Nicolas Gonzaba

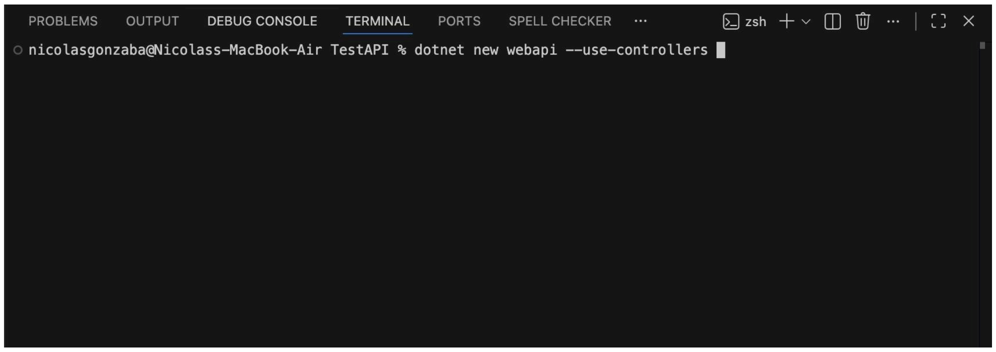
Ken Martinez

30 January 2026

API Walkthrough

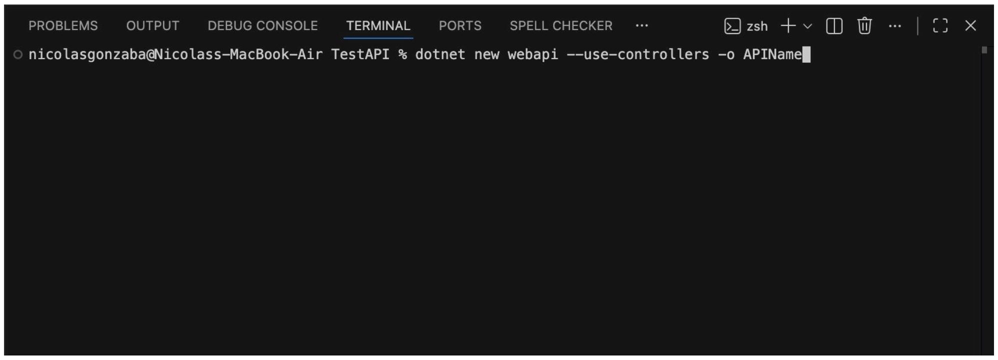
If you want to create an endpoint for an API you must first open the terminal on your project and enter dotnet new webapi --use-controllers into your terminal.
OPTIONAL: You can also put -o and put your desired Folder name after if you want your API to be in a separate Folder. This will matter if you want multiple folders in your project or you want to have your frontend and backend all in one project but for the sake of this test it wont.
NOTE: If you choose to have your API in a folder it will change some instructions later so pay attention.

Terminal without a folder:



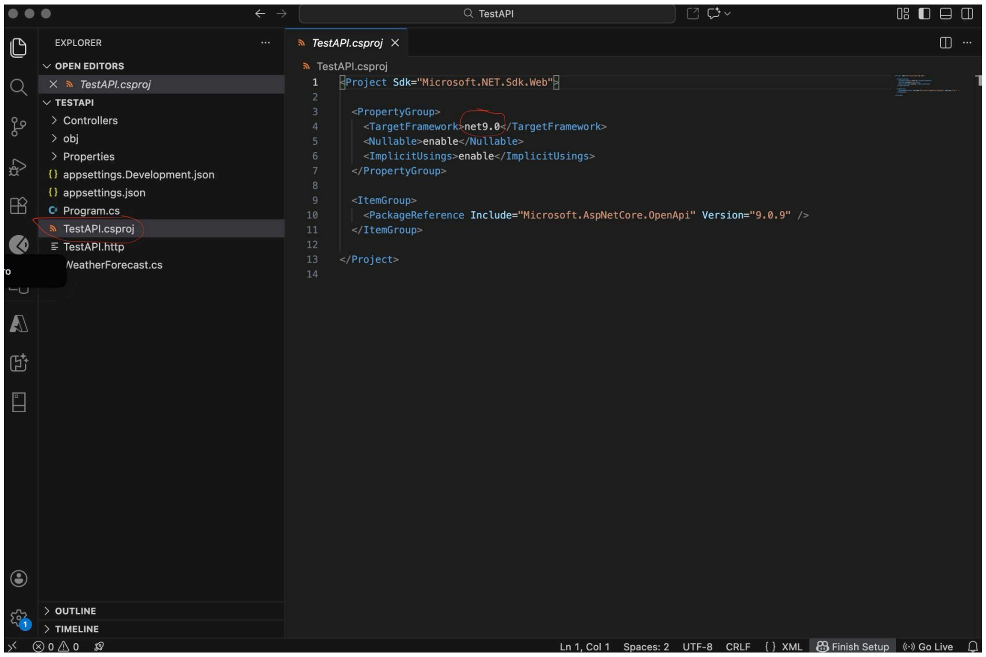
```
nicolasgonzaba@Nicolass-MacBook-Air TestAPI % dotnet new webapi --use-controllers
```

