middleware in the request pipeline?**  
A) Before app.UseRouting()  
B) After app.MapControllers()  
C) After app.Run()  
D) Inside ConfigureServices()  
**Answer:** A) Before app.UseRouting()

**20. What happens if await \_next(context) is not called in middleware?**  
A) It logs an error  
B) Pipeline continues anyway  
C) The request is short-circuited  
D) It throws an exception  
**Answer:** C) The request is short-circuited

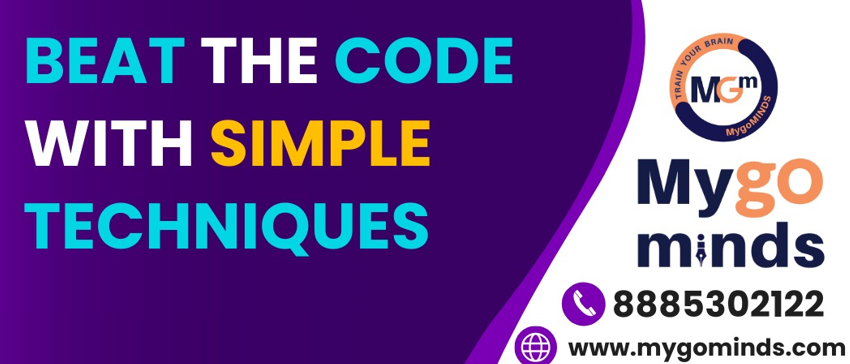
✍️ **Fill in the Blanks**

1. ASP.NET Core provides \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ dependency injection by default.  
   **Answer: built-in**
2. Services are registered in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   Answer: Program.cs (or builder.Services)
3. Middleware components are executed in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ order.  
   **Answer: registration**
4. The AddScoped method creates an instance per \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: HTTP request**
5. Custom middleware requires a method named \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: InvokeAsync**
6. Middleware has access to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ context.  
   **Answer: Http**
7. UseMiddleware method is called on the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ object.  
   **Answer: app**
8. Dependency Injection promotes \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ coupling.  
   **Answer: loose**
9. The RequestDelegate is used to call the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ middleware.  
   **Answer: next**
10. The default method to register services for one-time use is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
    **Answer: AddTransient**
11. ASP.NET Core uses \_\_\_\_\_\_\_\_\_\_ injection as the default DI mechanism.  
    **Answer:** Constructor
12. A \_\_\_\_\_\_\_\_\_\_ service is created once per client request (per scope).  
    **Answer:** Scoped
13. Middleware components are added to the request pipeline using \_\_\_\_\_\_\_\_\_\_ methods.  
    **Answer:** Use
14. The \_\_\_\_\_\_\_\_\_\_ method is required in a custom middleware class.  
    **Answer:** InvokeAsync
15. Middleware components must call \_\_\_\_\_\_\_\_\_\_ to pass control to the next component.  
    **Answer:** \_next(context)
16. \_\_\_\_\_\_\_\_\_\_ lifetime means the service is created once and reused across the app.  
    **Answer:** Singleton
17. Services are registered in the \_\_\_\_\_\_\_\_\_\_ method of Program.cs or Startup.cs.  
    **Answer:** ConfigureServices
18. \_\_\_\_\_\_\_\_\_\_ is a common use case for creating custom middleware.  
    **Answer:** Logging
19. ASP.NET Core middleware components form a \_\_\_\_\_\_\_\_\_\_ pipeline.  
    **Answer:** Request-processing
20. Middleware components can both handle and \_\_\_\_\_\_\_\_\_\_ requests.  
    **Answer:** Forward

✅  **True or False**

1. Middleware must have an InvokeAsync method.  
   **Answer: True**
2. Services cannot be registered without an interface.  
   **Answer: False**
3. Singleton services are created per request.  
   **Answer: False**
4. Middleware can modify both request and response.  
   **Answer: True**
5. UseMiddleware<MyMiddleware>() registers built-in middleware.  
   **Answer: False**
6. Transient services are created once and reused.  
   **Answer: False**
7. Middleware components are registered in Program.cs.  
   **Answer: True**
8. The DI container in .NET Core supports constructor injection.  
   **Answer: True**
9. Custom middleware requires a constructor accepting RequestDelegate.  
   **Answer: True**
10. Middleware is executed after the controller action.  
    **Answer: False**
11. Dependency Injection in ASP.NET Core is enabled by default.  
    **True**
12. Middleware in ASP.NET Core is executed randomly.  
    **False**
13. Singleton services are created once per HTTP request.  
    **False**
14. Custom middleware must define an InvokeAsync() method.  
    **True**
15. Services with Transient lifetime are reused throughout the application.  
    **False**
16. Middleware order in the pipeline does not matter.  
    **False**
17. You can register middleware using UseMiddleware<T>().  
    **True**
18. You must always call \_next(context) in middleware unless short-circuiting.  
    **True**
19. The RequestDelegate parameter refers to the next middleware in the pipeline.  
    **True**
20. ASP.NET Core does not support constructor injection.  
    **False**



****

**Chapter-6**

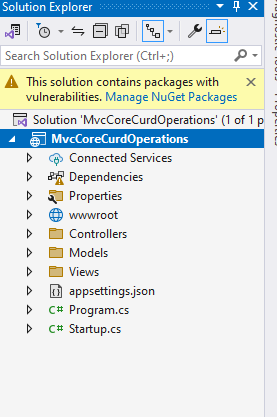
**ASP.NET Core MVC**

**📘 ASP.NET Core MVC**

**🔑 1. Introduction to MVC Pattern**

**MVC (Model-View-Controller)** is a design pattern that separates an application into three components:

* **Model**: Represents data and business logic.
* **View**: UI of the application.
* **Controller**: Handles user input, interacts with the model, and returns a view.



**Folder Structure Explanation**

## 📁 ****Connected Services****

* Used to **connect to external services** (e.g., REST APIs, Azure services, WCF).
* Helps integrate services like Azure Storage, Authentication, etc.

## 📁 ****Dependencies****

* Lists all **NuGet packages**, **SDKs**, and **project references** used in your project.
* You can manage versions and see if any package has vulnerabilities (as noted in the yellow warning).

## 📁 ****Properties****

* Contains the **launchSettings.json** file.
* This file defines how the app is started (e.g., browser launch, environment settings like Development or Production).

## 📁 ****wwwroot****

* The **web root folder** for serving static files like:
  + CSS
  + JavaScript
  + Images
* Publicly accessible — e.g., wwwroot/css/site.css becomes /css/site.css in the browser.

## 📁 ****Controllers****

* Contains all your **Controller classes** (e.g., HomeController.cs, EmployeeController.cs).
* Controllers handle requests, contain action methods, and return responses (like Views or JSON).

## 📁 ****Models****

* Contains **C# classes that represent data** (like Employee.cs, Product.cs).
* Used to define the structure and validation rules for your application's data.

## 📁 ****Views****

* Contains all **Razor View (.cshtml) files** grouped by controller name.
* Example:

Views/

Home/

Index.cshtml

Shared/

\_Layout.cshtml

## 📄 ****appsettings.json****

* Configuration file for settings like:
  + Connection strings
  + Logging levels
  + Custom app settings
* Example:

🔧 *Example Workflow*:

* User sends a request to /Home/Index.
* Controller (HomeController) handles it.
* Fetches data from Model.
* Returns a View to the browser.

**🔑 2. Controllers, Views, Models**

* **Controllers**: Handle requests and return responses.

public class HomeController : Controller

{

public IActionResult Index()

{

return View();

}

}

* **Models**: Hold application data.

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

}

* **Views**: Render HTML with Razor.

<h1>Hello @Model.Name</h1>

**🔑 3. Razor Syntax and Layouts**

* Razor allows C# code in HTML.
* Syntax: @variable, @if, @foreach

@foreach(var item in Model) {

<p>@item.Name</p>

}

* **\_Layout.cshtml**: Shared layout for all pages.

<body>

@RenderBody()

</body>

**🔑 4. Routing in ASP.NET Core**

* Controls how URLs are mapped to controllers/actions.

🔧 *Default Route* (in Program.cs):

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

* URL /Product/Details/3 → ProductController -> Details(int id)

**🔑 5. Model Binding and Validation**

* Automatically binds form values to model properties.
* Use attributes for validation:

public class Student

{

[Required]

public string Name { get; set; }

[Range(18, 60)]

public int Age { get; set; }

}

* In View:

<form asp-action="Create">

<input asp-for="Name" />

<span asp-validation-for="Name"></span>

</form>

**🔑 6. Tag Helpers and HTML Helpers**

**What are HTML Helpers?**

🔹 HTML Helpers are C# methods used in Razor views to generate HTML elements.

**🔧 Example:**

**@Html.TextBoxFor(model => model.Name)**

**@Html.LabelFor(model => model.Name)**

**What are Tag Helpers?**

🔹 Tag Helpers are HTML-like tags in Razor views that help render HTML dynamically using C#.

**🔧 Example:**

**<input asp-for="Name" class="form-control" />**

**<label asp-for="Name"></label>**

🔁 Difference Between HTML Helpers and Tag Helpers

| **Feature** | **HTML Helpers** | **Tag Helpers** |
| --- | --- | --- |
| Syntax | C# method-style | HTML-like attribute-based |
| Readability | Less readable for non-C# developers | More readable and closer to HTML |
| IntelliSense Support | Limited | Full IntelliSense in HTML editor |
| Customization | Harder | Easier with custom Tag Helpers |
| HTML Look | Doesn’t look like HTML | Looks like real HTML |
| Binding with Model | Yes | Yes |

✅ When to Use What?

 Use **Tag Helpers** if you prefer **cleaner and more HTML-like syntax** (recommended in modern ASP.NET Core).

 Use **HTML Helpers** if you’re working in **older MVC projects** or need method-based customization.

✅  **Multiple Choice Questions (MCQs)**

1.Which component is responsible for handling user input in MVC?

A) Model  
B) View  
C) Controller  
D) Startup

**Answer: C) Controller**

2.Razor syntax allows mixing of:

A) HTML and SQL  
B) JavaScript and CSS  
C) HTML and C#  
D) JSON and XML

**Answer: C) HTML and C#**

3.What is the default file used for layout in ASP.NET Core MVC?

A) site.cshtml  
B) \_Master.cshtml  
C) \_Layout.cshtml  
D) viewstart.cshtml

**Answer: C) \_Layout.cshtml**

4.Which attribute validates required input?

A) [String]  
B) [Check]  
C) [Required]  
D) [NotEmpty]

**Answer: C) [Required]**

5.What does asp-for Tag Helper bind to?

A) Action method  
B) Controller  
C) Model property  
D) Layout file

**Answer: C) Model property**

6.What file defines the default route in .NET 6+?

A) startup.cs  
B) \_Layout.cshtml  
C) appsettings.json  
D) Program.cs

**Answer: D) Program.cs**

7.HTML Helpers return:

A) JavaScript code  
B) Controller logic  
C) C# methods  
D) HTML markup

**Answer: D) HTML markup**

8.What Razor directive includes layout for views?

A) @Layout  
B) @SetLayout  
C) @UseLayout  
D) @Page

**Answer: A) @Layout**

9.What is the role of the Model in MVC?

A) Display UI  
B) Handle routing  
C) Store and manage data  
D) Render Razor pages

**Answer: C) Store and manage data**

10.Which helper is used for displaying validation messages?

