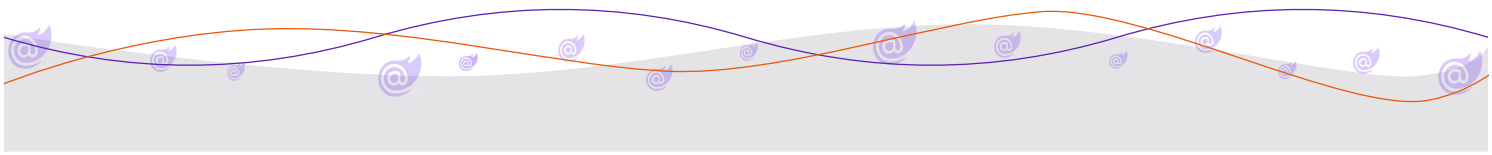
Chris Sainty

Blazor in .NET 8: Full stack Web UI

READ TIME
12 MINUTES

PUBLISHED
15 AUGUST 2023



SPONSORED BY

Telerik UI for Blazor – 100+ truly native Blazor UI components for any app scenario, including a high-performing Grid. Increase productivity and cut cost in half! Give it a try for free.

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

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

---

.NET 8 is bringing the biggest shake-up to Blazor since its debut. Remember the days when choosing a hosting model felt like being stuck between a rock and a hard place? Good news! Those days are almost over. With .NET 8, Blazor's WebAssembly and Server models will come together in a harmonious union, accompanied by some other exciting surprises.

Welcome to the first in a series where we're diving deep into the Blazor enhancements coming with .NET 8. Today, we're exploring the sea change in

how we'll construct Blazor apps, the fading relevance of the hosting model dichotomy, and a sparkling new concept termed *rendering modes*.

A quick history of hosting models

There was a time, very early in Blazor's experimental days, when hosting models weren't a thing. Blazor was going to run on WebAssembly and execute entirely inside the client browser, and that was that. However, around July 2018 server-side Blazor was announced, briefly known as ASP.NET Core Razor Components, then ultimately renamed to Blazor Server.

This new way of running Blazor had the application hosted on the server with clients connecting over a SignalR connection. All interactions on the client flowed through this connection to be processed on the server with UI updates sent back to the client where they were applied to the DOM.

And so, hosting models were born.

Currently, we have 4 hosting models for Blazor:

1. WebAssembly (web apps)
2. Server (web apps)
3. Hybrid (desktop & mobile apps)
4. Mobile Blazor Bindings (experimental)

The great thing? Components from the top three models are usually interchangeable. A component running in a Blazor WebAssembly app, can be lifted and run in a Blazor Hybrid application and vice versa (generally speaking). However, Mobile Blazor Bindings uses a different approach with components inspired from Xamarin Forms. This slight change of approach makes components written for the other hosting models incompatible with MBB.

The Hosting Model Dilemma

The problem that has been a thorn in the side of Blazor developers since almost the beginning: *Which hosting model to pick*. I'm specifically talking about building web applications now. Should it be WebAssembly or Server? What if I want to change further down the line? How much work would that be?

Of course, as developers, we're notorious for staking our claim and championing our chosen approach. Remember the age-old tabs vs. spaces feud? Yeah, same energy when it comes to hosting models!

The reality is that neither hosting model is perfect. Each of them have their pros and cons. Let's take a look.

Blazor Server

The server hosting model has some very compelling advantages:

- **Fast loading time** - It delivers a small payload to the client containing only initial HTML and JavaScript and from that point forward all interactions are sent back to the server for processing with diffs returned to the client and applied to the DOM.
- **Code Security** - All code stays on the server and away from prying eyes, it's never downloaded to the client.
- **Full .NET Runtime** - As a developer get access to the full .NET runtime. The application is running on the server and therefor you're not subject to any restrains.
- **Development Speed** - You don't need a separate API project as you can connect to resources such as databases or file systems directly.

However, it's not all roses. There are trade-offs to picking this hosting model:

- **Constant Network** - It requires a stable connection between client and server. Even a small blip in the connection will result in the dreaded attempting reconnect overlay which makes the application impossible to use.

- **Network Latency** - As all interactions are being processed on the server, they have to be sent from the client and a response awaited. If the client and the server are too far apart, this can result in perceivable lag for the user.
- **Cloud Costs** - As this is a server based hosting model, all clients are connecting to the server. This means the more connections the application has, the higher the specs of the server and more it will cost.

As you can see, there are some great pros, but the cons are not insignificant. Now let's take a look at Blazor WebAssembly.

