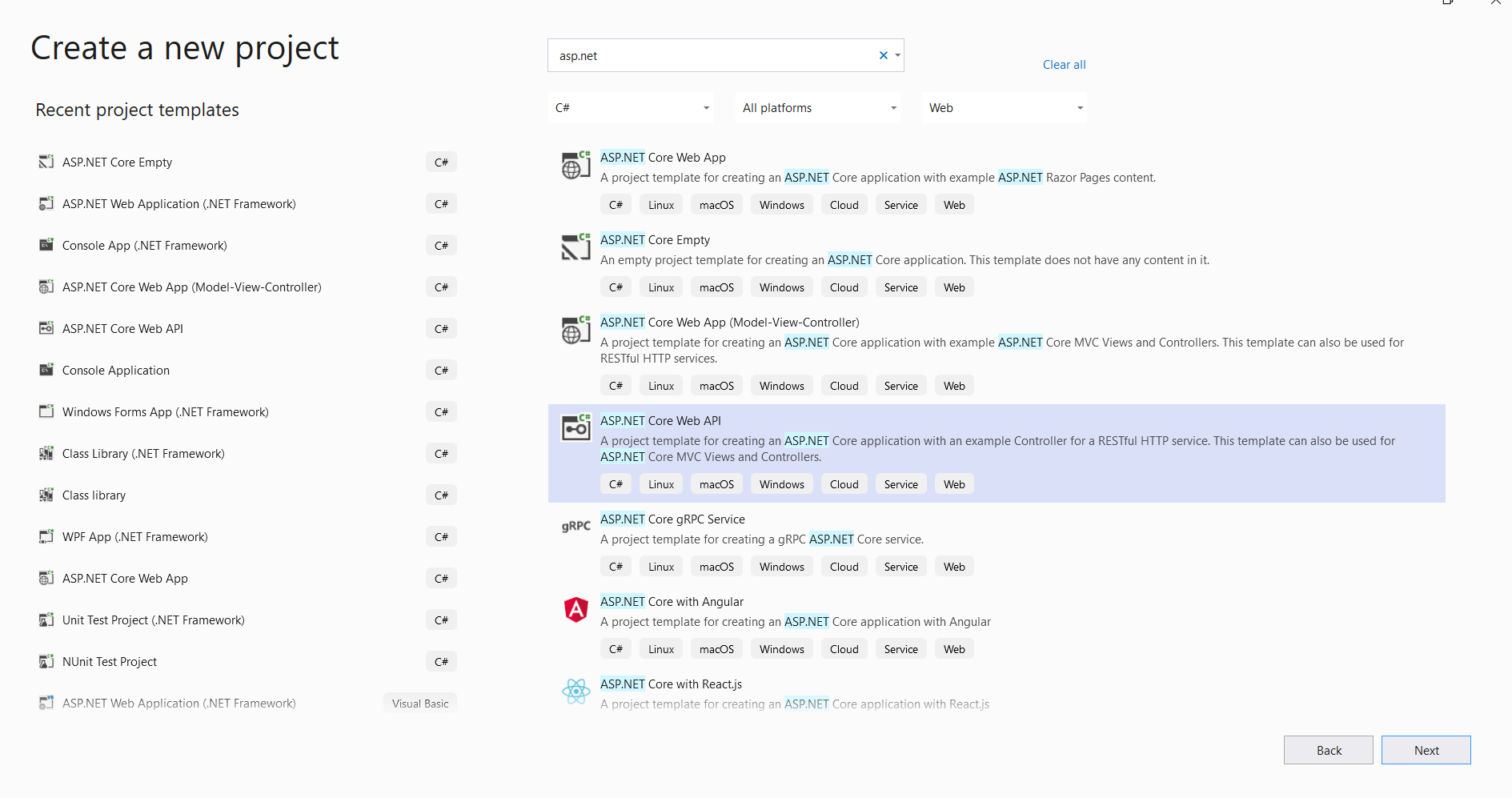
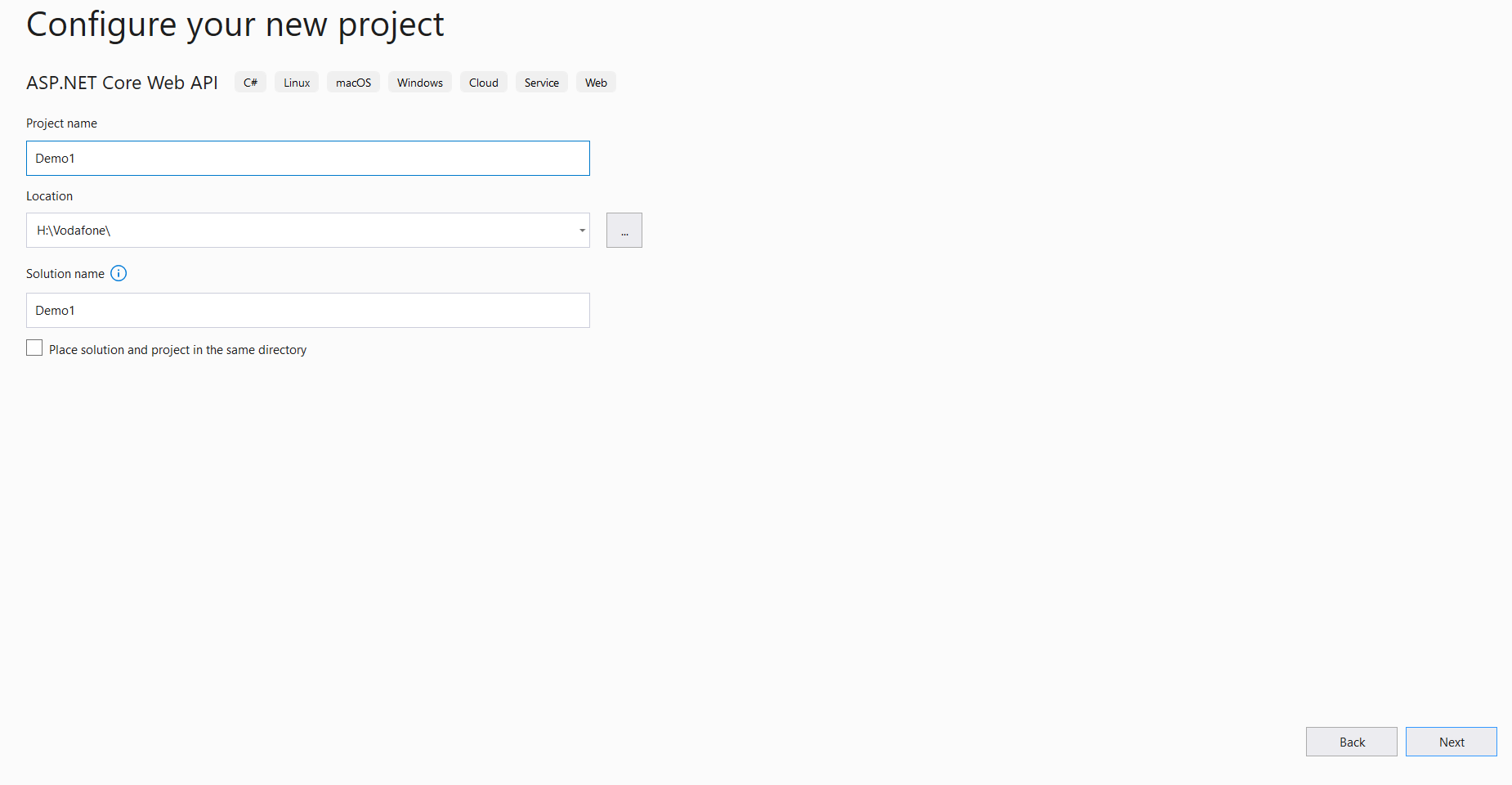
