Hello World API Backend Guide

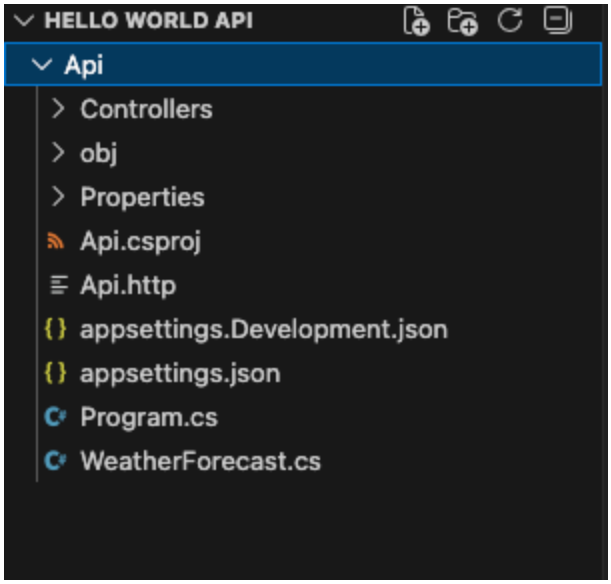
`Step 1`: Create a new folder on your computer and open it in VS Code

`Step 2`: Open up a terminal and type this command “dotnet new webapi —use-controllers -o Api”

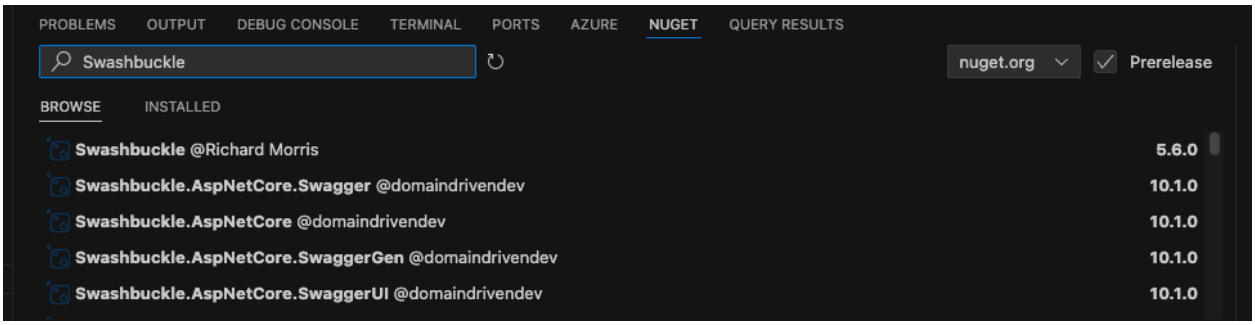
> Note: after the “-o” in the terminal command, you can choose to name it differently as that is just creating a folder to store everything in.

```
Hello World Api % dotnet new webapi --use-controllers -o Api
```

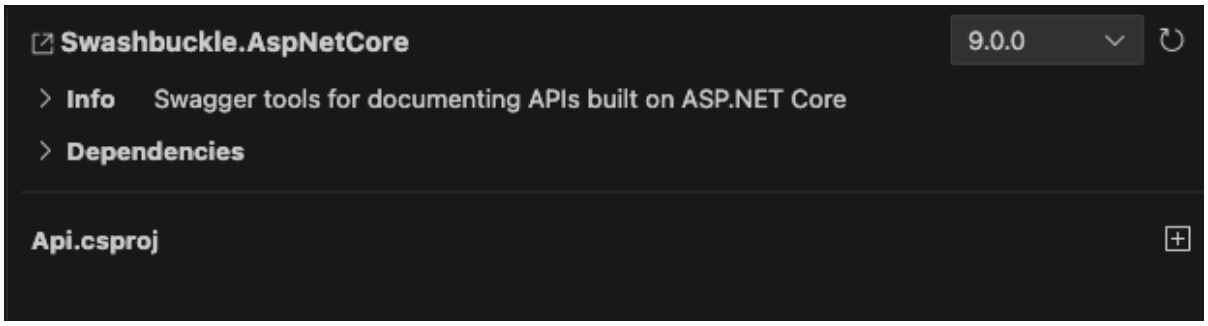
2.1 - This will create an Api and populate the files in the file explorer in VS Code