Blazor WebAssembly

Here are some of the advantages WebAssembly offers:

- **Free Hosting*** - It can be hosted for free as there is no need for a server based .NET runtime. Blazor WebAssembly apps can be run from places such as GitHub Pages, Netlify, Azure blob storage or Amazon S3 buckets.
- **Scalable** - Following on from low hosting costs, is low scaling costs. Due to the app running entirely on the client, there can be little to no additional cost for onboarding more users.
- **Network Resilience** - As it runs completely on the client, it also works well with unstable networks. By adding a service worker, Blazor WebAssembly apps can run as PWAs with very little effort. There is even a setting for this when creating a project.

But what's the cost?

- **Download Size** - It's great that apps can run entirely on the client, but that does means that all the files need to be downloaded to the client in order to execute the application. This means the entire .NET runtime along with framework and application assemblies. Even on fast connections this results in a pause before the application comes to life.
- **Restricted Runtime** - As the app is running in the browser, there are certain restrictions on how the runtime can execute. This means that instead of

using JIT, which is the most common way .NET applications execute, Blazor WebAssembly uses an interpreter which is much slower, especially for CPU bound tasks.

- **Code Security** - This isn't so much an issue with the hosting model as an extra consideration for the developer. But as the entire application gets downloaded to the browser, dlls can be decompiled. Meaning that any sensitive logic needs to be put behind an API.

As you can see, both hosting models have some fantastic advantages, but neither is without it's compromises.

Wouldn't it be marvellous if we could cherry-pick the best of both worlds? Imagine harnessing Blazor Server's lightning speed and combining it with Blazor WebAssembly's resilience. And here's the twist: .NET 8 seems to be promising exactly that!

Full Stack Web UI with Blazor

.NET 8 is ushering in a new era where you won't be boxed into a single hosting model. Think flexibility, modularity, and personalization. You can tailor how each page or even individual component renders. It's like having your cake and eating it too!

Instead of committing to a single hosting model for your entire application, you can now mix and match based on the needs of specific pages or even individual components. For instance, a static contact page might use server-side rendering for performance, while a real-time dashboard might use the WebAssembly mode to leverage client-side capabilities. It's all about giving developers the tools to make the best architectural decisions for each scenario.

How is all this going to be achieved I hear you ask, the answer is a new concept called *render modes*. Let's explore each of them and see what they can do.

Server-side Rendering

Channelling the power of traditional web apps, this mode is reminiscent of how Razor Pages or MVC applications function. Server-side Rendering, or SSR, is when HTML is generated by the server in response to a request.

When using this mode, applications will load extremely fast as there is no work required on the client, and no large WebAssembly assets to download. The server is just sending HTML that the browser then renders. This also means that each request for a new page results in a full page load, as summarised in the illustration below.

At this point you might be asking what's that point of this when Razor Pages and MVC already exist? That's a good question, and the main reason is that neither of those frameworks offer a good story around building reusable components. Blazor has an excellent component model and this mode allows that to be leveraged for more traditional server-rendered sites.

I think another interesting angle to consider is that Microsoft are positioning Blazor as their preferred UI framework going forward. Over the last few versions