A) @Html.LabelFor  
B) @Html.TextBox  
C) @Html.ValidationMessageFor  
D) @Html.Error

**Answer: C) @Html.ValidationMessageFor**

**11.** Which method in a controller returns a view to the user?  
o A) Show()  
o B) View()  
o C) Display()  
o D) Render()  
**Answer: B) View()**

**12.** Which Razor directive is used to define a section in a view?  
o A) @section  
o B) @render  
o C) @block  
o D) @area  
**Answer: A) @section**

**13.** Which method is used to redirect to another action?  
o A) Redirect()  
o B) GoToAction()  
o C) RedirectToAction()  
o D) MoveTo()  
**Answer: C) RedirectToAction()**

**14.** Where are controller classes typically located in an MVC project?  
o A) /Views  
o B) /Controllers  
o C) /Models  
o D) /Pages  
**Answer: B) /Controllers**

**15.** What does the [Bind] attribute do?  
o A) Hides the property  
o B) Binds only specified properties  
o C) Prevents model binding  
o D) Enables JavaScript binding  
**Answer: B) Binds only specified properties**

**16.** Which tag helper sets the form’s submission target to a controller action?  
o A) asp-action  
o B) asp-route  
o C) asp-submit  
o D) asp-controller  
**Answer: A) asp-action**

**17.** What does ModelState.IsValid check for?  
o A) Controller errors  
o B) Data formatting  
o C) Validation attribute compliance  
o D) Razor syntax errors  
**Answer: C) Validation attribute compliance**

**18.** What is the purpose of @Html.EditorFor(model => model.Name)?  
o A) Display name as plain text  
o B) Provide a rich-text editor  
o C) Generate input control for the property  
o D) Add model to database  
**Answer: C) Generate input control for the property**

**19.** What is ViewBag used for?  
o A) Strongly-typed model  
o B) Temporary key-value storage between controller and view  
o C) Routing configuration  
o D) File uploads  
**Answer: B) Temporary key-value storage between controller and view**

**20.** What does [HttpPost] attribute specify?  
o A) Only allow AJAX requests  
o B) Allow GET requests  
o C) Only handle POST requests  
o D) Handle all HTTP methods  
**Answer: C) Only handle POST requests**

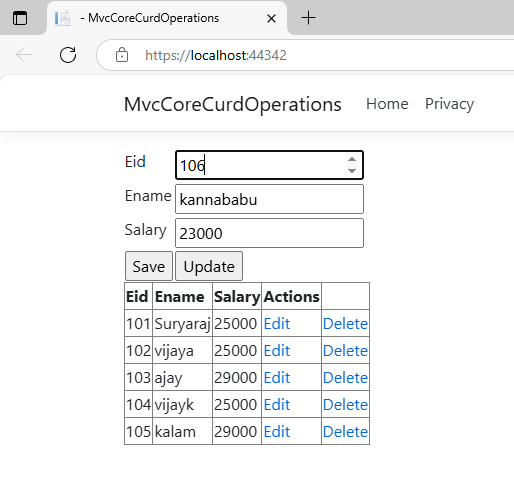
✍️ **Fill in the Blanks**

1. MVC stands for \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: Model View Controller**
2. The default method that handles incoming request is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: Index**
3. Razor syntax starts with \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ symbol.  
   **Answer: @**
4. asp-for tag helper is used to bind to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: model properties**
5. The Required attribute ensures a field is not \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: empty**
6. Views are stored inside the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ folder.  
   **Answer: Views**
7. Controllers return an object of type \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: IActionResult**
8. The method Html.TextBoxFor is an example of a(n) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: HTML Helper**
9. The shared layout is typically named \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: \_Layout.cshtml**
10. Validation attributes are defined in the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
    **Answer: model class**

✅  **True or False**

1. Controllers in ASP.NET Core must inherit from Controller class.  
   **Answer: True**
2. Razor code can include loops and conditions.  
   **Answer: True**
3. You cannot use both Tag Helpers and HTML Helpers together.  
   **Answer: False**
4. The route pattern {controller}/{action}/{id?} requires id.  
   **Answer: False**
5. Validation is done only on the client-side.  
   **Answer: False**
6. \_ViewStart.cshtml sets a default layout for views.  
   **Answer: True**
7. You can define custom validation attributes in ASP.NET Core MVC.  
   **Answer: True**
8. HTML Helpers generate C# code.  
   **Answer: False**
9. Routing maps URLs to controller methods.  
   **Answer: True**
10. The ModelState.IsValid property checks for validation success.  
    **Answer: True**

**MVC Core CURD Operations using ADO.NET**

****

Create table Employee(Eid int primary key,Ename varchar(30),Salary money)

Create procedure sp\_add@eid int,@ename varchar(30),@salary money

As begin

Insert into Employee values(@eid,@ename,@salary)

End;

Create procedure sp\_edit@eid int,@ename varchar(30),@salary money

As begin

Update Employee set Ename=@ename,Salary=@salary where Eid=@eid

End;

Create procedure sp\_view

As begin

Select \* from Employee

End;

Create procedure sp\_delete @eid

As begin

Delete from Employee where eid=@eid

End;

open visual studio 2022->

create new project->asp.net webapplication(model-view-controller)->next->

project name:MVCCORECURD

next->.dot net core 8.0(long-term-support->create

click on tools->install nuget manager->manage nuget packages for solution

click on-setting icon

click on-add icon

name:package source1

source:https://packages.nuget.org/api/v2

->update

click on-add icon

name:package source2

source:https://api.nuget.org/v3/index.json

->update

->ok

browse: Microsoft.Data.SqlClient ->install->I Accept->ok

Appsetting.json

"ConnectionStrings": {

"SetConnection": "server=localhost;Initial Catalog=mygominds;Integrated Security=SSPI;TrustServerCertificate=True;"

}

Right click on Model->add->class->name:EmpModel->add

using Microsoft.Data.SqlClient;

namespace MVCCORECURD.Models

{

public class EmpModel

{

public int Eid { get; set; }

public string? Ename { get; set; }

public double Salary { get; set; }

}

}

right click on Model->add class->name:EmpRepository->add

using Microsoft.Data.SqlClient;

using System.Data;

namespace MVCCORECURD.Models

{

public class EmpRepository

{

string constring;

public EmpRepository(IConfiguration configuration)

{

constring = configuration.GetConnectionString("getcon")!;

}

public void Adduser(EmpModel emp)

{

SqlConnection con = new SqlConnection(constring);

SqlCommand cmd = new SqlCommand("sp\_add", con);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@eid", emp.Eid);

cmd.Parameters.AddWithValue("@ename", emp.Ename);

cmd.Parameters.AddWithValue("@salary", emp.Salary);

con.Open();

cmd.ExecuteNonQuery();

con.Close();

}

public List<EmpModel> GetAllEmps()

{

List<EmpModel> emps = new List<EmpModel>();

SqlConnection con = new SqlConnection(constring);

con.Open();

SqlCommand cmd = new SqlCommand("sp\_view", con);

SqlDataAdapter da = new SqlDataAdapter(cmd);

DataTable dt = new DataTable();

da.Fill(dt);

con.Close();

foreach (DataRow dr in dt.Rows)

{

emps.Add(new EmpModel()

{

Eid = Convert.ToInt32(dr["Eid"]),

Ename = Convert.ToString(dr["Ename"]),

Salary = Convert.ToDouble(dr["Salary"])

});

}

return emps;

}

public void UpdateUser(EmpModel emp)

{

SqlConnection con = new SqlConnection(constring);

SqlCommand cmd = new SqlCommand("sp\_edit", con);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@eid", emp.Eid);

cmd.Parameters.AddWithValue("@ename", emp.Ename);

cmd.Parameters.AddWithValue("@salary", emp.Salary);

con.Open();

cmd.ExecuteNonQuery();

con.Close();

}

public void DeleteEmp(int id)

{

SqlConnection con = new SqlConnection(constring);

SqlCommand cmd = new SqlCommand("sp\_delete", con);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@eid", id);

con.Open();

cmd.ExecuteNonQuery();

con.Close();

}

}

}

right click on controller->add controller->mvc5 controller empty->

name:EmpController->add

using Microsoft.AspNetCore.Mvc;

using MVCCORECURD.Models;

namespace MVCCORECURD.Controllers

{

public class EmpController : Controller

{

private readonly EmpRepository empRepo;

public EmpController(EmpRepository \_empRepo)

{

this.empRepo = \_empRepo;

}

[HttpGet]

public IActionResult Index(int? id)

{

var employees = empRepo.GetAllEmps();

EmpModel empModel = new EmpModel();

if (id.HasValue)

{

empModel = employees.FirstOrDefault(e => e.Eid == id.Value) ?? new EmpModel();

}

ViewBag.Emplist = employees;

return View(empModel);

}

[HttpPost]

public IActionResult Index( EmpModel emp, string str)

{

if (str == "Save")

{

empRepo.Adduser(emp);

return RedirectToAction("Index");

}

else if (str == "Update")

{

empRepo.UpdateUser(emp);

return RedirectToAction("Index");

}

return View();

}

[HttpGet]

public IActionResult Delete(int id)

{

{

empRepo.DeleteEmp(id);

return RedirectToAction("Index");

}

}

}

}

Right click on -Index()in [HttpPost]->add view->razor view empty->

Name:Index->add

@model MVCCORECURD.Models.EmpModel

<**form** **asp-controller**="Emp" **asp-action**="Index" method="post">

<table>

<tr>

<td><**label** **asp-for**="Eid"></**label**></td>

<td><**input** **asp-for**="Eid" readonly="@(Model.Eid>0)" placeholder="New Id" /></td>

</tr>

<tr>

<td><**label** **asp-for**="Ename"></**label**></td>

<td><**input** **asp-for**="Ename" /></td>

</tr>

<tr>

<td><**label** **asp-for**="Salary"></**label**></td>

<td><**input** **asp-for**="Salary" /></td>

</tr>

<tr>

<td><input type="submit" value="Save" name="str" /></td>

<td><input type="submit" value="Update" name="str" /></td>

</tr>

</table>

</**form**>

@if (ViewBag.EmpList != null)

{

var employeeList = ViewBag.EmpList as List<MVCCORECURD.Models.EmpModel>;

if (employeeList != null)

{

<table border="1">

<tr>

<th> Eid</th>

<th> Ename</th>

<th> Salary</th>

<th> Actions</th>

</tr>

@foreach (var item in employeeList)

{

<tr>

<td>@item.Eid</td>

<td>@item.Ename</td>

<td>@item.Salary</td>

<td>

@Html.ActionLink("Edit", "Index", new { id = item.Eid }) |

@Html.ActionLink("Delete", "Delete", new { id = item.Eid }, new { onclick = "return confirm('Are you sure delete?');" })

</td>

</tr>

}

</table>

}

else

{

<p>No employees found.</p>

}

}

else

{

<p>Employee list not available.</p>

}

Program.ts

using MVCCORECURD.Models;

var builder = WebApplication.CreateBuilder(args);

// Register EmpRepository

builder.Services.AddScoped<EmpRepository>();

-------------------------------

--------------------------

pattern: "{controller=Emp}/{action=Index}/{id?}");

run application

✅ ASP.NET Core 3.1 MVC CRUD – MCQs (Answers after options)

1. What does EmpModel represent in this application?
   * A) View
   * B) Controller
   * C) Entity class for Employee
   * D) Razor page  
     **Answer: C**
2. What is the purpose of EmpRepository?
   * A) UI management
   * B) Business logic
   * C) Database interaction
   * D) Middleware configuration  
     **Answer: C**
3. What type of procedure is sp\_add?
   * A) Trigger
   * B) Scalar function
   * C) Stored procedure
   * D) Inline function  
     **Answer: C**
4. Which namespace is used for SQL operations?
   * A) System.Sql
   * B) Microsoft.Data.SqlClient
   * C) Microsoft.Sql
   * D) Data.Sql  
     **Answer: B**
5. What is the HTTP verb used for saving/updating data?
   * A) [HttpDelete]
   * B) [HttpGet]
   * C) [HttpPost]
   * D) [HttpPut]  
     **Answer: C**
6. What does cmd.CommandType = CommandType.StoredProcedure; do?
   * A) Executes inline SQL
   * B) Executes view
   * C) Calls a function
   * D) Executes a stored procedure  
     **Answer: D**
7. What does ViewBag.EmpList hold?
   * A) List of views
   * B) List of employees
   * C) List of controllers
   * D) List of models  
     **Answer: B**
8. What kind of parameter is used in cmd.Parameters.AddWithValue(...)?
   * A) Inline string
   * B) Hardcoded
   * C) SQL parameter
   * D) XML  
     **Answer: C**
9. What is the role of asp-controller in Razor view?
   * A) Set page title
   * B) Define route
   * C) Apply styling
   * D) Enable JS  
     **Answer: B**
10. What will the readonly="@(Model.Eid > 0)" attribute do?
    * A) Editable always
    * B) Read-only if new
    * C) Read-only if Eid exists
    * D) Always hidden  
      **Answer: C**
11. Which file contains the connection string?
    * A) Program.cs
    * B) appsettings.json
    * C) Startup.cs
    * D) launchSettings.json  
      **Answer: B**
12. What does builder.Services.AddScoped<EmpRepository>(); do?
    * A) Registers controller
    * B) Registers singleton service
    * C) Registers scoped service
    * D) Adds view component  
      **Answer: C**
13. What is used to prevent SQL Injection in this app?
    * A) Inline queries
    * B) SqlDataAdapter
    * C) Parameterized queries
    * D) ViewBag  
      **Answer: C**
14. Which method is used to retrieve all records?
    * A) sp\_add
    * B) sp\_edit
    * C) sp\_view
    * D) sp\_delete  
      **Answer: C**
15. The use of @Html.ActionLink in the view allows for:
    * A) CSS injection
    * B) JavaScript execution
    * C) Razor conditionals
    * D) Navigation link generation  
      **Answer: D**
16. How is the employee data shown in the form?
    * A) ViewData
    * B) ViewBag
    * C) Model Binding
    * D) TempData  
      **Answer: C**
17. In which method is SqlDataAdapter used?
    * A) Adduser
    * B) UpdateUser
    * C) GetAllEmps
    * D) DeleteEmp  
      **Answer: C**
18. What Razor syntax is used for looping through employee data?
    * A) @for
    * B) @foreach
    * C) @loop
    * D) @while  
      **Answer: B**
19. Integrated Security=SSPI means:
    * A) Uses app authentication
    * B) Uses Windows Authentication
    * C) Uses SQL login
    * D) No authentication  
      **Answer: B**
20. Which action handles both Save and Update?
    * A) Index() [HttpGet]
    * B) Index() [HttpPost]
    * C) Delete()
    * D) Create()  
      **Answer: B**
21. How is the database connection opened?
    * A) con.Open()
    * B) cmd.Open()
    * C) new SqlCommand.Open()
    * D) connection.OpenConnection()  
      **Answer: A**
22. empRepo.UpdateUser(emp); is called when:
    * A) Eid is 0
    * B) Salary is updated
    * C) Button value is "Update"
    * D) Form is not submitted  
      **Answer: C**
23. What happens when Delete action is triggered?
    * A) Adds new employee
    * B) Updates salary
    * C) Removes employee from DB
    * D) Shows error  
      **Answer: C**
24. Razor syntax asp-for="Ename" binds:
    * A) CSS classes
    * B) Model property
    * C) JS functions
    * D) SQL query  
      **Answer: B**
25. What does DataTable dt = new DataTable(); do?
    * A) Deletes data
    * B) Creates new SQL table
    * C) Holds data temporarily
    * D) Sends data to view  
      **Answer: C**
26. Where is dependency injection configured?
    * A) appsettings.json
    * B) EmpModel.cs
    * C) Program.cs
    * D) EmpRepository.cs  
      **Answer: C**
27. What is the use of cmd.ExecuteNonQuery();?
    * A) Run a scalar query
    * B) Run a reader
    * C) Execute insert/update/delete
    * D) Return DataTable  
      **Answer: C**
28. What is the default route pattern in your app?
    * A) "{controller}/{action}/{id}"
    * B) "{controller=Emp}/{action=Index}/{id?}"
    * C) "{controller=Home}/{action=Index}"
    * D) "{action}/{controller}/{id}"  
      **Answer: B**
29. Which type of SQL object is sp\_view?
    * A) Table
    * B) Function
    * C) Stored procedure
    * D) Trigger  
      **Answer: C**
30. Why is readonly="@(Model.Eid > 0)" used for Eid?
    * A) Prevent duplicate ID
    * B) Hide the field
    * C) Convert to dropdown
    * D) Enable password mode  
      **Answer: A**
31. What does ModelState.IsValid check in the controller?

