Topic-1 .Net Core Fundamental

❖ .Net Core Overview

- ➢ .NET Core is a new version of .NET Framework, which is a **free**, **open-source**, general-purpose development platform maintained by Microsoft.
- ➢ It is a cross-platform framework that runs on Windows, macOS, and Linux operating systems.
- ➢ .NET Core is written from scratch to make it modular, lightweight, fast, and cross-platform Framework.
- ➢ It includes the core features that are required to run a basic .NET Core app.
- ➢ Other features are provided as NuGet packages, which you can add to your application as needed.
- ➢ In this way, the .NET Core application speed up the performance, reduces the memory footprint and becomes easy to maintain.

❖ Why .NET Core?

- ➢ There are some limitations with the .NET Framework. For example, it only runs on the Windows platform. Also, you need to use different.
- ➢ .NET APIs for different Windows devices such as Windows Desktop, Windows Store, Windows Phone, and Web applications. In addition to this, the .NET Framework is a machine-wide framework.
- ➢ Any changes made to it affect all applications taking a dependency on it.
- ➢ The **main objective** of .NET Core is to make .NET open-source, cross-platform compatible that can be used in a wide variety of verticals, from the data center to touch-based devices

❖ .NET Core Characteristics

- ➢ **Open-source Framework:** NET Core is an open-source framework maintained by Microsoft and available on GitHub under MIT and Apache 2 licenses. It is a .NET Foundation project.

- ➢ **Cross-platform:** .NET Core runs on Windows, macOS, and Linux operating systems. There is a different runtime for each operating system that executes the code and generates the same output.
- ➢ **Consistent across Architectures:** Execute the code with the same behaviour in different instruction set architectures, including x64, x86, and ARM.
- ➢ **Wide-range of Applications:** Various types of applications can be developed and run on .NET Core platform such as mobile, desktop, web, cloud, IoT, machine learning, microservices, game, etc.
- ➢ **Supports Multiple Languages:** You can use C#, F#, and Visual Basic programming languages to develop .NET Core applications. You can use your favourite IDE, including Visual Studio 2019/2022, Visual Studio Code, Sublime Text, Vim, etc.
- ➢ **Modular Architecture:** .NET Core supports modular architecture approach using NuGet packages. There are different NuGet packages for various features that can be added to the .NET Core project as needed. Even the .NET Core library is provided as a NuGet package.
- ➢ **CLI Tools:** .NET Core includes CLI tools (Command-line interface) for development and continuous-integration.
- ➢ **Flexible Deployment:** .NET Core application can be deployed user-wide or system-wide or with Docker Containers.
- ➢ **Compatibility:** Compatible with .NET Framework and Mono APIs by using .NET Standard specification.

❖ ASP.Net Core

ASP.NET Core is an open source and cloud-optimized web framework for developing modern web applications that can be developed and run on Windows, Linux and the Mac. It includes the MVC framework, which now combines the features of MVC and Web API into a single web programming framework.

- ➢ ASP.NET Core apps can run on .NET Core or on the full .NET Framework.