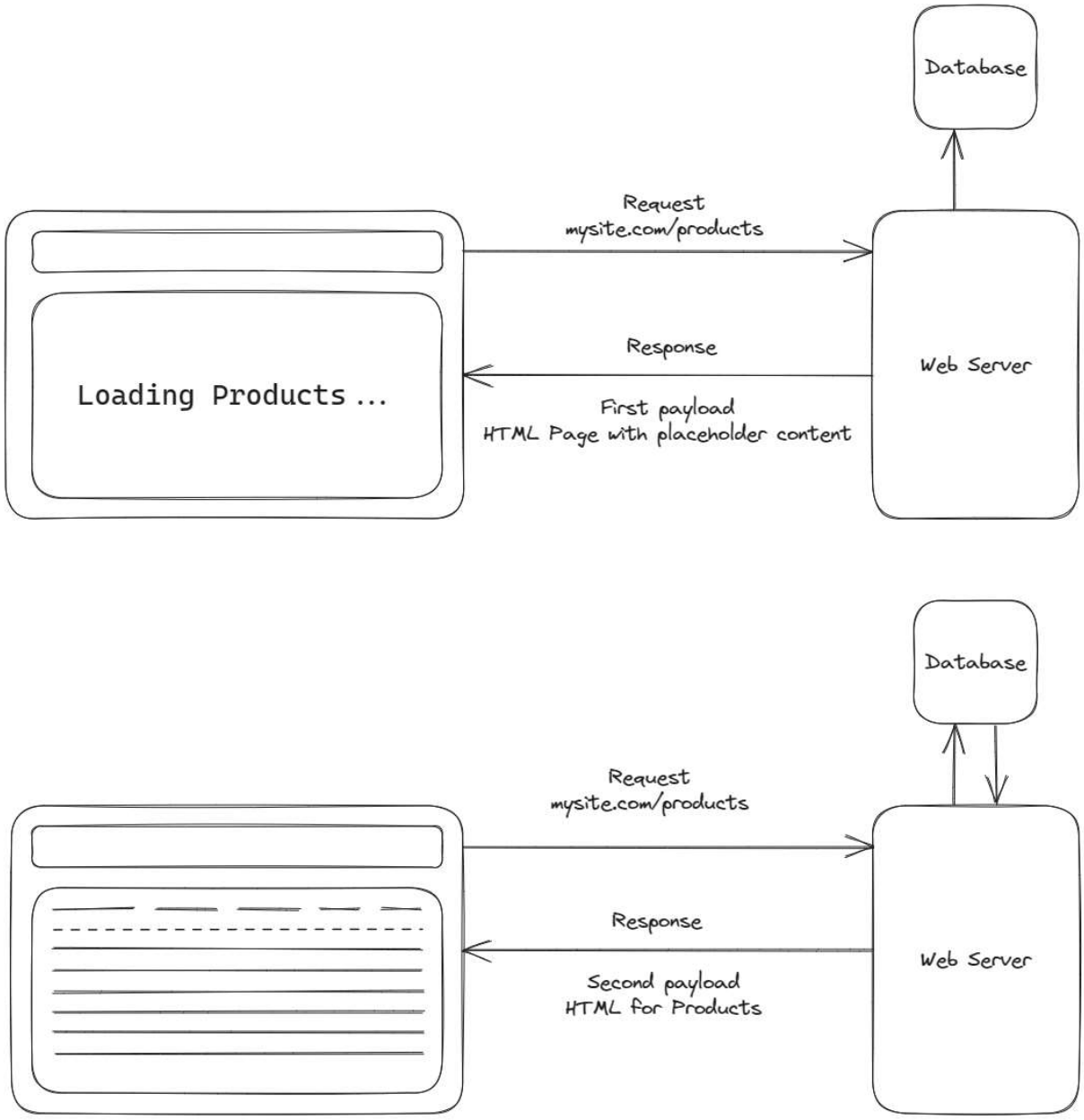
of .NET, there's been a lot of focus on making .NET more approachable to new generations of developers. Along this line, one criticism .NET has had is that there are so many options for doing things that it puts people off. This feel like an attempt to fix that. Learn Blazor and you'll be able to build any type of UI, web (both static and dynamic sites), mobile, and desktop. It's just a theory, but I think it has some legs.

Streaming Rendering

Think of this mode as the middle ground between server and client rendering. Let's pretend we had a page in an applications that needed to make an async call to fetch some real-time data–either from a database or another API. If we used the SSR mode we just covered, that would mean having to wait for that async call to complete before returning any HTML to the client. This could result in a delay loading the page. Also, there isn't any other interactivity in our page so using Server or WebAssembly mode would be a massive overkill. This is where streaming rendering comes in.

When using streaming rendering, the initial HTML for the page is generated server-side with placeholders for any content that is being fetched asynchronously. This initial response is then sent down to the browser to be rendered. However, the connection is kept open and when the async call completes, the remaining HTML is generated and sent down to the browser over the existing open connection. On the client, Blazor replaces the placeholder content with new HTML.

