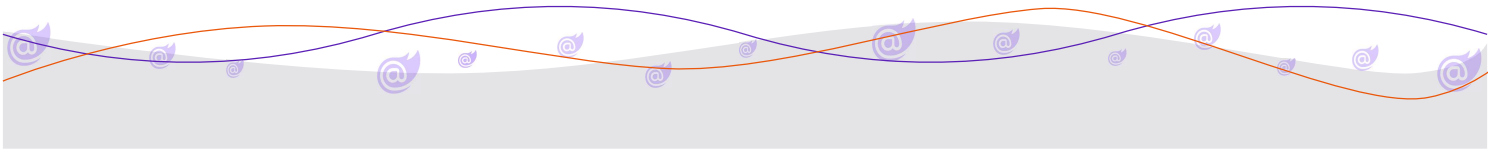
Chris Sainty

Blazor in .NET 8: Server-side and Streaming Rendering

READ TIME
11 MINUTES

PUBLISHED
22 AUGUST 2023



THIS POST IS PART OF THE SERIES: BLAZOR IN .NET 8.

- Part 1: Blazor in .NET 8: Full stack Web UI
- Part 2: Blazor in .NET 8: Server-side and Streaming Rendering (this post)

When it comes to modern web development, performance and user experience are at the forefront of every developer's mind. With .NET 8 introducing various rendering modes to Blazor, developers will be armed with an array of choices. Among these, server-side rendering and streaming rendering stand out, primarily due to their efficacy in delivering optimized web experiences.

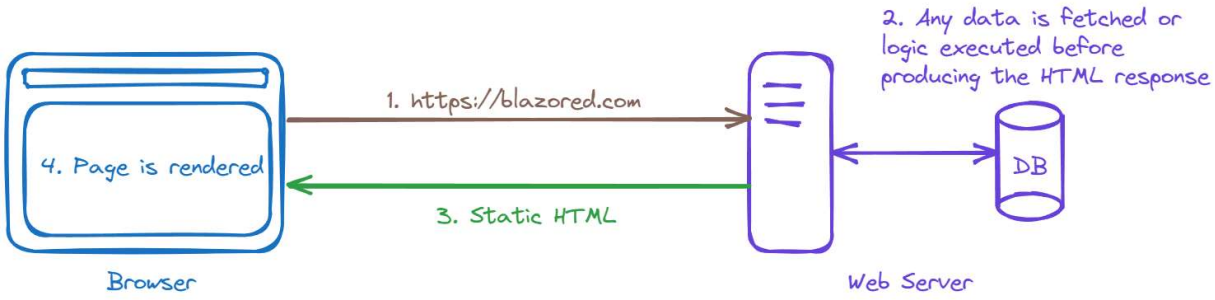
In this post, we'll delve deeper into these two modes and explore their significance in the new Blazor ecosystem of .NET 8.

Server-side Rendering: A Classic Powerhouse

In part 1 of this series I gave an overview of what server-side rendering (SSR) is, but now it's time to wade in a little deeper and explore this render mode in more detail.

With .NET 8 comes a new template for Blazor applications simply called *Blazor Web App*, and by default all components use server-side rendering. This is worth mentioning as you can think of the various render modes as progressive enhancements to each other, with server-side rendering as the base level and auto mode the most advanced.

Server-side rendered page components in Blazor are going to produce the same experience as Razor Pages or MVC apps. Each page is going to be processed and rendered on the web server, once all operations to collect data and execute logic have completed, then the HTML produced will be sent to the browser to be rendered.



There is no interactivity in this mode making applications very fast to load and render. This makes it an excellent choice for apps that deal with a lot of static data. I'm sure many line of business applications would fall into this category as well as online shopping apps where rendering speed is key.

Another boon of this mode is it's ability to allow excellent search engine optimisation (SEO). Applications just produce regular HTML, so there will be no

issues with web crawlers indexing pages, as can be the case when using single page applications (SPA).

But you might be wondering why would I choose to do that with Blazor when I could already use Razor Pages or MVC? You wouldn't be wrong for asking, here are a couple of reasons why.

First, Razor Pages and MVC don't offer very good options for building reusable components. Blazor on the other hand, has an excellent component model.

Second, when choosing either Razor Pages or MVC you are locked into a server-side rendering application. If you want to add any client-side interactivity, you either have to resort to JavaScript, or you could do it by adding in Blazor components. If you choose the latter, then why not just use Blazor for everything?

We've covered a lot of theory so far, let's put those learnings into action by creating and running a Blazor app using the new SSR mode.

Configuring server-side rendering

As previously mentioned, there is a new template for Blazor applications in .NET 8. This template removes the Blazor WebAssembly and Blazor server specifics and creates a standard starting point using SSR. To create a new Blazor app using this template we can use the following command via the .NET CLI.

```bash
dotnet new blazor -n BlazorSSR
```

That's it! We now have an application ready to go using server-side rendering. Remember, SSR is now the default mode for new Blazor applications and components. The other render modes are enhancements on top of it.

The key bits of configuration that make this work are contained in the `Program.cs` file.

CSHARP

```
using BlazorSSR;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorComponents(); // 👈 Adds services required to

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.MapRazorComponents<App>(); // 👈 Discovers routable components and
app.Run();
```

The `AddRazoreComponents` method will register the services needed to render server-side components in the application. The other bit of configuration is the `MapRazorComponents<T>` middleware. This takes in a root component that's used to identify the assembly to scan for routable components. For each routable component found, an endpoint is setup. Yes, you read that correctly. Each routable component is represented as an endpoint and this middleware generates those endpoints on application start up. Under the hood, this uses the minimal API technology and a new result type called `RazorComponentResult`. You can also setup your own endpoints to return the result of executing arbitrary Blazor components.

CSHARP

```
app.MapGet("/greeting", () => new RazorComponentResult<GreetingComponen
```

But that's something we'll delve deeper into in another post.

The other notable change in the new template is that there is no longer an `index.html` or `_Host.cshtml` page. The page markup that would normally be found in those pages is now all contained in the `App.razor` component along with the `Router` component. Previously, Blazor Server apps, or Blazor WebAssembly apps with pre-rendering enabled, would need to be hosted in a Razor Page as we couldn't route directly to a Blazor component. But with the new endpoint approach we just saw, that is no longer the case.

