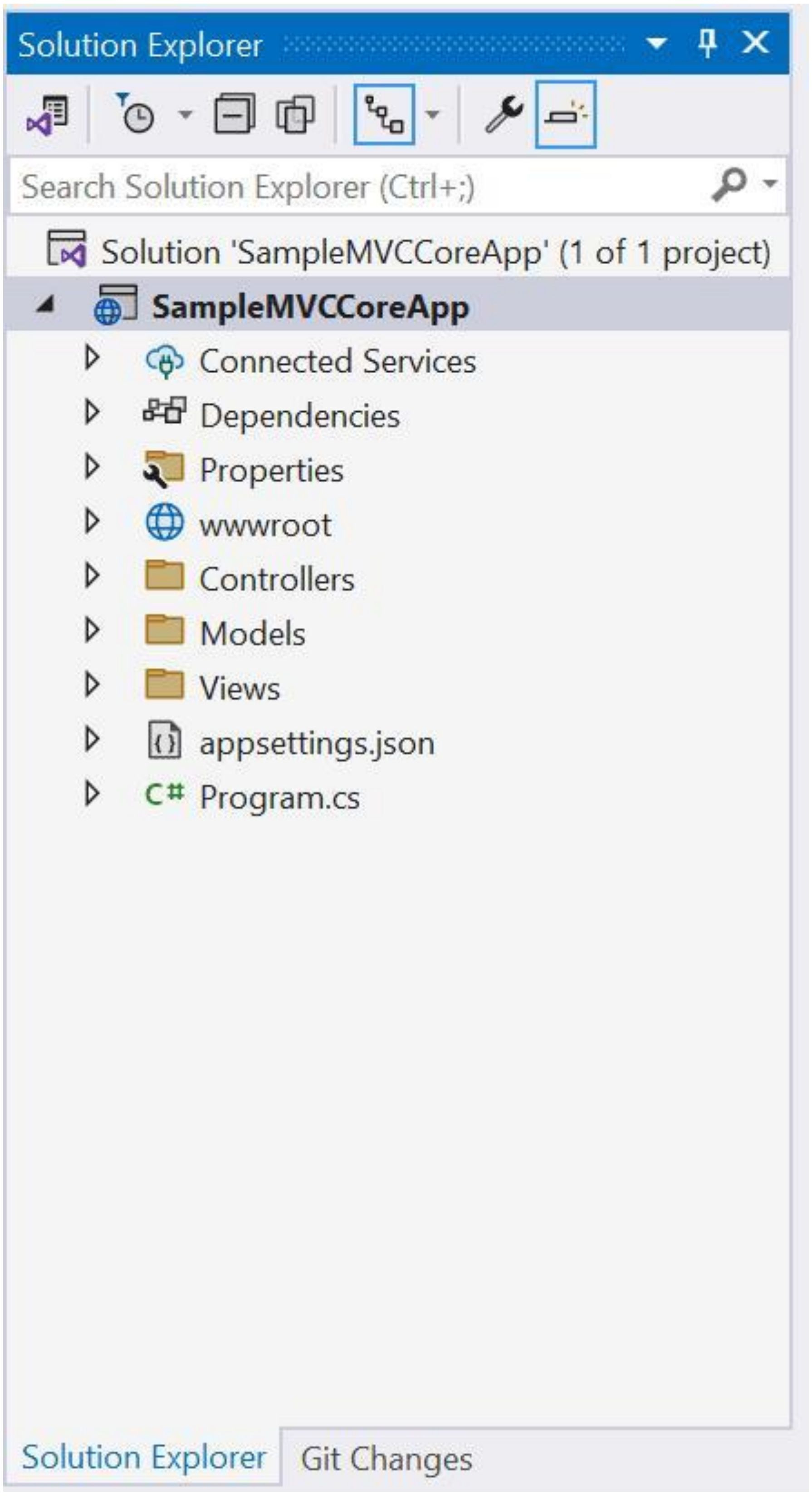
ASP.NET Core MVC Project Structure work flow

Here, you will learn about the project structure and significance of each file and folder in the ASP.NET Core MVC (.NET 7) application.