* A) If SQL connection is active
* B) If Razor syntax is correct
* C) If form data passes validation
* D) If user is authenticated  
  **Answer: C**

1. The command cmd.Parameters.AddWithValue("@Ename", emp.Ename); maps:

* A) SQL data to view
* B) ViewBag to model
* C) Razor data to SQL
* D) Model data to SQL parameter  
  **Answer: D**

1. Which Razor directive is used to declare the model at the top of the view?

* A) @class
* B) @page
* C) @model
* D) @using  
  **Answer: C**

1. What is the file extension for Razor views?

* A) .razor
* B) .aspx
* C) .cshtml
* D) .html  
  **Answer: C**

1. In Razor, @Html.HiddenFor(m => m.Eid) is used to:

* A) Show Eid in bold
* B) Hide Eid visually but include in form
* C) Disable Eid permanently
* D) Encrypt Eid  
  **Answer: B**

1. The SQL statement inside sp\_delete likely contains:

* A) INSERT INTO
* B) UPDATE
* C) SELECT
* D) DELETE FROM  
  **Answer: D**

1. public IActionResult Delete(int id) receives:

* A) Model object
* B) ViewBag
* C) Route parameter
* D) Connection string  
  **Answer: C**

1. What is the purpose of the return RedirectToAction("Index"); statement?

* A) Reloads the same page
* B) Redirects to another controller
* C) Redirects to Index action
* D) Restarts the app  
  **Answer: C**

1. Which statement is used to load data into a DataTable from a DataAdapter?

* A) adapter.Fill(dt);
* B) dt.Load();
* C) cmd.Fill();
* D) fill.Data();  
  **Answer: A**

1. What does the .ToList() method do in the repository?

* A) Runs a query
* B) Converts data to list format
* C) Saves to DB
* D) Deletes items  
  **Answer: B**

1. Which Razor tag helper generates a form?

* A) <form asp-action="Index">
* B) <form-data>
* C) <razor-form>
* D) <form helper>  
  **Answer: A**

1. What does DataRow row in dt.Rows represent?

* A) SQL rows
* B) HTML rows
* C) Rows in memory table
* D) Controller methods  
  **Answer: C**

1. What kind of project is created for this app?

* A) ASP.NET Core Web API
* B) ASP.NET Core MVC
* C) Razor Class Library
* D) WinForms  
  **Answer: B**

1. Which method checks if the form was submitted successfully?

* A) ModelState.IsValid
* B) HttpContext.IsLive
* C) form.Submitted
* D) Response.End  
  **Answer: A**

1. using(SqlConnection con = new SqlConnection(...)) ensures:

* A) Manual disposal
* B) Connection pooling
* C) Auto disposal after use
* D) Long-lived connection  
  **Answer: C**

1. The view shows employee data using:

* A) ViewBag
* B) TempData
* C) ViewModel
* D) List<EmpModel> passed via Model  
  **Answer: D**

1. The readonly attribute in Razor is used to:

* A) Prevent deletion
* B) Prevent editing
* C) Disable form submission
* D) Validate input  
  **Answer: B**

1. SqlConnection, SqlCommand, and SqlDataAdapter come from which library?

* A) System.Data.SqlClient
* B) System.IO.Sql
* C) Microsoft.Sql.Net
* D) ASP.Data.Sql  
  **Answer: A**

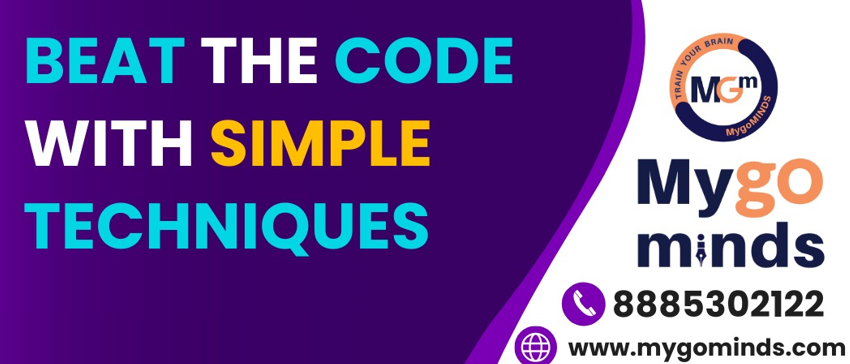
1. Which class is injected into the controller via constructor?

* A) ViewModel
* B) EmpModel
* C) EmpRepository
* D) SqlCommand  
  **Answer: C**

1. What happens when cmd.ExecuteNonQuery() is called in Adduser?

* A) Data is inserted
* B) Data is shown
* C) Form resets
* D) Connection opens  
  **Answer: A**



****

**Chapter-7**

**Entity Framework Core**

**📘 What is EF Core in ASP.NET Core?**

**EF Core** (Entity Framework Core) is an open-source, lightweight, cross-platform **ORM (Object Relational Mapper)** used with ASP.NET Core to interact with databases using C# objects instead of SQL queries.

**🔑 Key Features:**

* Allows developers to work with databases using .NET objects
* Supports **Code-First**, **Database-First**, and **Migration** approaches
* Works with **SQL Server**, **SQLite**, **PostgreSQL**, **MySQL**, etc.
* Supports **LINQ** for querying data
* Integrated with ASP.NET Core's **Dependency Injection (DI)** system

**🔧 Small Code Example:**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

}

public class AppDbContext : DbContext

{

public DbSet<Product> Products { get; set; }

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

}

In Startup.cs or Program.cs:

services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

**EF Core (Entity Framework Core)** is an **Object-Relational Mapper (ORM)** for .NET, provided by Microsoft. It allows developers to interact with a **relational database** (like SQL Server, PostgreSQL, MySQL, etc.) using **.NET objects and LINQ** instead of raw SQL queries.

### 🔑 Key Features of EF Core:

* ✅ **Cross-platform** (Windows, Linux, macOS)
* ✅ **Lightweight and extensible**
* ✅ Supports **LINQ queries**, **change tracking**, **migrations**, and **schema management**
* ✅ Can work with different databases (via providers)
* ✅ Integrates easily with **ASP.NET Core** projects

### 🔧 Basic Example

#### 1️ Define the Model

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

}

#### 2️ Create a DbContext

public class AppDbContext : DbContext

{

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

public DbSet<Product> Products { get; set; }

}

#### 3️ Register EF Core in Program.cs

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

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

#### 4️ Use in a Controller or Service

private readonly AppDbContext \_context;

public ProductService(AppDbContext context)

{

\_context = context;

}

public async Task<List<Product>> GetAllProductsAsync()

{

return await \_context.Products.ToListAsync();

}

### 📦 Supported Databases

* SQL Server
* SQLite
* PostgreSQL
* MySQL
* Oracle (via third-party)
* InMemory (for testing)

### 📝 Summary

| **Feature** | **Description** |
| --- | --- |
| ORM Tool | Maps .NET classes to database tables |
| Fluent API & Data Annotations | Used to configure model relationships and constraints |
| LINQ Support | Query data using LINQ expressions |
| Migration Support | Automatically create/update database schema |
| Dependency Injection | Easily injected into services and controllers using DI |

**📘 Difference Between EF (Entity Framework) and EF Core**

| **Feature/Aspect** | **Entity Framework (EF)** | **Entity Framework Core (EF Core)** |
| --- | --- | --- |
| **Platform** | .NET Framework (Windows only) | .NET Core & .NET (Cross-platform) |
| **Performance** | Slower | Faster and more optimized |
| **Modular Design** | Monolithic | Modular and lightweight |
| **LINQ Support** | Available | Improved and extended LINQ support |
| **Migrations** | Available | Enhanced, CLI-supported migrations |
| **Batch Operations** | Not supported | Supported |
| **Shadow Properties** | Not available | Available |
| **Support for NoSQL** | Not supported | Work in progress (experimental) |
| **Change Tracking** | Basic | Advanced and efficient tracking |
| **Mapping Config** | Via XML or Fluent API | Primarily via Fluent API |

**🔧 Summary:**

* **EF** is older and tightly coupled to the Windows-based .NET Framework.
* **EF Core** is the modern, cross-platform version optimized for ASP.NET Core development with better performance, flexibility, and new features.

**🔑 1. EF Core Introduction**

**Entity Framework Core** is an ORM (Object-Relational Mapper) for .NET that enables developers to interact with databases using C# objects instead of SQL queries.

✅ Lightweight, cross-platform, and extensible.  
✅ Supports LINQ queries, change tracking, schema migrations, and more.

**🔑 2. Database-First vs Code-First**

| **Approach** | **Description** |
| --- | --- |
| **Code-First** | Define models in code, then generate the database. |
| **Database-First** | Generate models from an existing database using scaffolding. |

🔧 **Code-First Example:**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

}

🔧 **Database-First Example Command:**

dotnet ef dbcontext scaffold "Connection\_String" Microsoft.EntityFrameworkCore.SqlServer -o Models

**🔑 3. DbContext and DbSet**

DbContext is the bridge between your C# code and the database.  
DbSet<TEntity> represents a table in the database.

🔧 **Example:**

public class AppDbContext : DbContext

{

public DbSet<Product> Products { get; set; }

}

**🔑 4. LINQ Queries**

You can query the database using LINQ (Language Integrated Query).

🔧 **Example:**

var products = context.Products.Where(p => p.Name.Contains("Phone")).ToList();

**🔑 5. Migrations and Updating DB**

Migrations help you update the database schema as your models evolve.

🔧 **Common Commands:**

dotnet ef migrations add InitialCreate

dotnet ef database update

✅  **Multiple Choice Questions (MCQs)**

1.What is EF Core?

A) UI Framework  
B) ORM Tool  
C) Web Server  
D) JavaScript Library

**Answer: B) ORM Tool**

2.In Code-First, the database is created from:

A) LINQ Queries  
B) Existing Tables  
C) C# Classes  
D) SQL Scripts

**Answer: C) C# Classes**

3.Which class is used to interact with the database in EF Core?

A) DbQuery  
B) DbTable  
C) DbContext  
D) SqlHelper

**Answer: C) DbContext**

4.What does DbSet<T> represent?

A) Database  
B) Table  
C) Row  
D) Column

**Answer: B) Table**

5.Which command adds a new migration?

