

ASP .NET CORE

WEB DEVELOPMENT



Overview

What is ASP.Net Core

History and Evolution of ASP.Net core

Asp.Net core web development Architecture

Basic Asp.net core Web development project structure



What is Asp.Net core

- ASP.NET Core is a cross-platform, open-source web framework for building modern web applications.
- It was developed by Microsoft and first released in 2016.
- ASP.NET Core offers a high-performance, modular architecture that is designed to be lightweight and scalable.
- It supports multiple programming languages, including C#, F#, and Visual Basic.
- ASP.NET Core includes built-in for cloud-based deployments and microservices architecture.

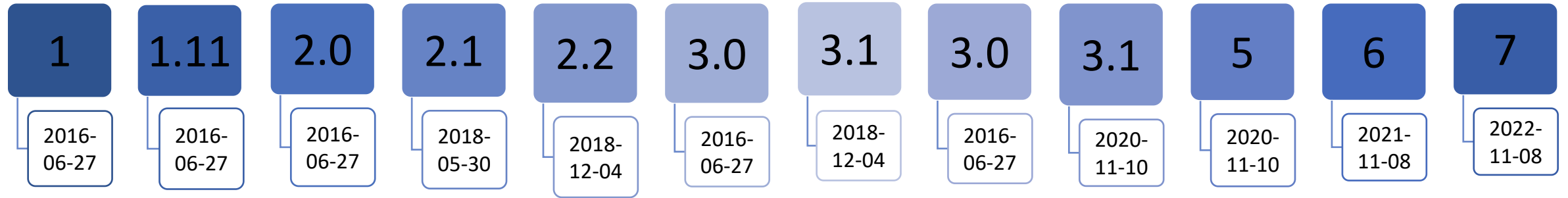


What is Asp.Net core

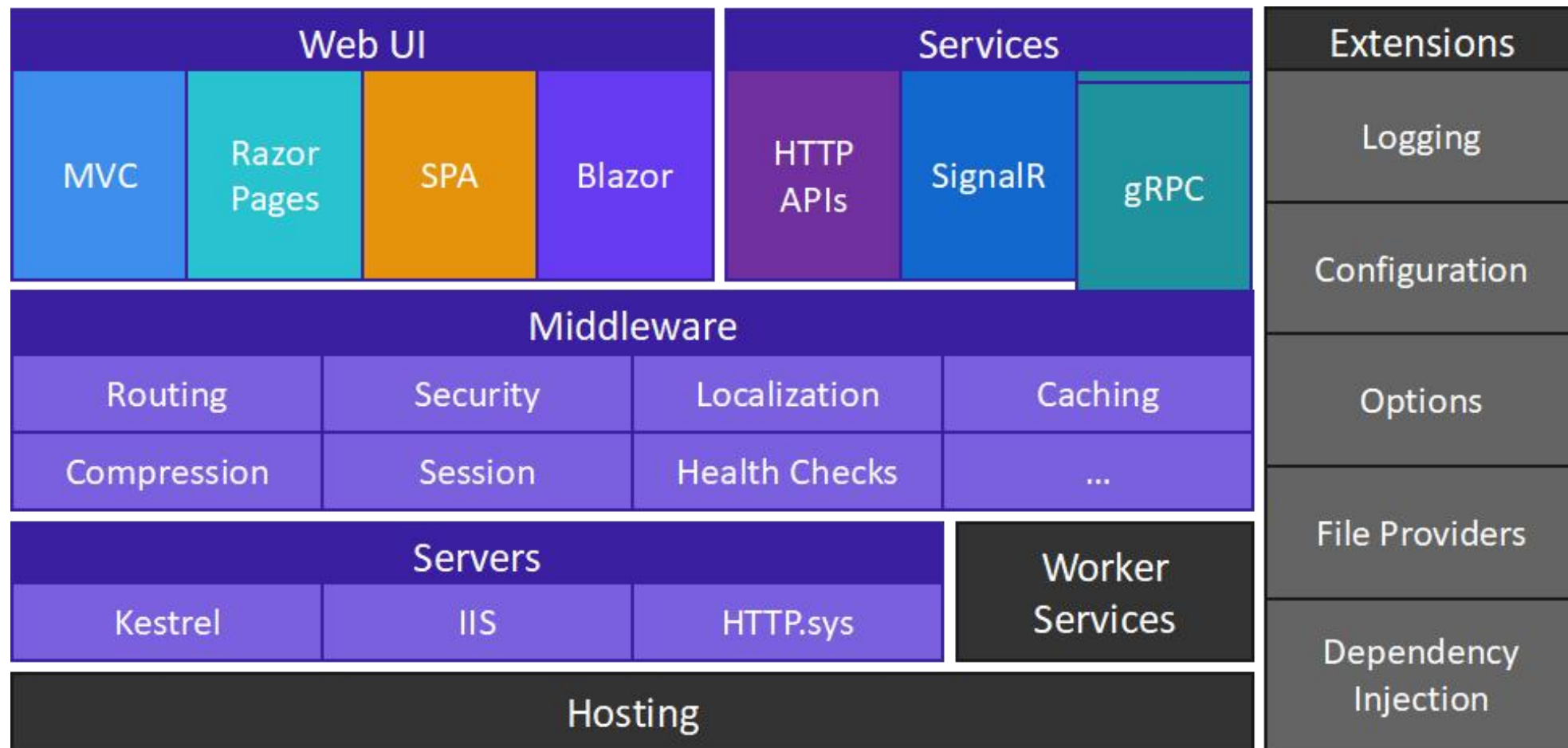
- It offers a range of features for building web applications, including model binding, routing, middleware, and dependency injection.
- ASP.NET Core integrates with a range of front-end frameworks and libraries, including React, Angular, and Vue.js.
- It offers built-in support for security, including authentication and authorization.
- ASP.NET Core is constantly evolving, with regular updates and new features being added.
- It is widely used in industry, with many large companies and organizations adopting it for their web development projects.