In the previous chapter, We created the ASP.NET Core MVC application. The following is a default project structure of the ASP.NET Core MVC application in Visual Studio.



Note:

ASP.NET Core project files and folders are synchronized with physical files and folders. If you add a new file or folder in project folder then it will directly reflect in the solution explorer. You don't need to add it in the project explicitly by right clicking on the project.

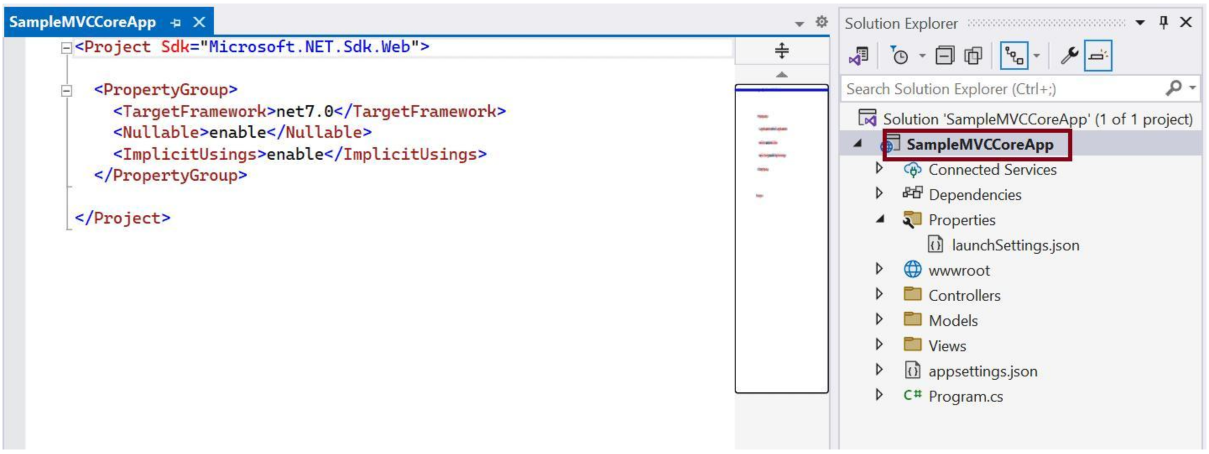
Let's understand the significanse of each file and folder.

Solution File

Visual Studio creates the top-level solution file (.sln) that can contain one or more ASP.NET projects. Here, "Solution 'SampleMVCCoreApp' (1 of 1 project)" is the solution file. You can right-click on it an click on 'Open Folder in File Explorer' and see the 'SampleMVCCoreApp.sln' file is created. This is our solution file.

Project File

Under the solution node in the Solution Explorer, you can see our project node 'SampleMVCCoreApp'. All the files under this node will be for 'SampleMVCCoreApp' project.