A) dotnet ef build  
B) dotnet ef database update  
C) dotnet ef migrations add  
D) dotnet ef restore

**Answer: C) dotnet ef migrations add**

6.Which approach requires a connection to an existing database?

A) Code-First  
B) Table-First  
C) Model-First  
D) Database-First

**Answer: D) Database-First**

7.What is the output of dotnet ef migrations add?

A) Updates database  
B) Deletes model  
C) Creates migration files  
D) Connects database

**Answer: C) Creates migration files**

8.Which namespace is used for EF Core?

A) System.Data.SqlClient  
B) Microsoft.SqlServer  
C) Microsoft.EntityFrameworkCore  
D) System.Data.Linq

**Answer: C) Microsoft.EntityFrameworkCore**

9.What does context.SaveChanges() do?

A) Saves model  
B) Commits changes to DB  
C) Deletes context  
D) Updates migration

**Answer: B) Commits changes to DB**

10.Which method retrieves data from a table?

A) Add()  
B) Remove()  
C) Find()  
D) Execute()

**Answer: C) Find()**

11. What does DbContext act as in EF Core?  
A) Web Controller  
B) Bridge between DB and code  
C) Table creator  
D) Query optimizer  
**Answer: B**

12. Which method is used to configure a database connection string in DbContext?  
A) Configure()  
B) SetConnection()  
C) OnConfiguring()  
D) Connect()  
**Answer: C**

13. EF Core uses which of the following to translate LINQ queries into SQL?  
A) Compiler  
B) SQL Engine  
C) Query Provider  
D) SQL Builder  
**Answer: C**

14. Which command updates the database schema to the latest migration?  
A) dotnet ef schema update  
B) dotnet ef update schema  
C) dotnet ef database update  
D) dotnet ef schema refresh  
**Answer: C**

15**.** Which method is used to insert a new record?  
A) Add()  
B) Insert()  
C) Save()  
D) Create()  
**Answer: A**

16. What does ToList() do in EF Core?  
A) Returns one record  
B) Deletes records  
C) Converts query result to a list  
D) Adds data to DB  
**Answer: C**

17. Lazy loading requires what type of navigation property?  
A) Static  
B) Virtual  
C) Abstract  
D) Readonly  
**Answer: B**

18. Which EF Core command is used to remove a record from a table?  
A) Drop()  
B) Remove()  
C) Delete()  
D) Clear()  
**Answer: B**

19. How do you filter data in EF Core?  
A) Search()  
B) FilterBy()  
C) Select()  
D) Where()  
**Answer: D**

20. What is the default fetch behavior in EF Core?  
A) Eager loading  
B) Lazy loading  
C) Explicit loading  
D) All of the above  
**Answer: B**

21. Which EF Core feature automatically maps C# classes to database tables?  
A) Dependency Injection  
B) Code-first mapping  
C) Reflection  
D) Lazy loading  
**Answer: B**

22. In which file is the connection string usually stored in ASP.NET Core?  
A) Startup.cs  
B) Program.cs  
C) appsettings.json  
D) web.config  
**Answer: C**

23**.** Which method is used to begin configuring relationships between entities?  
A) UseModel()  
B) OnModelCreating()  
C) ConfigureModel()  
D) SetupModel()  
**Answer: B**

24. How can you fetch related data using EF Core?  
A) WithTable()  
B) Join()  
C) Include()  
D) Related()  
**Answer: C**

25. What does migrationBuilder.CreateTable() do?  
A) Deletes a table  
B) Creates a migration  
C) Defines a table in migration file  
D) Syncs models  
**Answer: C**

26. What is Fluent API used for?  
A) Writing SQL queries  
B) Advanced configuration of models  
C) Writing LINQ  
D) Logging database changes  
**Answer: B**

27. Which method is used to remove tracking from queries?  
A) AsNoTracking()  
B) DisableTracking()  
C) NoTracking()  
D) TrackOff()  
**Answer: A**

28. What is eager loading in EF Core?  
A) Loads data on demand  
B) Loads related data immediately  
C) Loads data after SaveChanges  
D) Skips related data  
**Answer: B**

29. What is shadow property in EF Core?  
A) Hidden DB table  
B) Property not defined in class but tracked by EF  
C) Static property  
D) Deprecated property  
**Answer: B**

30**.** What’s the primary benefit of using migrations?  
A) Data backup  
B) Automatic UI generation  
C) Schema version control  
D) Network optimization  
**Answer: C**

31. Which method is used to update a record in EF Core?  
A) Change()  
B) Modify()  
C) Update()  
D) Save()  
**Answer: C**

32. What is HasKey() used for in Fluent API?  
A) Set a foreign key  
B) Set a primary key  
C) Configure lazy loading  
D) Enable encryption  
**Answer: B**

33**.** What is the purpose of HasOne() and WithMany()?  
A) Logging  
B) Indexing  
C) Defining relationships  
D) Configuration binding  
**Answer: C**

34. What does Scaffold-DbContext command do?  
A) Runs migrations  
B) Drops all tables  
C) Generates models from existing database  
D) Creates a new DB  
**Answer: C**

35. What does HasData() method do in EF Core?  
A) Validates input  
B) Tracks queries  
C) Seeds initial data  
D) Connects database  
**Answer: C**

36**.** Which relationship is defined using HasMany().WithOne()?  
A) One-to-One  
B) Many-to-Many  
C) Many-to-One  
D) One-to-Many  
**Answer: D**

37**.** What kind of property is Id conventionally treated as in EF Core?  
A) Foreign Key  
B) Primary Key  
C) Index  
D) Computed Column  
**Answer: B**

38**.** How do you delete the database in EF Core?  
A) context.DropDatabase()  
B) context.RemoveDb()  
C) context.Database.EnsureDeleted()  
D) context.Delete()  
**Answer: C**

39. Can EF Core work without migrations?  
A) Yes, with EnsureCreated()  
B) No, it's mandatory  
C) Only for SQLite  
D) Only in Web Apps  
**Answer: A**

40**.** Which file contains the Up() and Down() methods for migrations?  
A) DbContext.cs  
B) Program.cs  
C) Migration file  
D) Startup.cs  
**Answer: C**

41. What is the return type of Find() method?  
A) int  
B) IQueryable  
C) Object  
D) Entity  
**Answer: D**

42**.** Which method rolls back the last migration?  
A) dotnet ef rollback  
B) dotnet ef migrations remove  
C) dotnet ef reset  
D) dotnet ef revert  
**Answer: B**

43. Which EF Core feature helps in tracking changes to entities?  
A) Snapshot tracking  
B) Query monitor  
C) Change Tracker  
D) Logger  
**Answer: C**

44. What’s the use of Database.Migrate() in code?  
A) Rebuilds the DB  
B) Creates a migration  
C) Applies pending migrations at runtime  
D) Resets schema  
**Answer: C**

45**.** How do you define a composite key using Fluent API?  
A) UseKey()  
B) HasMultipleKeys()  
C) HasKey(x => new { x.A, x.B })  
D) SetKey()  
**Answer: C**

46. Which method is used to execute raw SQL in EF Core?  
A) ExecuteQuery()  
B) RawQuery()  
C) FromSqlRaw()  
D) SQL()  
**Answer: C**

47. What does context.Entry(entity).State = EntityState.Modified do?  
A) Deletes record  
B) Marks record for update  
C) Refreshes entity  
D) Adds new record  
**Answer: B**

48**.** What is the purpose of navigation properties?  
A) Store HTML  
B) Hold SQL Queries  
C) Define relationships  
D) Validate input  
**Answer: C**

49**.** Which EF Core method creates the database if it does not exist?  
A) CreateIfNotExists()  
B) EnsureCreated()  
C) TryCreate()  
D) InitDatabase()  
**Answer: B**

50. Can EF Core work with NoSQL databases like MongoDB directly?  
A) Yes  
B) No  
C) Only with plugins  
D) Only with migrations  
**Answer: B**

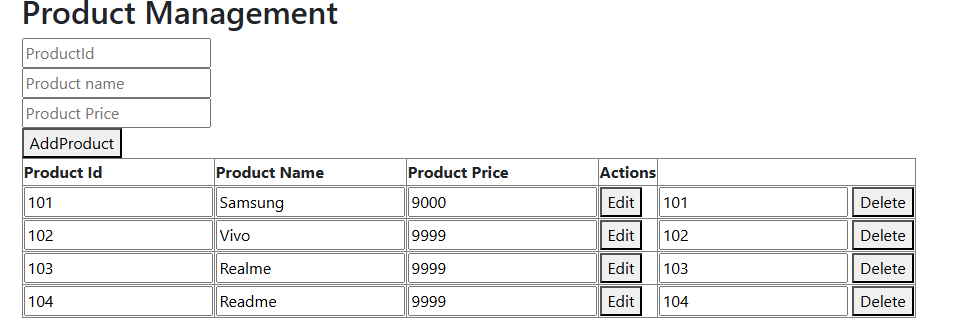
✍️ **Fill in the Blanks**

1. EF Core is a/an \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ tool.  
   **Answer: ORM**
2. In EF Core, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is used to represent database tables.  
   **Answer: DbSet**
3. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ class represents the session with the database.  
   **Answer: DbContext**
4. Code-First approach starts with \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: C# classes / models**
5. The method to apply migrations to DB is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: dotnet ef database update**
6. LINQ stands for \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.  
   **Answer: Language Integrated Query**
7. Database-First uses \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ to reverse engineer models.  
   **Answer: Scaffold-DbContext**
8. Migrations track \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ changes.  
   **Answer: schema**
9. To fetch all products, we use \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ method.  
   **Answer: ToList()**
10. EF Core is a part of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ namespace.  
    **Answer: Microsoft.EntityFrameworkCore**

✅  **True or False**

1. EF Core supports only SQL Server.  
   **False**
2. DbContext must inherit from a base class in EF Core.  
   **True**
3. Code-First is the only approach in EF Core.  
   **False**
4. LINQ queries are supported in EF Core.  
   **True**
5. DbSet represents a database column.  
   **False**
6. Migrations are optional in EF Core.  
   **True**
7. Database-First uses scaffolding to create models.  
   **True**
8. SaveChanges() method is used to persist changes.  
   **True**
9. EF Core is compatible with .NET Framework only.  
   **False**
10. The default behavior of EF Core is lazy loading.  
    **False**

**CURD Operations Using EF Core**

****

Create table Products(Pid int primary key,Pname varchar(50),Price deciml(18,0))

open visual studio 2022->

create new project->asp.net webapplication(model-view-controller)->next->

project name:MvcCoreCurdOperations

next->.dot net core 3.1->create

click on tools->install nuget manager->manage nuget packages for solution

click on-setting icon

click on-add icon

name:package source1

source:https://packages.nuget.org/api/v2

->update

click on-add icon

name:package source2

source:https://api.nuget.org/v3/index.json

->update

->ok

click on Browse:

Microsoft.EntityFrameworkCore->select project->version:5.0.0->install->ok->I Accept

Microsoft.EntityFrameworkDesign->select project->version:5.0.0->install->ok->I Accept

Microsoft.EntityFrameworkSqlServer->select project->version:5.0.0->install->ok->I Accept

Microsoft.EntityFrameworkTools->select project->version:5.0.0->install->ok->I Accept

Appsettings.json

"ConnectionStrings": {

"con": "server=localhost;Initial Catalog=mygominds;Integrated Security=SSPI;TrustServerCertificate=True;"

}

Right click on Model->add->class->name:Product->add

using System.ComponentModel.DataAnnotations;

namespace CoreEFCurd.Models

{

public class Product

{

[Key]

public int Pid { get; set; }

[Required]

public string Pname { get; set; }

[Required]

public decimal Price { get; set; }

}

}

right click on Model->add class->name:ProductDbContext->add

using Microsoft.EntityFrameworkCore;

using System.Diagnostics.Contracts;

namespace CoreEFCurd.Models

{

public class ProductDbContext:DbContext

{

public ProductDbContext(DbContextOptions<ProductDbContext> options) : base(options)

{ }

public DbSet<Product> Products { get; set; }

}

}

using CoreEFCurd.Models;

using Microsoft.EntityFrameworkCore;

Write the bellow line in ConfigureService()

services.AddDbContext<ProductDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("con")));

right click on controller->add controller->mvc controller empty->

using CoreEFCurd.Models;

using Microsoft.AspNetCore.Mvc;

using System.Linq;

namespace CoreEFCurd.Controllers

{

public class ProductController : Controller

{

private readonly ProductDbContext \_context;

public ProductController(ProductDbContext context)

{

\_context = context;

}

public IActionResult Index()

{

var products = \_context.Products.ToList() ;

return View(products);

}

[HttpPost]

public IActionResult Create(Product product)

{

if (ModelState.IsValid)

{

\_context.Products.Add(product);

\_context.SaveChanges();