History



Architecture



Web UI Components

MVC

Razor Pages

Single Page Apps (SPA)

Blazor



ASP .NET Core Razor Pages

- Razor Pages is a page-based model.
- UI and business logic concerns are kept separate, but within the page.
- Razor Pages is the recommended way to create new page-based or form-based apps for developers new to ASP.NET Core.
- Razor Pages provides an easier starting point than ASP.NET Core MVC.



Blazor Server

Blazor is a framework for building interactive client-side web UI with .net

Create rich interactive UIs using C# instead of Javascript.

Share server-side and client-side app logic written in .NET.

Render the UI as HTML and CSS for wide browser support, including mobile browsers.

Integrate with modern hosting platforms, such as Docker.

Build hybrid desktop and mobile apps with .NET and Blazor.

Single Page Applications (SPA)

Single Page Application (SPA) is a type of web application that loads a single HTML page and dynamically updates the content as the user interacts with the page.

ASP.NET Core SPA is a framework for building SPA applications using ASP.NET Core. It includes several tools and libraries for building modern web applications, including client-side libraries, server-side APIs, and middleware components.

ASP.NET Core SPA applications typically use a client-side JavaScript framework such as Angular, React, or Vue.js to handle the user interface and application logic.

The server-side component of the ASP.NET Core SPA application provides APIs and services for interacting with data, managing authentication and authorization, and performing other server-side tasks.

Services

HTTP APIs

SignalR

gRPC

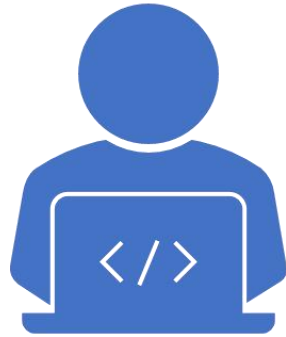


HTTP APIs

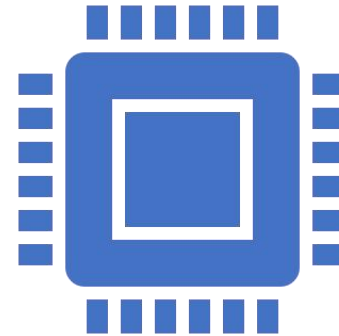
HTTP APIs are a common way of building RESTful services using HTTP as the underlying protocol.

ASP.NET Core provides built-in support for building HTTP APIs using the MVC framework, which includes controllers, actions, and routes for handling incoming requests and returning responses.