If you look at the `App.razor` component you will notice that there is a JS file referenced at the bottom.

```HTML
<script src="_framework/blazor.web.js" suppress-error="BL9992"></script
```

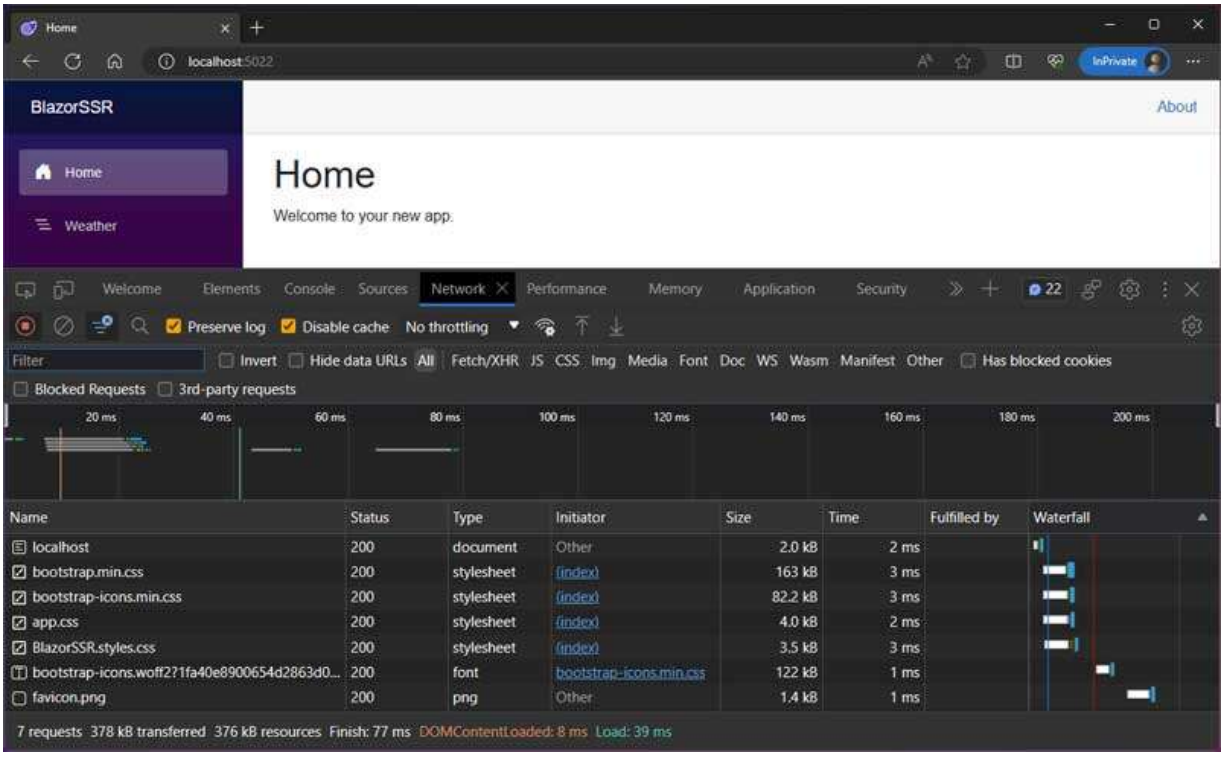
This is **not needed** for SSR. By default, the template can do streaming rendering as well and that's why the script is referenced. If you only want to use SSR then it can be safely deleted.

Running the app

Let's see what this looks like when we run the app. Before we do, we're going to make one other quick adjustment. The `Weather.razor` component is configured to use streaming rendering by default. Right now, we're just focusing on SSR, so we're going to delete the reference to the streaming rendering attribute at the top of the page.

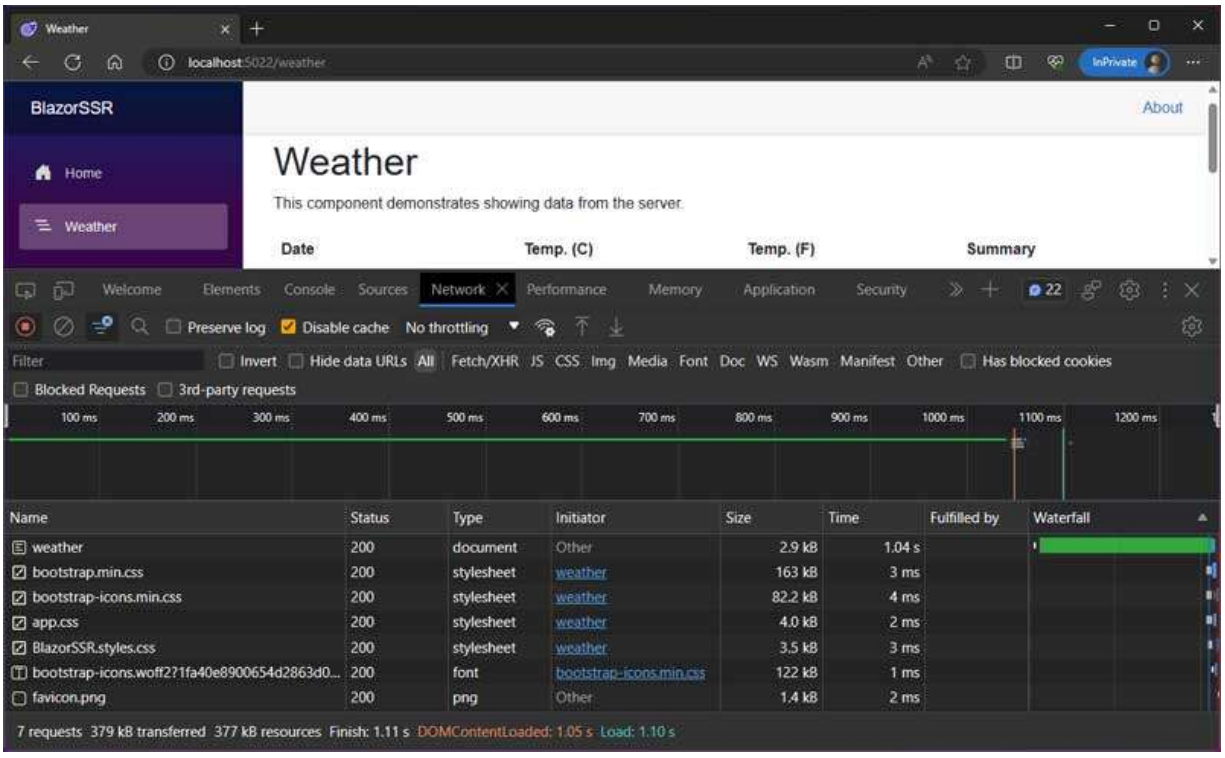
```RAZOR
@attribute [StreamRendering(true)]
```

We're now ready to run that application.



