№ 1 (6), 2024
JOURNAL OF INFORMATICS AND INNOVATIVE TECHNOLGIES
ISSN: 2682 – 9517 (print) ISSN: 2683 – 0930 (online)
INSTITUTE OF INFORMATICS AND INNOVATIVE TECHNOLOGIES

Render Modes of .NET Blazor Technology

Vladimir Sulov
University of Economics – Varna
Varna, Bulgaria
vsulov@ue-varna.bg; ORCID ID: 0000-0003-0301-5665

***Abstract*. With its latest version, Microsoft's .NET Blazor technology offers several render modes when creating web applications. This paper studies the different modes, their specifics, advantages, and drawbacks to help developers choose the proper one for their needs.**

Keywords: .NET, Blazor, web applications, render modes

I. Introduction

Microsoft's .NET platform is one of the popular choices for developing web applications [1]. It consists of a framework, servers, development tools and languages [2, 3].

The platform and the .NET Framework were announced in 2000 and the first version was released in 2002 [4]. Gradually it included tools to develop for different targets: desktop, web, mobile applications, including games, Internet of Things (IoT) apps, etc. [5, 6, 7].

In 2016 Microsoft released a new primarily web-targeted framework called .NET Core with better optimization and multi-platform support [8]. In 2020 the .NET Framework and .NET Core were unified and simply called .NET as they support both legacy and new features.

In 2018 a new technology and project type became part of .NET called Blazor. Blazor allows web developers to create rich web applications without the need to learn JavaScript or JavaScript-based frameworks but instead use C# both for the server- and client-side [9].

Blazor is component-based, where a component is a portion of content, user-interface, and logic. Components could be nested or could be an entire standalone page.

Currently in the latest version 8 of .NET, Blazor is improved and offers several types of rendering modes: static server, interactive server, interactive WebAssembly, interactive auto [10]. A vast improvement of this version is also the feature that render modes could be applied independently to components, including usage of several different render modes in a single application. The aim of this paper is to describe the different modes with their specifics and by analyzing them help developers choose and utilize the best one or combination of them for different types of applications and scenarios.

II. Static Server Rendering

When using static server-side rendering (SSR) after the client requests a web component/page, the server generates the necessary HTML/CSS and sends it to the client's browser. Afterwards no interaction with the server is possible so the user can only browse the page and potentially jump to a new page (including by clicking on a link on the statically-rendered page). The model is shown on Fig. 1.

Of course, the typical scenarios would include brochure like, informational only pages or even entire websites, or about, contacts, and similar pages in an otherwise interactive content site.

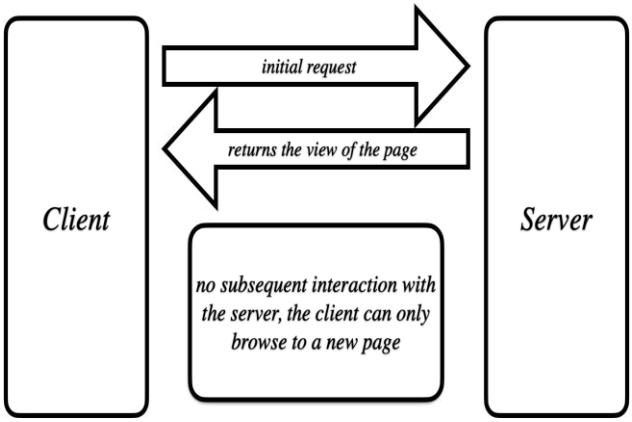

Fig. 1. Staric server rendering
Source: Own Elaboration

The functionality looks limited, but this is not completely so because, even though there is no interactivity after the initial load, the page itself could be dynamically generated. Let's consider a few examples.

In an online store, a products search results page could be implemented using static SSR. The server can extract from the database the matching products, then list a name/photo/description of each with a link to a products details page. This could be served to the client and does not need any interactivity apart from the user being able to click (open) the respective details pages which is supported using basic/static HTML.

Each product's page in theory could also be statically rendered by one-time initial extraction from the database but would probably need a different mode if it is to integrate functionality like adding to a shopping cart.

III. Interactive Server Rendering

Interactive server-side rendering provides the richest possible client experience and developer functionality, combining server and client-side interactivity.