SignalR

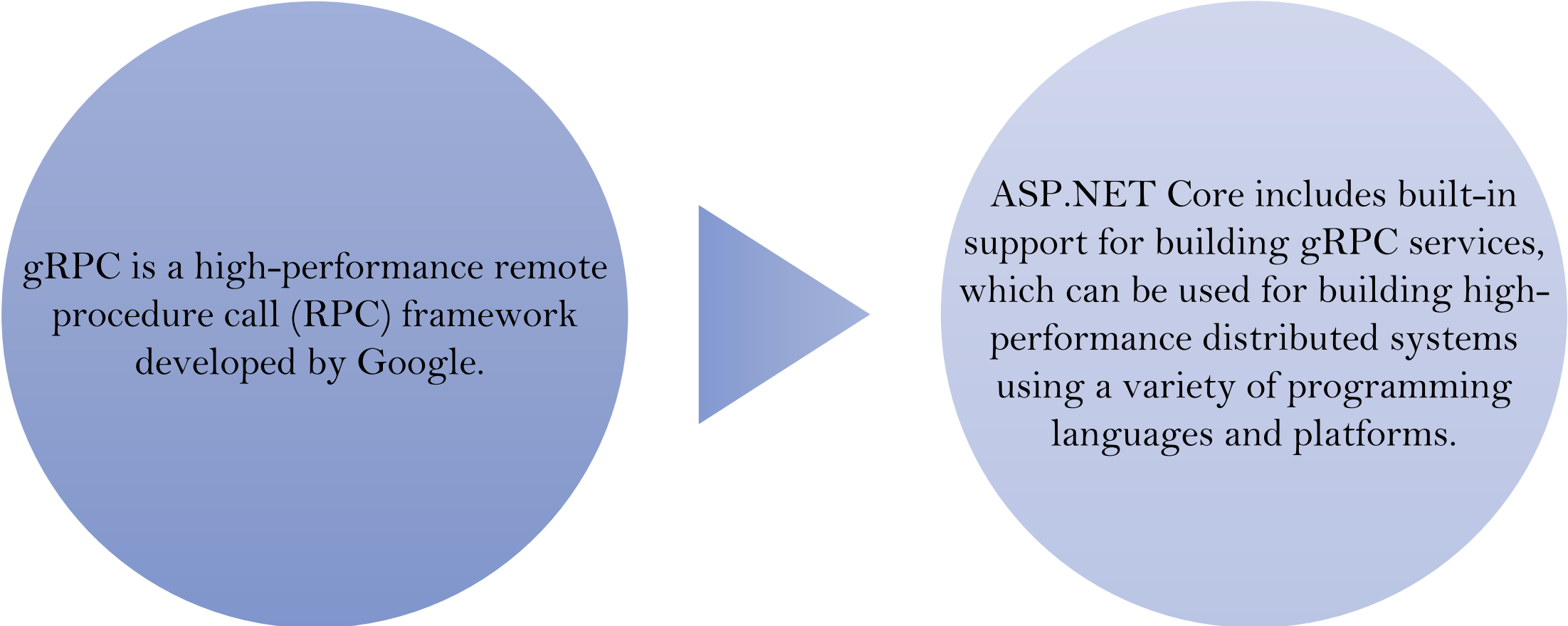


SignalR is a real-time communication library that enables bidirectional communication between clients and servers using WebSockets,.Server-Sent Events, or other transport protocols.



ASP.NET Core includes built-in support for building SignalR applications, which can be used for building real-time applications such as chat applications, collaborative editing tools, and online gaming applications.

gRPC

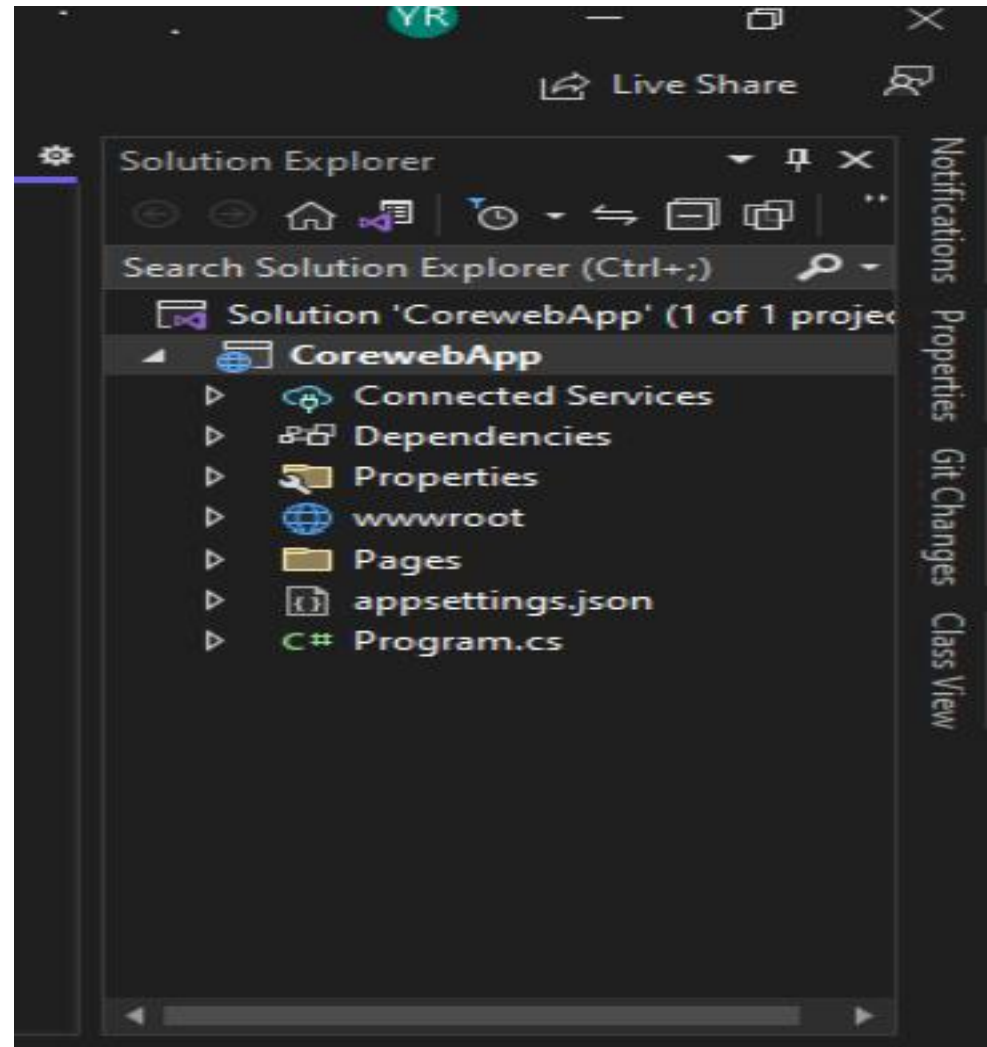


gRPC is a high-performance remote procedure call (RPC) framework developed by Google.