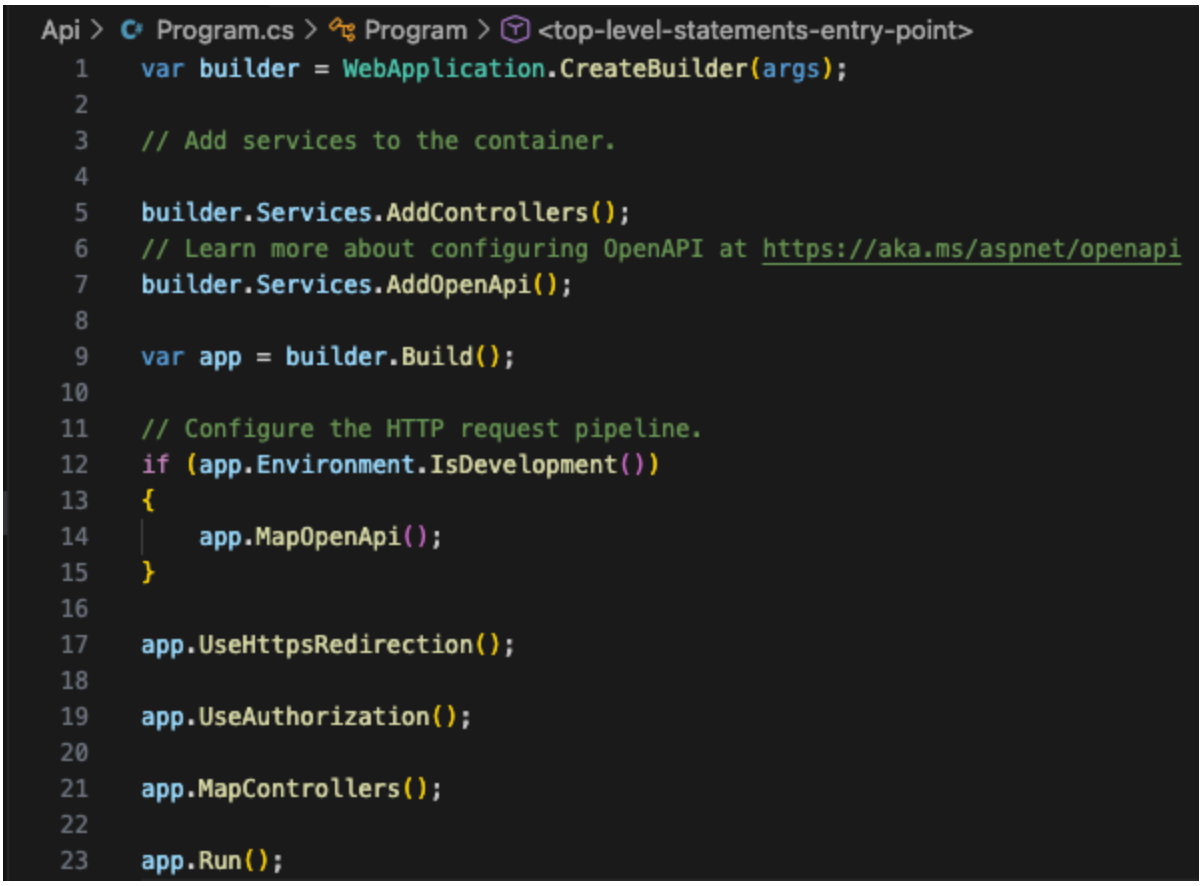
`Step 3`: With the terminal still open, navigate to `NUGET` and search for “Swashbuckle”



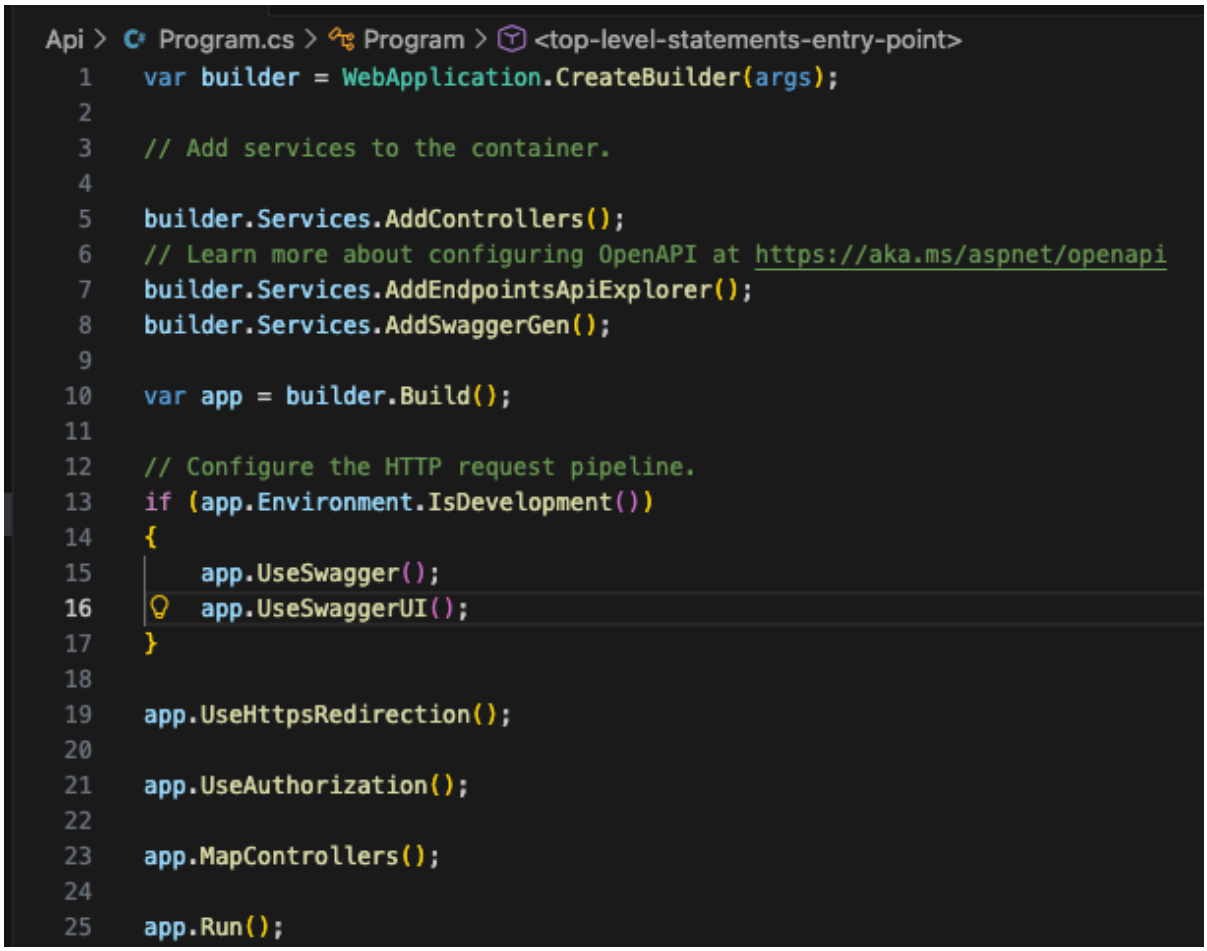
3.1 - We are going to download 3 files by making sure these are all on `version 9.0`, “Swashbuckle.AspNetCore”, “Swashbuckle.AspNetCore.SwaggerGen”, and “Swashbuckle.AspNetCore.SwaggerUI” and click the “+” icon to download, for each file.



