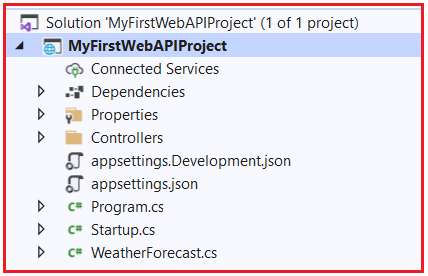
**ASP.NET Core Web API Files and Folders**

In this article, I am going to discuss the **Files and Folders which are created by default when we create a new ASP.NET Core Web API** Application. Please read our previous article where we discussed[**How to Create, Build, and Run ASP.NET Core Web API**](https://dotnettutorials.net/lesson/asp-net-core-web-api-project-in-visual-studio-2019/) in Visual Studio 2019.

**Folders and Files in ASP.NET Core Web API:**

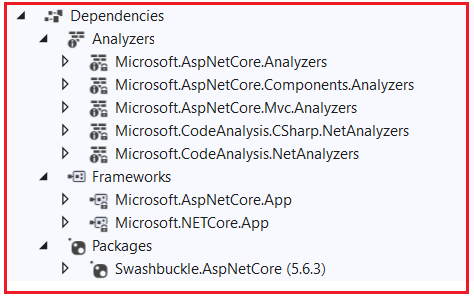
In our previous article, we created the first ASP.NET Core Web API Project and we also see that the project is created with the following files and folder structure.



Now let us proceed and understand the above files and folders in detail.

**Dependencies:**

The Dependencies contains all the packages and SDKs that are installed in this project. Further, if you expand, it contains three files (Analyzers, Frameworks, and Packages) as shown in the below image.



**Packages:**

At the moment the Packages folder contains one package i.e. Swashbuckle.AspNetCore. If you remember when we run the application, then it opens the swagger page. This package is basically for Swagger. Later if you add any new packages to your project, then that package will be shown here.

**Frameworks:**

The framework contains two folders i.e. **Microsoft.AspNetCore.App** and **Microsoft.NETCore.App**. All the packages that are required to run ASP.NET Core Application will be there inside Microsoft.AspNetCore.App file. The ASP.NET Core is a framework that is written on top of the .NET Core. So, all those packages that are required for the .NET core are available inside Microsoft.NETCore.App. So, the point that you need to remember is .NET Core and ASP.NET Core are two different things. All the packages which are specific to ASP.NET Core will reside inside **Microsoft.AspNetCore.App** and all the package which are specific to .NET Core will reside inside the **Microsoft.NETCore.App.**

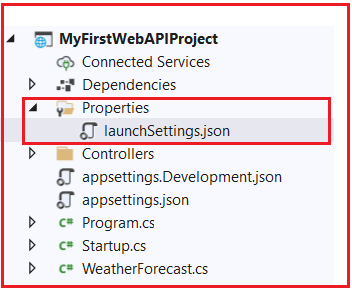
**Analyzers:**

The ASP.NET Core 2.2 and later provides analyzer packages intended for use with Web API projects. The analyzers will work with controllers annotated with ApiControllerAttribute while building on web API conventions. The analyzers package notifies you of any controller action that:

1. Returns an undeclared status code.
2. Returns an undeclared success result.
3. Documents a status code that isn’t returned.
4. Includes an explicit model validation check.

**Properties:**

The Properties Folder in ASP.NET Core Web Application by default contains one JSON file called as launchsettings.json file as shown in the below image.



The launchsettings.json file contains some settings that are going to be used by .NET Core Framework when we run the application either from Visual Studio or by using .NET Core CLI. Another point that you need to keep in mind, the launchSettings.json file is only used within the local development machine. So, this file is not required when we publishing our ASP.NET Core Web API application into the production server. Now, open the launchSettings.json file, by default you will see the following settings.



If you remember, when we are running our application using .NET Core CLI, we are getting two URLs (**https://localhost:5001** and **http://localhost:5000**). These two URLs are coming from this MyFirstWebAPIProject Profile.