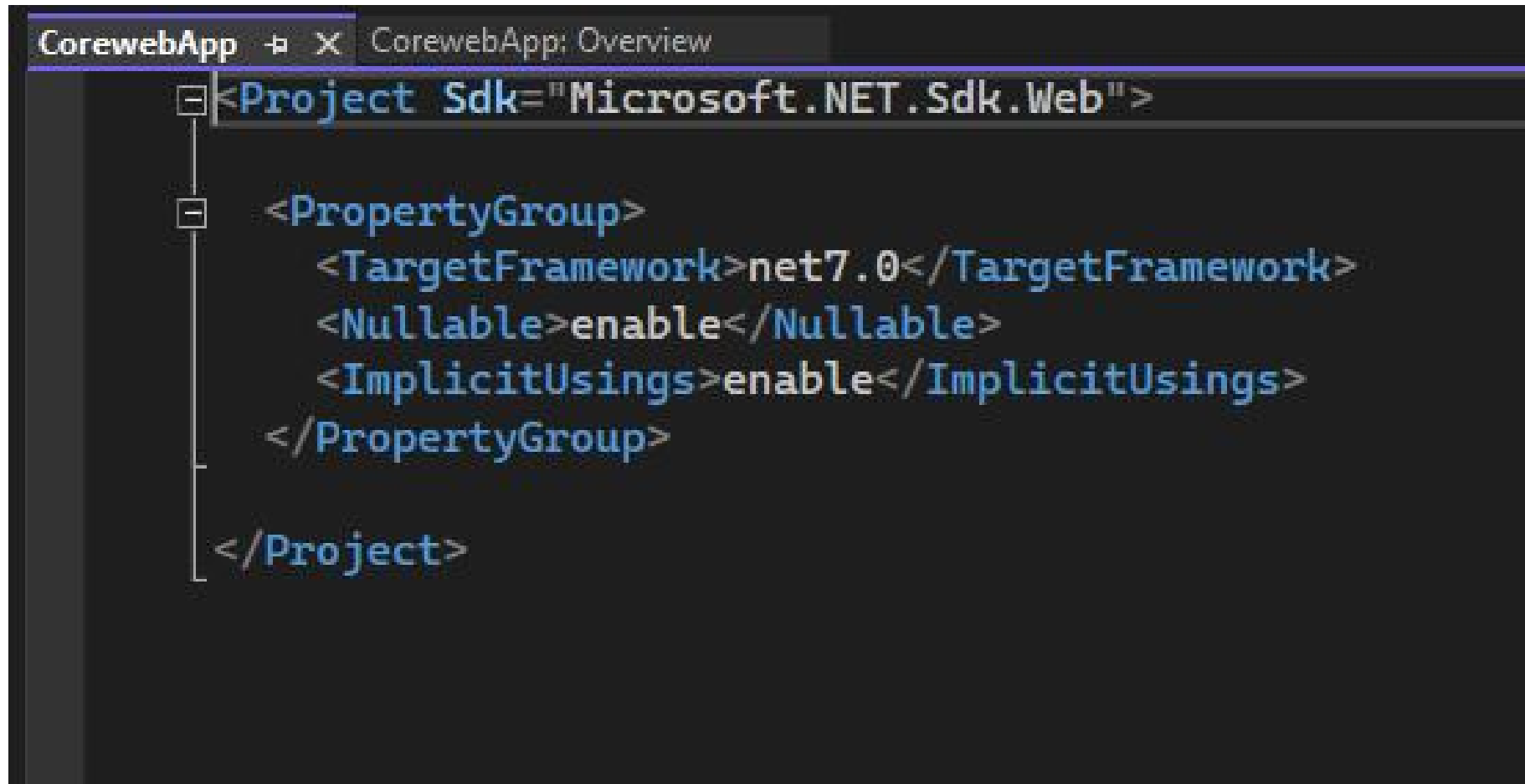
The diagram consists of two light blue circles connected by a blue arrow pointing from left to right. The left circle contains text about gRPC, and the right circle contains text about ASP.NET Core support for gRPC.

ASP.NET Core includes built-in support for building gRPC services, which can be used for building high-performance distributed systems using a variety of programming languages and platforms.

