Work with images in ASP.NET Core Blazor

Article • 02/09/2024

This article describes common scenarios for working with images in Blazor apps.

Throughout this article, the terms **server/server-side** and **client/client-side** are used to distinguish locations where app code executes:

- **Server/server-side**: Interactive server-side rendering (interactive SSR) of a Blazor Web App.
- **Client/client-side**
  - Client-side rendering (CSR) of a Blazor Web App.
  - A Blazor WebAssembly app.

Documentation component examples usually don't configure an interactive render mode with an `@rendermode` directive in the component's definition file (`.razor`):

- In a Blazor Web App, the component must have an interactive render mode applied, either in the component's definition file or inherited from a parent component. For more information, see ASP.NET Core Blazor render modes.

- In a standalone Blazor WebAssembly app, the components function as shown and don't require a render mode because components always run interactively on WebAssembly in a Blazor WebAssembly app.

When using the Interactive WebAssembly or Interactive Auto render modes, component code sent to the client can be decompiled and inspected. Don't place private code, app secrets, or other sensitive information in client-rendered components.

For guidance on the purpose and locations of files and folders, see ASP.NET Core Blazor project structure, which also describes the location of the Blazor start script and the location of <head> and <body> content.

The best way to run the demonstration code is to download the `BlazorSample_{PROJECT TYPE}` sample apps from the Blazor samples GitHub repository that matches the version of .NET that you're targeting. Not all of the documentation examples are currently in the sample apps, but an effort is currently underway to move most of the .NET 8 article examples into the .NET 8 sample apps. This work will be completed in the first quarter of 2024.

Dynamically set an image source

The following example demonstrates how to dynamically set an image's source with a C# field.

For the example in this section:

- Obtain three images from any source or right-click each of the following images to save them locally. Name the images `image1.png`, `image2.png`, and `image3.png`.

- Place the images in a new folder named `images` in the app's web root (`wwwroot`). The use of the `images` folder is only for demonstration purposes. You can organize images in any folder layout that you prefer, including serving the images directly from the `wwwroot` folder.

In the following `ShowImage1` component:

- The image's source (`src`) is dynamically set to the value of `imageSource` in C#.
- The `ShowImage` method updates the `imageSource` field based on an image `id` argument passed to the method.
- Rendered buttons call the `ShowImage` method with an image argument for each of the three available images in the `images` folder. The file name is composed using the argument passed to the method and matches one of the three images in the `images` folder.

`ShowImage1.razor`:

razor

```
@page "/show-image-1"

<PageTitle>Show Image 1</PageTitle>

<h1>Show Image Example 1</h1>

@if (imageSource is not null)
{
    <p>
        <img src="@imageSource" />
```

```
    </p>
}

@for (var i = 1; i <= 3; i++)
{
    var imageId = i;
    <button @onclick="() => ShowImage(imageId)">
        Image @imageId
    </button>
}

@code {
    private string? imageSource;

    private void ShowImage(int id)
    {
        imageSource = $"images/image{id}.png";
    }
}
```

The preceding example uses a C# field to hold the image's source data, but you can also use a C# property to hold the data.

> **Note**
>
> Avoid using a loop variable directly in a lambda expression, such as `i` in the preceding `for` loop example. Otherwise, the same variable is used by all lambda expressions, which results in use of the same value in all lambdas. Capture the variable's value in a local variable. In the preceding example:
>
> - The loop variable `i` is assigned to `imageId`.
> - `imageId` is used in the lambda expression.
>
> Alternatively, use a `foreach` loop with **Enumerable.Range**, which doesn't suffer from the preceding problem:
>
> razor
>
> ```
> @foreach (var imageId in Enumerable.Range(1,3))
> {
>     <button @onclick="() => ShowImage(imageId)">
>         Image @imageId
>     </button>
> }
> ```

For more information, see **ASP.NET Core Blazor event handling**.

Stream image data

An image can be directly sent to the client using Blazor's streaming interop features instead of hosting the image at a public URL.

The example in this section streams image source data using JavaScript (JS) interop. The following `setImage` JS function accepts the `<img>` tag `id` and data stream for the image. The function performs the following steps:

- Reads the provided stream into an ArrayBuffer .
- Creates a Blob to wrap the `ArrayBuffer`.
- Creates an object URL to serve as the address for the image to be shown.
- Updates the `<img>` element with the specified `imageElementId` with the object URL just created.
- To prevent memory leaks, the function calls revokeObjectURL to dispose of the object URL when the component is finished working with an image.

HTML

```
<script>
  window.setImage = async (imageElementId, imageStream) => {
    const arrayBuffer = await imageStream.arrayBuffer();
    const blob = new Blob([arrayBuffer]);
    const url = URL.createObjectURL(blob);
    const image = document.getElementById(imageElementId);
    image.onload = () => {
      URL.revokeObjectURL(url);
    }
    image.src = url;
  }
</script>
```

Note

For general guidance on JS location and our recommendations for production apps, see **ASP.NET Core Blazor JavaScript interoperability (JS interop)**.

The following `ShowImage2` component:

- Injects services for an System.Net.Http.HttpClient and Microsoft.JSInterop.IJSRuntime.
- Includes an `<img>` tag to display an image.
- Has a `GetImageStreamAsync` C# method to retrieve a Stream for an image. A production app may dynamically generate an image based on the specific user or retrieve an image from storage. The following example retrieves the .NET avatar for the `dotnet` GitHub repository.
- Has a `SetImageAsync` method that's triggered on the button's selection by the user. `SetImageAsync` performs the following steps:
  - Retrieves the Stream from `GetImageStreamAsync`.
  - Wraps the Stream in a DotNetStreamReference, which allows streaming the image data to the client.
  - Invokes the `setImage` JavaScript function, which accepts the data on the client.

> **Note**
>
> Server-side apps use a dedicated **HttpClient** service to make requests, so no action is required by the developer of a server-side Blazor app to register an **HttpClient** service. Client-side apps have a default **HttpClient** service registration when the app is created from a Blazor project template. If an **HttpClient** service registration isn't present in the `Program` file of a client-side app, provide one by adding `builder.Services.AddHttpClient();`. For more information, see **Make HTTP requests using IHttpClientFactory in ASP.NET Core**.

`ShowImage2.razor`:

razor

```razor
@page "/show-image-2"
@inject HttpClient Http
@inject IJSRuntime JS

<PageTitle>Show Image 2</PageTitle>

<h1>Show Image Example 2</h1>

<p>
    <img id="image" />
</p>

<button @onclick="SetImageAsync">
    Set Image
```

```
</button>

@code {
    private async Task<Stream> GetImageStreamAsync()
    {
        return await Http.GetStreamAsync(
            "https://avatars.githubusercontent.com/u/9141961");
    }

    private async Task SetImageAsync()
    {
        var imageStream = await GetImageStreamAsync();
        var dotnetImageStream = new DotNetStreamReference(imageStream);
        await JS.InvokeVoidAsync("setImage", "image", dotnetImageStream);
    }
}
```

Additional resources

- ASP.NET Core Blazor file uploads
- File uploads: Upload image preview
- ASP.NET Core Blazor file downloads
- Call .NET methods from JavaScript functions in ASP.NET Core Blazor
- Call JavaScript functions from .NET methods in ASP.NET Core Blazor
- Blazor samples GitHub repository (dotnet/blazor-samples)