Terminal with folder:



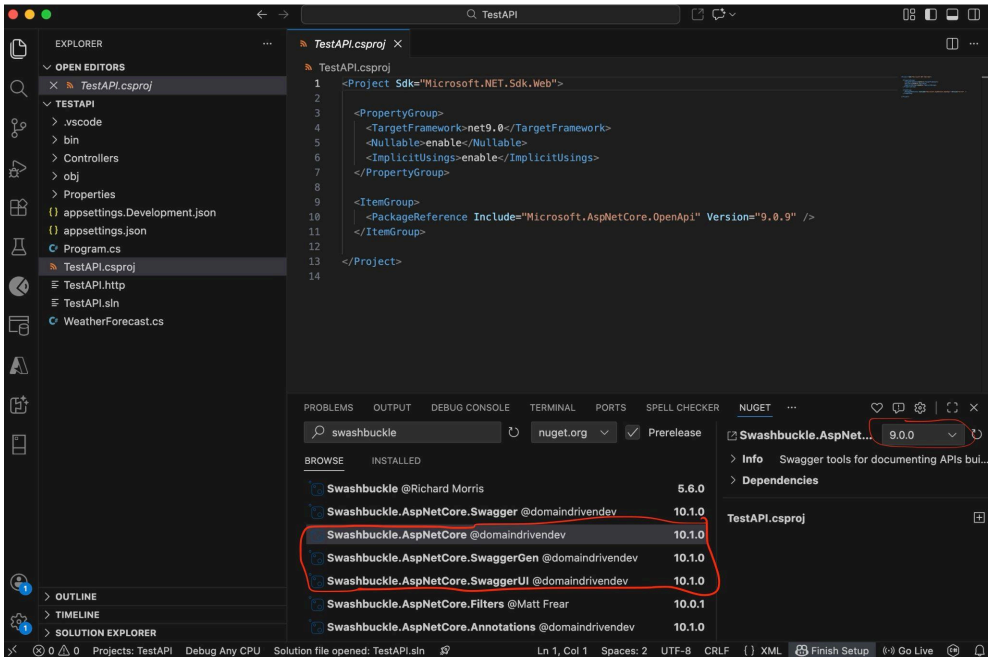
```
nicolasgonzaba@Nicolass-MacBook-Air TestAPI % dotnet new webapi --use-controllers -o APIName
```

Next we will be adding swagger to our project via nuget. You want to add the versions that match what version of .Net you have, for example I have .Net9 so I will be adding the 9.0 version of all the swashbuckles I need. If you don't know your dot net you can go to your csproj folder and check.



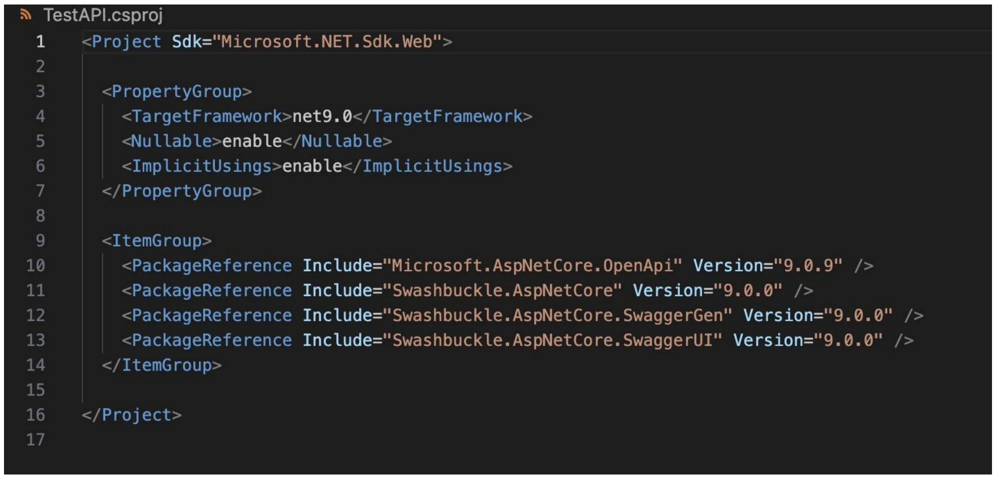
```xml
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net9.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="9.0.9" />
  </ItemGroup>

</Project>
```

Once you know your .Net version open up nuget and search for “Swashbuckle”. Add “Swashbuckle.AspNetCore” and make sure you have the version that makes your dot net selected when you do. Repeat for “Swashbuckle.AspNetCore.SwaggerGen” and “Swashbuckle.AspNetCore.SwaggerUI”. When you’re done you should see all 3 in your csproj folder