At this point, things don't look much different to a normal Blazor application. The key thing to spot is that there is no JavaScript being downloaded in this case. It's just HTML and CSS files.

Let's navigate to the Weather page and see what happens...



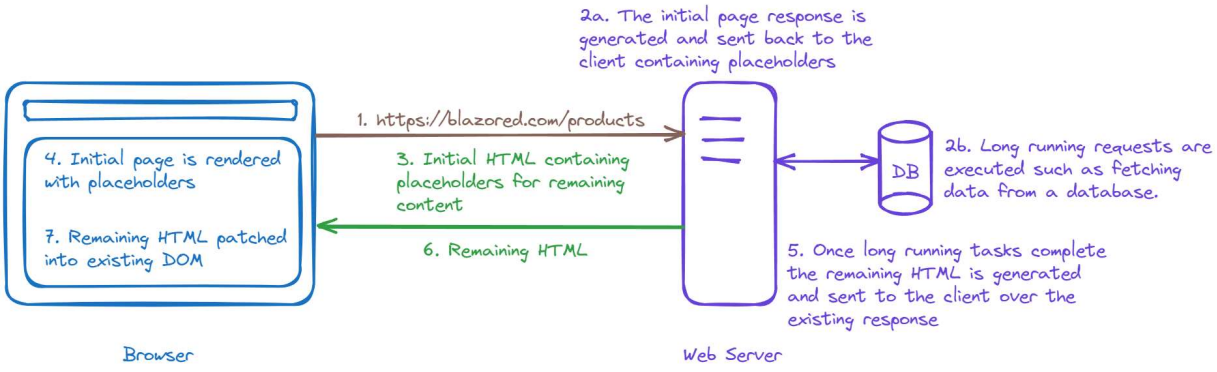
When we navigated there was a full page refresh. As you can see from the dev tools, all of the assets have been downloaded again, along with the new page we've navigated to. There has been no client-side navigation or routing involved, just classic request and response.

By the way, if you're curious about the load time and haven't looked at the code, the Weather page has a built-in 1 second delay to simulate getting data from a database. Something we'll leverage when looking at streaming rendering next.

Streaming Rendering: A Modern Marvel

As previously mentioned, render modes can be thought of as progressive enhancements over each other, the base mode being SSR. The next layer is streaming rendering.

Streaming rendering is a small enhancement over SSR. It allows pages that need to execute long running requests to load quickly via an initial payload of HTML. That HTML can contains placeholders for the areas to be filled with the results of the long running call. Once that long running call completes, the remaining HTML is *streamed* to the browser and is seamlessly patched into the existing DOM on the client. The cool thing here is that this is all done in the same response. There are no additional calls involved.



Streaming rendering is going to have the same types of use cases as SSR does. But really any application that is happy to be server-rendered overall, but might need a little help to improve loading times, is a good candidate.

What about caching? Can't we use that to solve the long running call issue on the server? Why do we need a new render mode?

Fair questions, the answer here is that not all long running calls can be cached. For example, a page might need to load live data from an external API, such as stocks or currency exchange rates. It's also perfectly reasonable that live data needs to be loaded from the apps database. In these cases, streaming rendering offers a much nicer experience for the user compared to waiting on a blank screen for a few seconds.

Configuring streaming rendering

In order to configure streaming rendering, we need to have the same basic configuration used for SSR. The two additional things required are the inclusion of the Blazor Web javascript file in `App.razor`.

```HTML
<script src="_framework/blazor.web.js" suppress-error="BL9992"></script
```

The other is the addition of the streaming rendering attribute on any page that we want to be streaming rendered.

```RAZOR
@attribute [StreamRendering(true)]
```

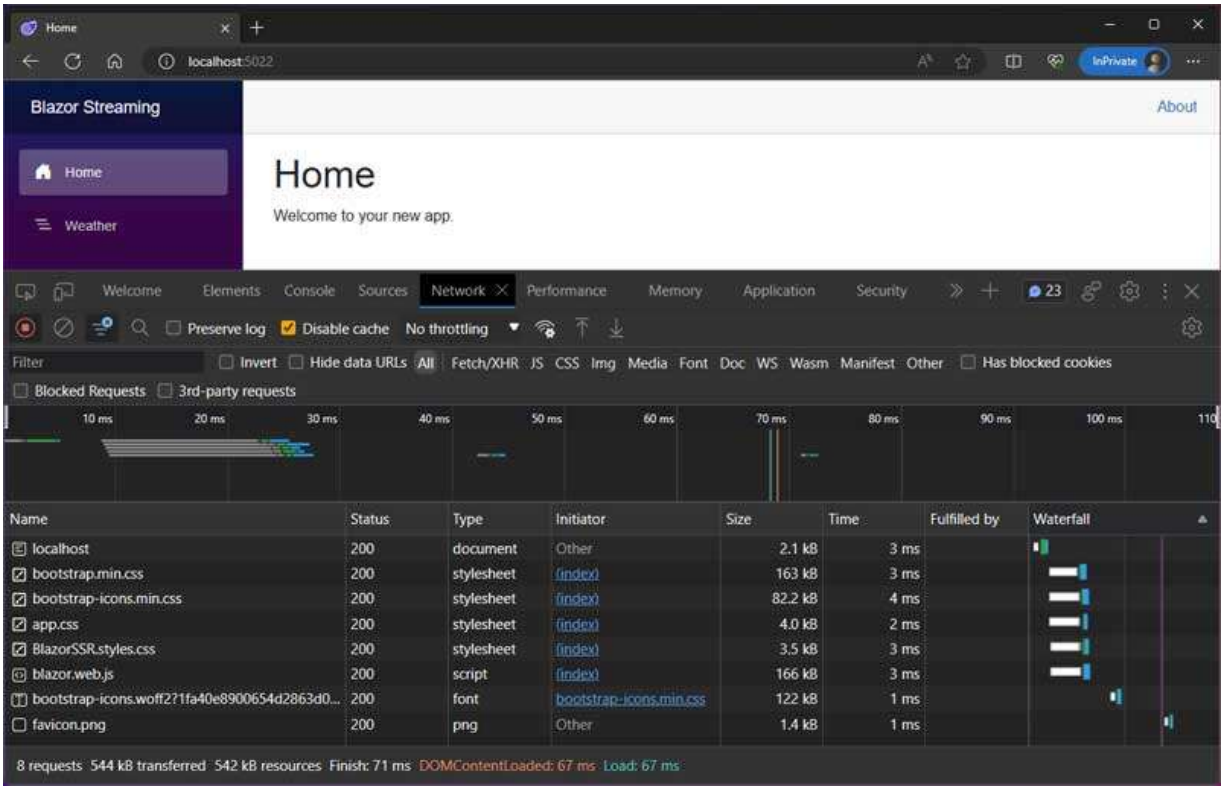
If you are keen to do as little typing as possible, the attribute is defaulted to `true`, meaning that just the presence of it is enough to enable streaming rendering. So the following is also valid:

```RAZOR
@attribute [StreamRendering]
```

As previously mentioned, these are both included by default in the new Blazor template with .NET 8. We just removed them when we were looking at SSR so we could see exactly how that mode behaved in isolation. So if you're following along, you can just add those two bits back into your existing project and you'll be good to go.