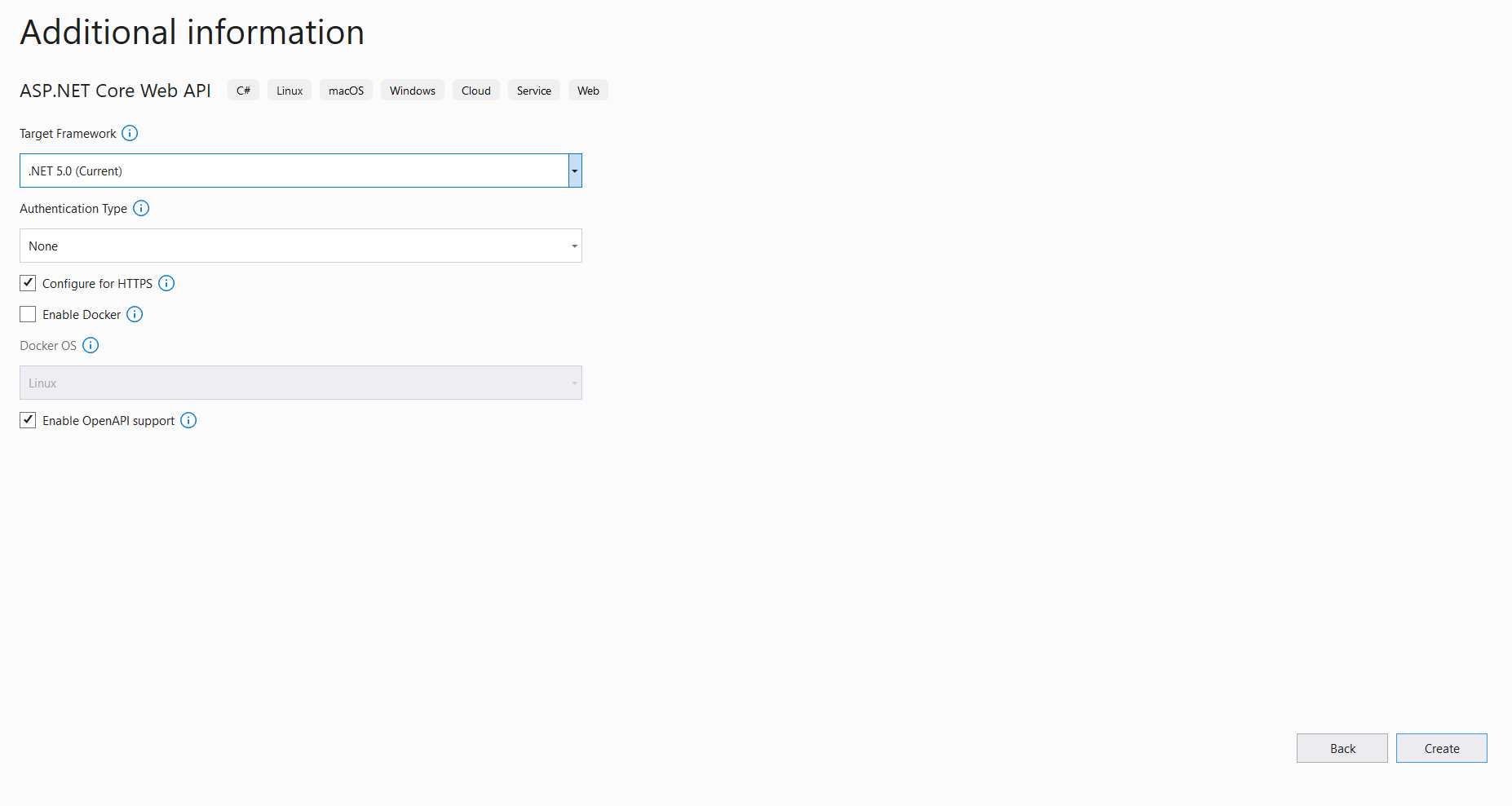
**Creating a New Project and Modifying LaunchSettings.json File**