Basic Structure



.csproj file

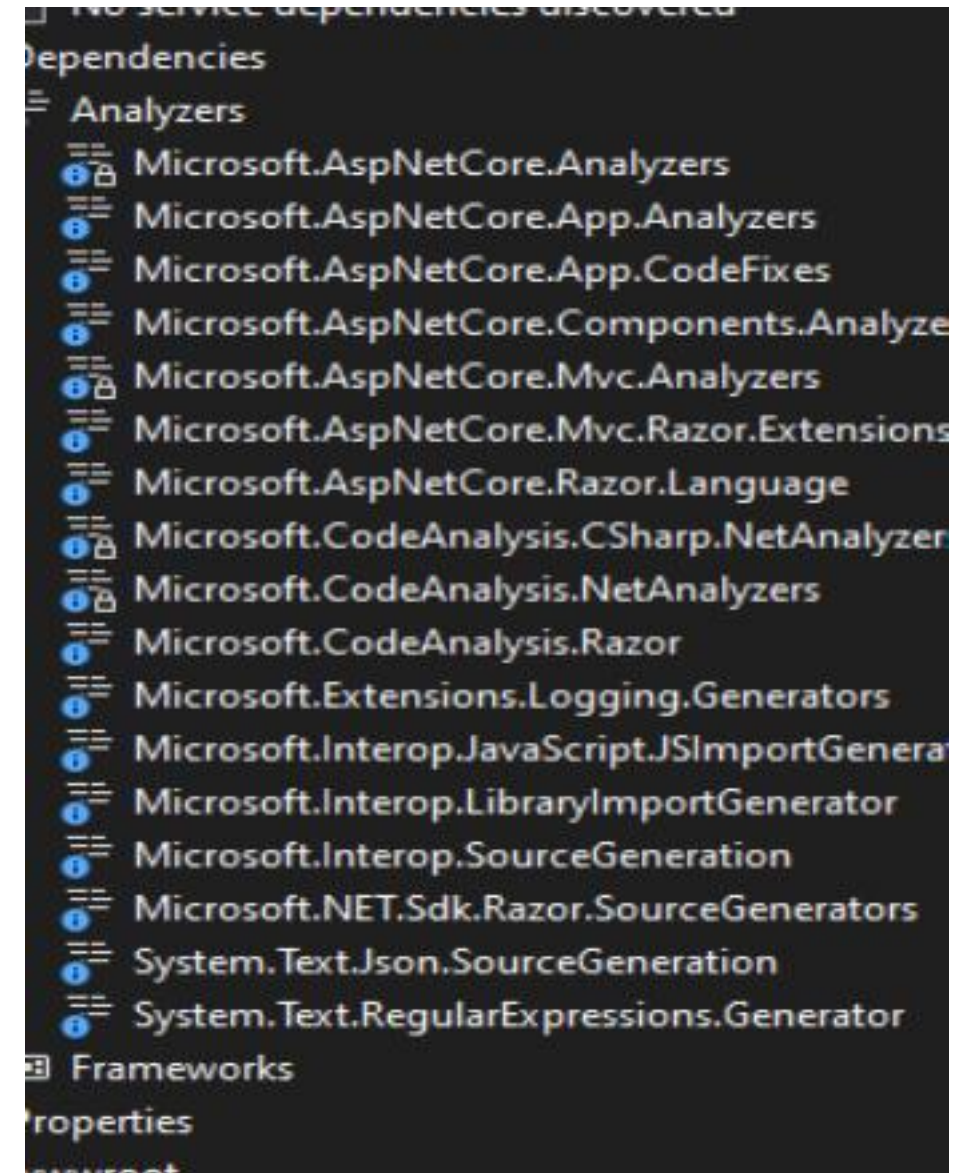


The screenshot shows a code editor window with a tab labeled "CorewebApp" and a sub-tab "CorewebApp: Overview". The XML content is displayed with a tree view on the left. The XML structure is as follows:

```
<Project Sdk="Microsoft.NET.Sdk.Web">  
  <PropertyGroup>  
    <TargetFramework>net7.0</TargetFramework>  
    <Nullable>enable</Nullable>  
    <ImplicitUsings>enable</ImplicitUsings>  
  </PropertyGroup>  
</Project>
```