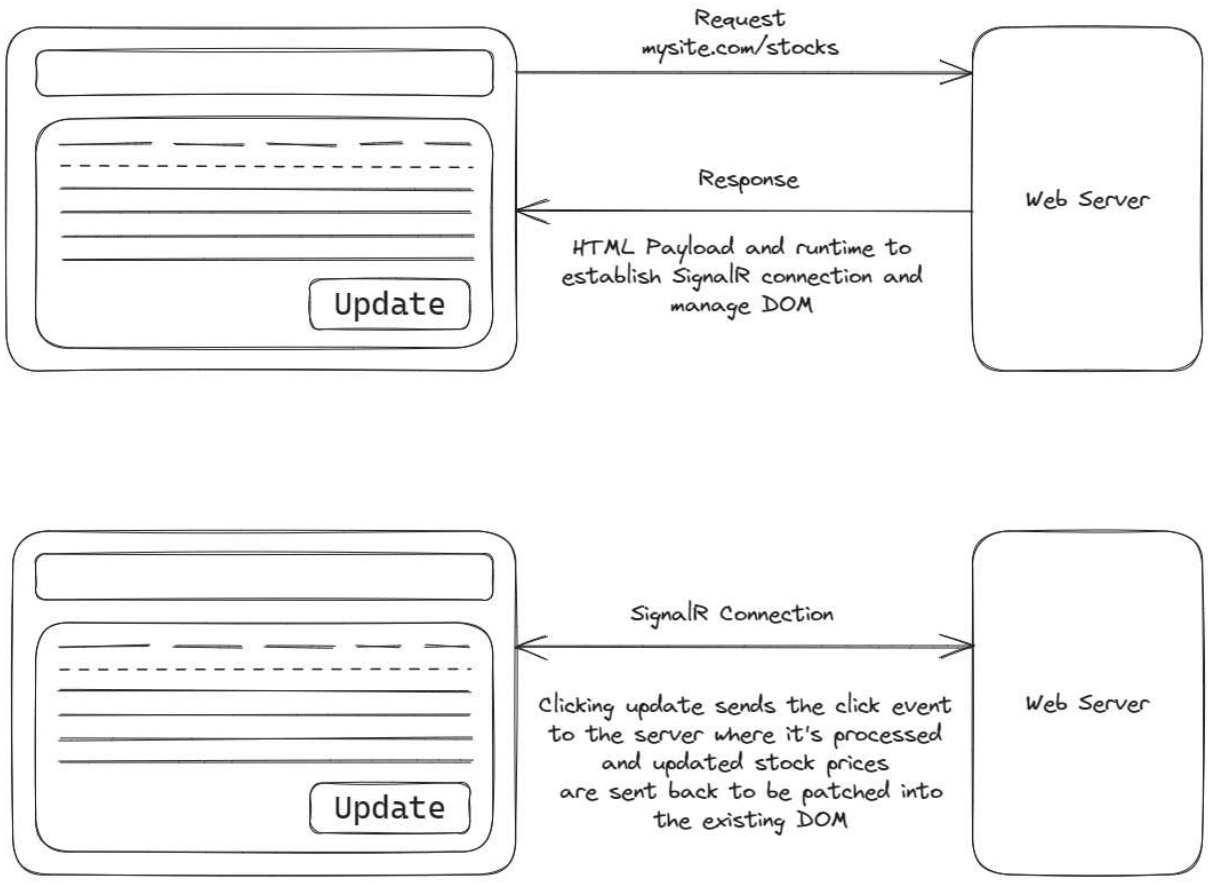
It's all about enhancing the user experience by minimizing wait times.



Server Mode

While this retains the essence of the classic Blazor Server model, its granular application is the standout feature.

When selecting this mode a page, or component, will be optionally pre-rendered on the server and then made interactive on the client via a SignalR connection. Once interactive, all events on the client will be transmitted back to the server over the SignalR connection to be processed on the server. Any updates required to the DOM will then be packaged up and sent to the client over the same SignalR connection where a small Blazor runtime will patch the updates into the DOM.