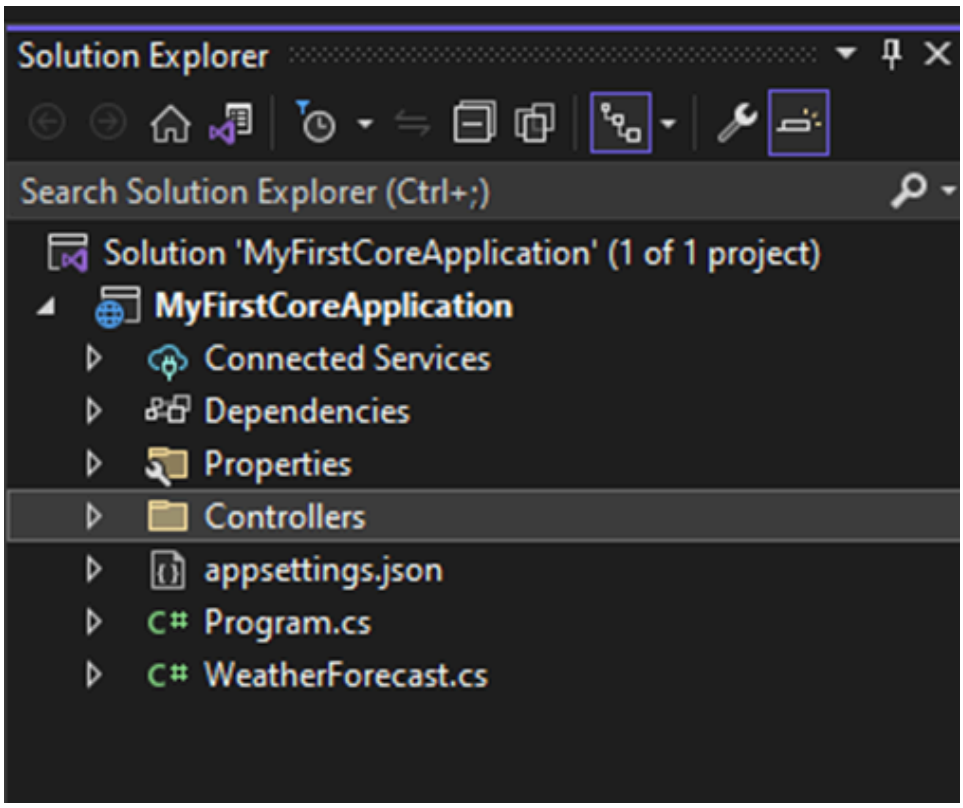
- ➢ It was architected to provide an optimized development framework for apps that are deployed to the cloud or run on-premises.
- ➢ You can develop and run your ASP.NET Core apps cross-platform on Windows, Mac and Linux.

❖ Advantages of ASP.NET Core

ASP.NET Core comes with the following advantages −

- ➢ ASP.NET Core has a number of architectural changes that result in a much leaner and modular framework.
- ➢ ASP.NET Core is no longer based on System.Web.dll. It is based on a well factored NuGet packages.
- ➢ This allows you to optimize your app to include just the NuGet packages you need.
- ➢ The benefits of a smaller app surface area include tighter security, reduced servicing, improved performance, and decreased costs.
- ➢ Single aligned web stack for Web UI and Web APIs.
- ➢ Cloud-ready environment-based configuration.
- ➢ Built-in support for dependency injection.
- ➢ Ability to host on IIS or self-host in your own process.