}

return RedirectToAction("Index");

}

[HttpPost]

public IActionResult Update(Product product)

{

if (ModelState.IsValid)

{

\_context.Products.Update(product);

\_context.SaveChanges();

}

return RedirectToAction("Index");

}

[HttpPost]

public IActionResult Delete(int Pid)

{

var product = \_context.Products.Find(Pid);

if (product != null)

{

\_context.Products.Remove(product);

\_context.SaveChanges();

}

return RedirectToAction("Index");

}

}

}

Right click on -Index()in [HttpPost]->add view->razor view empty->

Name:Index->add

@model List<Product>

<h2>Product Management</h2>

<form method="post" action="/Product/Create">

<input type="number" name="Pid" placeholder="ProductId" /> <br />

<input type="text" name="Pname" placeholder="Product name" /> <br />

<input type="number" name="Price" placeholder="Product Price" /><br />

<button type="submit">AddProduct</button>

</form>

<table border="1">

<thead>

<tr>

<th>Product Id</th>

<th>Product Name</th>

<th>Product Price</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

@if(Model!=null &&Model.Count>0)

{

@foreach(var product in Model)

{

<tr>

<form method="post" action="/Product/Update">

<td>

<input type="number" name="Pid" value="@product.Pid" readonly />

</td>

<td>

<input type="text" name="Pname" value="@product.Pname" />

</td>

<td>

<input type="number" name="Price" value="@product.Price" />

</td>

<td>

<button type="submit">Edit</button>

</td>

</form>

<td>

<form method="post" action="/Product/Delete">

<input type="number" name="Pid" value="@product.Pid" readonly />

<button type="submit">Delete</button>

</form>

</td>

</tr>

}

}

else

{

<tr>

<td colspan="4"> No Products Available

</td>

</tr>

}

</tbody>

</table>

run the application

**✅ MCQs Based on ASP.NET Core MVC CRUD Project**

1. What does the [Key] attribute in the Product class signify?

A) It defines a method

B) It marks the primary key

C) It is used to encrypt data

D) It validates the model

Answer: B) It marks the primary key

2. What is the purpose of DbSet<Product> in ProductDbContext?

A) To configure database connection

B) To store controller logic

C) To represent a collection of Product entities

D) To create a SQL table

Answer: C) To represent a collection of Product entities

3. Where is the connection string defined?

A) In Startup.cs

B) In appsettings.json

C) In Program.cs

D) In web.config

Answer: B) In appsettings.json

4. Which NuGet package allows EF Core to work with SQL Server?

A) Microsoft.EntityFrameworkCore.Tools

B) Microsoft.EntityFrameworkCore.SqlServer

C) Microsoft.EntityFrameworkDesign

D) Microsoft.AspNetCore.Mvc

Answer: B) Microsoft.EntityFrameworkCore.SqlServer

5. Which method is used to fetch all records from the database?

A) ToList()

B) GetAll()

C) Find()

D) All()

Answer: A) ToList()

6. What is the return type of the Index() method in ProductController?

A) ViewResult

B) JsonResult

C) IActionResult

D) PartialViewResult

Answer: C) IActionResult

7. How are POST requests handled in the controller?

A) Using [HttpGet]

B) Using [HttpPost]

C) Using GET

D) Using HttpPut

Answer: B) Using [HttpPost]

8. Which tag is used in Razor to iterate over a list?

A) @loop

B) @for

C) @foreach

D) @repeat

Answer: C) @foreach

9. What is UseSqlServer used for in Startup.cs?

A) Creating a SQL server

B) Connecting to SQL Server

C) Validating models

D) Compiling the project

Answer: B) Connecting to SQL Server

10. What happens if a product with a given Pid is not found during deletion?

A) Error is thrown

B) A new product is created

C) No action is taken

D) Page crashes

Answer: C) No action is taken

11. What does services.AddDbContext<ProductDbContext>() do?

A) Registers EF Core services

B) Starts a new thread

C) Adds logging

D) Compiles Razor views

Answer: A) Registers EF Core services

12. Which version of EF Core is installed in the project?

A) 3.1

B) 6.0

C) 5.0

D) 2.2

Answer: C) 5.0

13. What HTTP verb is used for updating a product?

A) GET

B) POST

C) PUT

D) PATCH

Answer: B) POST

14. What does ModelState.IsValid check?

A) Database connection

B) Route validation

C) Input validation

D) Controller naming

Answer: C) Input validation

15. What is the name of the Razor view file for Index action?

A) Main.cshtml

B) Home.cshtml

C) Index.cshtml

D) Product.cshtml

Answer: C) Index.cshtml

16. What keyword is used to inject ProductDbContext in the controller?

A) implements

B) base

C) this

D) constructor injection

Answer: D) constructor injection

17. Which directive sets the model type in a Razor view?

A) @model

B) @inherits

C) @using

D) @type

Answer: A) @model

18. What placeholder is used for connection string name in UseSqlServer()?

A) "connection"

B) "ConnectionString"

C) "con"

D) "conn"

Answer: C) "con"

19. What is the namespace of the Product class?

A) Microsoft.EntityFrameworkCore

B) CoreEFCurd.Controllers

C) CoreEFCurd.Models

D) ProductDb.Models

Answer: C) CoreEFCurd.Models

20. What HTML element is used to submit the form data?

A) <submit>

B) <a>

C) <button>

D) <form>

Answer: C) <button>

21. What type is Price in the Product model?

A) int

B) float

C) string

D) decimal

Answer: D) decimal

22. Which action returns the product list to the view?

A) Create

B) Index

C) Update

D) Delete

Answer: B) Index

23. What does Add(product) do in EF Core?

A) Updates a record

B) Deletes a record

C) Inserts a new record

D) Validates a record

Answer: C) Inserts a new record

24. What is the purpose of appsettings.json?

A) View settings

B) Code editor configuration

C) Storing configuration like connection strings

D) Styling

Answer: C) Storing configuration like connection strings

25. What database is used in the connection string?

A) testdb

B) mygominds

C) mydata

D) defaultdb

Answer: B) mygominds

26. What does Update(product) do in EF Core?

A) Deletes the product

B) Validates the product

C) Modifies the existing product

D) Adds a new product

Answer: C) Modifies the existing product

27. What is the default port used by Visual Studio for local web hosting?

A) 8080

B) 1433

C) Random high port

D) 80

Answer: C) Random high port

28. Where do you register the ProductDbContext service?

A) Configure()

B) appsettings.json

C) ConfigureServices()

D) Program.cs

Answer: C) ConfigureServices()

29. What is the role of Integrated Security=SSPI in the connection string?

A) Enables password authentication

B) Uses Windows Authentication

C) Requires admin privileges

D) Disables SSL

Answer: B) Uses Windows Authentication

30. What is Razor in ASP.NET Core?

A) CSS engine

B) A server

C) A markup syntax for embedding C# in HTML

D) An ORM

Answer: C) A markup syntax for embedding C# in HTML

31. What method is used to delete a record in EF Core?

A) Delete()

B) Remove()

C) Drop()

D) Clear()

Answer: B) Remove()

32. What does Find(Pid) do in EF Core?

A) Deletes a record

B) Finds the entity by primary key

C) Adds a new product

D) Connects to SQL

Answer: B) Finds the entity by primary key

33. Which property in the Product model ensures that the Pname is not empty?

A) [Key]

B) [NotNull]

C) [Required]

D) [Validate]

Answer: C) [Required]

34. What method is used to save changes to the database?

A) Save()

B) SubmitChanges()

C) SaveChanges()

D) Apply()

Answer: C) SaveChanges()

35. What will happen if you forget to call SaveChanges()?

A) App crashes

B) Changes are not persisted

C) Database gets deleted

D) App becomes slow

Answer: B) Changes are not persisted

36. Where should UseSqlServer() be placed?

A) In the constructor of the controller

B) In ConfigureServices() method

C) In Main() method

D) In Index() method

Answer: B) In ConfigureServices() method

37. What kind of pattern is used in MVC?

A) Design pattern

B) Data structure

C) Middleware pattern

D) Algorithm

Answer: A) Design pattern

38. What keyword is used to inherit from DbContext?

A) : DbSet

B) : Controller

C) : DbContext

D) : ContextBase

Answer: C) : DbContext

39. Which method is used to return a view from a controller?

A) Show()

B) Redirect()

C) View()

D) Get()

Answer: C) View()

40. What tag is used to create an HTML form in Razor view?

A) @form

B) <form>

C) <razor-form>

D) <mvc-form>

Answer: B) <form>

41. Why do we use readonly for \_context in the controller?

A) To prevent deletion

B) For better readability

C) To avoid reassignment

D) To protect data

Answer: C) To avoid reassignment

42. What directive must be present to use model properties in the view?

A) @using

B) @model

C) @inject

D) @add

Answer: B) @model

43. How is dependency injection used in this app?

A) Through config file

B) Through attribute

C) Through constructor

D) Not used

Answer: C) Through constructor

44. What is the type of data passed to the Index view?

A) List<string>

B) Product

C) string

D) List<Product>

Answer: D) List<Product>

45. What does [HttpPost] do?

A) Handles GET requests

B) Handles POST requests

C) Hides the method

D) Compiles the view

Answer: B) Handles POST requests

46. Which EF Core tool is used for command-line migrations and scaffolding?

A) Microsoft.EntityFrameworkCore.Tools

B) Microsoft.AspNetCore.Mvc

C) Microsoft.Extensions.Logging

D) Newtonsoft.Json

Answer: A) Microsoft.EntityFrameworkCore.Tools

47. What symbol is used in Razor syntax to embed C# code?

A) #

B) $

C) %

D) @

Answer: D) @

48. Which input type is used to accept price in the view?

A) text

B) date

C) number

D) currency

Answer: C) number

49. What does the Create action do in the controller?

A) Updates product

B) Deletes product

C) Adds new product

D) Shows product list

Answer: C) Adds new product

50. Why is Pid marked as readonly in HTML form?

A) It should not be modified

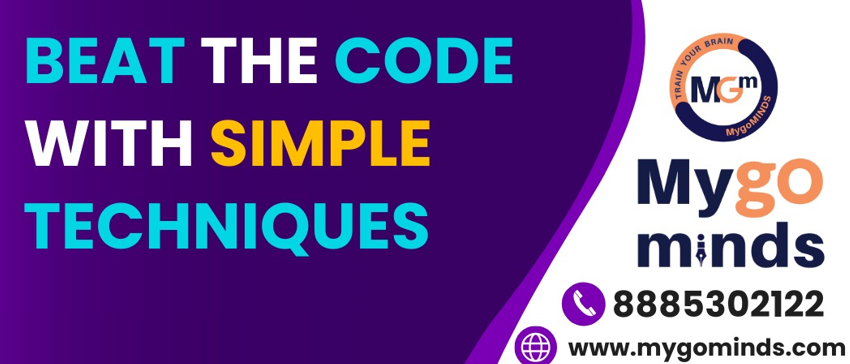
B) It's required

C) It's hidden

D) It needs to be editable

Answer: A) It should not be modified



****

**Chapter-8**

**Web API With ASP.NET Core**

**📘 What is Web API in ASP.NET Core?**

ASP.NET Core **Web API** is a framework for building **RESTful HTTP services** that return data (mostly JSON) to clients such as web apps, mobile apps, or other servers.

**🔑 Key Features:**

* Lightweight and high-performance
* Uses ControllerBase class (not full MVC Controller)
* Built-in support for **routing**, **model binding**, and **serialization**
* Can be tested easily with tools like **Postman** and **Swagger**
* Returns data (JSON/XML), not HTML views

**🔧 Basic Example:**

[ApiController]

[Route("api/[controller]")]

public class ProductsController : ControllerBase

{

[HttpGet]

public IActionResult Get() => Ok(new[] { "Product1", "Product2" });

}

This example creates a REST API endpoint at /api/products.

📘 **Difference Between Web API and Web API Core**

| **Feature/Aspect** | **ASP.NET Web API (Classic)** | **ASP.NET Core Web API** |
| --- | --- | --- |
| **Platform** | Windows-only | Cross-platform (Windows, Linux, macOS) |
| **Framework** | .NET Framework | .NET Core / .NET 5/6/7/8+ |
| **Performance** | Lower | Much faster and optimized |
| **Modular Architecture** | Monolithic | Modular (only load what you need) |
| **Hosting Model** | IIS only | IIS, Kestrel, Self-hosting, Nginx, Apache |
| **Dependency Injection** | Requires third-party tools | Built-in DI support |
| **Configuration** | Web.config (XML) | appsettings.json (JSON) |
| **Development** | Tightly coupled with Windows | Modern, flexible, open-source |
| **MVC & Web API** | Separate frameworks | Unified under ASP.NET Core MVC |

**🔧 Summary:**

* **ASP.NET Web API (Classic)**: Part of full .NET Framework, Windows-only, older architecture.
* **ASP.NET Core Web API**: Cross-platform, lightweight, high-performance, unified model for building modern web services.