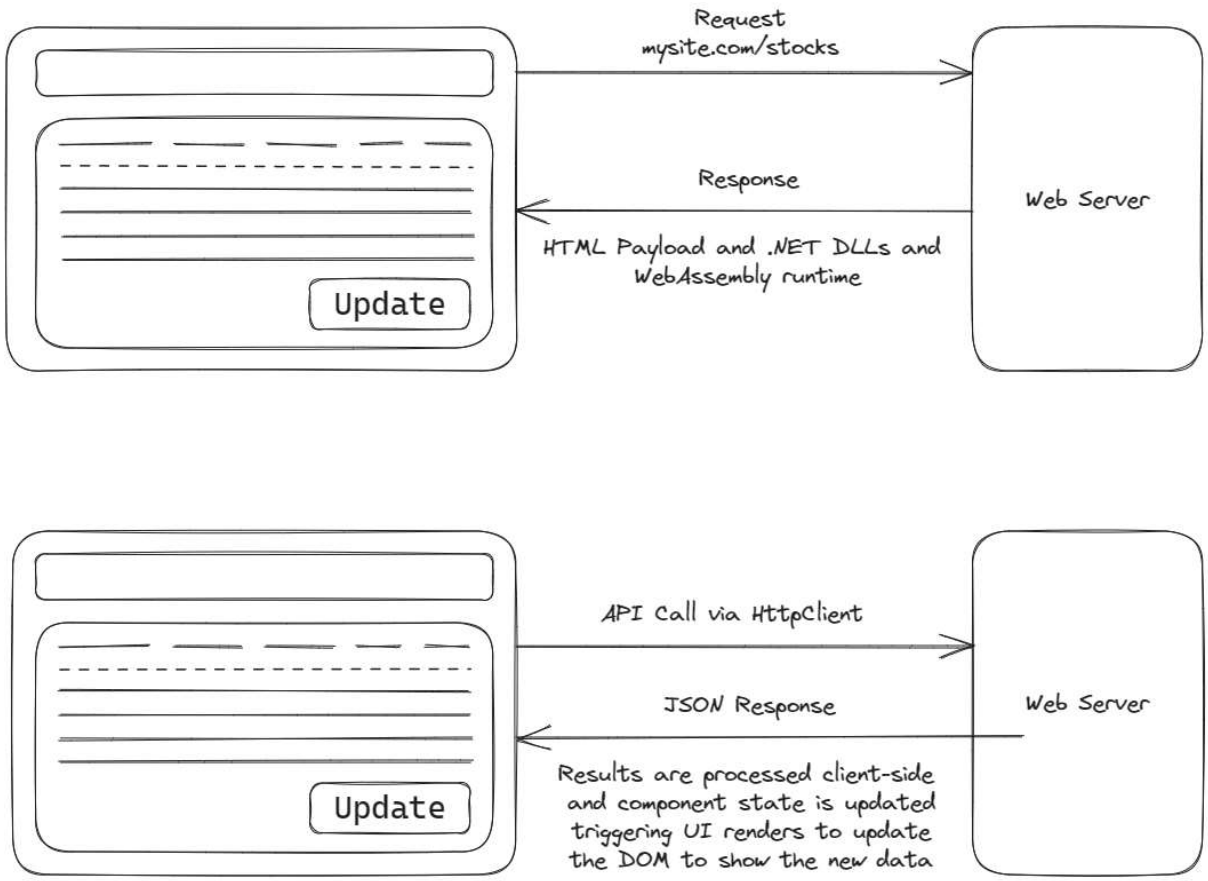


If you've been working with Blazor for a while, none of this is new, except that you'll now be able to decide this on a page by page or component by component basis!

WebAssembly Mode

The OG traditional SPA approach. This model is derived from the Blazor WebAssembly hosting model and fully capitalises on client-side capabilities, allowing C# code to run in the user's browser.

Using the same example as Server mode, the stock prices page would be downloaded to the client along with the various framework DLLs and WebAssembly runtime. Once on the client, it would be bootstrapped and the page loaded. Any API calls to get data would be made and the UI would be re-rendered as necessary to display any data returned.