`Step 4`: Once those are downloaded, go to the Program.cs file and delete these lines 7, and 14

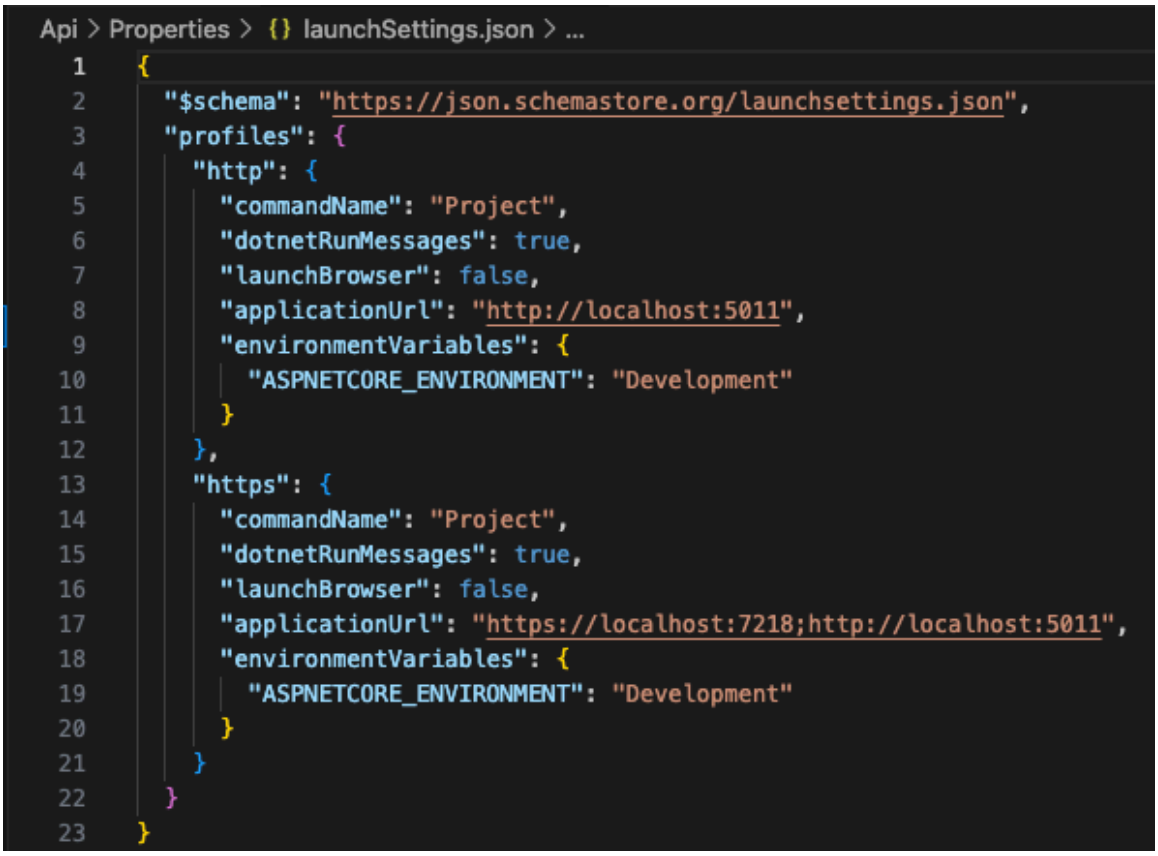


4.1 - Replace those lines with these shown below. This will allow us to utilize swagger UI.



4.2 - Now we must make a few changes in another file. Navigate to the launchSettings.json file under Properties folder of your API. In this file we will be editing values and adding two lines so we can use Swagger.

