**Blazor Full-Stack Web Dev in ASP .NET Core 3.1**

By Shahed C on January 13, 2020

[8 Replies](https://wakeupandcode.com/blazor-full-stack-web-dev-in-asp-net-core-3-1/#comments)



This is the second of a new [series of posts](https://wakeupandcode.com/aspnetcore/#aspnetcore2020) on ASP .NET Core 3.1 for 2020. In this series, we’ll cover 26 topics over a span of 26 weeks from January through June 2020, titled **ASP .NET Core A-Z!** To differentiate from the [2019 series](https://wakeupandcode.com/aspnetcore/#aspnetcore2019), the 2020 series will mostly focus on a growing single codebase ([NetLearner!](https://wakeupandcode.com/netlearner-on-asp-net-core-3-1/)) instead of new unrelated code snippets week.

Previous post:

* [Authentication & Authorization in ASP .NET Core 3.1](https://wakeupandcode.com/authentication-authorization-in-asp-net-core-3-1/)

**NetLearner on GitHub**:

* Repository: <https://github.com/shahedc/NetLearnerApp>
* v0.2-alpha release: <https://github.com/shahedc/NetLearnerApp/releases/tag/v0.2-alpha>

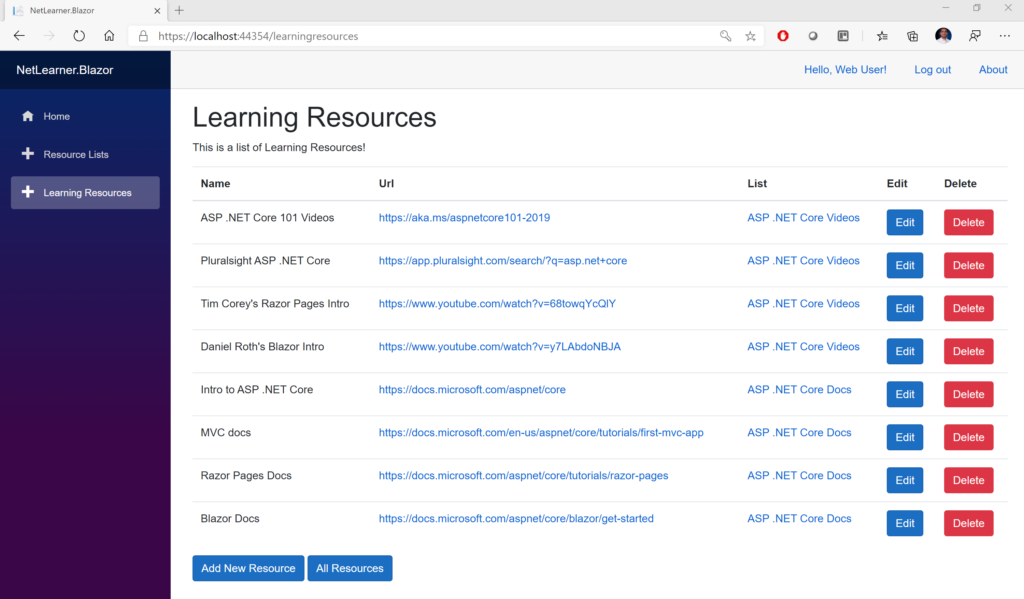
**In this Article:**

* B is for Blazor Full-Stack Web Dev
* Entry Point and Configuration
* Rendering the Application
* LifeCycle Methods
* Updating the UI
* Next Steps
* References

**B is for Blazor Full-Stack Web Dev**

In my 2019 A-Z series, I covered [Blazor for ASP .NET Core](https://wakeupandcode.com/blazor-full-stack-web-dev-in-asp-net-core/) while it was still experimental. As of ASP .NET Core 3.1, server-side Blazor has now been released, while client-side Blazor (currently in preview) is expected to arrive in May 2020. This post will cover server-side Blazor, as seen in [NetLearner](https://wakeupandcode.com/netlearner-on-asp-net-core-3-1/).