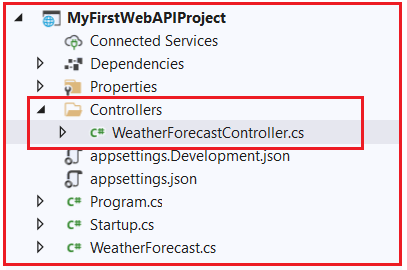
Further, if you remember when we run our application using Visual Studio, the application run on port number 44395. For IIS Express Profile, it uses **iissettings**. If you notice the iisSettings, the applicationUrl is **http://localhost:63044** and sslPort as **44395**. That means if you are running the application using the HTTP protocol, then it will use the **63044** port number and if you run the application using HTTPS protocol then it will use the 44395 port number.

**Note**: If you are running your application from the visual studio then IIS Express Profile will be used (for HTTP the port number will be 63044 and for HTTPS the port number will be 44395). On the other hand, if you are running your application using .NET Core CLI, then MyFirstWebAPIProject profile will be used which is nothing but using Kestrel Web Server and for HTTP protocol it uses the port number 5000 and for HTTPS protocol it uses the port number 5001.

[**Learn More about the launchsettings.json file.**](https://dotnettutorials.net/lesson/asp-net-core-launchsettings-json-file/)

**Controllers:**

The ASP.NET Core Web API is a controller-based approach. All the controllers of your ASP.NET Core Web API Application should and must reside inside the Controllers folder. Here, by default one controller is there inside the Controllers folder as shown in the below image.



If you open the WeatherForecastController file, then you will find the following code in it. The Controller class us inherited from the ControllerBase class and decorated with ApiController and Route attribute. We will discuss all these things in our upcoming article.

**using** *Microsoft.AspNetCore.Mvc;*

**using** *Microsoft.Extensions.Logging;*

**using** *System;*

**using** *System.Collections.Generic;*

**using** *System.Linq;*

**using** *System.Threading.Tasks;*

**namespace** *MyFirstWebAPIProject.Controllers*

**{**

**[**ApiController**]**

**[**Route**(**"[controller]"**)]**

**public** **class** WeatherForecastController : ControllerBase

**{**

**private** **static** **readonly** string**[]** Summaries = new**[]**

**{**

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

**}**;

**private** **readonly** ILogger**<**WeatherForecastController**>** \_logger;

**public** WeatherForecastController**(**ILogger**<**WeatherForecastController**>** logger**)**

**{**

\_logger = logger;

**}**

**[**HttpGet**]**

**public** IEnumerable**<**WeatherForecast**>** Get**()**

**{**

var rng = new Random**()**;

**return** Enumerable.Range**(**1, 5**)**.Select**(**index =**>** new WeatherForecast

**{**

Date = DateTime.Now.AddDays**(**index**)**,

TemperatureC = rng.Next**(**-20, 55**)**,

Summary = Summaries**[**rng.Next**(**Summaries.Length**)]**

**})**

.ToArray**()**;

**}**

**}**

**}**

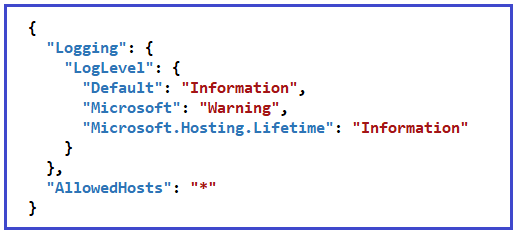
As you can in the above image, we have one Get method which is decorated with the HttpGet attribute. And when we call the following URL using any client (Swagger, Postman, and Browser), this is the method going to return the data. We will discuss Controller in detail in our upcoming articles.

**https://localhost:44395/WeatherForecast**

**appsettings.json file:**

The next file that we are going to discuss is the appsettings.json file. This is the same as web.config or app.config of our traditional .NET Application. The appsettings.json file is the application configuration file in ASP.NET Core Web Application used to store the configuration settings such as database connections strings, any application scope global variables, etc.