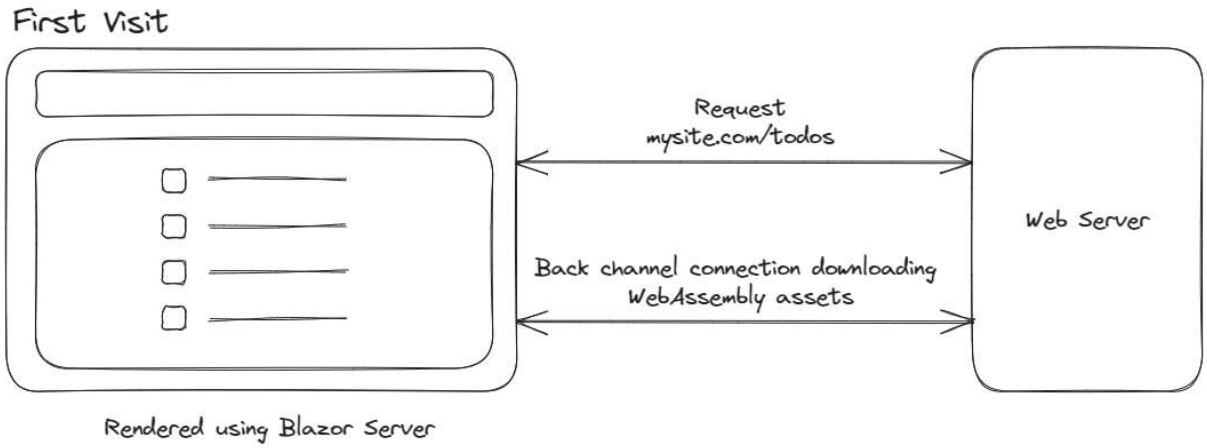


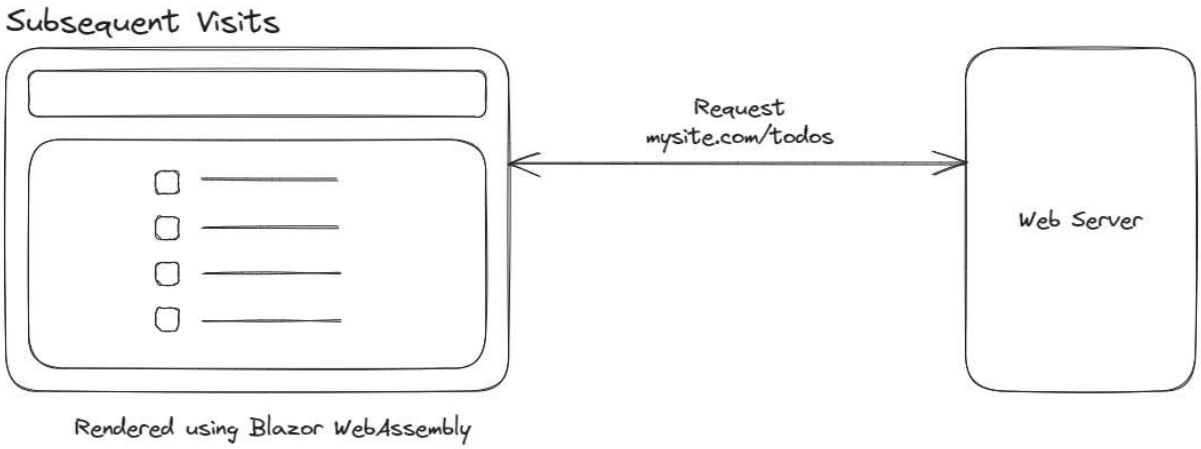
One thing to note about any components marked as `RenderMode.WebAssembly` is that they need to be referenced in a separate Blazor WebAssembly project to the main Blazor Web project. This is so that the framework can determine what code and dependencies need to be sent down to the client. If you want a component to run on both the server and the client, then the component should be placed in a Razor Class Library project that is references from the main Blazor Web project and the Blazor WebAssembly project.

Auto Mode

If there were an MVP among the rendering modes, this might just be it.

Since the early days of Blazor, developers have been asking for a way to combine the benefits of Blazor Server and Blazor WebAssembly. And with .NET 8 that request will become a reality. When setting a page or component to use Auto mode, the initial load of that component will be via server mode making it super fast. But in the background Blazor will download the necessary assets to the client so that on the next load it can be done using WebAssembly mode.





While this rendering mode will address the biggest pain point for developers when embarking on a new Blazor project, *what hosting model should we use?* There are no free lunches.

Auto mode will increases the complexity of applications. Every component marked with `RenderMode.Auto` will need to execute on both the server and the client. Meaning that there will need to be some form of abstraction in place if the component needs to fetch any data.

Other things that spring to mind are pre-rendering and security. However, at the time of writing, Auto mode isn't available as part of the .NET 8 previews so I've not been able to delve into these topics just yet, but as soon as I can, I will.

Summary

In this post, we've taken a first look at the major change coming to Blazor in .NET 8: Full stack web UI. Full stack web UI represents the biggest shift in the Blazor eco-system since the introduction of hosting models and is set to position Blazor as the "go to UI framework" for modern web applications built with .NET.

The new render modes give developers a huge amount of flexibility with their applications. We'll be able to control how our applications are rendered at a per-component level. And Auto mode addresses one of the major pain points for developers getting started with new Blazor projects. But it's not all sunshine and rainbows.

With great power comes great responsibility. Potentially having multiple render modes per page will create additional cognitive load for developers. If using Auto mode, developers will have to write server based code for fetching data, as well as traditional APIs as components will be run on both the client and server.

Personally, I'm extremely excited for what's coming, but I'd love to know what your thoughts are? Leave a comment and let me know.

Tags: BLAZOR .NET 8

Thanks for reading!

 Buy me a coffee

SHARE THIS POST

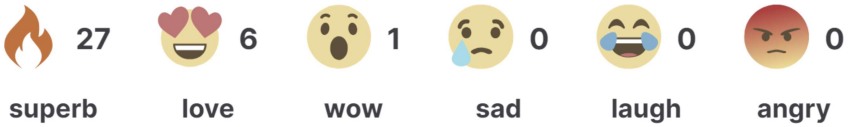
SUBSCRIBE TO MY NEWSLETTER

Get the latest posts delivered right to your inbox

you@example.com

SUBSCRIBE

What do you think?



31 Comments

LOGIN

Write your comment...

Newest

unixbob 2 months ago

I've seen the Microsoft pages on how to turn my .Net 7 Blazor WASM app into a .Net 8 app. We've promoted gRPC (gRPC-Web in this case) for speed client server performance. How does that work in .Net 8? If I'm going to take advantage of the Hybrid hosting mode for a better user experience do I now need to unpick all my gRPC code? I get the impression it's easy to go from a Blazor Server app to a Blazor Hybrid app. But less so from WebAssembly to a proper Hybrid app.

(not expecting an answer, just highlighting there could be better guidance on how we're supposed to upgrade apps to properly take advantage of .Net 8)

0 0 Reply

Steve 2 months ago

Hi Chris,

Any plans to update Blazor in Action with .NET 8, particularly Static SSR and Blazor United? Would be really looking forward to that!