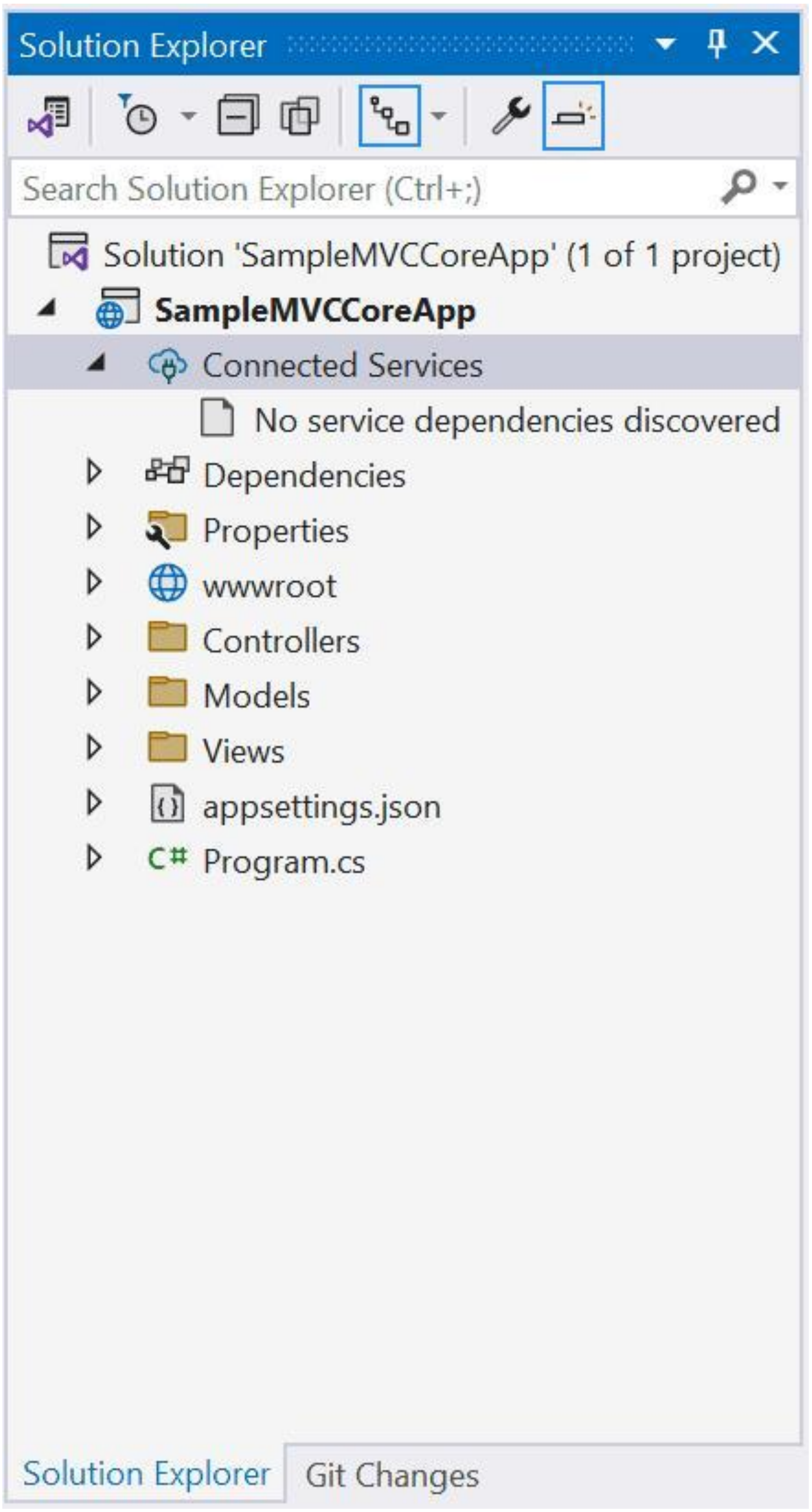


```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

</Project>
```

Double click on the project file and you will see some project settings related to the targeted target .NET Frameworks, project folders, NuGet package references, etc.