To see the code in action, open the solution in Visual Studio 2019 and run the [NetLearner.Blazor](https://github.com/shahedc/NetLearnerApp/tree/master/src/NetLearner.Blazor) project. All modern web browsers should be able to run the project.

NetLearner.Blazor web app in action

**Entry Point and Configuration**

Let’s start with [Program.cs](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Program.cs), the entry point of the application. Just like any ASP .NET Core web application, there is a Main method that sets up the entry point. A quick call to **CreateHostBuilder**() in the same file ensures that two things will happen: The Generic Host will call its own **CreateDefaultBuilder**() method (*similar to how it works in a typical ASP .NET Core web application*) and it will also call **UseStartup**() to identify the Startup class where the application is configured.

public class Program  
{  
 public static void Main(string[] args)  
 {  
 CreateHostBuilder(args).Build().Run();  
 }  
   
 public static IHostBuilder CreateHostBuilder(string[] args) =>  
 Host.CreateDefaultBuilder(args)  
 .ConfigureWebHostDefaults(webBuilder =>  
 {  
 webBuilder.UseStartup<Startup>();  
 });  
 }

Note that the Startup class doesn’t *have*to be called Startup, but you do have to tell your application what it’s called. In the [Startup.cs](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Startup.cs) file, you will see the familiar **ConfigureServices**() and **Configure**() methods, but you won’t need any of the regular MVC-related lines of code that set up the HTTP pipeline for an MVC (or Razor Pages) application. Instead, you just need a call to **AddServerSideBlazor**() in **ConfigureServices**() and a call to **MapBlazorHub**() in **Configure**() while setting up endpoints with **UseEndPoints**().

public class Startup  
{  
 public void ConfigureServices(IServiceCollection services)  
 {  
 ...  
 services.AddServerSideBlazor();  
 ...  
 }   
  
 public void Configure(IApplicationBuilder app, IWebHostEnvironment env)   
 {  
 ...  
 app.UseEndpoints(endpoints =>   
 {  
 endpoints.MapControllers();  
 endpoints.MapBlazorHub();  
 endpoints.MapFallbackToPage("/\_Host");  
 });  
 }  
}

Note that the **Configure**() method takes in an app object of type **IApplicationBuilder**, similar to the IApplicationBuilder we see in regular ASP .NET Core web apps.  A call to **MapFallBackToPage**() indicates the “/\_Host” root page, which is defined in the [\_Host.cshtml](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Pages/_Host.cshtml) page in the Pages subfolder.

**Rendering the Application**

This “app” (formerly defined in a static “index.html” in pre-release versions) is now defined in the aforementioned “\_Host.cshtml” page. This page contains an <app> element containing the main App component.

<html>  
...   
<body>  
 <app>  
 <component type="typeof(App)" render-mode="ServerPrerendered" />  
 </app>  
...  
</body>  
</html>

The HTML in this page has two things worth noting: an <**app**> element within the <**body**>, and a <**component**> element of type “**App**” with a render-mode attribute set to “ServerPrerendered”. This is one of 3 render modes for a Blazor component: Static, Server and ServerPrerendered. For more information on render modes, check out the official docs at:

* Blazor Hosting Models: <https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-3.1>

According to the documentation, this setting “*renders the component into static HTML and includes a marker for a Blazor Server app. When the user-agent starts, this marker is used to bootstrap a Blazor app.*“

The App component is defined in [App.razor](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/App.razor), at the root of the Blazor web app project. This App component contains nested [authentication-enabled](https://wakeupandcode.com/authentication-authorization-in-asp-net-core-3-1/) components, that make use of the MainLayout to display the single-page web application in a browser. If the user-requested routedata is found, the requested page (or root page) is displayed. If the user-requested routedata is invalid (not found), it displays a “sorry” message to the end user.