0 0 Reply

Chris Sainty 1 month ago

I've answered this on another post in this series in more detail. But the short answer is no.

0 0 Reply

JMA 3 months ago

I'm an old webform coder, I know, I know, very dated, but it worked pretty well for in house applications. Also years ago I coded in C and Java. I've grown to love C# and dot net, developed an early Blazor app still in use.

.Net 8 Blazor is everything I've hoped for. Your posts are succinct and appreciated. Bought your book when it first came out and found it useful. Keep it up!

0 0 Reply

Chris Sainty 3 months ago

Thank you for the kinds words. I also built WebForms apps at the start of my career and, as you said, for better or worse they got the job done and were very productive to build. Great to hear you're excited for Blazor in .NET 8. I feel like it should make Blazor appeal to a wider audience.

0 0 Reply

F **FKM** 2 months ago

i still have my asp.net 2.0 web forms running in production now... when I moved to MVC i had nightmares for months..haha...

0 0 Reply

B **Basil Thomas** 4 months ago

Thanks for the excellent article!!! Very well written but...

I have a WinForms UI app that I want to upgrade to .Net 8 from old .net framework 4.7
which is used for real-time trading. Works very well but I want my system completely running on .Net 8.0 now.

Blazor Hybrid really looks the closest to what I currently have and I have sadly resided to learning HTML5, CSS and Javascript.

I am still quite uneasy about running a real-time trading app in the browser but would love to use Blazor Hybrid in WinForms even if the coding will be quite up hill though I do prefer C# over Javascript.

Can you give complete info/article using Blazor Hybrid with WinForms??

0 0 Reply

Chris Sainty 3 months ago

So you wouldn't be running it in a browser with Blazor Hybrid, it would just be rendered to a Web View. The code is all executing on .NET, it just uses web tech to render the UI. Does that make sense?

0 0 Reply

P **Paul Duggan** 5 months ago

Great post Chris - keep 'em coming - I'm 2/3 of the way through Blazor in Action - do you think it holds up ok with these changes in .NET 8?

0 0 Reply

Chris Sainty 5 months ago

Thanks Paul. Hope you're enjoying the book. The content in the book is most definitely still valid, nothing in there has gone away or become irrelivant. But obviously it lacks any information on render modes and the new approaches introduced in .NET 8.

0 0 Reply

Willem Meints 5 months ago

How badly is this going to break my upgrade to .NET 8? I hope there's going to be something in the upgrade assistant?

0 0 Reply

Chris Sainty 5 months ago

What hosting model does your app currently use? What would you want to use from the .NET8 changes?

0 0 Reply

CodeFoxtrot 5 months ago

Awesome Chris, thanks for taking the time to put this piece together.

I still consider myself a rather new ASP.NET Core developer, going on about 4 years now. I came from the desktop backend— Console, Windows Services, WinForms, WPF, always avoiding the web programming and Azure until about 2019.

Learning ASP.NET Core (beginning anyway) from Tim Corey's overview course, Blazor immediately caught my eye. While I've come to appreciate MVC, and the likes of GET-POST-REDIRECT, Blazor allows you to focus on content, pages and components, and less on the semantics— though admittedly I adopted MVVM during my WPF days, and despite Blazor not directly supporting ICommand, I still code Blazor with MVVM principles, because I just can't go back at this point.

Anyway, one of the most confusing things about Blazor is pre-rendering. For the newb developer, and I know because I was one of them, while Blazor is attractive, there are several nuances— hosting model, firm understanding of server vs client, pre-rendering, the whole StateHasChanged() circumstances, including when to await Task.Delay(1), as well as familiarity with page lifecycle methods, and what to do when.

So I'm confused with the streaming rendering, as whenever I have an async method that loads data, I'm doing this in OnAfterRenderAsync(), so I can load the rudimentary page with OnInitialized[Async](). Usually I render a wait spinner or loading method, which will then go away and display the data. Seems with streaming rendering, I can do the whole page server-side, and move my await for the data to OnInitializeAsync, but lose the wait spinner? This is the part that's confusing... did my over-thinking of the lifecycle methods and gracefulness of a wait spinner come back to bite me? Of course I want to use SSR whenever possible, and reduce SignalR usage.

So that's my dilemma that I look for leaders/MVPs like you and Tim Corey to reconcile— SSR with page lifecycle methods, pre-rendering and wait

SHOW MORE

1 0 Reply

Chris Sainty 5 months ago

Loads of interesting stuff in there and I have a few questions/observations.

So it sounds to me from what you've described that things might have gotten a little overcomplicated in your code. Using things like

`Task.Delay` shouldn't be necessary and, in my experience, show a problem in the design of the code.

