Use ASP.NET Core SignalR with Blazor

Article • 09/15/2023

This tutorial provides a basic working experience for building a real-time app using SignalR with Blazor. This article is useful for developers who are already familiar with SignalR and are seeking to understand how to use SignalR in a Blazor app. For detailed guidance on the SignalR and Blazor frameworks, see the following reference documentation sets and the API documentation:

- Overview of ASP.NET Core SignalR
- ASP.NET Core Blazor
- .NET API browser

Learn how to:

- ✓ Create a Blazor app
- ✓ Add the SignalR client library
- ✓ Add a SignalR hub
- ✓ Add SignalR services and an endpoint for the SignalR hub
- ✓ Add a Razor component code for chat

At the end of this tutorial, you'll have a working chat app.

Prerequisites

Visual Studio Code

- Visual Studio Code
- C# for Visual Studio Code (latest version)
- Download and install .NET if it isn't already installed on the system or if the system doesn't have the latest version installed.

The Visual Studio Code instructions use the .NET CLI for ASP.NET Core development functions such as project creation. You can follow these instructions on macOS, Linux, or Windows and with any code editor. Minor changes may be required if you use something other than Visual Studio Code.

Sample app

Downloading the tutorial's sample chat app isn't required for this tutorial. The sample app is the final, working app produced by following the steps of this tutorial.

View or download sample code

Hosted Blazor WebAssembly experience

Create the app

Follow the guidance for your choice of tooling to create a hosted Blazor WebAssembly app:

Visual Studio Code

In a command shell, execute the following command:

.NET CLI

```
dotnet new blazorwasm -ho -o BlazorWebAssemblySignalRApp
```

The `-ho|--hosted` option creates a hosted Blazor WebAssembly solution. For information on configuring VS Code assets in the `.vscode` folder, see the **Linux** operating system guidance in Tooling for ASP.NET Core Blazor.

The `-o|--output` option creates a folder for the solution. If you've created a folder for the solution and the command shell is open in that folder, omit the `-o|--output` option and value to create the solution.

In Visual Studio Code, open the app's project folder.

Confirm that a hosted Blazor WebAssembly app was created: Confirm the presence of a **Client** project and a **Server** project in the app's solution folder. If the two projects aren't present, start over and confirm passing the `-ho` or `--hosted` option to the `dotnet new` command when creating the solution.

To configure Visual Studio Code debugging assets, see:

- Tooling for ASP.NET Core Blazor (use the guidance for the *Linux / macOS* operating system regardless of platform)
- Debug ASP.NET Core Blazor apps

Add the SignalR client library

Visual Studio Code

In the **Integrated Terminal** (**View** > **Terminal** from the toolbar), execute the following command:

.NET CLI

```
dotnet add Client package Microsoft.AspNetCore.SignalR.Client
```

To add an earlier version of the package, supply the `--version {VERSION}` option, where the `{VERSION}` placeholder is the version of the package to add.

Add a SignalR hub

In the `BlazorWebAssemblySignalRApp.Server` project, create a `Hubs` (plural) folder and add the following `ChatHub` class (`Hubs/ChatHub.cs`):

C#

```
using Microsoft.AspNetCore.SignalR;

namespace BlazorWebAssemblySignalRApp.Server.Hubs;

public class ChatHub : Hub
{
    public async Task SendMessage(string user, string message)
    {
        await Clients.All.SendAsync("ReceiveMessage", user, message);
    }
}
```

Add services and an endpoint for the SignalR hub

In the `BlazorWebAssemblySignalRApp.Server` project, open the `Program.cs` file.

Add the namespace for the `ChatHub` class to the top of the file:

C#

```
using BlazorWebAssemblySignalRApp.Server.Hubs;
```

Add SignalR and Response Compression Middleware services:

C#

```
builder.Services.AddSignalR();
builder.Services.AddResponseCompression(opts =>
{
      opts.MimeTypes = ResponseCompressionDefaults.MimeTypes.Concat(
         new[] { "application/octet-stream" });
});
```

Use Response Compression Middleware at the top of the processing pipeline's configuration immediately after the line that builds the app:

C#

```
app.UseResponseCompression();
```

Between the endpoints for controllers and the client-side fallback, add an endpoint for the hub. Immediately after the line `app.MapControllers();`, add the following line:

C#

```
app.MapHub<ChatHub>("/chathub");
```

Add Razor component code for chat

In the `BlazorWebAssemblySignalRApp.Client` project, open the `Pages/Index.razor` file.

