Blazor project structure

Files and folders in ASP.NET Core Blazor project

Program.cs

This file contains the Main() method which is the entry point for both the project types (i.e Blazor WebAssembly and Blazor Server).

In a Blazor server project, the Main() method calls CreateHostBuilder() method which sets up the ASP.NET Core host.

In a Blazor WebAssembly project, the App component, which is the root component of the application, is specified in the Main method. This root component is present in the root project folder in App.razor file.

Components are the building blocks of a Blazor application. We will discuss everything you need to know build effective and reusable blazor components in our upcoming videos. For now, just understand the App component is the root component of the application.

wwwroot

For both the project types, this folder contains the static files like images, stylesheets etc.

App.razor

This is the root component of the application. It uses the built-in Router component and sets up client-side routing. It is this Router component that intercepts browser navigation and renders the page that matches the requested address. The Router uses the Found property to display the content when a match is found. If a match is not found, the NotFound property is used to display the  message - Sorry, there's nothing at this address.

Pages folder

This folder contains the \_Host razor page and the routable components that make up the Blazor app. The components have the .razor extension.

* Index component (Index.razor) – Rendered when we navigate to the root application URL.
* Counter component (Counter.razor) – Rendered when we navigate to the path /counter.
* FetchData component (FetchData.razor) – Rendered when we navigate to the path /fetchdata.
* Error component (Error.razor) – Rendered when an unhandled exception occurs in the blazor app.

Shared folder

As the name implies, contains the shared components

MainLayout component (MainLayout.razor)

The application's main layout component

NavMenu component (NavMenu.razor)

Implements the navigation menu on the sidebar. NavLink component, renders navigation links to other Razor components like the index, counter and fetchdata components. This NavLink component is intelligent enough to highlight the navigation menu item, if it's component is currently displayed.

\_Imports.razor

This is like the \_ViewImports.cshtml file in an asp.net core MVC project. This file contains the common namespaces so we do not have to include them in every razor component.

wwwroot/index.html

This is the root page in a Blazor WebAssembly project and is implemented as an html page. When a first request hits the application, it is this page, that is initially served. It has the standard HTML, HEAD and BODY tags. It specifies where the root application component App.razor should be rendered. You can find this App.razor root component in the root project folder. It is included on the page as an HTML element <app>. We will discuss razor components in detail in our upcoming videos.

This index.html page also loads the blazor WebAssembly JavaScript file (\_framework/blazor.webassembly.js). It is this file that is responsible for downloading

* The compiled blazor application, it's dependencies and the .NET runtime
* It also initializes the runtime to run the blazor app in the browser

Startup.cs

This file is present only in the Blazor Server project. As the name implies it contains the applications's startup logic. The Startup class contains the following two methods.

**ConfigureServices** - Configures the applications DI i.e dependency injection services. For example AddServerSideBlazor() method adds Blazor server side services. On the IServiceCollection interface there are several methods that start with the word Add. These methods add different services for the Blazor application. We can even add our own services to the DI container. We will see this in action in our upcoming videos.

**Configure** - Configures the app's request processing pipeline. Depending on what we want the Blazor application to be capable of doing we add or remove the respective middleware components from request processing pipeline. For example, UseStaticFiles() method adds the middleware component that can server static files like images, css etc.

If you have any experience with ASP.NET core, then you already know what middleware components are. We discussed them in detail in [Part 10](https://youtu.be/ALu4jtvjSYw) of our [ASP.NET Core MVC tutorial for beginners](https://www.pragimtech.com/courses/asp-net-core-mvc-tutorial-for-beginners/) course.

MapBlazorHub sets up the endpoint for the SignalR connection with the client browser.

Pages/\_Host.cshtml

This is the root page of the application and is specified by calling MapFallbackToPage("/\_Host") method. It is implemented as a razor page.

It is this page, that is initially served when a first request hits the application. It has the standard HTML, HEAD and BODY tags. It also specifies where the root application component, App component (App.razor) must be rendered. Finally, it also loads the blazor.server.js JavaScript file, which sets up the real-time SignalR connection between the server and the client browser. This connection is used to exchange information between the client and the server. SignalR, is a great framework for adding real-time web functionality to apps.

Data folder (Blazor Server)

Contains code files related to the sample WeatherForecast service

appsettings.json (Blazor Server)

Just like an asp.net core mvc project, a blazor project also uses this file to store the Configuration settings.

Conclusion

One very important point to keep in mind is, Blazor Server and Blazor WebAssembly are just 2 different ways we can host a Blazor application. Remember everything in a Blazor application is a razor component. Components are the fundamental building blocks of a Blazor application. The way we build these components is the same for a Blazor server app as well as for a Blazor WebAssembly app. So, the point that I am trying to make is, there is just one Blazor framework and the way we build, the Blazor server app and the Blazor WebAssembly app is very similar. The only difference is in the way the app is hosted. If implemented properly, it is easy to convert a Blazor server app to a Blazor WebAssembly app and vice-versa.

Blazor WebAssembly is still in preview and not yet officially released. So, as part of this course we will be building a Blazor server app.