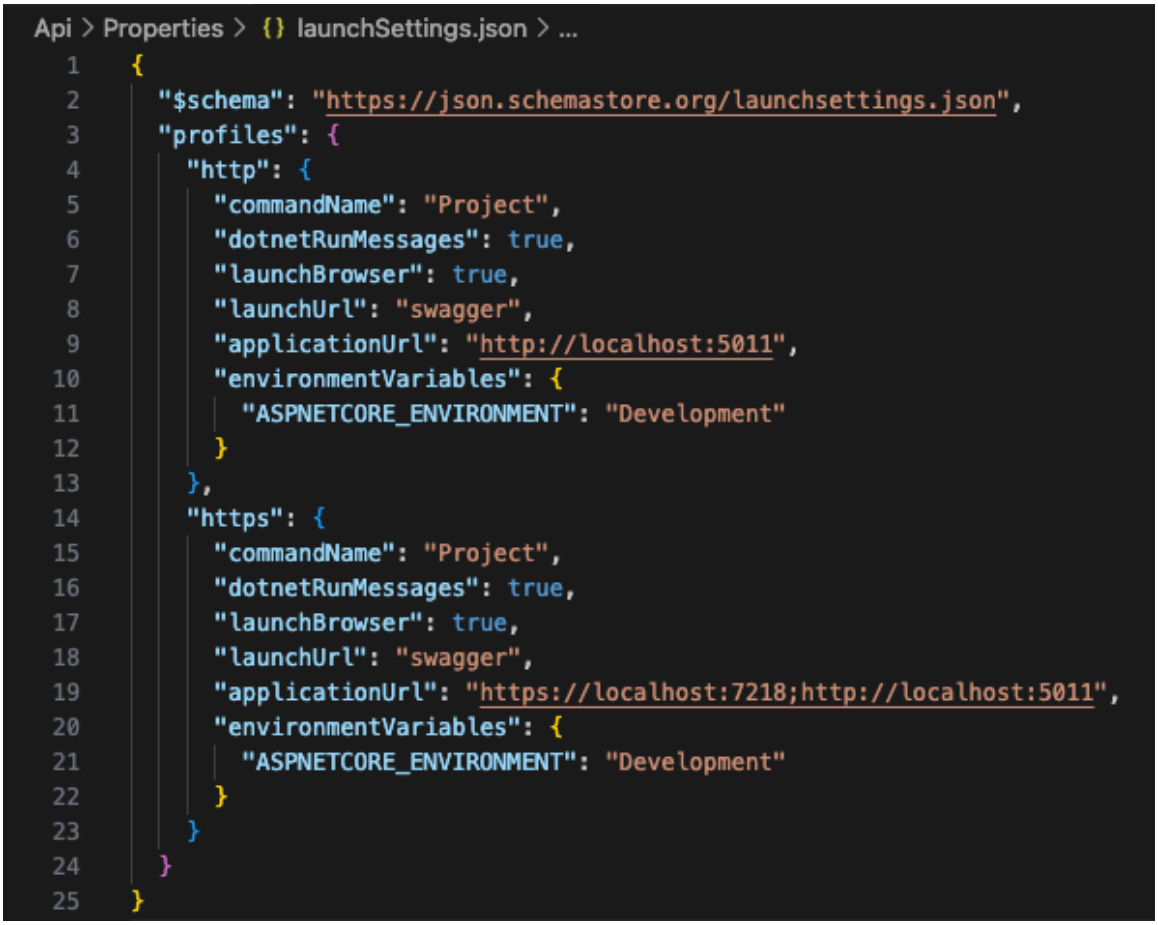
This is what the launchSettings.json looks like (*before the changes*)



4.3 - Edit lines 7 and 16 and change “false” to “true”, and at the end of both lines hit ENTER ⇒ and type `“launchURL”: “swagger”,`

This is what launchSettings.json should look like after the changes.

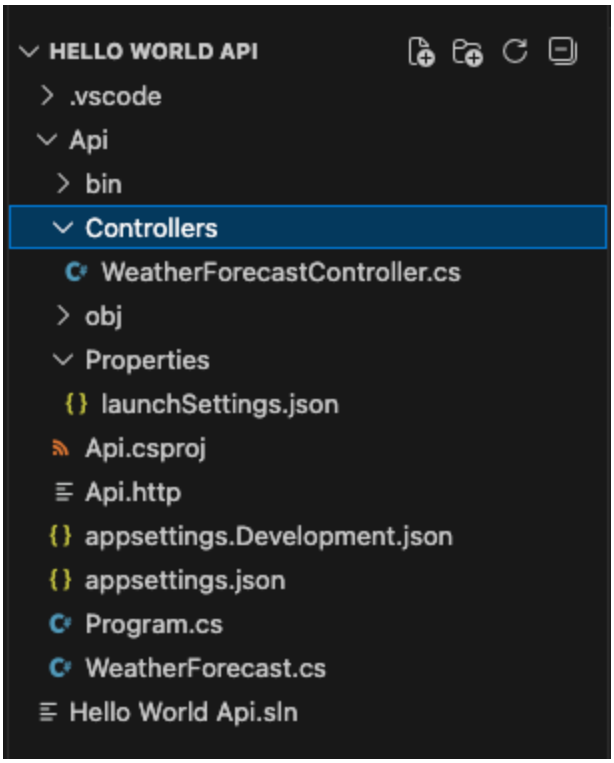
Api > Properties > {} launchSettings.json > ...