EXPLORER
OPEN EDITORS
TestAPI.csproj
TESTAPI
.vscode
bin
Controllers
obj
Properties
appsettings.Development.json
appsettings.json
Program.cs
TestAPI.csproj
TestAPI.http
TestAPI.sln
WeatherForecast.cs
OUTLINE
TIMELINE
SOLUTION EXPLORER
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER NUGET
swashbuckle
nuget.org
Prerelease
BROWSE INSTALLED
Swashbuckle.AspNet... 9.0.0
Info Swagger tools for documenting APIs bui...
Dependencies
TestAPI.csproj
Projects: TestAPI Debug Any CPU Solution file opened: TestAPI.sln Ln 1, Col 1 Spaces: 2 UTF-8 CRLF { } XML Finish Setup Go Live

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

  <PropertyGroup>
    <TargetFramework>net9.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="9.0.9" />
  </ItemGroup>

</Project>
```

```
Swashbuckle @Richard Morris                                  5.6.0
Swashbuckle.AspNetCore.Swagger @domaindrivendev             10.1.0
Swashbuckle.AspNetCore @domaindrivendev                     10.1.0
Swashbuckle.AspNetCore.SwaggerGen @domaindrivendev          10.1.0
Swashbuckle.AspNetCore.SwaggerUI @domaindrivendev           10.1.0
Swashbuckle.AspNetCore.Filters @Matt Frear                  10.0.1
Swashbuckle.AspNetCore.Annotations @domaindrivendev         10.1.0
```

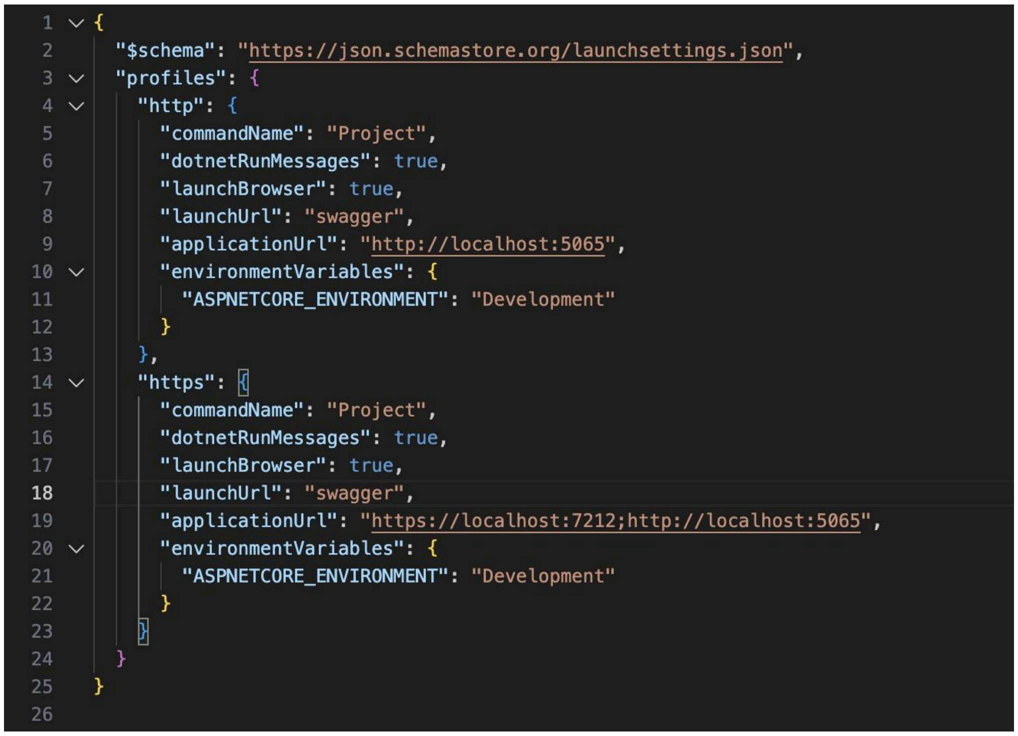


TestAPI.csproj

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

  <PropertyGroup>
    <TargetFramework>net9.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="9.0.9" />
    <PackageReference Include="Swashbuckle.AspNetCore" Version="9.0.0" />
    <PackageReference Include="Swashbuckle.AspNetCore.SwaggerGen" Version="9.0.0" />
    <PackageReference Include="Swashbuckle.AspNetCore.SwaggerUI" Version="9.0.0" />
  </ItemGroup>

</Project>
```