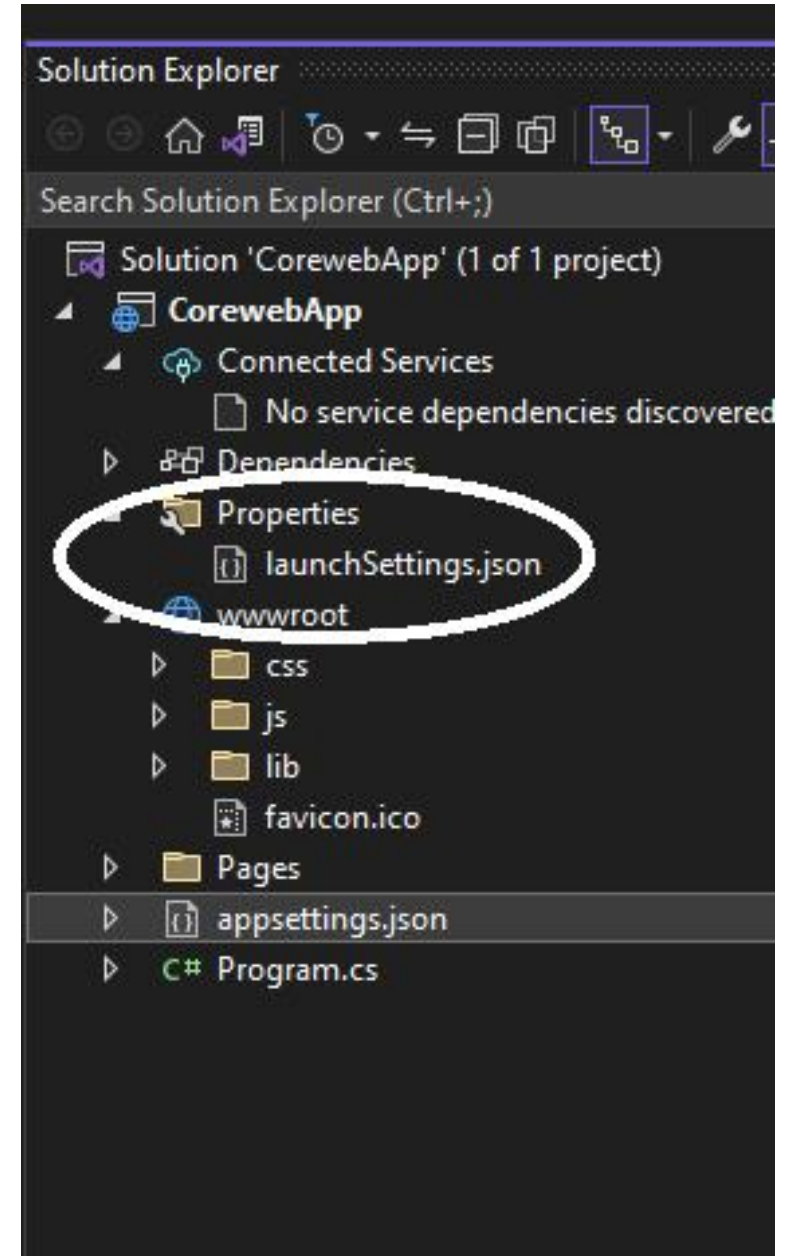

Dependencies

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



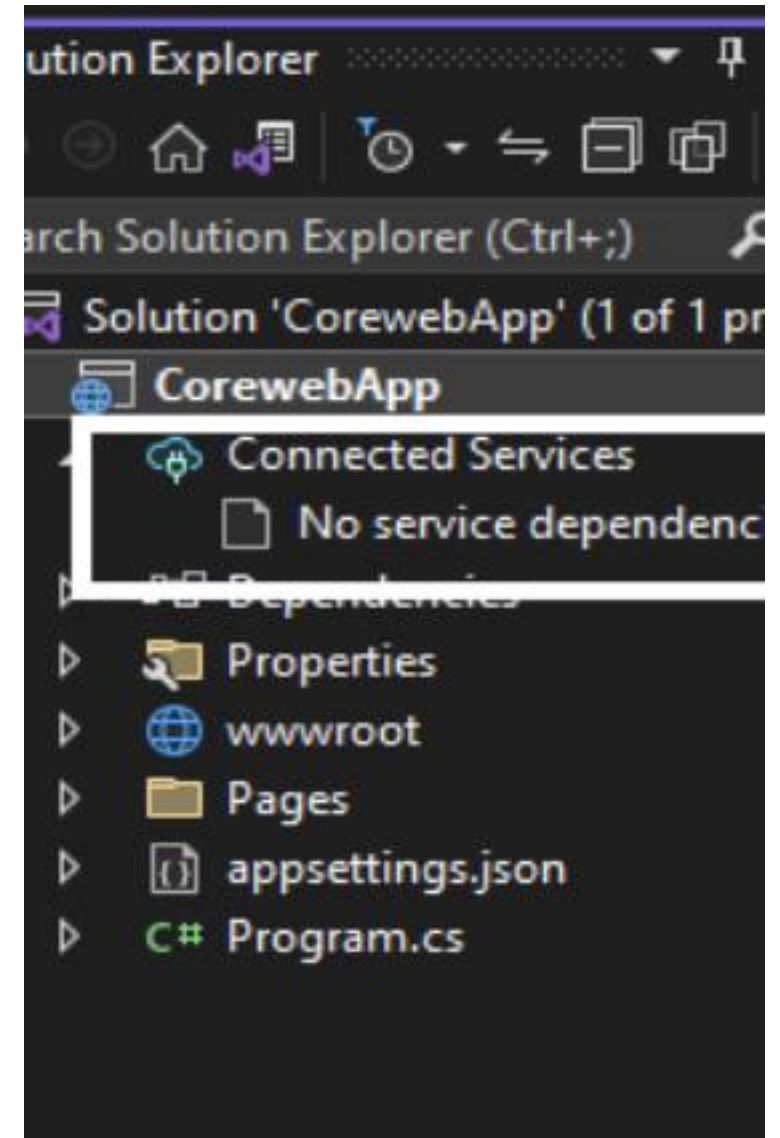
Properties

Properties folder contains a **launchSettings.json** file, which containing all the information required to lunch the application. Configuration details about what action to perform when the application is executed and contains details like IIS settings, application URLs, authentication, SSL port details, etc.