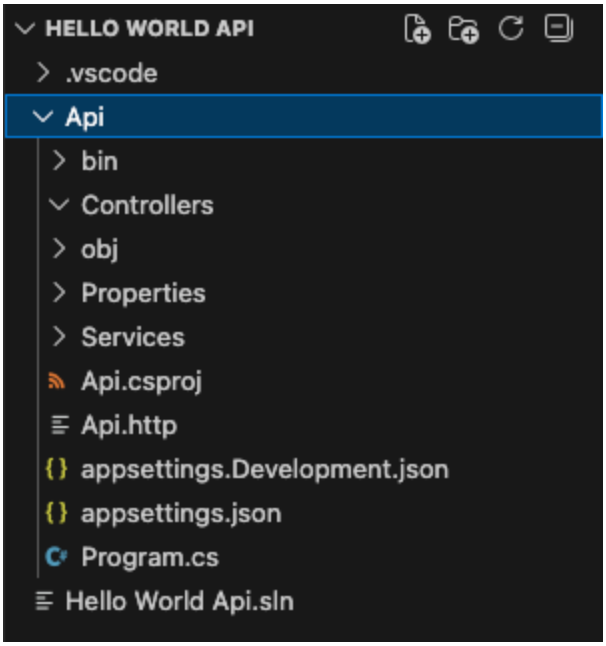
```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "http://localhost:5011",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "https://localhost:7218;http://localhost:5011",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

`Step 5`: Now we can start working in the file explorer. Delete the `WeatherForecast.cs` file and the `WeatherForecastController.cs` file (found under the “Controllers” folder)

> Note: These two files are from the template made from the command we typed from Step 1.



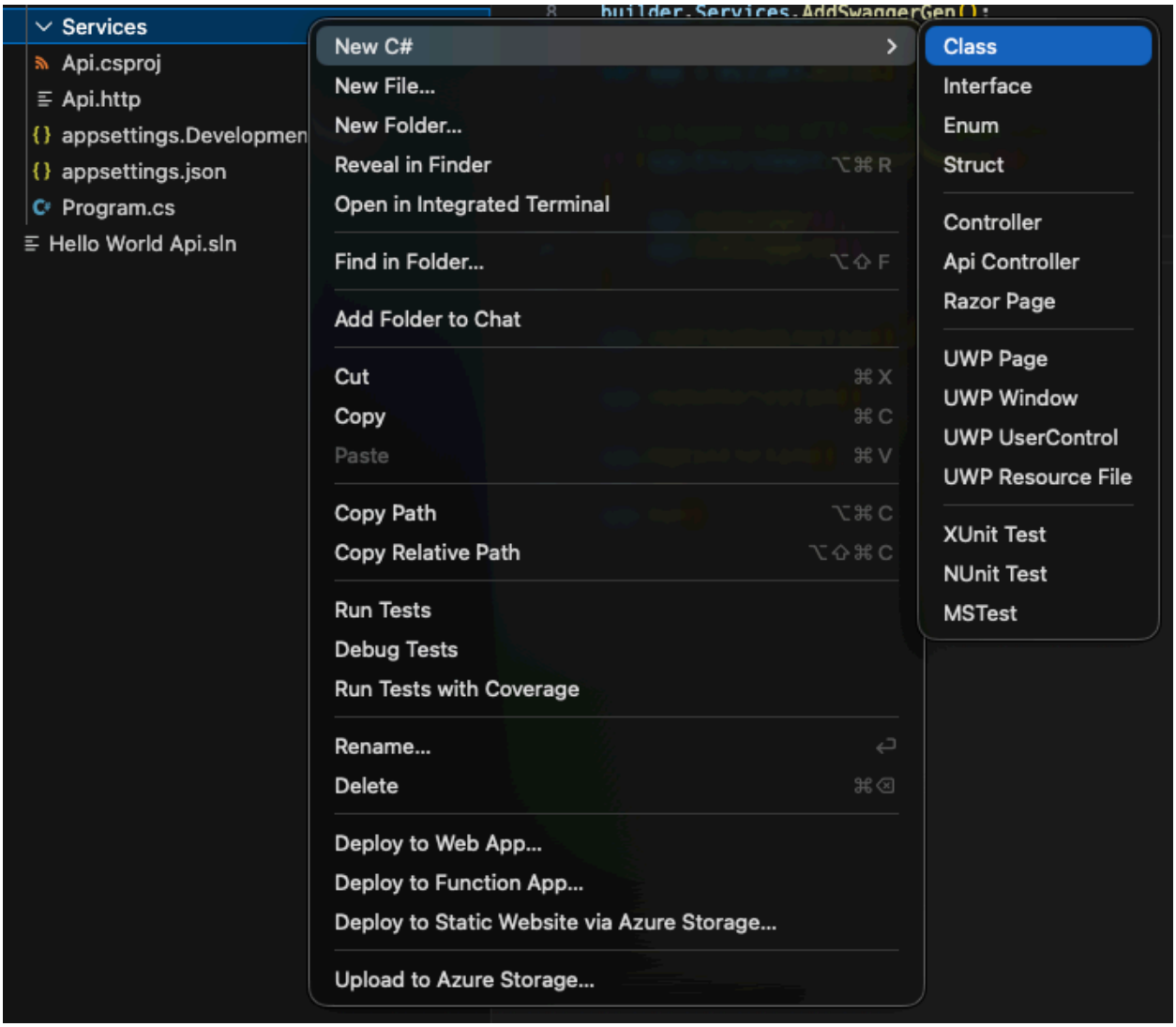
5.1 - Once you have deleted those files, click on the Api folder and make a new folder and title it “Services”



`Step 6`: Now we can start setting up our Api for coding. We’ll need to create two files. Let’s start by creating a service file. Right click the services file and create a

new C# class file. See picture below!

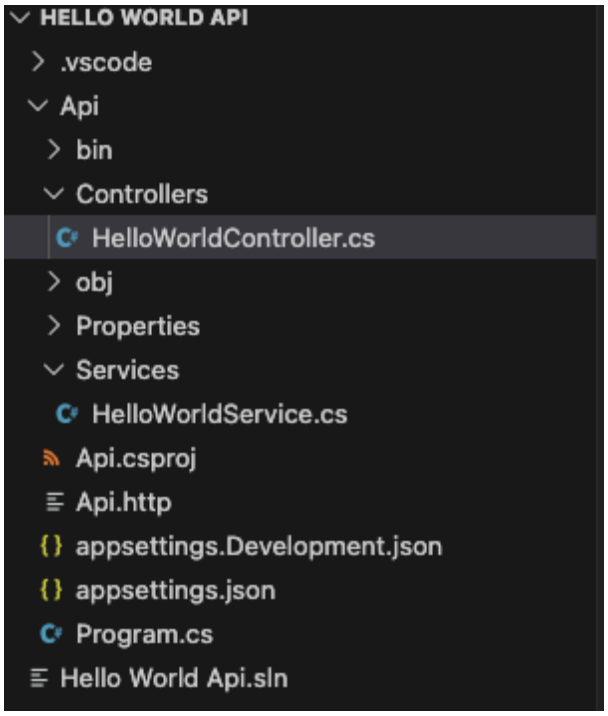
Note: You will be prompted to enter a name for the file, you can name it what you want and add “Service” at the end but for this guide we will name is “`HelloWorldService`”.



6.1 - Do the same for the Controllers folder but instead of a class file, we are going to create a new C# Api Controller file.

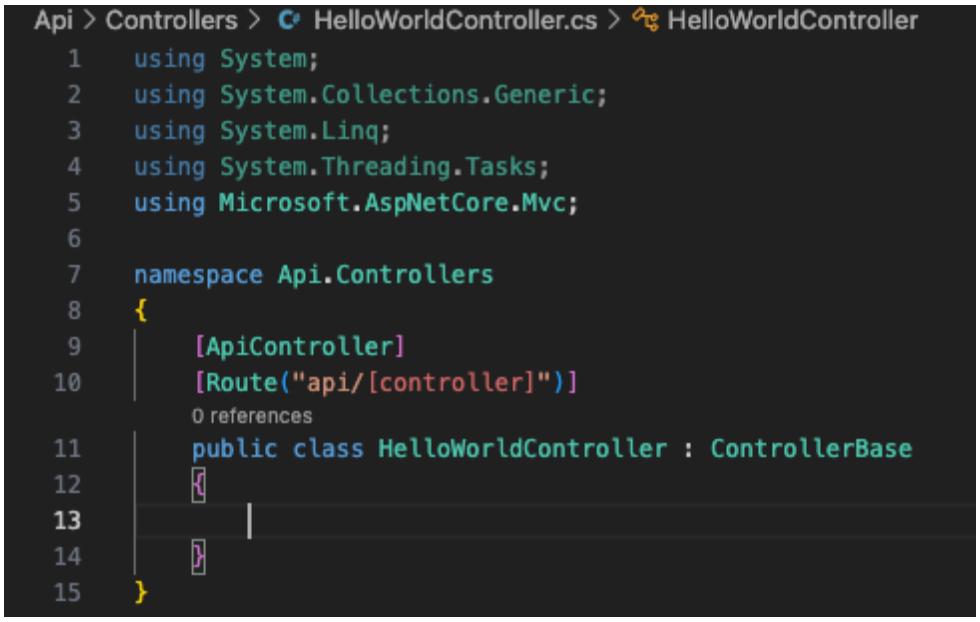