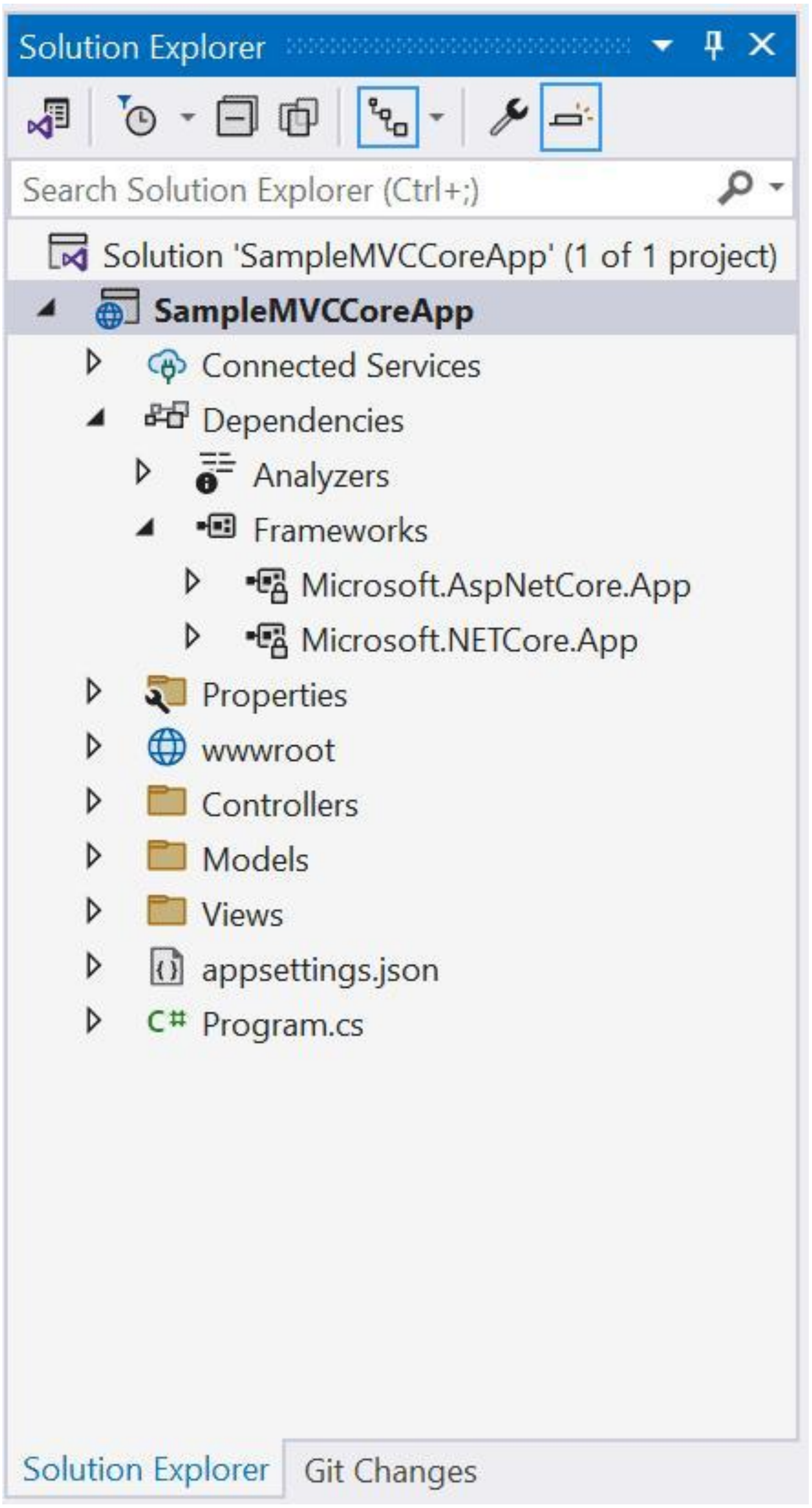
Connected Services

The 'Connected Services' node contains the list of external services, APIs, and other data sources. It helps in the integration with various service providers, such as Azure, AWS, Google Cloud, and third-party services like authentication providers or databases. We are not using any service yet so it will be empty for now.



Dependencies

The Dependencies node contains the list of all the dependencies that your project relies on, including NuGet packages, project references, and framework dependencies.



Solution 'SampleMVCCoreApp' (1 of 1 project)
- **SampleMVCCoreApp**
  - Connected Services
  - Dependencies
    - Analyzers
    - Frameworks
      - Microsoft.AspNetCore.App
      - Microsoft.NETCore.App
  - Properties
  - wwwroot
  - Controllers
  - Models
  - Views
  - appsettings.json
  - Program.cs

It contains two nodes, Analyzers and Frameworks.