Initially the server generates the component/page and sends it to the client. A real-time connection is kept active so when the user interacts with the component and content needs updating, the server sends the new information, which in turn leads to changes in the active view. The process is shown on Fig. 2.

Any kind of interaction is possible, including client-only logic, or actual two-way information exchange with the server, i.e. dynamically loading and/or updating information from/to a database.

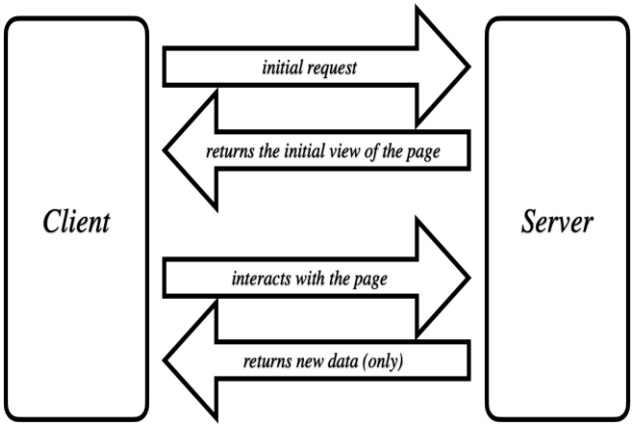

Fig. 2. Interactive server rendering
Source: Own Elaboration

Although interactive SSR mode can be used for any purposes, it does have its drawbacks. A connection with the server is always kept, even in cases where client-only or even no further interaction is needed. Therefore, if no interaction is needed, it is advisable to use the already described static SSR mode, while if only client-side interaction is needed (but only for this purpose), the further described two modes could be used.

IV. Interactive WebAssembly Rendering

Interactive WebAssembly is a client-side rendering (CSR) mode. WebAssembly is a portable binary-code format which is supported by most major current browsers and allows for very fast, almost native speed, execution on the client side [11]. WebAssembly was developed jointly by W3C, Mozilla, Microsoft, Google, and Apple.

When using interactive WebAssembly render mode, when the user requests the respective component/page, the entire .NET runtime and the app's component/page code/bundle are downloaded to the user's machine and afterwards the page is rendered, and the app runs entirely in the client browser without any subsequent connection to the server.

Potential drawbacks are the slower initial loading time (while downloading the .NET runtime and the page's code) before the user sees anything, the greater load on the client's hardware, and the lack of a connection with the server afterwards. So, for example, if database access is needed, it could be done indirectly through creating and using a web service as a separate server-run part of the solution which makes things a bit

more complicated for the developers. The process is shown on Fig. 3.

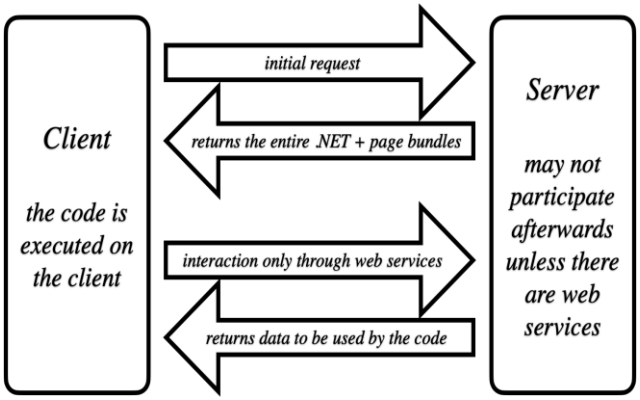

Fig. 3. Interactive WebAssembly rendering
Source: Own Elaboration

On the other hand, if no server connection is needed at all, the entire app could be hosted on any server, including a static one which could greatly simplify distribution and reduce hosting costs.

V. INTEACTIVE AUTO RENDERING

As mentioned above, a potential disadvantage of the interactive WebAssembly render mode is the long time to render the initial view of the page/component. In order to overcome this drawback, the latest version of Blazor offers interactive auto rendering. In this mode, when the user tries to load a page/component, the server first sends the initial view of the page/component in pure HTML/CSS so it could be rendered fast and afterwards, while the user browses the page and decides what to do, the complete .NET and page/component bundles are downloaded. Therefore, after the successful download, no waiting time is necessary to perform interactive actions.