If you open the appsettings.json file, then you see the following code by default which is created by .NET Core Framework when we created the ASP.NET Core Web API Application.



[**Learn More about the appsettings.json file.**](https://dotnettutorials.net/lesson/asp-net-core-appsettings-json-file/)

**appsettings.Development.json:**

If you want to configure some settings based on the environments then you can do such settings in appsettings.{Environment}.json file. You can create n number of environments like development, staging, production, etc. I

If you set some settings in the appsettings.Development.json file, then such settings can only be used in the development environment, can not be used in other environments. If you open the appsettings.Development.json file, then you will get the following code in it. In our upcoming articles, we will see how to create and use different environment appSettings file in ASP.NET Core Web API Application.

**{**

"Logging": **{**

"LogLevel": **{**

"Default": "Information"**,**

"Microsoft": "Warning"**,**

"Microsoft.Hosting.Lifetime": "Information"

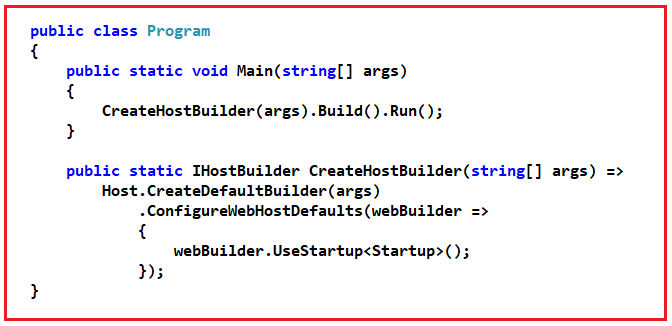
**}**

**}**

**}**

**Program.cs class file:**

The Program.cs class file of our ASP.NET Core Webb API Application contains the following code.



As shown in the above Program class code, it has a **public static void Main()**method. The Main method is the entry point of our Application.

Each ASp.NET Core Web API Application initially starts as a Console Application and the Main() method is the entry point to the application. So, when we execute the ASP.NET Core Web API application, it first looks for the Main() methodand this is the method from where the execution starts for the application. The Main() method then configures ASP.NET Coreand starts it. At this point, the application becomes an ASP.NET Core Web API application.

[**Learn More about ASP.NET Core Main Method.**](https://dotnettutorials.net/lesson/asp-net-core-main-method/)

**Startup.cs class file:**

The Startup class is like the Global.asax file of our traditional .NET application. As the name suggests, it is executed when the application starts. You will find the following code in your Startup class.

**namespace** *MyFirstWebAPIProject*

**{**

**public** **class** Startup

**{**

**public** Startup**(**IConfiguration configuration**)**

**{**

Configuration = configuration;

**}**

**public** IConfiguration Configuration **{** **get**; **}**

// This method gets called by the runtime. Use this method to add services to the container.

**public** **void** ConfigureServices**(**IServiceCollection services**)**

**{**

services.AddControllers**()**;

services.AddSwaggerGen**(**c =**>**

**{**

c.SwaggerDoc**(**"v1", new OpenApiInfo **{** Title = "MyFirstWebAPIProject", Version = "v1" **})**;

**})**;

**}**

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

**public** **void** Configure**(**IApplicationBuilder app, IWebHostEnvironment env**)**

**{**

**if** **(**env.IsDevelopment**())**

**{**

app.UseDeveloperExceptionPage**()**;

app.UseSwagger**()**;

app.UseSwaggerUI**(**c =**>** c.SwaggerEndpoint**(**"/swagger/v1/swagger.json", "MyFirstWebAPIProject v1"**))**;

**}**

app.UseHttpsRedirection**()**;

app.UseRouting**()**;

app.UseAuthorization**()**;

app.UseEndpoints**(**endpoints =**>**

**{**

endpoints.MapControllers**()**;