Replace the markup with the following code:

razor

```
@page "/"
@using Microsoft.AspNetCore.SignalR.Client
@inject NavigationManager Navigation
@implements IAsyncDisposable

<PageTitle>Index</PageTitle>

<div class="form-group">
    <label>
        User:
        <input @bind="userInput" />
    </label>
</div>
<div class="form-group">
    <label>
        Message:
        <input @bind="messageInput" size="50" />
    </label>
</div>
<button @onclick="Send" disabled="@(!IsConnected)">Send</button>

<hr>

<ul id="messagesList">
    @foreach (var message in messages)
    {
        <li>@message</li>
    }
</ul>

@code {
    private HubConnection? hubConnection;
    private List<string> messages = new List<string>();
    private string? userInput;
    private string? messageInput;

    protected override async Task OnInitializedAsync()
    {
        hubConnection = new HubConnectionBuilder()
            .WithUrl(Navigation.ToAbsoluteUri("/chathub"))
            .Build();

        hubConnection.On<string, string>("ReceiveMessage", (user, message) =>
        {
            var encodedMsg = $"{user}: {message}";
            messages.Add(encodedMsg);
            StateHasChanged();
        });

        await hubConnection.StartAsync();
    }

    private async Task Send()
    {
        if (hubConnection is not null)
            {
                await hubConnection.SendAsync("SendMessage", userInput, mes-
sageInput);
            }
    }

    public bool IsConnected =>
        hubConnection?.State == HubConnectionState.Connected;

    public async ValueTask DisposeAsync()
    {
        if (hubConnection is not null)
        {
            await hubConnection.DisposeAsync();
        }
    }
}
```

ⓘ **Note**

Disable Response Compression Middleware in the `Development` environment when using **Hot Reload**. For more information, see **ASP.NET Core Blazor SignalR guidance**.

Run the app

Follow the guidance for your tooling:

Visual Studio Code

For information on configuring VS Code assets in the `.vscode` folder, see the **Linux** operating system guidance in Tooling for ASP.NET Core Blazor.

Press `F5` to run the app with debugging or `Ctrl`+`F5` (Windows)/`⌘`+`F5` (macOS) to run the app without debugging.

ⓘ **Important**

When executing a hosted Blazor WebAssembly app, run the app from the solution's **Server** project.

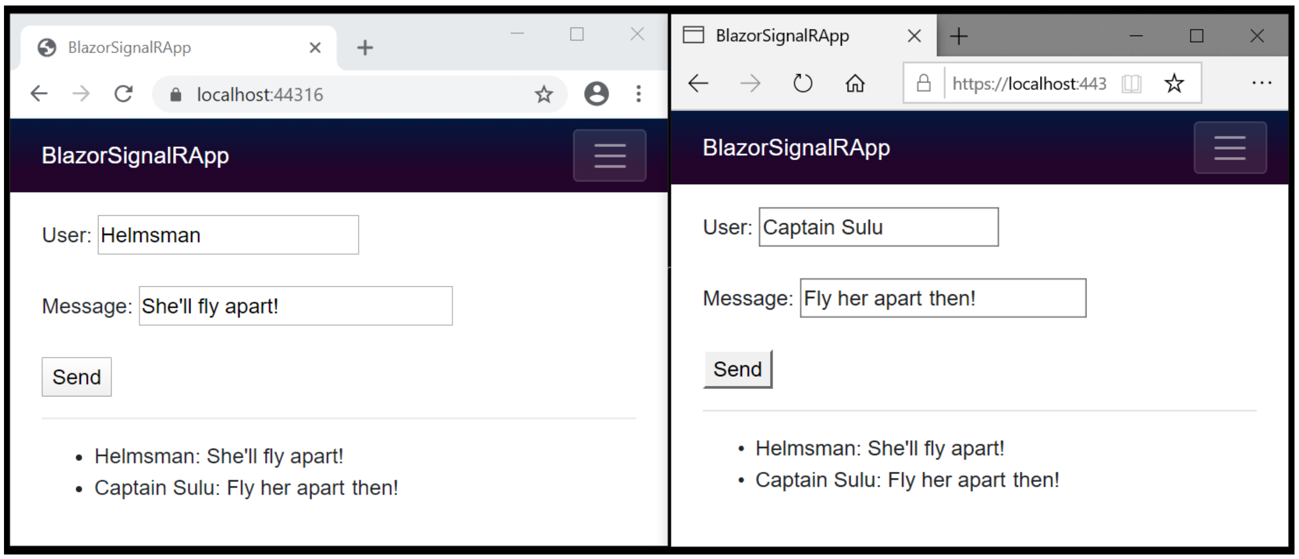
Google Chrome or Microsoft Edge must be the selected browser for a debugging session.