❖ Project Structure

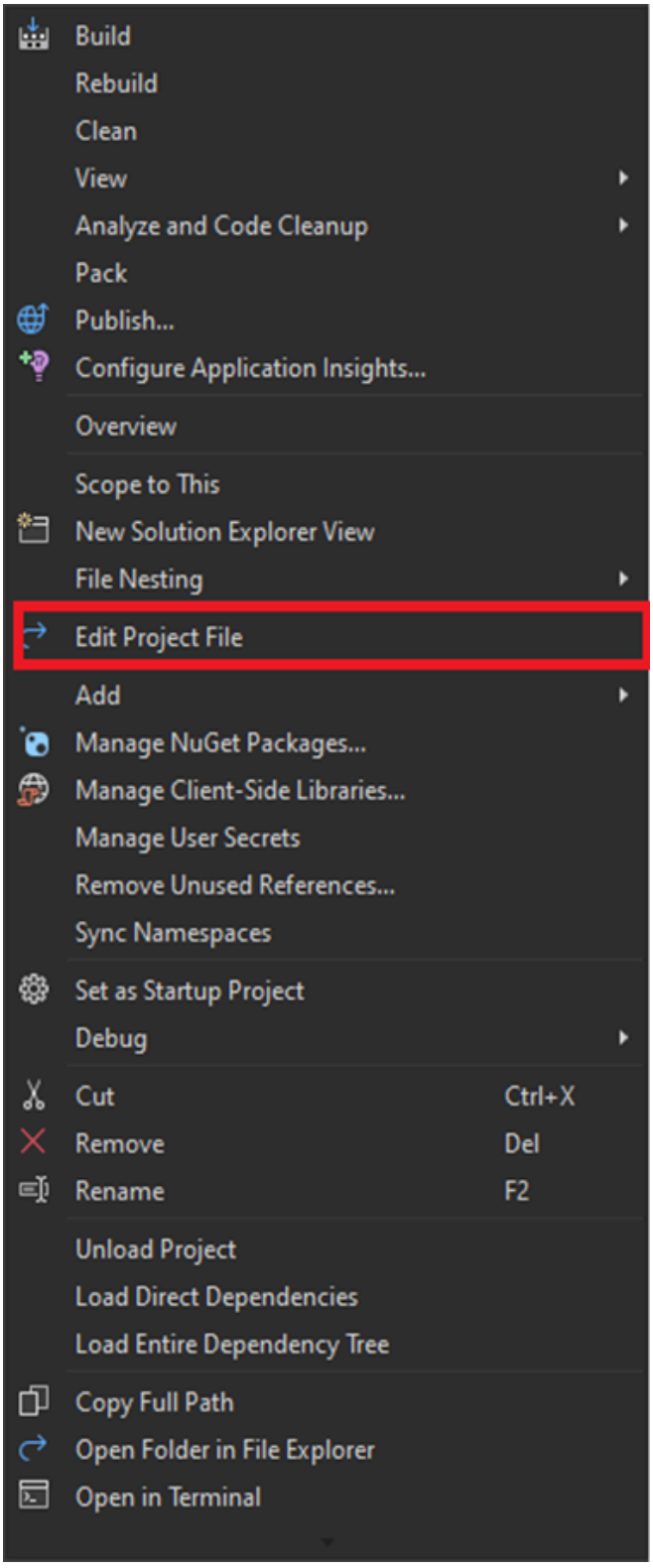


❖ Project Folder Structure Description

- ➢ The details of the default project structure can be seen in the solution explorer, it displays all the projects related to a single solution.

❖ .csproj File

- ➢ Double click on the project name in Solution Explorer to open the .csproj file in the editor.
- ➢ Right-click on the project and then click on Edit Project File in order to edit the .csproj file.



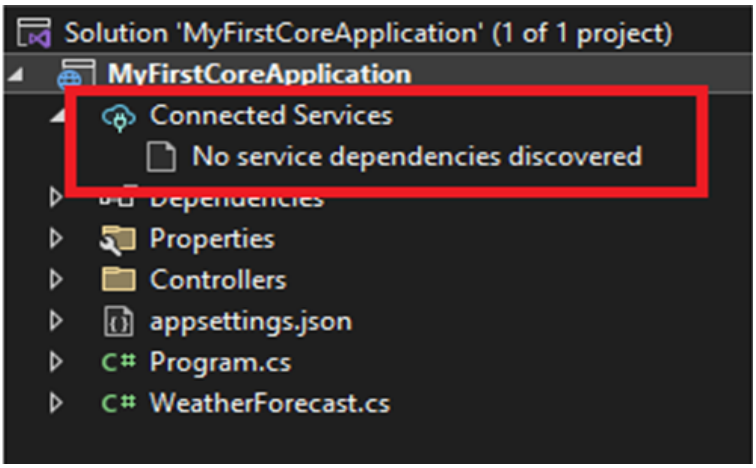
Once clicked on Edit Project File, .csproj file will be opened in Visual Studio as shown below.