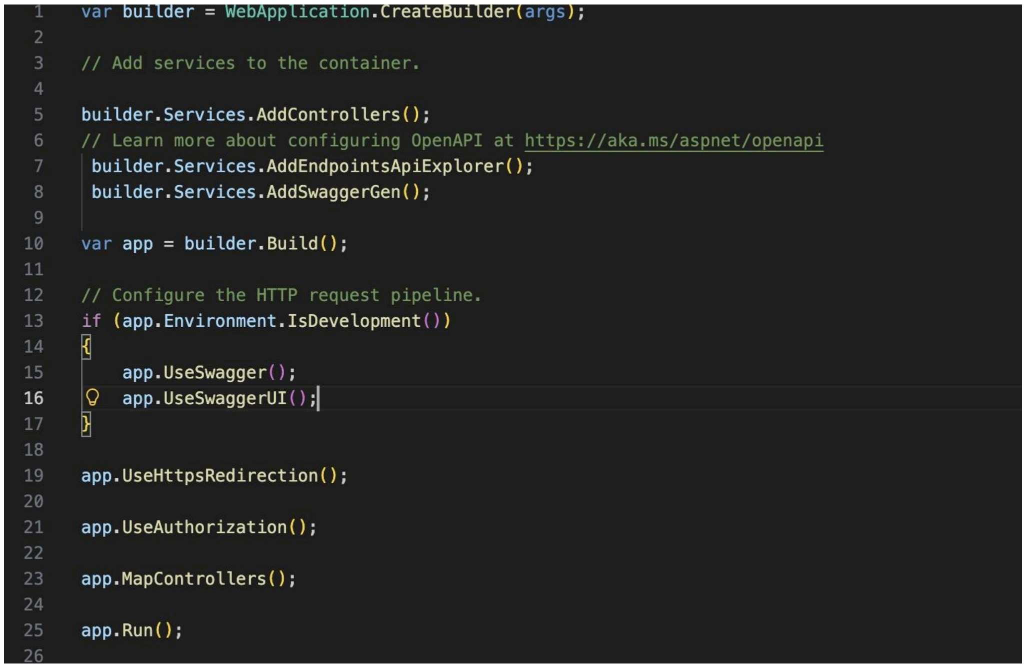
Next you want to go to your launchSettings.json in your Properties folder and change both "launchBrowser"s from false to true. Below both you will add "launchURL" : "swagger".

Your launchSettings.json should look something like this (your local host numbers may differ)



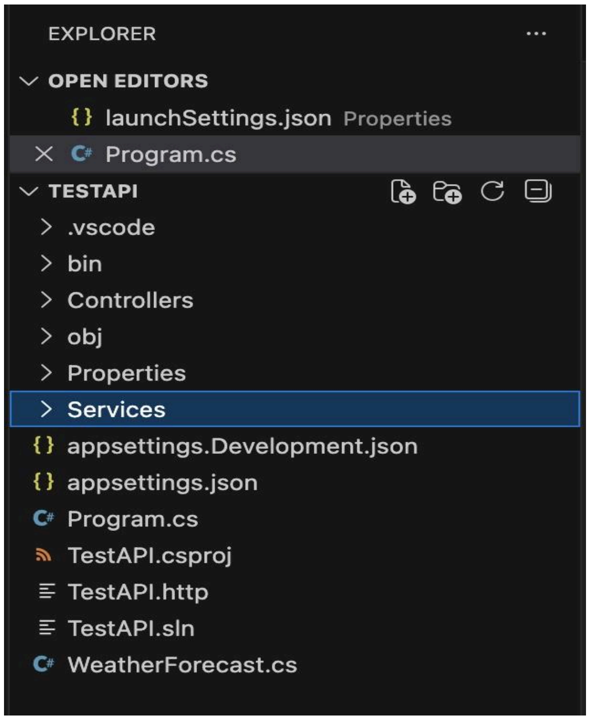
```json
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "http://localhost:5065",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "https://localhost:7212;http://localhost:5065",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Next you will move to program.cs. First you should change builder.services.AddOpenApi(); to builder.services.AddEndpointsApiExplorer(); and builder.services.AddSwaggerGen();. Next you should replace app.MapOpenApi(); with app.UseSwagger(); and app.UseSwaggerUi();



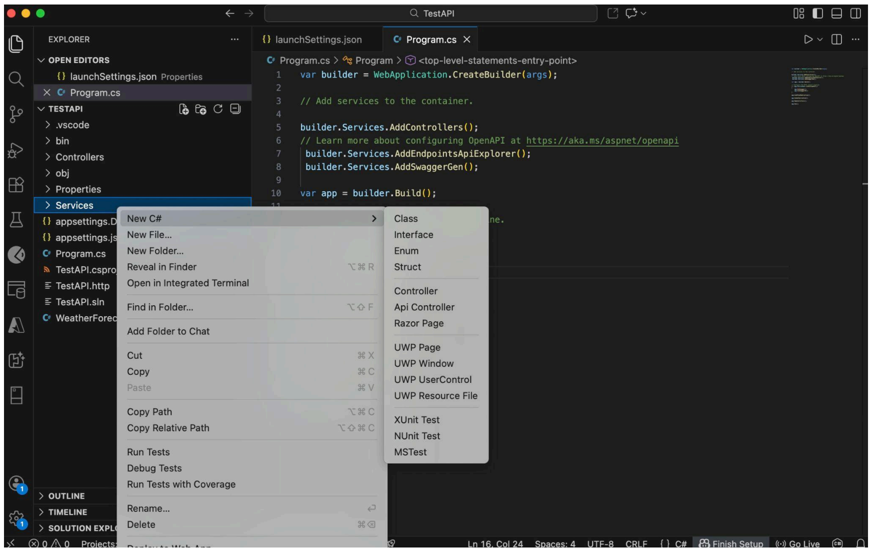
```
var builder = WebApplication.CreateBuilder(args);