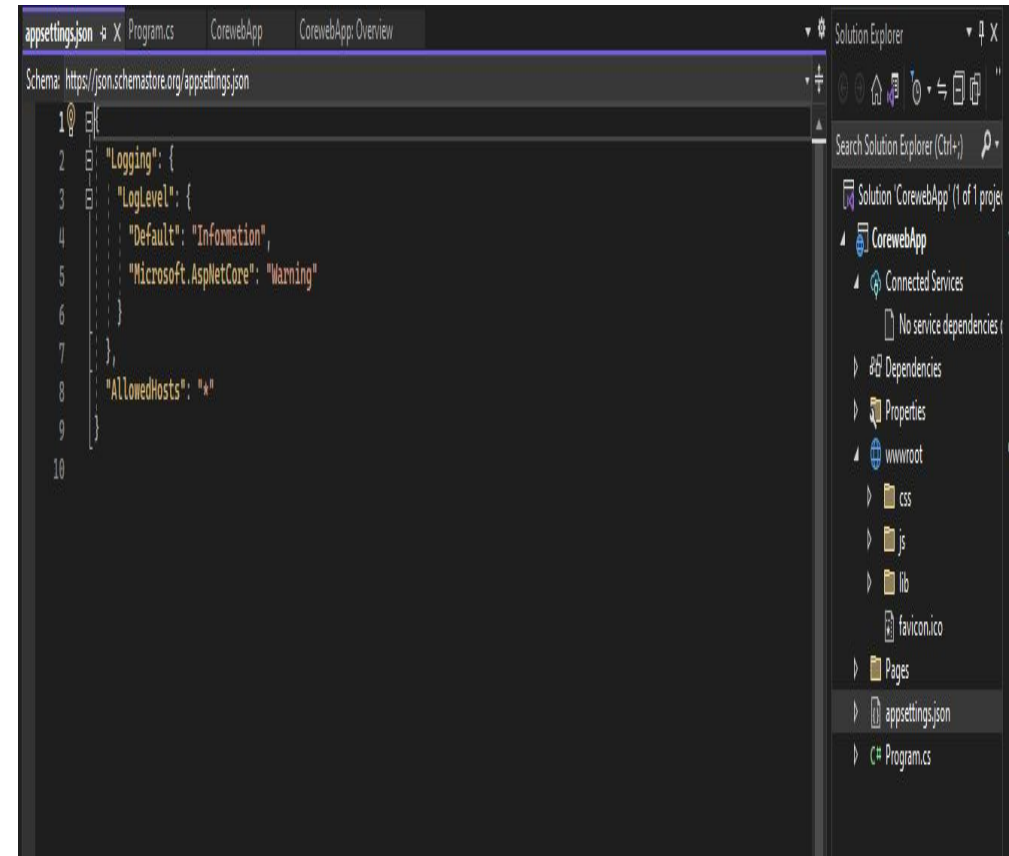
Connected Services

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.



appsettings.json

This file contains the application settings, for example, configuration details like logging details, database connection details.



The screenshot shows the Visual Studio IDE with the `appsettings.json` file open in the editor. The file content is as follows:

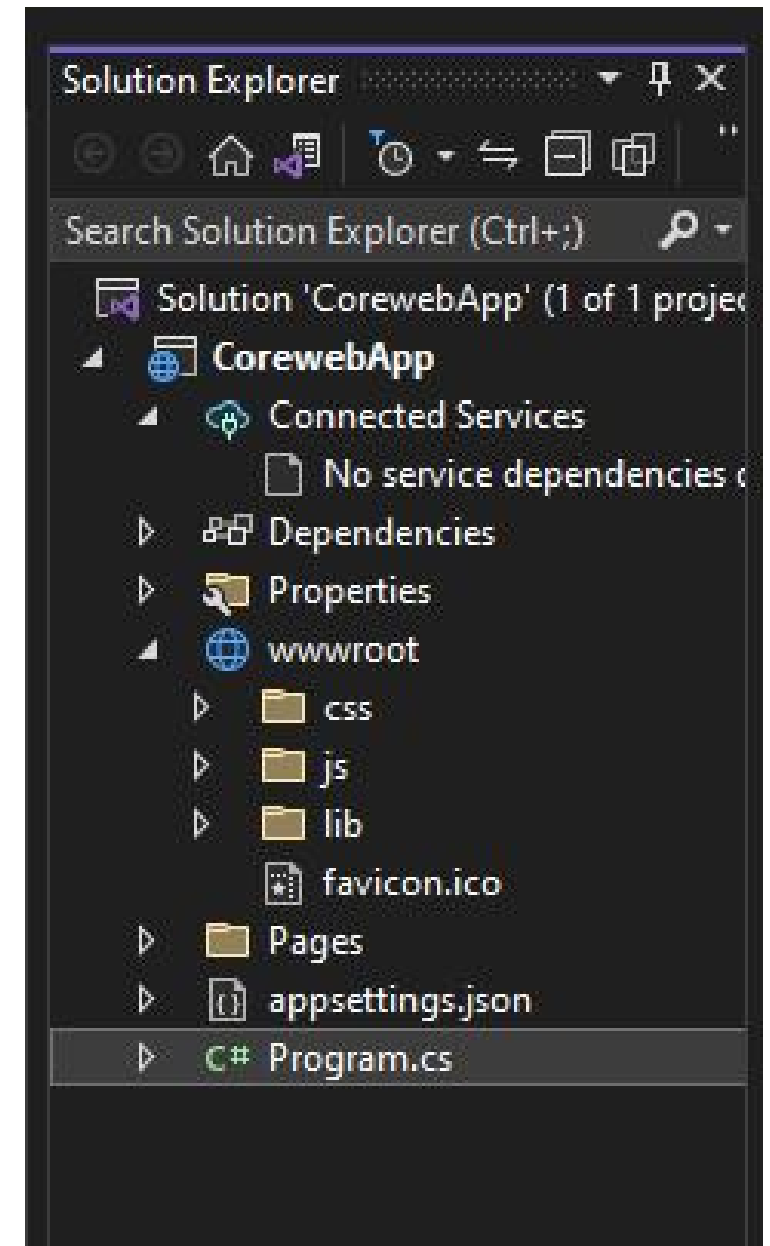
```
1 {  
2   "Logging": {  
3     "LogLevel": {  
4       "Default": "Information",  
5       "Microsoft.AspNetCore": "Warning"  
6     }  
7   },  
8   "AllowedHosts": "*"   
9 }  
10
```

The Solution Explorer on the right shows the project structure for `CorewebApp`, including `wwwroot`, `css`, `js`, `lib`, `faviconico`, `Pages`, `appsettings.json`, and `Program.cs`.