Note: You will be prompted again to enter a name for the file, enter a name for the file and add “Controller” at the end. For this guide, we will be naming it “`HelloWorldController`”

6.2 - You should now have two files, one in each folder. See picture below.

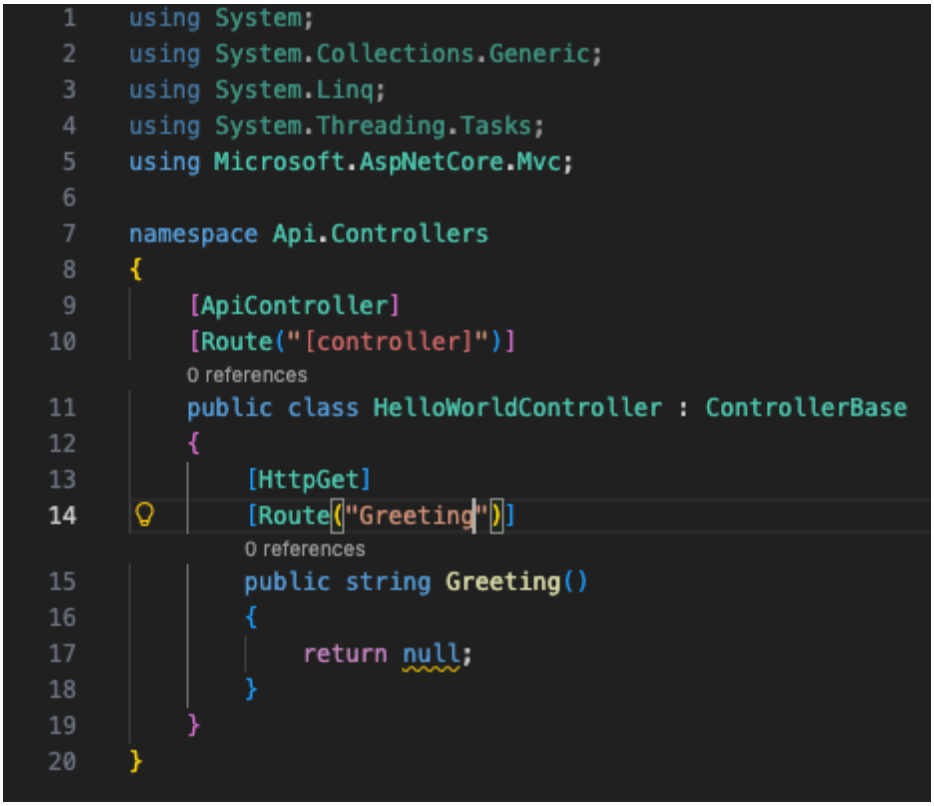


`Step 7`: Let's work in the newly created Controller file first. Yours should look like picture 7.1 below. Start by removing "`api/`" from line 10. On line 13, we will be adding a few lines of code that will allow us to call our api through an endpoint. See picture 7.2.

(Picture 7.1) Controller file before the additions.



(Picture 7.2) Controller file with an endpoint calling the api and a function titled Greeting.



7.1 - We can leave our Controller file like this for the time being as we will being coming back to connect with our service file through dependency injection.

`Step 8`: Go to the service file we created from step 6 as we will start working in this file now. This is where your code will live, here we can just make a simple function titled Greeting that will return the string “Hello World”.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5
6   namespace Api.Services
7   {
        0 references
8       public class HelloWorldService
9       {
            0 references
10          public string Greeting()
11          {
12              return "Hello World";
13          }
14      }
15  }
```