// 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();

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();

```

Now we will create a Services folder, making sure it is not nested in any of our other folders. If done correctly you should have it appear below your properties folder.
NOTE:If you put your API in a separate folder make sure Services is in that folder. It should still not be nested in other folders and appear below properties.

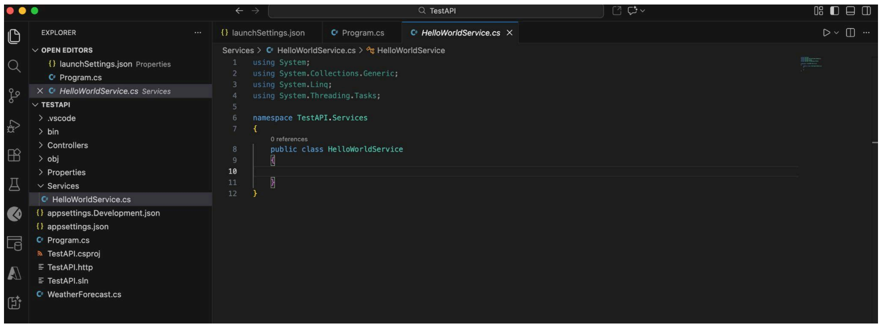


```
EXPLORER
OPEN EDITORS
  launchSettings.json Properties
  Program.cs
TESTAPI
  > .vscode
  > bin
  > Controllers
  > obj
  > Properties
  > Services
  appsettings.Development.json
  appsettings.json
  Program.cs
  TestAPI.csproj
  TestAPI.http
  TestAPI.sln
  WeatherForecast.cs
```

Next we will make a class file inside the Services folder where we will hold the logic for our endpoint. To do this right click on our Services folder, go to New C#, then Class. We will name this class HelloWordService for this example.

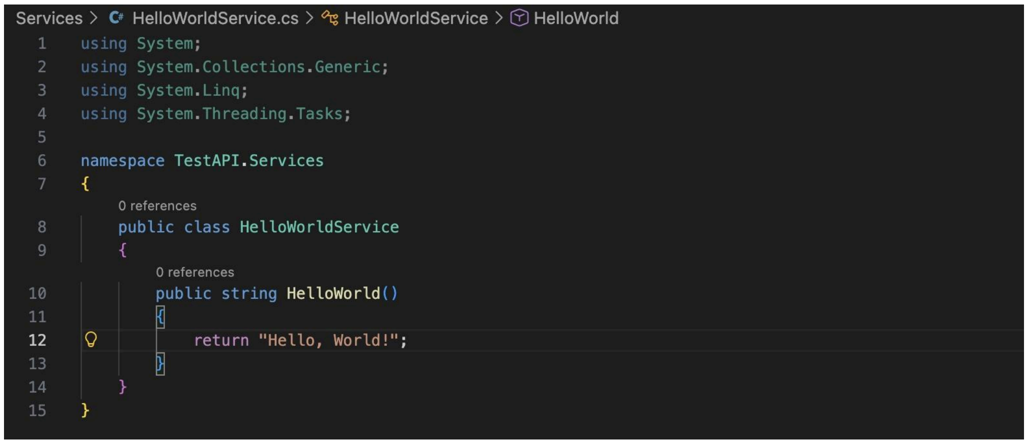


```
var builder = WebApplication.CreateBuilder(args);

// 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();

var app = builder.Build();
```



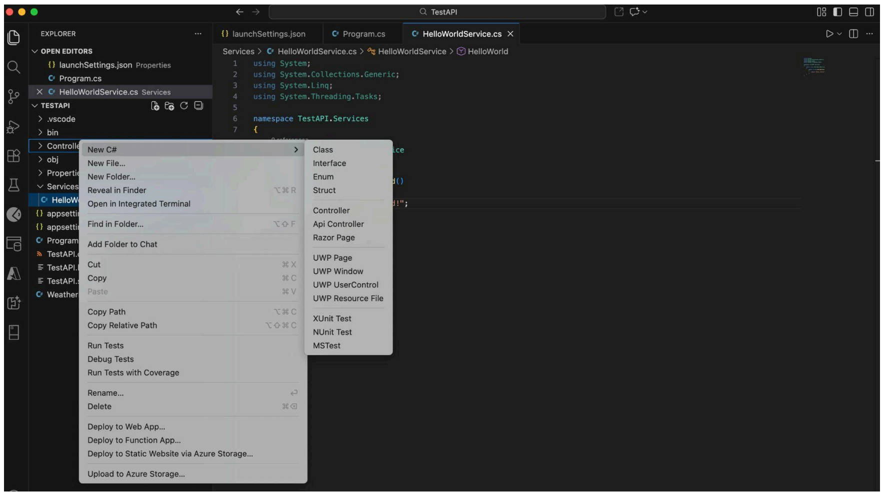
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace TestAPI.Services
{
    public class HelloWorldService
    {

    }
}
```

