## Blazor WebAssembly Template Files

Before we wrap this module up, let’s go through the Blazor WebAssembly template and inspect some of the most important files we’ll be looking at and modifying throughout this course. The first file we want to look at is the **launchSettings.json** file, located inside the Properties folder of our template. This is the file that Visual Studio uses to determine which debugging profiles are available to use for our concrete application. By default, there are two debugging profiles available, “IIS Express” one, and “{Name of the Project}” one.This means we can currently launch our application both as an IIS hosted application or as a self-hosted application. Both of these options can be seen in the navigation menu on the top: And IIS Express is selected as the default option. We can also define different environment variables within it. There is one example of it already in this file, and that’s the **ASPNETCORE_ENVIRONMENT** variable that is set to “**Development**” and is used by our application to indicate that we’re working in the development environment. Other options present at default are: commandName which is basically a name of the profile launchBrowser which tells whether the application should launch in the browser or not inspectUrl, the URL we can access to debug Blazor WebAssembly application. We’ll talk more about this one later on. applicationUrl which is the URL on which our application will be running and environmentVariables, which is a list of all the environment variables for our application. Okay, let’s move on to the next file we want to look at and that’s **Program.cs** file. If you’ve done any work in .NET Core, you are familiar with this file. This is the main entry to our project, but the main thing to take away here is that we’ll be using this file to register our services, unlike Startup.cs in ASP.NET Core Web API and other types of applications. Blazor WebAssembly doesn’t have a Startup class, so all the service registrations are to be done here. This is a very basic Program.cs file, but we’ll be changing it several times during this course. Next on, we have the **App.razor** which is the main component of the application. Inside it, we can see that client routing is set up, and that we can define/change the main layout file. The router component is in charge of intercepting the browser navigation and directing it to the right page of our application. Right beside our App.razor file, there is a file called **_Imports.razor**. Inside this file, we can define common Razor directives as @using directive to include our dependencies. We’ve already seen how our main layout is called in the App.razor file, and it’s located right in the Shared folder. The **MainLayout.razor** file inherits from the LayoutComponentBase and contains two important components of our application, NavMenu component, dedicated to the navigation menu to the left, and @Body which is all the content that needs to be rendered. NavMenu.razor file contains a bit more logic that’s responsible for making our menu work as it does. We’ll go into more detail about navigation later on the course. There is another folder called **Pages**, which contains all the routable components/pages in our application. By default, we have **Index.razor, Counter.razor, and FetchData.razor** files in here. Each of these represents a single navigation route route in our menu Finally, we have the wwwroot folder, which is a folder that contains all the static files for our application. This includes files such as stylesheets, images, favicon… and of course index.html file which is the main entry file of the application. We’ve already mentioned that inside **index.html** we can find the reference to **blazor.webassembly.js** file which we need for the Blazor WebAssembly project type. That’s it for the important files for now. We’ll be adding some more like the appsettings.json file to define values for different environments, but more on that later. Now that we’ve learned a bit of theory about Blazor and we’ve seen how it works in practice, let’s conclude this module and recap what we’ve learned so far.