**🔑 1. Introduction to REST and HTTP Verbs**

**REST** (Representational State Transfer) is an architecture style for designing networked applications using HTTP.

| **HTTP Verb** | **Description** |
| --- | --- |
| GET | Retrieve data |
| POST | Create data |
| PUT | Update data |
| DELETE | Remove data |

**🔧 Example:**

GET /api/products // Get all products

POST /api/products // Add a new product

PUT /api/products/1 // Update product with ID 1

DELETE /api/products/1 // Delete product with ID 1

**🔑 2. Creating Web API Controllers**

API controllers handle incoming HTTP requests and return data in JSON or XML.

[ApiController]

[Route("api/[controller]")]

public class ProductsController : ControllerBase

{

[HttpGet]

public IActionResult GetAll() => Ok(new[] { "Apple", "Banana" });

}

**🔑 3. Attribute Routing**

You define custom routes using attributes.

[HttpGet("GetById/{id}")]

public IActionResult GetById(int id) => Ok($"Product {id}");

**🔑 4. Consuming Web API using Postman / Swagger**

* **Postman**: Tool to test API endpoints manually.
* **Swagger**: UI generated from API using Swashbuckle package.

Install-Package Swashbuckle.AspNetCore

Add in Startup.cs:

services.AddSwaggerGen();

app.UseSwagger();

app.UseSwaggerUI();

**🔑 5. Input Validation and ModelState**

Validate request body using data annotations and ModelState.

public class Product

{

[Required]

public string Name { get; set; }

}

[HttpPost]

public IActionResult Create(Product p)

{

if (!ModelState.IsValid)

return BadRequest(ModelState);

return Ok(p);

}

**🔑 6. Returning JSON and Status Codes**

ASP.NET Core Web API returns JSON by default.

return Ok(product); // 200 OK

return NotFound(); // 404

return BadRequest(); // 400

return StatusCode(500); // 500 Internal Server Error

✅  **Multiple Choice Questions (MCQs)**

1.What does the HTTP GET verb do?

A) Creates data  
B) Retrieves data  
C) Updates data  
D) Deletes data

**Answer: B) Retrieves data**

2.What is the purpose of [ApiController]?

A) Adds routing support  
B) Enables tag helpers  
C) Enables automatic model validation  
D) Enables Entity Framework

**Answer: C) Enables automatic model validation**

3.What does the [HttpPost] attribute do?

A) Handles HTTP GET  
B) Handles HTTP POST  
C) Handles HTTP DELETE  
D) Handles HTTP PUT

**Answer: B) Handles HTTP POST**

4.What format does Web API return by default?

A) XML  
B) CSV  
C) JSON  
D) HTML

**Answer: C) JSON**

5.What tool helps to test APIs visually?

A) Git  
B) Swagger  
C) LINQPad  
D) Visual Studio Code

**Answer: B) Swagger**

6.What method returns HTTP 404?

A) Ok()  
B) NotFound()  
C) BadRequest()  
D) Created()

**Answer: B) NotFound()**

7.Which package is used to enable Swagger?

A) Microsoft.OpenApi  
B) Swashbuckle.AspNetCore  
C) Swagger.UI  
D) Newtonsoft.Json

**Answer: B) Swashbuckle.AspNetCore**

8.What attribute is used for model validation?

A) [Valid]  
B) [Required]  
C) [Check]  
D) [Assert]

**Answer: B) [Required]**

9.What method is used to create a new resource?

A) GET  
B) POST  
C) PUT  
D) DELETE

**Answer: B) POST**

10.Which controller base class is used for Web API?

A) Controller  
B) ControllerPage  
C) ControllerBase  
D) ApiController

**Answer: C) ControllerBase**

**11. Which status code indicates success with data returned?**A) 200  
B) 400  
C) 500  
D) 404  
**Answer: A) 200**

12.Which HTTP verb is used to delete a resource?  
A) GET  
B) POST  
C) PUT  
D) DELETE  
**Answer: D) DELETE**

13.Which method is used to return a 201 Created response?  
A) Ok()  
B) Created()  
C) Accepted()  
D) NoContent()  
**Answer: B) Created()**

14.What is the use of [FromBody] attribute?  
A) Binds route data  
B) Binds query string  
C) Binds data from request body  
D) Binds header values  
**Answer: C) Binds data from request body**

15.Where are the API routes typically configured in ASP.NET Core?  
A) appsettings.json  
B) Program.cs  
C) launchSettings.json  
D) controller.cs  
**Answer: B) Program.cs**

16.Which HTTP verb is idempotent?  
A) POST  
B) PUT  
C) PATCH  
D) None  
**Answer: B) PUT**

17.What is the purpose of AddControllers() in Program.cs?  
A) Enable MVC views  
B) Enable Razor Pages  
C) Register API controllers  
D) Enable Blazor  
**Answer: C) Register API controllers**

18.Which return type allows sending custom HTTP status codes and content?  
A) string  
B) int  
C) IActionResult  
D) void  
**Answer: C) IActionResult**

19.What is Model Binding in Web API?  
A) Binds data to Razor pages  
B) Binds input data to action parameters  
C) Binds views to the controller  
D) Binds middleware to endpoints  
**Answer: B) Binds input data to action parameters**

20.Which feature enables OpenAPI documentation in Web API?  
A) Entity Framework  
B) SignalR  
C) Swagger  
D) Middleware  
**Answer: C) Swagger**

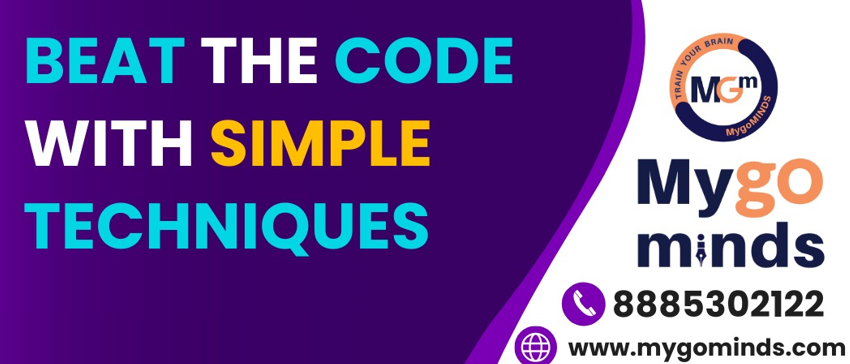
✍️ **Fill in the Blanks**

1. The HTTP verb used to update a resource is \_\_\_\_\_\_\_\_\_\_.  
   **Answer: PUT**
2. The default response format in Web API is \_\_\_\_\_\_\_\_\_\_.  
   **Answer: JSON**
3. You annotate a Web API class with \_\_\_\_\_\_\_\_\_\_ to enable features like automatic model validation.  
   **Answer: [ApiController]**
4. The method NotFound() returns status code \_\_\_\_\_\_\_\_\_\_.  
   **Answer: 404**
5. Swagger is generated using the \_\_\_\_\_\_\_\_\_\_ package.  
   **Answer: Swashbuckle.AspNetCore**
6. Input model validation can be done using \_\_\_\_\_\_\_\_\_\_ annotations.  
   **Answer: Data**
7. A Web API controller inherits from \_\_\_\_\_\_\_\_\_\_.  
   **Answer: ControllerBase**
8. You use \_\_\_\_\_\_\_\_\_\_ method to return HTTP 200 with data.  
   **Answer: Ok()**
9. The tool used for manual API testing is \_\_\_\_\_\_\_\_\_\_.  
   **Answer: Postman**
10. \_\_\_\_\_\_\_\_\_\_ is the method that applies a status code manually.  
    **Answer: StatusCode()**

✅  **True or False**

1. Web API returns HTML by default.  
   **False**
2. [HttpPost] is used to handle POST requests.  
   **True**
3. [ApiController] automatically validates models.  
   **True**
4. You must use Entity Framework with Web API.  
   **False**
5. You can use Postman to test Web API endpoints.  
   **True**
6. Ok() method returns 404.  
   **False**
7. Swagger can be used to document APIs.  
   **True**
8. [Route] attribute defines the endpoint URL.  
   **True**
9. Web API controllers must inherit from Controller.  
   **False**
10. Model validation is not required in Web API.  
    **False**



****

**Chapter-9**

**Authentication & Authorization**

**📘 Authentication & Authorization in ASP.NET Core**

### 🔑 Key Concepts:

* **Authentication**: Verifies the identity of the user.  
  “Who are you?”  
  🔹 Example: Login using username/password, external providers (Google, Facebook)
* **Authorization**: Determines access levels or permissions.  
  “What are you allowed to do?”  
  🔹 Example: Admins can delete data; regular users cannot.

### 📝 Brief Explanation:

| **Feature** | **Description** |
| --- | --- |
| **Authentication Schemes** | Determines how the app authenticates users (Cookies, JWT, OAuth) |
| **Identity** | ASP.NET Core framework for managing users, roles, logins, etc. |
| **Policies** | Allow complex authorization logic using custom rules |
| **Claims** | Key-value pairs about the user (e.g., email, role, id) |
| **Roles** | Group users with similar permissions (e.g., Admin, User) |
| **Authorization Attributes** | [Authorize], [AllowAnonymous], [Authorize(Roles="Admin")] |
| 🔧 Code Examples:✅ Setup Authentication in Program.cs (for Cookie) builder.Services.AddAuthentication("MyCookieAuth")  .AddCookie("MyCookieAuth", options =>  {  options.LoginPath = "/Account/Login";  });  ✅ Enable Authorization |  |

### builder.Services.AddAuthorization();

### app.UseAuthentication();

### app.UseAuthorization();

### ✅ Use in Controller

### [Authorize]

### public IActionResult SecurePage() => View();

### [Authorize(Roles = "Admin")]

### public IActionResult AdminOnly() => View();

### [AllowAnonymous]

### public IActionResult Login() => View();

**📘 1. ASP.NET Core Identity Basics**

ASP.NET Core Identity is a membership system for user authentication and management, including:

* Register, Login, Logout
* Password hashing
* Role management
* Two-factor auth, email confirm

✅ **Setup in Program.cs or Startup.cs**:

services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

services.AddIdentity<IdentityUser, IdentityRole>()

.AddEntityFrameworkStores<AppDbContext>();

✅ **Configure Identity Middleware**:

app.UseAuthentication();

app.UseAuthorization();

**📘 2. Cookie Authentication**

Used for traditional web apps to store user login state in browser cookies.

✅ **Add in Program.cs:**

services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)

.AddCookie(options =>

{

options.LoginPath = "/Account/Login";

options.AccessDeniedPath = "/Account/AccessDenied";

});

✅ **Use Authentication Middleware**:

app.UseAuthentication();

**📘 3. Role-based Authorization**

Allows access control based on roles like **Admin**, **User**, etc.

✅ **Assign Roles:**

await userManager.AddToRoleAsync(user, "Admin");

✅ **Authorize Role in Controller:**

[Authorize(Roles = "Admin")]

public IActionResult AdminOnly()

{

return View();

}

✅ **Check in Razor:**

@if (User.IsInRole("Admin"))

{

<p>You are an Admin.</p>

}

**📘 4. JWT Authentication (Token-based)**

JWT (JSON Web Token) is used for stateless authentication, especially in **APIs**.