I'm also curious why you choose to load data in `OnAfterRender` rather than `OnInitialized`? Generally speaking `OnAfterRender` is used for JS interop code.

Pre-rendering pre .NET 8 is just a way to deliver a complete page to the client quicker and has the biggest use when you need to have good SEO for an application. Honestly, if a site doesn't need SEO I wouldn't use pre-rendering as it doesn't offer much value outside of that use case.

Streaming rendering is an interesting one. As I said in the post, it's a bit of a SSR+ mode. The different is that instead of a page being built and rendered on the server and then sent down to the client once complete. Blazor will built it without await for any async calls to complete and send that markup to the client and then deliver the remaining markup once the async calls finish.

The common example would be a page that renders a table of data. Let's say it needs to get that data from a few different sources and it could take a bit of time. By using streaming rendering, the initial page load would be everything on the page except the grid, as the calls to populate that haven't finished yet. Once they do finish the markup for the grid is built and sent down to the client where it is patched into the existing page without having to do a full page reload. But the page is still a static, server-rendered page.

I don't know if that helps at all. If not, don't worry. I'm going to be covering these concepts in great detail in future posts, I'm also doing some YouTube videos as well as I think some of this stuff is just easier to see working rather that statically explained in text.

0 0 Reply

CodeFoxtrot 5 months ago *(Edited)*

And for the record, I never use pre-rendering. I've had folks tell me, *ohh don't worry, it's only for the initial load of the site*. But that's not entirely true. You can refresh the site, or navigate directly to any route. Thus every page has to position handling pre-rendering, so it's easier to just say, don't use it.

That and, how does pre-rendering now mash up with/against SSR for .NET 8?

0 0 Reply

candritzky 5 months ago

@CodeFoxtrot I had a look at your sample and the reason why the spinner doesn't work from OnInitializeAsync is that you hide the spinner in LocationChanged. That's because NavigationManager.LocationChanged is raised quickly after OnInitializeAsync.

Remove the Hide() call there and the spinner will be visible. But then of course, you will have other problems when navigating away while the previous page is loading...

Best regards from another Chris

1 0 Reply

CodeFoxtrot candritzky 2 months ago

THANK YOU

0 0 Reply

Chris Sainty 5 months ago

Pre-rendering isn't a thing with SSR as the whole page is rendered server-side and delivered to the client in the same way Razor Pages, MVC or even a regular static HTML page would be.

2 0 Reply

Ayo Dahunsi 5 months ago

Nice article!
Do you have any simple CRUD sample/code that demonstrates this... not the Weather example. There have been many articles, even youtube videos showcasing this .NET 8 Blazor enhancement but none with any "realistic" code. This article though, is one of the best read. Even a TODO app with 2 tables will suffice.

0 0 Reply

Chris Sainty 5 months ago

There isn't any example code for this post as it's an overview. But there will be plenty of examples in other posts.

0 0 Reply

Ayo Dahunsi 5 months ago

👍

BTW. I really enjoyed Blazor In Action... It is a very good book. Any plans for 2nd edition? So much has changed in Blazor

0 0 Reply

Chris Sainty Ayo Dahunsi 5 months ago

I'm glad you liked the book. There aren't any plans at the moment but that doesn't mean there wouldn't be in the future.

0 0 Reply

Craig 5 months ago

I've been developing both Server side and Wasm Blazor sites. One of the big issues has been making a site SEO friendly (simple Authorisation is the other). I'm hoping for some clear support on that topic in Blazor 8. Looking forward to your articles.

0 0 Reply

CaseySpaulding 5 months ago

I just add the meta data to the top of the page.

0 0 Reply

Chris Sainty 5 months ago *(Edited)*

What's the SEO issues you've been having? Do you use pre-rendering?

0 0 Reply

J

Jlm 5 months ago

Very interesting reading. Have been using Blazor for 4 years and find your teachings excellent

0 0 Reply

Chris Sainty 5 months ago

Thank you. Glad to have helped

0 0 Reply

C

Casey 5 months ago

Great post. Can't wait for it!

0 0 Reply

Chris Sainty 5 months ago

Thanks, Casey. Same here!

1 0 Reply

CaseySpaulding 5 months ago

Do you think this will prevent the lost connection to server thing when blazor server times out?

0 0 Reply

Chris Sainty CaseySpaulding 5 months ago

If you don't use any components with RenderMode.Server, then yes. But that's only because there would be no SignalR connection 🙂.

If you choose to use the Server render mode then you will always have the potential for that message.

1 0 Reply

by Hyvor Talk

© 2017-present Chris Sainty. All Rights Reserved.