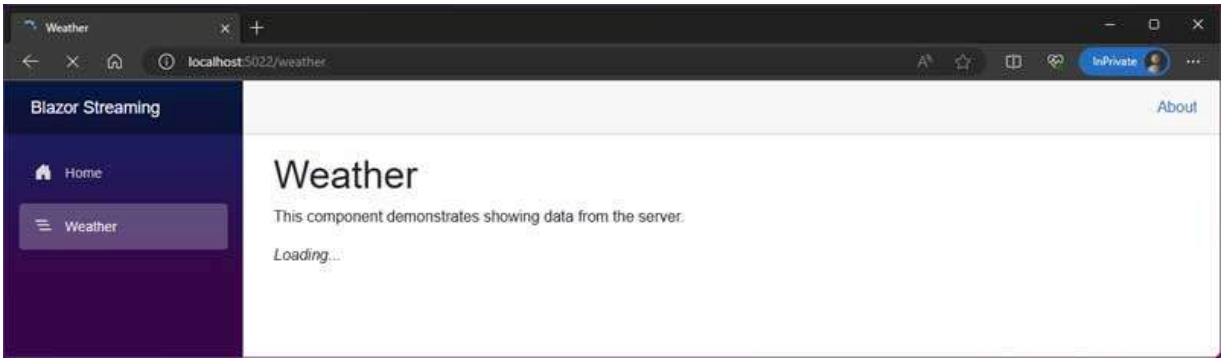
Running the app

When running a streaming rendered app, the initial load isn't much different to that of an SSR one. The key difference is the inclusion of the `blazor.web.js` file.



The differences start to show when navigating around. If we navigate to the Weather page as we did before, we'll see a couple of interesting changes.

The first is that we now see some placeholder text when that page first renders.



This wasn't something we saw before when using SSR. If we take a look at the code for the page we can see where this comes from.

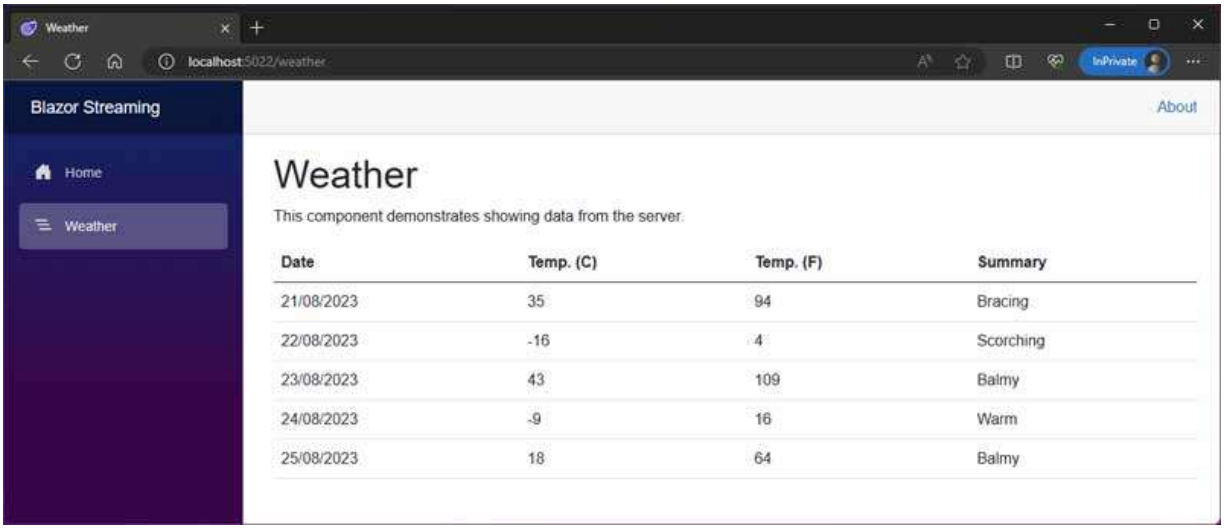
RAZOR

```
@page "/weather"
@attribute [StreamRendering(true)]

<PageTitle>Weather</PageTitle>

<h1>Weather</h1>

<p>This component demonstrates showing data from the server.</p>

@if (forecasts == null)
{
    <p><em>Loading...</em></p> @* 👈 Placeholder while data is loaded *
}
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Date</th>
                <th>Temp. (C)</th>
                <th>Temp. (F)</th>
                <th>Summary</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var forecast in forecasts)
            {
```