Inside our new Service will go to the public class HelloWordService already set up for us and add a public string inside the curly brackets. The name of this string will be HelloWorld() and we will add a set of curly brackets that will hold a response for the string. Inside those curly brackets we will make the response return "Hello, World!";


Services > HelloWorldService.cs > HelloWorldService > HelloWorld

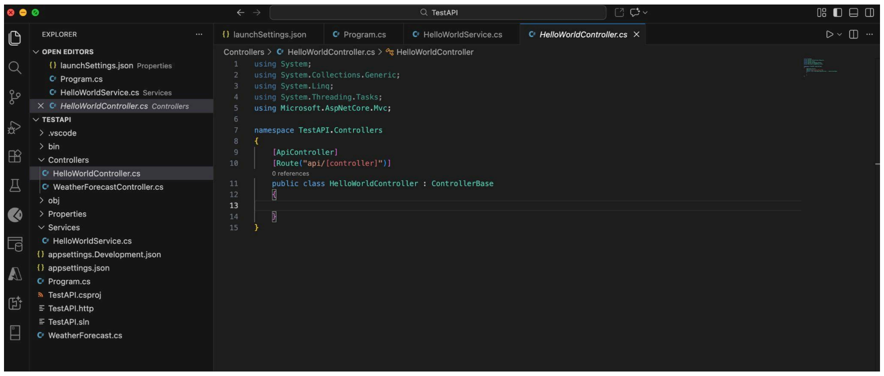
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace TestAPI.Services
{
    0 references
    public class HelloWorldService
    {
        0 references
        public string HelloWorld()
        {
            return "Hello, World!";
        }
    }
}
```

Now we will go to our Controllers folder and make a new API controller. Once again right click on our Controllers folder, go to New C#, then API controller. Name this HelloWorldController.



Services > HelloWorldService.cs > HelloWorldService > HelloWorld

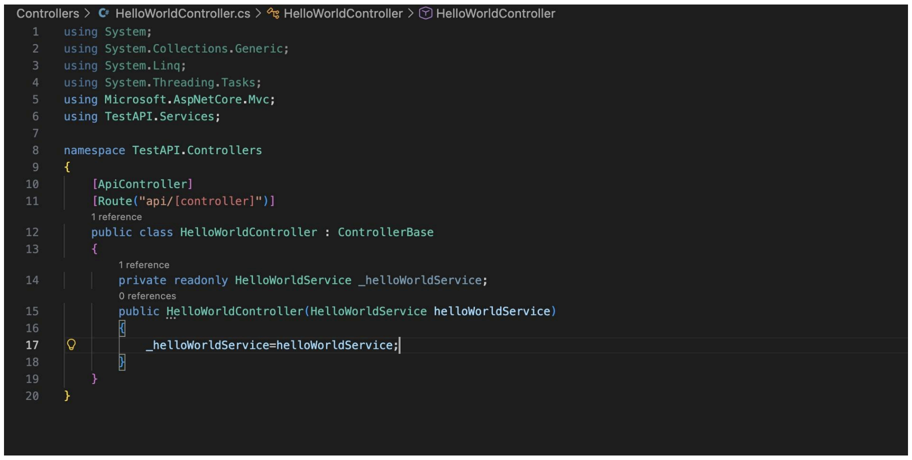
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace TestAPI.Services
{
```



Controllers > HelloWorldController.cs > HelloWorldController

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

namespace TestAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    0 references
    public class HelloWorldController : ControllerBase
    {