✅ **Install NuGet package:**

Microsoft.AspNetCore.Authentication.JwtBearer

✅ **Configure JWT in Program.cs:**

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidIssuer = "yourdomain.com",

ValidAudience = "yourdomain.com",

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("YourSecretKey"))

};

});

✅ **Protect API Endpoint:**

[Authorize]

[HttpGet]

public IActionResult GetData() => Ok("Protected Data");

**🔧 Summary Table**

| **Feature** | **Purpose** | **Used In** |
| --- | --- | --- |
| ASP.NET Core Identity | Full user management system | Web apps |
| Cookie Authentication | Session-based authentication | MVC, Razor Pages |
| Role-based Authorization | Access control using roles | All types of apps |
| JWT Authentication | Token-based, stateless authentication | Web APIs, mobile apps |

✅  **Multiple Choice Questions (MCQs)**

**1. What is the purpose of AddIdentity<>() in ASP.NET Core?**  
A) Adds middleware for file uploading  
B) Adds user authentication and management  
C) Enables Razor syntax  
D) Registers MVC services  
**Answer: B**

**2. Which authentication mechanism is commonly used for Web APIs?**  
A) Cookie Authentication  
B) Windows Authentication  
C) JWT Authentication  
D) OAuth2 only  
**Answer: C**

**3. Which class is used to define identity roles in ASP.NET Core?**  
A) IdentityManager  
B) IdentityRole  
C) RoleIdentity  
D) UserManager  
**Answer: B**

**4. What middleware must be called to enable user login support?**  
A) app.UseMvc()  
B) app.UseAuthentication()  
C) app.UseEndpoints()  
D) app.UseStaticFiles()  
**Answer: B**

**5. Which service helps with issuing secure tokens for JWT?**  
A) TokenManager  
B) SecurityKeyProvider  
C) SymmetricSecurityKey  
D) HashAlgorithm  
**Answer: C**

**6. Which file typically contains JWT secret key and issuer info?**  
A) app.config  
B) appsettings.json  
C) web.xml  
D) startup.js  
**Answer: B**

**7. What is required in a controller to restrict it to Admin role only?**  
A) [Authorize()]  
B) [AllowAnonymous]  
C) [Authorize(Roles = "Admin")]  
D) [AdminOnly()]  
**Answer: C**

**8. What does User.Identity.IsAuthenticated return?**  
A) The user’s password  
B) The user's claims  
C) True if the user is logged in  
D) Always false  
**Answer: C**

**9. JWT is short for:**  
A) Java Web Token  
B) JSON Web Token  
C) JavaScript Web Token  
D) Java WCF Token  
**Answer: B**

**10. Which service is used to manage user accounts in Identity?**  
A) UserAccountService  
B) UserIdentityManager  
C) UserManager<T>  
D) LoginService  
**Answer: C**

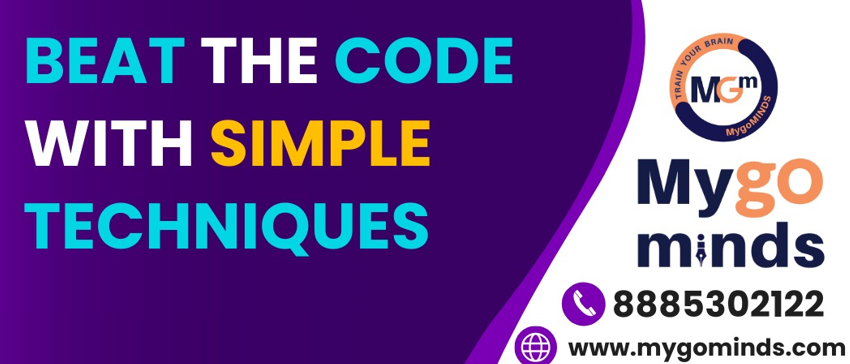
✍️ **Fill in the Blanks**

1. The method services.AddAuthentication() is used to configure \_\_\_\_\_\_\_\_\_\_ schemes.  
   **Answer: authentication**
2. JWTs are \_\_\_\_\_\_\_\_\_\_, meaning the server does not store them.  
   **Answer: stateless**
3. [Authorize(Roles = "Admin")] restricts access to users in the \_\_\_\_\_\_\_\_\_\_ role.  
   **Answer: Admin**
4. ASP.NET Core Identity uses \_\_\_\_\_\_\_\_\_\_ hashing to secure passwords.  
   **Answer: password**
5. JWT tokens are typically passed in the \_\_\_\_\_\_\_\_\_\_ header of HTTP requests.  
   **Answer: Authorization**
6. The class UserManager<T> is used to manage \_\_\_\_\_\_\_\_\_\_-related operations.  
   **Answer: user**
7. Role-based authorization can be implemented using the \_\_\_\_\_\_\_\_\_\_ attribute.  
   **Answer: Authorize**
8. Cookies are stored on the \_\_\_\_\_\_\_\_\_\_ side.  
   **Answer: client**
9. ASP.NET Core Identity is built on top of the \_\_\_\_\_\_\_\_\_\_ Core framework.  
   **Answer: ASP.NET**
10. JWT tokens include a \_\_\_\_\_\_\_\_\_\_ signature to prevent tampering.  
    **Answer: cryptographic**

✅  **True or False**

1. JWT authentication is suitable for stateless APIs.  
   **Answer: True**
2. The AddIdentity method automatically adds roles and user management.  
   **Answer: True**
3. Cookie authentication is mostly used in single-page APIs.  
   **Answer: False**
4. ASP.NET Core Identity supports email confirmation out of the box.  
   **Answer: True**
5. [AllowAnonymous] is used to restrict access to only authenticated users.  
   **Answer: False**
6. JWT tokens can be decoded without a secret key.  
   **Answer: True**
7. ASP.NET Core can use multiple authentication schemes.  
   **Answer: True**
8. Authorize attribute works without registering authentication middleware.  
   **Answer: False**
9. Identity roles can be stored in the database.  
   **Answer: True**
10. You can manually create and assign roles using RoleManager.  
    **Answer: True**



****

**Chapter-10**

**Advanced Topics**

## 🔄 Asynchronous Programming (async/await)

📘 **Explanation**:  
Async programming improves application responsiveness by freeing up threads while awaiting I/O-bound operations.

🔧 **Example**:

public async Task<IActionResult> GetDataAsync()

{

var data = await \_service.GetDataFromDbAsync();

return Ok(data);

}

🔑 **Use**: Improves scalability and performance, especially in I/O-heavy applications like APIs.

## 🪵 Logging with ILogger

📘 **Explanation**:  
ILogger is the built-in logging abstraction in ASP.NET Core. Supports different providers like Console, Debug, and File.

🔧 **Example**:

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

public IActionResult Index()

{

\_logger.LogInformation("Index page loaded.");

return View();

}

🔑 **Use**: Logs runtime information for debugging and monitoring.

## 🧠 Caching (In-Memory & Distributed)

📘 **Explanation**:  
Caching stores frequently accessed data in memory or a distributed store to improve performance.

🔧 **In-Memory Example**:

services.AddMemoryCache();

public class MyService

{

private readonly IMemoryCache \_cache;

public MyService(IMemoryCache cache)

{

\_cache = cache;

}

public string GetCachedData()

{

return \_cache.GetOrCreate("key", entry => "cached value");

}

}

🔧 **Distributed Example (e.g., Redis)**:

services.AddStackExchangeRedisCache(options => {

options.Configuration = "localhost:6379";

});

services.AddDistributedMemoryCache();

🔑 **Use**: Reduces load on database, increases speed.

## 🚨 Global Error Handling

📘 **Explanation**:  
Handles unhandled exceptions globally using middleware.

🔧 **Example** (Configure in Startup.cs):

app.UseExceptionHandler("/Home/Error");

app.Use(async (context, next) =>

{

try

{

await next();

}

catch (Exception ex)

{

// Log and handle error

context.Response.StatusCode = 500;

}

});

🔑 **Use**: Centralized error handling for better control and logging.

## ⚙️ Configuration and appsettings.json

📘 **Explanation**:  
ASP.NET Core reads configuration from appsettings.json, environment variables, or custom providers.

🔧 **Example (appsettings.json)**:

{

"AppSettings": {

"SiteTitle": "My Awesome App"

}

}

🔧 **Access in Code**:

public class HomeController : Controller

{

private readonly IConfiguration \_config;

public HomeController(IConfiguration config)

{

\_config = config;

}

public IActionResult Index()

{

var title = \_config["AppSettings:SiteTitle"];

ViewBag.Title = title;

return View();

}

}

🔑 **Use**: Externalizes config for flexibility and environment support.

✅  **Multiple Choice Questions (MCQs)**

**1. What is the main benefit of using async/await in ASP.NET Core?**  
A) Reduces CPU usage  
B) Improves database speed  
C) Increases scalability by freeing threads  
D) Avoids dependency injection  
**Answer: C**

**2. What does the ILogger<T> interface help you do?**  
A) Connect to the database  
B) Log application messages  
C) Perform data caching  
D) Handle authentication  
**Answer: B**

**3. Which caching mechanism is suitable for multi-server environments?**  
A) MemoryCache  
B) FileCache  
C) Distributed Cache  
D) Static Cache  
**Answer: C**

**4. What is the role of UseExceptionHandler("/Home/Error")?**  
A) To catch syntax errors  
B) To globally handle unhandled exceptions  
C) To configure Razor views  
D) To define custom routes  
**Answer: B**

**5. Where are configuration values typically stored in ASP.NET Core?**  
A) web.config  
B) startup.json  
C) appsettings.json  
D) config.xml  
**Answer: C**

**6. What namespace contains IMemoryCache?**  
A) Microsoft.Extensions.Hosting  
B) Microsoft.Extensions.Caching.Memory  
C) Microsoft.AspNetCore.Caching  
D) Microsoft.Data.SqlClient  
**Answer: B**

**7. What happens when an awaited async method is executing?**  
A) The thread is blocked  
B) The server sleeps  
C) The thread is released for other work  
D) The app stops  
**Answer: C**

**8. Which of these is not a valid log level in ASP.NET Core?**  
A) Information  
B) Debug  
C) Fatal  
D) Critical  
**Answer: C**

**9. What does IConfiguration help you access?**  
A) Database schema  
B) Middleware pipeline  
C) Application settings  
D) User roles  
**Answer: C**

**10. Which method is used to store items in memory using IMemoryCache?**  
A) Add()  
B) Save()  
C) GetOrCreate()  
D) Cache()  
**Answer: C**

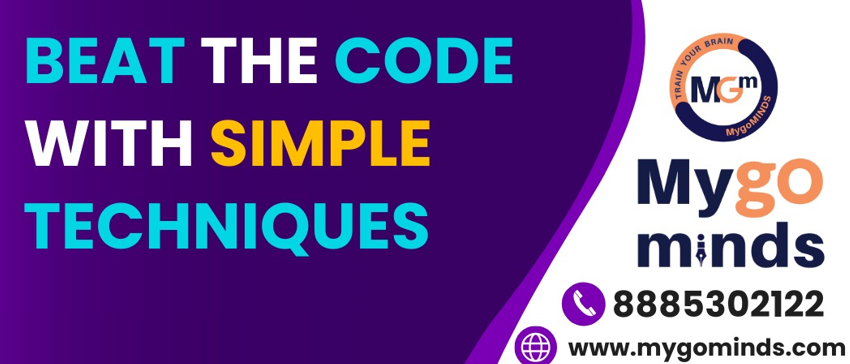
✍️ **Fill in the Blanks**

1. The async keyword allows a method to be run \_\_\_\_\_\_\_\_\_\_.  
   **Answer: asynchronously**
2. The method used to write logs in ASP.NET Core is logger.\_\_\_\_\_\_\_\_\_\_.  
   **Answer: LogInformation**
3. The appsettings.json file stores \_\_\_\_\_\_\_\_\_\_ values for an application.  
   **Answer: configuration**
4. To use Redis, you add a distributed cache using \_\_\_\_\_\_\_\_\_\_.  
   **Answer: AddStackExchangeRedisCache**
5. UseDeveloperExceptionPage() shows detailed error messages during \_\_\_\_\_\_\_\_\_\_.  
   **Answer: development**
6. IMemoryCache is used for \_\_\_\_\_\_\_\_\_\_ caching.  
   **Answer: in-memory**
7. In async/await, methods that return no value use \_\_\_\_\_\_\_\_\_\_ return type.  
   **Answer: Task**
8. ASP.NET Core logging supports multiple \_\_\_\_\_\_\_\_\_\_ (Console, File, etc.).  
   **Answer: providers**
9. The IConfiguration interface is injected via \_\_\_\_\_\_\_\_\_\_ injection.  
   **Answer: dependency**
10. In appsettings.json, configuration values are stored as \_\_\_\_\_\_\_\_\_\_ pairs.  
    **Answer: key-value**

✅  **True or False**

1. Asynchronous programming blocks the thread until the task completes.  
   **Answer: False**
2. You can inject ILogger<T> into controllers for logging.  
   **Answer: True**
3. Memory cache can be shared between servers automatically.  
   **Answer: False**
4. UseExceptionHandler() should be placed before routing middleware.  
   **Answer: True**
5. The appsettings.json file supports hierarchical settings.  
   **Answer: True**
6. You must always manually create a configuration provider in ASP.NET Core.  
   **Answer: False**
7. Async methods can return either Task or Task<T>.  
   **Answer: True**
8. Logging in ASP.NET Core is only possible via third-party libraries.  
   **Answer: False**
9. You can set expiration policies in memory caching.  
   **Answer: True**
10. Exception handling middleware can prevent the application from crashing.  
    **Answer: True**



****

**Chapter-11**

**Frontend Integration**

**📘 Frontend Integration in ASP.NET Core**

**🌐 Consuming APIs with Angular / React / jQuery**

**Explanation**:  
ASP.NET Core Web APIs can be consumed from frontend libraries/frameworks like Angular, React, or jQuery using HTTP clients (HttpClient, fetch, $.ajax).