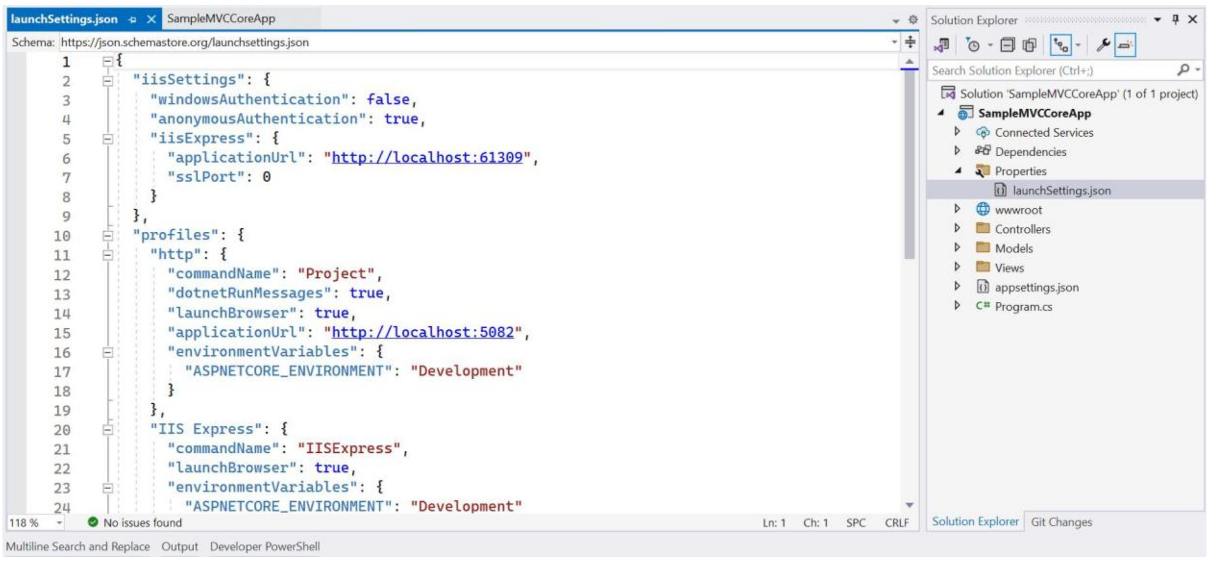
Analyzers are extensions for static code analysis. They help you to enforce coding standards, identify code quality issues, and detect potential problems in your code. Analyzers can be custom rules or third-party analyzers provided by NuGet packages.

Frameworks node contains the target framework that your project is designed to run on. We have created an ASP.NET Core MVC application. So, it contains two frameworks, the .NET Core (Microsoft.NETCore.App) and ASP.NET Core (Microsoft.AspNetCore.App) framework. Click on any node and press F4 to see it's version, file path, etc.

You can expan each node to see all the assemblies it contains. In the future, whatever NuGet packages you install, all will be displayed under the dependencies node. So that you can uninstall it when you don't use it.

Properties

The Properties node includes launchSettings.json file which includes Visual Studio profiles of debug settings. launchSettings.json helps developers to configure the debugging and launch profiles of their ASP.NET (also known as ASP.NET Core) applications for different environments such as development, staging, production, etc.. The following is a default launchSettings.json file.