**})**;

**}**

**}**

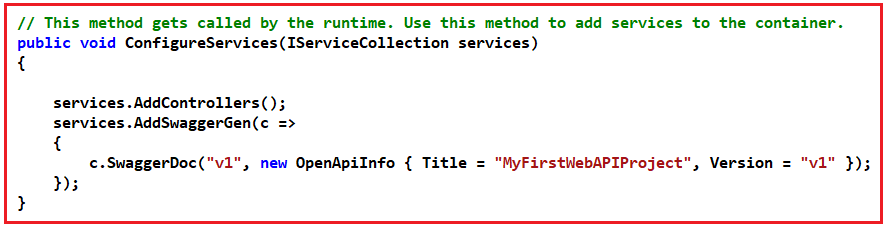
**}**

As you can see in the above code, the Startup class includes two public methods: ConfigureServices and Configure.

**ConfigureServices() method:**

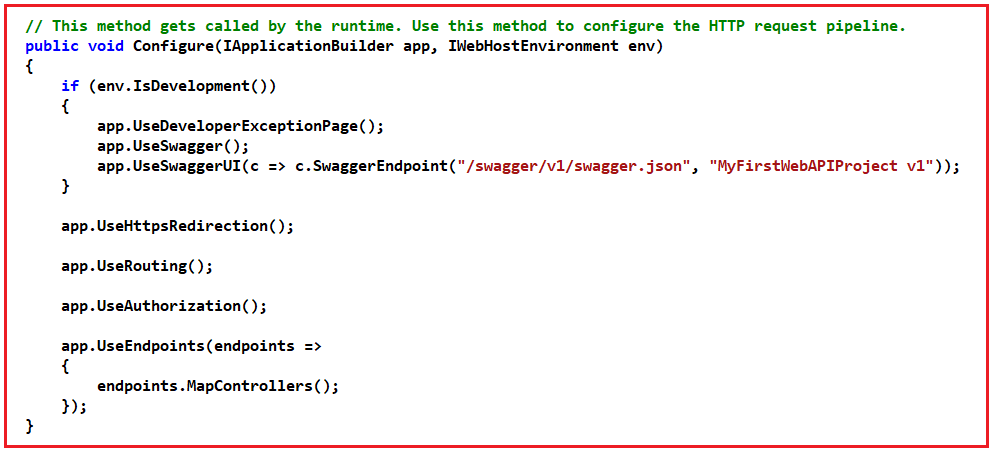
The ConfigureServices method of the Startup class is the place where we can register our dependent classes with the built-in IoC container. Once we register the dependent classes, they can be used anywhere within the application. The ConfigureServices method includes the IServiceCollection parameter to register services to the IoC container.

If you notice, currently it adds two services i.e. AddControllers and AddSwaggerGen to the build-in IoC Container as shown in the below image.



**Configure() method:**

The Configure method of the Startup class is the place where we configure the application request pipeline using the IApplicationBuilder instance that is provided by the built-in IoC container. ASP.NET Core introduced the middleware components to define a request pipeline, which will be executed on every request. If you look at the Configure method, it registered UseDeveloperExceptionPage, UseSwagger, UseSwaggerUI, UseHttpsRedirection, UseRouting, UseAuthorization, and UseEndpoints middleware components to the request processing pipeline as shown in the below image. We will discuss each middleware component in detail in our upcoming articles.



[**Learn more about the ASP.NET Core Startup class.**](https://dotnettutorials.net/lesson/asp-net-core-satrtup-class/)

**WeatherForecast.cs class file:**

This is the model class and you can find the following code in it.

**public** **class** WeatherForecast

**{**

**public** DateTime Date **{** **get**; **set**; **}**

**public** **int** TemperatureC **{** **get**; **set**; **}**

**public** **int** TemperatureF =**>** 32 + **(int)(**TemperatureC / 0.5556**)**;

**public** string Summary **{** **get**; **set**; **}**

**}**

**Note:** The Model can be placed anywhere. It can be placed inside the Models folder. Directly at the project level. Even they can be crated as separate class library projects.