As with interactive WebAssembly rendering, further connection with the server is not supported directly, web services need to be used if necessary. The interaction is shown on Fig. 4.

Interactive auto rendering mode tries to overcome the drawbacks of pure WebAssembly rendering, but it cannot run on a static server, so it has additional server requirements.

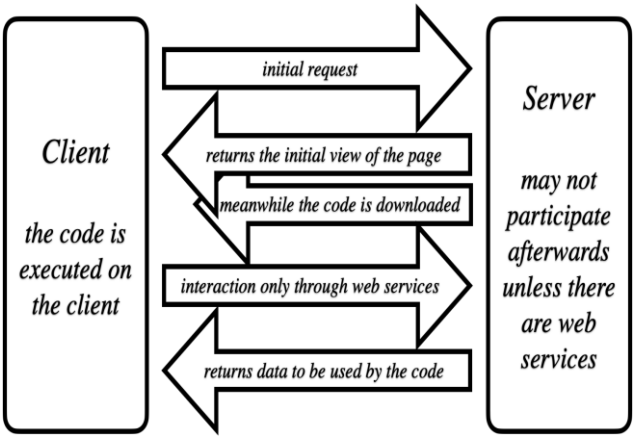

Fig. 4. Interactive auto rendering
Source: Own Elaboration

VI. CONCLUSION

We can summarize the features, potential advantages, and drawbacks of the different Blazor render modes in Table I.

TABLE I. COMPARISON OF BLAZOR'S RENDER MODES

Render Mode	Features			
	Interactivity	***Initial Loading Time***	***Server and Network Load***	***Client Load***
Static	none	fast	low	low
Interactive Server	full	fast	high	low
Interactive Web Assembly	client-only[1]	slow	low	high
Interactive Auto	client-only[1]	fast / but loading continues	low	high

Source: Own elaboration

Furthermore, Microsoft has also introduced streaming rendering. We have not covered it in detail because this is not a separate mode, but it can enhance interactive server-side rendering when loading pages with time-consuming requests [12] so developers should also consider it when interactive SSR is chosen.

[1] Server interaction is possible though web services.

The comparison shows that there is no single "best" mode, each having different advantages and disadvantages.

In conclusion, when starting a new .NET Blazor project, developers are advised to study thoroughly every page/component's necessary functionality and requirements and by using the results from this paper choose for each the most appropriate render mode.

REFERENCES

[1] Statista. Most used web frameworks among developers worldwide, as of 2023. https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/. 25.01.2024.

[2] J. Prosise. Programming Microsoft .NET. Microsoft Press, 2002.

[3] Microsoft. What is .NET? https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet. 25.01.2024.

[4] I. Linnik. A Brief History of .NET Framework. SoftTeco. https://softteco.com/blog/a-brief-history-of-net-framework. 25.01.2024.

[5] D. Esposito, D. Programming ASP.NET 3.5. Microsoft Press, 2008.

[6] D. Garcia, The History of ASP.NET – Part II (Covers ASP.NET MVC). Dot Net Curry. https://www.dotnetcurry.com/aspnet/1493/aspnet-history-part-2-mvc. 01.02.2024.

[7] V. Sulov. "Implementation of Programming Languages on the Platform .NET at Developing Software Applications", Izvestia, Journal of the University of Economics – Varna, Issue 1, 2014, pp. 13-22.

[8] D. Garcia. The History of ASP.NET – Part III (Covers ASP.NET Core). Dot Net Curry. https://www.dotnetcurry.com/aspnet/1494/aspnet-history-part-3-core. 01.02.2024.

[9] D. Roth, J. Fritz, T. Southwick. Blazor for ASP.NET Web Forms Developers, Microsoft Press, 2023.

[10] Microsoft. ASP.NET Core Blazor render modes. https://learn.microsoft.com/en-us/aspnet/core/blazor/components/render-modes?view=aspnetcore-8.0. 02.02.2024.

[11] E. Callahan. WebAssembly Essentials: Make code reusable and deployed for high performance web apps. GitforGits, 2023.

[12] C. Sainty. Blazor in .NET 8: Server-side and Streaming Rendering. https://chrissainty.com/blazor-in-dotnet-8-server-side-and-streaming-rendering/. 02.02.2024.