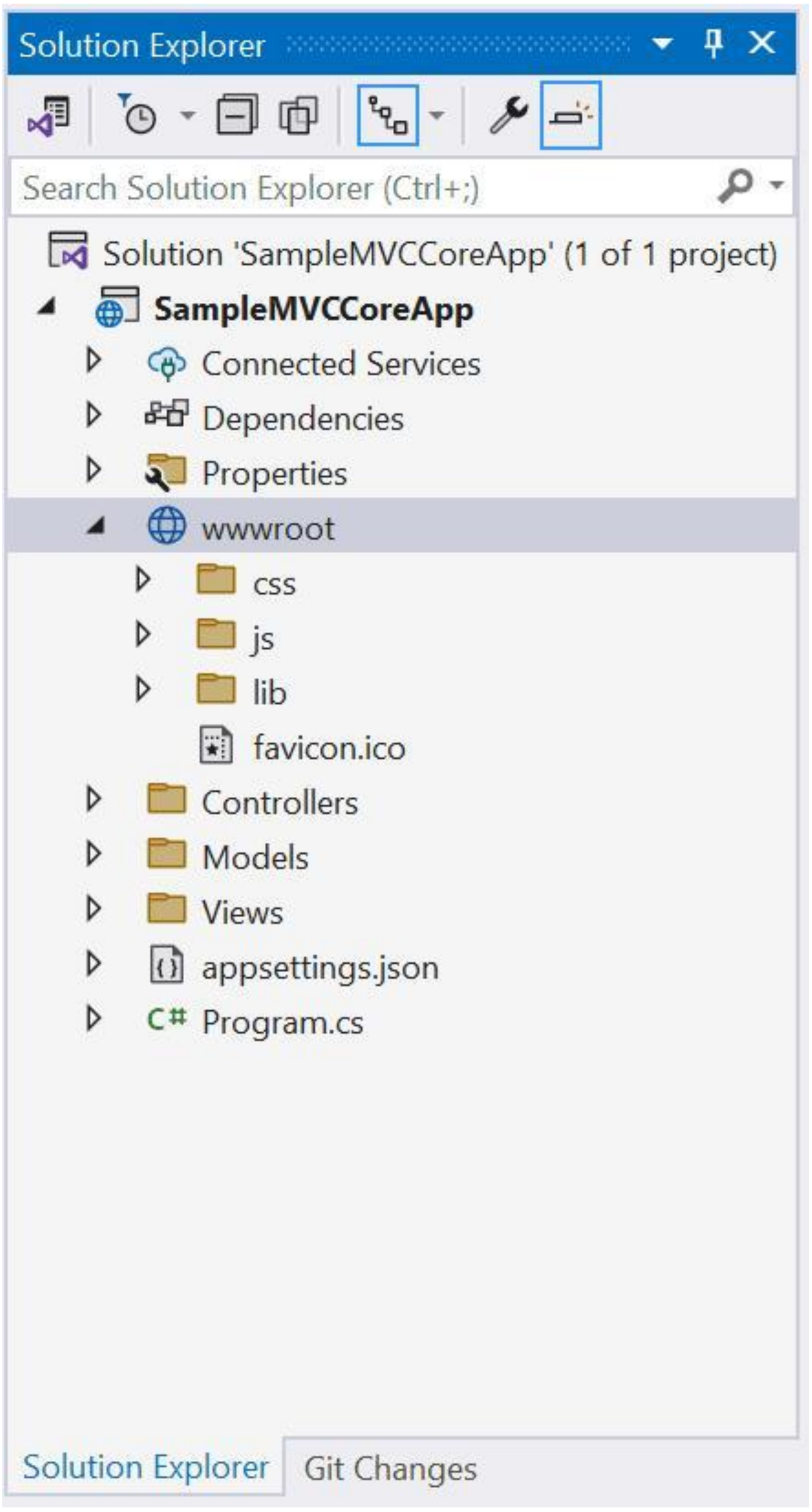
Schema: https://json.schemastore.org/launchsettings.json