**Example (Angular)**:

this.http.get("https://localhost:5001/api/products").subscribe(data => {

this.products = data;

});

**Example (jQuery)**:

$.get("https://localhost:5001/api/products", function(data) {

console.log(data);

});

**Use**: Enables decoupling of front-end and back-end with JSON-based communication.

**🆚 Razor Pages vs MVC**

**Explanation**:

| **Feature** | **Razor Pages** | **MVC** |
| --- | --- | --- |
| File Structure | Page-based (.cshtml + code-behind) | Separated (Controller, View, Model) |
| Suitable For | Simple, form-based web apps | Large, complex web applications |
| Routing | Uses folders to define routes | Uses Controllers and route attributes |
| View Model Binding | Direct via page model | Via controller method parameters |

**Use**: Razor Pages are great for smaller apps or pages with minimal logic; MVC is better for enterprise-level apps.

**🗂️ Static Files and Bundling**

**Explanation**:  
Static files (CSS, JS, images) are served from the wwwroot folder. Bundling and minification reduce file size and HTTP requests for performance.

**Enable Static Files**:

app.UseStaticFiles();

**Bundling (using third-party tools)**:

* Use **Gulp**, **Webpack**, or **BundlerMinifier** to create .min.js, .min.css files.

**Use**: Improves load speed and SEO performance.

✅  **Multiple Choice Questions (MCQs)**

**1. Which folder is used to serve static files in ASP.NET Core?**  
A) /content  
B) /appdata  
C) /wwwroot  
D) /static  
**Answer: C**

**2. What is used in Angular to call backend APIs?**  
A) AJAX  
B) HttpClient  
C) FetchData  
D) JsonParser  
**Answer: B**

**3. Razor Pages organize code in which format?**  
A) Controller-based  
B) Page-based  
C) Model-based  
D) Service-based  
**Answer: B**

**4. What is the default HTTP method used by $.get()?**  
A) POST  
B) DELETE  
C) GET  
D) PUT  
**Answer: C**

**5. Which of the following is NOT a valid bundling tool?**  
A) Gulp  
B) Webpack  
C) RazorBundler  
D) BundlerMinifier  
**Answer: C**

**6. Which is more suitable for large enterprise applications?**  
A) Razor Pages  
B) MVC  
C) Web Forms  
D) SignalR  
**Answer: B**

**7. Static files require what middleware to be served?**  
A) UseEndpoints  
B) UseRouting  
C) UseStaticFiles  
D) UseAuthorization  
**Answer: C**

**8. Razor Pages store logic in a:**  
A) Controller class  
B) .js file  
C) Page model class  
D) service.cs  
**Answer: C**

**9. What does bundling help achieve?**  
A) Increases HTTP requests  
B) Makes files larger  
C) Improves load time  
D) Disables caching  
**Answer: C**

**10. jQuery uses which method for HTTP GET calls?**  
A) $.call()  
B) $.get()  
C) $.fetch()  
D) $.connect()  
**Answer: B**

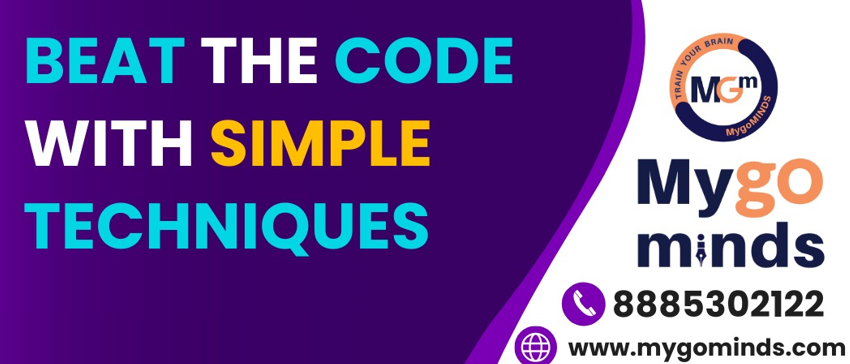
✍️ **Fill in the Blanks**

1. Static files are placed in the \_\_\_\_\_\_\_\_\_\_ folder.  
   **Answer: wwwroot**
2. Angular uses \_\_\_\_\_\_\_\_\_\_ to make HTTP calls.  
   **Answer: HttpClient**
3. MVC separates the application into Model, \_\_\_\_\_\_\_\_\_\_, and Controller.  
   **Answer: View**
4. Razor Pages combine view and logic in a \_\_\_\_\_\_\_\_\_\_ structure.  
   **Answer: page-based**
5. The command to enable static files in Startup.cs is \_\_\_\_\_\_\_\_\_\_.  
   **Answer: app.UseStaticFiles()**
6. jQuery’s shorthand method for HTTP GET is \_\_\_\_\_\_\_\_\_\_.  
   **Answer: $.get()**
7. Bundling and \_\_\_\_\_\_\_\_\_\_ improve client-side performance.  
   **Answer: minification**
8. MVC routing is typically configured in \_\_\_\_\_\_\_\_\_\_.  
   **Answer: Startup.cs**
9. Razor Pages typically end with a \_\_\_\_\_\_\_\_\_\_ extension.  
   **Answer: .cshtml**
10. React uses the \_\_\_\_\_\_\_\_\_\_ function to perform fetch operations.  
    **Answer: fetch**

✅  **True or False**

1. Static files require no special configuration in ASP.NET Core.  
   **Answer: False**
2. Razor Pages follow the MVC pattern.  
   **Answer: False**
3. You can use Angular and React to consume ASP.NET Core Web APIs.  
   **Answer: True**
4. Bundling increases the number of HTTP requests.  
   **Answer: False**
5. MVC is better for complex applications compared to Razor Pages.  
   **Answer: True**
6. jQuery can be used to perform both GET and POST requests.  
   **Answer: True**
7. Razor Pages keep logic and view in separate folders by default.  
   **Answer: False**
8. HttpClient is built into Angular’s core libraries.  
   **Answer: True**
9. You can configure static file caching in ASP.NET Core.  
   **Answer: True**
10. React cannot consume Web APIs directly.  
    **Answer: False**



****

**Chapter-12**

**Testing and Deployment**

**📘 UNIT TESTING WITH xUNIT**

**🔧 xUnit Overview:**

* xUnit is a free, open-source unit testing tool for .NET.
* Test class is created using [Fact] or [Theory] attributes.

**🧪 Example: xUnit Test for Calculator**

**Service to Test:**

public class Calculator

{

public int Add(int a, int b) => a + b;

}

**Unit Test:**

public class CalculatorTests

{

[Fact]

public void Add\_ReturnsCorrectSum()

{

var calc = new Calculator();

int result = calc.Add(2, 3);

Assert.Equal(5, result);

}

}

**📘 Explanation:**

* [Fact]: Marks a test method with no parameters.
* Assert.Equal(expected, actual): Validates test result.

**📘 MOCKING & DEPENDENCY INJECTION IN TESTS**

**🔧 Why Mock?**

* Helps test code **in isolation** by replacing dependencies with fake objects.

**🧪 Example using Moq**

**Service + Interface:**

public interface IMessageService

{

string GetMessage();

}

public class Greeting

{

private readonly IMessageService \_messageService;

public Greeting(IMessageService messageService)

{

\_messageService = messageService;

}

public string Greet() => \_messageService.GetMessage();

}

**Test with Moq:**

public class GreetingTests

{

[Fact]

public void Greet\_ReturnsMessage()

{

var mock = new Mock<IMessageService>();

mock.Setup(m => m.GetMessage()).Returns("Hello!");

var greeting = new Greeting(mock.Object);

Assert.Equal("Hello!", greeting.Greet());

}

}

**📘 Explanation:**

* Mock<IMessageService>(): Creates a mock object.
* Setup() defines method behavior for testing.
* mock.Object injects the fake service.

**📘 HOSTING ASP.NET CORE APPS**

**1️ IIS Hosting**

* Publish project → deploy to IIS using .exe or .dll
* Use **ASP.NET Core Hosting Bundle**

dotnet publish -c Release

**2️ Azure Hosting**

* Publish directly from Visual Studio or CLI
* Use App Services or Azure Container Apps

**3️ Docker Hosting**

* Create Dockerfile in root

**Dockerfile:**

FROM mcr.microsoft.com/dotnet/aspnet:6.0

WORKDIR /app

COPY ./publish .

ENTRYPOINT ["dotnet", "MyApp.dll"]

**📘 CI/CD USING GITHUB ACTIONS / AZURE DEVOPS**

**✅ GitHub Actions YAML Example:**

name: Build and Deploy

on: [push]

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- name: Setup .NET

uses: actions/setup-dotnet@v2

with:

dotnet-version: '6.0.x'

- name: Build

run: dotnet build

- name: Test

run: dotnet test

**✅ Azure DevOps Pipelines YAML:**

trigger:

- main

pool:

vmImage: 'windows-latest'

steps:

- task: UseDotNet@2

inputs:

packageType: 'sdk'

version: '6.0.x'

- script: dotnet build

- script: dotnet test

**✅ MCQs (Multiple Choice Questions)**

**1. What does xUnit use to define a test method?**  
A) [Test]  
B) [Run]  
C) [Fact]  
D) [Method]  
**Answer:** C) [Fact]

**2. What is the purpose of mocking?**  
A) Build production apps  
B) Replace dependencies for testing  
C) Encrypt data  
D) Publish apps  
**Answer:** B) Replace dependencies for testing

**3. Which package is commonly used for mocking in .NET tests?**  
A) NUnit  
B) Moq  
C) AutoMock  
D) FakeLib  
**Answer:** B) Moq

**4. What file is used to define Docker configurations?**  
A) web.config  
B) Dockerfile  
C) appsettings.json  
D) deploy.xml  
**Answer:** B) Dockerfile

**5. What CLI command builds and publishes your app for deployment?**  
A) dotnet serve  
B) dotnet build-release  
C) dotnet publish  
D) dotnet deploy  
**Answer:** C) dotnet publish

**6. Which attribute is used for parameterized tests in xUnit?**  
A) [Fact]  
B) [Theory]  
C) [TestCase]  
D) [Input]  
**Answer:** B) [Theory]

**7. Which CI/CD platform is integrated directly with GitHub?**  
A) Azure DevOps  
B) Jenkins  
C) GitHub Actions  
D) TeamCity  
**Answer:** C) GitHub Actions

**8. Which Azure service is commonly used to host web apps?**  
A) Blob Storage  
B) App Service  
C) Virtual Machines  
D) SQL Database  
**Answer:** B) App Service

**9. In CI/CD, which step usually comes right after the build?**  
A) Clone  
B) Test  
C) Notify  
D) Commit  
**Answer:** B) Test

**10. Which xUnit assertion checks for equality?**  
A) Assert.Same()  
B) Assert.True()  
C) Assert.Equal()  
D) Assert.Check()  
**Answer:** C) Assert.Equal()

**✍️ Fill in the Blanks**

1. xUnit uses the \_\_\_\_\_\_\_\_\_\_ attribute to define unit test methods.  
   **Answer:** [Fact]
2. \_\_\_\_\_\_\_\_\_\_ is a library used for creating fake objects in unit testing.  
   **Answer:** Moq
3. The method used to run xUnit tests from CLI is \_\_\_\_\_\_\_\_\_\_.  
   **Answer:** dotnet test
4. \_\_\_\_\_\_\_\_\_\_ allows you to deploy web apps on Microsoft's cloud platform.  
   **Answer:** Azure
5. A \_\_\_\_\_\_\_\_\_\_ file defines steps for GitHub Actions workflows.  
   **Answer:** YAML
6. ASP.NET Core apps can be containerized using \_\_\_\_\_\_\_\_\_\_.  
   **Answer:** Docker
7. \_\_\_\_\_\_\_\_\_\_ is the attribute in xUnit used for parameterized tests.  
   **Answer:** [Theory]
8. The \_\_\_\_\_\_\_\_\_\_ bundle is required to host ASP.NET Core apps on IIS.  
   **Answer:** Hosting
9. CI stands for Continuous \_\_\_\_\_\_\_\_\_\_.  
   **Answer:** Integration
10. CD stands for Continuous \_\_\_\_\_\_\_\_\_\_.  
    **Answer:** Deployment

**✔️ True or False**

1. xUnit is a third-party testing framework used in .NET.  
   **True**
2. The [Test] attribute is used in xUnit for test methods.  
   **False** (It's [Fact])
3. Moq can only be used with console applications.  
   **False**
4. Azure App Service can host ASP.NET Core applications.  
   **True**
5. GitHub Actions only supports Python projects.  
   **False**
6. The dotnet test command runs unit tests in .NET Core.  
   **True**
7. CI/CD helps automate building, testing, and deploying applications.  
   **True**
8. Docker allows apps to run the same way in any environment.  
   **True**
9. [Theory] attribute in xUnit is used for testing multiple inputs.  
   **True**
10. Azure DevOps is unrelated to CI/CD.  
    **False**