<CascadingAuthenticationState>  
 <Router AppAssembly="@typeof(Program).Assembly">  
 <Found Context="routeData">  
 <AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />  
 </Found>  
 <NotFound>  
 <LayoutView Layout="@typeof(MainLayout)">  
 <p>Sorry, there's nothing at this address.</p>  
 </LayoutView>  
 </NotFound>  
 </Router>  
</CascadingAuthenticationState>

The [MainLayout.razor component](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Shared/MainLayout.razor) (under /Shared) contains the following:

* NavMenu: displays navigation menu in sidebar
* LoginDisplay: displays links to register and log in/out
* \_CookieConsentPartial: displays GDPR-inspired cookie message
* @Body keyword: replaced by content of layout when rendered

The [\_Layout.cshtml layout file](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Pages/Shared/_Layout.cshtml) (under /Pages/Shared) acts as a template and includes a call to @RenderBody to display its content. This content is determined by route info that is requested by the user, e.g. [Index.razor](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Pages/Index.razor) when the root “/” is requested, [LearningResources.razor](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Pages/LearningResources.razor) when the route “/learningresources” is requested.

<div class="container">  
 <main role="main" class="pb-3">  
 @RenderBody()  
 </main>  
 </div>

The [NavMenu.razor component](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Shared/NavMenu.razor) contains NavLinks that point to various routes. For more information about routing in Blazor, check out the official docs at:

* Routing in Blazor: <https://docs.microsoft.com/en-us/aspnet/core/blazor/routing?view=aspnetcore-3.1>

**LifeCycle Methods**

A Blazor application goes through several lifecycle methods, including both asynchronous and non-asynchronous versions where applicable. Some important ones are listed below:

* **OnInitializedAsync**() and **OnInitialized**(): invoked after receiving initial params
* **OnParametersSetAsync**() and **OnParametersSet**(): called after receiving params from its parent, after initialization
* **OnAfterRenderAsync**() and **OnAfterRender**(): called after each render
* **ShouldRender**(): used to suppress subsequent rendering of the component
* **StateHasChanged**(): called to indicate that state has changed, can be triggered manually

These methods can be overridden and defined in the @**code** section (formerly @**functions** section) of a .razor page, e.g. [LearningResources.razor](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Pages/LearningResources.razor).

@code {  
 ...  
 protected override async Task OnInitializedAsync()  
 {  
 ...  
 }   
 ...  
}

**Updating the UI**

The C# code in [LearningResources.razor](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Pages/LearningResources.razor) includes methods to initialize the page, handle user interaction, and keep track of data changes. Every call to an event handler from an HTML element (e.g. OnClick event for an input button) can be bound to a C# method to handle that event. The **StateHasChanged**() method can be called manually to rerender the component, e.g. when an item is added/edited/deleted in the UI.

<div>  
 <input type="button" class="btn btn-primary" value="All Resources" @onclick="(() => DataChanged())" />  
</div>  
  
...  
private async void DataChanged()  
{  
 learningResources = await learningResourceService.Get();  
 ResourceLists = await resourceListService.Get();  
 StateHasChanged();  
}

Note that the **DataChanged**() method includes some asynchronous calls to **Get**() methods from a service class. These are the service classes in the shared library that are also used by the MVC and Razor Pages web apps in [NetLearner](https://wakeupandcode.com/netlearner-on-asp-net-core-3-1/).

Parameters defined in the C# code can be used similar to HTML attributes when using the component, including RenderFragments can be used like nested HTML elements. These can be defined in sub-components such as [ConfirmDialog.razor](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Shared/ConfirmDialog.razor) and [ResourceDetail.razor](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Pages/ResourceDetail.razor) that are inside the [LearningResources.razor](https://github.com/shahedc/NetLearnerApp/blob/master/src/NetLearner.Blazor/Pages/LearningResources.razor) component.