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:61309",
      "sslPort": 0
    }
  },
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5082",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
```

wwwroot

By default, the **wwwroot** folder in the ASP.NET Core project is treated as a web root folder. Static files can be stored in any folder under the web root and accessed with a relative path to that root.

All the css, JavaScript, and external library files should be stored here which are being reference in the HTML file.

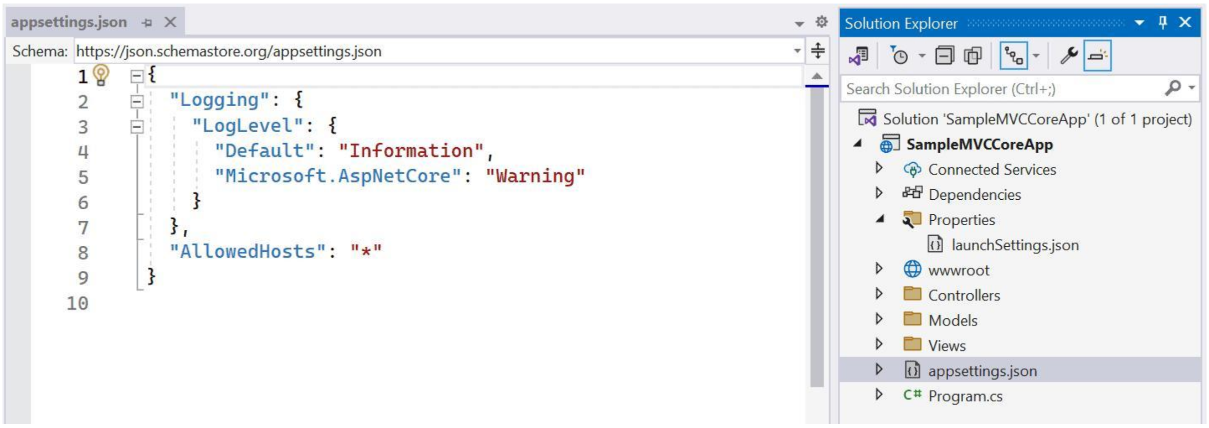


Controllers, Models, Views

The Controllers, Models, and Views folders include controller classes, model classes and cshtml or vbhtml files respectively for MVC application.

appsettings.json

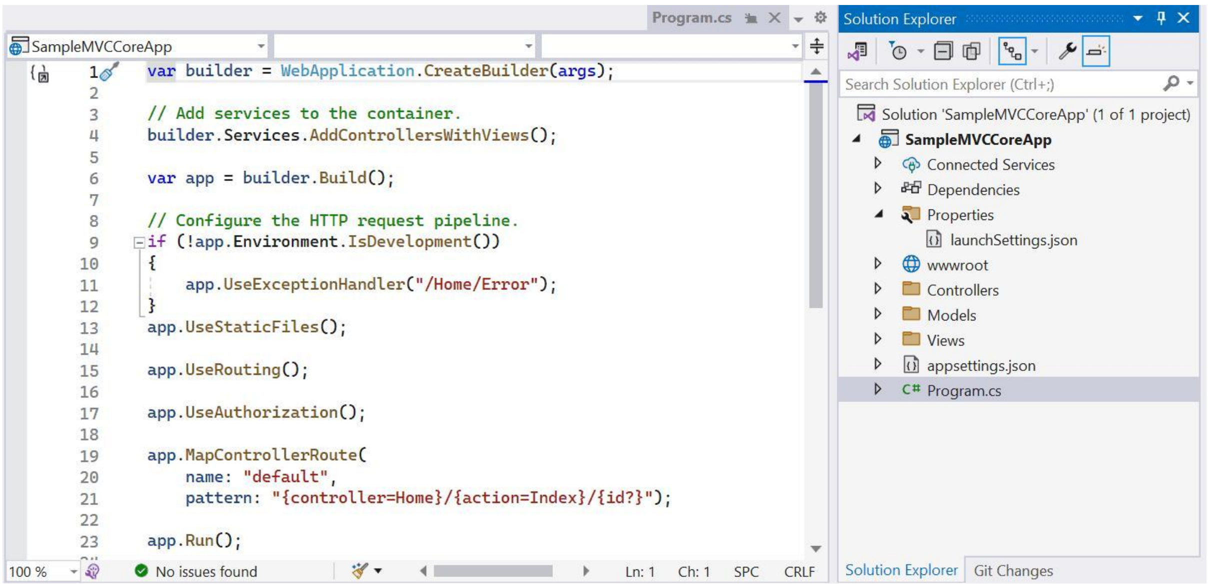
The appsettings.json file is a configuration file commonly used in .NET applications, including ASP.NET Core and ASP.NET 5/6, to store application-specific configuration settings and parameters. It allows developers to use JSON format for the configurations instead of code, which makes it easier to add or update settings without modifying the application's source code.



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

program.cs

The last file 'program.cs' is an entry point of an application. ASP.NET Core web application is a console application that builds and launches a web application.



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

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
}
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

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

app.Run();
```

The above program.cs file uses the top-level statements so when the application starts, it starts executing code from top to bottom.

Top-level statements in C# are a feature introduced in C# 9.0 to simplify the structure of your C# code by allowing you to write statements outside of traditional class and method definitions. With top-level statements, you can write C# code at the top level of a file without the need for a containing class or method.