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

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.2.3" />
  </ItemGroup>

</Project>
```

- As you can see the project's SDK is Microsoft.NET.Sdk.Web. The target framework is net 6.0 indicating that we are using .NET 6. Notice the Nullable and ImplicitUsings elements.
- The <Nullable> elements decide the project wide behaviour of Nullable of Nullable reference types. The value of enable indicates that the Nullable reference types are enabled for the project.
- The < ImplicitUsings > element can be used to enable or disable.
- < ImplicitUsings > is set to enable, certain namespaces are implicitly imported for you.

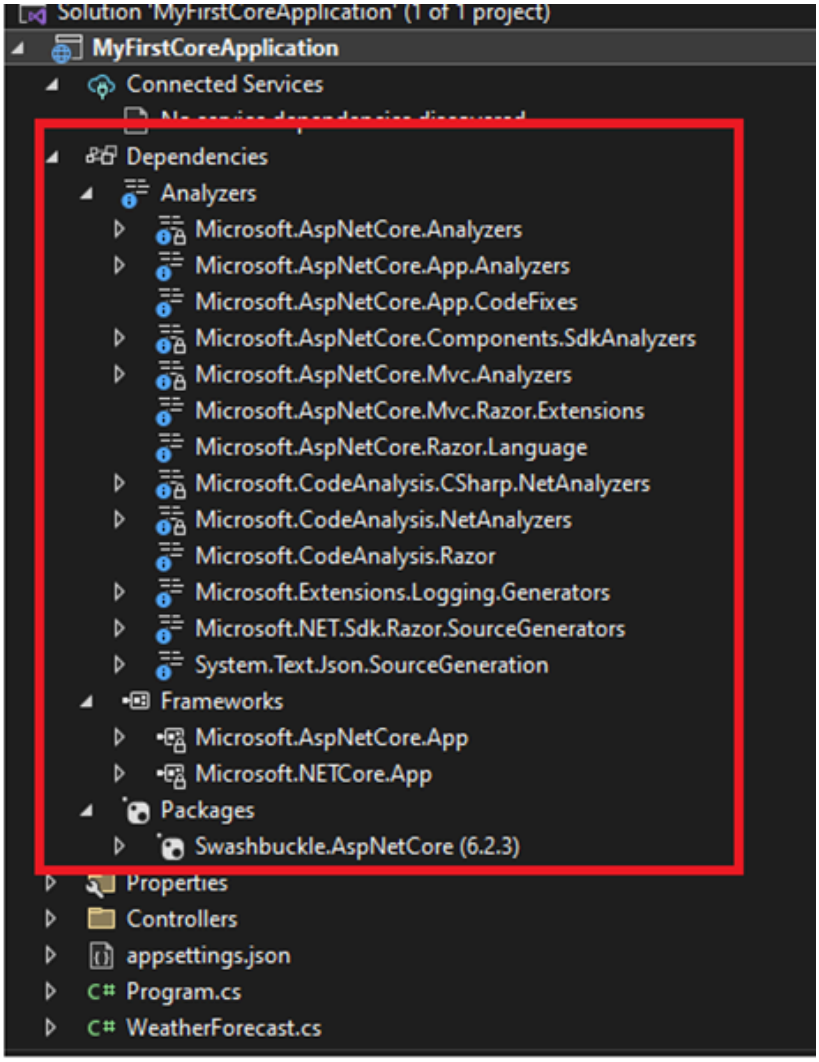
❖ Connected Services

- It contains the details about all the service references added to the project. A new service can be added here, for example, if you want to add access to Cloud Storage of Azure Storage you can add the service here.



❖ Dependencies

- ➢ The **Dependencies** node contains all the references of the NuGet packages used in the project.
- ➢ Here the **Frameworks** node contains reference two most important dotnet core runtime and asp.net core runtime libraries.
- ➢ Project contains all the installed server-side NuGet packages, as shown below.



❖ Properties

- ➢ The Properties folder contains a **launchSettings.json** file, which contains all the information required to launch the application.

- Configuration details about what action to perform when the application is executed.
- Configuration contains details like IIS settings, application URLs, authentication, SSL port details, etc.

❖ WWWroot

- This is the webroot folder and all the static files required by the project are stored and served from here.
- The webroot folder contains a sub-folder to categorize the static file types, like all the Cascading Style Sheet files, are stored in the **CSS folder**, all the javascript files are stored in the **js folder** and the external libraries like bootstrap, jquery are kept in the **library folder**.
- Generally, there should be separate folders for the different types of static files such as JavaScript, CSS, Images, library scripts, etc.
- You can access static files with base URL and file name. For example, we can access the above app.css file in the CSS folder by "**http://localhost:<port>/css/app.css"**.

❖ Controllers

- Controller handles all the incoming requests.
- All the controllers needed for the project are stored in this folder.
- Controllers are responsible for handling HTTP Requests, manipulating the model.
- Each controller class inherits a Controller class or ControllerBase class.
- Each controller class has "Controller" as a suffix on the class name, for example, the default "WeatherForecastController.cs" file can be found here.

❖ Program.CS

- Program.cs is where the application starts.