<ResourceDetail LearningResourceObject="learningResourceObject"  
 ResourceListValues="ResourceLists"  
 DataChanged="@DataChanged">  
 <CustomHeader>@customHeader</CustomHeader>  
</ResourceDetail>

Inside the subcomponent, you would define the parameters as such:

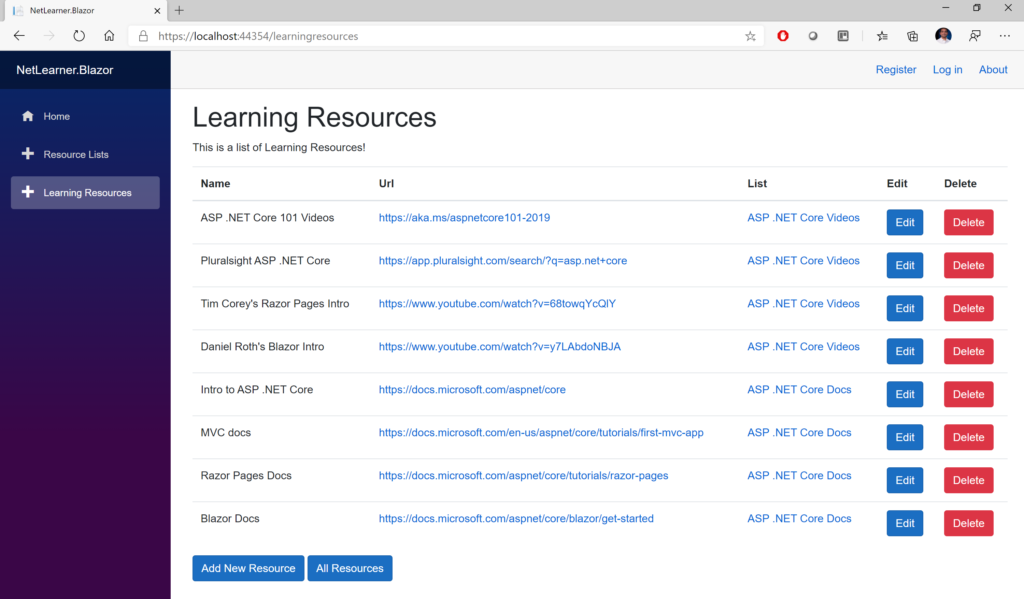
@code {  
 [Parameter]  
 public LearningResource LearningResourceObject { get; set; }  
  
 [Parameter]  
 public List<ResourceList> ResourceListValues { get; set; }  
  
 [Parameter]  
 public Action DataChanged { get; set; }  
  
 [Parameter]  
 public RenderFragment CustomHeader { get; set; }  
}

For more information on the creation and use of Razor components in Blazor, check out the official documentation at:

* Razor Components in Blazor: <https://docs.microsoft.com/en-us/aspnet/core/blazor/components?view=aspnetcore-3.1>

**Next Steps**

Run the Blazor web app from the [NetLearner repo](https://github.com/shahedc/NetLearnerApp) and try using the UI to add, edit and delete items. Make sure you remove the restrictions mentioned in a previous post about [NetLearner](https://wakeupandcode.com/netlearner-on-asp-net-core-3-1/), which will allow you to register as a new user, log in and perform CRUD operations.

NetLearner.Blazor: Learning Resources

There is so much more to learn about this exciting new framework. Blazor’s reusable components can take various forms. In addition to server-side Blazor (released in late 2019 with .NET Core 3.1), you can also host Blazor apps on the client-side from within an ASP .NET Core web app. Client-side Blazor is currently in preview and is [expected in a May 2020 release](https://devblogs.microsoft.com/aspnet/blazor-server-in-net-core-3-0-scenarios-and-performance/).

**References**

* Official Blazor website: <https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor>
* Intro to Blazor: <https://docs.microsoft.com/en-us/aspnet/core/blazor>
* Jeff Fritz on Blazor: <https://jeffreyfritz.com/2020/01/whats-old-is-new-again-web-forms-meets-blazor/>
* Michael Washington’s Blazor Tutorials: <https://blazorhelpwebsite.com/>
* Chris Sainty’s Blog: <https://chrissainty.com/blazor/>
* Edward Charbeneau on YouTube: <https://www.youtube.com/user/Backslider64/videos>
* Blazor on YouTube: <https://www.youtube.com/results?search_query=blazor>