If the app fails to start in the browser:

- In the .NET console, confirm that the solution is running from the "Server" project.
- Refresh the browser using the browser's reload button.

Copy the URL from the address bar, open another browser instance or tab, and paste the URL in the address bar.

Choose either browser, enter a name and message, and select the button to send the message. The name and message are displayed on both pages instantly:



- Helmsman: She'll fly apart!
- Captain Sulu: Fly her apart then!

Quotes: *Star Trek VI: The Undiscovered Country* ©1991 Paramount

Blazor Server experience

Create the app

Follow the guidance for your choice of tooling to create a Blazor Server app:

Visual Studio Code

In a command shell, execute the following command:

.NET CLI

```
dotnet new blazorserver -o BlazorServerSignalRApp
```

The `-o|--output` option creates a folder for the project. If you've created a folder for the project and the command shell is open in that folder, omit the `-o|--output` option and value to create the project.

In Visual Studio Code, open the app's project folder.

When the dialog appears to add assets to build and debug the app, select **Yes**. Visual Studio Code automatically adds the `.vscode` folder with generated `launch.json` and `tasks.json` files. For information on configuring VS Code assets in the `.vscode` folder, including how to manually add the files to the solution, see the **Linux** operating system guidance in Tooling for ASP.NET Core Blazor.

Add the SignalR client library

Visual Studio Code

In the **Integrated Terminal** (**View** > **Terminal** from the toolbar), execute the following command:

.NET CLI

```
dotnet add package Microsoft.AspNetCore.SignalR.Client
```

To add an earlier version of the package, supply the `--version {VERSION}` option, where the `{VERSION}` placeholder is the version of the package to add.

Add a SignalR hub

Create a `Hubs` (plural) folder and add the following `ChatHub` class (`Hubs/ChatHub.cs`):

C#

```
using Microsoft.AspNetCore.SignalR;

namespace BlazorServerSignalRApp.Server.Hubs;

public class ChatHub : Hub
{
    public async Task SendMessage(string user, string message)
    {
        await Clients.All.SendAsync("ReceiveMessage", user, message);
    }
}
```

Add services and an endpoint for the SignalR hub

Open the `Program.cs` file.

Add the namespaces for Microsoft.AspNetCore.ResponseCompression and the `ChatHub` class to the top of the file:

C#

```
using Microsoft.AspNetCore.ResponseCompression;
using BlazorServerSignalRApp.Server.Hubs;
```

Add Response Compression Middleware services:

C#

```
builder.Services.AddResponseCompression(opts =>
{
   opts.MimeTypes = ResponseCompressionDefaults.MimeTypes.Concat(
         new[] { "application/octet-stream" });
});
```

Use Response Compression Middleware at the top of the processing pipeline's configuration:

C#

```
app.UseResponseCompression();
```

Between the endpoints for mapping the Blazor hub and the client-side fallback, add an endpoint for the hub immediately after the line `app.MapBlazorHub();`:

C#

```csharp
app.MapHub<ChatHub>("/chathub");
```

Add Razor component code for chat

Open the `Pages/Index.razor` file.

Replace the markup with the following code:

razor

```razor
@page "/"
@using Microsoft.AspNetCore.SignalR.Client
@inject NavigationManager Navigation
@implements IAsyncDisposable

<PageTitle>Index</PageTitle>

<div class="form-group">
    <label>
        User:
        <input @bind="userInput" />
    </label>
</div>
<div class="form-group">
    <label>
        Message:
        <input @bind="messageInput" size="50" />
    </label>
</div>
<button @onclick="Send" disabled="@(!IsConnected)">Send</button>

<hr>

<ul id="messagesList">
    @foreach (var message in messages)
    {
        <li>@message</li>
    }
</ul>

@code {
    private HubConnection? hubConnection;
    private List<string> messages = new List<string>();
    private string? userInput;
    private string? messageInput;

    protected override async Task OnInitializedAsync()
    {
        hubConnection = new HubConnectionBuilder()
            .WithUrl(Navigation.ToAbsoluteUri("/chathub"))
            .Build();

        hubConnection.On<string, string>("ReceiveMessage", (user, message) =>
        {
            var encodedMsg = $"{user}: {message}";
            messages.Add(encodedMsg);
            InvokeAsync(StateHasChanged);
        });

        await hubConnection.StartAsync();
    }

    private async Task Send()
    {
        if (hubConnection is not null)
            {
                await hubConnection.SendAsync("SendMessage", userInput, mes-
sageInput);
            }
    }

    public bool IsConnected =>
        hubConnection?.State == HubConnectionState.Connected;

    public async ValueTask DisposeAsync()
    {
        if (hubConnection is not null)
        {
            await hubConnection.DisposeAsync();
        }
    }
}
```