    }
}
```

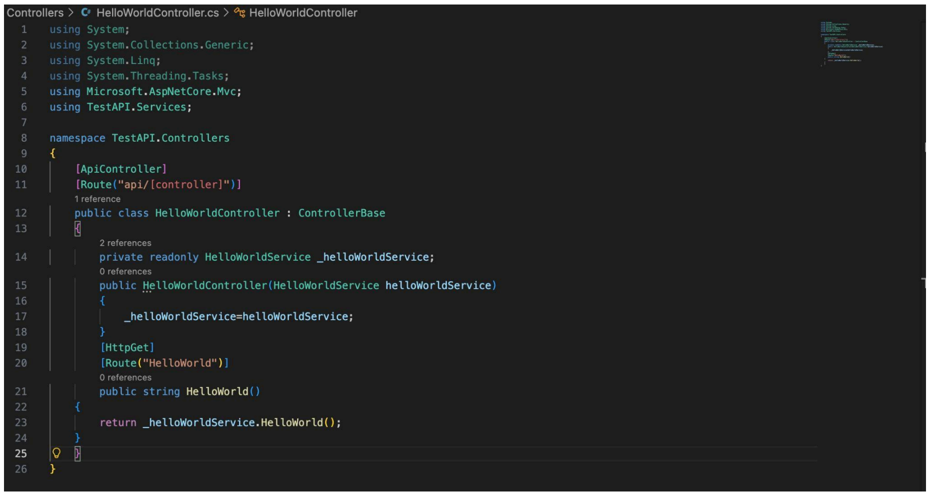
The first thing we must do is connect our controller to our service file. Inside our public class we start by adding private readonly HelloWorldService _helloWorldService;. Next we add a constructor below our readonly which can be shortcutted by typing ctor then pressing enter. We replace the Parameters in the constructor with HelloWorldService helloWorldService and add _helloWorldService=helloWorldService; in the curly brackets. All together this should connect our controller to our service.



Controllers > HelloWorldController.cs > HelloWorldController > HelloWorldController

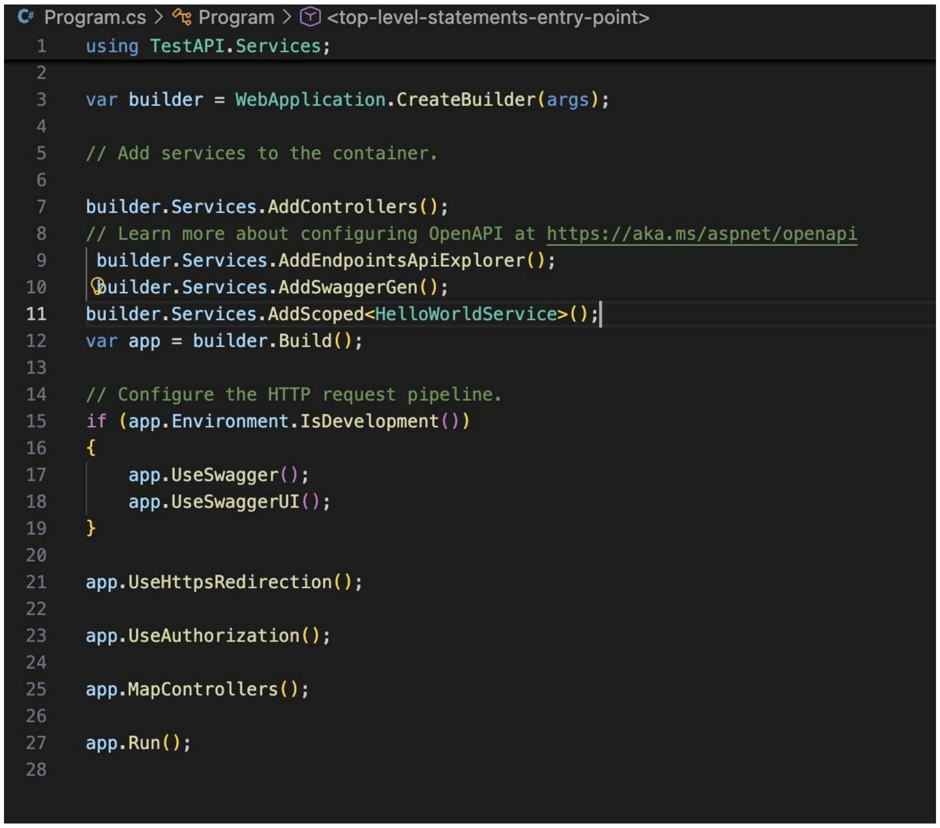
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using TestAPI.Services;

namespace TestAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class HelloWorldController : ControllerBase
    {
        1 reference
        private readonly HelloWorldService _helloWorldService;
        0 references
        public HelloWorldController(HelloWorldService helloWorldService)
        {
            _helloWorldService=helloWorldService;
        }
    }
}
```

Now we can finally create our actually endpoints. Below our constructor we add [HttpGet] and below that [Route(“HelloWorld”)] in order to give us access to our endpoint while hosting. Next we go down another line and add public string HelloWorld(){} to act as our endpoint and inside the curly brackets we add return _helloWorldService.HelloWorld(); allowing us to use the logic inside our service as an endpoint.


Controllers > HelloWorldController.cs > HelloWorldController

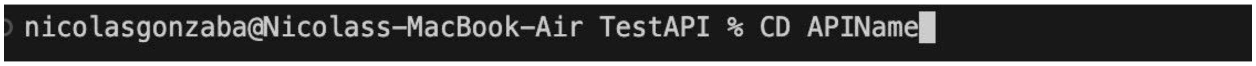
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using TestAPI.Services;

namespace TestAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class HelloWorldController : ControllerBase
    {
        2 references
        private readonly HelloWorldService _helloWorldService;
        0 references
        public HelloWorldController(HelloWorldService helloWorldService)
        {
            _helloWorldService=helloWorldService;
        }
        [HttpGet]
        [Route("HelloWorld")]
        0 references
        public string HelloWorld()
    {
        return _helloWorldService.HelloWorld();
    }
    }
}
```

We’re almost ready to host but we have one last step, we must link our service to our Program.cs. Go to your program.cs and add builder.services.AddScoped<HelloWorldService>(); Below your other builder.services

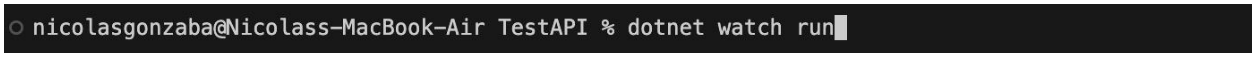