we are going to create a Visual Studio project for our server part of the application. Open Visual Studio 2019 and create a new ASP.NET Core Web Application with the name AccountOwnerServer:



In the next window, we are going to choose Web API and from the left drop-down list .NET Core. Also, from the right drop-down, we are going to choose ASP.NET Core 3.1. **Take a note that you can safely select the .NET 5 version if you want, and follow along with the entire series**. Finally, let’s click the Create button and the project will load :





After the project has been created, we are going to modify the launchSettings.json file which is quite an important file for the .NET Core configuration. Let’s change the applicationUrl property and the launchBrowser property to false, to prevent web browser to start when the project starts.

In the Solution Explorer expand the Properties and double click on the launchSettings.json file. Let’s modify that file:

**{**

"$schema": "http://json.schemastore.org/launchsettings.json"**,**

"iisSettings": **{**

"windowsAuthentication": false**,**

"anonymousAuthentication": true**,**

"iisExpress": **{**

"applicationUrl": "http://localhost:5000"**,**

"sslPort": 0

**}**

**},**

"profiles": **{**

"IIS Express": **{**

"commandName": "IISExpress"**,**

"launchBrowser": false**,**

"environmentVariables": **{**

"ASPNETCORE\_ENVIRONMENT": "Development"

**}**

**},**

"AccountOwnerServer": **{**

"commandName": "Project"**,**

"launchBrowser": false**,**

"applicationUrl": "http://localhost:5000"**,**

"environmentVariables": **{**

"ASPNETCORE\_ENVIRONMENT": "Development"

**}**

**}**

**}**

**}**

## Program.cs and Startup.cs Explanations

In the Program.cs we can find this piece of code:

public static void Main**(**string**[]** args**)**

**{**

CreateHostBuilder**(**args**)**.Build**()**.Run**()**;

**}**

public static IHostBuilder CreateHostBuilder**(**string**[]** args**)** =&gt;

Host.CreateDefaultBuilder**(**args**)**

.ConfigureWebHostDefaults**(**webBuilder =&gt;

**{**

webBuilder.UseStartup&lt;Startup&gt;**()**;

**})**;

So basically, the CreateHostBuilder method has been broken into two logical parts:

* CreateDefaultBuilder(args) – ASP.NET Core uses this to configure the app configuration, logging, and dependency injection container
* ConfigureWebHostDefaults – This adds everything else required by a typical ASP.NET Core application (kestrel configuration, and using the Startup class, middleware pipeline…)

We can also notice this part of the code: UseStartup<Startup>(). The Startup class is mandatory for the .NET Core, in which we configure embedded or custom services that our application needs. When we open the Startup class, we can see the constructor and the two methods, which content we are going to change during the application development. In the method ConfigureServices, we will do exactly that, configure our services. Furthermore, in the Configure method, we are going to add different middleware components to the application’s request pipeline.

All of our configuration code could be written inside the ConfigureServices method, but large applications could potentially contain many services. As a result, it could make this method unreadable and hard to maintain. Therefore we will create extension methods for each configuration and place the configuration code inside those methods.

## Extension Methods and CORS Configuration

An extension method is inherently the static method. They play a great role in .NET Core configuration because they increase the readability of our code for sure. What makes it different from the other static methods is that it accepts “this” as the first parameter, and “this” represents the data type of the object which uses that extension method. An extension method must be inside a static class. This kind of method extends the behavior of the types in .NET.

So, let’s start writing some code.