ⓘ Note

Disable Response Compression Middleware in the `Development` environment when using **Hot Reload**. For more information, see **ASP.NET Core Blazor SignalR guidance**.

Run the app

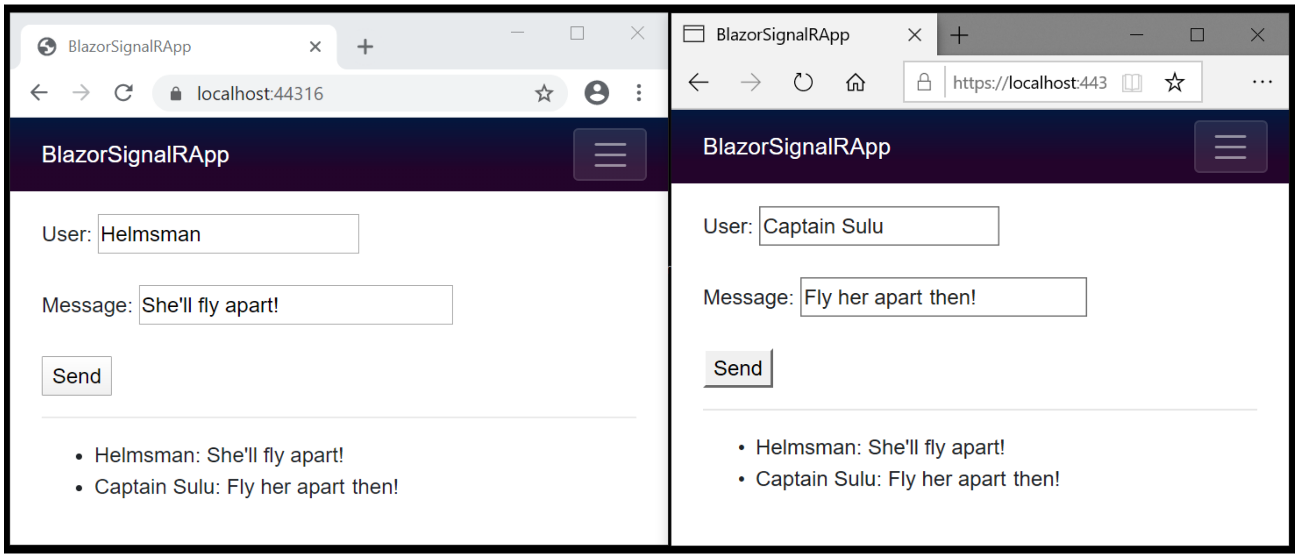
Follow the guidance for your tooling:

Visual Studio Code

Press `F5` to run the app with debugging or `Ctrl`+`F5` (Windows)/`⌘`+`F5` (macOS) to run the app without debugging.

Copy the URL from the address bar, open another browser instance or tab, and paste the URL in the address bar.

Choose either browser, enter a name and message, and select the button to send the message. The name and message are displayed on both pages instantly:



- Helmsman: She'll fly apart!
- Captain Sulu: Fly her apart then!

Quotes: *Star Trek VI: The Undiscovered Country* ©1991 Paramount

Next steps

In this tutorial, you learned how to:

- ✓ Create a Blazor app
- ✓ Add the SignalR client library
- ✓ Add a SignalR hub
- ✓ Add SignalR services and an endpoint for the SignalR hub
- ✓ Add a Razor component code for chat

For detailed guidance on the SignalR and Blazor frameworks, see the following reference documentation sets:

Overview of ASP.NET Core SignalR ASP.NET Core Blazor

Additional resources

- Bearer token authentication with Identity Server, WebSockets, and Server-Sent Events
- Secure a SignalR hub in hosted Blazor WebAssembly apps
- SignalR cross-origin negotiation for authentication
- SignalR configuration
- Debug ASP.NET Core Blazor apps
- Threat mitigation guidance for server-side ASP.NET Core Blazor
- Blazor samples GitHub repository (dotnet/blazor-samples)