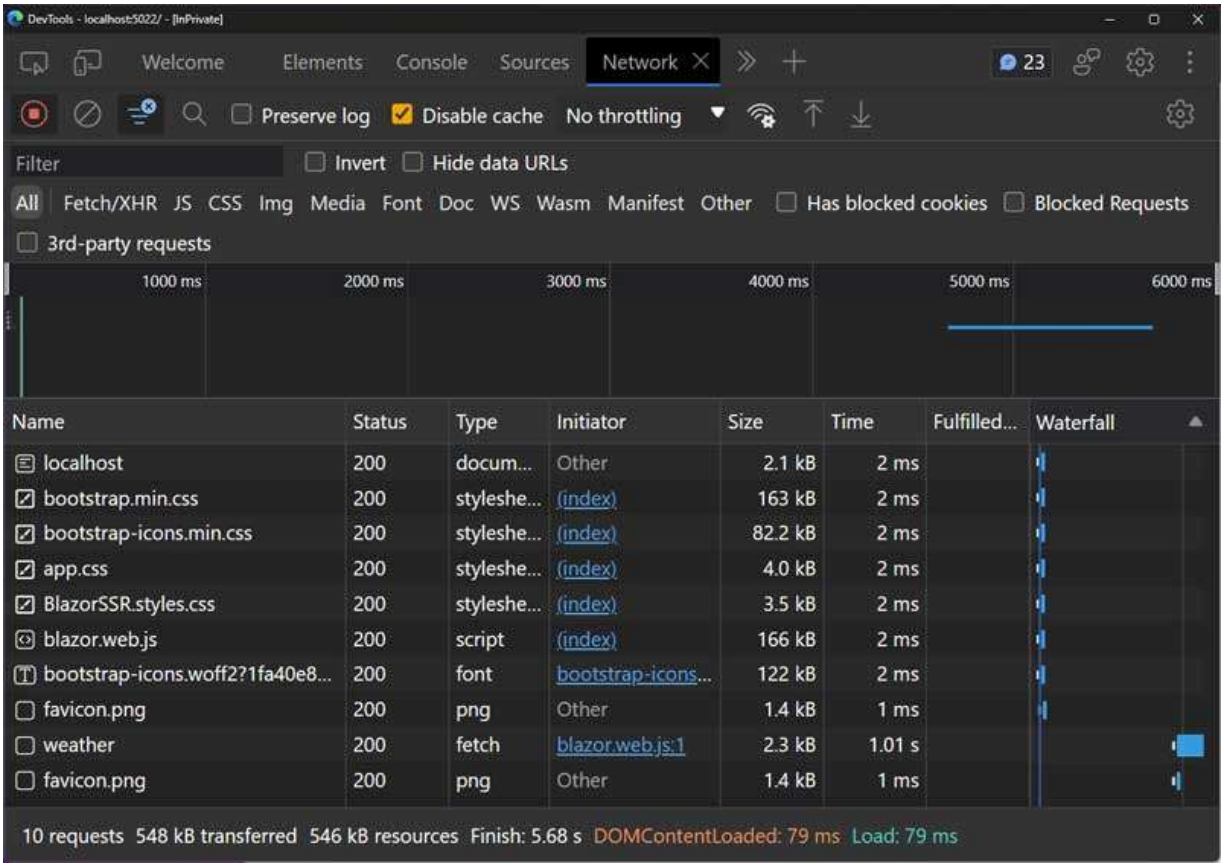
```
                    <tr>
                        <td>@forecast.Date.ToShortDateString()</td>
                        <td>@forecast.TemperatureC</td>
                        <td>@forecast.TemperatureF</td>
                        <td>@forecast.Summary</td>
                    </tr>
                }
            </tbody>
        </table>
    }

    @code {
        private static readonly string[] Summaries = new[]
        {
            "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy
        };

        private WeatherForecast[]? forecasts;

        protected override async Task OnInitializedAsync()
        {
            // Simulate retrieving the data asynchronously.
            await Task.Delay(1000);

            var startDate = DateOnly.FromDateTime(DateTime.Now);
            forecasts = Enumerable.Range(1, 5).Select(index => new WeatherF
            {
                Date = startDate.AddDays(index),
                TemperatureC = Random.Shared.Next(-20, 55),
                Summary = Summaries[Random.Shared.Next(Summaries.Length)]
            }).ToArray();
        }
    }
```

If `forecasts` is `null` then the page renders the loading markup. We can in the `OnInitializedAsync` method that `forecasts` gets populated after a simulated 1 seconds delay. Once this happens, the `else` condition of the `if` statement is

triggered. The table containing the forecasts is generated and streamed to the client where it's patched into the existing DOM.



The second interesting change can only be seen when checking the dev tools. When navigating to the Weather page we didn't do a full page refresh, it was actually a fetch request handled by Blazor.



Looking at the network tab, we can clearly see the request for the index page and the site assets (JS/CSS) are still present, even though *Preserve log* is disabled. We can also see that the request for the weather URL is of type `fetch` and was initiated by the `blazor.web.js` script. So what's going on here?

This is Blazor's enhanced page navigation in action. Once we include the `blazor.web.js` script, Blazor can intercept our page requests and apply the response to the existing DOM, keeping as much of what exists as possible. This mimics the routing experience of a SPA, resulting in a much smoother page load experience for the user, even though the page is being rendered on the server.

This enhanced navigation also works with a purely SSR application as well. It's not tied to streaming rendering in any way. You just need to include the `blazor.web.js` script.

Summary

In this post, we've explored two of the new rendering modes coming to Blazor in .NET 8: Server-side rendering, and Streaming rendering.

Server-side rendering, or SSR, is the new default for Blazor applications and components going forward. Using the classic approach of processing all logic on the server and producing static HTML that is sent to the client. This render mode is excellent for applications that need no client side logic. They just need to render pages quickly and efficiently.

Streaming rendering can be thought of as an SSR+ mode. If your application has to make long running calls when constructing a page, and those calls don't work well with caching approaches, then streaming rendering is going to be a great option. It will send the initial HTML for the page down to the browser to be rendered just as quickly as an SSR page. However, where the content from the long running call should be, placeholders will be rendered instead. Once the long running call has completed, the remaining HTML will be streamed down to the browser over the existing response where it will be seamlessly patched into the DOM.

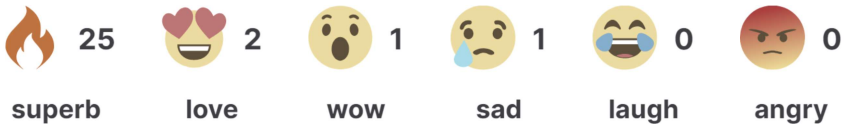
We also saw a perk of including the `blazor.web.js` script–enhanced navigation. This allows Blazor to provide a SPA style page navigation experience, even though pages are being rendered on the server.

What do you think to these two new render modes? Got any ideas what sort of applications you're going to use them for? I'd love to know your thoughts, leave a comment below.

Tags: BLAZOR .NET 8 SERVER-SIDE RENDERING STREAMING RENDERING

Thanks for reading!

What do you think?



71 Comments

LOGIN

Write your comment...

Newest

Alex 3 days ago

Great insights, Chris! In Canada, the federal government mandates using the 'Web Experience Toolkit' (WET) currently reliant on JQuery/Bootstrap for its websites. Those websites using Microsoft ecosystem opt for ASP.NET MVC (.NET Core or .NET Framework). Some using Razor Pages. For new project, still having to use WET and JQuery, would SSR Blazor + Streaming + JS Interop serve as a viable alternative? Are there potential obstacles that might necessitate reverting to traditional MVC or Razor Pages?