wwwroot

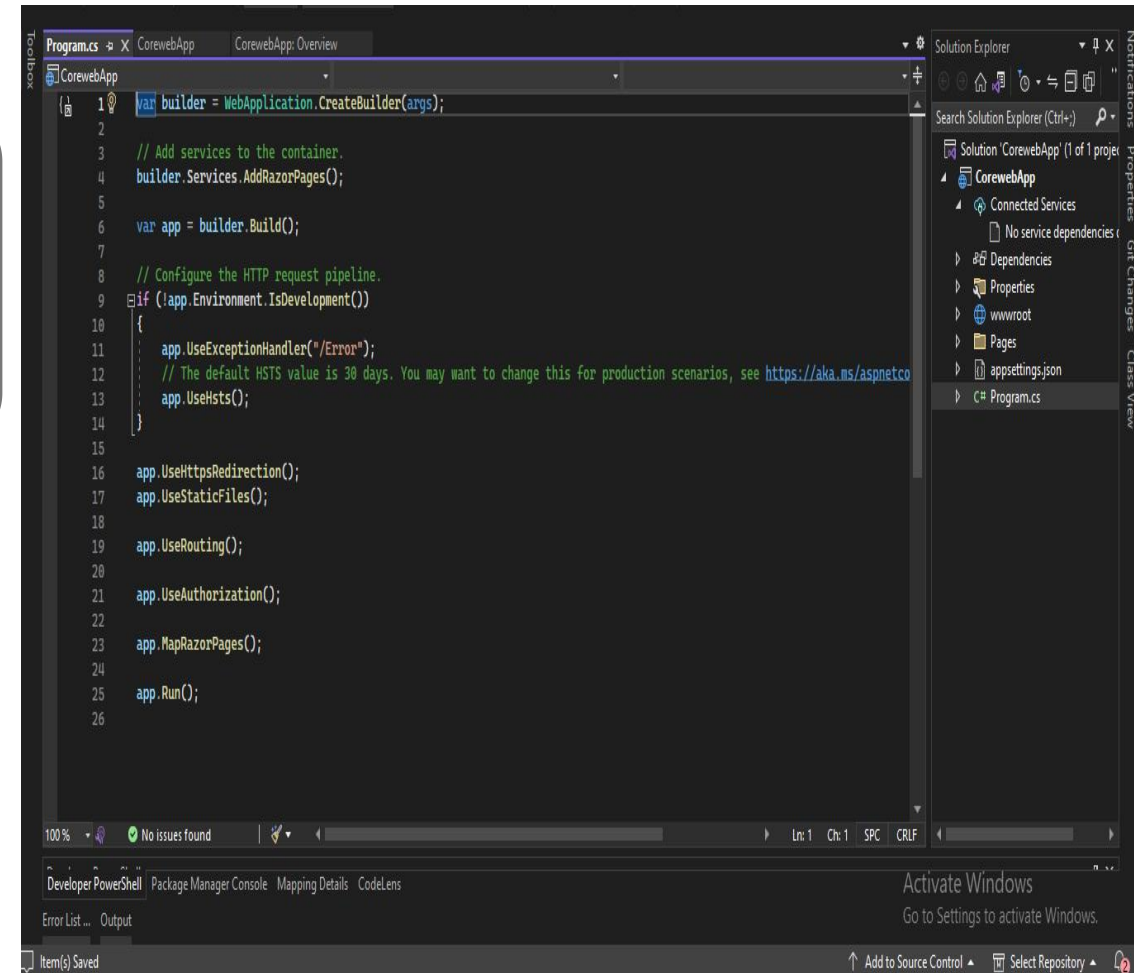
This is the webroot folder and all the static files required by the project are stored and served from here. The webroot folder contains a sub-folder to categorize the static file types, like all the Cascading Stylesheet files, are stored in the **CSS folder**, all the javascript files are stored in 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.



program.cs

This class is the entry point of the web application. It builds the host and executes the run method.



The screenshot shows the Visual Studio IDE with the 'Program.cs' file open in the editor. The code is for a web application project named 'CorewebApp'. The code starts with creating a builder, adding services, building the application, and configuring the HTTP request pipeline. It includes a conditional block for development environments to use exception handlers and HSTS. The code ends with calling 'Run()' on the application.

```
1 var builder = WebApplication.CreateBuilder(args);
2
3 // Add services to the container.
4 builder.Services.AddRazorPages();
5
6 var app = builder.Build();
7
8 // Configure the HTTP request pipeline.
9 if (!app.Environment.IsDevelopment())
10 {
11     app.UseExceptionHandler("/Error");
12     // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts
13     app.UseHsts();
14 }
15
16 app.UseHttpsRedirection();
17 app.UseStaticFiles();
18
19 app.UseRouting();
20
21 app.UseAuthorization();
22
23 app.MapRazorPages();
24
25 app.Run();
26
```

The Solution Explorer on the right shows the project structure, including 'Connected Services', 'Dependencies', 'Properties', 'wwwroot', 'Pages', 'appsettings.json', and 'Program.cs'. The status bar at the bottom indicates 'Item(s) Saved' and 'No issues found'.

Thank you

ASP .NET Core MVC

- ASP.NET MVC renders UI on the server and uses a Model-View-Controller (MVC) architectural pattern.
- The MVC pattern separates an app into three main groups of components: Models, Views, and Controllers.
- User requests are routed to a controller. The controller is responsible for working with the model to perform user actions or retrieve results of queries.
- The controller chooses the view to display to the user, and provides it with any model data it requires. Support for Razor Pages is built on ASP.NET Core MVC.