`Step 9`: Now that we have our service file ready, we can go back to the Controller file to connect it. See picture below!!! On line 14, we are making a variable for our service file that we will be assigning a value to in our constructor on line 18. With that we can edit our return line on line 25 and change that from “null” to the service file variable and the function from the our service file.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Api.Services;
using Microsoft.AspNetCore.Mvc;

namespace Api.Controllers
{
    [ApiController]
    [Route("[controller]")]
    1 reference
    public class HelloWorldController : ControllerBase
    {
        2 references
        private readonly HelloWorldService _HelloWorldService;

        0 references
        public HelloWorldController(HelloWorldService helloWorldService)
        {
            _HelloWorldService = helloWorldService;
        }

        [HttpGet]
        [Route("Greeting")]
        0 references
        public string Greeting()
        {
            return _HelloWorldService.Greeting();
        }
    }
}
```

`Step 10`: Before we start to launch it, we need to add one more in the Program.cs file. We need to add scope so that our Api can utilize our service file. We do this by adding to the builder services, see picture below.

This is how yours should look like after adding scope to the service file.

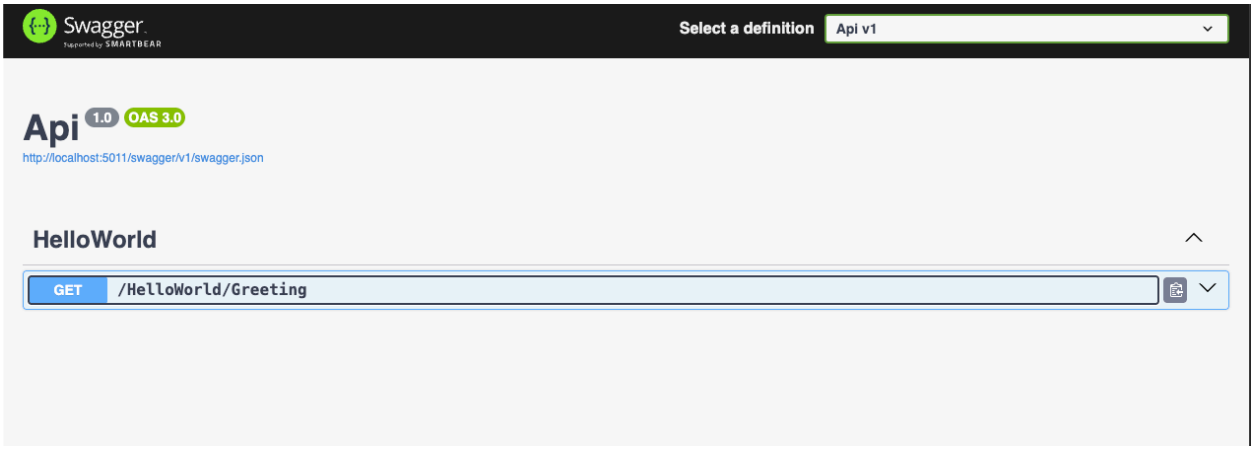
```
using Api.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddScoped<HelloWorldService>();