0 0 Reply

Chris Sainty 3 days ago

I think Blazor with SSR would be a great alternative to using MVC or Razor Pages. You would still get to use the great component model that Blazor offers and the mandated toolkit. The other great thing is that if in the future the requirement is lifted then you can move to the more interactive modes Blazor has with minimal disruption.

0 0 Reply

Alex 2 days ago

Thanks Chris.

Up to this point, this has been the primary factor preventing Blazor from being chosen when working with the Web Experience Toolkit open-source library (GitHub - wet-boew):

"The Blazor documentation cautions against using javascript to modify the DOM as Blazor tracks and manages the DOM. This can lead to DOM corruption causing application failures or security concerns. WET rely heavily on DOM manipulation."

Is this still valid and cause of concern with SSR Blazor/Net 8? (For instance, if the Web Experience Toolkit subtly alters the DOM without the developer's explicit awareness).

0 0 Reply

Chris Sainty A Alex 2 days ago

I don't believe it's valid when using SSR. In that mode there is no Blazor runtime on the client, you don't even reference the blazor.web.js script. You won't be able to use things like streaming rendering or the progressive routing but you wouldn't be getting those with MVC or Razor Pages so you're not losing anything.

0 0 Reply

wonsil 25 days ago

I love the idea of using SSR for simple sites. What I'd like to do is to deploy an SSR Blazor site to Azure Static Web Sites, but it does not appear that dotnet publish actually renders any html to a folder. It seems that only Blazor WASM sites work with Azure Static Sites. Since Azure Static Sites bill for data transfer, it would be nice to reduce that amount using prerendered html vs the whole dotnet runtime.

0 0 Reply

E **Echo** 1 month ago

Hi Christ

I have a question, in dotnet 7, the blazor server is equivalent to the streaming mode now ?

0 0 Reply

Chris Sainty 1 month ago

Hey. The short answer is no. Blazor Server is interactive and allows us to build those rich SPA style interfaces. Streaming mode only allows the initial page content to be streamed to the client. It's not interactive in anyway once it's loaded.

0 0 Reply

el vogel 1 month ago

Is there a way to have Blazor Server and SSR in the same app?

0 0 Reply

Chris Sainty 1 month ago

Yes. These render modes are all sort of pick and mix. You can have some components that are rendered server-side and you can have some that are fully interactive on the client. It's up to you.

0 0 Reply

J

James 1 month ago

As not an experienced programmer, I wonder where is the border line between "interactive" and "streaming". For instance if there is a grid coming from a slow db, what happens to pagination? Does clicking to the next block of records cause a page reload and then a stream of that next block? Or grids that are hidden and need to be loaded only when the user goes there, such as on a page with tabs.

0 0 Reply

Chris Sainty 1 month ago

Good questions.

For your first question, it would depend if the pagination code was included with the grid. If it wasn't then it would be rendered and the links on the paginator would be clickable by the user. So yes, potentially they would be able to click to a new page while the existing one was still streaming. That request would be cancelled and a new one opened for the next page.

Your tabs example would depend a bit on how the tabs were implemented. If they used some CSS tricks, then the data for the grid would be streamed down with the rest of the page, it wouldn't wait for the tab to be selected as it's only CSS hiding it anyway. If it wasn't CSS, then changing tabs would require a request to the server as there wouldn't be any client-side functionality to change the tab being displayed. So this would result in the grid data being streamed when the given tab was selected.

I hope that helps.

0 0 Reply

J

Jimbo 2 months ago

Just wondering what your opinion is on the value of SSR over using WASM?

0 0 Reply

Chris Sainty 2 months ago

They're totally different so it depends what you're trying to do. SSR is going to be really fast load time but you'll have no interactivity. So if I was creating a site with pages of static content, I'd choose that. WASM has a big initial download but once you're up and running everything is happening client-side. You also get the rich interactivity. So if I was writing an app, WASM might be a good fit.

0 0 Reply

Fallon 2 months ago

Very confusing!

Why is there no interactivity in SSR Mode? It's being served by a freaking server, why can't I call back to that server, without a websocket, just like any other server on the planet to execute code?

The interactive server opens a websocket connection, which I don't want. Am I missing something?

0 0 Reply

Chris Sainty 2 months ago

You can. But there is no Blazor functionality for that. You could add JavaScript if you wanted and have it act like any other static page. If you want interactivity through any of the Blazor mechanisms (Blazor Server or Blazor WebAssembly) then you use one of those render modes.

0 0 Reply

f **fauxbin** 2 months ago

Any plans to update your book?

0 0 Reply

Chris Sainty 2 months ago

The short answer is no.

The book was a massive project and took 2 years of my life. If I'm honest, I didn't really enjoy it and it was a massive slog which I've not really recovered from. I'm still finding it hard to get any joy from content creation since that project, as you can see from the lack of posts here since it was published. It's just made it all feel like work.

I also have a lot more going on in my personal life now, kids and a new house that needs a lot of work. I want to get back to blogging regularly again, at some point, but it's just not a priority right now.

0 0 Reply

Grant 1 month ago

Sorry to read you have no plans to update your book, although I totally appreciate your reason why. It really is a good book. Thanks for all your content. Like many out here I have found it very helpful.

0 0 Reply

Chris Sainty Grant 1 month ago

Thanks, Grant. That's really appreciated.

0 0 Reply

f **fauxbin** 2 months ago

Thanks. Just want to say that your book is really good one.👍

0 0 Reply

Chris Sainty f fauxbin 2 months ago

Thank you, that means a lot ☺

0 0 Reply

m **mathew** 2 months ago

Nice article. But in "production life" SSR is unuseful. Every web app needs pages with user interactives.

0 0 Reply

Chris Sainty 2 months ago

I think we can agree to disagree on this one. Not every page in the apps I've worked on requires interactivity. We've become obsessed with the idea that all pages need to be interactive when, in reality, they don't. A page that displays a table of data with paging, for example, could be statically rendered and use postbacks to load the next page or change filter settings.

Anyway, each to their own. You don't have to use SSR if you don't want to, that's the joy of having these options available to us as developers.

0 0 Reply

P **Ping** 1 month ago

I think change page or change filter settings is also a user interactive? Maybe a static home page doesn't need user interactive.

0 0 Reply

Chris Sainty P Ping 1 month ago

It's interactive, but that doesn't mean it needs to be client-side interactive or over a websocket. It can be handled via forms posts, it's what we did before SPAs came along.