```
using TestAPI.Services;

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();

```

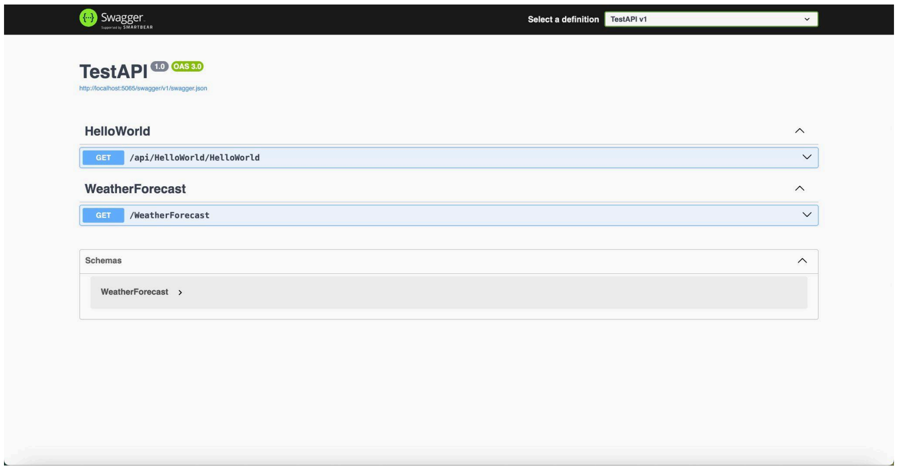
Finally you should be able to host your website and see your endpoints functions. To host your website go to your terminal and type dotnet watch run, which should open your hosted API on swagger in a new browser tab.
NOTE: If you put your API in a folder you must first put CD and your folder name in your terminal and hit enter in order to be seeded into your API folder before doing the above to host.



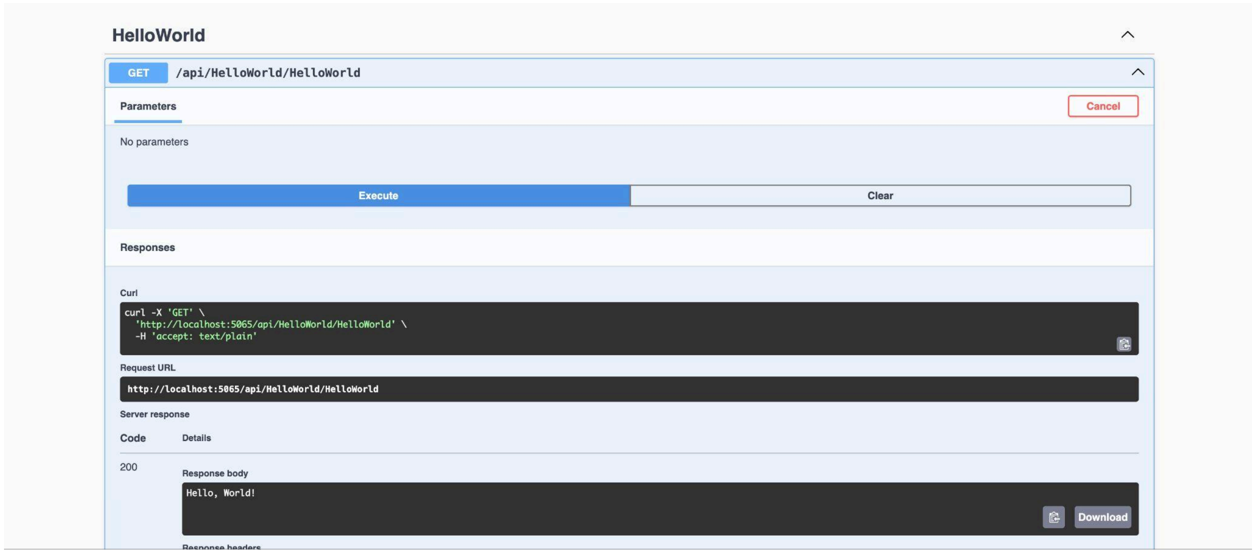
```
nicolasgonzaba@Nicolass-MacBook-Air TestAPI % CD APIName
```



```
nicolasgonzaba@Nicolass-MacBook-Air TestAPI % dotnet watch run
```



In order to test your endpoint simply unfold the get dropdown on your HelloWorld, Hit the try it out button, then execute. If you followed all the instructions then you should get the response “Hello, World!”



```
curl -X 'GET' \
  'http://localhost:5065/api/HelloWorld/HelloWorld' \
  -H 'accept: text/plain'
```

```
http://localhost:5065/api/HelloWorld/HelloWorld
```

```
Hello, World!
```