- Program.cs class file is the entry point of our application and creates an instance of IWebHost which hosts a web application.
- WebHost is used to create instance of IWebHost and IWebHostBuilder which are pre-configured defaults. The CreateDefaultBuilder() method creates a new instance of WebHostBuilder.

- UseStartup<startup>() method specifies the Startup class to be used by the web host.
- We can also specify our custom class in place of startup.
  Build() method returns an instance of IWebHost and Run() starts the web application until it stops.
- Program.cs in ASP.NET Core makes it easy for us to setup a web host.

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

❖ Startup.cs

- ASP.NET Core application must include the Startup class. It is like Global.aspx in the traditional .NET application. As the name suggests, it is executed first when the application starts. The inception of the startup class is in the OWIN (Open Web Interface for.NET) application that is a specification to reduce dependency of application on server.
  - OWIN defines a standard interface between .NET web servers and web applications.

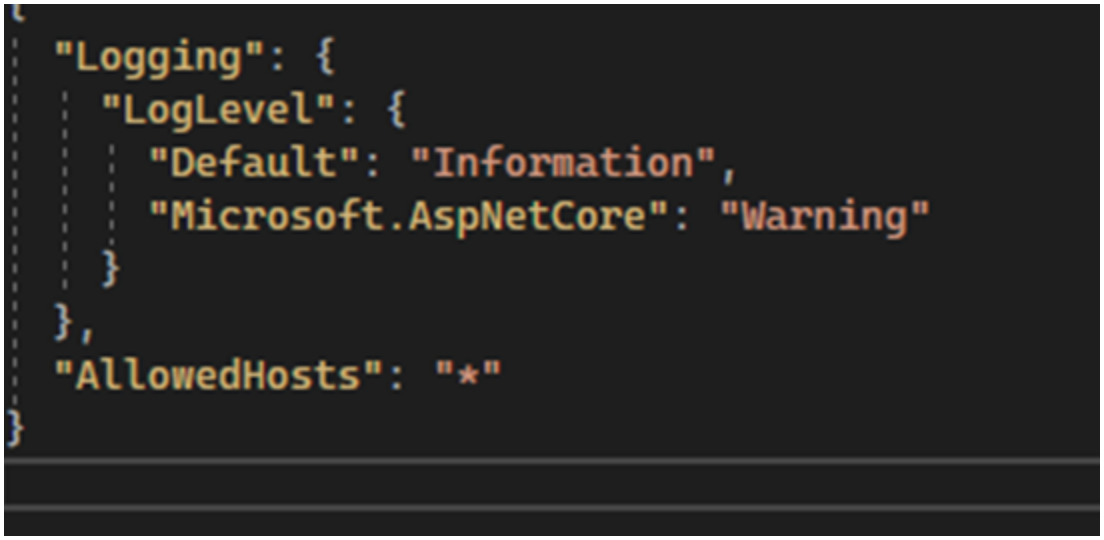
        - The goal of the OWIN interface is to decouple server and application, and encourage the development of simple modules for .NET web development.
    - ➢ It can have any access modifier (public, private, internal). This is the entry point of the ASP.net application.
    - ➢ It contains application configuration related items. it is not necessary that the class name be "Startup".
    - ➢ The ASP.net core application is a Console app and we have to configure a web host to start listening. The "Program" class does this configuration.

❖ launchSettings.json

- ➢ The settings in this file are used when we run this ASP.NET core project either from Visual Studio or by using .NET Core CLI.
- ➢ This file is only used on local development machines.
- ➢ We do not need it for publishing our asp.net core application.
- ➢ If there are certain settings that you want your asp.net core application to use when you publish and deploy your app, store them in appsettings.json file.
- ➢ We usually store our application configuration settings in this file.
- ➢ There are application url in **profiles** on which url our application will run.

❖ appsettings.json

- ➢ In the Asp.Net Core application we may not find the web.config file. Instead we have a file named "appsettings.json". So to configure the settings like database connections, Mail settings or some other custom configuration settings we will use the "appsettings.json".
- ➢ The appsettings in Asp.net core have different configuration sources as shown below.
  - appsettings.json File
  - Environment Variable
  - User Secrets
  - Command Line Arguments


{
"Logging": {
"LogLevel": {
"Default": "Information",
"Microsoft.AspNetCore": "Warning"
}
},
"AllowedHosts": "*"
}