0 0 Reply

D **Daniel** 2 months ago

I have another question that hopefully you can answer....

What happens if we have a page using SSR with StreamRendering, but within that page we include a component that uses Interactive Server rendering?

Would that work or would it break something?

0 0 Reply

Chris Sainty 2 months ago

As far as I'm aware, that would work.

0 0 Reply

D **Daniel** 2 months ago

Out of interest, say you in included jquery in a page using SSR and StreamRendering, when does document.ready fire? Is it after the final content is streamed to the page, or is it immediately as soon as the initial render is complete?

Purely academic reasons for asking this, not necessarily got a use case in mind

0 0 Reply

Chris Sainty 2 months ago

Good question. Not had time to look at that myself. If you find out please leave a comment and let me know!

0 0 Reply

AndrewA 2 months ago

Sorry for the stupid question. in previous .net versions, if I create a server side application and add task.delay, then it works the same way as with the new StreamRendering attribute, first the loading appears and then the content. The real question is, what has changed? why a new attribute?

0 0 Reply

Chris Sainty 2 months ago

Good question. Streaming Rendering is for static, server rendered sites. Not interactive ones. The key difference, using your example, is that with Blazor Server with Task.Delay you still have to have the constant SignalR connection. You don't need that with Streaming Rendering.

0 0 Reply

AndrewA 2 months ago

got it, thanks a lot! :)

0 0 Reply

previato 2 months ago

SSR seems amazing at first look and a good candidate to replace MVC web apps. That is what I trying to do, but I can't find a way to call a js function that is in a per page js file. Have you tried something with that?

0 0 Reply

Chris Sainty 2 months ago

What exactly do you mean by call a js function? How are you attempting to do that?

0 0 Reply

previato 2 months ago

For example, click on a button and make a simple update in the UI. <button onclick="update()">Update</button> is a custom js, but what I would like that is a per page js file, like a Index.razor.js which is loaded only when I am navigation on the Index.razor page.

0 0 Reply

p **ptr** 3 months ago

So the SSR + StreamRendering works like the "old" Blazor Server worked until now? Because today we can achieve the same behaviour in Blazor

Server without StreamRendering attribute.

0 0 Reply

Chris Sainty 2 months ago

Not quite. SSR + Stream Rendering doesn't have a persistent SignalR connection. It will steam the results of a long-running task at page load, but the page won't be interactive. With Blazor Server the page is interactive.

0 0 Reply

C **Chris Johnson** 3 months ago

One more gotcha that I've noticed... Once you go to a page where you're using a rendering mode that doesn't use a SignalR connection, all of your scoped services will disappear (along with any state data you might have saved in them). I suppose you can put that sort of thing in a database, but that seems extreme. Any suggestions for saving state information in a mixed rendering mode application?

0 0 Reply

D **D** 2 months ago

Sounds like it's by design. If you need to retrieve state later, you need to find a persistent store for it, regardless of the render mode.

0 0 Reply

dubik 3 months ago

Thanks for great article! Is there a way to call a Javascript after page is loaded (injected by blazor.web.js)? To setup listeners for instance.

0 0 Reply

Chris Sainty 3 months ago

Once a page is fully loaded it's just HTML and CSS. You can reference a JS file on the page and it will execute like it would on any other static HTML page.

0 0 Reply

Z **Zaspal Jsem** 4 months ago

Great post. Thanks for seamless explanation and nice visual representation of "what is going on".
I have few questions about WASM/auto mode.

1) When one uses the SSR or SSR+ (streamed rendering), the `@code{}` section always runs on the server. But when this component becomes

WASM, the c# code is expected to run on the client. Is that correct?

2) If the 1) is true. The weather component “generates“ weather data, which somehow simulates db access (for example). I guess this was the case with BlazorServer (I have never actually used it, just wasm).. When I switch to auto mode on weather component, this code (for db calls) gets into the client, which might be a security risk, I guess (or you just simple have no db there(??))... So what is the necessary change to get this component work on wasm?

3) Almost the same question as 2), but from different angle: How is the communication with wasm component supposed to look like? Currently (pure wasm) it is just API calls. Will there be any change to that? How that is gonna work on AutoMode? Api (OnInitializedAsync method) will be called on the server and then in wasm again? Or it will have enough brain capacity to realize the call has been already done on the server? Or this call will be delayed, delivered by SSR+, then wasm will be downloaded and just work as “pure wasm”? Or is there a new way of communication between wasm-server?

0 0 Reply

Chris Sainty 4 months ago

Hey Zaspal, glad you liked the post.

1) Yes.
2) You need to write components differently. They need to be decoupled from their data source so you can provide different implementations depending on where the component is executing. As an example, you could create an IWeatherService and have an implementation registered in the server project which gets the forecasts straight from the DB. Then register an implementation in the WASM project that gets the data via an API.
3) With auto mode, when a user first visits the app the component will execute on the server and the results sent to the client. In the background, the WASM assets will be downloaded and then on the next visit the app will load via WASM and the component will execute on the client.

Does that all make sense?

0 0 Reply

Auguste 3 months ago

So, like-a-magic it's going to decide direct db access or to go over HTTP? Strange. Where does the component live than? In the client I assume?

I see what you're saying, have the abstraction (interface) on the component but 1 server-side implemention & 1 client-side implementation of that service.

Do you think that's a real scenario people will need or is this just causing more confusion?

0 0 Reply

Chris Sainty Auguste 3 months ago

It's not magic. If the component renders on the server, it will use the service registered in the server project. If it renders on the client, it will use the service registered in the WASM project. Where a component renders is determined by what render mode you choose.

Anyone who wants to use the Auto render mode will need to implement things this way,

0 0 Reply

Auguste Chris Sainty 3 months ago

Thanks for the response, I didn't mean to sound so cynical, I often find myself on your blog for Blazor stuff.

I still don't see why to go for such an approach instead pre-rendering Blazor WASM or just using Blazor Server. A lot of confusion arose with these new (optional) possibilities.

0 0 Reply

Chris Sainty Auguste 3 months ago

With auto mode, I think it's better to think of it as 2 apps. A Blazor Server app that runs when a user first visits your site. This app also downloads the WASM version in the background ready for the users next visit. On the next visit, there is a WASM app. Does that help?

0 0 Reply

Z **Zaspal Jsem** 4 months ago

I understand now. It does make sense - with wasm, everything gets into client and a developer is responsible for handling data access in a correct way.
I am looking forward to work with .net 8 and Blazor, I am really glad I chose this tech.

Thank you for your content, Chris. Always helpful.