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();
```

`Step 11`: Now we can start to launch our API through Swagger. Open a terminal and type these commands one by one in order: `cd Api` ⇒ `dotnet watch run`

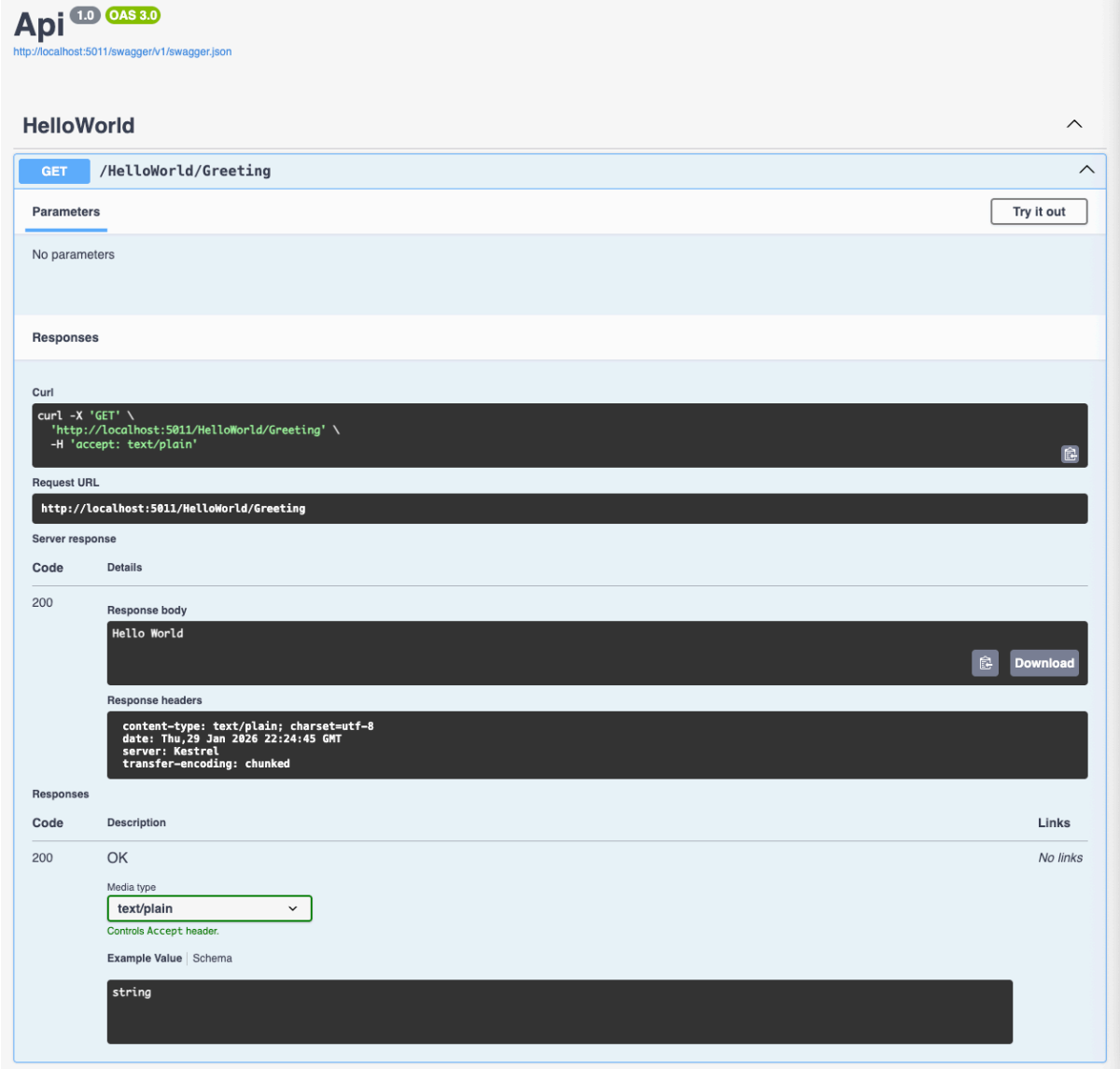
```
Hello World Api % cd Api
```

```
Api % dotnet watch run
```

`Step 12`: Running those commands should have opened a tab in your browser of Swagger with your newly created API.



12.1 - Click on the Get endpoint line to open it up and then click the “Try it out” button then hit “Execute” button to see your endpoint being called from your API


Api
1.0
OAS 3.0
http://localhost:5011/swagger/v1/swagger.json
HelloWorld
GET
/HelloWorld/Greeting
Parameters
Try it out
No parameters
Responses
Curl
curl -X 'GET' \
'http://localhost:5011/HelloWorld/Greeting' \
-H 'accept: text/plain'
Request URL
http://localhost:5011/HelloWorld/Greeting
Server response
Code
Details
200
Response body
Hello World
Download
Response headers
content-type: text/plain; charset=utf-8
date: Thu,29 Jan 2026 22:24:45 GMT
server: Kestrel
transfer-encoding: chunked
Responses
Code
Description
Links
200
OK
No links
Media type
text/plain
Controls Accept header.
Example Value
Schema
string