Let’s create a new folder Extensions in the main project and a new class inside with the name ServiceExtensions. We are going to make that class static and it will consist of our service extension methods:

namespace AccountOwnerServer.Extensions

**{**

public static class ServiceExtensions

**{**

**}**

**}**

First, we need to configure CORS in our application. CORS (Cross-Origin Resource Sharing) is a mechanism that gives rights to the user to access resources from the server on a different domain. Because we will use Angular as a client-side on a different domain than the server’s domain, configuring CORS is mandatory. So, let’s add this code to the ServiceExtensions class:

public static void ConfigureCors**(**this IServiceCollection services**)**

**{**

services.AddCors**(**options =**>**

**{**

options.AddPolicy**(**"CorsPolicy",

builder =**>** builder.AllowAnyOrigin**()**

.AllowAnyMethod**()**

.AllowAnyHeader**())**;

**})**;

**}**

We are using the basic settings for adding CORS policy because for this project allowing any origin, method, and header is quite enough. But we can be more restrictive with those settings if we want. Instead of the AllowAnyOrigin() method which allows requests from any source,we could use the WithOrigins("http://www.something.com") which will allow requests just from the specified source.Also, instead of AllowAnyMethod() that allows all HTTP methods, we can use WithMethods("POST", "GET")that will allow only specified HTTP methods. Furthermore, we can make the same changes for the AllowAnyHeader()method by using, for example, the WithHeaders("accept", "content-type")method to allow only specified headers.

## IIS Configuration as Part of .NET Core Configuration

Additionally, we need to configure an IIS integration which will help us with the IIS deployment. Add the following code to the ServiceExtensions class:

public static void ConfigureIISIntegration**(**this IServiceCollection services**)**

**{**

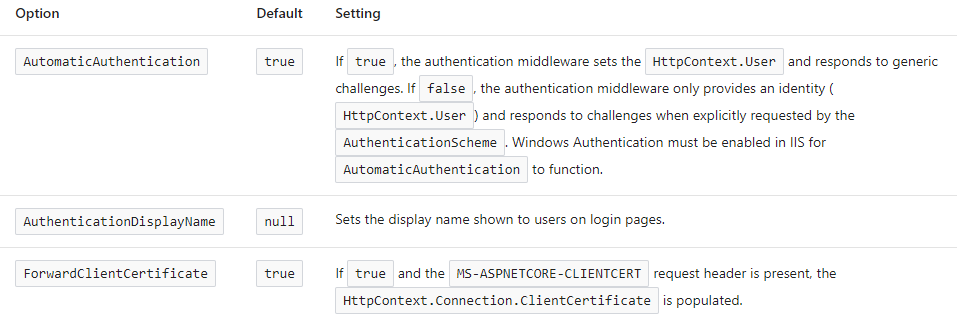
services.Configure**<**IISOptions**>(**options =**>**

**{**

**})**;

**}**

We do not initialize any of the properties inside the options because we are just fine with the default values. For more pieces of information about those properties, look at the explanation:



In the Startup class we are going to change the ConfigureServices and Configure methods:

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

**{**

services.ConfigureCors**()**;

services.ConfigureIISIntegration**()**;

services.AddControllers**()**;

**}**

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

**{**

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

**{**

app.UseDeveloperExceptionPage**()**;

**}**

app.UseHttpsRedirection**()**;

app.UseStaticFiles**()**;

app.UseForwardedHeaders**(**new ForwardedHeadersOptions

**{**

ForwardedHeaders = ForwardedHeaders.All

**})**;

app.UseRouting**()**;

app.UseCors**(**"CorsPolicy"**)**;

app.UseAuthorization**()**;

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

**{**

endpoints.MapControllers**()**;

**})**;

**}**

In the ConfigureServices method, CORS, and IIS configuration have been added. Furthermore, the CORS configuration has been added to the application’s pipeline inside the Configuration method. The important thing to notice here is that we call the UseCors method between the UseRouting and UseAuthorization methods.  This is the best practice, which Microsoft recommends.

But as you may notice, there is a little more code. Therefore, we are going to explain it now.

* app.UseForwardedHeaders will forward proxy headers to the current request. This will help us during the Linux deployment.
* app.UseStaticFiles() enables using static files for the request. If we don’t set a path to the static files, it will use a wwwroot folder in our solution explorer by default.

## Conclusion

Concerning the .NET Core configuration, we could say that this is quite enough. Now you know how to modify the launchSetting.json file, the purpose of the Program and the Startup classes, how to configure services, and finally how to create and use extension methods.