0 0 Reply

Boukenka 5 months ago *(Edited)*

Great post. I cannot wait to read the next one : *Blazor in .NET 8: Web Assembly-side and ...* ?

0 0 Reply

Chris Sainty 5 months ago

I think the next post will cover auto mode. I think people are pretty familiar with either Blazor Server or Blazor WebAssembly so those render modes probably speak for themselves. I'll probably still cover them at some point though for people completely new to Blazor.

0 0 Reply

Boukenka 2 months ago

I suppose you're waiting for the official release of .NET 8 before writing a new blog?

0 0 Reply

Chris Sainty Boukenka 2 months ago

No, a lot going on in my personal life. Just about priorities at the moment.

0 0 Reply

tharlab 5 months ago

thanks for notice "RazorComponentResult" and example snippet code "app.MapGet("/greeting", () ⇒ new RazorComponentResult<GreetingComponent>()); its exactly what i am looking for a week for my projeect, and return responds as expected..... cant wait for more your explanation and mind blowing idea abut that... now i have beautifull route for blazor, or maybe embed app.razor directly inside to get full component loaded, it more prety than custom route you explain here.... https://chrissainty.com/building-a-custom-router-for-blazor/

0 0 Reply

Khalid Abuhakmeh 5 months ago

"First, Razor Pages and MVC don't offer very good options for building reusable components. Blazor on the other hand, has an excellent component model."

Razor Pages and MVC have ViewComponents, TagHelpers, Render Actions, Partials, and HTML helper extension methods. In fact, you could argue it has a richer reuse model than what Blazor currently offers, which is currently just components and more components.

0 0 Reply

Chris Sainty 5 months ago

Well it depends whether more options = better.

You could argue that it's really confusing to do reusable components in Razor Pages and MVC as you need to know what you should choose between ViewComponents, TagHelpers, Render Actions, Partials, and HTML helper extension methods. Where as for Blazor, it's just components.

0 0 Reply

Khalid Abuhakmeh 5 months ago

True, but isn't this blog post about more options in Blazor? 😅

0 0 Reply

Chris Sainty Khalid Abuhakmeh 5 months ago

More places and scenarios to use it, and its simple and powerful component model. Not a 6th option for how to make something reusable. 😉

0 0 Reply

Khalid Abuhakmeh Chris Sainty 5 months ago

On a more serious note.

I am curious how component vendors will deal with the newer hosting models in Blazor. Server rendering has no mechanism to convert the interactivity of components into something it can ship separately from the rendered HTML (without reintroducing the overhead of WASM or WebSockets). You can see this in the newer templates, where the pop-out menu "breaks".

This means layouts cannot have any components with client-side behavior, or you lose the value of server-rendering. I can see this leading to confusion or component variants where the rendermode must be passed.

Do you have any particular thoughts on this issue?

0 0 Reply

Chris Sainty Khalid Abuhakmeh 5 months ago

Yeah, this is a burning issue for me, right now. I don't really have a good answer at the moment. But I share the concerns/thoughts you've expressed. Once I know me I'll get a post put together on it.

0 0 Reply

Khalid Abuhakmeh Chris Sainty 5 months ago

Awesome! Looking forward to what you find out.

I think the solution is what all Blazor developers fear it might be... "time to write some JavaScript". 😈

0 0 Reply

CodeFoxtrot 5 months ago

Thanks Chris! Love the description in terms of layers, SSR being the base layer, and streaming rendering being the next. Puts it all in perspective and helps me think more carefully (more clearly) when building something. .NET 8 is going to be revolutionary in a lot of ways, and for a new web project as a C# developer, I'm struggling to find a reason not to choose Blazor. Seems like we'll have a new king, and long may Blazor reign!

1 0 Reply

Chris Sainty 5 months ago

Glad the post made sense and made things clearer for you! 😀

0 0 Reply

A

Amjad 5 months ago

Ok, so I have a few questions. First, with existing applications written for WASM, how do we move to .NET8 auto mode? Do you see any roadblocks in doing so? Second, what are the use cases for the static rendering mode? With the auto mode, do we need the static rendering mode?

0 0 Reply

Chris Sainty 5 months ago

So moving to auto mode will require a few steps. Your components will need to be able to work when rendered on the server using Blazor Server and on the client using Blazor WebAssembly. You'll need to .NET 8 Blazor Web project to host the app but you'll need to keep the components in a Blazor WebAssembly project type. This is the choice the Blazor team made on how to know which components can render client-side. But I would suggest thinking hard about this and deciding if you need to do this and if you do, do you need everything to be on the client? You may do, but I think a lot of apps don't really need that but have done it anyway as we've just been building SPAs for so long now.

The use cases for static rendering, as I mentioned in the post, are probably more line of business apps. I know that pretty much every line of business app I've worked on in the last 5-10 years could have been server-side rendered with maybe the odd bit enhanced with some interactivity. Also apps like online shopping sites, blogs, anything with content that doesn't require client-side interactivity.

0 0 Reply

A **Angelo** 5 months ago

Wondering what the major Blazor component vendors (and open source component library developers) are going to do about these modes (SSR and stream rendering).

0 0 Reply

Chris Sainty 5 months ago

Good question. It's something I need to think about myself for the Blazored packages. But right now. I honestly don't have a good answer so I don't want to ramble. Once I know more I'll probably cover it in a post.

0 0 Reply

S **Steve** 5 months ago

Hi Chris,

Great post. One thing I am concerned about is it looks like web assembly components are sort of being "relegated" in favour of default server-side rendering. How would an existing .NET Blazor wasm project be upgraded

to a .NET 8 project? Do existing 7 wasm components have to be marked as wasm rendered components in 8?

Cheers.

1 0 Reply

Chris Sainty 5 months ago

Great question, Steve. So in .NET 8, as far as I can see, there will be 2 templates. Blazor and Blazor WebAssembly. The Blazor Server template will be dropped. So the quick answer is that WASM components definitely aren't being "relegated" in any way. You'll now have the choice to either have a purely WASM app or, using the new template, an app that can use multiple render modes, one of which is WebAssembly.

1 0 Reply

candritzky 5 months ago *(Edited)*

Hi Chris, thanks for this wonderful, detailed blog post. Just want to let you know that the " "adding in Blazor components" link is broken.

And there's a typo in "AddRazoreComponents".

0 0 Reply

Chris Sainty 5 months ago

You're welcome! Thanks for letting me know about the broken link. I'll get that's sorted 👍

0 0 Reply

by Hyvor Talk